Combinatorial Phrase Mining

Spring 2018 CS 512 Data Mining Principles
Sarah Christensen and Kent Quanrud
Presentation Overview

• Brief review of phrase mining

• Our approach
  • Problem formulation and model
  • Algorithm
  • Parallel implementation

• Results

• Future work
Phrase Mining Problem

- Phrases are multi-word lexical units such as compound nouns, scientific terms, entity names, and fixed collocations.
- Phrase mining seeks to take a document corpus (e.g. articles, titles, social media posts) and extract meaningful, high quality phrases.

My wife and I particularly enjoyed the mac and cheese.
Soft criteria for quality phrases (quick review)

- **Popularity**: Quality phrases should appear frequently

- **Concordance**: Phrase frequency deviates from what is expected

  e.g. “strong tea” vs “powerful tea”

- **Informativeness**: Indicates specific topic or concept

  e.g. “this paper” is frequent but not informative

- **Completeness**: Identify complete semantic units

  e.g. “vector machine” vs “support vector” vs “support vector machine”
Existing Approaches

• Automatic Term Recognition from NLP and IR communities
  • Careful combinations of learning and domain specific rules

• Alternatively, data driven text mining approaches work with non-standard text and can minimize human labor
  • Raw frequencies
  • Statistical based metrics
  • Supervised/semi-supervised learning
  • Non-Parametric methods
  & combinations thereof
Our Approach (at a high level)

We develop:

a) a *discrete* model for phrase mining,

b) a *combinatorial* algorithm with provable guarantees, and

c) an efficient (embarrassingly parallel) implementation.
Weighted Set Cover

**Input:**
- Elements $E = \{e_1, \ldots, e_m\}$, sets $N = \{S_1, \ldots, S_n \subseteq E\}$
- For each element $e \in E$, a positive weight $w(e) > 0$
- For each set $S \in N$, a positive cost
- A total budget $B$

**Goal:** Compute a collection of sets $T \subseteq N$ that
- Costs $\sum_{S \in T} c(S) \leq B$
- Maximizes weighted coverage $\sum_{e \in \overline{T}} w(e)$
  where $\overline{T} = \bigcup_{S \in T} S$
From Phrases to Sets

- Elements = Words
- Sets = Phrases
- A phrase covers underlying words wherever phrase appears in text corpus

Document 1: ... support vector machine ...
Document 2: ... machine learning ...
Document 3: ... a support vector is ...
Document 4: ... support vector machine ...

\[ S_1 \quad S_2 \quad S_3 \]

\[ e_1 \quad e_2 \quad e_3 \]
From Phrase Mining to Weighted Set Cover

• Weight of each word = \( \log\left(\frac{1}{\text{frequency of word}}\right) \)
  • “contribution to entropy”, rewards less frequent terms, e.g. punishes “the”

• Cost of a phrase = Number of documents a phrase appears in
  • Models the “descriptive complexity” of bag-of-phrases representation

• Selected phrases induce bag-of-phrases summary for each document.

• Goal: select phrases whose induced bag-of-phrases summaries:
  a) Maximize the total entropy of covered words
  b) Has total size at most a specified budget
Document 1:
... Kentucky fried chicken ...
... the fried chicken was crispy ...
... we will be back for the chicken ...

Document 2:
... chicken salad sandwich ...
... preferred fried chicken sandwich ...
... really tastes like chicken ...
The Chicken Example

Document 1:
... Kentucky fried chicken ...
... the fried chicken was crispy ...
... we will be back for the chicken...

Document 2:
... chicken salad sandwich ...
... preferred fried chicken sandwich ...
... really tastes like chicken ...

Words are the elements
The Chicken Example

**Document 1:**

... Kentucky fried chicken ...

... the fried chicken was crispy ...

... we will be back for the chicken...

**Document 2:**

... chicken salad sandwich ...

... preferred fried chicken sandwich ...

... really tastes like chicken ...

Phrases are the sets
The Chicken Example

Document 1:
... Kentucky fried chicken ...
... the fried chicken was crispy ...
... we will be back for the chicken ...

Document 2:
... chicken salad sandwich ...
... preferred fried chicken sandwich ...
... really tastes like chicken ...

Fried chicken has a cost of 2
The Chicken Example

Document 1:
... Kentucky fried chicken ...
... the fried chicken was crispy ...
... we will be back for the chicken...

Document 2:
... chicken salad sandwich ...
... preferred fried chicken sandwich ...
... really tastes like chicken ...

Fried chicken has a weight of

\[ f(S_{fried\ chicken}) = 3 \times (w_{fried} + w_{chicken}) \]
The Chicken Example

**Document 1:**

... Kentucky **fried chicken** ...
... the **fried chicken** was crispy ...
... we will be back for the chicken...

**Document 2:**

... **chicken salad sandwich** ...
... preferred **fried chicken** sandwich ...
... really tastes like chicken...

**Fried chicken** has a bang-for-buck ratio of

\[
f(S_{\text{fried chicken}}) \over \text{Cost}_{\text{fried chicken}}
\]
The Chicken Example

**Document 1:**

... Kentucky fried chicken...
... the fried chicken was crispy...
... we will be back for the chicken...

**Document 2:**

... chicken salad sandwich...
... preferred fried chicken sandwich...
... really tastes like chicken...

With fried chicken already selected,

\[ f_T(S_{Kentucky \ fried \ chicken}) = w_{Kentucky} \]
The Chicken Example

**Document 1:**
- Kentucky fried chicken
- the fried chicken was crispy
- we will be back for the chicken

**Document 2:**
- chicken salad sandwich
- preferred fried chicken sandwich
- really tastes like chicken

Goal is to select set of phrases to maximize total score
But stay within a budget
Properties of Quality Phrases

• **Popularity**: Quality phrases should appear frequently
  
  Higher frequency => Greater coverage

• **Concordance**: Phrase frequency deviates from what is expected
  
  Not concordance => lower frequency of phrase => less coverage

• **Informativeness**: Indicates specific topic or concept
  
  Phrases that occur uniformly across documents have higher cost
  
  Entropy weights for words reward more surprising terms

• **Completeness**: Identify complete semantic units
  
  Complete (i.e., longer) phrases have more coverage per occurrence
Uh-oh: Set Cover NP-Hard

- Weighted set cover is *submodular*, with well-studied properties.

- In particular, for any submodular function
  - Greedy algorithm gives a constant factor approximation
  - Greedy algorithm overshoots budget by bounded amount

**Theorem 2.3.** Let $\beta = \max_{e \in N} c(e)$ be the maximum cost of any item. Then the greedy-ratio returns a set $S$ with cost $\sum_{e \in S} c(e) < B + \beta$ and value $f(S) \geq (1 - e^{-1}) \text{OPT}$. 
Simple Greedy Algorithm

• While there is space in budget:

  1. For each phrase:
     • Compute marginal coverage
     • Score = \( \frac{\text{marginal coverage}}{\text{cost}} \)

  2. Select phrase w/ max score

• Good but slow:
  
  # iterations = # selected phrases
“Sideways Greedy” Heuristic

Start

\[ S = \emptyset \]

Calculate:

1. Word weights
2. Phrase costs

Calculate:  
1. Word weights
2. Phrase costs

While budget remains

1. Calculate marginal phrase weights
2. Update phrase scores

1. Calculate marginal phrase weights
2. Update phrase scores

1. Sort phrase scores
2. Select top phrase candidates until budget is full

1. Sort phrase scores
2. Select top phrase candidates until budget is full

1. Calculate incremental scores
2. Add phrases to \( S \) with incremental scores above threshold

1. Calculate incremental scores
2. Add phrases to \( S \) with incremental scores above threshold
“Sideways Greedy” Heuristic

Start

$S = \emptyset$

Calculate:
1. Word weights
2. Phrase scores

1. Calculate marginal phrase weights
2. Update phrase scores

1. Sort phrase scores
2. Select top phrase candidates until budget is full

$B$ iterations $\rightarrow$ 4 iterations (!)
Embarrassingly Parallel Implementation

- We implemented a simple version of MapReduce in Python.
- MapReduce to calculate word weights, phrase costs, and phrase weights in parallel.

```
Text Corpus

Document 1
Document 2
...
...
...
...

Map

(word₁, count₁)
(word₂, count₂)
...
(wordₘ, countₘ)

Reduce

(word₁, count₁)
(word₂, count₂)
...
(wordₘ, countₘ)

Select

(wordₖ, countₖ)
(wordₗ, countₗ)
```
Case Study

• We tested our algorithm on a dataset of Yelp reviews.
  • Dataset size: 4.2 GB
  • Number of unique words: ~810,000

• Rented a computer from Amazon Web Services.
  • 96 CPUs
  • 256 GB Memory

• Considered phrases of up to length 3 and had budget of 5*(# of documents)

• No stop words, POS tagging, or any text specific information incorporated
Case Study Selection of Phrases

1. bun bo hue
2. cirque du soleil
3. mon ami gabi
4. dim sum
5. banh mi
...
100. general tso's chicken
101. shaved ice
102. chocolate chip cookies
103. krispy kreme

500. dunkin donuts
501. save your money
502. manager on duty
503. nem nuong
504. peach cobbler
...
996. new york steak
997. baked goods are
998. dress code
999. kobe beef

2000. couple days later
2001. fried okra
2002. eager to help
2003. carne asada was
2004. never showed up
...
7000. like most places
7001. didn't make sense
7002. tell me what
7003. the beer list
<table>
<thead>
<tr>
<th>AutoPhrase</th>
<th>Greedy-Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>david copperfield</td>
<td>bun bo hue</td>
</tr>
<tr>
<td>shao mai</td>
<td>cirque du soleil</td>
</tr>
<tr>
<td>andy warhol</td>
<td>mon ami gabi</td>
</tr>
<tr>
<td>el campesino</td>
<td>dim sum</td>
</tr>
<tr>
<td>zu hause</td>
<td>banh mi</td>
</tr>
<tr>
<td>cajun creole</td>
<td>xiao long bao</td>
</tr>
<tr>
<td>blue jays</td>
<td>pico de gallo</td>
</tr>
<tr>
<td>jose cuervo</td>
<td>frozen hot chocolate</td>
</tr>
<tr>
<td>suckling pig</td>
<td>sticky toffee pudding</td>
</tr>
<tr>
<td>profit margin</td>
<td>dulce de leche</td>
</tr>
<tr>
<td>il fornaio</td>
<td>tacos el gordo</td>
</tr>
<tr>
<td>tofu hut</td>
<td>pad kee mao</td>
</tr>
<tr>
<td>grovewood tavern</td>
<td>chilean sea bass</td>
</tr>
<tr>
<td>jennifer aniston</td>
<td>laser hair removal</td>
</tr>
<tr>
<td>en masse</td>
<td>foie gras</td>
</tr>
<tr>
<td>bob marley</td>
<td>hot n juicy</td>
</tr>
</tbody>
</table>
Conclusions

• *Discrete, maximum entropy* approach can lead to quality phrases
  • Without training, domain specific techniques or much parametrization (not even stop words!)
• Sideways Greedy demonstrates that combinatorial approach may be *scalable*
Future Work

• Clear path to greater scalability:
  • Sketching data structures
  • Importance sampling
• More testing (particularly on emerging datasets)
• Applying general set cover solver to other problems