CS 419M Introduction to Machine Learning **Spring 2021-22**

Lecture 4: Evaluation protocol and Ranking losses

Lecturer: Abir De *Scribes: Group 1 and Group 2*

4.1 Basic machine learning evaluation protocol

Suppose we have dataset with n items, $D = \{1, 2, \ldots, n\}$, where the i^{th} element has the feature x_i. We wish to create a classifier with the linear map, $w^\top x = y$, given the label y where it is assumed to be a continuous quantity. While for the discrete label case, $y \in \{-1, 1\}$, we take the map to be $sign(w^\top x)$.

4.1.1 Problem Description

We are given a training data-set $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$. Our objective is find a relation or map between x_i and y_i where $i \in \{1, 2, \ldots, n\}$. Most importantly, this model must work well on general unseen data too.

4.1.2 Evaluation metric

It is important to choose the following:

- Model (like linear, exponential etc.)
- Loss function (like logistic loss, hinge loss, etc.)

before building a map for the classifier problem. Difference choice of the above two will give us different maps. It should be noted that we can optimize for the parameters of the model that we choose, but we can't optimize the loss function keeping the model itself as the argument.

Since both the model and the loss function vary, we cannot minimize the overall performance with respect to the set of models and the set of loss functions. What is instead done is the following.

Take the optimization problem as:

$$\min_{w_m} \sum_{i \in D} l(x_i, y_i, m)$$

where, w_m is the parameter of the model m and l is the loss function. **Here l and m are kept fixed**. We then find the accuracy of this model using some accuracy function $acc(m, l)$ for various m and l choices, which is applied on the validation data-set (introduced ahead). The m and l which give the best accuracy is chosen.

4.1.3 Validation set

Given a dataset and a machine learning task, we have to fix the (a) Model which we will use, and (b) the loss function we use for this task.

We split our dataset D into a training dataset D_{Tr} and a validation dataset D_v.

- We train different models on D_{Tr}
- Measure accuracy on validation set D_v
- Fix model/loss so that accuracy is maximized on D_v

D: D_{Tr} (on which we fix a model loss function) $\xrightarrow{m}$ $D_v \rightarrow$ *accuracy*

Accuracy: This is a measure of correctly predicted classifications over total classifications. This can be defined in different ways for different tasks.

$$Accuracy = \begin{cases} -E[\mathbb{I}(y \neq m(x))] & \text{for discrete y} \\ -E[(y - m(x))^2] & \text{for continuous y} \end{cases} \tag{4.1}$$

Note: Loss and accuracy are different. Loss is just an approximation of the accuracy, and accuracy is the actual measure of our model's effectiveness.

4.2 Extension of previous losses to a ranking Loss

4.2.1 Need for Ranking loss

Assume that we have vectors x_1 to x_n

Our job is to rank these vectors in some order, i.e. to find a permutation of these vectors representing their ranking.

$\{x_1, x_2....x_n\} \rightarrow \Pi$(permutation of x_1 to x_n)$\rightarrow$ *sorted vectors*

For example, when Google displays search results, It needs to sort the importance order of web pages in response to a query. Thus ranking loss is required here. This can be done by tracking the time users spend on a web page in response to a query before checking other search results. The mouse pointer of computers is also tracked, hence it can be known which web pages links the user finds most interesting when all the search results are shown. This information is used to further fine- tune the parameters of the ranking model.
Other examples of applications include ranking of candidate profiles for hiring, friend recommendations on Facebook, connection recommendations on Linkedin, etc.

$\Delta D \xrightarrow{\Delta x_i (new\ user\ information)} w - \epsilon$ (change in final parameters of model)

ϵ here is inversely proportional to $\mid D \mid$.

We are given x_i and x_j and their relevance scores y_i and y_j. If $y_i > y_j$, we rank x_i higher than x_j, else we rank x_j higher.

Thus we need a function h such that given $x_i > x_j$, we should have $h(x_i) > h(x_j)$. $h(x_i)$ and $h(x_j)$ should be a good enough approximation for the given y_i and y_j, which are called the relevance scores.

We try to approximate h(x) as $W^T x$, and try to find a W which predicts the relevance scores in the best possible way.

4.2.2 Pairwise Ranking loss

Consider the above example where we need to rank (x_i, x_j), we want $h(x_i) > h(x_j)$ should mimic $y_i > y_j$, and we need to define loss function accordingly.

One way to define could be

$$\sum_{\substack{i,j \in D \\ i>j}} \left(e^{-(h(x_i)-h(x_j))(y_i-y_j)} \right)$$

Better way to define would be using Pairwise Ranking loss

$$\sum_{i,j \in D} [1 - sign(y_i - y_j)(h(x_i) - h(x_j))]_+$$

Here, we could have put sign function on whole function but then it will be very difficult to train such model as we might not be able to find h using any sensible algorithm

4.2.3 What if we try regression loss?

Let us assume $y_i \in [0, 1]$, and for simplicity let $y_i = i * 0.1$ and there be total 10 $y_i s$.

Regression loss can be then defined as $\sum[(y_i - W^T x_i)^2]$

Now even if overall loss is low but change in small fraction in some of the values will not affect the error, but it can jeopardise the overall ranking, specifically where there is small change. The pairwise Ranking loss gives us the benefit as even for small perturbations, the difference will cancel out and it will not affect the overall ranking.

These ranking errors can be very significant. For example, as discussed in Google Search example, if ordering error occurs at 100^{th} position, it doesn't matter, but if it occurs at the top, it will significantly affect the utility and quality of search. That's why, while evaluating the ranking order, we can use matrix which penalises mismatching at top place compared to bottom.

4.2.4 Significance of considering every comparison while ranking

While computing ranking pairwise loss, comparing every distinct pair of items is necessary. That is, for every $i, j \in D$, loss is computed for every $i \neq j$. For instance, for three items x_1, x_2, x_3, if $x_1 > x_2$ and $x_2 > x_3$, even though it is pretty evident that $x_1 > x_3$ without theoretically comparing the two items, practically they need to be compared while ranking. This is because while training, there is no guarantee that the first two comparisons are error-less. If either of the two comparisons happen to lead to wrong ranking, the third comparison will automatically end up as a wrong result as well. Hence, every distinct pair of items are compared to ensure that the chances of wrong ranking are minimum.

4.2.5 Evaluation of Ranking

Suppose if we are given an ideal order, say $y_1 > y_2 > ... > y_n$, we need the ranking to be $h(x_1) > h(x_2) > ... > h(x_n)$ for maximum accuracy. To evaluate the actual accuracy, let us say that for every $y_i > y_j$,

$$\begin{aligned} h(x_i) > h(x_j) &\to +1 \\ h(x_i) < h(x_j) &\to 0 \\ h(x_i) = h(x_j) &\to \frac{1}{2} \end{aligned}$$

In this case, the accuracy of the ranking can be evaluated as

$$\sum_{\substack{i,j \\ y_i > y_j}} \left[\mathbb{I}\left(h(x_i) > h(x_j)\right) + \frac{1}{2}\mathbb{I}\left(h(x_i) = h(x_j)\right) \right]$$

However, this type of evaluation is not used in platforms like Google or Amazon because it penalises every mistake in ranking equally, which should not be the case as accurate ranking of first few pages in search results should have higher priority than the ranking of later pages for better quality of search results.

4.2.6 Convexity of Ranking Pairwise Loss Function

For the simplest training models, $h(x)$ is a linear function. In such cases, loss functions are expected to be convex. It is easy to prove that the ranking pairwise loss function is convex. We know that the general loss function

$$\sum_{i \in D} \left[1 - \left(w^T x_i\right) y_i\right]_+$$

is convex. An equivalent expression for the ranking pairwise loss function can be written as

$$\sum_{i,j \in D} \left[1 - sign(y_i - y_j) w^T (x_i - x_j)\right]_+$$
$$= \sum_{i,j \in D} \left[1 - y_{ij} \left(w^T x_{ij}\right)\right]_+$$

Hence, the above function is also convex with respect to w.

4.3 Practice Problem

Q. Assume that we are given a set of features $\{(x_i, y_i) | i \in \{1, 2, ..., N\}\}$ with $x_i \in R^d, y \in \{-1, +1\}$. We wish to train a function h : $R^d \rightarrow$ R, so that sign(h(x)) = y. To that aim, we seek to solve the following:

$$\underset{h \in H}{\text{minimize}} \sum_{i=1}^{N} \mathbb{I}[sign(h(x_i)) \neq y_i]$$

where $\mathbb{I}$[A] = 1 if A is true and 0 if A is false. Moreover, H is set of all functions that map from $\mathbb{R}^d$ to $\mathbb{R}$. We will resort to many approximations of $\mathbb{I}[sign(h(x_i)) \neq y_i]$. So mark and explain which ones are good approximator:

1. $\max\{0, 1 - y_i.h(x_i)\}$
2. $\min\{0, 1 - y_i.h(x_i)\}$
3. $\frac{exp(-y_i.h(x_i))}{1+exp(-y_i.h(x_i))}$
4. $\frac{1}{1+exp(-y_i.h(x_i))}$

Ans.

1. Yes, (hinge loss): so we push h(x) to $h(x_i) > 1$ for $y_i = 1$ and $h(x_j) < -1$ for y_j = -1 to push loss towards 0. In turn, if $sign(h(x_i)) \neq y_i$, loss > 0 (implying it approximates $\mathbb{I}[sign(h(x_i)) \neq y_i]$)
2. No. If $sign(h(x)) \neq y$, $1 - h(x)y > 0 \implies$ the approximator will be equal to 0. Hence, it is a bad approximator of the function.
3. Yes. However, the function is not convex and hence a bad 'loss function'(we can work around this by taking a log of this function as in lecture 2).

   The expression can be written as

$$loss = \frac{1}{1 + exp(y_i.h(x_i)} \geq 0 \ \forall i$$

Logic: $Loss \to 0 \iff y_i h(x_i) \to \infty$

$$\iff y_i = 1 \text{ and } h(x_i) \to \infty$$

or,

$$y_i = -1 \text{ and } h(x_i) \to -\infty$$

In turn, if $sign(h(x_i))) \neq y_i$, the approximator is closer to 1. Hence, it is a good approximator of $\mathbb{I}[sign(h(x_i)) \neq y_i]$.

4. No. The since the function does the exact opposite of that in 3.

4.4 Universal Approximation Theorem

Informally, a two layer neural network can approximate *any* function. Caveats: The function must be continuous and the 'width' (or the number of neurons in a layer) must be varied. In fact the 'resolution' depends on this width:

Theorem 4.1. *Given any continuous function f(x) and some $\epsilon > 0$, there exists a Neural Network g(x) with one hidden layer (with a reasonable choice of non-linearity, e.g. sigmoid) such that* $\forall x, |f(x) - g(x)| < \epsilon$

This intuitive explanation might convince you. There is also an arbitrary depth variant of this theorem.

4.5 Group Details

Group	Name
1	Rohan Gupta
1	Neeraj Patidar
1	Saeel Sandeep Nachane
1	Akshat Jain
1	Siddharth Anand
2	Hiya Gada
2	Akshay Iyer
2	Divyansh Natani
2	Harshit Kashyap
2	Udita Mehta