Data Mining

Text Mining
Outline

1. What is Text Mining?
2. Text Preprocessing
3. Feature Creation
4. Feature Selection
5. Solving Downstream Tasks
Motivation for Text Mining

Approximately 90% of the world’s data is held in unstructured formats.

Source: Oracle Corporation

- Examples:
  - web pages
  - emails
  - customer complaint letters
  - corporate documents
  - scientific papers
  - books in digital libraries
The extraction of implicit, previously unknown and potentially useful information from large amounts of textual resources.
Search versus Discovery

**Search/Query**
- Query Processing
- Information Retrieval

**Discovery**
- Data Mining
- Text Mining

**Structured Data**
- Goal-oriented

**Text**
- Opportunistic
Some Text Mining Applications

1. Classification of news stories
2. Email and news filtering / SPAM detection
3. Sentiment analysis
4. Grouping of documents or web pages
5. Information extraction
6. Text summarization / Text generation
7. Question answering
Mixture of Document Clustering and Classification
Sentiment Analysis

- The goal of sentiment analysis is to determine the polarity of a given text at the document, sentence, or feature/aspect level.

- Polarity values
  - positive, neutral, negative
  - likert scale (1 to 10)

- Application examples
  - Document level
    - analysis of tweets about politicians
  - Feature/aspect level
    - analysis of product reviews

![Image of sentiment analysis results for politicians and product features]
Information Extraction

- Information extraction is the task of automatically extracting structured information from unstructured or semi-structured documents.

- Subtasks
  1. Named Entity Recognition and Disambiguation
     - “The parliament in Berlin has decided …“
     - Which parliament? Which Berlin?
  2. Relationship Extraction
     - PERSON works for ORGANIZATION
     - PERSON located in LOCATION
  3. Fact Extraction
     - CITY has population NUMBER
     - COMPANY has turnover NUMBER [Unit]
The Text Mining Process

1. **Text Preprocessing**
   - syntactic and/or semantic analysis

2. **Feature Generation**
   - bag of words, word embeddings

3. **Feature Selection**
   - reduce large number

4. **Data Mining**
   - clustering
   - classification
   - association analysis
2. Text Preprocessing

1. Tokenization
2. Stopword Removal
3. Stemming
4. POS Tagging
Syntactic and Linguistic Text Preprocessing

- Simple Syntactic Processing
  • Text Cleanup (remove punctuation and HTML tags)
  • Tokenization (break text into single words or N-grams)

- Advanced Linguistic Processing
  • Word Sense Disambiguation
    • determine which sense a word is having.
    • normalize synonyms (United States, USA, US)
    • normalize pronouns (he, she, it)
  • Part Of Speech (POS) Tagging
    • parse sentences according to grammar
    • determine function of each term
    • e.g. John (noun) gave (verb) the (det) ball (noun)
Stopword Removal

- Many of the most frequently used words in English are likely to be useless for text mining.

- These words are called **Stopwords**
  - examples: the, of, and, to, an, is, that, …
  - typically text contains about 400 to 500 such words
  - for an application, an additional domain specific stopwords list may be constructed

- Why should we remove stopwords?
  - Reduce data set size
    - stopwords account for 20-30% of total word count
  - Improve effectivity of text mining methods
    - stopwords may confuse the mining algorithm
More Examples of Stopwords

about, above, across, after, again, against, all, almost, alone, along, already, also, although, always, am, among, an, and, another, any, anybody, anyone, anything, anywhere, are, areas, aren’t, around, as, ask, asked, asking, asks, at, away, b, back, backed, backing, backs, be, became, because, become, becomes, been, before, began, behind, being, beings, below, best, better, between, big, both, but, by, c, came, can, cannot, can’t, case, cases, certain, certainly, clear, clearly, come, couldn’t, d, did, didn’t, differ, different, differently, do, does, doesn’t, doing, done, don’t, down, downed, downing, downs, during, e, each, early, either, end, ended, ending, ends, enough, even, evenly, ever, everybody, everyone, everything, everywhere, f, face, faces, fact, facts, far, felt, few, find, finds, first, for, four, from, full, fully, further, furthered, furthering, furthers, g, gave, general, generally, get, gets, give, given, gives, go, going, good, goods, got, greater, greatest, group, grouped, grouping, groups, h, had, hadn’t, has, hasn’t, have, haven’t, having, he, he’d, he’ll, her, here, he’s, hers, himself, himself, his, how, however, i, i’d, if, i’ll, i’m, important, in, interest, interested, interesting, interests, into, is, isn’t, it, its, itself, i’ve, j, just, k, keep, keeps, kind, knew, know, known, knows, l, large, largely, last, later, latest, least, less, let, lets, like, likely, long, longer, longest, m, made, make, making, man, many, may, me, members, men, might, more, mostly, mr, mrs, much, must, mustn’t, n, necessary, need, needed, needing, needs, never, new, newer, newest, next, no, nobody, non, noone, nor, nothing, nowhere, number, numbers, o, of, off, often, old, older, oldest, on, once, one, only, open, opened, opening, opens, or, order, ordered, ordering, orders, other, others, ought, our, ours, ourselves, out, over, own, p, part, parted, parting, parts, per, perhaps, place, places, point, pointed, pointing, points, possible, present, presented, presenting, presents, problem, problems, put, puts, q, quite, r, rather, really, right, room, rooms, s, said, same, saw, say, says, second, seconds, see, seem, seemed, seeming, seems, sees, several, shall, shan’t, she, she’d, she’ll, she’s, should, shouldn’t, show, showed, showing, shows, side, sides, since, small, smaller, smallest, so, some, somebody, someone, something, somewhere, state, states, still, such, sure, t, take, taken, than, that, that’s, the, their, theirs, them, themselves, then, there, therefore, there’s, these, they, they’d, they’ll, they’re, there, they’ve, thing, things, think, thinks, this, those, though, thought, thoughts, three, through, thus, to, today, together, too, took, toward, turn, turned, turning, turns, two, under, until, upon, up, upon, use, used, uses, v, very, w, want, wanted, wanting, wants, was, wasn’t, way, ways, we’d, well, we’ll, wells, went, were, we’re, weren’t, we’ve, what’s, when, when’s, where, where’s, whether, which, while, who, whole, whom, who’s, whose, why, why’s, will, within, without, won’t, work, worked, working, works, would, wouldn’t, x, y, year, years, yes, you, you’d
Stemming

- Techniques to find the **stem of a word**
  - words: User, users, used, using → Stem: use
  - words: Engineering, engineered → Stem: engineer

- Usefulness for Text Mining
  - improve effectivity of text mining methods
    - matching of similar words
  - reduce term vector size
    - combing words with same stem may reduce the term vector as much as 40-50%
Some Basic Stemming Rules

- remove endings
  - if a word ends with a consonant other than s, followed by an s, then delete s.
  - if a word ends in es, drop the s.
  - if a word ends in ing, delete the ing unless the remaining word consists only of one letter or of th
  - if a word ends with ed, preceded by a consonant, delete the ed unless this leaves only a single letter
  - ......

- transform words
  - if a word ends with “ies” but not “eies” or “aies” then “ies → y”
Text Preprocessing in Python

- Simple preprocessing including stopword removal

```python
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.datasets import load_files

# Load documents
docs = load_files('directory_of_files', encoding='utf-8')

# Vectorize documents
vectorizer = CountVectorizer(analyzer='word', stop_words='english')
matrix = vectorizer.fit_transform(docs)
```

- Stemming using the Natural Language Toolkit (NLTK) library

```python
from nltk.stem.porter import PorterStemmer

# Stem tokens
stemmer = PorterStemmer()
tokens = ['Jupiter', 'is', 'the', 'largest', 'gas', 'planet']
stems = []
for item in tokens:
    stems.append(stemmer.stem(item))
```

https://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html
https://www.nltk.org/book/ch03.html
Text Preprocessing in RapidMiner

- To use the operators, you need to install the Text Processing Extension
3. Feature Generation

1. Bag-of-Words
2. Word Embeddings
### Bag-of-Words: The Term-Document Matrix

| Term     | A  | B  | C  | D  | E  | F  | G  | H  | I  | J  | K  | L  | M  | N  | O  | P  | Q  | R  | S  | T  | Σ   |
|----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| oil      | 5  | 12 | 2  | 1  | 1  | 7  | 3  | 3  | 5  | 9  | 5  | 4  | 5  | 4  | 3  | 4  | 5  | 3  | 3  | 1  | 85  |
| price    | 5  | 6  | 2  | 2  | 0  | 8  | 1  | 2  | 2  | 10 | 5  | 1  | 5  | 2  | 0  | 3  | 3  | 3  | 3  | 0  | 63  |
| opec     | 0  | 15 | 0  | 0  | 0  | 8  | 1  | 2  | 2  | 6  | 5  | 2  | 2  | 4  | 0  | 0  | 0  | 0  | 0  | 0  | 47  |
| mln      | 0  | 4  | 0  | 0  | 2  | 4  | 1  | 0  | 0  | 3  | 9  | 0  | 0  | 0  | 0  | 0  | 3  | 3  | 0  | 2   | 31  |
| market   | 2  | 5  | 0  | 0  | 0  | 3  | 0  | 2  | 0  | 10 | 1  | 2  | 2  | 0  | 0  | 0  | 0  | 0  | 3  | 0  | 30  |
| barrel   | 2  | 0  | 1  | 1  | 0  | 4  | 0  | 1  | 3  | 3  | 0  | 1  | 1  | 0  | 3  | 3  | 1  | 0  | 2   | 26  |
| bpd      | 0  | 4  | 0  | 0  | 0  | 7  | 0  | 0  | 3  | 6  | 0  | 0  | 2  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 23  |
| dirs     | 2  | 0  | 1  | 2  | 2  | 0  | 1  | 0  | 0  | 4  | 2  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 1  | 1  | 5  | 23  |
| crude    | 2  | 0  | 2  | 3  | 0  | 2  | 0  | 0  | 0  | 5  | 2  | 0  | 2  | 0  | 0  | 0  | 0  | 0  | 2  | 0  | 1   | 21  |
| saudi    | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 1  | 0  | 5  | 7  | 1  | 4  | 0  | 0  | 0  | 0  | 0  | 18  |
| kuwait   | 0  | 0  | 0  | 0  | 0  | 0  | 10 | 0  | 1  | 0  | 3  | 0  | 1  | 0  | 2  | 0  | 0  | 0  | 0  | 0  | 17  |
| offici   | 0  | 0  | 0  | 0  | 0  | 0  | 5  | 1  | 1  | 0  | 1  | 4  | 3  | 1  | 0  | 0  | 0  | 0  | 0  | 0  | 17  |
| meet     | 0  | 6  | 0  | 0  | 0  | 0  | 3  | 0  | 1  | 0  | 1  | 0  | 1  | 0  | 2  | 0  | 0  | 0  | 0  | 0  | 14  |
| pct      | 0  | 0  | 0  | 0  | 0  | 2  | 0  | 2  | 2  | 2  | 1  | 0  | 0  | 1  | 0  | 0  | 1  | 1  | 0  | 0  | 2   | 14  |
| product  | 1  | 6  | 0  | 0  | 0  | 0  | 0  | 0  | 4  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 1   | 13  |
| accord   | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 5  | 1  | 0  | 2  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 12  |
| futur    | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 12  |
| minist   | 0  | 0  | 0  | 0  | 0  | 0  | 3  | 0  | 0  | 1  | 3  | 1  | 2  | 1  | 0  | 0  | 0  | 0  | 0  | 0  | 12  |
| govern   | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 5  | 0  | 6  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 11  |
| month    | 0  | 1  | 0  | 0  | 0  | 0  | 2  | 2  | 0  | 1  | 0  | 5  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 11  |
| report   | 0  | 1  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 11  |
| sheikh   | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 11  |
| industri | 0  | 2  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 1  | 2  | 0  | 1  | 0  | 0  | 10  |
| produc   | 0  | 0  | 0  | 0  | 0  | 0  | 4  | 1  | 1  | 0  | 3  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 10  |
| quota    | 0  | 2  | 0  | 0  | 0  | 0  | 4  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 10  |
| reserv   | 0  | 0  | 0  | 0  | 0  | 3  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 10  |
| world    | 0  | 1  | 0  | 0  | 0  | 0  | 1  | 3  | 0  | 0  | 0  | 0  | 0  | 1  | 1  | 0  | 0  | 0  | 0  | 1  | 10  |

Σ | 48 | 204 | 34 | 39 | 46 | 219 | 219 | 73 | 161 | 180 | 208 | 57 | 61 | 54 | 56 | 68 | 89 | 44 | 147 | 32 | 2039

The above table represents the term-document matrix for a document collection. Each cell in the matrix represents the frequency of a term in a document.
- Document is treated as a bag of words (or terms)
  - each word/term becomes a feature
  - order of words/terms is ignored

- Each document is represented by a vector

- Different techniques for vector creation:
  1. Binary Term Occurrence: Boolean attributes describe whether or not a term appears in the document (one-hot encoding)
  2. Term Occurrence: Number of occurrences of a term in the document (problematic if documents have different length)
  3. Terms Frequency: Attributes represent the frequency in which a term appears in the document (number of occurrences / number of words in document)
  4. TF-IDF: see next slide
The TF-IDF Term Weighting Scheme

- The TF-IDF weight (term frequency–inverse document frequency) is used to evaluate how important a word is to a corpus of documents.
  - TF: Term Frequency (see last slide)
  - IDF: Inverse Document Frequency.
    
    \[ w_{ij} = tf_{ij} \times idf_i. \]
    
    \[ idf_i = \log \frac{N}{df_i} \]
    
    - \( N \): total number of docs in corpus
    - \( df_i \): the number of docs in which \( t_i \) appears

- Gives more weight to rare words
- Give less weight to common words (domain-specific stopwords)
Feature Generation in RapidMiner and Python

1. Specify files to process

2. Select feature generation method

Python

```python
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer

binary_term_occurrences_vectorizer = CountVectorizer(binary='True')
term_occurrences_vectorizer = CountVectorizer(binary='False')
term_frequency_vectorizer = TfidfVectorizer(use_idf='False')
tf_idf_vectorizer = TfidfVectorizer(use_idf='True')
```
Word Embeddings

- Embeddings represent words not as a single number in a word vector but represent each word as a vector of real numbers (distributed representation) e.g. 300 numbers

- Embeddings are chosen in a way that semantically related words (e.g. dog, puppy) end up at similar locations in the vector space
  - thus, embeddings can deal better with synonyms and related terms than bag-of-words vectors

- Embeddings are calculated based on the assumption that similar words appear in similar contexts (distributional similarity)
  - Skip-gram approach used by Word2Vec: predict context words for each word using a neural net
Embedding Methods and Pretrained Models

- Well known word embedding methods
  - **Word2Vec** (Google)
  - **GloVe** (Stanford NLP Group)
  - **fastText** (Facebook AI Research)

- pretrained embeddings can be downloaded
  - GloVe: trained on Common Crawl, Wikipedia, and Tweets
  - fastText: embeddings for 294 languages

- using Embeddings
  - Python: Gensim offers Word2Vec implementation
  - RapidMiner: Word2Vec extension on marketplace

- embeddings are used as features in downstream tasks
  - together with traditional symbolic methods as part of neural architectures

GloVe 50 embedding of the word the pretrained on Common Crawl

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.418</td>
<td>0.24968</td>
<td>-0.41242</td>
<td>0.1217</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.34527</td>
<td>-0.044457</td>
<td>-0.49688</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.17862</td>
<td>-0.00066023</td>
<td>-0.6566</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.27843</td>
<td>-0.14767</td>
<td>-0.55677</td>
<td>0.14658</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-0.0095095</td>
<td>0.011658</td>
<td>0.10204</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.12792</td>
<td>-0.8443</td>
<td>-0.12181</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.016801</td>
<td>-0.33279</td>
<td>-0.1552</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.23131</td>
<td>-0.19181</td>
<td>-1.8823</td>
<td>-0.76746</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.099051</td>
<td>-0.42125</td>
<td>-0.19526</td>
<td>4.0071</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-0.18594</td>
<td>-0.52287</td>
<td>-0.31681</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.0059213</td>
<td>0.0074449</td>
<td>0.17778</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.15897</td>
<td>0.012041</td>
<td>-0.054223</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.29871</td>
<td>-0.15749</td>
<td>-0.34758</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.045637</td>
<td>-0.44251</td>
<td>0.18785</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.0027849</td>
<td>-0.18411</td>
<td>-0.11514</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-0.78581</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
4. Feature Selection

- Not all features help!
- Learners might have difficulty with high dimensional data
Pruning Document Vectors in RapidMiner and Python

- Prune methods
  - specify if and how too frequent or too infrequent words should be ignored

- Different options:
  - **Percentual**: ignore words that appear in less / more than this percentage of all documents
  - **Absolute**: ignore words that appear in less / more than that many documents

```python
vectorizer = TfidfVectorizer(min_df=0.1, max_df=0.3) # Percentual
vectorizer = TfidfVectorizer(min_df=5, max_df=20) # Absolute
```
POS tagging may be helpful for feature selection
sometimes you want to focus on certain classes of words:
  - Adjectives (JJ.) for sentiment analysis
    - good, bad, great
  - Nouns (N.) for text clustering
    - red and blue cars are similar
    - red and blue trousers are similar

Rapidminer supports
  - PENN tag system for English
  - STTS tag system for German
  - filtering conditions are expressed as regular expressions

Python: NLTK library supports PENN tag system for English
5. Solving Downstream Tasks

Methods:
1. Cluster Analysis
2. Classification
3. Association Analysis
5.1 Document Clustering

Goal
- Given a set of documents and a similarity measure among documents find clusters such that
  - documents in one cluster are more similar to one another
  - documents in separate clusters are less similar to one another
- using some clustering algorithm

Applications
- Topical clustering of news stories
- Topical clustering of social media posts
- Grouping of document versions

Question
- Which similarity measures are a good choice for comparing document vectors?
The Jaccard coefficient is a popular similarity measure for vectors consisting of asymmetric binary attributes:

\[
dist(x_i, x_j) = \frac{M_{11}}{M_{01} + M_{10} + M_{11}}
\]

Number of 11 matches / number of not-both-zero attribute values

- used together with binary term occurrence vectors (one-hot vectors)
  - 1 represents occurrence of specific word
  - 0 represents absence of specific word
  - most values are 0 as only small subset of the vocabulary is used in a document
Example: Jaccard Coefficient

- Example document set
  
  \( d_1 = \) “Saturn is the gas planet with rings.”
  
  \( d_2 = \) “Jupiter is the largest gas planet.”
  
  \( d_3 = \) “Saturn is the Roman god of sowing.”

- Documents as binary term occurrence vectors

<table>
<thead>
<tr>
<th></th>
<th>Saturn</th>
<th>is</th>
<th>the</th>
<th>gas</th>
<th>planet</th>
<th>with</th>
<th>rings</th>
<th>Jupiter</th>
<th>largest</th>
<th>Roman</th>
<th>god</th>
<th>of</th>
<th>sowing</th>
</tr>
</thead>
<tbody>
<tr>
<td>( d_1 )</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( d_2 )</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( d_3 )</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- Jaccard similarities between the documents
  
  \( \text{sim}(d_1,d_2) = 0.44 \)
  
  \( \text{sim}(d_1,d_3) = 0.27 \)
  
  \( \text{sim}(d_2,d_3) = 0.18 \)
Cosine Similarity

- Popular similarity measure for comparing weighted document vectors such as term-frequency or TF-IDF vectors

\[
\cos(d_1, d_2) = \frac{d_1 \cdot d_2}{\|d_1\| \|d_2\|}
\]

where \( \cdot \) indicates vector dot product and \( \|d\|\) is the length of vector \(d\)

- Example

\[
d_1 = \begin{pmatrix} 3 & 2 & 0 & 5 & 0 & 0 & 0 & 2 & 0 & 0 \end{pmatrix}
\]

\[
d_2 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 2 \end{pmatrix}
\]

\[
d_1 \cdot d_2 = 3*1 + 2*0 + 0*0 + 5*0 + 0*0 + 0*0 + 0*0 + 2*1 + 0*0 + 0*2 = 5
\]

\[
\|d_1\| = \sqrt{3^2+2^2+0^2+5^2+0^2+0^2+0^2+2^2+0^2+0^2+0^2} = 6.481
\]

\[
\|d_2\| = \sqrt{1^2+0^2+0^2+0^2+0^2+0^2+0^2+1^2+0^2+2^2} = 2.245
\]

\[
\cos(d_1, d_2) = 0.3150
\]
Example: Cosine Similarity and TF-IDF

- A **commonly used combination** for text clustering
- Each document is represented by vectors of TF-IDF weights
- Sample document set:
  - “Saturn is the gas planet with rings.”
  - “Jupiter is the largest gas planet.”
  - “Saturn is the Roman god of sowing.”

- First document as TF-IDF vector:
  \[ w_{ij} = tf_{ij} \times idf_i \]
  \[ idf_i = \log \frac{N}{df_i} \]
  \( (1/7 \times \log(3/2), 1/7\times\log(3/3), 1/7\times\log(3/3), \ldots, 0, 0, 0, \ldots) \)

Saturn is the Jupiter largest Roman
Sample document set

- \( d_1 = \text{“Saturn is the gas planet with rings.”} \)
- \( d_2 = \text{“Jupiter is the largest gas planet.”} \)
- \( d_3 = \text{“Saturn is the Roman god of sowing.”} \)

Documents as TF-IDF vectors

<table>
<thead>
<tr>
<th></th>
<th>Saturn</th>
<th>is</th>
<th>the</th>
<th>gas</th>
<th>planet</th>
<th>with</th>
<th>rings</th>
<th>Jupiter</th>
<th>largest</th>
<th>Roman</th>
<th>god</th>
<th>of</th>
<th>sowing</th>
</tr>
</thead>
<tbody>
<tr>
<td>( d_1 )</td>
<td>0.03</td>
<td>0</td>
<td>0</td>
<td>0.03</td>
<td>0.03</td>
<td>0.07</td>
<td>0.07</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( d_2 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.03</td>
<td>0.03</td>
<td>0</td>
<td>0</td>
<td>0.08</td>
<td>0.08</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( d_3 )</td>
<td>0.03</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.07</td>
<td>0.07</td>
<td>0</td>
<td>0.07</td>
</tr>
</tbody>
</table>

Cosine similarities between the documents

- \( \cos(d_1,d_2) = 0.13 \)
- \( \cos(d_1,d_3) = 0.05 \)
- \( \cos(d_2,d_3) = 0.00 \)
Embedding-based Similarity

1. Translate documents into embedding vectors
   - using for example doc2vec or
   - average embeddings of all words in the document

2. Calculate similarity of document embedding vectors
   - using dot product or cosine similarity
   - apply nearest neighbor search

   Use useful libraries

   - Gensim: offers doc2vec and word2vec implementations
   - Faiss: offers scalable, approximate nearest neighbor search methods
     (used for information retrieval)

http://bionlp-www.utu.fi/wv_demo/
5.2 Document Classification

- Given: A collection of labeled documents (training set)
- Find: A model for the class as a function of the values of the features
- Goal: Previously unseen documents should be assigned a class as accurately as possible

- Applications
  - topical classification of news stories or web pages
  - SPAM detection
  - sentiment analysis, hate speech detection

- Classification methods commonly used for text
  1. naive bayes
  2. support vector machines (SVMs)
  3. recurrent neural networks (RNNs), e.g. long short-term memory (LSTMs)
  4. Transformers like BERT
  5. but KNN or random forests may also work
Example Application: Sentiment Analysis

- Given: A text
- Goal: Assign a class of sentiment to the text
  - e.g., positive, (neutral,) negative
  - e.g., sad, happy, angry, surprised

- Can be implemented as supervised classification task
  - requires training data
  - i.e., pairs like <text; sentiment label>
Example Application: Sentiment Analysis

- Labeling data for sentiment analysis
  - is expensive, like all data labeling tasks

- Reviews from the Web may be used as labeled data (distant supervision)
  - Reviews from the Web may be used as labeled data (distant supervision)
    - Amazon Product Data by Julian McAuley: 142 million reviews from Amazon
    - WebDataCommons: 70 million reviews from 50,000 websites that use RDFa or Microdata markup

There exist various large corpora of reviews for public download
  - Amazon Product Data by Julian McAuley: 142 million reviews from Amazon
  - WebDataCommons: 70 million reviews from 50,000 websites that use RDFa or Microdata markup

---

Listen Closer

Trent Reznor should just release an album with a new title, new artwork, and new song titles. But instead of actual new material, it should all just be the songs from The Downward Spiral.

It can be called There You Go, ****heads.

After all, it’s what everyone wants.

I remember the day I bought The Downward Spiral. My first thought after...

Read the full review »
Published 1 month ago by Philip Atherton

---

Good, But Not Their Best

It’s funny how immediately after an established band that’s been around for a while comes out with a new album all the fan-boys give reviews saying it’s the greatest thing ever. I am a Nine Inch Nails fan too and have all their albums, so I’d thought I’d give my review which I hope is a little more fair.

It’s an electronic based album with some guitar, bass,...

Read the full review »
Published 1 month ago by JKt
Preprocessing for Sentiment Analysis

– Recap – we started our processing with: Simple Syntactic Analysis
  • text cleanup (remove punctuation, HTML tags, …)
  • normalize case
  • …

– However, reasonable features for sentiment analysis might include
  • punctuation “!” “?”, “?!”
  • smileys encoded using punctuation: e.g. ;-) :-(
  • use of visual markup, where available (red color, bold face, …)
  • amount of capitalization (“SCREAMING”)

– Practical Approach
  • Replace smileys or visual markup with sentiment words in preprocessing
  • 😊 great, COOL 👍 cool cool
Sentiment Analysis Tricks

- Finding selective words
  - weight words according to their correlation with label
  - select top-k words with highest correlation

- Sentiment lexicons
  - use external dictionary of opinion words
    - Bing Liu’s lexicon
      http://www.cs.uic.edu/~liub/FBS/opinion-lexicon-English.rar
    - AFINN lexicon
      https://github.com/fnielsen/afinn
  - restrict RapidMiner word list to these words or combine them with your learned sentiment words
5.3 Pre-Trained Language Models

- introduce pre-training, fine-tuning paradigm
  - pre-trained on large text corpora
  - fine-tune model for specific downstream tasks
  - BERT model size: 60 to 345 million parameters

- outperform previous models on most NLP tasks
Example: BERT for Entity Matching

- DITTO applies BERT, DistilBERT and RoBERTa for determining if two records describe the same real-word entity.
  - Classes: Match / Non-Match

- Entity serialization for BERT
  - Pair of entity descriptions (records) are turned into a single sequence
  - [CLS] Entity Description 1 [SEP] Entity Description 2 [SEP]
  - Entity Description = [COL] attr\_1 [VAL] val\_1 . . . [COL] attr\_k [VAL] val\_k
  - Serialization Example =

DITTO: Architecture

- [CLS] token summarizes the pair of entities
- linear layer on top of [CLS] token for matching decision
### DITTO: Evaluation

<table>
<thead>
<tr>
<th>Type</th>
<th>Dataset</th>
<th>DITTO F1</th>
<th>DeepMatcher F1</th>
<th>Magellan F1</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Structured</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>iTunes-Amazon</td>
<td>97.0</td>
<td>88.5 -8.5</td>
<td>91.2 -5.8</td>
</tr>
<tr>
<td></td>
<td>DBLP-ACM</td>
<td>99.0</td>
<td>98.4 -0.6</td>
<td>98.4 -0.6</td>
</tr>
<tr>
<td></td>
<td>DBLP-Scholar</td>
<td>95.6</td>
<td>92.3 -3.3</td>
<td>94.7 -0.9</td>
</tr>
<tr>
<td></td>
<td>Walmart-Amazon</td>
<td>86.8</td>
<td>66.9 -19.9</td>
<td>71.9 -14.9</td>
</tr>
<tr>
<td><strong>Textual</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Abt-Buy</td>
<td>89.3</td>
<td>62.8 -26.5</td>
<td>43.6 -45.7</td>
</tr>
<tr>
<td></td>
<td>Amazon-Google</td>
<td>75.6</td>
<td>69.3 -6.3</td>
<td>49.1 -26.5</td>
</tr>
<tr>
<td></td>
<td>WDC Computer - Large</td>
<td>91.7</td>
<td>89.5 -3.2</td>
<td>64.5 -27.2</td>
</tr>
<tr>
<td></td>
<td>WDC Computer - Small</td>
<td>80.8</td>
<td>70.5 -10.3</td>
<td>57.6 -23.2</td>
</tr>
</tbody>
</table>

- constant improvement for structured data
- large performance gain for textual data
Potential Reasons for the Performance Gain

- Serialization allows to pay attention to all attributes
  - no strict separation between attributes

- WordPiece tokenizer breaks unknown terms into pieces
  - no problems with out of vocabulary terms

- Transfer learning from pre-training texts
  - different surface forms are already close in embedding space

- Contextualization of the embeddings
  - potentially more suited for capturing differing semantics

- using Transformers for your projects
  - Hugging Face Library: code for fine-tuning Transformers
  - Hugging Face Model Hub: download pre-trained models
Summary

- Main challenge in text mining: Preprocessing and vectorization
- There are lots of alternative techniques
  - Thus, you need to experiment in order to find out which work well for your use case
  - focus has shifted from bag-of-words approaches to embeddings and Transformers for end-to-end learning
- Currently lots of ground-breaking innovation triggered by generative transformer models

- Courses on Text Mining
  - IE 661: Text Analytics (Strohmaier)
  - IE 696: Advanced Methods in Text Analytics (Ponzetto)
  - IE663: Information Retrieval and Web Search (Ponzetto)
References

- **Excellent Online Course**
  - Stanford CS224N: Natural Language Processing with Deep Learning
    - videos: https://www.youtube.com/playlist?list=PLoROMvodv4rOSH4v6133s9LFPRHjEmbmJ
    - slides: https://web.stanford.edu/class/cs224n/

- **Textbook**
    - Draft version of 3rd edition available online
      https://web.stanford.edu/~jurafsky/slp3/

- **Useful tools**
  - Stanford CoreNLP: collection of neural as well as traditional NLP tools
  - Gensim: Python implementation of word2vec
  - HuggingFace: Transformer library and model hub