Aggregation

Aggregation: a mechanism for merging together two or more linguistic structures into a single sentence.

1 Dillon’s TD made the score 23-14.
2 Dillon scored in the fourth quarter on a 2-yard run.
3 The touchdown was set up by a 24-yard pass interference penalty on Routt.

Dillon’s TD made the score 23-14, and Dillon scored again in the fourth quarter on a 2-yard run set up by a 24-yard pass interference penalty on rookie Stanford Routt.

Outline

- Aggregation using integer linear programming
- Selection and ordering problems

Aggregation in Text Generation

(Passing (Cundiff 22/37 237 6.4 1 11))
(Passing (Carter 23/47 237 5.0 1 4))
(Interception (Lindell 1 52 1))
(Kicking (Lindell 3/3 100 38 1/1 10))
(Passing (Bledsoe 17/34 104 3.1 0 0))
(Rushing (Hambrick 13 33 2.5 10 1))
(Fumbles (Bledsoe 2 2 0 0 0))
Aggregation in Text Generation

- Input format: a set of pre-selected entries
- Training: a partitioning of database entries based on sentences which verbalize them
  - Locally: Learn compatibility function between pairs of entries
- Testing: predict a partitioning for a new set of entries
  - Apply pairwise compatibility function to the test set
  - Globally: Induce optimal partitioning using integer linear programming (ILP)

Learning a Compatibility Function

- Assumption: Entries that belong to the same sentence are related
- Similarity is assessed by comparing the values of database entries to be aggregated

<table>
<thead>
<tr>
<th>Player</th>
<th>CP/AT</th>
<th>YDS AVG</th>
<th>TD</th>
<th>INT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cundiff</td>
<td>22/37</td>
<td>237 6.4</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Carter</td>
<td>23/47</td>
<td>237 5.0</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Cundiff + Carter (3 attr) vs Hambrick + Carter (0 attr)

---

Learning a Compatibility Function

- Input: A pair of database entries
- Features: Based on attribute comparison
  - overlap in action (yes, no)
  - overlap in TDs (yes, no)
  - overlap in team (yes, no, partial)
  - ... 
- Learning: (Local) pairwise classification (e.g., maximum entropy)
  - output: \( c_{(e_i, e_j)} \), the probability of pair \( (e_i, e_j) \) to be aggregated
Beyond Pairwise Comparison

- We have to enforce partition consistency

- We want to incorporate global constraints
  - Sentence complexity
  - Number of sentences in a text
  - ...

Global Constraints: Transitivity

Goal: Enforce transitivity in the label assignment

\[
\text{if } x_{(e_i,e_j)} = 1 \text{ and } x_{(e_j,e_k)} = 1, \text{ then } x_{(e_i,e_k)} = 1
\]

Implementation:

\[
\forall e_i, e_j, e_k \quad x_{(e_i,e_k)} \geq x_{(e_i,e_j)} + x_{(e_j,e_k)} - 1
\]

Optimization Formulation: ILP

\( c_{(e_i,e_j)} \) the probability of pair \( (e_i,e_j) \) to be aggregated
\( x_{(e_i,e_j)} \) indicator variable

Goal: maximize the score of the global assignment

\[
\arg\max \sum_{(e_i,e_j) \in E \times E} c_{(e_i,e_j)} x_{(e_i,e_j)} + (1 - c_{(e_i,e_j)})(1 - x_{(e_i,e_j)})
\]

subject to

\[
x_{(e_i,e_j)} \in \{0,1\} \forall e_i, e_j \in E \times E
\]

Global Constraints: Hemingway vs. Faulkner

Manuel drank his brandy. He felt sleepy himself. It was too hot to go out into the town. Besides there was nothing to do. He wanted to see Zurito. He would go to sleep while he waited.

He did not feel weak, he was merely luxuriating in that supremely gutful lassitude of convalescence in which time, hurry, doing, did not exist, the accumulating seconds and minutes and hours to which it its well state the body is slave both waking and sleeping, now reversed and time now the lip-server and mendicant to the body’s pleasure instead of the body thrall to time’s headlong course.
Global Constraints: Sentence Length

Goal: Constrain the length of generated sentences

Implementation:
\[ \forall e_i \sum_{e_j \in E} x_{(e_i, e_j)} \leq k \]  \hspace{1cm} (3)

\(k\) is the average number of positively labeled pairs for an entry type

Solving ILP

- ILP is NP-hard (CLR, 1992)
- Mixed Integer Programming Solver can efficiently find a solution for our task

Evaluation Set-Up

- Corpus: aligned pairs of game summaries and NFL database
- Training: 300 texts, Testing: 68 texts, Development: 100 texts
- An average text contains 14.3 database entries verbalized in 9.1 sentences

Statistics on Aggregation

<table>
<thead>
<tr>
<th>Number of database entries per sentence</th>
<th>Relative Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.10</td>
</tr>
<tr>
<td>1</td>
<td>0.15</td>
</tr>
<tr>
<td>2</td>
<td>0.20</td>
</tr>
<tr>
<td>3</td>
<td>0.25</td>
</tr>
<tr>
<td>4</td>
<td>0.30</td>
</tr>
<tr>
<td>5</td>
<td>0.35</td>
</tr>
<tr>
<td>&gt;5</td>
<td>0.40</td>
</tr>
</tbody>
</table>
Results: Aggregation

- Precision (P) and Recall(R) are computed over the predicted links
- Baseline: single-link agglomerative clustering

<table>
<thead>
<tr>
<th>Method</th>
<th>Prec</th>
<th>Rec</th>
<th>F-score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Previous Methods</td>
<td>57.7%</td>
<td>66.9%</td>
<td>58.4%</td>
</tr>
<tr>
<td>Clustering</td>
<td>82.2%</td>
<td>65.4%</td>
<td>70.3%</td>
</tr>
</tbody>
</table>

Outline

- Aggregation using integer linear programming
- Selection and ordering problems

Selection and Ordering Problems in Generation and Summarization

- Task of selection and ordering appears in multiple contexts in text generation and summarization
  - Multidocument summarization: sentences are selected from multiple documents and are ordered into a text
  - Title Generation: words are selected from an input text and are ordered into a grammatical phrase
- Simple solution: decompose the task into two subtasks (selection and ordering)
  - We cannot guarantee that selected units can be ordered in a meaningful way

Our goal: simultaneously optimize selection and ordering preferences
Problem Formalization in a Graph-Theoretic Framework

We construct a weighted directed graph $G(V^*, E^*)$ such that:

- $V$ represents a set of selection units
- $E$ represents pairwise ordering scores between all pairs of vertices in $V$
- We add source vertex $s$ and sink vertex $t$
  - $V^* = V \cup \{s, t\}$
  - $E^* = E \cup \{(s, v) \forall v \in V\} \cup \{(v, t) \forall v \in V\}$
- Simplification: remove all vertex weights in our structure and instead shift the weight onto the incoming edges

The decoding task: Find the highest weighted acyclic path starting at $s$ and ending at $t$ with $k$ vertices in-between

Example: Decoding for Multidocument Summarization

- Vertices in the decoding graph represent sentences from different documents
- The selection scores represent the likelihood of a sentence to be extracted (can be obtained using the techniques discussed in the last lecture)
- The ordering scores capture precedence likelihood
- By selecting an acyclic path with the highest weight, we generate $k$-sentence summary that optimizes both selection and ordering scores.

Example: Decoding for Title Generation

- Vertices in the decoding graph represent words from an input text documents
- The selection scores represent the likelihood of a word to appear in a title
- The ordering scores represent scores obtained from a bigram language model
- By selecting an acyclic path with the highest weight, we generate a title of length $k$ that optimizes both selection and ordering scores.

Relation to Classical Problems

- The decoding problem is similar to the Traveling Salesman Problem

  Given a complete weighted graph (where the vertices would represent the cities, the edges would represent the roads, and the weights would be the cost or distance of that road), find a Hamiltonian cycle with the least weight.

- More specifically, our problem is an instance of the prize collecting traveling salesman problem, in which the salesman is required to visit $k$ vertices at best cost (Balas, 1989).
**ILP Formulation**

- \( w_{i,j} \) the weight of the \((i, j)\) edge
- \( I_{i,j} \) is an indicator variable that is set to 1 if the edge is selected for the optimal path and to 0 otherwise

The objective is then to maximize the following sum:

\[
\max \sum_{i \in V} \sum_{j \in V} w_{i,j} I_{i,j}
\]

This sum combines the weights of edges selected to be on the optimal path.

**Source-Sink Constraints**

- Exactly one edge originating at source \(s\) is selected:
  \[
  \sum_{j \in V} I_{s,j} = 1
  \]
- Exactly one edge ending at sink \(t\) is selected:
  \[
  \sum_{i \in V} I_{i,t} = 1
  \]

**Length Constraint**

- Exactly \(k + 1\) edges are selected:
  \[
  \sum_{i \in V} \sum_{j \in V} I_{i,j} = k + 1
  \]
- The \(k + 1\) selected edges connect \(k + 2\) vertices including \(s\) and \(t\).

**Balance Constraints**

- Every vertex \(v \in V\) has in-degree equal to its out-degree:
  \[
  \sum_{i \in V} I_{i,v} = \sum_{i \in V} I_{v,j} \quad \forall \ v \in V^*
  \]
- Note that with this constraint, a vertex can have at most one outgoing and one incoming edge.
Do you need to enforce acyclicity constraints?

This graph contains a cycle, while satisfying all of the above constraints.

**Acyclicity constraints (1)**

The variables $f_{i,j}$ number the edges on the path from 1 to $k + 1$

- The first edge gets the number $f_{i,j} = k + 1$
- The last edge gets the number $f_{i,j} = 1$
- All other edges get $f_{i,j} = 0$

To enforce this, we enforce the following capacity constraint:

$$0 \leq f_{i,j} \leq (k + 1) I_{i,j} \quad \forall i, j \in V$$

- When $I_{i,j} = 0$, this constraint forces $f_{i,j} = 0$
- When $I_{i,j} = 1$, this allows $0 \leq f_{i,j} \leq k + 1$

**Acyclicity constraints (2)**

We constrain demand variables $d_v$ by:

$$d_v = \sum_{i \in V} I_{i,v} \quad \forall v \in V^* - \{s\}$$

The right hand side sums the number of selected edges entering $v$, and will therefore be either 0 or 1.

- We now add variables $a_v$ and $b_v$ constrained by the equations:

$$a_v = \sum_{i \in V} f_{i,v} \quad b_v = \sum_{i \in V} f_{v,i}$$

$a_v$ is simply the $f$-value on the selected edge entering $v$, if one exists, and 0 otherwise.

$b_v$ is the $f$-value on the (at most one) selected edge leaving $v$

**Acyclicity constraints (3)**

Finally, we add the constraints

$$a_v - b_v = d_v \quad v \neq s$$

$$b_s = k + 1$$

$$a_t = 1$$

These constraints ensure that a $k + 1$ path must run from $s$ to $t$:

- previous constraints forced exactly one edge leaving $s$ to some $v$
- $b_s = k + 1$ means that the $f$-value on this edge must be $k + 1$
- $v$'s balance constraint means some selected edge leaves $v$
- $a_v - b_v = d_v$ means this edge must have $f$-value $k$
- Continue this way, building up a path
- balance constraints mean that the path must terminate at $t$
- $a_t = 1$ forces termination after exactly $k + 1$ edges
Performance on the title generation task

<table>
<thead>
<tr>
<th></th>
<th>Average</th>
<th>Median</th>
<th>ROUGE-L</th>
<th>Optimal Solutions (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Beam 1</td>
<td>0.6</td>
<td>0.4</td>
<td>0.0234</td>
<td>0.0</td>
</tr>
<tr>
<td>Beam 1345</td>
<td>368.6</td>
<td>224.4</td>
<td>0.2556</td>
<td>100.0</td>
</tr>
<tr>
<td>ILP</td>
<td>6,536.2</td>
<td>57.3</td>
<td>0.2556</td>
<td>100.0</td>
</tr>
</tbody>
</table>