Slide for logistics and news

- Torrent of AI news!
  - DeepSeek, Paris Summit, more.
- logistics
  - Assignment 2 posted

Agenda for this deck

- Discuss training data influence methods from our reading
- Goal: highlight key concepts to understand and think critically about

Methods we’ll discuss

- Leave one out influence (LOO)
- Downsampling
- Shapley values
- Influence functions

Leave-one-out influence

Our first retraining-based method

Notation reminder

- T: iteration count (how many times we did gradient descent)
- n: number of data points
- p: number of parameters

LOO: basic idea

- remove an instance and see the change in risk
- that’s pretty much all there is to it!
- LOO is cool, because we can explain this to basically everyone.
- "So you imagine what would happen if Google deleted all the data about, and then retrained all their models – that would tell you your LOO influence value"

LOO complexity

- time complexity $O(T)$ for one retrain
- so $O(nT)$ for n data points
- space complexity $O(n+p)$ (hold n values, each retrain has to hold p params). (exact value depends on concurrency)
- trade-off in terms of discarding models

Note on "incremental complexity"

- incremental complexity here refers to the time it takes to calculate influence for additional *test* instances

LOO pros and cons

- simple, easy to understand
- used for fairness, see BF21 in H+L
- but: extremely expensive!
- need to retrain even more if you want to account for variance in model training

Additional notes on LOO

- can efficiently calculate for KNN, see Jia+21a in H+L
- $O(nlog(n))$ to calculate very close to exact value (vs. $O(nT)$, where T is probably pretty big)
- can also efficiently calculate for decision tree, see Sha+18b
- can efficiently estimate for linear regression
- group influence: we can easily extend to "leave m out", $\binom{n}{m}$

Can we really get LOO influence values?

- So, while it’s very expensive to get LOO values, there’s a bunch of scenarios in which we can get actually get them
- But perhaps we want to try out some of these other methods...

Downsampling

Our second retraining-based approach!

Downsampling: basic idea

- Train an ensemble of submodels, i.e. sample dataset a bunch of times and retrain + evaluate for each sample
- calculate change in average risk when z_i is not used
- consistent estimator of LOO (in the sense of "statistically consistent")

Downsampling compexity

- instead of depending on size n, our complexity depends on our selected number of submodels K. More submodels costs more but improves quality of the estimator
- it’s otherwise very similar complexity analysis to LOO
- cheaper than LOO, and we can vary number of submodels as needed (i.e. if we’re running out of time and need to give some influence estimates to our boss!)
- still expensive!

Downsampling: params we control

- K: the number of submodels
- m: how big is each submodel training data D^k. In the work that introduced downsampling, authors recommend 0.7 * n, i.e. 70% of the full data
- Each instance z_i appears in K_i submodels

Downsampling influence definition

- Calculate average loss on one z_{te} across all submodels without z_i
- Calculate average loss on one z_{te} across all with z_i
- Subtract!

Downsampling influence intuition

- If the loss without z_i is big (i.e., bad) and the loss with z_i is small (i.e. good), example has positive influence
- recall positive influence = improve some quality measure
- If the loss without z_i is small (i.e. good) and the loss with z_i is big (i.e. bad), example has negative influence

Downsampling and training variance

- LOO had some issues with randomness in training / the idea that we’ll get variance in our model performance if we just retrain with the same architecture and data over and over
- Downsampling is likely better in this regard

Downsampling: bounds and using it for groups

- "In practice, K, n, and m are finite. Nonetheless, Feldman and Zhang FZ20, Lemma 2.1 prove that, with high probability, Downsampling's LOO influence estimation error is bounded given K and $\frac{m}{n}$." - H&L pg 13
- "Downsampling trivially extends to estimate the expected group influence of multiple training instances. Observe however that the expected fraction of u.a.r. training subsets that either contains all instances in a group or none of a group decays geometrically with $\frac{m}{n}$ and $(1 - \frac{m}{n})$, respectively"

Downsampling: bounds and using it for groups, a few comments

- as K goes up and m gets bigger (closer to n, $\frac{m}{n}$ gets closer to 1), we get closer to just calculating LOO
- if m is really small, we get something very different from LOO (but perhaps still interesting: what do you think?)

- we can do downsampling for groups: compare all the cases in which the whole group is missing!
- for groups: if m/n gets too small (e.g. imagine we’re only grabbing 2 instances at a time - what would happen), probably not so useful... but interesting?

Shapley values

Our third retraining-based approach!

Shapley value: the Idea

- from cooperative game theory
- idea: give people an idea about their "value added" in a game where they might choose from a variety of teams
- if we treat each instance as a "player" and imagine all the possible "teams" that might exist, we start to get a Shapley value

Shapley value: The Game

We’re imagining a game in which each data point is one "player" and the players want to get the best possible performance.

We want to give each player a score, but that score should account for all the possible "teams" they might join.

Why Shapley?

- basic idea is to devise a mathematically principled way to distribute rewards to people who contribute to some cooperative endeavor
- ex: carbon emissions
- ex: distribute bonus to employees
- ex: marketing attribution (how did IG ads and YouTube ads "work together" to get you to buy something)

Shapley value: The Teams

Imagine 4 players: Alice, Bob, Chen, and Di

Calculate $L(D_{ABCD})$, then calculate $L(D_{ABC})$ (what if it's just Alice, Bob, and Chen), then calculate $L(D_{ABD})$ (what if it's just Alice, Bob, and Di), and so on.

Including $L(D_A)$ (what if it's literally just Alice by herself), etc.

All groups of size 4, all groups of size 3, all groups of size 2, all groups of size 1.

Shapley value: How many teams?

The Power set - all subsets of all sizes.

How many subsets in the power set?

Well, each time we construct a set, each item can either be in the set or not.

$$2^n$$

See Wikipedia for much more detailed explanation:

https://en.wikipedia.org/wiki/Power_set

Shapley value: The Equations

(open paper)

- For all the "coalitions" S without item i, compare the "score" with i and without i.
- Each time, divide by the binomial coefficient (n-1, cardinality of S), i.e. n-1 choose size of S.
- This tells us how many other ways there are to create a team of the same size, and we want to weight things relative how many other combinations exist

Example

We have 10 data points. We omit point 1 (i=1). S has size 9, so the LOO value gets divided by "9 choose 9" = 1.

Next we examine all datasets of size 8 (missing 1+2, missing 1+3...). |S| = 8, so 9 choose 8 gives us 9. Each of these 9 coalitions gets divided by 9.

Next we examine all datasets of size 7 (missing 1+2+3, missing 1+2+4...). |S| = 8, so 9 choose 7 gives us 36. Each of these 36 coalitions gets its influence divided by 36.

- the goal is to weight all coalitions *sizes* equally

- denominator essentially weights each marginal contribution v(S ∪ \{i\}) - v(S) by the probability of that particular sequence occurring if we were to take all players, randomly shuffle them into a line to build teams
- hence, Shapley values represent an "average" or "expected" marginal contribution - they literally average over all possible sequences in which coalitions could form, with each sequence equally likely.

Shapley values

- Gives us the weighted impact on risk when z_i is added to a random training subset of any size
- We add weights so that e.g. for n=10 and i=1, all 9 teams of size 2 have the same weight as all 36 teams of size 7
- Looking forward: we don’t have to weight all sizes equally, though
- can think of it as a LOO influence that accounts for all possible subsets of D

Why account for all coalitions?

- What if you know a super secret recipe
- Your part of a AI chef training set
- But one other person knows your recipe too
- In LOO, you look like you have very little value
- But if you were added to a random subset of 10 other people, you look like you have amazing value!

Why weight all coalition sizes equally

We don’t have to!

The "Beta Shapley" is all about this

https://arxiv.org/abs/2110.14049

Idea: weight "low cardinality" examples more heavily (e.g. the "what if I was added to group of 10 other people instead of 1000 other people, will I look more impressive?") and get good data value estimates

Shapley and feature explanations

- you may have also seen Shapley values mentioned in explainable AI and/or fair AI materials
- also used for feature explanations (though some issues, see https://proceedings.mlr.press/v119/kumar20e.html)

Shapley properties

- researchers like the nice theoretical properties
- Null player: add no marginal value to any coalition → receive zero payoff
- Symmetrical - "equal treatment of equals" (from wp)
- Linear - Shapley value of a sum of games equals the sum of their individual Shapley values
- Pareto Optimality
- Accounts for multiple training set sizes

Shapley: empirical pros

- Better at detecting data poisoning?
- maybe better for compensating people (open research question!)

Shapley caveats

- super expensive. Worst case exponential time (pending P=NP...)
- 2^n models, not n models...
- can be estimated

Shapley estimation

- We’re just going to the discuss the Monte Carlo approach
- Basic idea: shuffle your training data
- Running example: 1,2,3,4 - > e.g. 4,2,3,1

- "Work left to right" through your shuffled data: Find L(4), then L(4,2), then L(4,2,3), then L(4,2,3,1)
- Each time you add a training data, this counts towards its running marginal contribution
- Optional: if we hit some "performance threshold", stop (if we already got to our expected accuracy halfway through, just give all the remaining data points score of 0)

Influence functions (gradient based)

- "training instances only influence a model through training gradients"
- use Taylor series approximations and some assumptions
    - this is where gradient and hessian come from!
- static vs dynamic: do we just focus on the fixed final params
- major limitation wrt high influence

Assumptions

- stationarity: model params have converged
- convexity[1]

[1] https://en.wikipedia.org/wiki/Jensen's_inequality ↩

Basic idea

- if model f and loss L are twice-differentiable and strictly convex, we can calculate the change in loss when a training instance is upweighted by ϵ
- *without actually retraining*
- using closed form expression that takes into account the Hessian (based on second derivatives of loss wrt params) and gradient (first derivative of loss wrt params)
- it's not exact, it's a Taylor series

Hessian review

Hessian = matrix of second derivatives of loss function

General ML uses:

- Shows curvature of loss landscape
- Enables better optimization (Newton’s method)
- Helps detect sharp vs flat minima

For influence functions:

- Needed because influence = how params change when you perturb training data
- Inverse Hessian tells you how to scale updates in different directions
- Key part of influence function formula (appears as H^-1)

Main challenge:

- $O(p^2)$ parameters for p-param model
- Usually need approximations
- For influence functions, can use Hessian-vector products instead of full matrix

Worked example:

- https://stats.stackexchange.com/questions/68391/hessian-of-logistic-function

Recap

- TLDR: if we have the gradients and Hessians[1] we can compute in closed form how model loss will change with z_i is removed
- it's still an estimate, but we expect it to be a good estimate

1. https://en.wikipedia.org/wiki/Hessian_matrix ↩

What should we know for 419

- If we want to do active maths research in this area, good to understand everything in the paper
- For our purposes, we want to understand the big idea behind each kind of data value
- Aim to understand the intuitive description of what’s going on (H+L make an effort to provide this)
- Understand why we can’t compute "true" influence for most models
- Understand why the methods vary in time and space complexity

Code for influence

Check out several implementations, e.g.

https://github.com/kohpangwei/influence-release,

https://github.com/nimarb/pytorch_influence_functions/blob/master/pytorch_influ

https://github.com/alstonlo/torch-influence/blob/main/torch_influence/base.py

My suggested mental model: which "teams" are allowed

We can think of a 2-D space to organize these different methods

The first dimension is: which "teams" data points does a given influence method consider?

The second dimension is: how "close" to accurate do we want to be (do we want just a few samples or many samples))

- LOO says: only one team allowed: everybody else + you versus everybody else without you
    - can alternatively view this as effectively, no variety in teams allowed
- Downsampling says: we’ll consider a variety of teams, of some fixed size set by the researchers, say 70%
    - but effectively similar to LOO

What does this all have to do with human-centered AI

All these methods estimate what will happen in some counterfactual world in which the data changes.

What if people actually cause that data to change?

That’s where we’ll be going next…