CS541 Cheat Sheet

Probability

Markov's Inequality

$P(X \geq t) \leq \frac{E(X)}{t}$

Definition: The probability that X is greater than some nonnegative value t is less than or equal to the expected value (average) of X over t.

Example for usage:

- Students on a campus carry an average of $20 in cash. If you meet a student at random, estimate the probability that they are carrying more than $100. Also estimate the probability they are carrying less than $0.
  - $P(X \geq 100) \leq \frac{20}{100} = \frac{1}{5}$
  - $P(X < t) \geq 1 - \frac{E(X)}{t}$
    $P(X \geq 80) \leq \frac{20}{80} = \frac{1}{4} \rightarrow P(X \leq 80) \geq 3/4$

Chebyshev's Inequality

$P(|X - E(X)| > t) \leq \frac{Var(X)}{t^2}$

Definition: The probability that X will differ from its expected value $E(X)$ by a magnitude greater than t is less than or equal to its variance $Var(X)$ over t^2.

Example for usage:

- Suppose a fair coin $P(X = H) = 0.5$ is tossed 50 times. What is the probability that the number of heads will be outside [15,35], or $P(|X - 25| > 10)$?
  - $E(X) = np = 50(0.5) = 25; Var(X) = np(1-p) = 50(0.5)(0.5) = 12.5$
    $P(|X - 25| > 10) \leq \frac{12.5}{10^2} = 0.125$

Hoeffding's Inequality

Let $a_i \leq x_i \leq b_i,$ and $S = \sum_{i=1}^{n} x_i$. Then $\forall t > 0$,

$P(S - E(S) \geq t) \leq e^{\frac{-2t^2}{\sum_{i=1}^{n}(b_i - a_i)^2}}$

$P(|S - E(S)| \geq t) \leq 2e^{\frac{-2t^2}{\sum_{i=1}^{n}(b_i - a_i)^2}}$

Definition: The probability that a sum of bounded random variables S deviates from its expected value $E(S)$ by more than t is less than or equal to a certain amount, called the *Failure Probability*.

Example of usage:

- Suppose we flip a weighted coin $P(x_i = 1) = 0.6$, where $H = 1, T = -1$. How many times would we need to flip the coin to get one hundred more Heads than Tails $S = 100$ with confidence of at least 99% (*Failure Probability* ≤ 0.01)?

- $a = -1, b = 1; E(S) = np = 0.2n$
- $P(|S - E(S)| \geq t) \leq 2e^{\frac{-2t^2}{\sum_{i=1}^{n}(b_i - a_i)^2}}$

  $P(|S - 0.2n| \geq t) \leq 2e^{\frac{-2t^2}{\sum_{i=1}^{n}(1-(-1))^2}} = 2e^{\frac{-2t^2}{4n}} = 2e^{\frac{-t^2}{2n}} \rightarrow P(|S - 0.2n| \geq t) \leq 2e^{\frac{-t^2}{2n}}$
  $P(|S - 0.2n| \leq t) = 1 - P(|S - 0.2n| \geq t)$
- RHS = *Failure Probability* $= 2e^{\frac{-t^2}{2n}} \leq 0.01; 100 - 0.2n \leq t$

  Solve for $n \rightarrow 2e^{\frac{-(100-0.2n)^2}{2n}} \rightarrow n \approx 1019.$
  We need to roll at least 1019 times to get one hundred more Heads than Tails with a probability of success of at least 0.99.

Moments

Definition: Quantitative measures related to the shape of the function's graph.

Most Notable Moments:

- Mean: $\mu \equiv E[X] = \sum_{i=1}^{n} x_i p_{x_i}$
- Variance: $\sigma^2 \equiv Var(X) = E[(X - E[X])^2]$
- Skewness: $\mu_3 = E[(\frac{X - \mu}{\sigma})^3]$
- Kurtosis

Online Learning (Hedge)

Concept: Method of machine learning in which data becomes available in a sequential order and is used to update the best predictor for future data at each step.

Structure: Have n experts, each with a different (unknown) probability of successfully identifying an input.

Goal: Pick a strategy to find the best player

Strategy:

- Let the weight of each expert's prediction start at $\frac{1}{N}$ where N is the number of experts.
- Let L_i^t denote the i^{th} player on their t^{th} iteration of the game, having a result $L \in \{0, 1\}$ corresponding to {success, failure}.
- Each expert's weight will be updated as $w_i^{t+1} = w_i^t * \beta^{L_i^t}$ where $\beta \in (0, 1)$.
  - If the expert succeeds, their weight does not change. If they fail, their weight diminishes.
  - An expert's loss:
    - $L_i = \sum_{t=1}^{T} L_i^t$
  - All experts' combined loss at t^{th} iteration:
    - $E_{i \sim w^t}[L_i] = \sum_{i=1}^{N} w_i L_i.$
  - Loss of Algorithm (all experts across all iterations), normalized with $\frac{1}{sum(w_i)}$:
    - $L_A = \frac{1}{\sum_{i=1}^{N} w_i^t} \sum_{t=1}^{T} (\sum_{i=1}^{N} w_i^t L_i^t)$

        - Loss of Algorithm is bounded:
          $L_A \leq min_{1 \leq i \leq N} L_i + log(N)$
    - The sum of the expert's weights at $t+1$ will be bounded by L_i and L_A:
        - $L_i \leq \sum_{i=1}^{N} w_i^{t+1} \leq L_A$
- Initially, there will be a gap between overall prediction and best expert's prediction. After many iterations, the overall prediction will agree with the best expert, which ensures the average loss trends toward 0.
    - $\frac{L_A}{T} \leq min(\frac{L_i}{T}) + \frac{log(N)}{T} \rightarrow 0$ as $T \rightarrow \infty$

Probably Approximately Correct (PAC) Model

Concept: The learner receives samples and must select a generalization function from a certain class of possible functions. The goal is that, with high probability, the selected function will have low generalization error.

Structure: Draw samples to get upper and lower bounds on the true boundary.

Goal: Approximate the true model $Y = \begin{cases} +1 : x \geq \alpha \\ -1 : x < \alpha \end{cases}$ using these bounds to find α_{pred} such that $|\alpha - \alpha_{pred}| \leq \epsilon$.

- Find a data sample that falls within $[a_-, a_+]$ where $\alpha_{+,-} = \alpha \pm \epsilon$.
- Goal Event 1: Data falls within $[a_-, a]$.
- Goal Event 2: Data falls within $[a, a_+]$.

Strategy: Draw data samples with an arbitrary starting α_{pred} to get an upper and lower bound.

- Draw a sample $(\alpha_{low}, -1)$ to get a lower bound.
- Draw a sample $(\alpha_{high}, +1)$ to get an upper bound.
- We can now update $\alpha_{pred} = \frac{\alpha_{low} + \alpha_{high}}{2}$ to refine these boundaries.

How many times do we need to draw to get a sample to fall within $[\alpha_-, \alpha]$ given the size of the interval is $\frac{\epsilon}{2}$?

- Let $z_i = \begin{cases} 1 : [\alpha_-, \alpha] \\ 0 : otherwise \end{cases} \rightarrow P(z_i = 1) = \frac{\epsilon}{2} \rightarrow E[z_i] = \frac{2}{\epsilon}$.
    - You'd expect to draw $\frac{2}{\epsilon}$ times to get one sample that falls within $[\alpha_-, \alpha]$.

How can we find the probability that one goal event will occur with probability $1 - \frac{\delta}{2}$?

- Use Hoeffding's: $P(|S - E(S)| \geq t) \leq 2e^{\frac{-2t^2}{\sum_{i=1}^{n}(b_i - a_i)^2}}$
    - RHS: $2e^{\frac{-2t^2}{n(b-a)^2}} \leq 1 - \frac{\delta}{2}$
    - $n \geq \frac{2}{\epsilon}$ from above
    - Solve for n to get $n = \frac{16}{e^2} log(\frac{2}{\delta})$ drawings to ensure that we'll get one point that satisfies a goal event with probability $1 - \frac{\delta}{2}$.
- Since both events are symmetric, we'll need $n = \frac{16}{e^2} log(\frac{2}{\delta})$ drawings to get both events with a success probability $1 - \delta$ (union bound).

- Union Bound: If $P(E_i = 1) = 1 - \delta_i$, then $P(\cap_{i=1}^{k} E_i) = 1 - \sum_{i=1}^{k} \delta_i$.

Label Efficiency

Concept: If the label in data $x \in [a, b]$ is based on a partition at $a \leq \alpha \leq b$, we do not need to label all of the data.

Goal: Find the partition α such that all entries to the left are labeled "-" and all entries to the right are labeled "+".

Strategy: Implement binary search. Query in the middle and label it. Based on its result, reduce our search to the left or right half of the data. Repeat until we find the partition and accomplished our goal.

- This reduces a $\Theta(n)$ label complexity down to a $\theta(log(n))$ label complexity.

Adaboost

Concept: The output of other 'weak' learning algorithms is combined into a weighted sum that represents the final output of the boosted classifier.

Goal: Combine weak learners to make a better model.

Strategy:

- AdaBoost is adaptive in the sense that subsequent weak learners $h_1, \ldots, h_k$ are tweaked in favor of those instances misclassified by previous classifiers. In some problems it can be less susceptible to the overfitting problem than other learning algorithms. The individual learners can be weak, but as long as the performance of each one is slightly better than random guessing, the final model can be proven to converge to a strong learner.
  - If many of the weak learners are correctly labeling an example, de-emphasize that example in the next iteration. Conversely, if too few weak learners are correctly labeling an image, emphasize it in the next iteration to ensure the model improves.

Structure:

- $(x_1, y_1), \ldots, (x_n, y_n)$ where all weights start at $\frac{1}{n}$
  - If w_i is large, hard example for $h_1, \ldots, h_k$.
  - If w_i is tiny, easy example for $h_1, \ldots, h_k$.
  - Loss in boosting is weighted
    - If loss approaches 0, we are done with that example. If loss is large, we need to keep training on it.
- Algorithm:
  - $t = 1, \ldots, T$
  - $L_i^t = |h_t(x_i) - y_i|$
    - ith sample (xi,yi)
  - $w_i^{t+1} = w_i^t \beta^{L_i^t}$
  - $h_{t+1} = min \sum_{i=1}^{n} w_i^{t+1} loss(x_i, y_i)$

- $T : h_1 \dots h_T$
- Given x, look at predictions $h_1(x), \dots, h_T(x)$, and take the majority vote.

Gradient Descent

Concept: First-order iterative algorithm to find a local minimum of a differentiable function. Take repeated steps in the opposite direction of the gradient because this is the direction of steepest descent.

Goal: Find the local minimum of a differentiable function.

- Gradient = 0 and moving in any direction would be moving upwards.

Strategy: Update your prediction w as such: $w_{t+1} = w_t - \eta \nabla F(w_t)$.

When does gradient descent fail?

- When it stops at a local minimum, but not the global optimum.
- If the function is concave

When does it succeed?

- GD is good for convex functions!
- Speed depends on learning rate: too big yields overstepping; too little takes too long.

Hessian Matrix

Definition: Hessian matrix is a square matrix of second-order partial derivatives of a scalar-valued function.

Idea: Hessian matrix is used to check if a function is convex, typically for gradient descent.

Examples of Usage:

- $min_{\mathbf{w} \in \mathbb{R}^d} F(\mathbf{w}) := \frac{1}{2} ||\mathbf{y} - \mathbf{X}\mathbf{w}||^2$
- $\frac{\partial F}{\partial \mathbf{w}} = -\mathbf{X}^\top ||\mathbf{y} - \mathbf{X}\mathbf{w}||$, or
  $- \begin{bmatrix} \sum_{i=1}^n x_{i1}(y_i - \sum_{j=1}^d w_j x_{ij}) \\ \vdots \\ \sum_{i=1}^n x_{id}(y_i - \sum_{j=1}^d w_j x_{ij}) \end{bmatrix}$
- $\nabla^2 F = \frac{\partial^2 F}{\partial \mathbf{w}^2} = \mathbf{X}^\top \mathbf{X}$, or $d \times n * n \times d$
  $\begin{bmatrix} \sum_{i=1}^n x_{i1}^2 & \dots & \sum_{i=1}^n x_{i1} x_{id} \\ \vdots & \ddots & \vdots \\ \sum_{i=1}^n x_{id} x_{i1} & \dots & \sum_{i=1}^n x_{id}^2 \end{bmatrix}$
- $F(\mathbf{w})$ *is convex* $\iff \nabla^2 F(x) \geq 0, \forall x \in D$
  - Represent a (assumedly linearly independent) $\mathbf{X}$ as $\mathbf{X} = [\mathbf{v_1} \quad \mathbf{v_2} \quad \dots \quad \mathbf{v_d}]$
    $k_1 \mathbf{v_1} + \dots + k_d \mathbf{v_d} = \mathbf{0} \iff k_1 = \dots = k_d = 0$
    Now let $\mathbf{k}$ be an eigenvector of $\nabla^2 F$,
    $\mathbf{k} \neq \mathbf{0} \implies (k_1 \mathbf{v_1} + \dots + k_d \mathbf{v_d})^2 > 0$

- $= [k_1 \dots k_d] \begin{bmatrix} \mathbf{v_1} \\ \vdots \\ \mathbf{v_d} \end{bmatrix} [\mathbf{v_1} \dots \mathbf{v_d}] \begin{bmatrix} k_1 \\ \vdots \\ k_d \end{bmatrix} = \mathbf{k}^\top \nabla^2 F \mathbf{k} = \lambda \mathbf{k}^\top \mathbf{k} > 0$
- $\lambda \mathbf{k}^\top \mathbf{k} > 0 \rightarrow \mathbf{k}^\top \mathbf{k} = \sum_{i=1}^{d} k_i^2 > 0 \implies \lambda > 0$
  Since k is arbitrary, all eigenvalues must be positive, and thus $\nabla^2 F$ is positive definite, which means it is also positive semi-definite.
- Therefore, $F(\mathbf{w})$ is convex.

L-Smoothness

Definition: $F(w)$ is smooth if for any $w_1, w_2 \in \mathbb{R}^d$,
$||\nabla F(w_2) - \nabla F(w_1)||_2 \leq L||w_2 - w_1||_2$.

Idea: In other words, the difference of gradients over the difference of weights must be bounded by a positive L.
$\frac{||\nabla F(w_2) - \nabla F(w_1)||_2}{||w_2 - w_1||_2} \leq L$.

- The function must continually increase $\in L * O(n^2)$. L-Smooth depends on the tightest L that fits.

Example of Usage:

- $F(w) = w^2, \nabla F = 2w$,
  $\frac{||2*(w1-w2)||}{||w1-w2||} \rightarrow$ ratio=2, L=2
  Constant L -> Smooth! 2-Smooth
- $F(w) = |w|, w_1 = -\epsilon, w_2 = +\epsilon, (\epsilon > 0)$
  $\nabla F(w_1) = -1, \nabla F(w_2) = +1$
  $\frac{-1-1}{2\epsilon} \leq L$
  ratio $= \frac{1}{\epsilon} \leq L, \epsilon \rightarrow 0$
  Since we cannot find a constant L, we know this function is not smooth around the origin. (w1,w2 cannot be bounded by a constant L)
- $F(w) = w^4, w_1, w_2$
  $\frac{4||w_2^3 - w_1^3||}{||w_2 - w_1||} \leq L$
  ratio $= 4(w_1^2 + w_1 w_2 + w_2^2) \leq L$
  Non-constant L -> non-smooth

Strong Convexity

Definition: $\forall w1, w2 \in \mathbb{R}^d$, $\frac{||\nabla F(w_2) - \nabla F(w_1)||_2}{||w_2 - w_1||_2} \geq \alpha$ where α is the min eigenvalue of $\nabla^2 F(w)$.

Idea: There exists a quadratic lower bound on the growth of the function. Implies strict and normal convexity. Basically, function's growth $\in \alpha * \Omega(n^2)$.

Example of Usage:

- If you add $F(w) = w^2$ to any function, it will be Strongly Convex
- $12w^2$ is considered 12-strongly convex.

Computational Complexity

Conditions	Guarantee	# of Iterations
α-SC, L-smooth	$\|\|w^t - w^*\|\|_2 \leq \epsilon$	$c * log(\frac{1}{\epsilon})$
L-smooth	$F(w^t) - F(w^*) \leq \epsilon$	$\frac{L}{\epsilon}$

Gradient Descent solves linear regression efficiently: d2(n+d) v.s. ndc*log(1 where d is the number of dimensions.

How can we improve GD w.r.t. computational cost?

- Stochastic GD
  - Initialize w0, say w0=0For t=1,2, ...
    Randomly draw i_t from $\{1, 2, \ldots, n\}$ and update
    $w_t = w_{t-1} - \eta_t \nabla f_{i_t}(w^{t-1})$
    $E[w^t] = E[w^{t-1}]\eta_t E[\nabla f_{i_t}(w^{t-1})]$
  - $SGD \in O(d), GD \in O(nd)$
  - total time = $\frac{cost}{iteration}$* # of iterations