# EB5204 : NEW MEDIA AND SENTIMENT MINING

## MODULE 2.1:  TRAINING CORPUS & FEATURES

Dr. Wang Aobo

isswan@nus.edu.sg

# Module Objectives 2.1

- Identify and **evaluate** methodologies for training **data sets** used in sentiment mining
- Identify and evaluate training **features** for sentiment mining
- Construct auto-learned training features from **Word Vector Representation**

# Agenda

- Training **data** set generation for sentiment mining
- Training **features** for sentiment mining
- Word vectors

# 1. Training data for sentiment analysis

# Training data set I

- The key point is to use the training data as **similar** to the test set, which applies generally for all supervised training models.

- The training and test data sets should be used from the **same domain** as far as possible. It solves problems of domain-specific terms. In most cases, best to generate a training data set for your specific objective.

- In generating training set, go for high-precision and **low recall**.
  - *High precision* means be sure those you say are positive are indeed positive. those you say are negative are indeed negative. Normally happens if you set a 'high bar'.
  - *Low recall* means a lot of the actual positives or actual negatives are actually "ignored" as they cannot clear the 'high bar'.
- **Balance** your dataset

# Training data set II

- Ways to create training reference data:
  - dictionary corpus
  - user-generated means
  - manual (by inspection – tedious)

The training data set is usually ***not static*** but requires fine-tuning even after production.

This helps to account for changing fads in expressions, languages slangs etc as well.

# Dictionary corpus

- Using existing dictionary corpus, egs are:
  i. SentWordNet
  ii. Public sources eg. Liu Bing, https://www.w3.org/community/sentiment/wiki/Datasets
  iii. nltk corpus

  Then **expand** and **modify** the dictionary corpus.

  First, a revision over synsets (revision primer from Text Mining)

# Wordnet synsets

- WordNet® is a large lexical database of English. Nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms (synsets), each expressing a distinct concept. https://wordnet.princeton.edu/

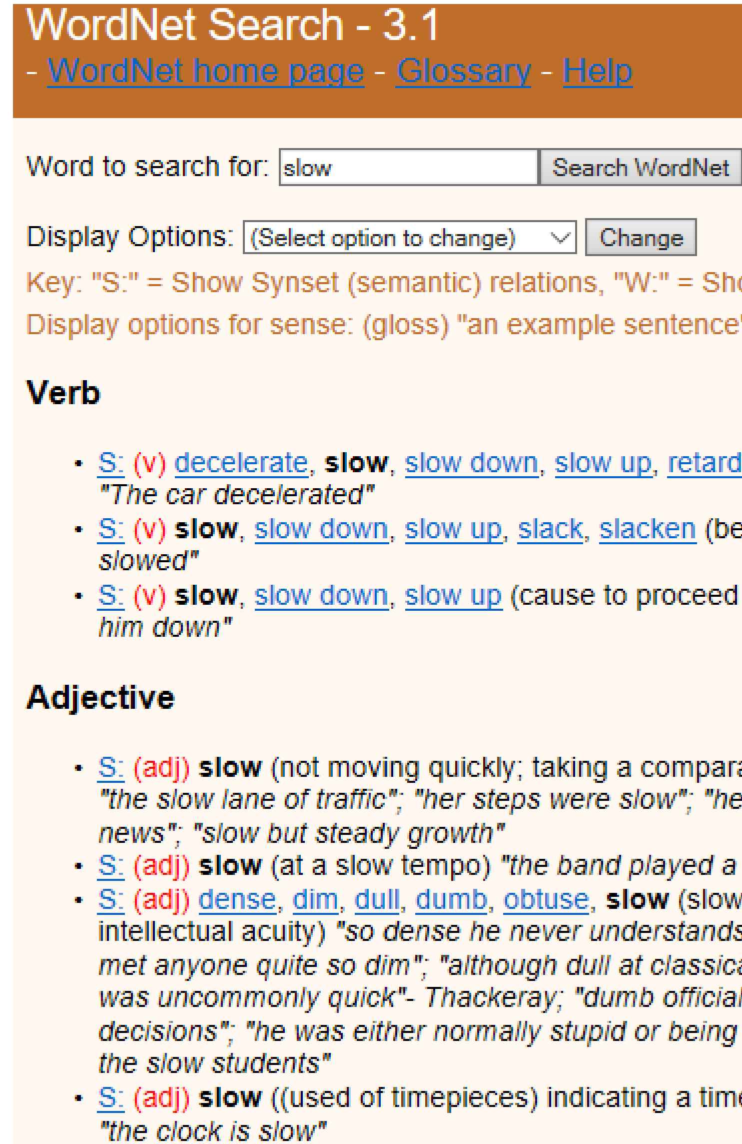
- The training corpus can be **expanded** using bootstrapping. through WordNet synsets, or related words.

- SentiWordnet adds on to WordNet by assigning sentiment polarity to these senses

More in the workshop today on ***WordNet*** and *SentiWordNet*…

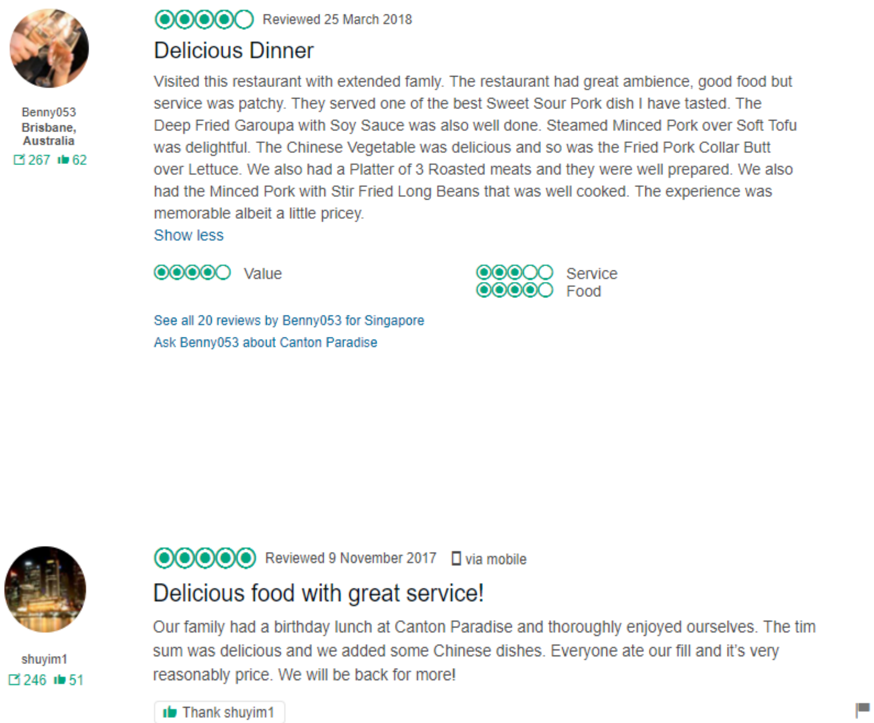
# Bootstrapping synsets

- The bootstrapping of wordnet synsets can be understood in 2 steps.
  - Use a seed set of positive and negative words with their sentiment. Iterate through one by one
  - Search for the seed word's synset of records. These words then takes on the original seedset's sentiment.



# User-generated ratings

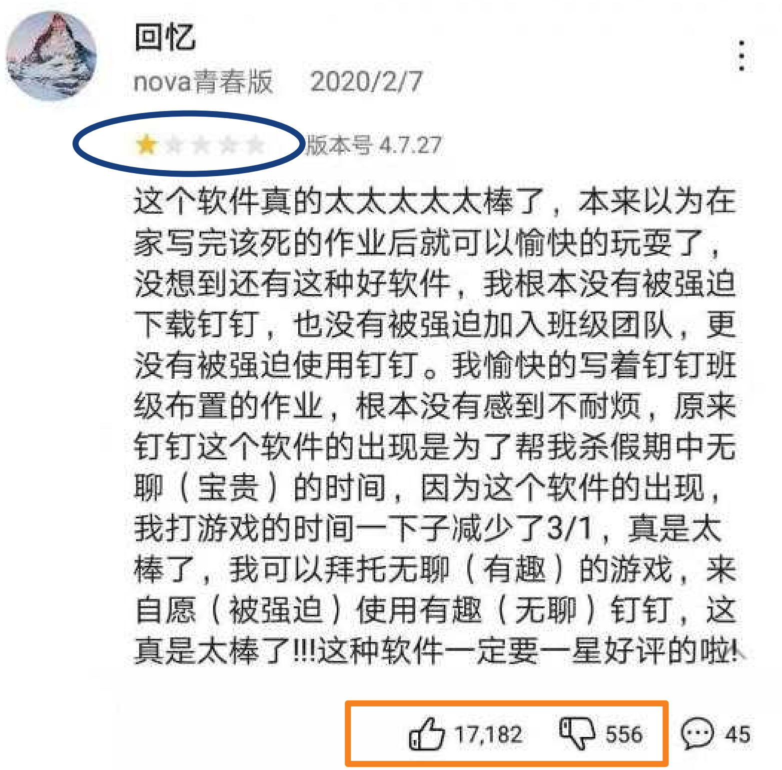
- Use the meta-data in social media to assign positive or negative ratings to the comment posts.

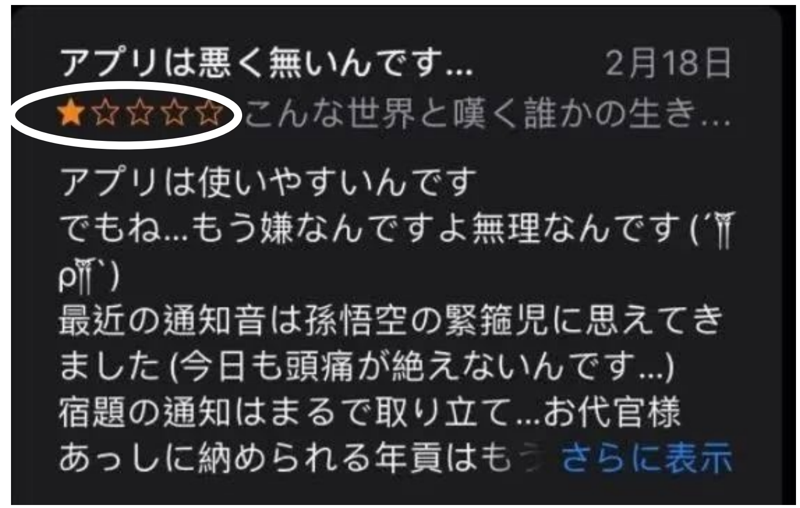



Simply use high ratings this as positive labels; and low ratings as negative labels

# User-generated ratings

- Mind the biased reviews.



# 2. Features for sentiment analysis

# Features used in sentiment mining

- From Wikipedia:
  - **Feature engineering** is the process of using domain knowledge of the data to create features that make machine learning algorithms work.





- Represent [sentences] with a vector of numbers, which can better/best distinguish the [polarity] among all the [sentences]

# Feature Engineering

- Some common features used in sentiment analysis are
  - Part of speech (POS) tags (adjectives or nouns)
  - Opinion lexicons and phrases (n-grams)
  - Negations
  - Syntactic dependency (more about this on Day 3)
  - Sentiment-aware tokens (recall 1st day)
  - Word vectors
  - Terms frequency and different information retrieval *weighting* schemes – tf-idf

What are other word features do you think will matter?

In an actual project, it is wise to look through some data sets in some detail, and identify what sets positive or negative polarity statements apart.

# Features Weighting

Some are covered in Text Mining.

- Binary
  - 0 or 1, simply indicating whether a word has occurred in the document.
- Frequency-based
  - ***term frequency***, the frequency of words in the document.
- *tf-idf* weighting: (considers document set)
  - $tf_{t,d}$ : term frequency – number of occurrences of term *t* in document *d*
  - $idf_t$ : inverted document frequency of term *t*
  - $df_t$ : the ***document frequency*** of term *t*, i.e., the number of documents that contain the term.
  - *N* : the total number of documents in the corpus

$$tf\text{-}idf_{t,d} = tf_{t,d} * idf_t$$

$$idf_t = \log \frac{N}{df_t}$$

# Frequency or presence?

- Pang et al 2002 found out that better performance of sentiment classification on movie review data is achieved by accounting only for feature presence, not feature frequency

| | Features | # of features | frequency or presence? | NB | ME | SVM |
|---|---|---|---|---|---|---|
| (1) | unigrams | 16165 | freq. | **78.7** | N/A | 72.8 |
| (2) | unigrams | ” | pres. | 81.0 | 80.4 | **82.9** |

# Feature Engineering

- **Pointwise Mutual Information**
  - Do the two words **co-occur** very often for a reason? or just by random

$$\mathrm{PMI}(w_1,w_2) = \log_2 \frac{P(w_1,w_2)}{P(w_1)P(w_2)}$$

$$P(w) = \frac{Freq(w)}{totalWordCount}$$

  - Positive PMI

$$PPMI = \begin{cases} PMI & if\ PMI>0 \\ 0 & else \end{cases}$$

# Feature Engineering

- **PMI for first 50 millions of words in Wikipedia**
  - Total word count is 50,000,952

| word 1 | word 2 | count word 1 | count word 2 | count of co-occurrences | **PMI** |
|---|---|---|---|---|---|
| puerto | rico | 1938 | 1311 | 1159 | **10.0349081703** |
| hong | kong | 2438 | 2694 | 2205 | **9.72831972408** |
| los | angeles | 3501 | 2808 | 2791 | **9.560676150** |
| to | and | 1025659 | 1375396 | 1286 | **-3.08825363041** |
| to | in | 1025659 | 1187652 | 1066 | **-3.12911348956** |
| of | and | 1761436 | 1375396 | 1190 | **-3.70663100173** |

# Term-document matrix

- Many text mining applications are based on vector representation of documents (term-document matrix) using "bag-of-words" approach

$$
\begin{array}{c}
 \\
D_1 \\
D_2 \\
\vdots \\
\vdots \\
D_n
\end{array}
\begin{pmatrix}
F_1 & F_2 & \cdots & F_t \\
w_{11} & w_{21} & \cdots & w_{t1} \\
w_{12} & w_{22} & \cdots & w_{t2} \\
\vdots & \vdots & & \vdots \\
\vdots & \vdots & & \vdots \\
w_{1n} & w_{2n} & \cdots & w_{tn}
\end{pmatrix}
$$

*F*: features
*D*: document
*w*: value of the features

- Usually only content words (adjectives, adverbs, nouns, and verbs) are used as unigram vector features.

# Classic NLP: Feature Engineering

- **Count-based vectors are**
  - e.g. TF-IDF, PPMI
  - long (|V| > 100,000)
  - sparse (lots of zero)

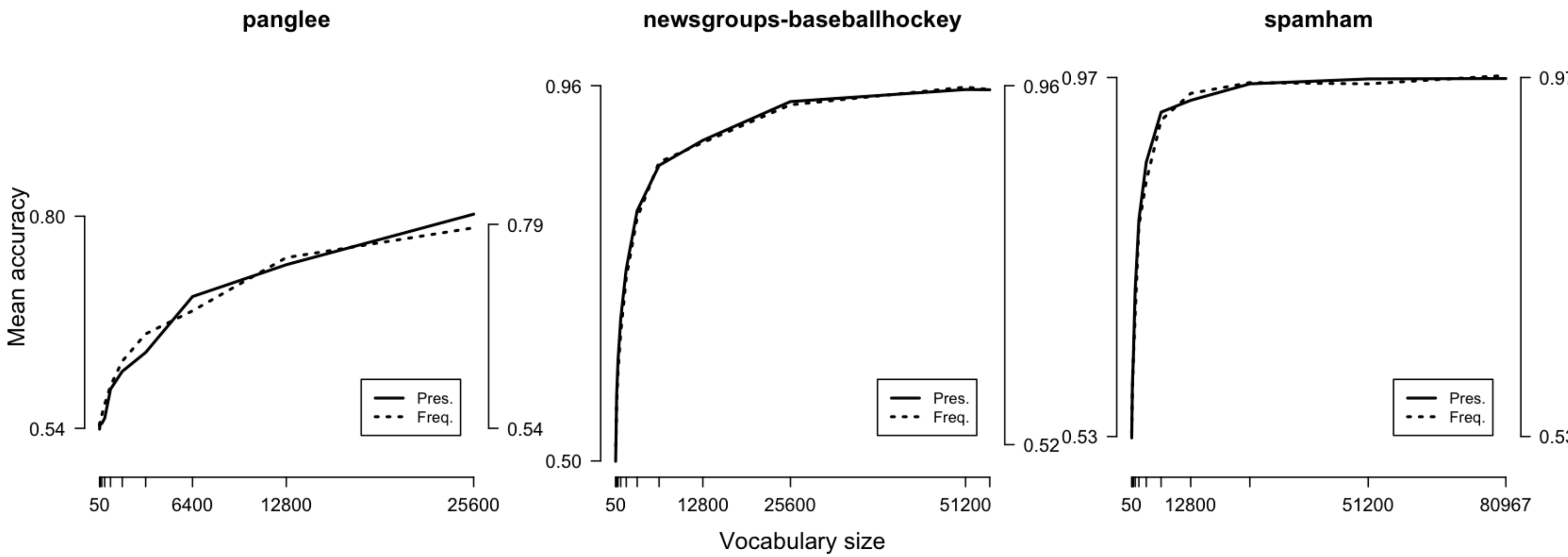
| | he | drink | hold | … | *tfidf* drink apple | *tfidf* hold apple | *tfidf* apple juice | … | *PPMI* drink apple | *PPMI* hold apple | *PPMI* apple juice |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **sent0** | 0.01 | 0.38 | 0.00 | … | 0.87 | 0.00 | 0.92 | … | 4.23 | 0.00 | 8.90 |
| **sent1** | 0.01 | 0.00 | 0.28 | … | 0.00 | 0.87 | 0.00 | … | 0.00 | 2.45 | 0.00 |

# The Curse of Dimensionality

- The feature-document matrix lies in **high-dimensional spaces**, (100,000+ features from variations of “Ngrams”).
- High-dimensional data requires an amount of time and memory that increases exponentially.
- Irrelevant “noise” features affect the performance of the algorithms – overfitting!
- Data sparsity – a lot of features with presence in very few documents.

# Feature Selection

- Overall sentiment classifier accuracy increases steadily with the no of features (e.g., size of the vocabulary), but risk over-fitting, not generalize well to new data.




Potts, 2011

# Feature Selection – more or less features?

- Pang et al 2002 found that simply using the 2633 most frequent unigrams can yield performance comparable to that of using all 16165.

| | Features | # of features | frequency or presence? | NB | ME | SVM |
|---|---|---|---|---|---|---|
| (1) | unigrams | 16165 | freq. | **78.7** | N/A | 72.8 |
| (2) | unigrams | " | pres. | 81.0 | 80.4 | **82.9** |
| (7) | top 2633 unigrams | 2633 | pres. | 80.3 | 81.0 | **81.4** |

# Feature Selection (or extraction)

- To select relevant features and reduce the number of features used in the matrix
- Various ways (trial-and-error):
  - Remove **features that appear rarely** in the documents
  - Select **top *K*** number of most **frequent** features
  - Leverage on the labels to pick ***K* most useful** features

*More about it in workshop.*

# Summary

- Two key steps before building a sentiment analysis are:
  i. *Training data (corpus)* selection/ generation
  ii. *Features selection*

These pre-steps are key to the success of a sentiment analysis and usually **more important than the training algorithms** themselves.

Training data selection needs to be as similar as possible to the production data. The features selection requires domain expertise.

# Features from WordToVec

- Feature generation and selection could be tedious
- How might we generate *"universal"* features automatically?
- Zooming into word level vector representation

# Features from Word Vectors

- **Count from Data**
  - Word Co-occurrence + SVD
  - Count-based model
- **Learn from Data**
  - CBOW and SKIPGRAM
  - NN Methods
  - Predictive Model
- **Count and Learn from Data**
  - GLOVE: Global Vectors for Word Representation
  - Count + SGD

# Count from Data

- **Word-level representation**
- **Counting context-words within a window_size**

Sent_1: *I like deep learning*
Sent_2: *I like NLP*
Sent_3: *I enjoy flying*

**Window_size=1**

| counts | I | like | enjoy | deep | learning | NLP | flying |
|---|---|---|---|---|---|---|---|
| I | 0 | 2 | 1 | 0 | 0 | 0 | 0 |
| like | 2 | 0 | 0 | 1 | 0 | 1 | 0 |
| enjoy | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| deep | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| learning | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| NLP | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| flying | 0 | 0 | 1 | 0 | 0 | 0 | 0 |

# Count From Data

| counts | I | like | enjoy | deep | learning | NLP | flying |
|---|---|---|---|---|---|---|---|
| I | 0 | 2 | 1 | 0 | 0 | 0 | 0 |
| like | 2 | 0 | 0 | 1 | 0 | 1 | 0 |
| enjoy | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| deep | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| learning | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| NLP | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| flying | 0 | 0 | 1 | 0 | 0 | 0 | 0 |

(7,7)

SVD

| | S1 | S2 |
|---|---|---|
| I | 1.5 | .1 |
| like | 3.14 | .23 |
| enjoy | 2.7 | -.98 |
| Deep | .55 | .1 |
| learning | .8 | 2.5 |
| NLP | -2.5 | 3 |
| flying | 4.5 | 4.9 |

(7,N)

| Sorted Singular Values | | |
|---|---|---|
| 12.29 | | |
| | 6.2 | |
| | | ... |

(N,N)

| | I | like | .. | .. | .. | .. |
|---|---|---|---|---|---|---|
| S1 | .1 | 2 | 3 | 4 | 6 | 7 |
| S2 | .5 | 6 | 7 | 3 | 1 | 8 |

(N,7)

# Count From Data

| |
|---|
| vec(I) = |
| vec(like) = |
| vec(enjoy) = |
| vec(deep) = |
| vec(learning) = |
| vec(NLP) = |
| vec(flying) = |

| | S1 | S2 |
|---|---|---|
| I | 1.5 | .1 |
| like | 3.14 | .23 |
| enjoy | 2.7 | -.98 |
| Deep | .55 | .1 |
| learning | .8 | 2.5 |
| NLP | -2.5 | 3 |
| flying | 4.5 | 4.9 |

(7,N)

| Sorted Singular Values | | |
|---|---|---|
| 12.29 | | |
| | 6.2 | |
| | | ... |

(N,N)

# Features from Word Vectors

- **Count from Data**
  - Word Co-occurrence + SVD
  - Count-based model
- **Learn from Data**
  - Word2Vec
  - NN Methods
  - Predictive Model
- **Count and Learn from Data**
  - GLOVE: Global Vectors for Word Representation
  - Count + SGD

# One-Hot Encoding (Sparse Representation)

Vocabulary of the corpus (big enough)

| | he | she | eats | drinks | sushi | ramen | hungry | coffee |
|---|---|---|---|---|---|---|---|---|
| he | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| drinks | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| coffee | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |

**v(*'drinks'*)**

# Word2Vec

$$\begin{pmatrix}0\\0\\0\\1\\0\\0\\0\\0\end{pmatrix}^T \times \begin{bmatrix}0.1 & 0.2 & -0.4 & 0.9\\0.2 & 0.1 & -0.3 & 0.9\\0.2 & -1.4 & 0.3 & -0.1\\0.3 & -2.0 & 0.5 & -0.5\\0.2 & -1.1 & 0.3 & -0.7\\0.9 & -1.3 & 0.4 & -0.9\\0.3 & -3.0 & 0.5 & -0.2\\0.5 & -0.1 & 0.2 & 0.1\end{bmatrix} = \begin{pmatrix}0.3\\-0.2\\-0.5\\-0.5\end{pmatrix}^T$$

**One-Hot Encoding** — 1 x |V| — ***v('drinks')***

**Word Embeddings** — |V| x d — ?

**Input** — 1 x d — ***v('drinks')***

# Word2Vec (CBOW)

Learn the Matrix through "classification" task,

**Sentence:** `the bulk of linguistic questions concern the dis- tinction between a and m. a linguistic account of phenomenon …`

| | |
|---|---|
| of | the bulk ______ linguistic questions |
| linguistic | bulk of ______ questions concern |
| questions | of linguistic ______ concern the |
| concern | linguistic questions ______ the dis- |
| the | questions concern ______ dis- tinction |
| dis- | concern the ______ tinction between |
| tinction | the dis- ______ between a |
| between | dis- tinction ______ a and |
| a | tinction between ______ and m. |
| and | between a ______ m. a |
| m. | a and ______ a linguistic |
| a | and m. ______ linguistic account |
| linguistic | m. a ______ account of |
| account | a linguistic ______ of a |
| of | linguistic account ______ a phenomenon |
| a | account of ______ phenomenon gen- |
| phenomenon | of a ______ gen- erally |

***window_size = 2***

More depth in TPML

# Features from WordToVec

- **Count from Data**
  - Word Co-occurrence + SVD
  - Count-based model
- **Learn from Data**
  - Word2Vec
  - NN Methods
  - Predictive Model
- **Count and Learn from Data**
  - **GLOVE**: Global Vectors for Word Representation
  - Count + SGD

# GLOVE-Global Vectors for Word Representation

- **Word-level representation**
- **Counting context-words within a window_size**

Sent_1: *I like deep learning*
Sent_2: *I like NLP*
Sent_3: *I enjoy flying*

**Window_size=1**

| counts | I | like | enjoy | deep | learning | NLP | flying |
|---|---|---|---|---|---|---|---|
| I | 0 | 2 | 1 | 0 | 0 | 0 | 0 |
| like | 2 | 0 | 0 | 1 | 0 | 1 | 0 |
| enjoy | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| deep | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| learning | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| NLP | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| flying | 0 | 0 | 1 | 0 | 0 | 0 | 0 |

(# of "***like***" as "***I***'s" context-words) = 2

(# of "***I***" as "***like***'s" context-words) = 2

# GLOVE-Global Vectors for Word Representation

- **Word-level representation**
- **Counting context-words within a window_size**

Sent_1: *I like deep learning*
Sent_2: *I like NLP*
Sent_3: *I enjoy flying*

**Window_size=1**

| counts | I | like | enjoy | deep | learning | NLP | flying |
|---|---|---|---|---|---|---|---|
| I | 0 | 2 | 1 | 0 | 0 | 0 | 0 |
| like | 2 | 0 | 0 | 1 | 0 | 1 | 0 |
| enjoy | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| deep | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| learning | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| NLP | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| flying | 0 | 0 | 1 | 0 | 0 | 0 | 0 |

P(I,like)= P(like | I ) = Count(# of “***like***” as “***I***’s” context-words)
/ Count( total # of “***I***’s” context-words)

$= C_{I,like} / C_I = 2/3$

$\log (P(I,like)) = \log(C_{I,like} / C_I) = \log(C_{I,like}) - \log(C_I) = \log 2 - \log 3$

# GLOVE-Global Vectors for **Word** Representation

Sent_1: *I like deep learning*
Sent_2: *I like NLP*
Sent_3: *I enjoy flying*

**Window_size=1**

| counts | I | like | enjoy | deep | learning | NLP | flying |
|---|---|---|---|---|---|---|---|
| I | 0 | 2 | 1 | 0 | 0 | 0 | 0 |
| like | 2 | 0 | 0 | 1 | 0 | 1 | 0 |
| enjoy | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| deep | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| learning | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| NLP | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| flying | 0 | 0 | 1 | 0 | 0 | 0 | 0 |

$\log(P(I,like)) = \log(C_{I,like} / C_I) = \log(C_{I,like}) - \cancel{\log(C_I)} = \mathbf{\log 2} - \cancel{\log 3}$

*Let* $\mathbf{v}_i$ = *the vector representing "I"* *i refers to "I"*
$\mathbf{v}_j$ = *the vector representing "like"* *j refers to "like"*

*Then we Expect : mapping* $\mathbf{v}_i \bullet \mathbf{v}_j$ *to* $\log(C_{ij})$

$$\underbrace{\begin{bmatrix} R & R & R & R \end{bmatrix}}_{\mathbf{v}_i} \bullet \underbrace{\begin{bmatrix} R \\ R \\ R \\ R \end{bmatrix}}_{\mathbf{v}_j} \Rightarrow \log(C_{ij})$$

# GLOVE-Global Vectors for Word Representation

Sent_1: *I like deep learning*
Sent_2: *I like NLP*
Sent_3: *I enjoy flying*

**Window_size=1**

| counts | I | like | enjoy | deep | learning | NLP | flying |
|---|---|---|---|---|---|---|---|
| I | 0 | 2 | 1 | 0 | 0 | 0 | 0 |
| like | 2 | 0 | 0 | 1 | 0 | 1 | 0 |
| enjoy | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| deep | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| learning | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| NLP | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| flying | 0 | 0 | 1 | 0 | 0 | 0 | 0 |

*Let* $\mathbf{v}_i$ *= the vector representing "I"* — *i refers to "I"*
$\mathbf{v}_j$ *= the vector representing "like"* — *j refers to "like"*

*Then we Expect : mapping* $\mathbf{v}_i \bullet \mathbf{v}_j$ *to* $\log(C_{ij})$

*Thus we Define: Least Square Loss Function :* $L = \sum_{ij} [\log(C_{ij}) - (\mathbf{v}_i \bullet \mathbf{v}_j + v_{bias})]^2$

$$L2LossFunction = \sum_{i=1}^{n} (y_{true} - y_{predicted})^2$$

# GLOVE-Global Vectors for Word Representation

Sent_1: *I like deep learning*
Sent_2: *I like NLP*
Sent_3: *I enjoy flying*

**Window_size=1**

| counts | I | like | enjoy | deep | learning | NLP | flying |
|---|---|---|---|---|---|---|---|
| I | 0 | 2 | 1 | 0 | 0 | 0 | 0 |
| like | 2 | 0 | 0 | 1 | 0 | 1 | 0 |
| enjoy | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| deep | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| learning | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| NLP | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| flying | 0 | 0 | 1 | 0 | 0 | 0 | 0 |

*Moreover we Define: weighted least square Loss Function :*

$$L = \sum_{i,j} [\log(C_{ij}) - (\mathbf{v}_i \bullet \mathbf{v}_j + v_{bias})]^2 \bullet Weight\_Func(C_{ij})$$

*Constrains:*

*Weight_Func(0) = 0*
*Bigger $C_{ij}$ leads to Bigger Weight_Func($C_{ij}$)*
*Weight_Func($C_{ij}$) should have a upper bound as $C_{ij}$ can be a big number*

# GLOVE-Global Vectors for Word Representation

Sent_1: *I like deep learning*
Sent_2: *I like NLP*
Sent_3: *I enjoy flying*

**Window_size=1**

| counts | I | like | enjoy | deep | learning | NLP | flying |
|---|---|---|---|---|---|---|---|
| I | 0 | 2 | 1 | 0 | 0 | 0 | 0 |
| like | 2 | 0 | 0 | 1 | 0 | 1 | 0 |
| enjoy | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| deep | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| learning | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| NLP | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| flying | 0 | 0 | 1 | 0 | 0 | 0 | 0 |

$$L = \sum_{i,j} [\log(C_{ij}) - (\mathbf{v}_i \bullet \mathbf{v}_j + v_{bias})]^2 \bullet Weight\_Func(C_{ij})$$

$$Weight\_Func(C_{ij}) = \begin{cases} 1, & when\ C_{ij} >= 100 \\ (C_{ij}/100)^{0.75}, & otherwise \end{cases}$$

*Constrains:*

*Weight_Func(0) = 0*

*Bigger $C_{ij}$ leads to Bigger Weight_Func($C_{ij}$)*

*Weight_Func($C_{ij}$) should have a upper bound as $C_{ij}$ can be a big number*

# GLOVE-Global Vectors for Word Representation

Sent_1: *I like deep learning*
Sent_2: *I like NLP*
Sent_3: *I enjoy flying*

**Window_size=1**

| counts | I | like | enjoy | deep | learning | NLP | flying |
|---|---|---|---|---|---|---|---|
| I | 0 | 2 | 1 | 0 | 0 | 0 | 0 |
| like | 2 | 0 | 0 | 1 | 0 | 1 | 0 |
| enjoy | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| deep | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| learning | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| NLP | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| flying | 0 | 0 | 1 | 0 | 0 | 0 | 0 |

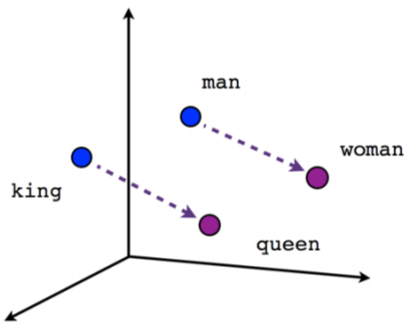
$\log(P(I,like)) = \log(C_{I,like} / C_I) = \log(C_{I,like}) - \cancel{\log(C_I)} = \mathbf{\log 2} - \cancel{\log 3}$

*Let* $\mathbf{v}_i$ = *the vector representing "I"* *i refers to "I"*
$\mathbf{v}_j$ = *the vector representing "like"* *j refers to "like"*

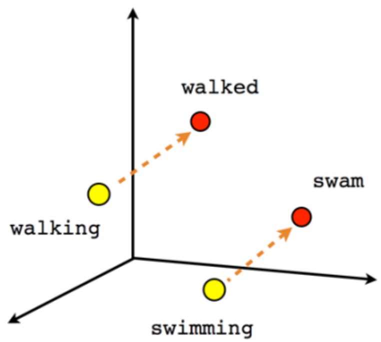
*Then we Expect : mapping* $\mathbf{v}_i \bullet \mathbf{v}_j$ *to* $\log(C_{ij})$

$$\underbrace{\begin{bmatrix} R & R & R & R \end{bmatrix}}_{\mathbf{v}_i} \bullet \underbrace{\begin{bmatrix} R \\ R \\ R \\ R \end{bmatrix}}_{\mathbf{v}_j} = \log(C_{ij})$$

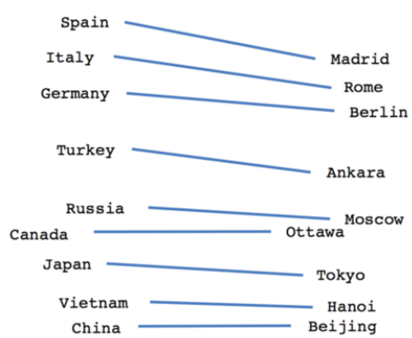
# Properties of Word Vectors




Male-Female




Verb tense




Country-Capital

## Ingredients

| | |
|---|---|
| **Corpus of text** | As large as possible |
| **Annotations** | 0 |
| **Initialize weights (aka Embeddings)** | 1x per word |
| **Deep Learning Model** | 1x |
| **Cost Function** | Appropriately |
| **GPU** | Lotsa of it |

# Features from WordToVec

- **Count from Data (SVD)**
  - Best for **Word-level Similarity**
  - significantly outperformed word2vec and GLOVE
  - Computational cost is High
  - Difficult to handle huge matrix
  - Keep window short (window_size = 2)
- **Learn from Data (Word2Vec)**
  - Cheap to train and Scales with corpus size
  - Generate improvements on other predictive tasks
  - Capture complex patterns beyond word level similarity
- **Count and Learn from Data (GLOVE)**
  - Fast to Train
  - Believed to have advantages from both
  - Differences are not obvious
  - Widely used (pre-trained model from Wikipedia and Twitter)

# Sentence Embedding

| | | | | |
|---|---|---|---|---|
| **v(‘*he*’)** | 0.5 | -1.3 | 0.6 | 1.1 |
| **v(‘*drinks*’)** | 0.3 | -0.2 | 0.5 | 0.5 |
| **v(‘*coffee*’)** | 1.3 | 2.1 | -0.8 | 1.1 |
| | AVG()/MAX()/MIN()/Concat() | | | |
| ***Sent0 (“he drinks coffee”)*** | 0.7 | 0.2 | 0.1 | 0.9 |

Instead of picking *K* most useful features,
here take **N** dimensional Word Embedding

# Combine Feature Sets

| | | | | |
|---|---|---|---|---|
| **v(*‘he’*)** | 0.5 | -1.3 | 0.6 | 1.1 |
| **v(*‘drinks’*)** | 0.3 | -0.2 | 0.5 | 0.5 |
| **v(*‘coffee’*)** | 1.3 | 2.1 | -0.8 | 1.1 |
| | AVG() | | | |
| ***Sent0 (“he drinks coffee”)*** | 0.7 | 0.2 | 0.1 | 0.9 |

| | he | drink | coffee | *tfidf*<br>drink<br>coffee | *tfidf*<br>he<br>drink | *PPMI*<br>he<br>drink | *PPMI*<br>drink<br>coffee | … | Sent Vector | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **sent0** | 0.01 | 0.38 | 0.00 | 0.87 | 0.00 | 4.23 | 0.00 | … | 0.7 | 0.2 | 0.1 | 0.9 |

```
final_train = np.c_[X_w2v_train,X_glove_train,k_best]
final_train.shape
```

# How to Choose Context?

- **Different contexts lead to different embeddings**

- **Small context window:** more syntax related

- **Large context window:** more semantics related

# Limitations

- **Sensitive to "tokens"** (cat vs cats)
- **Inconsistent across space**, embeddings for the same words trained with different data are different
- **Can encode bias** (stereotypical gender roles, racial bias)
- **Not interpretable**

# Summary

- Key steps before building a sentiment analysis are:
  i. *Training data (corpus)* selection/ generation
  ii. *Features selection*
  iii. *Features from embedding*

These pre-steps are key to the success of a sentiment analysis and usually **more important than the training algorithms** themselves.

Training data selection needs to be as similar as possible to the production data. The features selection requires domain expertise.

Word Embeddings can be retrained with domain data or downloaded from pre-trained data