Unit 7
Applications of NLP

Natural Language Processing (NLP)
MDS 555

Objective

- N-Gram Language Model
- TF-IDF
  - Algorithm
  - Implementation – already done
- Vector Semantics and Embeddings

N-Gram Language Models

- Models that assign probabilities to sequences of words are called language models or LMs
- An n-gram is a sequence n-gram of n words
  - **2-gram** (a bigram) is a two-word sequence of words like “please turn”, “turn your”, or ”your homework”
  - **3-gram** (a trigram) is a three-word sequence of words like “please turn your”, or “turn your homework”.

N-Grams

- The probability of a word *w* given some history *h*

  $P(w|h)$

- Suppose the history h is "its water is so transparent that" and we want to know the probability that the next word is "the"

  $P(\text{the}|\text{its water is so transparent that})$

N-Grams

- One way to estimate this probability is from relative frequency counts:
  - take a very large corpus,
  - count the number of times we see “its water is so transparent that”, and
  - count the number of times this is followed by ”the”

N-Grams

- This would be answering the question “Out of the times we saw the history h, how many times was it followed by the word w”, as follows

$$P(the \mid its\ water\ is\ so\ transparent\ that) = \frac{C(its\ water\ is\ so\ transparent\ that\ the)}{C(its\ water\ is\ so\ transparent\ that)}$$

- In many case counting this way will not results to the solution due to dynamic nature of the language
- It is not appropriate to count in larger corpus

N-Grams

- **Chain Rule of Probability**
  - The chain rule shows the link between computing the joint probability of a sequence and computing the conditional probability of a word given previous words

$$\begin{aligned} P(X_1 \ldots X_n) &= P(X_1)P(X_2|X_1)P(X_3|X_{1:2}) \ldots P(X_n|X_{1:n-1}) \\ &= \prod_{k=1}^{n} P(X_k|X_{1:k-1}) \end{aligned}$$

N-Grams

- We don’t know any way to compute the exact probability of a word given a long sequence of preceding words,

  $P(w_n|w1:n-1)$.

- As we said above, we can’t just estimate by counting the number of times every word occurs following every long string, because
  - language is creative and
  - any particular context might have never occurred before

Bigram

- The bigram model
  - Approximates the probability of a word given all the previous words $P(w_n|w_{1:n-1})$ by using only the conditional probability of the preceding word

    $P(w_n|w_{n-1})$

- In other words, instead of computing the probability

  P(the|Walden Pond's water is so transparent that)

  - we approximate it with the probability

    P(the|that)

Bigram

- When we use a bigram model to predict the conditional probability of the next word, we are thus making the following approximation:

$$P(w_n|w_{1:n-1}) \approx P(w_n|w_{n-1})$$

- The assumption that the probability of a word depends only on the previous word is called a Markov assumption

N-Gram Approximation

- N mean the n-gram size,
  - N = 2 means **bigrams** and N = 3 means **trigrams**
- Then we approximate the probability of a word given its entire context as follows:

$$P(w_n|w_{1:n-1}) \approx P(w_n|w_{n-N+1:n-1})$$

- Given the bigram assumption for the probability of an individual word, we can compute the probability of a complete word sequence by

$$P(w_{1:n}) \approx \prod_{k=1}^{n} P(w_k|w_{k-1})$$

Maximum likelihood estimation - MLE

- An intuitive way to estimate probabilities
- We get MLE estimate for the parameters of an n-gram model by getting counts from a corpus, and normalizing the counts so that they lie between 0 and 1

Maximum likelihood estimation - MLE

- An intuitive way to estimate probabilities
- We get MLE estimate for the parameters of an n-gram model by getting counts from a corpus, and normalizing the counts so that they lie between 0 and 1
  - To compute a particular bigram probability of a word w_n given a previous word w_{n-1},
    - we'll compute the count of the bigram $C(w_{n-1}w_n)$ and normalize by the sum of all the bigrams that share the same first word w_{n-1}

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{\sum_w C(w_{n-1}w)}$$

MLE

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{\sum_w C(w_{n-1}w)}$$

- We can simplify this equation,
  - the sum of all bigram counts that start with a given word w_{n-1} must be equal to the unigram count for that word w_{n-1}

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})}$$

Bigram Counts

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

Figure 3.1 Bigram counts for eight of the words (out of $V = 1446$) in the Berkeley Restaurant Project corpus of 9332 sentences. Zero counts are in gray.

Bigram probabilities

i	want	to	eat	chinese	food	lunch	spend
2533	927	2417	746	158	1093	341	278

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

Figure 3.2 Bigram probabilities for eight words in the Berkeley Restaurant Project corpus of 9332 sentences. Zero probabilities are in gray.

Use of bigram probability table

Here are a few other useful probabilities:

$$P(\texttt{i}|\texttt{<s>}) = 0.25 \qquad P(\texttt{english}|\texttt{want}) = 0.0011$$
$$P(\texttt{food}|\texttt{english}) = 0.5 \qquad P(\texttt{</s>}|\texttt{food}) = 0.68$$

Now we can compute the probability of sentences like *I want English food* or *I want Chinese food* by simply multiplying the appropriate bigram probabilities together, as follows:

$$\begin{aligned}
&P(\texttt{<s> i want english food </s>}) \\
&\quad = P(\texttt{i}|\texttt{<s>})P(\texttt{want}|\texttt{i})P(\texttt{english}|\texttt{want}) \\
&\qquad\quad P(\texttt{food}|\texttt{english})P(\texttt{</s>}|\texttt{food}) \\
&\quad = .25 \times .33 \times .0011 \times 0.5 \times 0.68 \\
&\quad = .000031
\end{aligned}$$

Evaluating Language Model

- **Extrinsic evaluation**
  - End-to-End
  - The best way to evaluate the performance of a language model is to embed it in an application and measure how much the application improves
  - Example: Speech recognition
    - we can compare the performance of two language models by running the speech recognizer twice,
    - once with each language model, and seeing which gives the more accurate transcription.
  - It is very expensive

Evaluating Language Model

- **Intrinsic evaluation**
  - Measures the quality of a model independent of any application
  - We need a test set
  - We can then measure the quality of an n-gram model by its performance on some unseen data called the test set or test corpus

Limitation of Probabilistic Evaluation

- It's important not to let the test sentences into the training set
- If our test sentence is part of the training corpus, we will mistakenly assign it an artificially high probability when it occurs in the test set. We call this situation **training on the test set**.

Perplexity (PPL)

- In practice we don't use raw probability as our metric for evaluating language models, but a variant called **perplexity**.
- The perplexity of a language model on a test set is the inverse probability of the test set, normalized by the number of words.

$$\begin{aligned}\text{perplexity}(W) &= P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}} \\ &= \sqrt[N]{\frac{1}{P(w_1 w_2 \ldots w_N)}}\end{aligned}$$

We can use the chain rule to expand the probability of W:

$$\text{perplexity}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_1 \ldots w_{i-1})}}$$

Self Study

- BOOK
  - Speech and Language Processing (3rd Edition)
    Chapter 3 : N-gram Language Models

Text Vectorization

Text Vectorization

- Text Vectorization is the process of converting text into numerical representation.
- A technique for converting text into finite length vectors
  - Bag-of-Words
  - TF-IDF
  - CBOW
  - Skip Gram
  - Word2Vec
- Code the text into the numeric values

Bag of words

- We represent a text document as if it were a bag of words, that is, an unordered set of words with their position ignored, keeping only their frequency in the document.

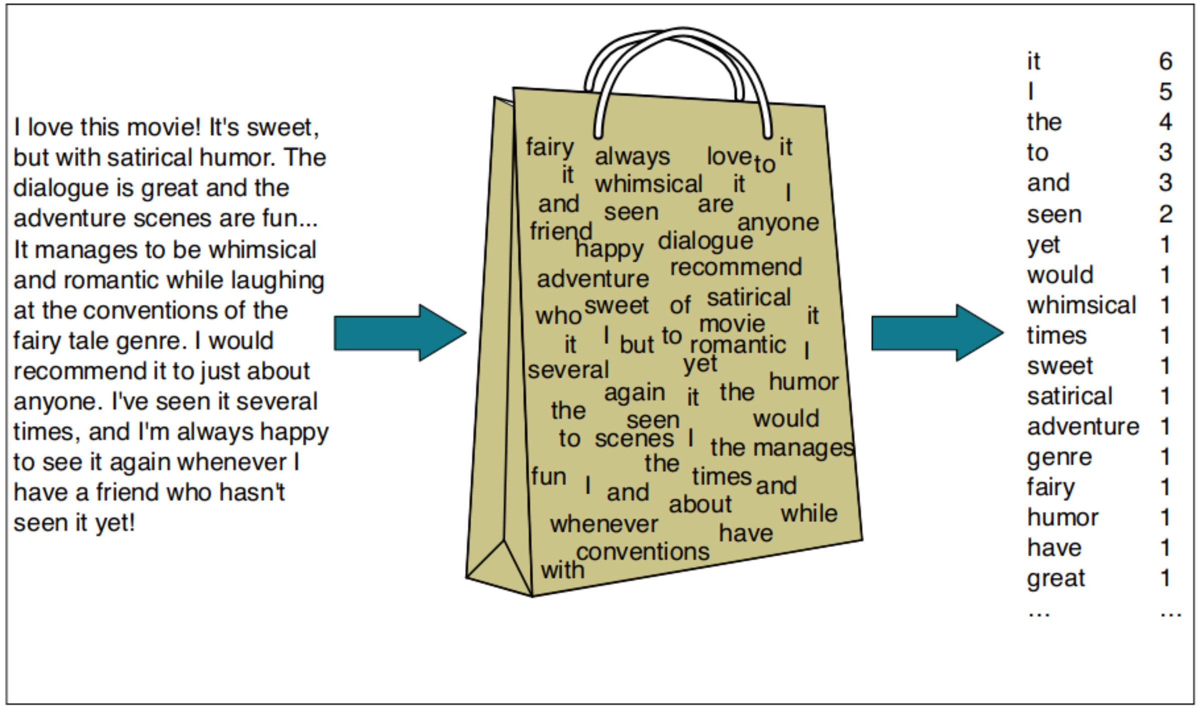


Figure 4.1 Intuition of the multinomial naive Bayes classifier applied to a movie review. The position of the words is ignored (the *bag-of-words* assumption) and we make use of the frequency of each word.

NLP Applications - Classifiers

- Self study
  - Text / Document Classification
    - Naive Bayes
    - Self study: how Naive Bayes can be used to build Language model
      - Chapter 4: Naive Bayes and Sentiment Classification
    - Evaluation: Precision, Recall and F-measure

Vector Semantics

Distributional Hypothesis

- Words that occur in similar contexts tend to have similar meanings. This link between similarity in how words are distributed and similarity in what they mean is called the distributional hypothesis.
  - The hypothesis was distributional hypothesis first formulated in the 1950s by linguists like Joos (1950), Harris (1954), and Firth (1957),
  - who noticed that
    - words which are synonyms (like oculist and eye-doctor) tended to occur in the same environment (e.g., near words like eye or examined) with the amount of meaning difference between two words “corresponding roughly to the amount of difference in their environments” (Harris, 1954, 1957).

Vector Semantics

- Vector semantics is the standard way to represent word meaning in NLP, helping us model many of the aspects of word meaning
- The idea of vector semantics is to represent a word as a point in a multidimensional semantic space that is derived from the distributions of word neighbors.
- Vectors for representing words are called **embeddings**
  - The word “embedding” derives from its mathematical sense as a mapping from one space or structure to another, although the meaning has shifted

Embeddings

Figure 6.1 A two-dimensional (t-SNE) projection of embeddings for some words and phrases, showing that words with similar meanings are nearby in space. The original 60-dimensional embeddings were trained for sentiment analysis. Simplified from Li et al. (2015) with colors added for explanation.

Words and Vectors

- Vector or distributional models of meaning are generally based on a co-occurrence matrix, a way of representing how often words co-occur.
- Two popular matrices:
  - Document Dimension: the term-document matrix and
  - Word Dimention: the term-term matrix

Term-document matrix

- In a term-document matrix,
  - each row represents a word in the vocabulary and
  - each term-document matrix column represents a document from some collection of documents
    - The term-document matrix of Fig. 6.2 was first defined as part of the **vector space model** of **information retrieval** (Salton, 1971).

	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

Figure 6.2 The term-document matrix for four words in four Shakespeare plays. Each cell contains the number of times the (row) word occurs in the (column) document.

Term-document matrix

- The term-document matrix has
  - |V| rows (one for each word type in the vocabulary) and
  - D columns (one for each document in the collection)

	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

Figure 6.3 The term-document matrix for four words in four Shakespeare plays. The red boxes show that each document is represented as a column vector of length four.

Visualizing the vectors

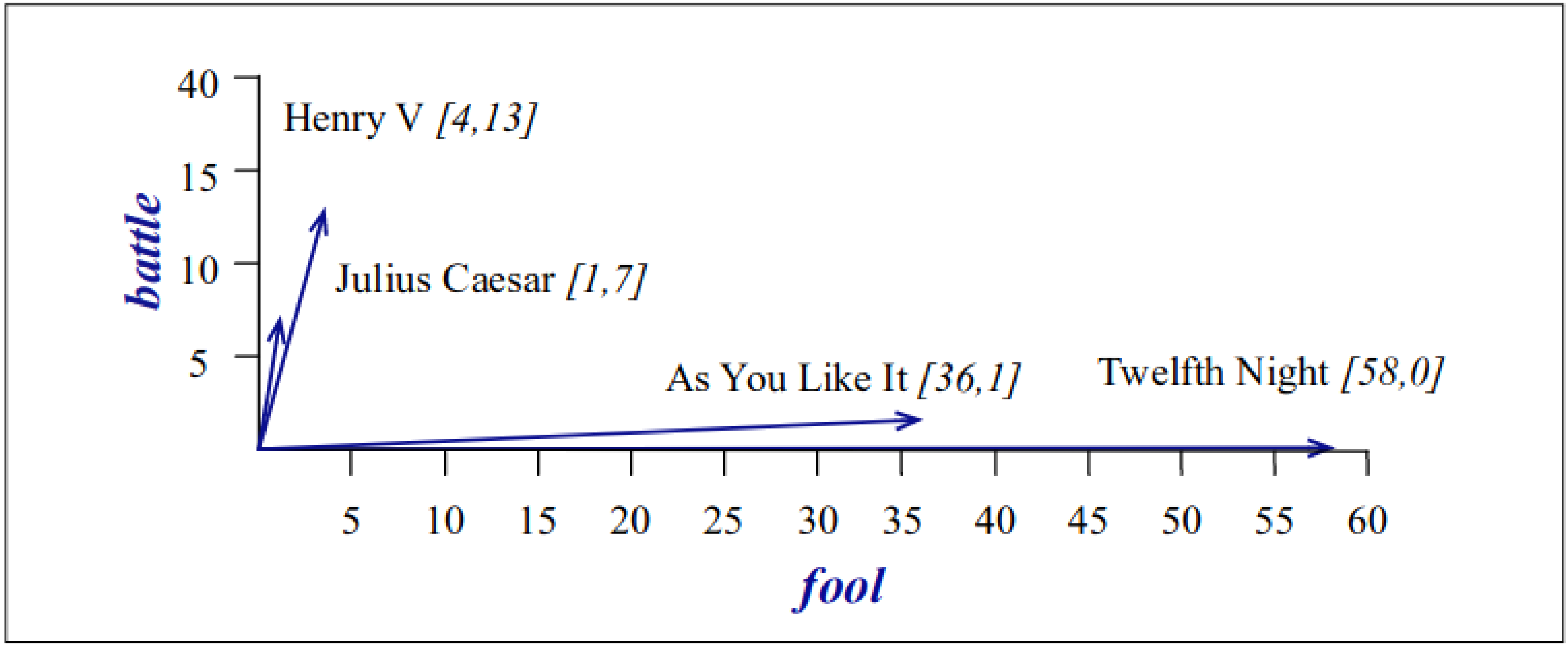


Figure 6.4 A spatial visualization of the document vectors for the four Shakespeare play documents, showing just two of the dimensions, corresponding to the words *battle* and *fool*. The comedies have high values for the *fool* dimension and low values for the *battle* dimension.

Term-Term Matrix

- A term-document matrix represents words as vectors of their counts across documents. An alternative is the **term-term matrix** (also called the word-word or term-context matrix), where the columns are labeled by words instead of documents.
  - This matrix is thus of dimensionality $|V| \times |V|$ and each cell records the number of times the row (target) word and the column (context) word co-occur in some context in some training corpus.

	aardvark	**...**	**computer**	**data**	**result**	**pie**	**sugar**	**...**
cherry	0	...	2	8	9	442	25	...
strawberry	0	...	0	0	1	60	19	...
digital	0	...	1670	1683	85	5	4	...
information	0	...	3325	3982	378	5	13	...

Figure 6.6 Co-occurrence vectors for four words in the Wikipedia corpus, showing six of the dimensions (hand-picked for pedagogical purposes). The vector for *digital* is outlined in red. Note that a real vector would have vastly more dimensions and thus be much sparser.

Term-Term Matrix

	aardvark	...	computer	data	result	pie	sugar	...
cherry	0	...	2	8	9	442	25	...
strawberry	0	...	0	0	1	60	19	...
digital	0	...	1670	1683	85	5	4	...
information	0	...	3325	3982	378	5	13	...

Figure 6.6 Co-occurrence vectors for four words in the Wikipedia corpus, showing six of the dimensions (hand-picked for pedagogical purposes). The vector for *digital* is outlined in red. Note that a real vector would have vastly more dimensions and thus be much sparser.

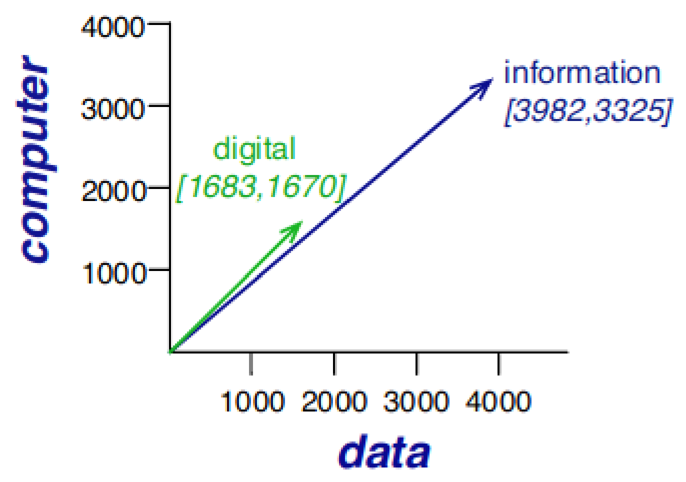


Figure 6.7 A spatial visualization of word vectors for *digital* and *information*, showing just two of the dimensions, corresponding to the words *data* and *computer*.

Cosine for measuring similarity

- To measure similarity between two target words v and w, we need a metric that takes two vectors and gives a measure of their similarity
  - same dimensionality needed
    - either both with words as dimensions, hence of length |V|,
    - or both with documents as dimensions, of length |D|
- By far the most common similarity metric is the cosine of the angle between the vectors

Vector Dot Product

- The dot product acts as a similarity metric because it will tend to be high just when the two vectors have large values in the same dimensions

$$\text{dot product}(\mathbf{v},\mathbf{w}) = \mathbf{v}\cdot\mathbf{w} = \sum_{i=1}^{N} v_i w_i = v_1 w_1 + v_2 w_2 + \ldots + v_N w_N \tag{6.7}$$

  - This raw dot product, however, has a problem as a similarity metric: it favors long vectors.
    - higher if a vector is longer
    - higher for frequent words

Normalized dot product

- Same as the cosine of the angle between the two vectors following from the definition of the dot product between two vectors a and b

$$\mathbf{a} \cdot \mathbf{b} = |\mathbf{a}||\mathbf{b}| \cos\theta$$
$$\frac{\mathbf{a} \cdot \mathbf{b}}{|\mathbf{a}||\mathbf{b}|} = \cos\theta$$

- The cosine similarity metric between two vectors v and w thus can be computed as:

$$\text{cosine}(\mathbf{v}, \mathbf{w}) = \frac{\mathbf{v} \cdot \mathbf{w}}{|\mathbf{v}||\mathbf{w}|} = \frac{\sum_{i=1}^{N} v_i w_i}{\sqrt{\sum_{i=1}^{N} v_i^2}\sqrt{\sum_{i=1}^{N} w_i^2}}$$

Cosine Similarity

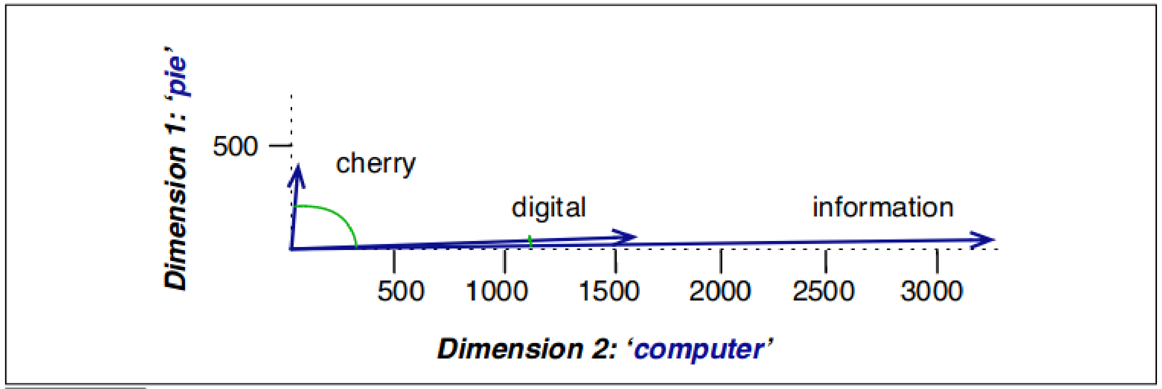
$$\text{cosine}(\mathbf{v},\mathbf{w}) = \frac{\mathbf{v}\cdot\mathbf{w}}{|\mathbf{v}||\mathbf{w}|} = \frac{\sum_{i=1}^{N} v_i w_i}{\sqrt{\sum_{i=1}^{N} v_i^2}\sqrt{\sum_{i=1}^{N} w_i^2}}$$

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

$$\cos(\text{cherry},\text{information}) = \frac{442*5+8*3982+2*3325}{\sqrt{442^2+8^2+2^2}\sqrt{5^2+3982^2+3325^2}} = .018$$

$$\cos(\text{digital},\text{information}) = \frac{5*5+1683*3982+1670*3325}{\sqrt{5^2+1683^2+1670^2}\sqrt{5^2+3982^2+3325^2}} = .996$$

- When two vectors are more similar, the cosine is larger but the angle is smaller; the cosine has its maximum (1) when the angle between two vectors is smallest (0◦); the cosine of all other angles is less than 1



TF-IDF: Weighing terms in the vector

- Words that occur nearby frequently (maybe pie nearby cherry) are more important than words that only appear once or twice.
- Yet words that are too frequent, like the or good— are less important
  - How can we balance these two conflicting constraints?

IF-IDF

- Term Frequency — Inverse Document Frequency (TF-IDF)
  - A technique for converting text into finite length vectors
  - Gives insights about the less relevant and more relevant words in a document
  - The importance of a word in the text is of great significance in information retrieval

Term Frequency

- It is a measure of the frequency of a word (w) in a document (d).
  - TF is defined as the ratio of a word's occurrence in a document to the total number of words in a document.
  - The denominator term in the formula is to normalize since all the corpus documents are of different lengths.

$$TF(w, d) = \frac{occurences\ of\ w\ in\ document\ d}{total\ number\ of\ words\ in\ document\ d}$$

Term Frequency (TF)

- The initial step is to make a vocabulary of unique words and calculate TF for each document.
- TF will be more for words that frequently appear in a document and less for rare words in a document.

Documents	Text	Total number of words in a document
A	Jupiter is the largest planet	5
B	Mars is the fourth planet from the sun	8

Words	TF (for A)	TF (for B)
Jupiter	1/5	0
Is	1/5	1/8
The	1/5	2/8
largest	1/5	0
Planet	1/5	1/8
Mars	0	1/8
Fourth	0	1/8
From	0	1/8
Sun	0	1/8

Inverse Document Frequency (IDF)

- It is the measure of the importance of a word.
  - Term frequency (TF) does not consider the importance of words
    - Some words such as’ of’, ‘and’, etc. can be most frequently present but are of little significance
  - IDF provides weightage to each word based on its frequency in the corpus D

Inverse Document Frequency (IDF)

- IDF of a word (w) is defined as
  - ln = $\log_e$

$$IDF(w, D) = \ln\left(\frac{Total\ number\ of\ documents\ (N)\ in\ corpus\ D}{number\ of\ documents\ containing\ w}\right)$$

Inverse Document Frequency (IDF)

- In our example, since we have two documents in the corpus, N=2.

Words	TF (for A)	TF (for B)	IDF
Jupiter	1/5	0	ln(2/1) = 0.69
Is	1/5	1/8	ln(2/2) = 0
The	1/5	2/8	ln(2/2) = 0
largest	1/5	0	ln(2/1) = 0.69
Planet	1/5	1/8	ln(2/2) = 0
Mars	0	1/8	ln(2/1) = 0.69
Fourth	0	1/8	ln(2/1) = 0.69
From	0	1/8	ln(2/1) = 0.69
Sun	0	1/8	ln(2/1) = 0.69

TF-IDF

- It is the product of TF and IDF.
  - TFIDF gives more weightage to the word that is rare in the corpus (all the documents).
  - TFIDF provides more importance to the word that is more frequent in the document.

$$TFIDF\ (w, d, D) = TF(w, d) * IDF(w, D)$$

Why Ln in the IDF?

- TFIDF is the product of TF with IDF.
  - Since TF values lie between 0 and 1,
  - Not using ln can result in high IDF for some words, thereby dominating the TFIDF. We don’t want that, and therefore
- We use ln so that IDF should not completely dominate the TFIDF.

TF-IDF

Words	TF (for A)	TF (for B)	IDF	TFIDF (A)	TFIDF (B)
Jupiter	1/5	0	ln(2/1) = 0.69	0.138	0
Is	1/5	1/8	ln(2/2) = 0	0	0
The	1/5	2/8	ln(2/2) = 0	0	0
largest	1/5	0	ln(2/1) = 0.69	0.138	0
Planet	1/5	1/8	ln(2/2) = 0	0.138	0
Mars	0	1/8	ln(2/1) = 0.69	0	0.086
Fourth	0	1/8	ln(2/1) = 0.69	0	0.086
From	0	1/8	ln(2/1) = 0.69	0	0.086
Sun	0	1/8	ln(2/1) = 0.69	0	0.086

Disadvantage of TFIDF

- It is unable to capture the semantics
  - For example, funny and humorous are synonyms, but TFIDF does not capture that.
- Moreover, TFIDF can be computationally expensive if the vocabulary is vast.

Pointwise mutual information

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

$$I(x,y) = \log_2 \frac{P(x,y)}{P(x)P(y)} \tag{6.16}$$

The pointwise mutual information between a target word w and a context word c (Church and Hanks 1989, Church and Hanks 1990) is then defined as:

$$\text{PMI}(w,c) = \log_2 \frac{P(w,c)}{P(w)P(c)} \tag{6.17}$$

Read more – yourself. Chapter 6 – PMI, PPMI

Text Embedding
Word2vec

Word2vec

- Unlike the vectors we've seen so far, embeddings are short, with number of dimensions d ranging from 50-1000, rather than the much larger vocabulary size |V | or number of documents D we've seen
  - These d dimensions don't have a clear interpretation
  - The vectors are dense: instead of vector entries being sparse, mostly-zero counts or functions of counts, the values will be real-valued numbers that can be negative

Word2vec: Skip-grams

- Instead of entire documents, Word2vec uses words a few positions away from each center word. The pairs of center word/context word are called "skip-grams"

"It was **a bright cold day in April, and** the clocks were striking"

Center word: red

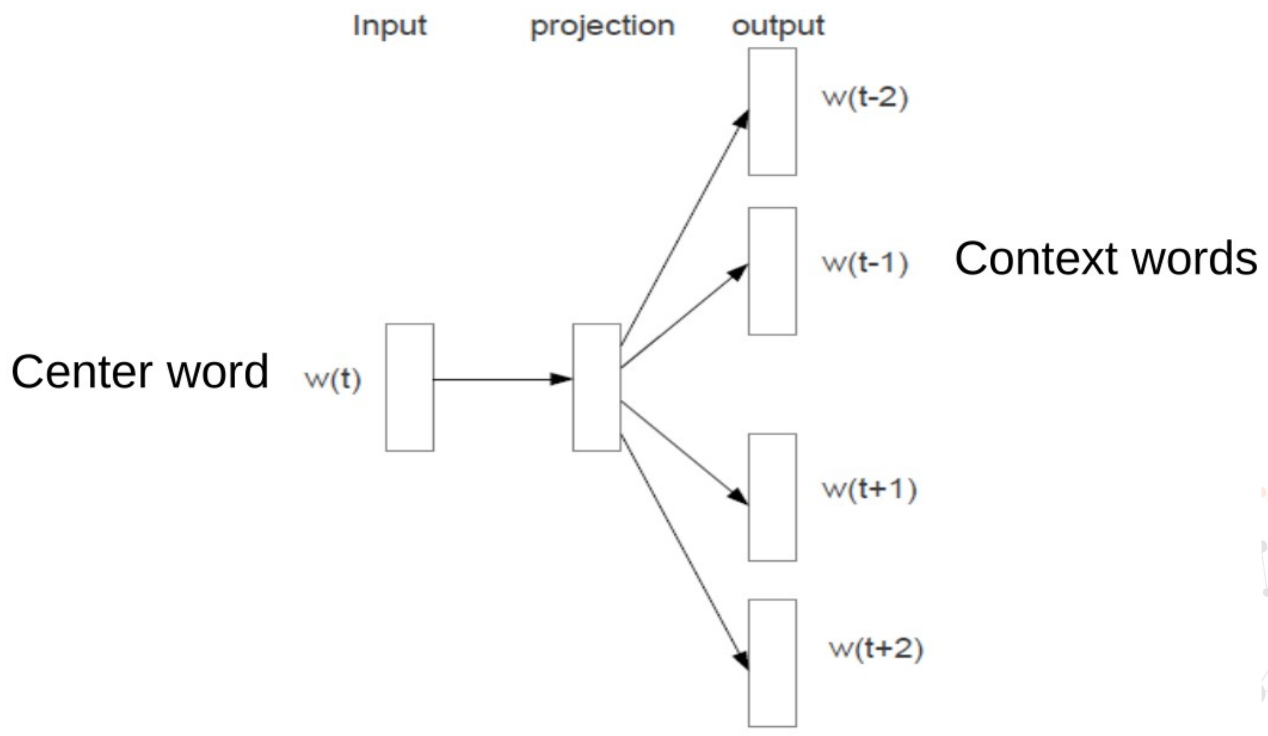
Context words: blue

- Word2vec considers all words as center words, and all their context words

Word2vec: Skip-grams

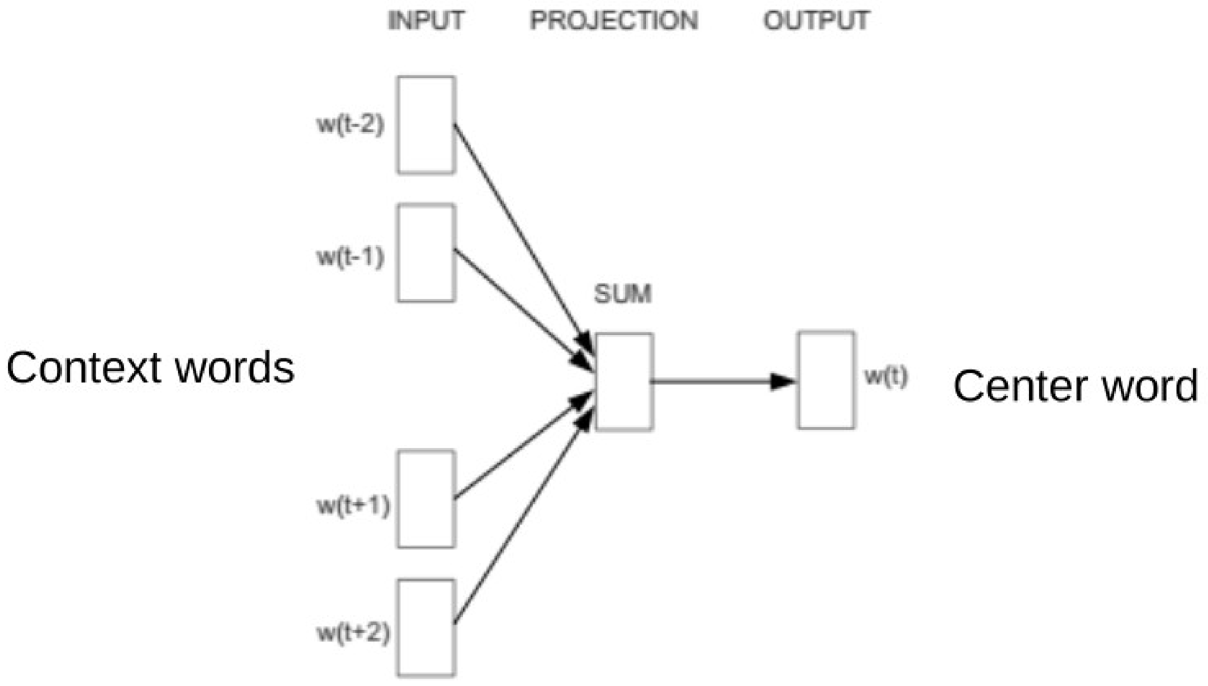
- The pairs of center word/context word are called "skip-grams"
- Typical distances are 3-5 word positions.

Skip-gram model



Word2vec: CBOW

- Models can also predict center word from context, CBOW model. Generally, skip-gram performs better



Skip-gram model

- The goal of the skip-gram model is to predict the context words given a target word.
- Specifically, it maximizes the probability of the surrounding words within a certain window size based on the current word.
- Skip-gram tends to work better when data is sparse and can learn high-quality embeddings even from large corpora

Skip-gram – Objective Function

- Given a corpus of words $\{w_1, w_2, \ldots, w_T\}$
  - the skip-gram model aims to maximize the average log probability of the context words

    $w_{t-j}, \ldots, w_{t+j}$
  - given the target word

    w_t (excluding w_t itself)
  - over all positions t in the corpus.

Skip-gram – Objective Function

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log P(w_{t+j}|w_t)$$

where

- w_t is the target word at position t,
- w_{t+j} are the context words surrounding w_t,
- c is the window size, determining how many words before and after the target word are considered as context,
- T is the total number of words in the corpus.

Skip-gram – Objective Function

- The conditional probability $P(w_{t+j} | w_t)$ is typically modeled using a softmax function over the entire vocabulary V, parameterized by the word embeddings:

$$P(w_O|w_I) = \frac{\exp(v_{w_O}^{\top} v_{w_I})}{\sum_{w \in V} \exp(v_w^{\top} v_{w_I})}$$

- Where:
  - v_{wI} is the vector representation (embedding) of the input word w_I,
  - v_{wO} is the vector representation (embedding) of the output word w_O
  - V is the vocabulary

Word2Vec

Word2vec - **DEMO**

Thank you