Web Data Mining

Lecture 5: Text Mining 2

Jaroslav Kuchař & Milan Dojčinovski
jaroslav.kuchar@fit.cvut.cz, milan.dojchinovski@fit.cvut.cz

Czech Technical University in Prague - Faculty of Information Technologies - Software and Web Engineering

Summer semester 2019/2020
Humla v0.3

Overview

- Opinion Mining
- Text Summarization
- Text Models
- Word Embeddings

Opinion Mining

- Main goals
  - *Studying opinions and its related concepts such as sentiments, evaluations, attitudes, and emotions*
  - *Identify the opinion expressed in a text*
    $\rightarrow$ *usually* positive, negative *or* neutral *opinion*
- General application
  - *When we need to make a decision we often seek out the opinions of others.*
- Example:
  - *positive - "Americans trust @realDonaldTrump to Make our Economy Great Again!"*
  - *negative - "Racial discord was conceived,nurtured,refined & perpetuated by Americans incl @realDonaldTrump's father. Get real!"*
- Practical applications:
  - *reviews of films, books, electronics, but also...*
  - *companies want to know people's opinion*
  - *find political opinion on social networks*
  - *investigate how people's opinion influences stock market*
  - *...*

Opinion Mining Sub-tasks

- Opinion Extraction
  - *extract text which express opinion*
    $\rightarrow$ *I just bought a camera.* ***It is cheap, but the quality is not good****.*
- Sentiment Analysis
  - *identify polarity of an opinion*
    $\rightarrow$ *positive:* ***cheap***
    $\rightarrow$ *negative:* ***quality is not good***
- Opinion Summarization
  - *identify the overall opinion*
    $\rightarrow$ *price: positive, quality: negative,* ***overall: 60%***
- Note: we will primarily focus on the sentiment analysis task

Sentiment Analysis types

- Document and sentence subjectivity and sentiment classification
  - *Polarity classification*
    → identify the polarity of a given text (positive, negative, neutral)
- Feature/aspect sentiment identification
  - *determining the opinions or sentiments expressed on different features or aspects of entities*
  - *e.g. screen of a phone, camera sensor, actor of a movie*
- Opinion Search and Retrieval
  - *Find public opinions about a particular entity or an aspect of the entity*
  - *Find opinions of a person or organization about a particular entity or an aspect of the entity*
- Opinion Spam Detection
  - *detecting fake opinions (e.g. reviews) in order to be trusted*

Example

```python
import nltk
from nltk.sentiment.util import *
from nltk.sentiment import SentimentIntensityAnalyzer

text = None
with open('speech.txt', 'r') as f:
    text = f.read()

vader_analyzer = SentimentIntensityAnalyzer()
print(vader_analyzer.polarity_scores(text))
```

```
{'compound': 1.0, 'pos': 0.2, 'neg': 0.094, 'neu': 0.706}
```

```
{
 'negs': 0.476,
 'negt': "Jamiel's 17-year-old son was viciously murdered by an illegal immigrant
 gang member, who had just been released from prison.",
 'neus': 1.0,
 'neut': 'That torch is now in our hands.',
 'poss': 0.751,
 'post': 'Thank you, God bless you, and God Bless these United States.'
}
```

Lexicon-based Sentiment

- Lexicon
  – *Lists with opinion words used in sentiment classification tasks*
- Example:
  – *Positive sentiment words*
    → good, wonderful, amazing, ...
  – *Negative sentiment words*
    → bad, poor, terrible, ...
- Issues:
  – *Opposite orientations*
    → "This camera sucks!"
    → "This vacuum cleaner really sucks."
  – *No sentiment in a sentence*
    → "Which camera is good?"
    → "If I can find a good camera, I will buy it!"
  – *Sarcasm*
    → "What a great car! It stopped working in two days!"
  – *No sentiment words*
    → "This washer uses a lot of water."

Supervised Sentiment Learning

- Learning sentiment based on supervised learning
  – *two/three learning classes: positive, negative, and neutral*
- The learning approach
  – *train on annotated data*
    → *e.g. product reviews with ratings*
    → *normalize ratings*
      → 4-5 *stars as positive*
      → 1-2 *stars as negative*
      → 3 *for neutral*
- Learning similar to classical document topic classification
  – *topic related words are not important*
  – *sentiment words play an important role*
    → *e.g. great, excellent, bad, worse, amazing*
- Features
  – *Terms and their frequency, bag of words, n-grams, tf-idf*
  – *Part-of-speech tags - adjectives, ...*
  – *Lexicons with sentiment words*
  – *Sentiment shifters, rules, ...*
  – *Embeddings*

Unsupervised Sentiment Learning

- Sentiment classification based on fixed syntactic phrases
  - *phrase that are likely used to express opinion*
  - *phrase consisting of two words*

First word	Second word	Third word (not extracted)
JJ	NN or NNS	anything
RB, RBR, or RBS	JJ	not NN nor NNS
JJ	JJ	not NN nor NNS
NN or NNS	JJ	not NN nor NNS
RB, RBR or RBS	VB, VBD, VBN, or VBG	anything

- "This camera produces beautiful pictures."

Sentiment orientation

- Patterns based on POS tags are able to work with contexts.
  - *similarly to n-grams, phrases, relations, …*
- An isolated adjective may indicate opinion
  - good, best, not good
- BUT, there are adjectives with variable orientation
  - "unpredictable"
    - → *negative in auto industry*
      - → *"unpredictable steering"*
    - → *positive for movies*
      - → *"unpredictable plot"*

Sentiment orientation of phrases

- Estimation of semantic orientation of phrases
  – *using **pointwise mutual information (PMI)***
    $\rightarrow$ *indicates the degree of statistical dependence between them*

$$PMI(term_1, term_2) = \log_2\left(\frac{Pr(term_1 \wedge term_2)}{Pr(term_1)Pr(term_2)}\right)$$

- $Pr(term_1 \wedge term_2)$
  – *co-occurrence probability of* $term_1$ *and* $term_2$
- $Pr(term_1)Pr(term_2)$
  – *probability that the two terms co-occur if they are statistically independent*

Sentiment orientation of phrases (cont.)

- The sentiment orientation (SO) of a phrase is then computed based on reference words
  – *"excellent" for positive*
  – *"poor" for negative*

$$SO = PMI(phrase, "excellent") - PMI(phrase, "poor$$

- The probabilities can be computed in different way
  – *e.g. by evaluating number of hits for issued query to a search engine*
  – *hits - the number of matching documents*
  – *NEAR operator constrains the search to documents that contain the words within ten words of one another*

$$SO = \log_2\left(\frac{hits(phrase \quad NEAR \quad "excellent")hits("poor")}{hits(phrase \quad NEAR \quad "poor")hits("excellent")}\right)$$

- Finally, a sentiment of a text is computed by averaging the sentiment of all extracted phrases.

Example

```python
import math
from nltk.corpus import stopwords
import nltk
from collections import Counter
import operator
stops = stopwords.words('english')

text = None
with open('speech.txt', 'r') as f:
    text = f.read()

# all tokens
tokens = [t.lower() for t in nltk.word_tokenize(text) if t.lower() not in stops]
sentences = nltk.sent_tokenize(text)
tuples = []
# all tuples
for sentence in sentences:
    for word in set(nltk.word_tokenize(sentence)):
        if word.lower() not in stops:
            for word2 in set(nltk.word_tokenize(sentence)):
                if word2.lower() not in stops:
                    tuples.append((word,word2))
# frequencies
fdistTuples = nltk.FreqDist(tuples)
fdistTokens = nltk.FreqDist(tokens)
# PMI
def PMI(t1,t2):
    pp = fdistTuples.freq((t1,t2)) if fdistTuples.freq((t1,t2))>0 \
        else fdistTuples.freq((t2,t1))
    den = (fdistTokens.freq(t1)*fdistTokens.freq(t2))
    return (math.log2( pp/den)) if den>0 and pp>0 else 0
```

Example (cont.)

```python
posVocab = [
    'good', 'nice', 'great', 'awesome', 'outstanding', 'fantastic', 'terrific', 'like', 'love'
]
negVocab = [
    'bad', 'terrible', 'crap', 'useless', 'hate', 'kill', 'murder', 'not', 'poor'
]

results = {}
for token in set(tokens):
    positives = sum(PMI(token,v) for v in posVocab)
    negatives = sum(PMI(token,v) for v in negVocab)
    results[token] = positives - negatives
results = sorted(results.items(),
                         key=operator.itemgetter(1),
                         reverse=True)
print(results[:20])
print(results[-20:])
```

```
[('deserves', 11.057463465454001), ('requires', 11.057463465454001),
('behalf', 11.057463465454001), ('ground', 11.057463465454001),
('cooperate', 11.057463465454001), ('advance', 11.057463465454001),
('brighter', 11.057463465454001), ('saved', 10.209466558899052),
('child', 9.207683574927273), ('common', 9.057463465454001),
('father', 8.887538464011689), ('taxpayers', 8.830954935645321), ('love', 8.73553537056664), (

[('condemning', -7.850659827614363), ('epidemic', -7.850659827614363),
('targeting', -7.850659827614363), ('well', -7.850659827614363),
('vandalism', -7.850659827614363), ('cemeteries', -7.850659827614363),
('week', -7.850659827614363), ('policies', -7.850659827614363),
('may', -7.850659827614363), ('ultimately', -7.850659827614363), ('hate', -7.850659827614363),
```

Lexicon Expansion

- How to bootstrap opinion lexicon
  – *lists with opinion words used in sentiment classification tasks*
- Manual approach
  – *manually populating the opinion lexicon*
  – *usually combined with an automated approach*
- Dictionary based approach
  – *provide initial seed of opinion words*
  – *extend the initial set of words by use of lexicon*
  → *e.g.* WordNet
  – *look for synonyms and antonyms*
  – *iterative approach*
  → *recursively search for new words*
  → *terminate when there are no new words*

Lexicon Expansion - WordNet

```python
from nltk.corpus import wordnet
token = "good"
synonyms = []
antonyms = []
syns = wordnet.synsets(token)
for syn in syns:
    for l in syn.lemmas():
        synonyms.append(l.name())
        if l.antonyms():
            antonyms.append(l.antonyms()[0].name())
print(set(synonyms))
print(set(antonyms))
```

```
{'unspoiled', 'trade_good', 'dependable', 'soundly',
'thoroughly', 'safe', 'full', 'upright', 'just',
'respectable', 'skillful', 'good', 'beneficial',
'commodity', 'proficient', 'adept', 'estimable',
'ripe', 'near', 'in_effect', 'in_force', 'sound',
 'serious', 'effective', 'expert', 'salutary',
 'unspoilt', 'secure', 'dear', 'honest',
 'goodness', 'undecomposed', 'skilful', 'right',
 'well', 'practiced', 'honorable'}

{'evilness', 'badness', 'evil', 'bad', 'ill'}
```

Lexicon Expansion (cont.)

- Corpus based approach
  - *based on syntactic or co-occurrence patterns*
  - *uses a seed list of opinion words to find other opinion words in a large corpus.*
- The method:
  - *start with seed list of opinion adjectives (e.g.* beautiful*)*
  - *use them in combination with linguistic constraints to identify new opinion words*
    → *E.g. linguistic constraints* AND
  - *assumption conjoined adjectives usually have the same orientation*
    → "This car is beautiful and spacious"
  - "beautiful" *in seed, by using the AND constraint we can identify also* "spacious"

Lexicon Expansion - Linguistic constraints

```
import nltk

text = None
with open('speech.txt', 'r') as f:
    text = f.read()

text_pos = nltk.pos_tag(nltk.word_tokenize(text))
grammar = """LC: {<JJ><CC><JJ>}
{<NN|NNS><CC><NN|NNS>}"""
cp = nltk.RegexpParser(grammar)
result = cp.parse(text_pos)
print(result)
```

```
...
(LC non-military/JJ and/CC non-essential/JJ)
...
(LC real/JJ and/CC positive/JJ)
...
(LC drugs/NNS and/CC crime/NN)
...
(LC dealers/NNS and/CC criminals/NNS)
...
```

Domain-specific Expansions

- Acronyms

Acronym	English
grt8,grt8t	great
lol	laughing out loud
bff	best friend forever
…	…

- Emoticons

Emoticon	polarity
:-) :) :o) :] :3 :c)	positive
:D C:	extremely positive
:-(:(:c :[	negative
D8 D; D= DX v.v	extremely negative
: |	neutral

Aspect-Based Opinion Mining

- Opinions expressed on certain entity aspect
  - *"The picture quality of this camera is not great , but the battery life is long."*
    → the aspects are "picture quality" *and* "battery"
- Steps:
  - ***mark opinion words and phrases***
    → The picture quality of this camera is not great [+1]*, but the battery life is long."*
    → +1 *because of* "great"
  - ***handle opinion shifters***
    *→ words that change opinion orientation (*not, never, none*)*
    → The picture quality of this camera is not great [-1]*, but the battery life is long."*
    → from +1 *to* -1 *because of the* "not" *shifter*

Aspect-Based Opinion Mining (cont.)

- Steps (cont.):
  - ***Handle but-clauses***
    - → *in English*
      - → but *means contrary*
    - → *the orientation of the opinions before and after* BUT *are opposite*
    - → *"The picture quality of this camera is not great* [-1], *but the battery life is long* [+1]*"*
    - → *we know the orientation of the phrase before BUT (negative)*
    - → *we can derive the orientation of the phrase after BUT (positive)*

Comparative Opinion Mining

- Opinions can be expressed in comparative manner
  - *"The picture quality of Camera-x is better than that of Camera-y."*
- Comparative opinions are expressed with comparative or superlative form of an adjective or adverb.
- Comparative expression
  - *"The picture quality of Camera-x is better than that of Camera-y."*
- Superlative expression
  - *"The picture quality of Camera-x is the best among the new cameras in 2017."*

Comparative Opinion Mining Example

```python
import nltk

text = None
with open('speech.txt', 'r') as f:
    text = f.read()

text_pos = nltk.pos_tag(nltk.word_tokenize(text))
grammar = """CP: {<DT>*<JJR|JJS><NN|NNS>}
{<RBR|RBS>}
"""
cp = nltk.RegexpParser(grammar)
result = cp.parse(text_pos)
print(result)
```

```
(CP more/JJR dollars/NNS)
...
(CP the/DT highest/JJS rates/NNS)
...
(CP poorest/JJS workers/NNS)
...
```

Sarcasm

- The speakers or the writers say or write the opposite of what they mean.
  – *Common in discussions e.g. political*
- Examples:
  – *"I'm so pleased mom woke me up with vacuuming my room this morning."*
  – *"I had never seen snow in Holland before but thanks to twitter and facebook I know what it looks like. Thanks guys, amazing!"*
- Tips on sarcasm identification:
  – *use of hashtag #sarcasm*
  – *collect tweets with #sarcasm and use them to train machine learning model*
  $\rightarrow$ *"To the hospital #fun #sarcasm"*

Sarcasm (cont.)

- BUT still, how you know which part of the sentence is sarcastic
  - *"There's no better start into the working week than a construction site right beneath your office. Sounds a bit like Neubauten."*
- To identify sarcasm you can also:
  - *incorporate background knowledge*
    - → *e.g. it is generally known that people don't like going to dentist*
    - → *"Going to the dentist on my weekend home. Great. I'm totally pumped. #sarcasm"*

Overview

- Opinion Mining
- Text Summarization
- Text Models
- Word Embeddings

Text Summarization

- Goal
  - *Reducing a text with a computer program in order to create summary that retains the most important points of the original text.*
- Main applications
  - *generating abstracts*
  - *articles, web pages summaries*
- Main categories
  - *Single document vs Multi document*
  - *Extractive vs Abstractive*
  - *Generic vs Query-driven*
- Stages
  - *Content selection*
    → select sentences to extract
  - *Information ordering*
    → choose order of information
  - *Sentence realization*
    → clean up sentences

Frequency-based Summarization

```python
import nltk
from nltk.corpus import stopwords
from string import punctuation
stops = stopwords.words('english') + list(punctuation) + ["--"]

text = None
with open('speech.txt', 'r') as f:
    text = f.read()

sentences = [[t for t in nltk.word_tokenize(sentence)] \
                for sentence in nltk.sent_tokenize(text)]

results = []
for sentence in sentences:
    if len(sentence)>0:
        ntokens = len(sentence)
        tagged=nltk.pos_tag(sentence)
        nnouns=len([word for word,pos in tagged if pos.startswith("NN")])
        ners=nltk.ne_chunk(tagged,binary=False)
        nners= len(extractEntities(ners))
        score=(nnouns+nners)/float(ntokens)
        results.append((ntokens,nnouns,nners,score,sentence))
for item in sorted(results,key=lambda x: x[3],reverse=True)[:5]:
    print(" ".join(item[4]))
```

```
Their names are Jamiel Shaw , Susan Oliver , Jenna Oliver , and Jessica Davis .
Thank you , God bless you , and God Bless these United States .
Alexander Graham Bell displayed his telephone for the first time .
America is friends today with former enemies .
The office is called VOICE -- - Victims Of Immigration Crime Engagement .
```

Baseline Single Document Summarization

- Abstract creation
  - *Introduced in 1958 by H.P. Luhn*
    - *→ H. P. Luhn. 1958. The automatic creation of literature abstracts. IBM J. Res. Dev. 2, 2 (April 1958), 159-165.*
- Based on filtering out sentences containing frequently occurring words that appear near one another.
- Algorithm:
  1. *Detect most important words in text and use only top N (*WORDS*)*
     - *remove stopwords, punctuations, convert to lemmas/stems*
  2. *For each sentence find clusters of important words that are not far away from each other (max. distance* DISTANCE*)*
  3. *Compute score for each cluster*
     - $score = \frac{|significantWordsInCluster| \times |significantWordsInCluster|}{|clusterWords|}$
     - *Score of the sentence is the max score of its clusters*
  4. *Return top sentences (*SENTENCES*)*

Baseline Single Document Summarization (cont.)

```python
import nltk
from nltk.corpus import stopwords
from string import punctuation
from nltk.stem.porter import PorterStemmer
stops = stopwords.words('english') + list(punctuation) + ["--"]
stemmer = PorterStemmer()

text = None
with open('speech.txt', 'r') as f:
    text = f.read()

# original sentences
sentences = nltk.sent_tokenize(text)

# sentences with converted tokens
processedSentences = [ \
    [stemmer.stem(t.lower()) \
    for t in nltk.word_tokenize(sentence) if t.lower() not in stops] \
    for sentence in nltk.sent_tokenize(text) \
    ]

# constants
WORDS = 200
DISTANCE = 3
SENTENCES = 5

# top most important words
fdist = nltk.FreqDist([w for sentence in processedSentences for w in sentence])
importantWords = [w[0] for w in fdist.most_common()[:WORDS]]
```

Baseline Single Document Summarization (cont.)

```python
results = []
for si,s in enumerate(processedSentences):
    clusters = []
    cluster = []
    for wi,w in enumerate(s):
        # remember only index of the important word in the sentence
        if w in importantWords:
            if len(cluster) == 0:
                cluster.append(wi) # empty cluster
            elif (wi-cluster[-1])<DISTANCE:
                cluster.append(wi)  # if is not far away
            # end of cluster, add to the list and init a new one
            else:
                clusters.append(cluster)
                cluster = [wi]
    clusters.append(cluster)
    # irrelevant sentences with no important words
    if len(cluster)==0:
        continue
    # compute score for each cluster
    scores = [len(cluster)*len(cluster)/(cluster[-1]-cluster[0]+1) \
                                            for cluster in clusters]
    # remember score and original sentence
    results.append((max(scores),sentences[si]))
for item in sorted(results,key=lambda x: x[0],reverse=True)[:SENTENCES]:
    print(item[1])
```

```
Finally, to keep America Safe we must provide the men and women of the United States military
According to the National Academy of Sciences, our current immigration system costs America's
On this and so many other things, Democrats and Republicans should get together and unite for
To any in Congress who do not believe we should enforce our laws, I would ask you this questio
By finally enforcing our immigration laws, we will raise wages, help the unemployed, save bill
```

Overview

- Opinion Mining
- Text Summarization
- Text Models
- Word Embeddings

Models of Information Retrieval

- IR models
  - *specify how a document and a query are represented and how the relevance of a document to a user query is defined.*
- Main models
  - *Boolean model, Vector space model, language model, probabilistic model*
- Common characteristics
  - *Bag of Words (BoW)*
    $\rightarrow$ *A document or query is represented as* bag of words *or* bag of terms
  - *Vocabulary*
    $\rightarrow$ *Set of distinct words/terms for the collection of documents:*
    $V = \{t_1, t_2, \ldots, t_{|V|}\}$
  - *Term vector*
    $\rightarrow$ *Represents each document as a vector of weights* $\mathbf{w}_{ij}$
    $\rightarrow d_j = (w_{1j}, w_{2j}, \ldots, w_{|V|j})$

Boolean Model

- The simplest IR baseline model
  - *Based on Boolean algebra*
- Representations
  - *Documents*
    $\rightarrow$ *Considered only presence or absence of the term*
    $\rightarrow w_{ij} \in \{0, 1\}$
  - *Queries*
    $\rightarrow$ *Precise semantic using Boolean operators such as* AND, OR, NOT
  - *Retrieval*
    $\rightarrow$ *Only* exact match *is considered - document is either relevant or irrelevant*
    $\rightarrow$ *Can lead to poor results*
- Example
  - $V = \{information, mining, text\}$
  - $d_1 = (1, 0, 1)$, $d_2 = (0, 0, 1)$, $d_3 = (1, 1, 0)$
  - *query*
    $\rightarrow$ *mining* AND *text* $\rightarrow \emptyset$
    $\rightarrow$ *mining* OR *text* $\rightarrow \{d_1, d_2, d_3\}$

Vector Space Model

- Well known and widely used IR model
  - *Uses any number to represent the weight in the term vector*
- TF-IDF weighting schema
  - *TF - Term Frequency*
    - → *How many times the term occurs in a document + normalizations*
    - → *(Number of times term appears in a document) / (Total number of terms in the document)*
  - *IDF - Inverse Document Frequency*
    - → *Measures how common a word is among all documents*
    - → *log(Total number of documents / Number of documents with the term)*
  - *TF-IDF* $= \boldsymbol{TF} \times \boldsymbol{IDF}$
- Example
  - *Word is mentioned 5 times in a document of 100 unique words*
    - → $\boldsymbol{TF} = 5/100$
  - *We have 10 000 documents and the word is in 100 of them*
    - → *IDF = log(10 000/100) = 2*
  - *TF-IDF = 0.05 * 2 = 0.1*

Simple Scoring - Jaccard Coefficient

- Measures overlap of two sets
  - $jaccard(\text{doc-A}, \text{doc-B}) = \frac{|\text{doc-A} \cap \text{doc-B}|}{|\text{doc-A} \cup \text{doc-B}|}$
    - → *how many terms doc-A and doc-B have in common*
- Assigns a number between 0 and 1
- Issues:
  - *doesn't consider term frequency*
  - *rare are more informative then frequent terms*
- Example:
  - *doc-A* $= \{information, mining, text\}$
  - *doc-B* $= \{information, data, knowledge\}$
  - $jaccard(\text{doc-A}, \text{doc-B}) = \frac{1}{5}$

Euclidean Based Proximity Measure

- Represent query and document in a vector space model
  - *represent a query as a vector*
  - *represent a document as a vector*
- Compute the distance of the two vectors in a multi-dimensional space
- An Eucledian distance of two vectors

$$\mathrm{d}(\mathbf{p},\mathbf{q}) = \mathrm{d}(\mathbf{q},\mathbf{p}) = \sqrt{(q_1-p_1)^2 + (q_2-p_2)^2 + \cdots + (q_n-p_n)^2} = \sqrt{\sum_{i=1}^{n}(q_i-p_i)^2}$$

- Eucledian distance is a bad idea
  - *if a query term occurs more times in the document, the distance will be larger*
  - *distance is large, altough the distribution of terms are similar*

Cosine Based Proximity Measure

- Use angle instead of distance
  - *compute cosine of the angle between the query and document vectors*
- Cosine similarity of two vectors (query vs. document)

$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n}(A_i)^2} \times \sqrt{\sum_{i=1}^{n}(B_i)^2}}$$

- Long and short vectors now have comparable weights
- If two documents are identical, then the angle is 0
  - *cosine of 0 angle = 1*
  - *similarity of two identical vectors is 1*
- Rank the documents according to the angle of the query

Summary

- Represent the query as a vector
- Represent each document candidate as a vector
- Compute the cosine of the angle between each query-document pair
  - computing similarity with every document is time intensive!
  - ***solution: do not compute similarity with each document***
  - *compute similarity only with:*
    → documents containing many query terms
    → documents containing rear query terms
- Rank the documents according to their similarity with the query
- Return the top-K ranked documents

Overview

- Opinion Mining
- Text Summarization
- Text Models
- Word Embeddings

Word Embeddings

- Word Embeddings
  - *Based on distributional hypothesis*
    → words that occur in similar contexts tend to have similar meanings
  - *Embeddings*
    → learning representations of the meaning of words directly from their distributions in texts
    → (contextualized word representations - e.g. ELMo, BERT, ...)
  - *Language modeling technique used for mapping words to vectors of real numbers*
  - *Allows words with similar meaning to have a similar representation (a dense distributed representation for each word)*
  - *Can be generated using various methods:*
    → neural networks, co-occurrence matrix, probabilistic models, etc.

Word Embeddings (cont.)

- Issues:
  - *We can produce an infinite number of sentences, words will have different meanings based on the context used.*
  - *Tens or hundreds of dimensions. This is contrasted to the thousands or millions of dimensions required for sparse word representations, such as a one-hot encoding.*

```
# one hot encoding
# can't find the similarities between different words in this way
car     = [0 0 0 0 1 0 0 0 0 0 0 0 0 0 0]
vehicle = [0 0 0 0 0 0 0 0 1 0 0 0 0 0 0]
```

  - *Possible solution can be using synonyms from WordNet*

Classic Text Representations

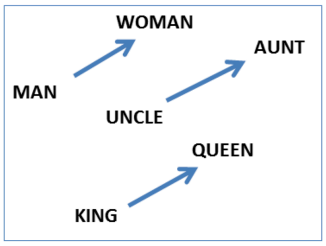
- Bag-of-words (one-hot encoding)
  - *the vocabulary is represented as a set of* n *words*
  - *the representation supress grammar and ordering information*

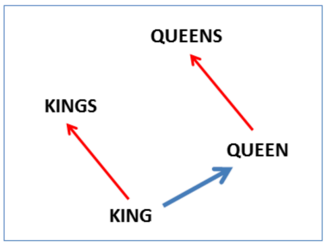
```
# one hot encoding
# can't find the similarities between different words in this way
car     = [0 0 0 0 1 0 0 0 0 0 0 0 0 0 0]
vehicle = [0 0 0 0 0 0 0 0 1 0 0 0 0 0 0]
```

- Bag-of-n-grams
  - *brings back the context and ordering information*
- TF-IDF weighting
- Latent Dirichlet allocation (LDA)
  - *Topic modelling - Each document can be described by a distribution of topics and each topic can be described by a distribution of words*

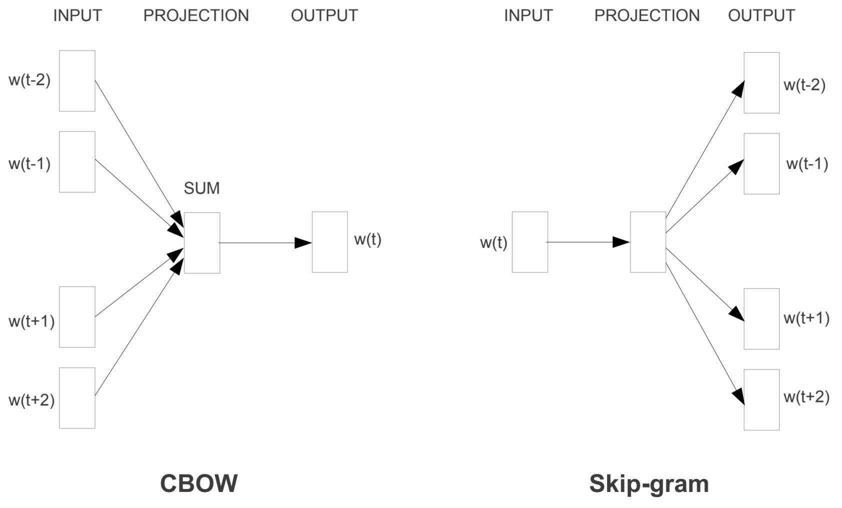
Word2Vec

- Word2Vec
  - *the most popular technique to learn word embeddings using shallow neural network*
  - *Tomas Mikolov, et al. at Google in 2013*
- Two main models
  - *Continuous Bag-of-Words (CBOW)*
    - *→ predicting the current word based on its context*
  - *Continuous Skip-Gram*
    - *→ predicting the surrounding words given a current word*
- Vector operations
  - *T. Mikolov et al. : Linguistic Regularities in Continuous Space Word Representations, NAACL 2013.*





Word2Vec models



- Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space.

Word2Vec example

```python
# imports
import gensim
from gensim.models import Word2Vec

# data
from nltk.corpus import brown
data = brown.sents()

# Create CBOW model
cbow_model = gensim.models.Word2Vec(data, min_count = 5, size = 100, window

# Create Skip Gram model
sg_model = gensim.models.Word2Vec(data, min_count = 5, size = 100, window =

# results
print(cbow_model.wv.most_similar("nice", topn=3))
# [('happy', 0.9289487600326538),
# ('fine', 0.9271352291107178),
# ('lonely', 0.9264320135116577)]

print(sg_model.wv.most_similar("nice", topn=3))
# [('awfully', 0.9151874780654907),
# ('sad', 0.9106264114379883),
# ('lonely', 0.9073173999786377)]
```