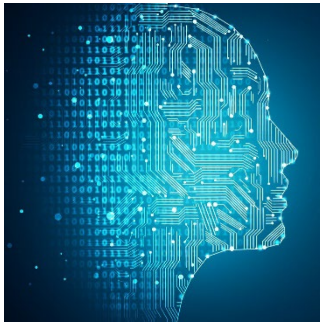
Machine Learning

Learning to Rank

Kevin Moon (kevin.moon@usu.edu)

STAT/CS 5810/6655



Learning to Rank

- Standard supervised learning tasks
  - **Regression**: Learn a function $f(\boldsymbol{x})$ to predict $y \in \mathbb{R}$
  - **Classification**: Learn a function $f(\boldsymbol{x})$ to predict $y \in \{1, \dots, C\}$
- Learning to rank (LTR) is also supervised
- **LTR**: Learn a function $f(q, D)$ to predict the order (ranking) of all items within the list D
  - q is a query

Example: Web Search Ranking

- Given a search query, rank the relevance of the resulting webpages
  - More relevant pages should be presented first
- Suppose we have a query q that results in n documents $D = \{d_1, \ldots, d_N\}$
- **Goal**: learn a function $f(q, D)$ that predicts the relevance of all documents in D
  - Can be presented in the form of an ordered list of the documents

Other LTR Examples

- Recommender systems
  - Rank a user's potential preferences
- Stock portfolio selection
  - Rank the potential return
- Message auto reply
  - Rank the potential replies
- Image captioning
  - Rank the possible captions
- Others…

LTR Approaches

Differ in their loss functions

- Pointwise
- Pairwise
- Listwise

- Recall the ranking task:
  - Given a query q and the resulting list of N objects $D = \{d_1, \dots, d_N\}$, learn a function $f(q, D)$ that ranks the objects in D

Pointwise LTR

- Reformulate LTR as a regression or classification task
- Score the relevance of each entry in the list independently
  - I.e. learn $f(q, d_i)$ instead of $f(q, D)$
- **Example**: two queries with matching results
  - $q_1 \rightarrow d_1, d_2$
  - $q_2 \rightarrow d_3, d_4, d_5$
- Training examples are query-document pairs:
  - $x_1: q_1, d_1$
  - $x_2: q_1, d_2$
  - $x_3: q_2, d_3$
  - $x_4: q_2, d_4$
  - $x_5: q_2, d_5$

Pointwise LTR

- Each document is scored independently with the absolute relevance as the label
- Apply regression (continuous relevance) or classification (discrete/categorical relevance)
- Advantages
  - Simple—existing methods can be applied
- Disadvantages
  - Often sub-optimal
  - Pointwise relevance labels are required
    - May not be available

Pairwise LTR

- Still trying to learn pointwise scoring function $f(q, d_i)$
- But training examples are pairs of documents w/ same query
  - $x_1: q_1, (d_1, d_2)$
  - $x_2: q_2, (d_3, d_4)$
  - $x_3: q_2, (d_3, d_5)$
  - $x_4: q_2, (d_4, d_5)$
- Obtain binary labels by comparing the individual relevance scores of each pair
- Reduces to a binary classification task

Pairwise LTR

- Often model the score difference probabilistically (using sigmoid function):

$$\Pr(d_i \text{ is preferred over } d_j) = \frac{1}{1 + \exp\left(-\left(f(q, d_i) - f(q, d_j)\right)\right)}$$

- Advantages
  - Model learns how to rank more directly
  - Only need pairwise preferences, which can be easier to collect
- Disadvantages
  - Scoring function is still pointwise – is still suboptimal

Listwise LTR

- Listwise methods model the probability of the entire ranking
- Training data:
  - $x_1 : q_1, (d_1, d_2)$
  - $x_2 : q_2, (d_3, d_4, d_5)$
- Multiple methods exist
  - ListNet was the first

ListNet

- Based on permutation probabilities
- $\pi =$ a specific permutation of a given list of length N
- $\phi(s_i) = f(q, d_i)$ is any increasing function of relevance score s_i given query q and document d_i

$$\Pr(\pi) = \prod_{i=1}^{N} \frac{\phi(s_i)}{\sum_{k=i}^{N} \phi(s_k)}$$

- Advantages
  - Theoretically sound approach to LTR
- Disadvantages
  - Computationally costly
  - Scoring functions still pointwise

ListNet

- To improve computational complexity, look at the top-one probability of each item
  - Sum of the permutation probabilities of permutations in which d_i is ranked on the top

$$\Pr(i) = \frac{\phi(s_i)}{\sum_{k=1}^{n} \phi(s_k)}$$

  - Still may be sub-optimal

LTR Metrics

- Binary metrics consider relevant vs. irrelevant
  - Mean average precision (MAP)
  - Mean reciprocal rank (MRR)
- Graded metrics consider the ranking among items
  - Normalized discounted cumulative gain (NDCG)
  - Expected reciprocal rank (ERR)

Mean Average Precision (MAP)

- Based on binary label of relevance
  - Relevant items should come before irrelevant items
- Define the precision at k items given query q:

$$P_k(q) = \frac{\sum_{i=1}^{k} r_i}{k}$$

  - r_i is the prediction of the ith document
    - $r_i = 1$ if d_i is relevant and 0 otherwise
- Average precision:

$$AP_k(q) = \frac{1}{\sum_{i=1}^{k} r_i} \sum_{j=1}^{k} P_j(q) r_j$$

- MAP averages across queries:

$$MAP = \frac{\sum_{q=1}^{Q} AP(q)}{Q}$$

Reciprocal Rank (RR)

- Focuses on the first correctly predicted relevant item
- Given a ranking list, let r_i be the rank of the highest relevant item
  - Reciprocal rank (RR) is $1/r_i$
- Mean reciprocal rank (MRR): average the RRs

$$MRR = \frac{1}{Q}\sum_{i=1}^{Q}\frac{1}{r_i}$$

Expected Reciprocal Rank (ERR)

- Recall the web search problem
- The likelihood a user looks at the document at rank i depends on his/her satisfaction with higher ranked documents
- ERR tries to quantify this
- Denote R_i as the probability the user is satisfied at position i
- Likelihood the user is satisfied and stops at position r:

$$\prod_{i=1}^{r-1}(1 - R_i)R_r$$

Expected Reciprocal Rank (ERR)

- Likelihood the user is satisfied and stops at position r:

$$\prod_{i=1}^{r-1}(1-R_i)R_r$$

- Let $g_i =$ labeled relevance of object at rank i
- Model for R_i:

$$R_i = \frac{2^{g_i}-1}{2^{g_{max}}}$$

$$ERR = \sum_{r=1}^{N}\frac{1}{r}R_r\prod_{i=1}^{r-1}(1-R_i)$$

Expected Reciprocal Rank (ERR)

$$ERR = \sum_{r=1}^{N} \frac{1}{r} R_r \prod_{i=1}^{r-1} (1 - R_i)$$

- ERR is calculated for a single query
  - Can average across queries
- ERR is a graded measure
  - Takes into account the relevance score
- ERR is less popular than NDCG due to computational complexity

Normalized Discounted Cumulative Gain (NDCG)

- Discounted Cumulative Gain at position k:

$$DCG_k = \sum_{i=1}^{k} \frac{2^{g_i} - 1}{\log_2(i+1)}$$

  - $g_i =$ labeled relevance at rank i
  - Numerator increases with relevance – called the "gain"
  - Denominator decreases with rank position – the "discounted" part
  - Prefers items with higher relevance to be ranked higher
- Normalized DCG divides by the maximum DCG you can get from a given ranking list
  - Sort the list based on true relevance labels

LTR Labeling Issues

Two approaches to obtain labels:

- Human judgement
  - Requires a lot of time and resources
  - May also be unreliable if labelers are not experts
- Derive them from user behavior
  - Try to determine latent ranking/preferences based on behavior
  - E.g. using click data to infer web page relevance
  - Often why pairwise methods are popular as pairwise labels are more plentiful and reliable

Recent LTR methods

- **RankNet** does pairwise ranking where $f(q, d_i)$ is parameterized using a neural network
  - Cross entropy is used as the loss function
- **LambdaNet** modifies RankNet by 1) improving speed of gradient calculations and 2) optimizing a ranking metric (e.g. NDCG)
- **LamdaMART** replaces the neural network in LambdaNet with gradient boosted regression trees
- **LambdaLoss** is a framework that provides theoretical justification that the model is optimizing a ranking metric

Further Reading

- Slides are adapted from https://everdark.github.io/k9/notebooks/ml/learning_to_rank/learning_to_rank.html