Natural Language Processing

Unit 3

Semantic Analysis

Vector Semantics - Words and Vectors - Cosine Similarity - TF-IDF - Posinve PMI - Word2Vec - Semantic properties of embeddings ; Lexical Semantics : word senses - relations between senses - WordNet - Word Sense Disambiguation

* Vector Semantics

→ refers to the study of the vector representation of words - and models used to represent the meaning of words

→ Refers to representation of a word as some point in some multi-dimensional space

→ Each word is represented as a vector of numerical feature values.

These features are called word embeddings. (embed a word into a vectorspace)

* Motivation behind vector representations

→ Words that occur in similar contexts have similar meanings.

→ The meaning of a word is related to the distribution of words around it

fast ≈ rapid

tall ≈ similar

* Word Similarity

→ can refer to relations that exist beyond synonyms / antonyms

→ Word similarity: words with similar meanings but not synonyms
eg. cat:dog

→ Words are related by a semantic frame or field

eg. cat, dog = similar

student, teach = related, but not similar

Relatedness refers to co-participation in a shared event

* Semantic Field

→ set of words covering a particular semantic domain and having structured relations between them

eg. University

→ teacher, student, project, class, assignment, project

House

→ room, door, furniture, bedroom

* Generating Embeddings for Vector Semantics

→ The meaning of a word is computed from the distribution of words around it.

→ The word vectors are an array of numbers, related in some way to counts.

→ Vector semantics utilizes:

(i) distributionalist intuition

(ii) vector intuition

→ The meaning of word can be defined simply by how often it occurs near other words.

→ This method results in very long vectors that are sparse. This can be changed into short, dense vectors that have useful semantic properties.

* Vectors and Documents

→ Vector or distributional models of meaning are based on a co-occurrence matrix - it shows how often words co-occur.

↳ term-term matrix

→ In a term-document matrix, each row represents a word in the vocabulary and each column represents a document.

Term - Document Representation

→ An example of a term-document matrix is as follows:

→ a vector.

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

→ Each cell in the matrix represents the number of times a particular word occurs in a particular document.

→ The vector for a document identifies a point in a |V|-dimensional space.

→ The documents are points in a 4-dimensional space.

~~Here it is a~~

→ This is the representation of documents as vectors in vectorspace

each vector = column

→ The term-document representation can also represent the meaning of words, by associating each word with a vector.

→ The word ~~vector~~ matrix is a **row vector**

eg.

fool = [36 58 1 4]

→ Similar words have similar vectors because they tend to occur in similar documents.

Term-Term Representation

→ also called word-word matrix or term-context matrix.

→ The rows & columns of the term-term matrix are labeled by words rather than documents

→ This matrix has a $|V| \times |V|$ dimensionality.

→ Each cell records the number of times the row word and the column word co-occur in some context.

→ A window can be created around the word, say 4 words to the left and 4 words to the right.

→ The cell would thus represent the number of times the column word occurs in a ± 4 word window around the row word

eg.

	aardvark	...	computer	data	pinch	result	sugar
apricot	0		0	0	1	0	1
pineapple	0		0	0	1	0	1
digital	0		2	1	0	1	0
information	0		1	6	0	4	0

→ This co-occurrence matrix shows that words like digital & information are more similar to each other.

* Measuring Similarity between Words : Cosine Similarity

→ To find the similarity between 2 word vectors v and w, the cosine of the angle between the vectors can be computed.

→ The cosine similarity between vectors $\vec{v}$ and $\vec{w}$ can be computed as:

$$\cos(\vec{v}, \vec{w}) = \frac{\vec{v} \cdot \vec{w}}{|\vec{v}||\vec{w}|} = \frac{\sum_{i=1}^{n} v_i w_i}{\sqrt{\sum_{i=1}^{n} v_i^2}\sqrt{\sum_{i=1}^{n} w_i^2}}$$

eg. Use cosine similarity to find which of the words apricot or digital is closer in meaning to information.

	large	data	computer
apricot	2	0	0
digital	0	1	2
information	1	6	1

$$\cos(\text{apricot}, \text{information}) = \frac{2+0+0}{\sqrt{4+0+0}\sqrt{1+36+1}} = \frac{2}{2\sqrt{38}} = 0.16$$

$$\cos(\text{digital}, \text{information}) = \frac{0+6+2}{\sqrt{4+0+0}\sqrt{1+36+1}} = \frac{8}{\sqrt{38}\sqrt{5}} = 0.58$$

→ Digital is closer in meaning to information

eg② Find out the cosine similarity between the words (cherry, information) and (digital, information)

	pie	data	computer
cherry	442	8	2
digital	5	1683	1670
information	5	3982	3325

$$\cos(\text{cherry, information}) = \frac{(442*5) + (8*3982) + (2*3325)}{\sqrt{442^2+8^2+2^2}\ \sqrt{5^2+3982^2+3325^2}}$$

$$= \underline{0.18}$$

$$\cos(\text{digital, information}) = \frac{(5*5) + (1683*3982) + (1670*3325)}{\sqrt{5^2+1683^2+1670^2}\ \sqrt{5^2+3982^2+3325^2}}$$

$$= \underline{0.996}$$

⇒ Information is much closer to digital than to cherry

* Term-Frequency - Inverse Document Frequency (TF-IDF)

Disadvantages of using Frequency for word Association

→ Simple frequency is not the best measure of association between words.

→ Words that occur nearby frequently are more important than words that appear only once or twice.

→ On the other hand, words that are too frequent such as 'the' or 'a' are important.

→ These 2 conflicting constraints must be balanced.

* Term Frequency

→ Refers to the frequency of the word in the document

→ Frequency is weighted down by using its log value.

(This is because a word appearing 100 times in a document does not make that word 100 times more likely to be relevant to the meaning of the document)

$$tf_{t,d} = \log_{10}(count(t,d) + 1)$$

* Document Frequency

The document frequency df_t of a term t is the number of documents it occurs in

Inverse Document Frequency

$$idf_t = \log_{10}\left(\frac{N}{df_t}\right)$$

N = total no. of documents

* TF-IDF

$$w_{t,d} = tf_{t,d} \times idf_t$$

→ The TF-IDF vector model of a target word is a vector with dimensions corresponding to all the words in the vocabular.

→ The values in each ~~document~~ dimension are the frequency with with the target word co-occurs with each neighboring context word, weighted by tf-idf.

→ The model computes the similarity between two words x & y by taking the cosine of their tf-idf vectors.

high cosine ⇒ high similarity

Example: Use TF-IDF model and rank the documents

Q : The cat.

D1 : The cat is on the mat.

D2 : My dog and cat are the best.

D3 : The locals are playing.

TF(word, doc) = no. of occurrences of the word / no. of words in the document

Ans TF ("the", D) = $\frac{2}{6} = \frac{1}{3} = 0.33$

TF (the, D2) = $\frac{1}{7} = 0.14$

TF (the, D3) = $\frac{1}{4} = 0.25$

TF (cat, D1) = $\frac{1}{6} = 0.17$

TF (cat, D2) = $\frac{1}{7} = 0.14$

TF (cat, D3) = $\frac{0}{4} = 0$

IDF = $\log\left(\frac{\text{no. of docs}}{\text{no. of docs w/ that word}}\right)$

IDF (the) = $\log\left(\frac{3}{3}\right) = \log(1) = 0$

IDF (cat) = $\log\left(\frac{3}{2}\right) = 0.18$

TF-IDF (word, doc) = TF (word, doc) * IDF(word)

TF IDF (the, D_1)

= 0.33 * 0

= 0

TF IDF (the, D_2) = 0

TF IDF (the, D3) = 0

TF - IDF (cat, D1) = 0.17 x 0.18 = 0.0306

TF - IDF (cat, D_2) = 0.14 x 0.18 = 0.0252

TF IDF (cat, D3) = 0

TF-IDF score for D_1 = 0 + 0.0306 = 0.0306

TF-IDF score for D_2 = 0 + 0.0252 = 0.0252

TF-IDF score for D_3 = 0

* Pointwise Mutual Information (PMI)

→ An alternative weighting function to tf-idf is called PMI.

→ PMI is a measure of how often two events x and y occur, compared with what we expect if they were independent

$$PMI(x,y) = \log_2 \frac{P(x,y)}{P(x)\,P(y)}$$

→ PMI ranges from $-\infty$ to $+\infty$, but the negative values may be problematic.

(i) Things co-occur less than expected by chance

(ii) unreliable without an enormous corpora

Solution - Replace the negative PMI values by 0, thus the positive PMI between 2 words are:

$$PPMI(word1, word2) = \max\left(\log_2 \frac{P(word1, word2)}{P(word1)\,P(word2)}, 0\right)$$

* Computing PPMI on a Term-Context Matrix

Given a co-occurrence matrix with W rows (target words) and C columns (contexts), and f_{ij} gives the number of times word w_i occurs in context c_j

$$p_{ij} = \frac{f_{ij}}{\sum_{i=1}^{W}\sum_{j=1}^{C} f_{ij}} \qquad p_{i*} = \frac{\sum_{j=1}^{C} f_{ij}}{\sum_{i=1}^{W}\sum_{j=1}^{C} f_{ij}}$$

$$p_{*j} = \frac{\sum_{i=1}^{W} f_{ij}}{\sum_{i=1}^{W}\sum_{j=1}^{C} f_{ij}}$$

$$pmi_{ij} = \log_2 \frac{p_{ij}}{p_{i*}\,p_{j*}}$$

$$ppmi_{ij} = \begin{cases} pmi_{ij}, & pmi_{ij} > 0 \\ 0, & \text{otherwise} \end{cases}$$

Example: Compute the PPMI of information, data from the following corpus:

	computer	data	result	pie	sugar	count(w)
cherry	2	8	9	442	25	486
strawberry	0	0	1	60	19	80
digital	1670	1683	85	5	4	3447
information	3325	3982	378	5	13	7703
count(context)	4997	5673	473	512	61	11716

Ans $p_{ij} = \frac{f_{ij}}{\sum_{i=1}^{w}\sum_{j=1}^{c} f_{ij}}$ (no. of times the word w_i occurs in the context c_i)

$p_{ij}(w = \text{information}, c = \text{data}) = \frac{3982}{11716} = 0.3399$

$p_{i*} = P(w = \text{information}) = \frac{7703}{11716} = 0.6575$

$p_{*j} = P(c = \text{data}) = \frac{5673}{11716} = 0.4842$

$PPMI(\text{information}, \text{data}) = \log_2\left(\frac{0.3399}{0.6575 \times 0.4842}\right)$

$= 0.0944$

Example Consider a small corpus consisting of three documents:

D1: The quick brown fox jumps over the lazy dog

D2: The dog barked at the fox, but the fox was too quick.

D3: The lazy dog sleeps all day.

Calculate the TF-IDF for the term 'fox' in each document.

Ans ~~TF(the, D1) = 2/9 = 0.22~~

~~TF(the, D2) =~~

$TF(fox, D1) = \frac{1}{9} = 0.11$

$TF(fox, D2) = \frac{2}{12} = \frac{1}{6} = 0.17$

$TF(fox, D3) = 0$

$IDF(the) = \log\left(\frac{3}{3}\right) = 1 = 0$

$IDF(fox) = \log\left(\frac{3}{2}\right) = 0.176$

≈ 0.18

$TF\text{-}IDF(fox, D1) = 0.11 \times 0.18 = 0.0198$

$TF\text{-}IDF(fox, D2) = 0.17 \times 0.18 = 0.0306$

$TF\text{-}IDF(fox, D3) = 0$

* Short and Dense vector Representations

→ short vectors : 50 - 500

dense : most values are non-zero

short

→ Dense vectors work better in every NLP task than sparse vectors

→ This may be because of the following reasons

(1) Short vectors are easier to use as features in ML models, as there would be fewer weights to tune.

(2) Dense vectors may generalize better than storing explicit counts

(3) Capture synonymy better

(i) Car and automobile are synonyms, but are distinct dimensions in sparse vectors.

(ii) a word w| car as a neighbor and a word with automobile as a neighbor should be similar, but they aren't.

* Word2Vec

→ Word2Vec is a popular embedding method, that is based on the intuition that instead of counting how often each word occurs near another word, a classifier is trained on a binary prediction task.

→ The learned classifier weights are taken as embeddings.

→ The running text is used as implicitly supervised training data. - there is no need for hand-labeled supervision

* Word2Vec - Skipgram with negative sampling

1. Treat the target word and a neighboring context word as positive examples
2. Randomly sample other words in the lexicon to get negative samples
3. Use logistic regression to train a classifier to distinguish the two cases
4. Use the weights as the embeddings.

* Training Word2Vec

Goal : Train the classifier such that a given tuple (t,c)

t = target word

c = context word

will return the probability that c is a real context word

eg. (apricot, jam) = True

(apricot, aardvark) = False

→ This is denoted by $P(+|t,c)$

→ The probability that the word c is not a real context word for t is:

$$P(-|t,c) = 1 - P(+|t,c)$$

Step1 :→ To compute the probability P, consider the similarity

Similarity (t,c) ≈ t · c

→ The dot product t · c is not a probability, it is just a number ranging from 0 − ∞

Step2 : Convert the dot product into a probability (the logistic or sigmoid function can be used

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

→ The probability that word c is a real context word for the target word t is:

$$P(+|t,c) = \frac{1}{1+e^{-t\cdot c}}$$

→ The probability that the word c is not a real context word is :

$$P(-|t,c) = 1 - P(+|t,c)$$

$$= \frac{e^{-t\cdot c}}{1+e^{-t\cdot c}}$$

→ $P(+|t,c)$ and $P(-|t,c)$ give us the probability for one word, but it is necessary to take account of the multiple context words in the window.

* Skip-gram Modelling

→ Skipgram makes the assumption that all context words are independent.

$$P(+|t, c_{1:L}) = \prod_{i=1}^{L} \frac{1}{1+e^{-t\cdot c_i}}$$

$$\log P(+|t, c_{1:L}) = \sum_{i=1}^{L} \log \frac{1}{1+e^{-t\cdot c_i}}$$

→ The skipgram trains a probabilistic classifier that given a test target word t and a context window of L words $c_{1:L}$, a probability is assigned based on how similar the context window is similar to the target window.

→ Word2vec learns embeddings by starting w/ an initial set of embedding vectors and then iteratively shifting the embeddings of each word to be more like the embeddings of words that occur nearby in texts, and less like the embeddings of words that don't occur nearby.

* Skipgram Training

Step1 : Creation of Positive and Negative Training Instances

Positive Training Instances → For a target word, if there is a ±2 window, then there would be 4 positive training instances

eg.

t	c
apricot	tablespoon
apricot	of
aprkot	preserves
apricot	or.

Negative Training Instances → For training the binary classifier, negative examples are also needed and, skip-gram uses more negative examples than positive examples

→ For each (t,c) instance, create k negative samples, each consisting of the target and a noise word.

→ A ~~random~~ noise word is a random word from the lexicon.

→ When k=2, there would be 2 negative examples in the training set for each positive example (t,c).

eg.

t	c
apricot	aardvark
apricot	puddle
apricot	where
apricot	coaxial

t	c
apricot	twelve
apricot	hello
apricot	dear
apricot	forever.

→ The noise words are chosen according to their unigram frequency $p(w)$

→ If the sampling is done with unweighted frequency $p(w)$, then by the unigram probability $p(\text{'the'})$, 'the' would be chosen as the noise word.

→ In practice, a weighted unigram frequency $p_\alpha(w)$ is used, where α is a weight

$$p_\alpha(w) = \frac{\text{count}(w)^\alpha}{\sum_{w'} \text{count}(w')^\alpha}$$

usually $\alpha = 0.75$

Step 2 : Initialization and Optimization

→ Represent the words as vectors of some length, randomly initialized (say 300)

There is thus, $300 * V$ random parameters

→ Over the training set, adjust the word vectors such that :

(i) Maximize the similarity of the target word, context word pairs (t,c) from the positive data

(ii) Minimize the similarity of the (t,c) pairs drawn from the negative data.

i.e maximize

$$\sum_{(t,c) \in +} \log P(+|t,c) + \sum_{(t,c) \in -} \log P(-|t,c)$$

→ Focusing on one word/context pair (t, c) with k noise words $n_1, n_2 \ldots n_k$, the learning objective is:

$$L(\theta) = \log P(+|t,c) + \sum_{i=1}^{k} \log P(-|t,n_i)$$

$$= \log \sigma(c \cdot t) + \sum_{i=1}^{k} \log \sigma(-n_i \cdot t)$$

$$= \log\left(\frac{1}{1+e^{-ct}}\right) + \sum_{i=1}^{k} \log\left(\frac{1}{1+e^{n_i t}}\right)$$

→ Use stochastic gradient descent to train to this objective, iteratively modifying the parameters (embeddings for t and c)

→ The skip-gram model thus has two separate embeddings for each word w: the target embedding t, and the context embedding c.

→ The embeddings are stored in 2 matrices: target matrix W and context matrix C.

Target Matrix = ⌈ ⌉ ⌈ ⌉ ⌈ ⌉ → Each row is a 1 × d vector of embedding t_i

Context Matrix [ ] [ ] [ ] → each column is a d × 1 vector embedding c_i for word i in V

* Visualizing Embeddings

→ The simplest way to visualize the meaning of a word w is embeddings is to list the most similar words to w sorting all the words in the vocabulary by their cosines.

→ Another method is to use a clustering algorithm to show a hierachical representation of which words are similar to others in the embedding space.

* Semantic Properties of Embeddings

Similarity - similarity depends on the window size C

eg. C ±5 The nearest words to Hogwarts:

(i) Dumbledore

(ii) Malfoy

(iii) halfblood

Co-occurrence

First Order Co-occurrence → If 2 words are typically nearby one another

eg. wrote is a first-order associate of book or poem.

Second-Order Co-occurrence → Two words have second-order cooccurrate if they have similar neighbors

eg. wrote is a second-order associate of said / remarked

Relational Meanings - Embeddings have the ability to capture relational meanings.

For eg By projecting vectors onto two dimensions:

'king' - 'man' are close

'queen' - 'woman' are close

→ Similarity offsets in the 2D projections can capture comparative and superlative morphology.

* Lexical Semantics

→ Lexical semantics is the study of meaning-related connections among lexemes, and their internal meaning-related structure of individual lexemes

→ Lexeme = individual entry in the lexicon

Lexicon = finite list of lexemes

→ Semantics of individual lexemes can be captured by analyzing and labelling their relations to other lexemes

* Word Senses

A: **Homonymy**

→ Relation that holds between words that have the same form with unrelated meanings

eg. bank has 2 senses

Financial institute

sloping mound

B. **Homophone**

→ words with the same pronunciation but different spellings

eg. would | wood

sun | son

c. **Homographs**

→ words with the same orthographic form but different pronunciations

eg. bass → fish

bass → music

Text-to-speech & IR systems are vulnerable to homographs.

Speech recognition, spelling correction systems are vulnerable to homophones

---

Homonyms vs. Homophones vs. Homographs

↓ ↓ ↓

multiple meaning words | words that sound alike | same spelling, different pronunciation, different meanings

(sprucetree, spruce up) | (to, too) | (desert, dessert, bow, bow)

D. **Polysemy**

→ Polysemy relates to multiple related meanings within a single lexeme

eg. bank = repository of entities

eg. blood bank, egg bank, eye bank

→ This sense of bank is derived from the financial institution sense.

* Relations between Senses

A. Synonymy

→ Different lexicons with the same meaning

eg. couch / sofa

vomit / throw up

car / automobile

→ Two words are synonymous if they are substitutable one for another in any sentence without changing the truth conditions of the sentence

→ However, no two words are exactly identical in meaning - eg big & large sometimes cannot be used interchangeably

→ In practice synonym is used to describe a relationship of rough synonymy.

B. Antonymy

→ words with opposite meaning

→ Two senses can be antonyms if they define a binary opposition or are at opposite ends of some scale

eg. long / short

fast / slow

→ Though antonyms have very different meanings - they also share all aspects of their meaning except their position on a scale or their direction.

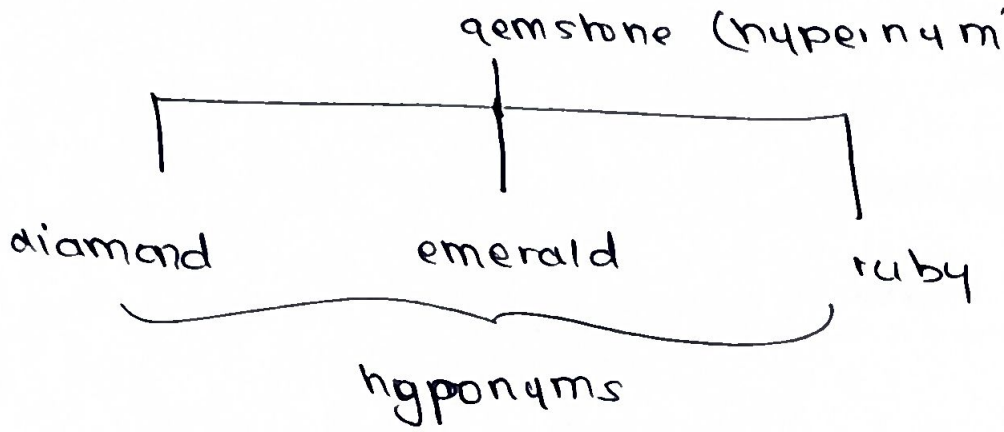
c. Hyponymy

→ One lexeme denotes a subclass of the other

→ eg. car is a hyponym of vehicle

mango is a hyponum of fruit

→ Conversely, vehicle is a hypernym of car.



* WordNet → a large electronic lexical database, that can be used to find synonyms, antonyms and other semantic relationships between words

→ It is a hierarchically organized lexical database.

* Word Sense Disambiguation

* Disambiguation

→ Many words have several meanings or senses, and thus there is an ambiguity about how they are to be interpreted.

→ The task of disambiguation is to determine which of the senses of an ambiguous word invoked in a particular use of the word

→ This is done by looking at the context of the word's use.

→ The task of WSD is to make a choice between senses of an ambiguous word, based on the context of use

* Supervised Word Sense Disambiguation

→ Extract features from the text and then train a classifier to assign the correct sense given these feature.

→ A feature vector consisting of numeric values is used to encode this linguistic information as an input to ML algorithms

→ Two classes of features are:

(i) collocational features

(ii) bag-of-words features

A. Co-locational Features

→ A collocation is a word or a phrase in a position-specific relationship to a target word

→ Collocational features encode information about specific positions located to the left or right of the target word

eg. exactly 1 word to the right, or exactly 4 words to the left.

→ Typical features include the word itself, the root form of the word, and the word's pos.

For eg. An electric guitar and bass plar stand off to one side

The collocational feature-vector to disambiguate bass could be:

$[w_{i-2}, POS_{i-2}, POS_{i-1}, w_{i+1}, POS_{i+1}, w_{i+2}, POS_{i+2}]$

[guitar, NN, and, CC, player, NN, stand, VB]

B. Bag-of-words features

→ A bag of words means an unordered set of words, ignoring their exact position.

→ The bag of words approach aims to find the context of a target word by a vector of features. Each binary feature indicates whether a vocabulary word w does or does not occur in the context.

eg. An electric guitar and bass player stand off to one side.

= [Fishing, big, sound, player, fly, rod, pound, double, runs, playing, guitar, band]

= [0, 0, 0, (1), 0, 0, 0, 0, 0, 0, (1), 0]

* Methods for WSD

① Supervised ML – Naive Bayes Approach

② Dictionary-based – & Lesk Algorithm

③ Bootstrapping – Yarowsky Algorithm

A. Naive Bayes Approach

→ Choosing the best senses, for a feature vector f involves choosing the most probable sense given that vector.

$$\hat{s} = \text{argmax}_s P(s|f)$$
$$= \text{argmax}_s P(f|s) P(s) / P(f)$$
$$= \text{argmax}_s P(f|s) P(s)$$
$$= \text{argmax}_s P(s) \prod P(f|s)$$

P(f) = constant for all senses

P(s) = prior probability of each case

P(f|s) = individual feature probability

Ex1 Identify the word sense of the test data from the given training data.

Training

Doc	Words	class
1	fish smoked fish	f
2	fish line	f
3	fish haul smoked	f
4	guitar jazz line	g

Test

line guitar jazz jazz : ?

Ans

Prior Probabilities

$P(f) = \frac{3}{4}$

$P(g) = \frac{1}{4}$

$V = \{fish, smoked, line, haul, guitar, jazz\}$

Conditional probabilities

$P(w|c) = \frac{count(w,c)+1}{count(c)+|V|}$

↓

no. of words that are there in the training set of that class

$P(line|f) = \frac{1+1}{8+6} = \frac{2}{14}$

$P(guitar|f) = \frac{0+1}{8+6} = \frac{1}{14}$

$P(jazz|f) = \frac{0+1}{8+6} = \frac{1}{14}$

$P(line|g) = \frac{1+1}{3+6} = \frac{2}{9}$

$P(guitar|g) = \frac{1+1}{3+6} = \frac{2}{9}$

$P(jazz|g) = \frac{1+1}{3+6} = \frac{2}{9}$

Choosing a class

$$P(\text{flas}) = \frac{3}{4} \times \frac{2}{14} \times \frac{1}{14} \times \left(\frac{1}{14}\right)^2$$

$$= 0.00003$$

$$P(\text{9l df}) = \frac{1}{14} \times \frac{2}{9} \times \left(\frac{2}{9}\right)^2 \times \frac{2}{9}$$

$$= 0.0006$$

The test sentence belongs to class g.

B. **Dictionary-based Approach - Lesk Algorithm**

→ Choose the senso whose dictionary gloss or definiron shares the most words with the target word's neighborhood.

Algorithm

```
function LESK (word, sentence) returns best sense of word
    best-sense ← most frequent sense for word
    max-overlap ← 0
    context ← set of words in sentence
    for each sense in senses of word do
        signature ← set of words in the gloss
        overlap ← COMPUTEOVERLAP(signature, context)
        if overlap > max-overlap then
            max-overlap ← overlap
            best-sense ← sense
    end
    return (best sense)
```

eg. "The bank can guarantee deposits will eventually cover future tuition costs because it invests in adjustable-rate mortgage securities"

bank' → financial institution — has more overlapping words than bank² (sloping land), so bank 1 is chosen.

Limitations of Lesk Algorithm

→ The dictionary entries for the target words are short, and may not provide enough chance of overlap with the context

Solution Apply a weight to each overlapping word. The weight is the inverse document frequency (IDF)

c. Bootstrapping - Yarowsky Algorithm

→ Given a small seed set $\Lambda^{(0)}$ of labelled instances of each sense, and a much larger unlabelled corpus $V^{(0)}$

→ The algorithm first trains an initial decision-list classifier on the seed set $\Lambda^{(0)}$

→ Use this classifier to label the unlabeled corpus $V^{(0)}$

→ Select the examples in $V^{(0)}$ that it is most confident about, removes them, and adds them to the training set - called it $\Lambda^{(1)}$

→ Iterate by applying the classifier to the unlabeled set $V^{(1)}$ extracting a new training set $\Lambda^{(2)}$ and so on.

→ With each iteration, the training corpus grows and the untagged corpus shrinks.

→ The process is repeated until there is some sufficiently low error-rate on the training set, or until no further examples from the untagged corpus are above threshold.

→ The key to bootstrapping - an accurate & initial set of seeds. One way to generate the initial seeds is to hand-label a small set of examples or to use a heuristic to select accurate seeds.

→ Yarowsky used the One Sense per Collocation heuristic : which relies on, certain words or phrases strongly associated with the target senses, do not occur with the other sense.