AGENDA

1. Preference Learning Tasks (Eyke)
2. Loss Functions (Johannes)
3. Preference Learning Techniques (Eyke)
4. **Complexity of Preference Learning** (Johannes)
   a. Training Complexity
      - SVMRank
      - Pairwise Methods
   b. Prediction Complexity
      - Aggregation of Preference Relations is hard
      - Aggregation Strategies
      - Efficient Aggregation
5. Conclusions
Training Complexity: Number of Preferences

we have \( d \) binary preferences for items \( X = \{x_1, \ldots, x_c\} \)

- total ranking: \( d = \frac{c \cdot (c - 1)}{2} \)
- multi-partite ranking (\( k \) partitions with \( p_i \) items each): \( d = \sum_{i \neq j} p_i \cdot p_j \)
- bi-partite ranking (with \( p \) and \( c-p \) items): \( d = p \cdot (c - p) \) (e.g., multi-label classification)
- top rank: \( d = c - 1 \) (e.g. classification)
Training Complexity of Relational Approach

We generate one training example for each binary preference

- complexity of the binary base learner is $f(d)$
  - e.g. $f(d) = O(d^2)$ for a learner with quadratic complexity

Single-set ranking:

- We have $c$ items with ranking information
- Total complexity $f(d)$ depends on the density of the ranking information
  - quadratic in $c$ for (almost) full rankings
  - linear in $c$ for bipartite rankings with a constant $p$

Multi-set ranking:

- We have $n$ sets of $c$ items with ranking information
  - label ranking is a special case of this scenario
  - object ranking where multiple sets of objects are ranked is also a special case
- Total complexity is
  - $f(n \cdot d)$ for approaches where all preferences are learned jointly
    - can be more efficient if $f$ is super-linear and problem is decomposed into smaller subproblems (pairwise label ranking)
Example: Complexity of SVMRank

- **Reformulation as Binary SVM** [Herbrich et al. 2000, Joachims 2002]
  - $d$ constraints of the form $\mathbf{w}^T (x_i - x_j) \geq 1 - \xi_{ij}$
  - $d$ slack variables $\xi_{ij}$

  Total complexity: $f(d)$
  - where $f(.)$ is the complexity for solving the quadratic program
  - super-linear for conventional training algorithms like SMO, SVM-light, etc.

- **Reformulation as Structural SVM** [Joachims 2006]
  - $2^d$ constraints of the form $\frac{1}{d} \cdot \sum_{x_i > x_j} c_{ij}(x_i - x_j) \geq \frac{1}{d} \cdot \sum c_{ij} - \xi$
  - 1 slack variable $\xi$

  Total complexity: $d$

  - **Cutting-Plane algorithm:**
    - iterative algorithm for solving the above problem in linear time
      - iteratively find an appropriate subset of the constraints
      - convergence independent of $d$
    - further optimization could even yield a total complexity of $\min(n \cdot \log(n), d)$
Example: Complexity of Pairwise Label Ranking

- $n$ examples, $c$ classes, $d$ preferences in total, $\overline{d} = \frac{d}{n}$ preferences on average
  - decomposed into $\frac{c \cdot (c-1)}{2}$ binary problems
  - each problem has $n_{ij}$ examples $\sum_{ij} n_{ij} = d$

→ total training complexity

$$\sum_{ij} f(n_{ij}) \leq \overline{d} \cdot f(n) \leq f(d) = f \left( \sum_{ij} n_{ij} \right)$$

- upper bounds are tight if $f$ is linear
- big savings are possible super-linear complexities $f(n) = n^o (o > 1)$
  - distributing the same number of examples over a larger number of smaller dataset is more efficient

$$o > 1 \rightarrow \sum n_i^o < \left( \sum n_i \right)^o$$

[Hüllermeier et al. 2008]
Example: Complexity of Pairwise Classification

- Pairwise classification can be considered as a label ranking problem
  - for each example the correct class is preferred over all other classes

\[ \text{Total training complexity} \leq (c - 1) \cdot f(n) \]

For comparison:

- **Constraint Classification:**
  - Utility-based approach that learns one theory from all \((c - 1) \cdot n\) examples
  - Total training complexity: \(f((c - 1) \cdot n)\)

- **One-Vs-All Classification:**
  - different class binarization that learns one theory for each class
  - Total training complexity: \(c \cdot f(n)\)
AGENDA

1. Preference Learning Tasks (Eyke)
2. Loss Functions (Johannes)
3. Preference Learning Techniques (Eyke)
4. **Complexity of Preference Learning** (Johannes)
   a. Training Complexity
      - SVMRank
      - Pairwise Methods
   b. **Prediction Complexity**
      - Aggregation of Preference Relations is hard
      - Aggregation Strategies
      - Efficient Aggregation

5. Conclusions
Prediction Complexity

$f$ complexity for evaluating a single classifier, $c$ items to rank

- **Utility-Based Approaches:**
  - compute the utilities for each item: $c \cdot f$
  - sort the items according to utility: $c \cdot \log(c)$

- **Relational Approaches:**
  - compute all pairwise predictions: $\frac{c \cdot (c-1)}{2} \cdot f$
  - aggregate them into an overall ranking
    - method-dependent complexity

- Can we do better?
Aggregation is NP-Hard

- The key problem with aggregation is that the learned preference function may not be transitive.
  - Thus, a total ordering will violate some constraints

Aggregation Problem:
- Find the total order that violates the least number of predicted preferences.

- equivalent to the Feedback Arc Set problem for tournaments
  - What is the minimum number of edges in a directed graph that need to be inverted so that the graph is acyclic?
- This is NP-hard [Alon 2006]
  - but there are approximation algorithms with guarantees
  - For example, [Ailon et al. 2008]
    - propose Kwiksort, a straight-forward adaption of Quicksort to the aggregation problem
    - prove that it is a randomized expected 3-approximation algorithm
Aggregating Pairwise Predictions

- Aggregate the predictions $P(\lambda_i > \lambda_j)$ of the binary classifiers into a final ranking by computing a score $s_i$ for each class $I$

  - **Voting**: count the number of predictions for each class (number of points in a tournament)
    $$s_i = \sum_{j=1}^{c} \delta \left( P(\lambda_i > \lambda_j) > 0.5 \right)$$

  - **Weighted Voting**: weight the predictions by their probability
    $$s_i = \sum_{j=1}^{c} P(\lambda_i > \lambda_j)$$

- **General Pairwise Coupling problem** [Hastie & Tibshirani 1998; Wu, Lin, Weng 2004]
  - Given $P(\lambda_i > \lambda_j) = P(\lambda_i | \lambda_i, \lambda_j)$ for all $i, j$
  - Find $P(\lambda_i)$ for all $i$
  - Can be turned into a system of linear equations
Pairwise Classification & Ranking Loss
[Hüllermeier & Fürnkranz, 2010]

→ Weighted Voting optimizes Spearman Rank Correlation
   ▪ assuming that pairwise probabilities are estimated correctly

→ Kendall's Tau can in principle be optimized
   ▪ NP-hard (feedback arc set problem)

Different ways of combining the predictions of the binary classifiers optimize different loss functions
   ▪ without the need for re-training of the binary classifiers!

However, not all loss functions can be optimized
   ▪ e.g., 0/1 loss for rankings cannot be optimized
   ▪ or in general the probability distribution over the rankings cannot be recovered from pairwise information
Speeding Up Classification Time

- Training is efficient, but pairwise classification still has to
  - store a quadratic number of classifiers in memory
  - query all of them for predicting a class

Key Insight:
- Not all comparisons are needed for determining the winning class
- More precisely:
  - If class X has a total score of \( s \)
  - and no other class can achieve an equal score
  \( \rightarrow \) we can predict X even if not all comparisons have been made

Algorithmic idea:
- Keep track of the loss points
- if class with smallest loss has played all games, it is the winner
  \( \rightarrow \) focus on the class with the smallest loss
- Can be easily generalized from voting (win/loss) to weighted voting (e.g., estimated pairwise win probabilities)
Quick Weighted Voting
[Park & Fürnkranz, ECML 2007]

\[
\text{while } c_{\text{top}} \text{ not determined do}
\]
\[
c_a \leftarrow \text{class } c_i \in K \text{ with minimal } l_i;
\]
\[
c_b \leftarrow \text{class } c_j \in K \setminus \{c_a\} \text{ with minimal } l_j \text{ & classifier } C_{a,b} \text{ has not yet been evaluated;}
\]
\[\text{if } \text{no } c_b \text{ exists then}
\]
\[
c_{\text{top}} \leftarrow c_a;
\]
\[\text{else}
\]
\[
v_{ab} \leftarrow \text{Evaluate}(C_{a,b});
\]
\[
l_a \leftarrow l_a + (1 - v_{ab});
\]
\[
l_b \leftarrow l_b + v_{ab};
\]

select class with fewest losses
pair it with unplayed class with fewest losses
we're done if no such class can be found
evaluate the classifier and update loss statistics
Decision-Directed Acyclic Graphs
[Platt, Cristianini & Shawe-Taylor, NIPS 2000]

**DDAGS**
- construct a **fixed decoding scheme** with $c-1$ decisions
- unclear what loss function is optimized

**Comparison to QWeighted**
- DDAGs slightly faster
- but considerably less accurate

<table>
<thead>
<tr>
<th>dataset</th>
<th>JRip</th>
<th>NB</th>
<th>C4.5(J48)</th>
<th>SVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>vehicle</td>
<td>73.88</td>
<td>72.46</td>
<td>45.39</td>
<td>71.99</td>
</tr>
<tr>
<td>glass</td>
<td>74.77</td>
<td>74.30</td>
<td>49.07</td>
<td>71.50</td>
</tr>
<tr>
<td>image</td>
<td>96.62</td>
<td>96.41</td>
<td>80.09</td>
<td>96.93</td>
</tr>
<tr>
<td>yeast</td>
<td>58.96</td>
<td>58.09</td>
<td>57.55</td>
<td>58.56</td>
</tr>
<tr>
<td>vowel</td>
<td>82.42</td>
<td>76.67</td>
<td>63.84</td>
<td>82.93</td>
</tr>
<tr>
<td>soybean</td>
<td>94.00</td>
<td>93.56</td>
<td>92.97</td>
<td>93.56</td>
</tr>
<tr>
<td>letter</td>
<td>92.33</td>
<td>88.33</td>
<td>63.08</td>
<td>91.50</td>
</tr>
</tbody>
</table>

Accuracy: left - QWeighted, right - DDAG
Average Number of Comparisons for QWeighted algorithm

<table>
<thead>
<tr>
<th>dataset</th>
<th>c</th>
<th>$\frac{c(c-1)}{2}$</th>
<th>QW</th>
<th>DDAG</th>
</tr>
</thead>
<tbody>
<tr>
<td>vehicle</td>
<td>4</td>
<td>6</td>
<td>3,98</td>
<td>3</td>
</tr>
<tr>
<td>glass</td>
<td>7</td>
<td>21</td>
<td>9,75</td>
<td>6</td>
</tr>
<tr>
<td>image</td>
<td>7</td>
<td>21</td>
<td>8,75</td>
<td>6</td>
</tr>
<tr>
<td>yeast</td>
<td>10</td>
<td>45</td>
<td>15,87</td>
<td>9</td>
</tr>
<tr>
<td>vowel</td>
<td>11</td>
<td>55</td>
<td>17,42</td>
<td>10</td>
</tr>
<tr>
<td>soybean</td>
<td>19</td>
<td>171</td>
<td>27,65</td>
<td>18</td>
</tr>
<tr>
<td>letter</td>
<td>26</td>
<td>325</td>
<td>45,01</td>
<td>25</td>
</tr>
</tbody>
</table>

[Graph showing the relationship between the number of classes c and the number of comparisons for different datasets, including vehicle, glass, image, yeast, vowel, soybean, and letter.]
References

- T. Joachims, Training Linear SVMs in Linear Time, Proceedings of the ACM Conference on Knowledge Discovery and Data Mining (KDD), 2006