A Brief Review of Machine Learning for Text

From Basic Techniques to Advanced Large Language Models

Jerome Niyirora, Ph.D.

March 14, 2024

Outline

Text-Based Machine Learning Basic Techniques

Text-Based Machine Learning

- **Objective**: Extracting useful insights from text

Text-Based Machine Learning

Common Applications

- Search engines: Crawling, indexing, and ranking
- Spam filters for emails
- News aggregation and categorization
- Recommender systems for content personalization
- Opinion mining and sentiment analysis for market insights

Text Preprocessing

Text Preprocessing

- **Preprocessing:** To clean and transform raw text into a standardized format suitable for machine learning models.
- **Relation to Bias-Variance Tradeoff:**
  - **Reducing Variance:** decreasing sensitivity to training data specifics and mitigating overfitting.
  - **Adjusting Bias:** Impacting model assumptions and its ability to capture complex patterns.
- **Challenge:** Finding the optimal preprocessing level to balance the bias-variance tradeoff effectively.

Text Preprocessing

- **Preprocessing:** To clean and transform raw text into a standardized format suitable for machine learning models.
- **Relation to Bias-Variance Tradeoff:**
  - **Reducing Variance:** decreasing sensitivity to training data specifics and mitigating overfitting.
  - **Adjusting Bias:** Impacting model assumptions and its ability to capture complex patterns.
- **Challenge:** Finding the optimal preprocessing level to balance the bias-variance tradeoff effectively.

Text Preprocessing

- **Preprocessing:** To clean and transform raw text into a standardized format suitable for machine learning models.
- **Relation to Bias-Variance Tradeoff:**
  - **Reducing Variance:** decreasing sensitivity to training data specifics and mitigating overfitting.
  - **Adjusting Bias:** Impacting model assumptions and its ability to capture complex patterns.
- **Challenge:** Finding the optimal preprocessing level to balance the bias-variance tradeoff effectively.

Text Preprocessing Steps

1. **Tokenization:** Splitting text into individual words or tokens.
   - Removes punctuation and splits on whitespace.
   - Example: “The quick brown fox” → [“The”, “quick”, “brown”, “fox”]
2. **Lowercasing:** Converting all characters in the text to lowercase.
   - Aids in uniformity and reduces vocabulary size.
   - Example: “Quick” → ”quick”
3. **Stop Words Removal:** Eliminating common words that add little value.
   - Words like “the”, “is”, “in” are often removed.
   - Example: [“the”, “quick”, “brown”, “fox”] → [“quick”, “brown”, “fox”]

Text Preprocessing Steps

1. **Tokenization:** Splitting text into individual words or tokens.
   - Removes punctuation and splits on whitespace.
   - Example: "The quick brown fox" → ["The", "quick", "brown", "fox"]
2. **Lowercasing:** Converting all characters in the text to lowercase.
   - Aids in uniformity and reduces vocabulary size.
   - Example: "Quick" → "quick"
3. **Stop Words Removal:** Eliminating common words that add little value.
   - Words like "the", "is", "in" are often removed.
   - Example: ["the", "quick", "brown", "fox"] → ["quick", "brown", "fox"]

Text Preprocessing Steps

1. **Tokenization:** Splitting text into individual words or tokens.
   - Removes punctuation and splits on whitespace.
   - Example: "The quick brown fox" → ["The", "quick", "brown", "fox"]
2. **Lowercasing:** Converting all characters in the text to lowercase.
   - Aids in uniformity and reduces vocabulary size.
   - Example: "Quick" → "quick"
3. **Stop Words Removal:** Eliminating common words that add little value.
   - Words like "the", "is", "in" are often removed.
   - Example: ["the", "quick", "brown", "fox"] → ["quick", "brown", "fox"]

Text Preprocessing Steps

4. **Stemming:** Reducing words to their root form.
   - Strips suffixes; "running" becomes "run".
   - Example: "jumps" $\rightarrow$ "jump"
5. **Lemmatization:** Reducing words to their base or dictionary form.
   - More sophisticated than stemming; uses vocabulary analysis.
   - Incorporates Part-of-Speech (POS) tagging
     - As a verb: "He is running." (running $\rightarrow$ run)
     - As a noun (gerund): "Running is fun." (running $\rightarrow$ running)
6. **Removing Special Characters and Numbers:** Cleansing text of non-linguistic elements.
   - Example: "Password123!" $\rightarrow$ "Password"
7. **Vectorization:** Converting text to numerical format for machine learning.
   - Techniques include Bag of Words, TF-IDF, Word Embeddings.

Text Preprocessing Steps

4. **Stemming:** Reducing words to their root form.
    - Strips suffixes; "running" becomes "run".
    - Example: "jumps" → "jump"
5. **Lemmatization:** Reducing words to their base or dictionary form.
    - More sophisticated than stemming; uses vocabulary analysis.
    - Incorporates Part-of-Speech (POS) tagging
        - As a verb: "He is running." (running → run)
        - As a noun (gerund): "Running is fun." (running → running)
6. **Removing Special Characters and Numbers:** Cleansing text of non-linguistic elements.
    - Example: "Password123!" → "Password"
7. **Vectorization:** Converting text to numerical format for machine learning.
    - Techniques include Bag of Words, TF-IDF, Word Embeddings.

Text Preprocessing Steps

4. **Stemming:** Reducing words to their root form.
   - Strips suffixes; "running" becomes "run".
   - Example: "jumps" → "jump"
5. **Lemmatization:** Reducing words to their base or dictionary form.
   - More sophisticated than stemming; uses vocabulary analysis.
   - Incorporates Part-of-Speech (POS) tagging
     - As a verb: "He is running." (running → run)
     - As a noun (gerund): "Running is fun." (running → running)
6. **Removing Special Characters and Numbers:** Cleansing text of non-linguistic elements.
   - Example: "Password123!" → "Password"
7. **Vectorization:** Converting text to numerical format for machine learning.
   - Techniques include Bag of Words, TF-IDF, Word Embeddings.

Text Preprocessing Steps

4. **Stemming:** Reducing words to their root form.
   - Strips suffixes; "running" becomes "run".
   - Example: "jumps" → "jump"
5. **Lemmatization:** Reducing words to their base or dictionary form.
   - More sophisticated than stemming; uses vocabulary analysis.
   - Incorporates Part-of-Speech (POS) tagging
     - As a verb: "He is running." (running → run)
     - As a noun (gerund): "Running is fun." (running → running)
6. **Removing Special Characters and Numbers:** Cleansing text of non-linguistic elements.
   - Example: "Password123!" → "Password"
7. **Vectorization:** Converting text to numerical format for machine learning.
   - Techniques include Bag of Words, TF-IDF, Word Embeddings.

Text Preprocessing Steps

Python Code (Preprocessing)

```python
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords, wordnet
from nltk.tokenize import word_tokenize
from nltk import pos_tag
ENGLISH_STOP_WORDS = set(stopwords.words('english'))
#
class LemmaTokenizer(object):
    def __init__(self, additional_stopwords=set()):
        self.wnl = WordNetLemmatizer()
        self.stopWords = ENGLISH_STOP_WORDS.union(additional_stopwords)
    def get_wordnet_pos(self, word_tag):
        """Map POS tag to first character lemmatize() accepts"""
        tag = word_tag[1][0].upper()
        tag_dict = {"J": wordnet.ADJ,
            "N": wordnet.NOUN,
            "V": wordnet.VERB,
            "R": wordnet.ADV}
        return tag_dict.get(tag, wordnet.NOUN)
    def __call__(self, doc):
        # Tokenize and lower case
        tokens = word_tokenize(doc.lower())
        # POS tagging before lemmatization
        tagged_tokens = pos_tag(tokens)
        # Lemmatize with POS tagging, filter out stop words and non-alphabetic words
        return [self.wnl.lemmatize(token, self.get_wordnet_pos(tagged_token))
            for token, tagged_token in zip(tokens, tagged_tokens)
            if token.isalpha() and token not in self.stopWords]
# Example usage
tokenizer = LemmaTokenizer()
doc = "The foxes are quickly jumping over the lazy dogs."
print(tokenizer(doc))
```

Vectorization

Vectorization: Bag of Words (BoW)

Description:

- Represents text documents as vectors of word counts.
- Ignores word order, context, and grammar.

Mathematical Representation: A document D is represented as a vector $V = (v_1, v_2, ..., v_n)$ where each v_i corresponds to the count of word w_i in the document, and n is the vocabulary size.

Example

Word	Document 1	Document 2	Document 3
apple	2	0	1
banana	0	1	1
orange	1	0	0

Vectorization: Bag of Words (BoW)

Description:

- Represents text documents as vectors of word counts.
- Ignores word order, context, and grammar.

Mathematical Representation: A document D is represented as a vector $V = (v_1, v_2, ..., v_n)$ where each v_i corresponds to the count of word w_i in the document, and n is the vocabulary size.

Example

Word	Document 1	Document 2	Document 3
apple	2	0	1
banana	0	1	1
orange	1	0	0

Vectorization: Bag of Words (BoW)

Description:

- Represents text documents as vectors of word counts.
- Ignores word order, context, and grammar.

Mathematical Representation: A document D is represented as a vector $V = (v_1, v_2, ..., v_n)$ where each v_i corresponds to the count of word w_i in the document, and n is the vocabulary size.

Example

Word	Document 1	Document 2	Document 3
apple	2	0	1
banana	0	1	1
orange	1	0	0

Vectorization: Term Frequency-Inverse Document Frequency (TF-IDF)

Description:

- Balances term frequency against the word's commonness across documents.

Mathematical Formulas:

- $\text{TF}(w, D) = (\text{Number of times word } w \text{ appears in document } D) / (\text{Total number of words in } D)$
- $\text{IDF}(w, D) = \log(\frac{\text{Total number of documents}}{\text{Number of documents containing } w})$
- $\text{TF-IDF}(w, D) = \text{TF}(w, D) \times \text{IDF}(w, D)$

Example

Word	Document 1	Document 2	Document 3
apple	0.27	0.00	0.20
banana	0.00	0.41	0.20
orange	0.37	0.00	0.00

Vectorization: Term Frequency-Inverse Document Frequency (TF-IDF)

Description:

- Balances term frequency against the word's commonness across documents.

Mathematical Formulas:

- $\text{TF}(w, D) = (\text{Number of times word } w \text{ appears in document } D) / (\text{Total number of words in } D)$
- $\text{IDF}(w, D) = \log\left(\frac{\text{Total number of documents}}{\text{Number of documents containing } w}\right)$
- $\text{TF-IDF}(w, D) = \text{TF}(w, D) \times \text{IDF}(w, D)$

Example

Word	Document 1	Document 2	Document 3
apple	0.27	0.00	0.20
banana	0.00	0.41	0.20
orange	0.37	0.00	0.00

Vectorization: Term Frequency-Inverse Document Frequency (TF-IDF)

Description:

- Balances term frequency against the word's commonness across documents.

Mathematical Formulas:

- $\text{TF}(w, D) = (\text{Number of times word } w \text{ appears in document } D) / (\text{Total number of words in } D)$
- $\text{IDF}(w, D) = \log(\frac{\text{Total number of documents}}{\text{Number of documents containing } w})$
- $\text{TF-IDF}(w, D) = \text{TF}(w, D) \times \text{IDF}(w, D)$

Example

Word	Document 1	Document 2	Document 3
apple	0.27	0.00	0.20
banana	0.00	0.41	0.20
orange	0.37	0.00	0.00

Vectorization: Term Frequency-Inverse Document Frequency (TF-IDF)

Implementation in Sklearn:

- tfidfTransformer = TfidfTransformer(smooth_idf=False, norm=None)
- $\text{TF}(w, D) = \text{Number of times word } w \text{ appears in document } D$
- $\text{IDF}(w, D) = \log(\frac{\text{Total number of documents}}{\text{Number of documents containing } w})$
- $\text{TF-IDF}(w, D) = \text{TF}(w, D) \times \text{IDF}(w, D) + \text{TF}(w, D)$

Example

Word	Document 1	Document 2	Document 3
apple	2.81	0.00	1.41
banana	0.00	1.41	1.41
orange	2.10	0.00	0.00

Vectorization: Term Frequency-Inverse Document Frequency (TF-IDF)

Implementation in Sklearn:

- tfidfTransformer = TfidfTransformer(smooth_idf=False, norm=None)
- $\text{TF}(w, D)$ = Number of times word w appears in document D
- $\text{IDF}(w, D) = \log(\frac{\text{Total number of documents}}{\text{Number of documents containing } w})$
- $\text{TF-IDF}(w, D) = \text{TF}(w, D) \times \text{IDF}(w, D) + \text{TF}(w, D)$

Example

Word	Document 1	Document 2	Document 3
apple	2.81	0.00	1.41
banana	0.00	1.41	1.41
orange	2.10	0.00	0.00

Vectorization: Term Frequency-Inverse Document Frequency (TF-IDF)

Python Code (TF-IDF)

```python
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer
import pandas as pd

text = ['apple apple orange',
'banana',
'apple banana']

count_vect = CountVectorizer()
tfidf_transformer = TfidfTransformer(smooth_idf=False, norm=None)

# Generate term frequency (TF) matrix
counts = count_vect.fit_transform(text)

# Generate TF-IDF matrix
tfidf = tfidf_transformer.fit_transform(counts)

# Create a pandas DataFrame for the TF-IDF matrix
# The rows correspond to the documents, and the columns correspond to the terms.
feature_names = count_vect.get_feature_names_out()
tfidf_df = pd.DataFrame(tfidf.toarray(), columns=feature_names, index=['Document 0
    ', 'Document 1', 'Document 2'])

print(tfidf_df)
```

Similarity Computation in Text

- Euclidean distance is **not ideal** for computing distances in multidimensional sparse spaces due to the varying number of zeros (due to different sizes of documents).

$$\text{Distance}(X, Y) = \sqrt{\sum_{i=1}^{d} (x_i - y_i)^2}$$

- The smaller the distance, the more similar the documents are.

Example

Consider the documents: ['She sat down.', 'She drank coffee.', 'She spent much time learning text mining.', 'She invested significant efforts in learning text mining.'] The Euclidean distance, in boolean form, between doc 1 and doc 2 is $\sqrt{2}$, while the distance between doc 3 and doc 4 is $\sqrt{6}$. This does not make intuitive sense since docs 3 and 4 are more related than docs 1 and 2.

- **Solution:** Cosine similarity

Similarity Computation in Text

- Euclidean distance is **not ideal** for computing distances in multidimensional sparse spaces due to the varying number of zeros (due to different sizes of documents).

$$\text{Distance}(X, Y) = \sqrt{\sum_{i=1}^{d} (x_i - y_i)^2}$$

- The smaller the distance, the more similar the documents are.

Example

Consider the documents: ['She sat down.', 'She drank coffee.', 'She spent much time learning text mining.', 'She invested significant efforts in learning text mining.'] The Euclidean distance, in boolean form, between doc 1 and doc 2 is $\sqrt{2}$, while the distance between doc 3 and doc 4 is $\sqrt{6}$. This does not make intuitive sense since docs 3 and 4 are more related than docs 1 and 2.

- **Solution:** Cosine similarity

Similarity Computation in Text

Example

Consider the documents: ['She sat down.', 'She drank coffee.', 'She spent much time learning text mining.', 'She invested significant efforts in learning text mining.'] The Euclidean distance, in boolean form, between doc 1 and doc 2 is $\sqrt{2}$, while the distance between doc 3 and doc 4 is $\sqrt{6}$. This does not make intuitive sense since docs 3 and 4 are more related than docs 1 and 2.

- **Solution:** Cosine similarity

$$\text{cosine}(X, Y) = \frac{\sum_{i=1}^{d} x_i y_i}{\sqrt{\sum_{i=1}^{d} x_i^2}\sqrt{\sum_{i=1}^{d} y_i^2}}$$

- Measures the angle between the multidimensional vectors.
- Does not depend on the size of the documents.
- Ranges between 0 and 1 for BoW problems. The higher the measure the more similar the documents.

Feature Engineering and Dimension Reduction in Text

Feature Engineering and Dimension Reduction I

Feature Engineering

Definition: The process of creating features from raw text to improve model performance.

Examples:

- *Bag of Words (BoW)*: Counts the occurrence of each word within a document, ignoring order.
- *TF-IDF (Term Frequency-Inverse Document Frequency)*: Weighs the words' counts by how unique they are across documents, highlighting important words.

Feature Engineering and Dimension Reduction II

Dimension Reduction

Definition: Techniques used to reduce the number of features in NLP to improve computational efficiency and model performance.

Examples:

- *Principal Component Analysis (PCA)*: A mathematical technique to reduce dataset dimensions while preserving as much variance as possible.
- *Latent Semantic Analysis (LSA)*: Uses singular value decomposition (SVD) on the term-document matrix to identify patterns and reduce dimensions, often applied after TF-IDF.

Feature Engineering and Dimension Reduction III

Application Example

Preprocessing Pipeline:

1. Start with raw text data from documents.
2. Apply TF-IDF to highlight important words while reducing common words' impact.
3. Use LSA to reduce feature space, focusing on latent topics.
4. The reduced feature set is now ready for machine learning model training (e.g., classification or regression tasks).

Note: Dimension reduction mitigates the issue of sparsity in data, which is beneficial as sparsity can adversely affect certain classifiers, such as decision trees.

Feature Engineering and Dimension Reduction III

Application Example

Preprocessing Pipeline:

1. Start with raw text data from documents.
2. Apply TF-IDF to highlight important words while reducing common words' impact.
3. Use LSA to reduce feature space, focusing on latent topics.
4. The reduced feature set is now ready for machine learning model training (e.g., classification or regression tasks).

Note: Dimension reduction mitigates the issue of sparsity in data, which is beneficial as sparsity can adversely affect certain classifiers, such as decision trees.

Word Embeddings

Word Embeddings

Description:

- Represents words as dense vectors capturing semantic meanings.
- Vectors are learned from text by predicting a word given its context (or vice versa).

Characteristics: Words with similar meanings have similar vectors. Not explicitly mathematical, but based on optimization during training (e.g., via neural networks).

Example

Word	Vector
apple	(0.23, -1.0, 0.32, ...)
banana	(0.21, -0.97, 0.31, ...)
orange	(0.20, -0.99, 0.33, ...)

Table 1: Simplified Word Embedding Vectors

Word Embeddings

Description:

- Represents words as dense vectors capturing semantic meanings.
- Vectors are learned from text by predicting a word given its context (or vice versa).

Characteristics: Words with similar meanings have similar vectors. Not explicitly mathematical, but based on optimization during training (e.g., via neural networks).

Example

Word	Vector
apple	(0.23, -1.0, 0.32, ...)
banana	(0.21, -0.97, 0.31, ...)
orange	(0.20, -0.99, 0.33, ...)

Table 1: Simplified Word Embedding Vectors

Word Embeddings

Description:

- Represents words as dense vectors capturing semantic meanings.
- Vectors are learned from text by predicting a word given its context (or vice versa).

Characteristics: Words with similar meanings have similar vectors. Not explicitly mathematical, but based on optimization during training (e.g., via neural networks).

Example

Word	Vector
apple	(0.23, -1.0, 0.32, ...)
banana	(0.21, -0.97, 0.31, ...)
orange	(0.20, -0.99, 0.33, ...)

Table 1: Simplified Word Embedding Vectors

Word Embeddings Training Using Word2Vec

Foundation:

- Word2Vec utilizes a shallow neural network architecture to learn word embeddings.
- Two main models: **Continuous Bag of Words (CBOW)** and **Skip-Gram**.
- **CBOW** predicts a target word from a window of surrounding context words.
- **Skip-Gram** does the opposite, predicting context words from a target word.
- The objective function maximizes the probability of observing a word given its context (or vice versa) using softmax or negative sampling.

Word Embeddings Training Using Word2Vec

Foundation:

- Word2Vec utilizes a shallow neural network architecture to learn word embeddings.
- Two main models: **Continuous Bag of Words (CBOW)** and **Skip-Gram**.
- **CBOW** predicts a target word from a window of surrounding context words.
- **Skip-Gram** does the opposite, predicting context words from a target word.
- The objective function maximizes the probability of observing a word given its context (or vice versa) using softmax or negative sampling.

Word Embeddings Training Using Word2Vec

Foundation:

- Word2Vec utilizes a shallow neural network architecture to learn word embeddings.
- Two main models: **Continuous Bag of Words (CBOW)** and **Skip-Gram**.
- **CBOW** predicts a target word from a window of surrounding context words.
- **Skip-Gram** does the opposite, predicting context words from a target word.
- The objective function maximizes the probability of observing a word given its context (or vice versa) using softmax or negative sampling.

Word Embeddings Training Using Word2Vec

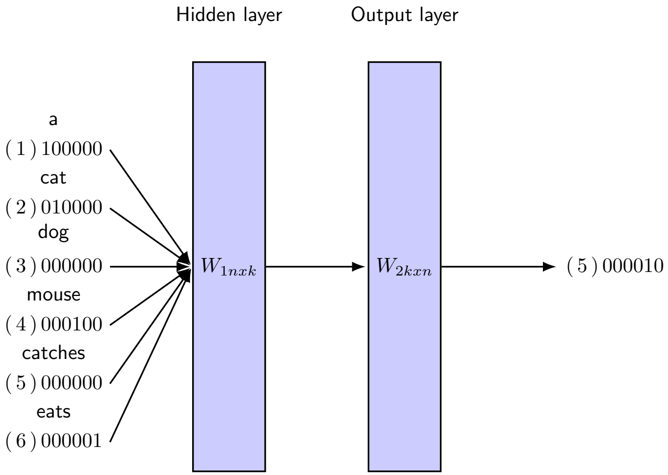

Figure 1: Adapted from `https://www.researchgate.net/figure/word2vec-CBOW-model_fig1_313247648`

Vectorization: Word Embeddings Using Word2Vec

Python Code (Word2Vec)

```python
from gensim.models import Word2Vec
import nltk

nltk.download('punkt')

text = ['apple apple orange','banana','apple banana']

# Tokenize the documents (split them into words)
tokenized_text = [nltk.word_tokenize(doc) for doc in text]

# Train a Word2Vec model on the tokenized documents
word2vec_model = Word2Vec(sentences=tokenized_text, vector_size=100,
    window=5, min_count=1, workers=4)

# For example, to get the embedding for the word 'apple':
apple_embedding = word2vec_model.wv['apple']

# Here's how you could embed a documente:
document_embedding = sum(word2vec_model.wv[word] for word in
    tokenized_text[0]) / len(tokenized_text[0])

print(apple_embedding)  # This will print the embedding for 'apple'
```

Word Embeddings Using Pre-trained Embedding Models I

Models

- Pre-trained embedding models based on neural network models
- Trained on large text corpora to map words to high-dimensional vectors.
- Examples include **Word2Vec, GloVe, and FastText**.

Word Embeddings Using Pre-trained Embedding Models II

Advantages

- **Rich Semantic Representations:** Capture deep linguistic patterns and relationships.
- **Ready to Use:** Eliminate the need for extensive computational resources for training.
- **Versatility:** Can be utilized across various NLP tasks and applications.
- **Improved Performance:** Often enhance model performance, especially on smaller datasets.

Word Embeddings Using Pre-trained Embedding Models III

Disadvantages

- **Fixed Vocabulary:** Struggle with out-of-vocabulary (OOV) words not present in the training corpus.
- **Generic Context:** May not capture domain-specific nuances if trained on general corpora.
- **Storage and Speed:** Large models require significant memory and can slow down applications.
- **Lack of Customizability:** Difficult to adjust the embeddings to specific needs or biases in the data.

Note: Choosing the right pre-trained model depends on the specific requirements and constraints of the NLP task at hand.

Word Embeddings Using Pre-trained Embedding Models IV

Example

Retrieving an Embedding Vector for "Apple"

- **Word Index Lookup:** Models map words to unique indices. "Apple" is looked up in the model's dictionary to find its index.
- **Embedding Matrix:** Embeddings are stored in a matrix where rows correspond to words and their indices.
- **Retrieving the Vector:** The index from the lookup step is used to retrieve "apple"'s embedding vector from the matrix.
- **Result:** The retrieved vector is a dense representation capturing semantic and syntactic properties of "apple".
- **Out-of-Vocabulary (OOV) Words:** If "apple" is not in the model's vocabulary, handling varies (e.g., error or generating a vector from subwords in models like FastText).

Word Embeddings Using Pre-trained Embedding Models V

Python Code (Pre-trained Embedding Models)

```python
from gensim.models.keyedvectors import KeyedVectors

# Specify the path to where you've saved the
    extracted GloVe file
glove_file = 'glove.6B.50d.txt'

# Load the model (this might take a while)
model = KeyedVectors.load_word2vec_format(glove_file,
     binary=False, no_header=True)

# Get the vector for 'apple'
apple_vector = model['apple']
print(apple_vector[:10])
```

BoW vs. CBoW

BoW vs. CBoW

- **Word Frequency vs. Context**:
  - BoW models focus on the frequency of words within documents
  - CBoW learns word representations by considering the context
- **Simplicity vs. Semantic Depth**:
  - BoW's simplicity makes it suitable for a range of traditional NLP tasks
  - CBoW offers foundational support for advanced models (e.g.,named entity recognition and machine translation).
- **Applications**:
  - BoW is often utilized in applications where the presence of specific words is more critical than their order or context.
  - CBoW is used when the understanding of word usage and meaning within context is important.

BoW vs. CBoW

- **Word Frequency vs. Context**:
  - BoW models focus on the frequency of words within documents
  - CBoW learns word representations by considering the context
- **Simplicity vs. Semantic Depth**:
  - BoW's simplicity makes it suitable for a range of traditional NLP tasks
  - CBoW offers foundational support for advanced models (e.g.,named entity recognition and machine translation).
- **Applications**:
  - BoW is often utilized in applications where the presence of specific words is more critical than their order or context.
  - CBoW is used when the understanding of word usage and meaning within context is important.

BoW vs. CBoW

- **Word Frequency vs. Context**:
  - BoW models focus on the frequency of words within documents
  - CBoW learns word representations by considering the context
- **Simplicity vs. Semantic Depth**:
  - BoW's simplicity makes it suitable for a range of traditional NLP tasks
  - CBoW offers foundational support for advanced models (e.g.,named entity recognition and machine translation).
- **Applications**:
  - BoW is often utilized in applications where the presence of specific words is more critical than their order or context.
  - CBoW is used when the understanding of word usage and meaning within context is important.

BoW vs. Matrix Factorization

CBoW vs. Matrix Factorization

CBoW

Use

- When context of use is crucial for understanding word meaning.
- For capturing semantic similarities based on surrounding words.
- Efficient for large datasets with rich contextual information.

Matrix Factorization

Use

- When handling very sparse data or aiming for dimensionality reduction.
- For probabilistic interpretations of word co-occurrences.
- Useful in tasks requiring detailed probabilistic models of text.

CBoW vs. Matrix Factorization

CBoW

Use

- When context of use is crucial for understanding word meaning.
- For capturing semantic similarities based on surrounding words.
- Efficient for large datasets with rich contextual information.

Matrix Factorization

Use

- When handling very sparse data or aiming for dimensionality reduction.
- For probabilistic interpretations of word co-occurrences.
- Useful in tasks requiring detailed probabilistic models of text.

Word2Vec vs. Recurrent neural networks (RNNs)

Word2Vec vs. RNNs

Key Distinctions

- Use **Word2Vec** for generating word embeddings where local context suffices.
- Use **RNNs** when sequential order and longer contexts (e.g., a sentence) significantly impact meaning.
- Transformers are now often preferred for complex sequence modeling due to parallelization and handling long-range dependencies.

Leveraging the Power of Large Language Models

A Brief Introduction to LLMs

Understanding Large Language Models (LLMs) I

What are LLMs?

- LLMs are **advanced neural network architectures** trained on vast amounts of text data to understand, generate, and interpret human language.

Understanding Large Language Models (LLMs) II

How are LLMs Generated?

- **Data Collection:** Gather large, diverse text corpora from books, websites, articles, and other sources.
- **Pre-training:** Train the model on this text data using unsupervised learning techniques to learn the statistical properties of the language.
  - For GPT: Focus on predicting the next word in a sequence.
  - For BERT: Learn by predicting masked words in a sentence, understanding context in both directions.

Understanding Large Language Models (LLMs) III

Key Characteristics

- **Scale:** Encompass billions or even trillions of parameters, enabling them to capture a wide array of linguistic subtleties.
- **Versatility:** Can perform a variety of language tasks, such as translation, summarization, question answering, and creative writing, without task-specific training.
- **Contextual Understanding:** Excel at grasping context and generating responses that are relevant and coherent over extended passages of text.

Understanding Large Language Models (LLMs) IV

Example

- **GPT Series (OpenAI)**: The GPT series, including GPT-3.5 and GPT-4, are used in applications like ChatGPT and Microsoft Copilot.
- **PaLM and Gemini (Google)**: These models are used in Google's applications.
- **LLaMA (Meta)**: This is a family of open-source models developed by Meta.
- **Claude (Anthropic)**: These models are used in applications like Slack, Notion, and Zoom.

The Transformative Transformer Architecture

Transformer Architecture in Large Language Models I

Attention is All You Need

The **Transformer architecture**, introduced in the paper "Attention is All You Need" by Vaswani et al. (2017), revolutionized NLP by **enabling models to process words in parallel and capture contextual** information from sequences efficiently.

Transformer Architecture in Large Language Models II

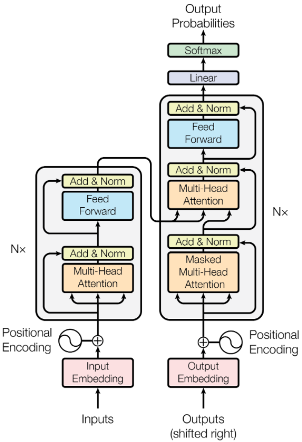

Figure 2: Source and Implementation Example:
`https://nlp.seas.harvard.edu/2018/04/03/attention.html`

Transformer Architecture in Large Language Models III

Key Components

- **Self-Attention Mechanism**: Allows each word to dynamically focus on other parts of the sentence, capturing intricate dependencies.
- **Positional Encoding**: Injects information about the position of each word in the sequence, compensating for the absence of recurrence.
- **Stacked Encoders and Decoders**: Multiple layers (stacks) of encoders for input processing and decoders for output generation, enabling deep understanding.
- **Feed-Forward Neural Networks**: Each layer contains feed-forward networks for transforming attention-combined inputs into outputs.

Interacting with Large Language Models

Interacting with Large Language Models I

Source: Document Link: SUNY Albany

Choosing the Right LLM

- Avoid using OpenAI's original ChatGPT due to its limitations.
- **Bing Chat/Copilot**: Free, internet-connected, suitable for general inquiries.
- **OpenAI GPT-4**: Offers deeper insights for a subscription fee, excelling in specific tasks.
- **Claude.ai**: Ideal for processing and summarizing larger documents.

Interacting with Large Language Models II

Effective Prompting Strategies

- Experiment with Bing Chat/Copilot's "creative" and "precise" settings to suit your query needs.
- Request more examples than needed (e.g., "give me 10 examples") and select the most relevant ones.
- Engage in a conversational manner with the LLM. If initial responses don't meet expectations, prompt adjustments.

Interacting with Large Language Models III

Best Practices

LLMs serve as a source of ideas and support. However, critical engagement and verification of the information provided are essential:

- Always review and verify the LLM's output.
- Use the model's feedback constructively but remain cautious about its limitations.

Query Examples with Openai API

Comparing Tesla's Financial Reports

Query Structure

- Clearly specify the aspect of the financial reports you're interested in, such as revenue growth, expense changes, profitability, or investment activities.
- Include any relevant context, such as market conditions, product launches, or significant corporate events, that might impact year-over-year comparisons.
- Ask for a concise summary of key financial indicators and their variance over the two periods.
- Encourage the identification of trends or patterns in the financial data across the two reports.

Example

1. User: "Can you compare the key financial differences in Tesla's final reports between December 2023 and December 2022, focusing on

Comparing Tesla's Financial Reports

Query Structure

- Include any relevant context, such as market conditions, product launches, or significant corporate events, that might impact year-over-year comparisons.
- Ask for a concise summary of key financial indicators and their variance over the two periods.
- Encourage the identification of trends or patterns in the financial data across the two reports.

Example

1. User: "Can you compare the key financial differences in Tesla's final reports between December 2023 and December 2022, focusing on revenue growth and R&D investments?"

Comparing Tesla's Financial Reports

Query Structure

- Include any relevant context, such as market conditions, product launches, or significant corporate events, that might impact year-over-year comparisons.
- Ask for a concise summary of key financial indicators and their variance over the two periods.
- Encourage the identification of trends or patterns in the financial data across the two reports.

Example

1. User: "Can you compare the key financial differences in Tesla's final reports between December 2023 and December 2022, focusing on revenue growth and R&D investments?"

Comparing Tesla's Financial Reports

Query Structure

- Include any relevant context, such as market conditions, product launches, or significant corporate events, that might impact year-over-year comparisons.
- Ask for a concise summary of key financial indicators and their variance over the two periods.
- Encourage the identification of trends or patterns in the financial data across the two reports.

Example

1. User: "Can you compare the key financial differences in Tesla's final reports between December 2023 and December 2022, focusing on revenue growth and R&D investments?"

Comparing Tesla's Financial Reports III

Python Code (Compare Texts)

```
import openai
openai.api_key = 'sk-...y'
#
report1 = data.iloc[0]['1A_Text']
report2 = data.iloc[1]['1A_Text']
period1 = data.iloc[0]['Reporting Period'].strftime('%B-%Y')
period2 = data.iloc[1]['Reporting Period'].strftime('%B-%Y')

# query
query = f"Can you compare the key financial differences in Tesla \
financial report from {period1}, as given in {report1}, and the financial
    report from {period2}, as given in {report2},focusing on various risks
    and \operational challenges that could impact Tesla's finacial condition?
    "

response = openai.chat.completions.create(
messages=[
{'role': 'system', 'content': 'You are asked to compare two financial
    reports from TESLA and \
     identify key differences.'},
{'role': 'user', 'content': query},
],
model='gpt-4-0125-preview',
temperature=0,  # Keep response consistent
)

print(response.choices[0].message.content)
```

Text Summarization with GPT Models

Python Code (Text Summarization)

```python
import openai
openai.api_key = 'sk-...y'
text1 = data.iloc[0]['1A_Text']
# query
query = f' In one paragraph, summarize {text1} and speculate on
    the financial outlook of this company as Positive, Negative, or
     Unsure'

response = openai.chat.completions.create(
messages=[
{'role': 'system', 'content': 'You are asked to summarize
    financial reporting from TESLA and speculate on the financial
    outlook.'},
{'role': 'user', 'content': query},
],
model='gpt-4-0125-preview',
temperature=0,
)

print(response.choices[0].message.content)
```

Sentiment Analysis with GPT Models

Python Code (Sentiment Analysis)

```python
def sentiment_analysis(text):
    # Create a prompt for the model
    prompt = f"""...."""

    # Call the OpenAI API to generate a response
    response = openai.chat.completions.create(....
    )
    # Extract the sentiment from the response
    sentiment = response.choices[0].message.content.strip().lower()
    return sentiment
texts = [
"Battery life is too short for my liking.",
"Broke after a week",
"Not worth the price.",
"The material feels cheap and unpleasant.",
"Arrived on time, works perfectly"
]

# Perform sentiment analysis on each text
results = [(text, sentiment_analysis(text)) for text in texts]
```

Future Directions of LLMs

Future Directions for Large Language Models I

Technology

- **Combining LLMs with other AI technologies**: Integrating reinforcement learning for better decision-making and multimodal capabilities for processing various data types.
- **Efficiency**: Developing more efficient models to reduce computational demands.
- **Domain Adaptation**: Tailoring LLMs for specialized domains or tasks more effectively.
- **Interpretability and Personalization**: Enhancing models' explainability and ability to offer personalized experiences.

Future Directions for Large Language Models II

Society

- **Model Transparency and Ethical Considerations**: Focusing on interpretability, bias mitigation, fairness, and privacy.
- **Regulation and Policy**: Developing frameworks to govern the use and impacts of LLMs.
- **Human-AI Collaboration**: Facilitating synergistic collaborations between humans and LLMs.
- **Addressing the Digital Divide and Global Perspectives**: Ensuring equitable access and considering diverse cultural and linguistic needs.
- **Education and Workforce Development**: Adapting education systems and workforce training to align with evolving AI technologies.

References

References

- Géron, A. (2022). Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow. " O'Reilly Media, Inc.".
- Shmueli, G., Bruce, P. C., Yahav, I., Patel, N. R., & Lichtendahl Jr, K. C. (2020). Data mining for business analytics: concepts, techniques, and applications in Python. John Wiley & Sons.
- Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. MIT press.
- Aggarwal, C. C., & Aggarwal, C. C. (2018). Machine learning for text: An introduction. Springer International Publishing.
- https://docs.google.com/document/d/1Lo4aeiWT4f5xhcsAbWAfQRITghBhcmFN2m-JEX5OkJA/edit
- https://www.researchgate.net/figure/word2vec-CBOW-model_fig1_313247648
- https://nlp.seas.harvard.edu/2018/04/03/attention.html
- GPT-4 for Beamer frame code and proofreading

?