Foundations of NLP

M. Tech DSc & AI

Text pre-processing
(Stop Words, Bag-of-Words, TF-IDF, POS Tagging, NER)

Mahindra™ University
Global Thinkers. Engaged Leaders.

Acknowledgments

These slides were adapted from the book

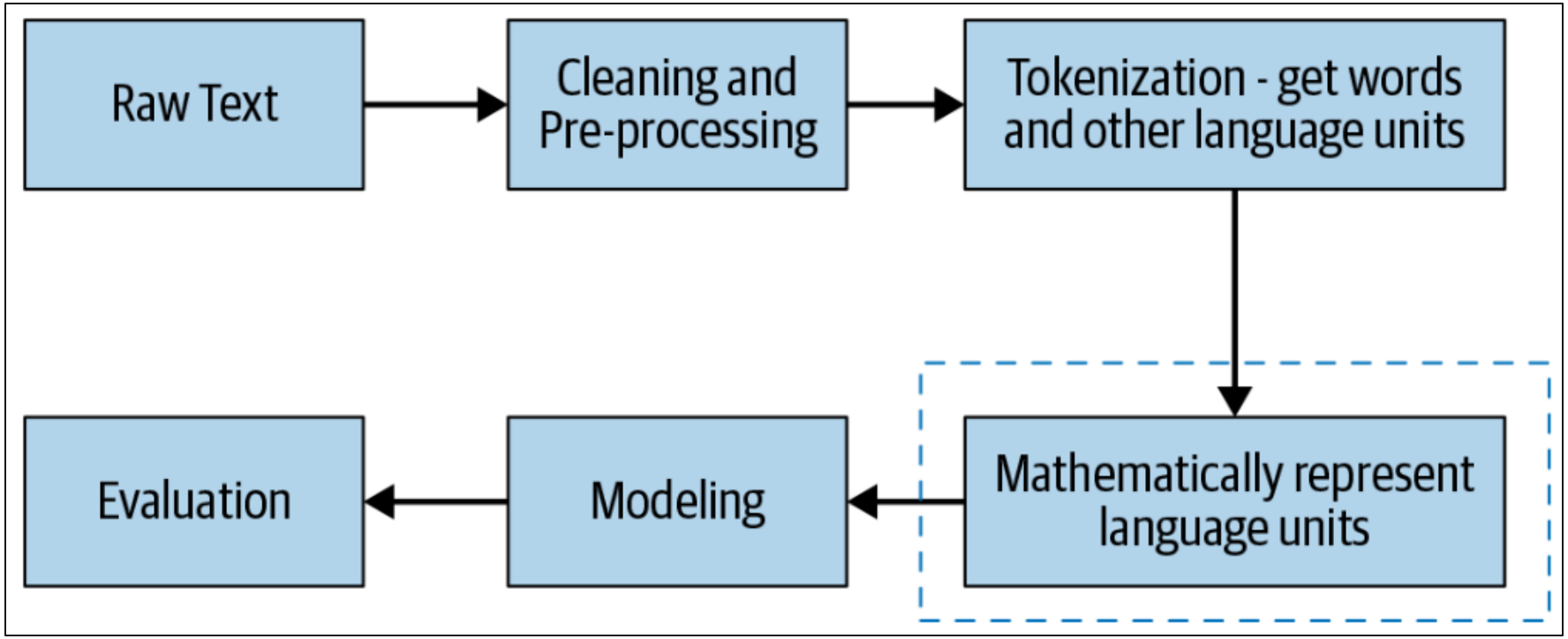
SPEECH and LANGUAGE PROCESSING: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition and

Some modifications from presentations and resources found in the WEB by several scholars.

Recap

- NLP
- Applications
- Regular expressions
- Hands-on assignment (Regex)
- Tokenization
- Hands-on assignment (Tokenization)
- Case folding
- Stemming
  - Porter Stemmer
- Hands-on assignment (Porter Stemmer)
- Lemmatization
- Normalization

NLP pipeline



Source: Practical Natural Language Processing- Oreilly

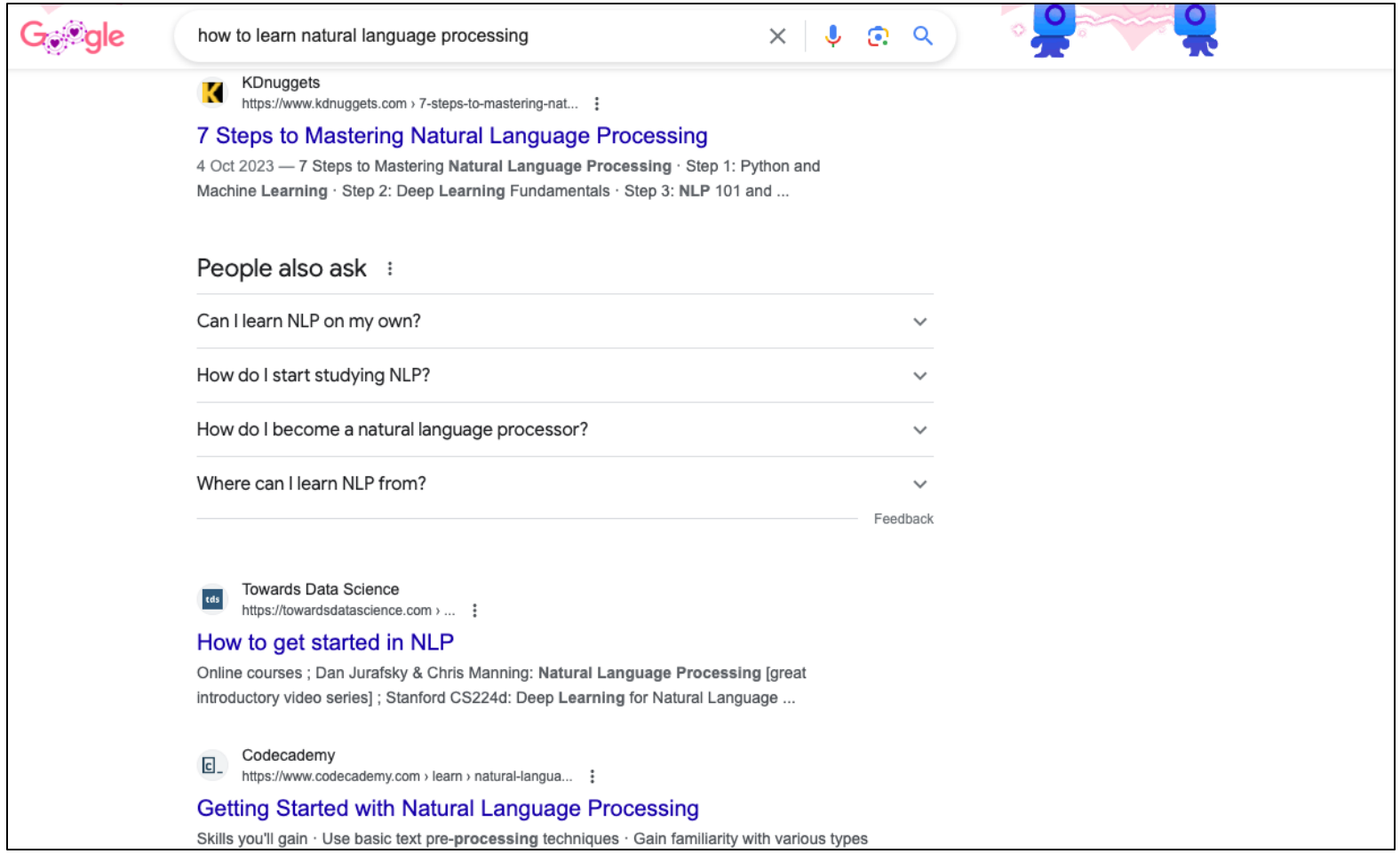
Stop words

- Stopwords are basically list of commonly used words in any language not just english!!
- Most common words in any language (like articles, prepositions, pronouns, conjunctions, etc.) and does not add much information to the text.
- Examples of a few stop words in English are **“the”, “a”, “an”, “so”, “what”, etc.**

Different types of stop words

- Determiners- the, a, an, another, etc.
- Coordinating conjunctions- for, an, nor, but, or, yet, so, etc.
- Prepositions- in, under, towards, before, etc.
- Many more…..(depends on application)

Range of Applications (Ex: Search engines)



Natural language Processing

How

To

Learn

In-class activity

Domain-specific stop word list construction?

https://sparknlp.org/2020/07/14/stopwords_hi.html

https://github.com/WorldBrain/remove-stopwords

- Find out stop words list for indian languages?
- For instance, https://github.com/Xangis/extra-stopwords/blob/master/telugu

Stopword removal

- Critical to remove them to focus on important words

- Be careful (Choose your own Stop word list)
  - Removing words like "good", "fair", "worst" in applications such as sentiment analysis -----> Not to be done

How to find important/relevant words?

- Bag of Words
- One-hot encoding
- TF-IDF
- POS-tagging
- Named Entity Recognition
- Dependency parsing
- Topic modeling
- Word Embeddings

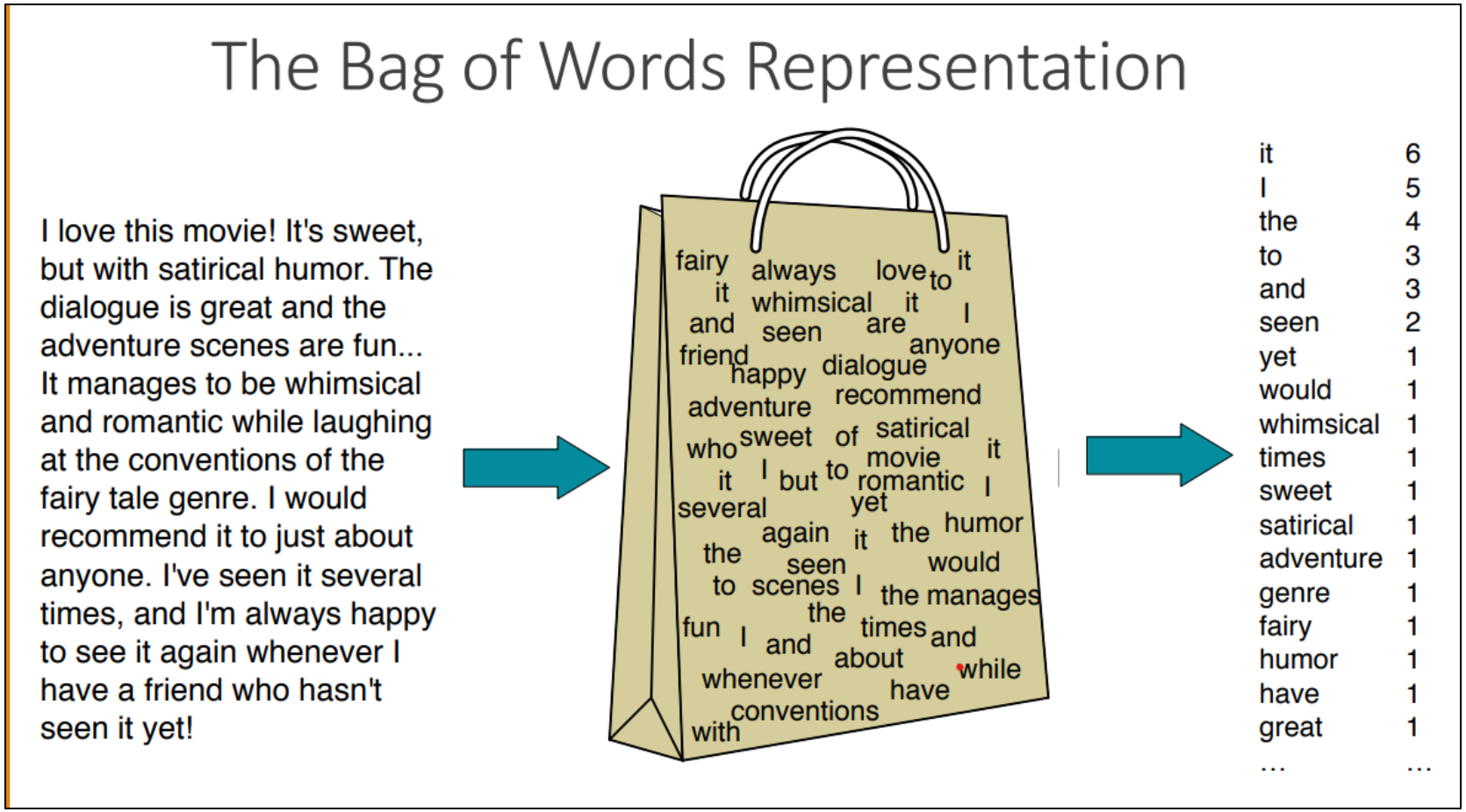
How to represent words?

Neural Networks which are part of Machine Learning models require their input in tensors or vectors whose constituent elements are in numerical form.

So how is the data present in the form of text fed as input to such a neural network model?

- Bag of Words
- One-hot encoding
- TF-IDF

Bag of Words


The Bag of Words Representation
I love this movie! It's sweet, but with satirical humor. The dialogue is great and the adventure scenes are fun... It manages to be whimsical and romantic while laughing at the conventions of the fairy tale genre. I would recommend it to just about anyone. I've seen it several times, and I'm always happy to see it again whenever I have a friend who hasn't seen it yet!
fairy always love to it it whimsical it I and seen are anyone friend happy dialogue adventure recommend who sweet of satirical movie it it I but to romantic I several yet again it the humor the seen would to scenes I the manages fun I and the times and about while whenever have conventions with
it 6
I 5
the 4
to 3
and 3
seen 2
yet 1
would 1
whimsical 1
times 1
sweet 1
satirical 1
adventure 1
genre 1
fairy 1
humor 1
have 1
great 1
… …

Bag of Words

- Based on word frequency- Count of number of words
- If two pieces of text have nearly the same words, then they belong to the same bag (class)

Document	**Content**
D1	Dog bites man.
D2	Man bites dog.
D3	Dog eats meat.
D4	Man eats food.

Question?

- Create a bag of words for toy corpus D1?
- Create a bag of words for toy corpus D2?
- Create a bag of words for toy corpus D3?
- Create a bag of words for toy corpus D4?

Solution (D1 vector using Bag-of-words)

dog = 1

bites = 2

man = 3

meat = 4

food = 5

eats = 6

Here, 1 to 6 are array indexes

D1 -> [1 1 1 0 0 0]

Advantages

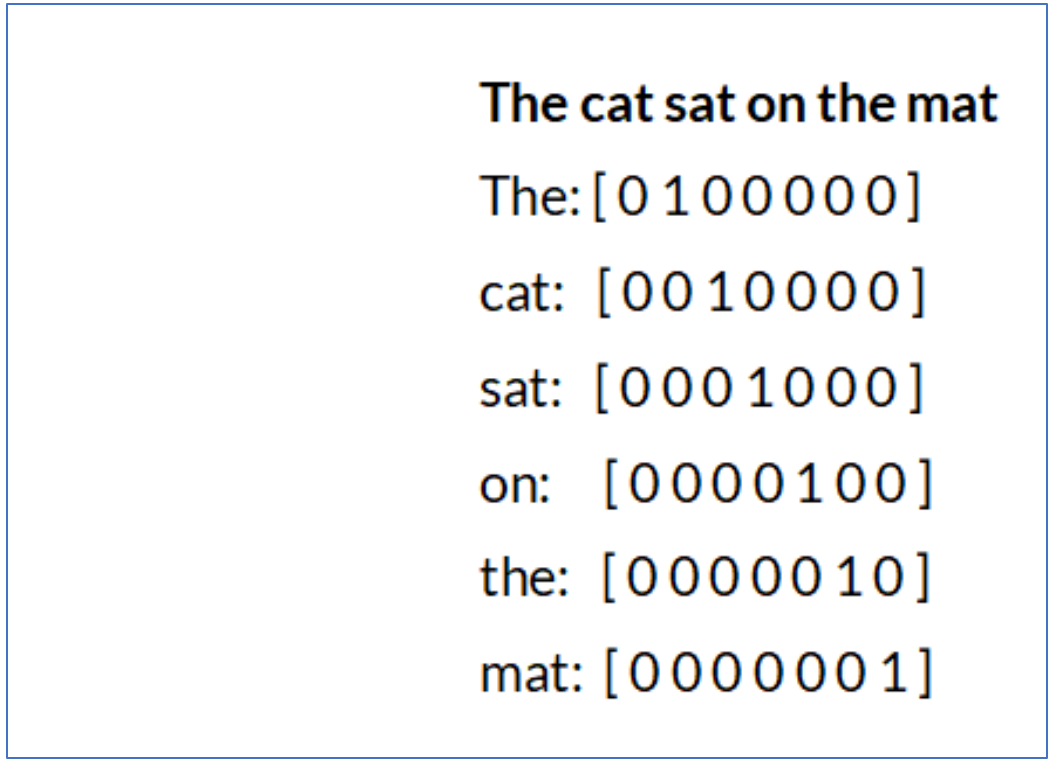
- Simple and easy to understand
- If two documents have similar vocabulary, they’ll be closer to each other in the vector space and vice versa.
- Fixed length encoding

Disadvantages

- The size of the vector increases with the size of the vocabulary. Thus, sparsity continues to be a problem. One way to control it is by limiting the vocabulary to n number of the most frequent words.
- It does not capture the similarity between different words that mean the same thing. Say we have three documents: “I run”, “I ran”, and “I ate”. BoW vectors of all three documents will be equally apart.
- Cannot handle out of vocabulary words
- Order and context information lost

One hot encoding

- Each word is written or encoded as one hot vector, with each one hot vector being unique.



TF-IDF

- TF-IDF is a numerical statistic that reflects the importance of a word in a document. It is commonly used in NLP to represent the relevance of a term to a document or a corpus of documents.
- The TF-IDF algorithm takes into account two main factors:
  - the frequency of a word in a document, Term Frequency (TF) and
  - the frequency of the word across all documents in the corpus, Inverse Document Frequency (IDF).

Term Frequency (TF)

- The term frequency (TF) is a measure of how frequently a term appears in a document.

$$tf_{t,d} = \frac{\text{frequency of term 't' in document 'd'}}{\text{total terms in document 'd'}}$$

Formula: tf(t,d) = count of t in d / number of words in d

Inverse Document Frequency (IDF)

- The inverse document frequency (IDF) is a measure of how important a term is across all documents in the corpus.

$$IDF(w) = log(\frac{Number\ of\ documents}{Number\ of\ documents\ with\ word\ w})$$

  - The resulting value is a number greater than or equal to 0.

Different weighting schemes for IDF

weighting scheme	idf weight ($n_t = \left\vert \{d \in D : t \in d\} \right\vert$)
unary	1
inverse document frequency	$\log \frac{N}{n_t} = -\log \frac{n_t}{N}$
inverse document frequency smooth	$\log \left(\frac{N}{1+n_t} \right) + 1$
inverse document frequency max	$\log \left(\frac{\max_{\{t' \in d\}} n_{t'}}{1+n_t} \right)$
probabilistic inverse document frequency	$\log \frac{N-n_t}{n_t}$

Example

```
Doc1: The quick brown fox jumps over the lazy dog
Doc2: The lazy dog likes to sleep all day
Doc3: The brown fox prefers to eat cheese
Doc4: The red fox jumps over the brown fox
Doc5: The brown dog chases the fox
```

Now, let's say we want to calculate the TF-IDF scores for the word "fox" in each of these documents.

Step 1: Calculate the term frequency (TF)

- TF = (Number of times word appears in the document) / (Total number of words in the document)

Word -Fox

```
Doc1: The quick brown fox jumps over the lazy dog
Doc2: The lazy dog likes to sleep all day
Doc3: The brown fox prefers to eat cheese
Doc4: The red fox jumps over the brown fox
Doc5: The brown dog chases the fox
```

```
Doc1: 1 / 9
Doc2: 0 / 8
Doc3: 1 / 7
Doc4: 2 / 8
Doc5: 1 / 6
```

Step 2: Calculate the document frequency (DF)

- The document frequency (DF) is the number of documents in the corpus that contain the word. We can calculate the DF for the word "**fox**" as follows:

```
Doc1: The quick brown fox jumps over the lazy dog
Doc2: The lazy dog likes to sleep all day
Doc3: The brown fox prefers to eat cheese
Doc4: The red fox jumps over the brown fox
Doc5: The brown dog chases the fox
```

```
DF = 4 (Doc1, Doc3, Doc4 and Doc5)
```

Step 3: Calculate the inverse document frequency (IDF)

- The inverse document frequency (IDF) is a measure of how rare the word is across the corpus. It is calculated as the logarithm of the total number of documents in the corpus divided by the document frequency. In our case, we have:

```
Doc1: The quick brown fox jumps over the lazy dog
Doc2: The lazy dog likes to sleep all day
Doc3: The brown fox prefers to eat cheese
Doc4: The red fox jumps over the brown fox
Doc5: The brown dog chases the fox
```

```
IDF = log(5/4) = 0.2231
```

Step 4: Calculate the TF-IDF score

- The TF-IDF score for the word “fox” in each document can now be calculated using the following formula:

```
Doc1: The quick brown fox jumps over the lazy dog
Doc2: The lazy dog likes to sleep all day
Doc3: The brown fox prefers to eat cheese
Doc4: The red fox jumps over the brown fox
Doc5: The brown dog chases the fox
```

```
TF-IDF = TF * IDF

Doc1: 1/9 * 0.2231 = 0.0247
Doc2: 0/8 * 0.2231 = 0
Doc3: 1/7 * 0.2231 = 0.0318
Doc4: 2/8 * 0.2231 = 0.0557
Doc5: 1/6 * 0.2231 = 0.0372
```

Question?

Document	**Content**
D1	Dog bites man.
D2	Man bites dog.
D3	Dog eats meat.
D4	Man eats food.

Vector Space models

Textual Data: Vector Space Model and TF-IDF

Libraries in Python for TF-IDF

sklearn library has inbuilt classes like Tfidfvectorizer, TfidfTransformer, CountVectorizer to calculate tfidf:

CountVectorizer — Converts a collection of text documents to a matrix of token counts

TfidfVectorizer — Convert a collection of raw documents to a matrix of TF-IDF features

TfidfTransformer — Transform a count matrix to a normalized tf-idf representation

Code Implementation:

In-class activity to replace the TF-IDF function with (TF-IIDF from scratch)

```
import nltk
nltk.download('punkt')
nltk.download('stopwords')
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import re
from sklearn.feature_extraction.text import TfidfVectorizer

# Sample corpus of documents
corpus = ['The quick brown fox jumps over the lazy dog.',
          'The lazy dog likes to sleep all day.',
          'The brown fox prefers to eat cheese.',
          'The red fox jumps over the brown fox.',
          'The brown dog chases the fox'
         ]

# Define a function to preprocess the text
def preprocess_text(text):
    # Remove punctuation and other non-alphanumeric characters
    text =  re.sub('[^a-zA-Z]', ' ', text)
    # Tokenize the text into words
    words = word_tokenize(text.lower())
    # Remove stop words
    words = [word for word in words if word not in stopwords.words('english')]
    # Join the words back into a string
    return ' '.join(words)

# Preprocess the corpus
corpus = [preprocess_text(doc) for doc in corpus]
print('Corpus: \n{}'.format(corpus))

# Create a TfidfVectorizer object and fit it to the preprocessed corpus
vectorizer = TfidfVectorizer()
vectorizer.fit(corpus)

# Transform the preprocessed corpus into a TF-IDF matrix
tf_idf_matrix = vectorizer.transform(corpus)

# Get list of feature names that correspond to the columns in the TF-IDF matrix
print("Feature Names:\n", vectorizer.get_feature_names_out())

# Print the resulting matrix
print("TF-IDF Matrix:\n",tf_idf_matrix.toarray())
```

Class Homework/Implementation

- Let’s go into Colab and find the Dr. Seuss book that is most similar to One Fish, Two Fish, Red Fish, Blue Fish.

- Vector Space Model and TF-IDF -Code/Notebook

Advantages of TF-IDF

- **Measures relevance:** TF-IDF measures the importance of a term in a document, based on the frequency. This helps to identify which terms are most relevant to a particular document.

Advantages of TF-IDF

- **Measures relevance:** TF-IDF measures the importance of a term in a document, based on the frequency. This helps to identify which terms are most relevant to a particular document.
- **Handles large text corpora**: TF-IDF is scalable making it suitable for processing and analyzing large amounts of text data.
- **Handles stop words**: TF-IDF automatically down-weights common words that occur frequently in the text corpus making it a more accurate measure of term importance.
- **Application**s: TF-IDF can be used for various natural language processing tasks, such as text classification, information retrieval, and document clustering.
- **Interpretable**: The scores generated by TF-IDF are easy to interpret and understand.
- **Many languages**: Works well with different languages.

Limitations of TF-IDF

- **Ignores the context:** TF-IDF only considers the frequency of each term and does not take into account the context in which the term appears. This can lead to incorrect interpretations of the meaning of the document.
- **Assumes independence**: TF-IDF assumes that the terms in a document are independent of each other. However, this is often not the case in natural language, where words are often related to each other in complex ways.
- **No concept of word order:** TF-IDF treats all words as equally important, regardless of their order or position in the document. This can be problematic for certain applications, such as sentiment analysis, where word order can be crucial for determining the sentiment of a document.
- **Limited to term frequency**: TF-IDF only considers the frequency of each term in a document and does not take into account other important features, such as the length of the document or the position of the term within the document.

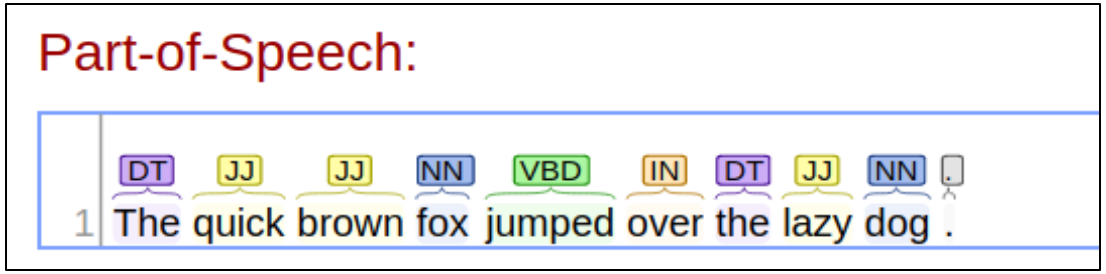
Applications of TF-IDF

- **Text vectorization:** TF-IDF is used to represent the text as a vector.
- **Search engines:** Rank documents based on their relevance to a query.
- **Text classification**: To identify the most important features in a document.
- **Information extraction**: To Identify the most important entities and concepts in a document.
- **Keyword extraction**: To identify the most important keywords in a document.
- **Recommender systems**: To recommend items to users based on their preferences.
- **Sentiment analysis**: To identify the most important words in a document that contribute to the sentiment.

POS-tagging

- Part-of-Speech (POS) tagging involves assigning specific grammatical categories or labels (such as nouns, verbs, adjectives, adverbs, pronouns, etc.) to individual words within a sentence.
- Advantages:
  - Provides insights into the syntactic structure of the text
  - Aiding in understanding word relationships
  - Disambiguating word meanings
  - Facilitating various linguistic and computational analyses of textual data

Demo - CoreNLP (stanfordnlp.github.io)



Types of POS tags

- Universal POS Tags
- Detailed POS Tags

Tag	Description
ADJ	Adjective
ADV	Adposition
ADP	Adverb
AUX	Auxiliary
CCONJ	Coordinating Conjunction
DET	Determiner
INTJ	Interjection
NOUN	Noun
NUM	Numeral
PART	Particle
PRON	Pronoun
PROPN	Proper Noun
PUNCT	Punctuation
SCONJ	Subordinating Conjunction
SYM	Symbol
VERB	Verb
X	Other

Detailed POS Tags

These tags are the result of the division of universal POS tags into various tags, like NNS for common plural nouns and NN for the singular common noun compared to NOUN for common nouns in English. These tags are language-specific.

```
It => PRON => PRP
took => VERB => VBD
me => PRON => PRP
more => ADJ => JJR
than => SCONJ => IN
two => NUM => CD
hours => NOUN => NNS
to => PART => TO
translate => VERB => VB
a => DET => DT
few => ADJ => JJ
pages => NOUN => NNS
of => ADP => IN
English => PROPN => NNP
. => PUNCT => .
```

```
1  import spacy
2  nlp=spacy.load('en_core_web_sm')
3
4  text='It took me more than two hours to translate a few pages of English.'
5
6  for token in nlp(text):
7   print(token.text, '=>',token.pos_,'=>',token.tag_)
```

In the above code sample, Load **spacy's en_web_core_sm** model and used it to get the POS tags. You can see that the **pos_ returns** the universal POS tags, and **tag_ returns** detailed POS tags for words in the sentence.

Named entity recognition

Demo

displaCy Named Entity Visualizer · Explosion

Class Projects /NLP

- Group size allowed (4-5) teams
- Pick a project , (1+1) mid evaluations, Final project presentation (1)
- Real-world problems (Look around for NLP problems)

  Do not restrict yourselves to sentiment analysis, recommender systems, searching, etc.

  Project topic and Team – Deadline – 10 sep, 2024 6:00 AM

  Sheet link has been provided on slack

Reference materials

- https://vlanc-lab.github.io/mu-nlp-course/teachings/fall-2024-AI-nlp.html

- Lecture notes

- (A) Speech and Language Processing by Daniel Jurafsky and James H. Martin
- (B) Natural Language Processing with Python. (updated edition based on Python 3 and NLTK
- 3) Steven Bird et al. O'Reilly Media