CS 260 9/23/24

- **Recap SGD** (Stochastic Gradient Descent)
  - key idea: take the derivative of 1 datapoint at a time and use that to update w
    - Moving in the direction to minimize cost function - against the derivative, by the α step size
    - Useful when you can't find an analytic solution

for $i = 1, 2, \ldots n$: (shuffle)

$$\vec{w} \leftarrow \vec{w} - \alpha \underbrace{(\vec{w} \cdot \vec{x}_i - y_i)\vec{x}_i}_{\text{derivative}}$$

(α: step size)

if $|J(\vec{w}^t) - J(\vec{w}^{t+1})| < \varepsilon$ ← check for convergence, ε is very small

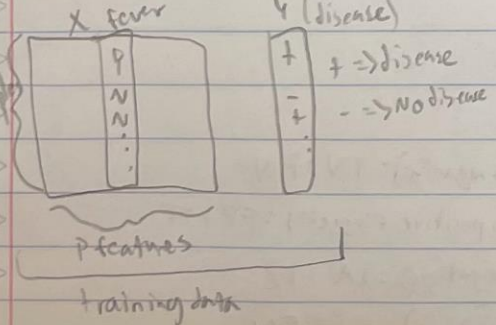
Stop!

- Choosing step size alpha - if it's too small, convergence is too slow - if it's too large, we can overshoot the minimum and may fail to converge (or even diverge)

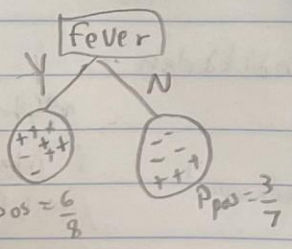
Pros and Cons

Gradient Descent	Normal Equations
• requires multiple iterations	• Non-iterative (1 computation - sometimes large)
• need to choose α	• No need for α
• works well when p is large	• slow if p is large
• can support online learning - just add new data to the dataset w/out changing model	- matrix inversion is $O(p^3)$
• We have more control, but then have to make decisions	

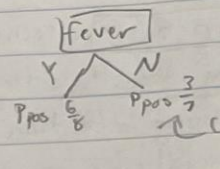
- **Introduction to Classification**
- **Binary Classification** - 2 classes
  - ex: transactions that indicate credit card fraud, finding genes under natural selection ← in these, trying to find unusual items - we call these positives
- discrete classes data can be put into



decision tree - w/ a single feature is called a "stump"



- Use probabilities to classify test examples



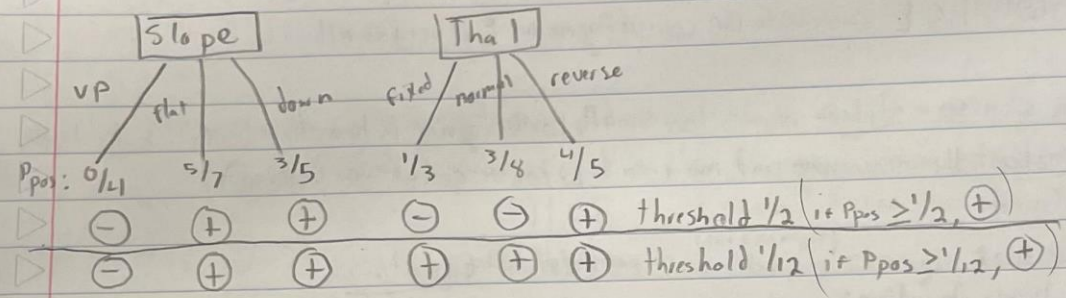
threshold ex 0.5 $\Rightarrow \hat{y}_{test} = \ominus$ (no disease)

0.25 $\Rightarrow \hat{y}_{test} = \oplus$ (disease)

if $P_{pos} \geq$ threshold, then classify example as positive

↑ used to decide how to classify data points

Handout 7



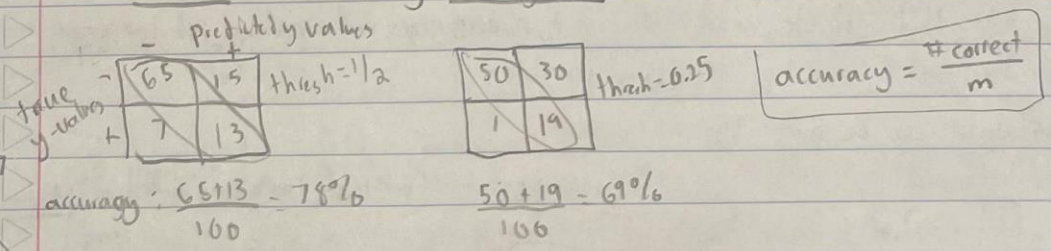
- Evaluation Metrics

- Goals of Evaluation:

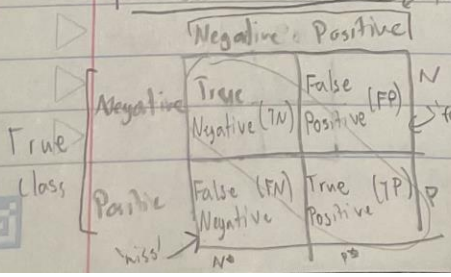
- think about what metrics are important for problem at hand
- Compare diff methods or models on same problem (ex comparing cost using cost function)
- Common set of tools that other users can understand

- Training and Testing - separate data into 'train' and 'test' (n = num training ex, m = num testing ex)

- fit (create) model using training data
- evaluate model using testing data



- Confusion Matrices



N* (what we said was negative) = TN + FN

P* (what we said was positive "flagged") = FP + TP

N (total num true negatives) = TN + FP

P (total num true positives) = TP + FN

Error: $\frac{(FN+FP)}{(TN+FP+FN+TP)} = \frac{FN+FP}{N+P}$

False Positive rate: $\frac{FP}{(TN+FP)} = \frac{FP}{N}$

Accuracy: 1 - Error

Precision: $\frac{TP}{(FP+FP)} = \frac{TP}{P^*}$

Recall: $\frac{TP}{(FN+TP)} = \frac{TP}{P}$ (True positive rate)

Precision and Recall - Evaluation methods

- of all flagged exs, which ones are actually relevant? (i.e. positive) - (purity)
- recall - of all relevant results, which ones did I actually return? (completeness)
- to increase recall - lower threshold to increase results classified as positive, which means higher chance we catch all TPs and reduce FN (but also increases FP which hurts precision)
- to increase precision - more selective in what we flag (this then affects recall as FN ↑)
- generally models cannot have both high precision (less FP) and high recall