Unit 3

Language Modelling

Contents

- Probabilistic language modelling
- Markov models
- Generative models of language
- Log Liner Models
- Graph-based Models N-gram models: Simple n-gram models
- Estimation parameters and smoothing
- Evaluating language models
- Word Embeddings/ Vector Semantics: Bag-of-words, TFIDF, word2vec, doc2vec
- Contextualized representations (BERT) Topic Modelling: Latent Dirichlet Allocation (LDA), Latent Semantic Analysis, Non Negative Matrix Factorization

Word Embedding

- Word embeddings and vector semantics are techniques used in natural language processing (NLP) **to represent text data numerically**, allowing machines to **understand the semantic meaning of words and documents**.

Bag-of-Words (BoW)

- The Bag-of-Words model represents text as a collection of words, **disregarding grammar and word order**, but keeping multiplicity.
- **Each word** in the corpus is treated as **a unique feature**, and its **frequency in a document is recorded**.
- It converts text into a "bag" of words, ignoring grammar and word order but keeping track of word frequency.

BoW Steps

1. **Tokenization** – Split the text into words.
2. **Build a Vocabulary** – Create a list of all unique words across the documents.
3. **Vectorization** – Convert each document into a vector based on word frequency.

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!

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
…	…

Tokenization

Build a Vocabulary (BoW)

Vectorization

Example:

Text 1: "I love machine learning"

Text 2: "I love AI and machine learning"

Vocabulary:

["I", "love", "machine", "learning", "AI", "and"]

Vector Representation:

- Text 1: [1, 1, 1, 1, 0, 0] (corresponding to the presence of the words in the vocabulary)
- Text 2: [1, 1, 1, 1, 1, 1]

- Instead of just marking presence (1 or 0), we can count how many times each word appears……i.e Frequency of each word

Consider these three sentences:

1. "The sun is bright."
2. "The sun in the sky is bright."
3. "We can see the shining sun, the bright sun.“

Vocabulary: ["the", "sun", "is", "bright", "in", "sky", "we", "can", "see", "shining"]

Word	The	Sun	Is	Bright	In	Sky	We	Can	See	Shining
S1	1	1	1	1	0	0	0	0	0	0
S2	1	1	1	1	1	1	0	0	0	0
S3	2	2	0	1	0	0	1	1	1	1

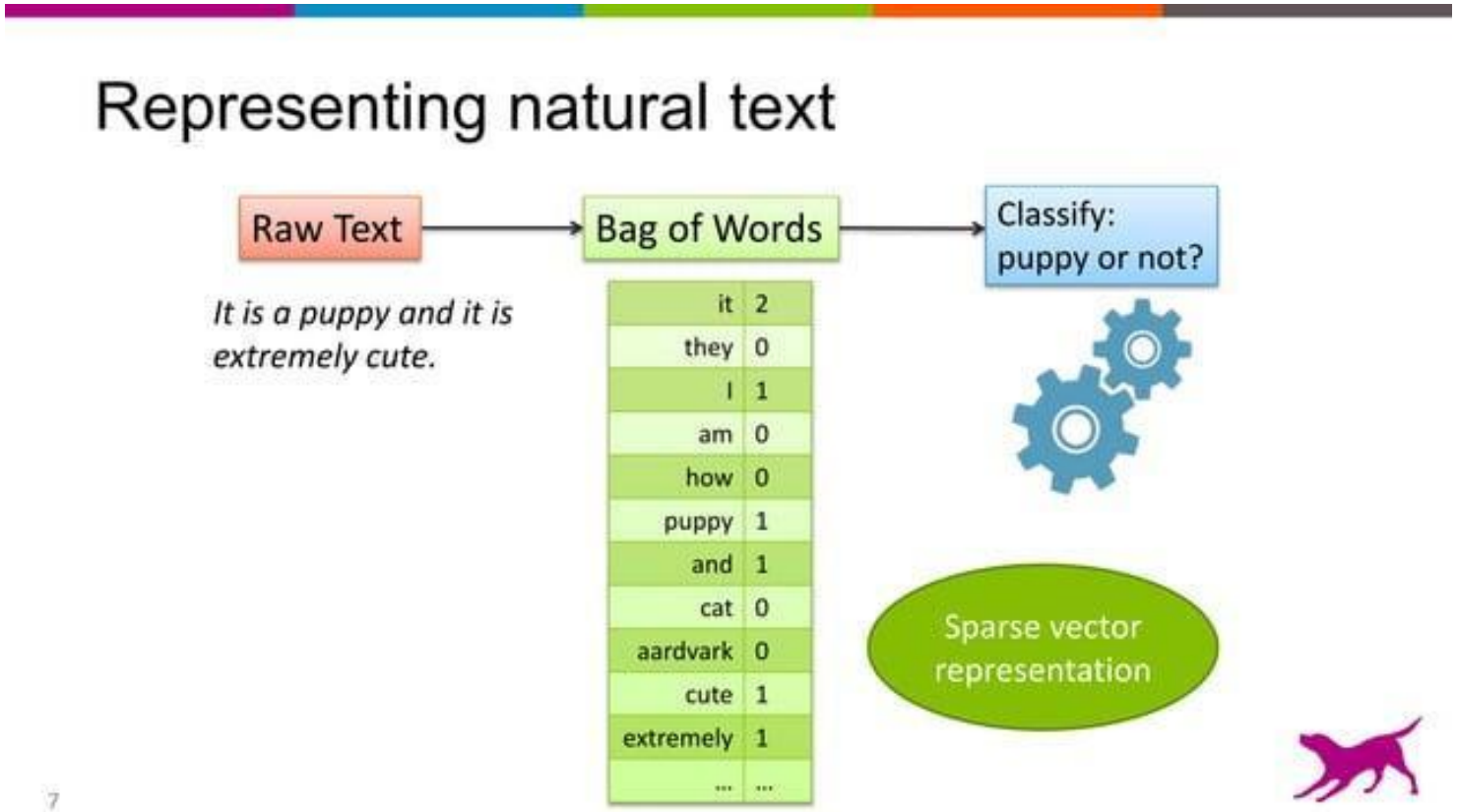
Here, each number represents how many times a word appears in the sentence.

Limitations of BoW

1. **Ignores Word Order:** "Yoga heals life" and "Life heals yoga" get the same representation.
2. **Sparsity:** Large vocabularies create big, sparse vectors.
3. **Ignores Meaning:** It does not consider context or semantics.

To overcome these issues, techniques like **TF-IDF** (Term Frequency-Inverse Document Frequency) and **Word Embeddings (Word2Vec, BERT, etc.)** are used.

Application



Example: Sparse BoW Matrix

Consider Three Short Sentences

"Yoga brings peace."

"Meditation calms the mind."

"AI transforms the future.“

Step 1: Create Vocabulary

Unique words:

["yoga", "brings", "peace", "meditation", "calms", "the", "mind", "AI", "transforms", "future"]

Step 2: Construct BoW Matrix

Word	Yoga	Brings	Peace	Meditation	Calms	The	Mind	AI	Transforms	Future
Sentence 1	1	1	1	0	0	0	0	0	0	0
Sentence 2	0	0	0	1	1	1	1	0	0	0
Sentence 3	0	0	0	0	0	1	0	1	1	1

Observations on Sparsity

- **Most values are 0** since each sentence uses only a few words from the vocabulary.
- **Large Vocabulary = More Sparse Matrix**: If we had thousands of words and short sentences, the table would be filled mostly with zeroes.
- **Computational Inefficiency**: Sparse matrices **require more memory** and slow down computations.

How to Handle Sparsity?

- **TF-IDF**: Reduces importance of frequent words.
- **Word Embeddings (Word2Vec, BERT)**: Represent words in dense vectors with context awareness.
- **Dimensionality Reduction (PCA, LSA)**: Compresses sparse vectors.

TF-IDF (Term Frequency - Inverse Document Frequency)

- TF-IDF adjusts the **frequency of words based on their importance** in a specific document relative to the whole corpus. It is a **statistical measure** used to evaluate **how important a word is** to a document in a collection or corpus.
- **Term Frequency (TF):** Measures **how frequently a word occurs** in a document.
- **Inverse Document Frequency (IDF):** Measures how important a word is across the entire corpus (the **more common a word** is across documents, the **less weight** it carries).

Formula:

- TF = (Number of times term t appears in a document) / (Total number of terms in the document)
- IDF = log(Total number of documents / Number of documents containing term t)
- TF-IDF = TF * IDF

Example:

Document 1: "Machine learning is fun"

Document 2: "AI and machine learning are amazing"

If the word "**machine" appears frequently** in both documents, **its TF-IDF score will be low**, while a **rare word like "amazing**" will have a **higher TF-IDF score**.

Example Comparison BoW & TF-IDF

- **Consider Three Documents**

1. **D1**: "Yoga brings peace and health."
2. **D2**: "Meditation brings health benefits."
3. **D3**: "Peace of mind comes from meditation.“

Step 1: Create Vocabulary

Unique words: ["yoga", "brings", "peace", "and", "health", "meditation", "benefits", "mind", "comes", "from", "of"]

Step 2: BoW Representation

BoW Representation

Word	D1	D2	D3
yoga	1	0	0
brings	1	1	0
peace	1	0	1
and	1	0	0
health	1	1	0
meditation	0	1	1
benefits	0	1	0
mind	0	0	1
comes	0	0	1
from	0	0	1
of	0	0	1

Here, BoW just counts word occurrences, ignoring importance.

TF-IDF adjusts word importance based on:

- **Term Frequency (TF)** → How often a word appears in a document.
- **Inverse Document Frequency (IDF)** → How rare a word is across all documents.
- TF-IDF = TF * IDF

The TF-IDF values for each document might look like this (simplified):

Word	D1	D2	D3
yoga	0.69	0	0
brings	0.23	0.23	0
peace	0.34	0	0.34
and	0.69	0	0
health	0.23	0.23	0
meditation	0	0.34	0.34
benefits	0	0.69	0
mind	0	0	0.69
comes	0	0	0.69
from	0	0	0.69
of	0	0	0.69

Key Differences

Feature	Bag of Words (BoW)	TF-IDF
Definition	Counts word occurrences in a document.	Adjusts word importance based on frequency across documents.
Focus	Word presence/frequency.	Importance of words in a document relative to a collection.
Handles Common Words	Treats all words equally (e.g., "**the**" and “**Language**" have the same weight).	Reduces weight of common words and increases weight of rare words.
Interpretability	Easy to understand.	Requires **more processing** but better for meaning extraction.
Sparsity	High (large matrices with many zeroes).	Lower than BoW but still sparse.
Context Awareness	Ignores word order and meaning.	Partially considers word importance across documents.

When to Use BoW vs. TF-IDF

Scenario	Use **BoW**	Use **TF-IDF**
Simple models or feature extraction	✅	❌
Large datasets where word meaning isn't key	✅	❌
When word importance matters	❌	✅
When filtering out common words is necessary	❌	✅
Search engines or recommendation systems	❌	✅

word2vec

- Word2Vec is a **neural network-based technique** that **transforms words into dense vectors (embeddings),** where similar words are mapped to nearby points in the vector space.

It **uses two main models**:

- **Continuous Bag-of-Words (CBOW):** Predicts a word based on its context.
- **Skip-Gram:** Predicts the context based on a word.

Example:

- Words like "king" and "queen" will be closer in the word2vec space because they share similar semantic meaning related to royalty.
- Similarly, "man" and "woman" would be close.

Example of word2vec:

Suppose we have the following sentence:
"The king loves to rule the kingdom."

In this case:

Context words: "The", "loves", "to", "rule", "the", "kingdom"

Target word: "king"

If we use **Skip-Gram** (which **predicts context from the target word**), we would try to predict the words surrounding "king", such as "the", "loves", "to", "rule", and "kingdom".

Through multiple examples like this from a large corpus, word2vec learns vector representations (embeddings) of the words, where similar words are close together in the vector space.

For example, after training word2vec, the words "king" and "queen" might end up with the following vectors:

- **king**: [0.23, -0.12, 0.45, ...]
- **queen**: [0.21, -0.10, 0.47, ...]

Notice that the **vectors for "king" and "queen" are very similar**, which reflects the fact that they are related in meaning (both are related to **royalty**).

In this vector space, relationships between words can also be represented mathematically:

$$\textbf{king} - \textbf{man} + \textbf{woman} \approx \textbf{queen}$$

- This is one of the key features of word2vec: **it captures semantic relationships**, such as **gender (king vs queen) or pluralization (dog vs dogs)**, using vector arithmetic.

doc2vec

- Doc2Vec is an **extension of word2vec**, designed for representing entire documents instead of individual words.
- It learns a fixed-length vector for each document in addition to the word vectors.

Example:

- Given two documents:
  - Document 1: "Machine learning is great."
  - Document 2: "AI and machine learning are future technologies.“

- Doc2Vec will learn a vector representation for each document, capturing the semantic meaning of the entire document, not just individual words.
- This is useful for tasks like **document classification or clustering**.

Use Case:

- **Document Classification:** If you want to **classify documents** (e.g., news articles) into categories like "Sports", "Politics", or "Technology", doc2vec allows you to create document-level embeddings that capture the overall content and semantic meaning. You can then use these embeddings for training a classifier.
- For example, **after training doc2vec on a large corpus of news articles**, you might get the following vector representations:

Document 1 (Technology article): [0.50, -0.30, 0.60, ...]

Document 2 (Technology article): [0.55, -0.25, 0.62, ...]

Document 3 (Politics article): [-0.70, 0.40, -0.50, ...]

- Since **Document 1 and Document 2** are about technology, their vectors are close to each other, while **Document 3 (a politics article) has a vector that is far from the others.**

Contextualized Representations: BERT (Bidirectional Encoder Representations from Transformers)

- Unlike traditional methods like TF-IDF or word embeddings (Word2Vec, GloVe), BERT provides **contextualized word representations**, meaning that the representation of a word changes based on the surrounding words.

- This is achieved through the **Transformer architecture**, which uses self-attention to capture long-range dependencies in text.

Example:

- Consider the word **"bank"** in two sentences:

1. *"I deposited money in the bank."*
2. *"I sat by the river bank.“*

- Traditional word embeddings (e.g., Word2Vec) would assign the same vector representation to "bank" in both cases.

- However, BERT understands the different contexts and generates different embeddings for "bank" in each sentence.

Application:

BERT can be used for text classification, question answering, named entity recognition, and semantic search by leveraging deep contextual understanding.

Topic Modeling

Topic modeling is an **unsupervised learning technique** used to discover hidden topics in a collection of texts. The most common methods include:

a) Latent Dirichlet Allocation (LDA)

LDA is a probabilistic model that **assumes each document is a mixture of topics** and each topic is a mixture of words. It uses **Dirichlet priors** to assign words to topics probabilistically.

Example:

Suppose we have three documents:

1. *"Yoga improves mental health and reduces stress."*
2. *"AI is transforming the future of technology."*
3. *"Meditation and breathing exercises promote relaxation."*

LDA might identify:

- **Topic 1 (Well-being):** yoga, meditation, stress, relaxation
- **Topic 2 (Technology):** AI, future, technology

Each document is assigned a probability distribution over topics, e.g., Document 1 might be 80% about **Well-being** and 20% about **Technology**.

Application:

- Organizing research papers by topic
- Content recommendation systems

Latent Semantic Analysis (LSA)

LSA uses **Singular Value Decomposition (SVD)** to reduce the dimensionality of a term-document matrix and find latent structures in text.

Unlike LDA, LSA is a **linear algebra-based approach** that captures semantic similarities between words.

Example:

Given a document-term matrix, SVD helps in clustering similar words. For example, words like **"meditation"**, **"yoga"**, and **"relaxation"** might appear in similar contexts and be grouped under the same topic.

Application:

- Information retrieval (e.g., search engines)
- Document similarity measurement
- Topic modelling
- Semantic analysis

The Mathematical Foundation of LSA

- LSA is based on **Singular Value Decomposition (SVD)**, a linear algebra technique. The steps involved are:

Step 1: Creating a Term-Document Matrix (TDM)

- Construct a matrix **A** where:
  - Rows represent **terms (words)**.
  - Columns represent **documents**.
  - Each cell contains the frequency of a word in a document (often weighted using TF-IDF).

Applying Singular Value Decomposition (SVD)

- SVD decomposes the matrix **A** into three smaller matrices:

$$A = U \cdot S \cdot V^T$$

where:

- **U**: A matrix containing **word-topic relationships** (left singular vectors).
- **S**: A **diagonal matrix** containing singular values, representing the importance of each topic.
- **V^T**: A matrix containing **document-topic relationships** (right singular vectors).

Step 3: Dimensionality Reduction

By keeping only the **top k singular values** (choosing a smaller rank approximation of A), we remove noise and retain the most important **semantic features**. The new matrix Ak represents a **lower-dimensional representation** of the original data.

Need?

- Traditional methods like **Term Frequency-Inverse Document Frequency (TF-IDF)** struggle with synonymy and polysemy:

Synonymy: Different words can have the same meaning (e.g., "car" and "automobile").

Polysemy: A single word can have multiple meanings depending on context (e.g., "bank" as a financial institution vs. riverbank).

- LSA overcomes these issues by identifying **latent (hidden) relationships** between words and documents, rather than relying solely on direct word occurrences.

✅ Strengths of LSA:

- Captures latent meanings in text.
- Reduces noise and improves generalization.
- Useful in applications where word meanings are important.

❌ Weaknesses:

- **Computationally expensive** (SVD is slow for large datasets).
- **Cannot handle polysemy completely** (words with multiple meanings still have one fixed representation).
- **Context loss**: Unlike deep learning methods, it does not consider word order or deep contextual meaning.

```python
from gensim import corpora, models
from pprint import pprint

# Sample documents
documents = [
"I love natural language processing and machine learning.",
"Topic modeling is an interesting technique in NLP.",
"Python programming language is widely used for NLP tasks.",
"Machine learning algorithms play a crucial role in NLP applications."
]

# Tokenize the documents
tokenized_docs = [doc.split() for doc in documents]

# Create a dictionary representation of the documents
dictionary = corpora.Dictionary(tokenized_docs)

# Create a bag-of-words corpus
corpus = [dictionary.doc2bow(doc) for doc in tokenized_docs]

# Build the TF-IDF model
tfidf_model = models.TfidfModel(corpus)
corpus_tfidf = tfidf_model[corpus]

# Build the LSA model
lsi_model = models.LsiModel(corpus_tfidf, id2word=dictionary, num_topics=2)
```

```python
# Print the topics
pprint(lsi_model.print_topics())

# Transform the documents to the LSA space
corpus_lsi = lsi_model[corpus_tfidf]

# Print the document similarities in LSA space
from gensim import corpora, models
from pprint import pprint
```

```
[(0,
  '0.229*"Python" + 0.229*"programming" + 0.229*"widely" + 0.229*"used" + '
  '0.229*"for" + 0.229*"tasks." + 0.215*"is" + 0.200*"modeling" + '
  '0.200*"Topic" + 0.200*"interesting"'),
 (1,
  '0.300*"love" + 0.300*"machine" + 0.300*"learning." + 0.300*"processing" + '
  '0.300*"natural" + 0.300*"and" + 0.300*"I" + 0.197*"language" + '
  '-0.141*"interesting" + -0.141*"Topic"')]
```

Non-Negative Matrix Factorization (NMF)

NMF is another dimensionality reduction technique that **factorizes the document-term matrix into two lower-rank matrices**, where all values are non-negative.

Unlike **LSA, which allows negative values** (which can be hard to interpret), NMF ensures that topics are made of **only additive combinations of words**, making it more interpretable.

Example:

For the same set of documents, NMF may extract:

Topic 1: {meditation, yoga, relaxation, breathing}

Topic 2: {AI, technology, future, machine learning}

Application:

- Extracting key topics from news articles
- Clustering customer reviews

Comparison Table

Method	Technique	Strengths	Weaknesses
BERT	Transformer-based deep learning	Captures contextual meaning	Computationally expensive
LDA	Probabilistic model (Dirichlet priors)	Good for topic distribution	Struggles with short texts
LSA	Matrix factorization (SVD)	Captures latent semantic relationships	Less interpretable than LDA
NMF	Non-negative matrix factorization	Easy to interpret	Requires careful tuning