Text Preprocessing

POS tagging

```
import nltk
from nltk.corpus import stopwords

paragraph = ''
sentences = nltk.sent_tokenize(paragraph)

nltk.download('averaged_perceptron_tagger')

for i in range(len(sentences)):
    words = nltk.word_tokenize(sentences[i])
    words = [word for word in words if word not in
set(stopwords.words('english'))]
    pos_tag = nltk.pos_tag(words)
    print(pos_tag)
```

stop word removal

```
from nltk.corpus import stopwords

import nltk
nltk.download('stopwords')

paragraph = ''
sentences = nltk.sent_tokenize(paragraph)
for i in range(len(sentences)):
    words = nltk.word_tokenize(sentences[i])
    words = [stemmer.stem(word) for word in words if word not in
set(stopwords.words('english'))]
    sentences[i] =' '.join(words)
```

Stemming

```
from nltk.stem import PorterStemmer

paragraph = ''
stemmer = PorterStemmer()
```

```
sentences = nltk.sent_tokenize(paragraph)

for i in range(len(sentences)):
    words = nltk.word_tokenize(sentences[i])
    sentences[i]=' '.join(words)
```

One Hot Encoding

```
import nltk
from nltk.tokenize import word_tokenize
from sklearn.preprocessing import OneHotEncoder
import numpy as np

# Sample text
text = "I love cats and I love dogs."

# Tokenize the text into words
nltk.download('punkt')  # Download the punkt tokenizer if you haven't
already
tokens = word_tokenize(text.lower())  # Convert to lowercase and tokenize

# Create a set of unique words (vocabulary)
vocabulary = list(set(tokens))
print("Vocabulary:", vocabulary)

# Reshape the tokens for the OneHotEncoder
tokens_reshaped = np.array(tokens).reshape(8, 1)

# Create the OneHotEncoder instance and fit it to the vocabulary
encoder = OneHotEncoder(sparse_output=False)
encoder.fit(tokens_reshaped)

# Transform the tokens into one-hot encoded vectors
one_hot_encoded = encoder.transform(tokens_reshaped)

# Display the one-hot encoded result
for word, encoded in zip(tokens, one_hot_encoded):
    print(f"{word}: {encoded}")
```

Bag of Words

```
import nltk
from nltk.tokenize import word_tokenize
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer
```

```
# Sample documents
documents = [
    "I love cats.",
    "I love dogs.",
    "Cats are great pets.",
    "Dogs are also great pets."
]

# Initialize CountVectorizer
vectorizer = CountVectorizer(max_features=5, binary=True)

# Fit and transform the documents to get the Bag of Words representation
bow_matrix = vectorizer.fit_transform(documents)

# Convert the sparse matrix to a dense array
bow_array = bow_matrix.toarray()

# Display the Bag of Words matrix
print("Bag of Words representation:")
print(bow_array)

# Get the feature names (vocabulary)
vocabulary = vectorizer.get_feature_names_out()
print("Vocabulary:", vocabulary)
```

N-Gram

```


import nltk
from nltk.util import ngrams
from nltk.tokenize import word_tokenize

# Sample text
text = "I love cats and dogs."

# Tokenize the text into words
nltk.download('punkt')  # Download the punkt tokenizer if you haven't
already
tokens = word_tokenize(text.lower())  # Convert to lowercase and tokenize

# Define the n value for n-grams
n = 2  # Change this value for unigrams (1), trigrams (3), etc.

# Create n-grams
n_grams = list(ngrams(tokens, n))

# Display the n-grams
print(f"{n}-grams:", n_grams)
```

TF-IDF

```
from sklearn.feature_extraction.text import TfidfVectorizer

# Sample documents
documents = [
    "I love cats.",
    "I love dogs.",
    "Cats are great pets.",
    "Dogs are also great pets."
]

# Create a TfidfVectorizer instance
vectorizer = TfidfVectorizer()

# Fit and transform the documents to get the TF-IDF representation
tfidf_matrix = vectorizer.fit_transform(documents)

# Convert the sparse matrix to a dense array
tfidf_array = tfidf_matrix.toarray()

# Display the TF-IDF matrix
print("TF-IDF representation:")
print(tfidf_array)

# Get the feature names (vocabulary)
vocabulary = vectorizer.get_feature_names_out()
print("Vocabulary:", vocabulary)
```

word embeddings

```
# install gensim
%pip install gensim

from gensim.models import KeyedVectors

# Load pre-trained Word2Vec embeddings (e.g., Google News vectors)
# You can download the Google News vectors from
https://code.google.com/archive/p/word2vec/ and load it here
# Make sure to provide the correct path to the model file
model = KeyedVectors.load_word2vec_format('GoogleNews-vectors-
negative300.bin', binary=True)

# Example: Get the vector for a word
vector = model['king']
```

```
# Example: Find similar words
similar_words = model.most_similar('king', topn=5)

print("Vector for 'king':", vector)
print("Most similar words to 'king':", similar_words)
```

Modelling

Simple RNN

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf

# load the data
from tensorflow.keras.datasets import imdb
(input_train, y_train), (input_text, y_test) =
imdb.load_data(num_words=10000)

# pre-processing
from tensorflow.keras.preprocessing import sequence
input_train = sequence.pad_sequences(input_train, maxlen=500)

# train the model
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense, Embedding, SimpleRNN

model = Sequential()
model.add(Embedding(10000, 32))
model.add(SimpleRNN(32))
model.add(Dense(128))
model.add(Dense(64))
model.add(Dense(16))
model.add(Dense(1, activation="sigmoid"))
# model.summary()
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=
['accuracy'])
model.fit(input_train, y_train, epochs=10)

# prediction
from tensorflow.keras.preprocessing.sequence import pad_sequences

reviews = [
    "I really loved this movie! It was fantastic.",
    "This was the worst film I have ever seen."
]

word_index = imdb.get_word_index()
def preprocess_review(review):
    # Tokenize the review
    tokens = review.lower().split()
    sequences = [word_index.get(word, 0) for word in tokens]

    # Pad the sequence
    padded_sequence = pad_sequences([sequences], maxlen=500)
    return padded_sequence

preprocessed_reviews = [preprocess_review(review) for review in reviews]
predictions = [model.predict(review) for review in preprocessed_reviews]
for review, pred in zip(reviews, predictions):
    sentiment = 'Positive' if pred[0][0] > 0.5 else 'Negative'
    print(f'Review: "{review}" => Sentiment: {sentiment} (Probability:
{pred[0][0]:.4f})')
```

LSTM

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf

# load the data
from tensorflow.keras.datasets import imdb
(input_train, y_train), (input_text, y_test) =
imdb.load_data(num_words=10000)

# pre-processing
from tensorflow.keras.preprocessing import sequence
input_train = sequence.pad_sequences(input_train, maxlen=500)

# train the model
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense, Embedding, LSTM

model = Sequential()
model.add(Embedding(10000, 32))
model.add(LSTM(32))
model.add(Dense(128))
model.add(Dense(64))
model.add(Dense(16))
model.add(Dense(1, activation="sigmoid"))
# model.summary()
```

```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=
['accuracy'])
model.fit(input_train, y_train, epochs=10)

# prediction
from tensorflow.keras.preprocessing.sequence import pad_sequences

reviews = [
    "I really loved this movie! It was fantastic.",
    "This was the worst film I have ever seen."
]

word_index = imdb.get_word_index()
def preprocess_review(review):
    # Tokenize the review
    tokens = review.lower().split()
    sequences = [word_index.get(word, 0) for word in tokens]

    # Pad the sequence
    padded_sequence = pad_sequences([sequences], maxlen=500)
    return padded_sequence

preprocessed_reviews = [preprocess_review(review) for review in reviews]
predictions = [model.predict(review) for review in preprocessed_reviews]
for review, pred in zip(reviews, predictions):
    sentiment = 'Positive' if pred[0][0] > 0.5 else 'Negative'
    print(f'Review: "{review}" => Sentiment: {sentiment} (Probability:
{pred[0][0]:.4f})')
```

GRU

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf

# load the data
from tensorflow.keras.datasets import imdb
(input_train, y_train), (input_text, y_test) =
imdb.load_data(num_words=10000)

# pre-processing
from tensorflow.keras.preprocessing import sequence
input_train = sequence.pad_sequences(input_train, maxlen=500)

# train the model
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense, Embedding, GRU

model = Sequential()
model.add(Embedding(10000, 32))
model.add(GRU(32))
model.add(Dense(128))
model.add(Dense(64))
model.add(Dense(16))
model.add(Dense(1, activation="sigmoid"))
# model.summary()
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=
['accuracy'])
model.fit(input_train, y_train, epochs=10)

# prediction
from tensorflow.keras.preprocessing.sequence import pad_sequences

reviews = [
    "I really loved this movie! It was fantastic.",
    "This was the worst film I have ever seen."
]

word_index = imdb.get_word_index()
def preprocess_review(review):
    # Tokenize the review
    tokens = review.lower().split()
    sequences = [word_index.get(word, 0) for word in tokens]

    # Pad the sequence
    padded_sequence = pad_sequences([sequences], maxlen=500)
    return padded_sequence

preprocessed_reviews = [preprocess_review(review) for review in reviews]
predictions = [model.predict(review) for review in preprocessed_reviews]
for review, pred in zip(reviews, predictions):
    sentiment = 'Positive' if pred[0][0] > 0.5 else 'Negative'
    print(f'Review: "{review}" => Sentiment: {sentiment} (Probability:
{pred[0][0]:.4f})')
```

Bidirectional RNN

```

```

Encoder-Decoder architecture

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf

# load the data
from tensorflow.keras.datasets import imdb
(input_train, y_train), (input_text, y_test) =
imdb.load_data(num_words=10000)

# pre-processing
from tensorflow.keras.preprocessing import sequence
input_train = sequence.pad_sequences(input_train, maxlen=500)

# train the model
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, LSTM, Dense, Embedding, Dropout

maxlen = 500

# Encoder
encoder_inputs = Input(shape=(maxlen,))
encoder_embedding = Embedding(input_dim=10000, output_dim=128)
(encoder_inputs)
encoder_lstm = LSTM(128, return_sequences=False, return_state=True)
encoder_outputs, state_h, state_c = encoder_lstm(encoder_embedding)

# Decoder
decoder_inputs = Input(shape=(1,))
decoder_embedding = Embedding(input_dim=10000, output_dim=128)
(decoder_inputs)
decoder_lstm = LSTM(128, return_sequences=False)(decoder_embedding,
initial_state=[state_h, state_c])
decoder_dense = Dense(1, activation='sigmoid')(decoder_lstm)

# Define the model
model = Model([encoder_inputs, decoder_inputs], decoder_dense)

# compile the model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=
['accuracy'], run_eagerly=True)

# Prepare decoder inputs (dummy inputs for classification)
decoder_inputs_train = np.zeros((len(input_train), 1))

# train the model
model.fit([input_train, decoder_inputs_train], y_train, epochs=5)
```

Transformers

```
from transformers import pipeline

# sentiment analysis
sentiment_pipeline = pipeline("sentiment-analysis")
text = "I love apple products"
result = sentiment_pipeline(text)
print(result)

# text generation
text_generation_pipeline = pipeline("text-generation")
text = "Today I started jogging in the morning."
result = text_generation_pipeline(text)
print(result)

# summarization
summarization_pipeline = pipeline("summarization")
text = "Apple’s latest iPhone 16 Pro ad ends with the bold tagline,
“Hollywood in your pocket,” highlighting its advanced filmmaking
capabilities. For years, Apple has been the dominant force in smartphone
videography, and while Android competitors like Samsung and Google have
made strides, the competition is still mostly one-sided. The iPhone 15 Pro
reinforced Apple’s lead with the introduction of “pro-grade” features like
shooting in ProRes LOG up to 4K at 60FPS. Now, the iPhone 16 Pro raises
the bar even higher, offering 4K recording at 120FPS, enabling users to
capture smooth, cinematic slow-motion without compromising on resolution.
But here’s the real question: does owning the latest iPhone make you an
instant cinematographer? The answer is both yes and no, and the reality is
more nuanced than it first appears. Keep reading to find out.iPhone 15 Pro
was used to film several Apple keynotes last year, and the iPhone 16 Pro
takes those capabilities even further.iPhone 15 Pro was used to film
several Apple keynotes last year, and the iPhone 16 Pro takes those
capabilities even further."
result = summarization_pipeline(text)
print(result)

# question answering
question_answering_pipeline = pipeline("question-answering")
context = (
        "Hugging Face is creating a tool that democratizes AI. "
        "The company provides open-source libraries and models for natural
language processing."
)
text = "what is hugging face used for"
result = question_answering_pipeline(text, context)
print(result)
```