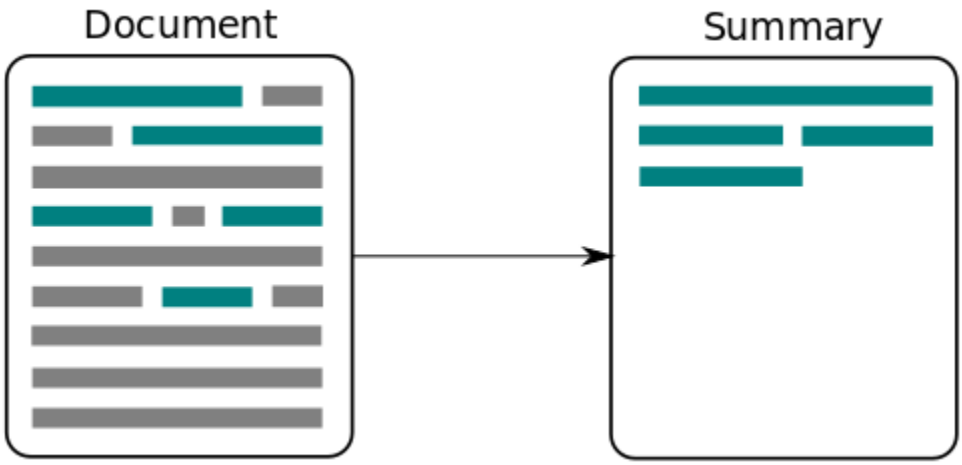
Lesson 13: Text Summarization

Text summarization is the process of automatically generating a shorter version of a longer text document while preserving its most important information and main ideas. The goal of text summarization is to reduce the time and effort required to read and comprehend a large amount of text, while still conveying the key points and main message of the original document.



There are two main types of text summarization: extractive and abstractive. Extractive summarization involves selecting the most important sentences or phrases from the original text and combining them to form a summary. Abstractive summarization, on the other hand, involves generating new sentences that capture the main ideas of the original text, often using natural language processing techniques such as language generation models.

Text summarization algorithms can be based on various methods, such as statistical analysis, graph-based algorithms, and deep learning models. These algorithms analyze the text and identify the most important information based on various factors, such as sentence length, keyword frequency, and semantic similarity.

Text summarization has many practical applications, such as news summarization, document summarization, and email summarization. It can help to quickly and efficiently process and comprehend large volumes of text data, such as news articles or research papers. However, the quality of the summarization heavily depends on the accuracy and comprehensiveness of the original text and the capabilities of the summarization algorithm used.

Extractive Summarization

Extractive summarization is a text summarization technique that involves selecting the most important sentences or phrases from the original text to create a summary. This technique doesn't modify or restructure the selected sentences, but rather takes them directly from the original text.

To perform extractive summarization, the original text is first pre-processed by removing unnecessary elements such as stop words and punctuation. Then, each sentence in the pre-processed text is scored based on factors such as word frequency, sentence length, and semantic similarity with other sentences. The top-scoring sentences are selected to create a summary, and the number of sentences chosen depends on the desired length of the summary.

Extractive summarization is advantageous because it preserves the exact wording and structure of the original text, making it useful in legal or scientific contexts. Additionally, this technique is relatively fast and easy to implement compared to other summarization techniques.

However, extractive summarization has some limitations. It may not be able to capture the full meaning or context of the original text, leading to a summary that is incomplete or misleading. It also struggles with complex sentences or idiomatic expressions, which can result in a summary that is difficult to comprehend.

EXAMPLE CODE

The code below shows an example implementation of TextRank, a popular graph-based algorithm for extractive summarization. The **textrank_summary** function takes a text and the desired number of summary sentences as input, tokenizes the text into sentences using the **sent_tokenize** function from the **nltk** library, and creates a graph of the sentences using the **networkx** library. The **get_word_embeddings** function computes the average word embedding for each sentence using pre-trained word embeddings. The similarity between sentences is calculated using cosine similarity between their word embeddings, and the **nx.pagerank** function is used to compute the TextRank scores for each sentence. Finally, the top-scoring sentences are selected and returned as the summary.

```
import networkx as nx
import nltk
nltk.download('punkt')


from nltk.tokenize import sent_tokenize, word_tokenize
from sklearn.metrics.pairwise import cosine_similarity


def textrank_summary(text, num_sentences):
    # Tokenize the text into sentences
    sentences = sent_tokenize(text)


    # Create a graph of the sentences
    graph = nx.Graph()
    for i, sentence_i in enumerate(sentences):
        for j, sentence_j in enumerate(sentences):
            if i != j:
                similarity =
cosine_similarity(get_word_embeddings(sentence_i),
get_word_embeddings(sentence_j))[0][1]
                graph.add_edge(i, j, weight=similarity)


    # Compute the TextRank scores for each sentence
    scores = nx.pagerank(graph)


    # Select the top-scoring sentences and return them as the summary
    summary = []
    for index, score in sorted(scores.items(), key=lambda x: x[1],
reverse=True)[:num_sentences]:
        summary.append(sentences[index])


    return ' '.join(summary)
```

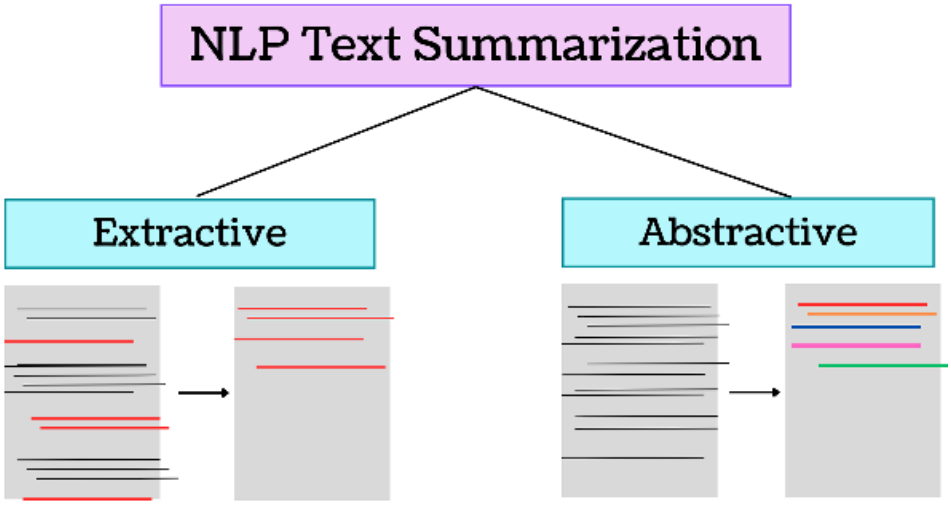
```
def get_word_embeddings(text):
    # Tokenize the text into words
    words = word_tokenize(text)


    # Compute the average word embedding for the text
    word_embeddings = []
    for word in words:
        try:
            word_embeddings.append(embeddings[word])
        except KeyError:
            continue
    if len(word_embeddings) == 0:
        return np.zeros(embeddings.vector_size)
    else:
        return np.mean(word_embeddings, axis=0)


# Example usage
text = "Deep learning is a subset of machine learning that focuses on
training artificial neural networks to perform tasks such as image
recognition and natural language processing. In recent years, deep
learning has become increasingly popular due to its ability to
achieve state-of-the-art performance on a wide range of tasks.
However, training deep neural networks can be computationally
expensive and requires large amounts of data. In this paper, we
review the history and development of deep learning, discuss its
applications in various fields, and highlight some of the challenges
and future directions of the field."
num_sentences = 2
summary = textrank_summary(text, num_sentences)
print(summary)
```

Abstractive Summarization

Abstractive summarization is a type of text summarization that aims to generate a summary that captures the essence of the original text, while still conveying the main ideas and important information. It differs from extractive summarization, which selects sentences directly from the original text without any modification, by using natural language processing techniques to generate new sentences that convey the meaning of the original text.

To perform abstractive summarization, the original text is pre-processed to remove stop words, punctuation, and other non-essential information. The pre-processed text is then transformed into a numerical representation, such as a vector or matrix, which can be used as input for the summarization algorithm. The summarization algorithm then generates new sentences that capture the main ideas of the original text, using natural language processing techniques such as language generation models.

One of the key advantages of abstractive summarization is its ability to capture the underlying meaning and context of the original text, making the summary more informative and readable. This technique is also more flexible than extractive summarization, as it can handle complex sentences and idiomatic expressions. Abstractive summarization is useful in various applications, such as news summarization, video summarization, and conversational summarization.



However, abstractive summarization can be more challenging to implement and requires more training data than extractive summarization. The generated sentences may sometimes contain grammatical errors or be less coherent compared to the original text. Evaluating the quality of the generated summary is also more challenging than in extractive summarization, as it involves assessing the coherence, relevance, and readability of the summary.

To address some of these challenges, ongoing research is focused on improving the accuracy and efficiency of abstractive summarization techniques, such as developing better language models and optimizing the summarization algorithm. Additionally, hybrid approaches that combine both extractive and abstractive summarization techniques are being explored to improve the quality and efficiency of summarization.

EXAMPLE CODE

This code is a basic implementation of a sequence-to-sequence model for abstractive text summarization using LSTM neural networks. It loads and preprocesses data, builds the model using Keras, trains the model, and then generates a summary for a given input text. The model uses two input sequences, the original text and the current summary, to predict the next summary word. The summary is generated word-by-word until the maximum summary length is reached.

```
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Input, LSTM, Dense, Embedding,
Masking
from tensorflow.keras.optimizers import RMSprop


# Define hyperparameters
max_text_length = 200
max_summary_length = 50
vocab_size = 10000
embedding_size = 100
hidden_size = 300
batch_size = 64
epochs = 50
```

```
# Load data
train_data = load_data('train.csv')
test_data = load_data('test.csv')


# Tokenize text and summary
text_tokenizer = Tokenizer(num_words=vocab_size, oov_token='<OOV>')
text_tokenizer.fit_on_texts(train_data['text'])
text_sequences =
text_tokenizer.texts_to_sequences(train_data['text'])
text_sequences_padded = pad_sequences(text_sequences,
maxlen=max_text_length, padding='post')


summary_tokenizer = Tokenizer(num_words=vocab_size,
oov_token='<OOV>')
summary_tokenizer.fit_on_texts(train_data['summary'])
summary_sequences =
summary_tokenizer.texts_to_sequences(train_data['summary'])
summary_sequences_padded = pad_sequences(summary_sequences,
maxlen=max_summary_length, padding='post')


# Build model
text_input = Input(shape=(max_text_length,), dtype='int32')
embedded_text = Embedding(vocab_size, embedding_size,
input_length=max_text_length)(text_input)
masked_text = Masking(mask_value=0.)(embedded_text)
encoder = LSTM(hidden_size)(masked_text)


summary_input = Input(shape=(max_summary_length,), dtype='int32')
embedded_summary = Embedding(vocab_size, embedding_size,
input_length=max_summary_length)(summary_input)
masked_summary = Masking(mask_value=0.)(embedded_summary)
decoder = LSTM(hidden_size, return_sequences=True)(masked_summary,
initial_state=[encoder, encoder])
```

```
output = Dense(vocab_size, activation='softmax')(decoder)


model = Model([text_input, summary_input], output)
model.compile(loss='categorical_crossentropy',
optimizer=RMSprop(learning_rate=0.001), metrics=['accuracy'])


# Train model
history = model.fit([text_sequences_padded,
summary_sequences_padded], summary_sequences_padded,
                    batch_size=batch_size, epochs=epochs,
validation_split=0.2)


# Generate summary
def generate_summary(text):
    text_sequence = text_tokenizer.texts_to_sequences([text])
    text_sequence_padded = pad_sequences(text_sequence,
maxlen=max_text_length, padding='post')
    summary_sequence = np.zeros((1, max_summary_length))

    for i in range(max_summary_length):
        prediction = model.predict([text_sequence_padded,
summary_sequence])[0]
        summary_sequence[0, i] = np.argmax(prediction[i])

    summary =
summary_tokenizer.sequences_to_texts([summary_sequence[0]])[0]
    return summary
```

Evaluation Metrics for Text Summarization

Evaluation metrics for text summarization are used to measure the quality and effectiveness of generated summaries compared to the original text. The choice of evaluation metrics depends on the type of summarization, such as extractive or abstractive, and the specific application.

For extractive summarization, commonly used evaluation metrics include precision, recall, and F1 score. These metrics are based on comparing the extracted sentences from the original text to the reference summary, which is a human-generated summary. Precision measures the proportion of correctly extracted sentences from the reference summary, while recall measures the proportion of all relevant sentences from the original text that were extracted. The F1 score is the harmonic mean of precision and recall, which provides a single metric to evaluate the performance of the summarization algorithm.

For abstractive summarization, evaluation metrics such as ROUGE (Recall-Oriented Understudy for Gisting Evaluation) and BLEU (Bilingual Evaluation Understudy) are commonly used. ROUGE measures the overlap between the generated summary and the reference summary, using measures such as ROUGE-1 (unigram overlap) and ROUGE-L (longest common subsequence). BLEU measures the similarity between the generated summary and the reference summary based on n-gram matches, and has been widely used in machine translation evaluation.

Other evaluation metrics for text summarization include coherence, readability, and fluency. Coherence measures the logical flow and organization of the summary, while readability measures the ease of comprehension for the target audience. Fluency measures the grammatical correctness and naturalness of the generated summary.

Overall, evaluation metrics for text summarization are important for assessing the performance of summarization algorithms and identifying areas for improvement. However, no single metric can fully capture the quality and effectiveness of a summary, and a combination of metrics should be used to provide a more comprehensive evaluation.