CSCI E-89B Introduction to Natural Language Processing

Harvard Extension School

Dmitry Kurochkin

Fall 2024
Lecture 5

Contents

Contents

Introduction to TF-IDF

- **Term Frequency-Inverse Document Frequency (TF-IDF)**
  - TF-IDF is a statistical measure used to evaluate the importance of terms in documents relative to a corpus.
    - ★ **Term Frequency (TF):** Measures how often a term appears in a document. In simplest form, TF is calculated as:

$$\text{TF(term, doc)} = \frac{\text{Number of times term appears in the document}}{\text{Total number of terms in the document}}$$

    - ★ **Inverse Document Frequency (IDF):** Assesses how much information a word provides by downscaling terms that occur frequently across the corpus:

$$\text{IDF(term)} = \ln\left(\frac{\text{Total number of documents}}{\text{Number of documents containing the term}}\right)$$

  - **TF-IDF:**

$$\text{TF-IDF(term, doc)} = \text{TF(term, doc)} \times \text{IDF(term)}$$

Contents

TF-IDF Example: Manual Computation

- **Documents:**
  1. Doc 1: "cat dog cat"
  2. Doc 2: "dog mouse dog"
  3. Doc 3: "dog mouse"
  4. Doc 4: "mouse cat dog"
- **Vocabulary:** {"cat", "dog", "mouse"}
- **Term Frequency (TF):**
  - TF(cat): Doc 1 $= 2/3$, Doc 2 $= 0/3$, Doc 3 $= 0/2$, Doc 4 $= 1/3$
  - TF(dog): Doc 1 $= 1/3$, Doc 2 $= 2/3$, Doc 3 $= 1/2$, Doc 4 $= 1/3$
  - TF(mouse): Doc 1 $= 0/3$, Doc 2 $= 1/3$, Doc 3 $= 1/2$, Doc 4 $= 1/3$
- **Inverse Document Frequency (IDF):**

$$\text{IDF(cat)} = \ln(4/2) = 0.693$$

$$\text{IDF(dog)} = \ln(4/4) = 0$$

$$\text{IDF(mouse)} = \ln(4/3) = 0.288$$

TF-IDF Example: Manual Computation (Continued)

- **TF-IDF Values:**
  1. Doc 1:
     - cat: $2/3 \times \ln(4/2) = 2/3 \times 0.693 \approx 0.462$
     - dog: $1/3 \times \ln(4/4) = 1/3 \times 0 = 0$
     - mouse: $0/3 \times \ln(4/3) = 0/3 \times 0.288 = 0$
  2. Doc 2:
     - cat: $0/3 \times \ln(4/2) = 0/3 \times 0.693 = 0$
     - dog: $2/3 \times \ln(4/4) = 2/3 \times 0 = 0$
     - mouse: $1/3 \times \ln(4/3) = 1/3 \times 0.288 \approx 0.096$
  3. Doc 3:
     - cat: $0/2 \times \ln(4/2) = 0/2 \times 0.693 = 0$
     - dog: $1/2 \times \ln(4/4) = 1/2 \times 0 = 0$
     - mouse: $1/2 \times \ln(4/3) = 1/2 \times 0.288 \approx 0.144$
  4. Doc 4:
     - cat: $1/3 \times \ln(4/2) = 1/3 \times 0.693 \approx 0.231$
     - dog: $1/3 \times \ln(4/4) = 1/3 \times 0 = 0$
     - mouse: $1/3 \times \ln(4/3) = 1/3 \times 0.288 \approx 0.096$

Contents

Advantages of TF-IDF over BoW

- Enhances Bag of Words (BoW) by incorporating global term significance.
  - **Limitations of BoW:** Simply counts occurrences, potentially leading common words to overshadow meaningful terms.
  - **Advantage of TF-IDF:** Adjusts term weights based on their occurrence in the corpus, offering a more comprehensive measure of term importance.

- Balances term frequency with significance across a corpus.
  - **Balancing Act:** Combines local document frequency with global significance, refining how insights are drawn.
  - **Applications:** Widely used for keyword extraction, document classification, and enhancing search engine results.
  - **Normalization:** Often employed to make TF-IDF scores comparable across documents.

Contents

Modified Versions of Term Frequency

- **Raw Term Count:**
  - $f(t, d)$ represents the raw count of term t occurrences in document d.
  - Can be affected by document length.
- **Term Frequency (TF):**
  - $\text{TF}(t, d) = \frac{f(t,d)}{N_d}$
  - N_d is the total number of terms in document d.
  - Default method in Scikit-learn's 'TfidfVectorizer'.
- **Logarithmically Scaled TF:**
  - $\text{TF}_{\ln}(t, d) = 1 + \ln(f(t, d))$ if $f(t, d) > 0$; else 0.
  - Reduces the impact of high frequency terms.
- **Double Normalization (K):**
  - $\text{TF}_{\text{norm}}(t, d) = K + (1 - K) \times \frac{f(t,d)}{\max\{f(w,d):w\in d\}}$
  - Balances term frequencies relative to the maximum term frequency in the document.
  - K is typically 0.5.
- **Boolean Term Frequency:**
  - $\text{TF}_{\text{bool}}(t, d) = 1$ if the term appears in the document, otherwise 0.
  - Ignores actual frequency, focusing on presence.

Modified Versions of Inverse Document Frequency (IDF)

- **Standard IDF:**
  - $\text{IDF}(t) = \ln\left(\frac{N}{\text{DF}(t)}\right)$, where $\text{DF}(t)$ is the document frequency of term t.
  - High IDF for rare terms across documents.
- **Smoothed IDF:**
  - $\text{IDF}_\text{s}(t) = \ln\left(\frac{1+N}{1+\text{DF}(t)}\right) + 1$
  - Default method in Scikit-learn's 'TfidfVectorizer'.
  - Includes smoothing to prevent division by zero.
- **Probabilistic IDF (ProbIDF):**
  - $\text{IDF}_\text{prob}(t) = \ln\left(\frac{N-\text{DF}(t)}{\text{DF}(t)}\right)$
  - Considers the probability of document exclusion.
- **Maximal IDF:**
  - $\text{IDF}_\text{max}(t) = \ln\left(\frac{\max(\text{DF})}{1+\text{DF}(t)}\right)$
  - Uses maximum document presence as a benchmark.

Contents

Hands-On: TF-IDF in Python Using Sklearn

```python
from sklearn.feature_extraction.text import TfidfVectorizer

# Example text
documents = ["cat dog cat",
             "dog mouse dog",
             "cat mouse",
             "mouse cat dog"]

# Initialize TfidfVectorizer
vectorizer = TfidfVectorizer()

# Fit and Transform the documents
tfidf_matrix = vectorizer.fit_transform(documents)

# Display the Vocabulary and TF-IDF Representation
print("Vocabulary:\n", vectorizer.get_feature_names_out(), "\n")
print("TF-IDF Representation:\n", tfidf_matrix.toarray())
```

```
Vocabulary:
['cat', 'dog', 'mouse']

TF-IDF Representation:
[[0.81649658 0.40824829 0.        ]
 [0.         0.81649658 0.40824829]
 [0.70710678 0.         0.70710678]
 [0.40824829 0.40824829 0.40824829]]
```

Contents

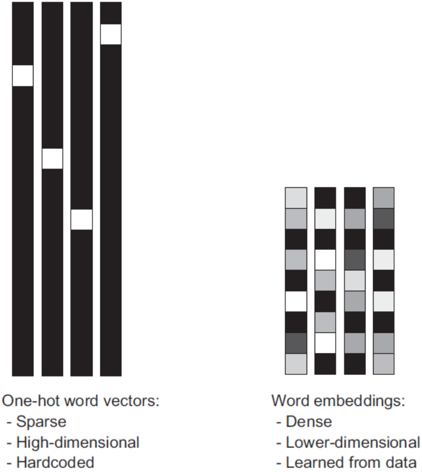
Introduction to Word Embeddings

- **Concept**:
  - Word embeddings are dense vector representations of words.
  - They encode words into numerical vectors where semantically similar words have similar representation.
  - Unlike traditional Bag of Words, embeddings capture context by considering the proximity of words in text.

- **Goal**:
  - Aim to create a high-dimensional space where relationships and meanings between words reflect their context.
  - Facilitate understanding of words in multiple dimensions, addressing ambiguity through context.
  - Enable machines to understand and process text in ways similar to human cognition.

Introduction to Word Embeddings

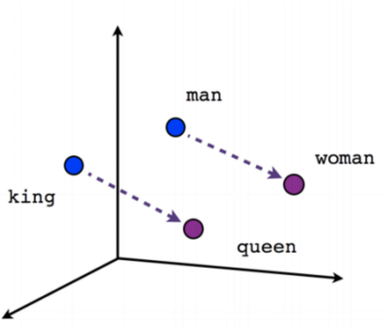
Word embeddings can be considered an alternative to one-hot encoding:



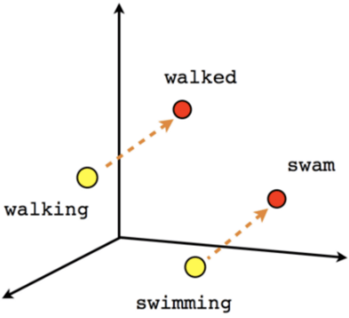
Source: *Deep Learning* by F. Chollet

Introduction to Word Embeddings

Example:



Male-Female



Verb tense

Applications of Word Embeddings

1. **Sentiment Analysis:** Improve accuracy by understanding context and nuances in opinions.
2. **Machine Translation:** Capture word equivalencies across languages, enhancing translation quality.
3. **Information Retrieval and Search:** Enable intuitive and relevant search results by understanding related terms and concepts.
4. **Recommendation Systems:** Use vectors to determine item and user similarities, improving recommendations.
5. **Word Similarity and Analogy Tasks:** Enable finding similar or related words, and solve analogy tasks (e.g., "king" is to "queen" as "man" is to "woman").
6. **Text Classification:** Enhance accuracy by providing semantically rich word representations.
7. **Named Entity Recognition (NER):** Facilitate identifying and classifying named entities like people, organizations, and locations.

Applications of Word Embeddings (Continued)

8. **Topic Modeling and Clustering:** Assist in grouping similar documents to uncover underlying themes within datasets.
9. **Question Answering Systems:** Improve retrieval and understanding of relevant information by capturing word relationships.
10. **Language Modeling and Generation:** Enhance sequence probability prediction, aiding in text understanding and generation.
11. **Relation Extraction:** Enable the extraction of semantic relationships between entities, enhancing databases and knowledge graphs.
12. **Social Media Monitoring:** Detect sentiment trends and conduct topic analysis in social media text.
13. **Spelling Correction and Typing Suggestions:** Provide more accurate recommendations by understanding context.

Contents

Limitations of Word Embeddings

- **Out-of-Vocabulary (OOV) Words:**
  - Static embeddings cannot handle words not seen during training.
  - Subword approaches like FastText can mitigate this.
- **Context Insensitivity (Static Embeddings):**
  - Single vector representation per word fails to capture different meanings in diverse contexts.
  - Modern context-specific embeddings address this limitation.
- **Bias Reflection and Amplification:**
  - Embeddings can encapsulate biases present in training data.
  - These biases can influence downstream applications, requiring debiasing techniques.
- **Resource Intensive:**
  - Training and fine-tuning embeddings require significant computational resources.
  - Infeasible for real-time deployment in limited-resource environments.

Limitations of Word Embeddings (Continued)

- **Semantic Drift:**
  - Static embeddings do not adapt to changes in language over time.
  - Requires periodic retraining to maintain relevance with language evolution.
- **Low-Resource Language Limitations:**
  - Less effective for languages with limited training data, potentially yielding poorer performance.

Contents

Word2Vec: Introduction and Key Models

- **Developed by Google**:
  - Created by a team at Google led by Tomas Mikolov in 2013.
  - Aimed to efficiently process vast amounts of text to produce high-quality word embeddings.
  - Revolutionized NLP by significantly improving the computational efficiency of training word vectors.
- **Key Models**:
  - **Skip-gram**:
    - ⋆ Predicts surrounding context words for a given target word.
    - ⋆ Works well with small datasets.
    - ⋆ Effective for modeling infrequent words by leveraging context.
  - **CBOW (Continuous Bag of Words)**:
    - ⋆ Predicts a target word based on a given context of surrounding words.
    - ⋆ More computationally efficient than Skip-gram for large datasets.
    - ⋆ Tends to perform better with frequent words.

Word2Vec: CBOW and Applications

- **Properties**:
  - Utilizes shallow neural networks with one hidden layer.
  - Trained using techniques like negative sampling or hierarchical softmax to optimize learning.
  - Capable of capturing semantic relationships, enabling vector arithmetic (e.g., "king" - "man" + "woman" = "queen").
- **Applications**:
  - Enhances document clustering, sentiment analysis, and recommendation systems.
  - Provides foundational embeddings for more complex models like BERT and GPT.

Contents

GloVe: Introduction and Core Idea

- **Developed by Stanford**:
  - Created by researchers at the Stanford NLP Group, including Jeffrey Pennington, Richard Socher, and Christopher D. Manning.
  - Released in 2014 to address limitations in word embeddings by integrating statistical and contextual information.
- **Core Idea**:
  - Utilizes a word co-occurrence matrix, recording how frequently words appear together across a corpus.
  - This matrix serves as the foundation for learning embeddings that reflect both direct and indirect word relations.
- **Training Objective**:
  - Aims to reconstruct the log-probabilities of word co-occurrences, where X_{ij} is the frequency at which word j appears in the context of word i.
  - Optimizes embeddings so that the dot product between two word vectors approximates the logarithm of their probability of co-occurrence.
  - Captures global context from the entire corpus while also focusing on local word relationships.

GloVe: Training Objective and Advantages (Continued)

- **Advantages**:
  - **Captures Complex Linguistic Patterns**:
    - GloVe embeddings encapsulate both semantic (meaning-related) and syntactic (grammar-related) relationships.
    - Effective at distinguishing between words with similar meanings and different contexts through subtle variations captured by co-occurrence.
  - **Holistic Text Understanding**:
    - Utilizes global statistical information by leveraging co-occurrence frequencies across the entire corpus.
    - Facilitates understanding of broader language structure, supporting tasks like sentiment analysis with nuanced context comprehension.
  - **Semantic Arithmetic with Embeddings**:
    - Supports intuitive operations, such as vector arithmetic: "king" - "man" + "woman" = "queen".
    - Demonstrates the ability to perform analogy tasks, reflecting the interconnected semantic space mapped by embeddings.

Comparison: Word2Vec vs. GloVe

- **Word2Vec**:
  - Uses local context.
  - Faster training, lower memory footprint.
- **GloVe**:
  - Combines global statistical information with local context.
  - More effective for capturing deeper patterns.
- **Selecting Models**:
  - Choice depends on dataset size and task specifics.
  - Word2Vec for large-scale, real-time adjustments.
  - GloVe for in-depth semantic understanding.

Practical Application in Python

```python
from gensim.models import Word2Vec
from glove import Glove, Corpus

# Sample Data
sentences = [["cat", "sat", "on", "the", "mat"],
             ["dog", "barked"],
             ["cat", "chased", "dog"]]

# Word2Vec Model
print("Training Word2Vec model...")
word2vec_model = Word2Vec(sentences, vector_size=100, window=5, min_count=1, workers=4)
print("Word2Vec model trained.")

# Accessing Word2Vec vector
print("Word2Vec Vector for 'cat':")
print(word2vec_model.wv['cat'])

# Preparing corpus for GloVe
corpus = Corpus()
print("Fitting GloVe corpus...")
corpus.fit(sentences, window=10)

# GloVe Model
print("Training GloVe model...")
glove_model = Glove(no_components=100, learning_rate=0.05)
glove_model.fit(corpus.matrix, epochs=30, no_threads=4, verbose=True)
print("GloVe model trained.")

# Accessing GloVe vector
glove_model.add_dictionary(corpus.dictionary)  # Aligning GloVe with corpus
print("GloVe Vector for 'cat':")
print(glove_model.word_vectors[corpus.dictionary['cat']])
```

Practical Application in Python

```
from gensim.models import Word2Vec
from glove import Glove, Corpus

# Sample Data
sentences = [["cat", "sat", "on", "the", "mat"],
             ["dog", "barked"],
             ["cat", "chased", "dog"]]

# Word2Vec Model
word2vec_model = Word2Vec(sentences, vector_size=100, window=5, min_count=1, workers=4)

# Preparing corpus for GloVe
corpus = Corpus()
corpus.fit(sentences, window=10)
glove_model = Glove(no_components=100, learning_rate=0.05)
glove_model.fit(corpus.matrix, epochs=30, no_threads=4, verbose=True)
```