Outline

- Ranking from Pairwise Comparisons
- Learning to Rank
Ranking from Pairwise Comparisons
Example: Rating System for Chess Players

- How to rank chess players?
- Need a ranking system for video games, football, MLB, . . .

Figure from Wikipedia
Example: Multiclass Prediction

- One vs one model in multiclass/multilabel prediction problem.
- Goal: retrieve top $k$ labels.

<table>
<thead>
<tr>
<th>Testing data</th>
<th>Label</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>0.8</td>
<td>0.95</td>
<td>0.73</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td></td>
<td>0.5</td>
<td>0.4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td></td>
<td></td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Click-through data

- Use the click-through data to build a user-preference model
- Hard to get “rating data”, but easy to get comparison-based data
  Examples: Amazon, Ebay, Google, …
Problem Definition

- Given \( n \) items and a subset of pair comparisons \( \Omega \):
  \[
  Y_{ij} \in \{+1, 0, -1\} \quad \forall (i, j) \in \Omega
  \]

- Goal: output a total ranking or the score of each item \( s_1, s_2, \ldots, s_n \) where \( s_i \) is the rating of user \( i \) (the larger the better)

- What's the ranking?

```
  A > B
  B > C
  D > C
```
Let \( \bar{s}_1, \ldots, \bar{s}_n \) be the “correct scores”

\[ s_1, \ldots, s_n \] be the scores learned by the algorithm

Measure the difference between two rankings by **Kendall tau distance**

\[
\text{Kendall tau} = \frac{1}{n(n-1)/2} \sum_{(i,j): \bar{s}_i < \bar{s}_j} I(s_i > s_j)
\]

Measure number of misclassified pairs.
## Evaluation

### Real ranking:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

### Prediction:

<table>
<thead>
<tr>
<th>A,B</th>
<th>A,C</th>
<th>A,D</th>
<th>B,C</th>
<th>B,D</th>
<th>C,D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1&lt;2</td>
<td>1&lt;3</td>
<td>1&lt;4</td>
<td>2&lt;3</td>
<td>2&lt;4</td>
<td>3&lt;4</td>
</tr>
<tr>
<td>3&lt;4</td>
<td>3&gt;1</td>
<td>3&gt;2</td>
<td>4&gt;1</td>
<td>4&gt;2</td>
<td>1&lt;2</td>
</tr>
</tbody>
</table>

**Kendall tau = 4/6**
Elo Rating System

- Developed by Arpad Elo and officially used in United States Chess Federation (USCF) in 1960
- It is an online algorithm.
- For a match between player $A$ and $B$ with current scores $s_a$ and $s_b$
- Player $A$ wins with probability $E_A = \frac{Q_A}{Q_A + Q_B}$
- Player $B$ wins with probability $E_B = \frac{Q_B}{Q_A + Q_B}$
  where $Q_A = 10^{s_A/400}$, $Q_B = 10^{s_B/400}$
- If $A$ wins the game, then
  $s_A^{new} = s_A^{old} + K(1 - E_A)$
  $s_B^{new} = s_B^{old} + K(0 - E_B)$
- $K$ is called $K$-factor: can be adjusted over time
Example: $s_A = 2400$, $s_B = 2000$

$Q_A = 10^{2400/400} = 1,000,000$, $Q_B = 10^{2000/400} = 100,000$

Player A wins with probability

$E_A = 1,000,000/(1,000,000 + 100,000) = 0.91$

Player B wins with probability

$E_B = 100,000/(1,000,000 + 100,000) = 0.09$

If player A wins:

$s_A \leftarrow s_A + 32(1 - 0.91) = 2403$ (+3)

$s_B \leftarrow s_B + 32(0 - 0.09) = 1997$ (−3)

If player B wins:

$s_A \leftarrow s_A + 32(0 - 0.91) = 2371$ (−29)

$s_B \leftarrow s_B + 32(1 - 0.09) = 2029$ (+29)

The scores change more if the outcome does not match the expectation.
Glicko Rating System

- Proposed in (Glickman, 1999)
- Main idea: develop a better way to choose $K$?
  measurement of “rating reliability”, or “variance”
- Update scores by

$$s_A^{\text{new}} \leftarrow s_B^{\text{new}} + gK(\text{score} - E_A^{\text{old}})$$

- Change $g$ according to the outcome
Bradley-Terry Model

- Maximum likelihood estimator using a set $\Omega$ of pairwise comparisons.
- Probability that $Y_{ij} = 1$ ($i$ wins over $j$):

$$P(Y_{ij} = 1) = \frac{e^{s_i}}{e^{s_i} + e^{s_j}}$$

- The joint probability:

$$P(Y \mid s_1, \ldots, s_n) = \prod_{(i,j) \in \Omega} \frac{e^{s_i}}{e^{s_i} + e^{s_j}}$$

$$\log P(Y \mid s_1, \ldots, s_n) = \sum_{(i,j) \in \Omega} \log \left( \frac{e^{s_i}}{e^{s_i} + e^{s_j}} \right)$$

- Maximum likelihood:

$$\arg \min_{s_1, \ldots, s_n} - \sum_{(i,j) \in \Omega} \log \left( \frac{e^{s_i}}{e^{s_i} + e^{s_j}} \right)$$
Bradley-Terry Model

- Estimate $s_1, \ldots, s_n$ by solving an optimization problem
- Can be solved by any optimization method
- Example: using **Stochastic Gradient Method**:

$$\frac{\partial}{\partial s_i} \log \left( \frac{e^{s_i}}{e^{s_i} + e^{s_j}} \right) = 1 - \frac{e^{s_i}}{e^{s_i} + e^{s_j}}$$

- For $t = 1, 2, \ldots$
  - Randomly sample a pair $(i, j) \in \Omega$
  - Update $s_i$ and $s_j$ by: (assume $i$ wins)

$$s_i \leftarrow s_i + \eta_t \left( 1 - \frac{e^{s_i}}{e^{s_i} + e^{s_j}} \right)$$

$$s_j \leftarrow s_j + \eta_t \left( 0 - \frac{e^{s_j}}{e^{s_i} + e^{s_j}} \right)$$

- Exactly the same update rule with Elo and Glicko (expect the step size)!
Find $s_1, s_2, \ldots, s_n$ to minimize the error on the observed set $\Omega$

Can be formulated as an empirical risk minimization problem:

Find optimal parameters $s_1, \ldots, s_n$ by minimizing the ranking loss

\[
\min_{s_1, \ldots, s_n} \sum_{i, j \in \Omega} \ell(Y_{i,j}, s_i - s_j)
\]

Loss function $\ell$ can be

- Logistic loss (B-T model)
- Hinge loss (Ranking SVM): $\max(0, 1 - Y_{ij}(s_i - s_j))$
- \ldots
Balanced Rank Estimation

- Proposed in “Efficient Ranking from Pairwise Comparisons” in ICML 2013.
- The ranking is estimated by a simple cumulated sum:
  \[ s_i = \sum_{j: (i,j) \in \Omega} Y_{ij} \]

- Unbiased estimation, even in the noisy case
- \( O(|\Omega|) \) time.
- \( O(n) \) observations to achieve an \( \epsilon \)-Kendall tau error
Matrix Completion

- Proposed in Gleich and Lim, “Rank Aggregation via Nuclear Norm Minimization”, in KDD 2011.
- If we observe the score difference instead of the sign, rank aggregation can be formulated as a matrix completion problem.
- Problem setting: there is an underlying ranking $\bar{s}_1, \ldots, \bar{s}_n$, and we observe a subset of elements from $Y$ where $Y_{ij} = \bar{s}_i - \bar{s}_j$.
- $Y$ is rank 2!

$$
\begin{array}{c c}
5 & -1 \\
4 & -1 \\
3 & -1 \\
2 & -1 \\
1 & -1 \\
\end{array} \times \begin{array}{c c c c c}
1 & 1 & 1 & 1 & 1 \\
5 & 4 & 3 & 2 & 1 \\
\end{array} = \begin{array}{c c c c c}
0 & 1 & 2 & 3 & 4 \\
-1 & 0 & 1 & 2 & 3 \\
-2 & -1 & 0 & 1 & 2 \\
-3 & -2 & -1 & 0 & 1 \\
-4 & -3 & -2 & -1 & 0 \\
\end{array}
$$
Matrix Completion for Rank Aggregation

- Recover by solving the matrix completion problem:
  \[
  \arg\min_{\text{rank}(X)=2} \sum_{(i,j) \in \Omega} (X_{ij} - Y_{ij})^2
  \]

- Nonconvex form: find \( X^* = U^* (V^*)^T \) by solving
  \[
  \arg\min_{U,V} \sum_{(i,j) \in \Omega} ((UV^T)_{ij} - Y_{ij})^2 + \lambda \|U\|_F^2 + \lambda \|V\|_F^2
  \]

- Exact recover the underlying ranking with \( O(n \log^2 n) \) observed entries
  (Using the recovery guarantee for matrix completion)
Group Comparison

- Ranking players by group comparison

  Given $n$ items and a subset of group comparisons, what’s the ranking for each player?

  Examples: Halo, LOL, Heroes of the storm player ranking, ...
Learning to Rank
Document Retrieval

$D = \{d_1, d_2, \ldots, d_N\}$

Figure from “A short introduction to learning to rank” by Hang Li
Learning to Rank

Similar problems for other retrieval problems such as internet query.

Figure from “A short introduction to learning to rank” by Hang Li
Problem Formulation

- The original data set:
  - Training queries: $q_1, \ldots, q_m$
  - For each training query ($q_i$), a list of documents with rating
    
    $$(d_{i,1}, y_{i,1}), (d_{i,2}, y_{i,2}), \ldots, (d_{i,n_i}, y_{i,n_i})$$

- $y_{i,j}$ measures how relevant is this document to query $q_i$

  $$y_{i,u} > y_{i,v} \Rightarrow \text{ document } u \text{ is more relevant than } v$$
Problem Formulation

- The original data set:
  - Training queries: $q_1, \ldots, q_m$
  - For each training query ($q_i$), a list of documents with rating
    $$(d_{i,1}, y_{i,1}), (d_{i,2}, y_{i,2}), \ldots, (d_{i,n_i}, y_{i,n_i})$$
  - $y_{i,j}$ measures how relevant is this document to query $q_i$

$$y_{i,u} > y_{i,v} \Rightarrow \text{document } u \text{ is more relevant than } v$$

- Feature extraction: for each document-query pair ($q_i, d_{i,j}$), construct a $d$-dimensional feature $\varphi(q_i, d_{i,j}) = x_{i,j}$
- Goal: find a function $f(\cdot)$ such that

$$f(x_{i,u}) > f(x_{i,v}) \quad \text{if} \quad y_{i,u} > y_{i,v}$$
Empirical Risk Minimization

- Find the function by minimizing:

  \[
  \min_{f \in \mathcal{F}} \sum_{i=1}^{m} \sum_{(u,v) : y_{i,u} \neq y_{i,v}} \ell(f(x_{i,u}) - f(x_{i,v}), \text{sign}(y_{i,u} - y_{i,v})) + R(f)
  \]

  where \( R(f) \) is the regularization.

- Different function set \( \mathcal{F} \) corresponds to different algorithms
Empirical Risk Minimization

- Find the function by minimizing:

$$\min_{f \in \mathcal{F}} \sum_{i=1}^{m} \sum_{(u,v): y_i, u \neq y_i, v} \ell(f(x_i, u) - f(x_i, v), \text{sign}(y_i, u - y_i, v)) + R(f)$$

where $R(f)$ is the regularization.

- Different function set $\mathcal{F}$ corresponds to different algorithms

- Let’s assume $f(x) = w^T x$ is a linear function, then

$$f(x_{i,u}) - f(x_{i,v}) = w^T (x_{i,u} - x_{i,v})$$

- Linear Rank-SVM: let $\ell(\cdot)$ be the hinge loss

$$\min_w \frac{1}{2} \|w\|^2 + \sum_{i=1}^{m} \sum_{(u,v): y_i, u \neq y_i, v} \max(0, 1 - \text{sign}(y_i, u - y_i, v)w^T (x_{i,u} - x_{i,v}))$$
Rank SVM

- Equivalent to SVM with the following training set:

\[
\left\{ \left( x_{i,u} - x_{i,v}, \ \text{sign}(y_{i,u} - y_{i,v}) \right) \mid \forall (i, u, v) \text{ such that } y_{i,u} \neq y_{i,v} \right\}
\]

Figure from “A short introduction to learning to rank” by Hang Li.
Scalability

- If each query has \( n \) samples, then totally we have to generate \( mn^2 \) samples for the rank SVM problem.
- Observation: for each query, the ratings only have several levels:
  \[
y_{i,u} = \{0, 1, 2, \ldots, K\} \quad \text{for all } i, u
\]

  Microsoft Letor dataset: \( K \leq 4 \)

  Yahoo Learning to Rank: \( K \leq 4 \)

- If \( K \) is a constant, the computational complexity can be reduced to \( O(nd + n \log n) \)

  (See “Efficient Algorithms for Ranking with SVMs” by Chapelle and Keerthi)
Many other algorithms have been proposed:

- RankBoost
- AdaRank
- ListNet
- LambdaMart
- ...
Coming up

- Next class: Clustering and Dimensional Reduction

Questions?