WORKSHOP ON INTRODUCTION TO NLP (NATURAL LANGUAGE PROCESSING) AND GENERATIVE AI

Organized by

Department of Computer Science

Hajee Karutha Rowther Howdia College

Uthamapalayam

Tamil Nadu-India

Presenter

Dr.A.Saleem Raja

Faculty/IT Department, College of Computing and Information Sciences,

University of Technology and Applied Sciences-Shinas

Sultanate of Oman

Tuesday, 14 August 2024

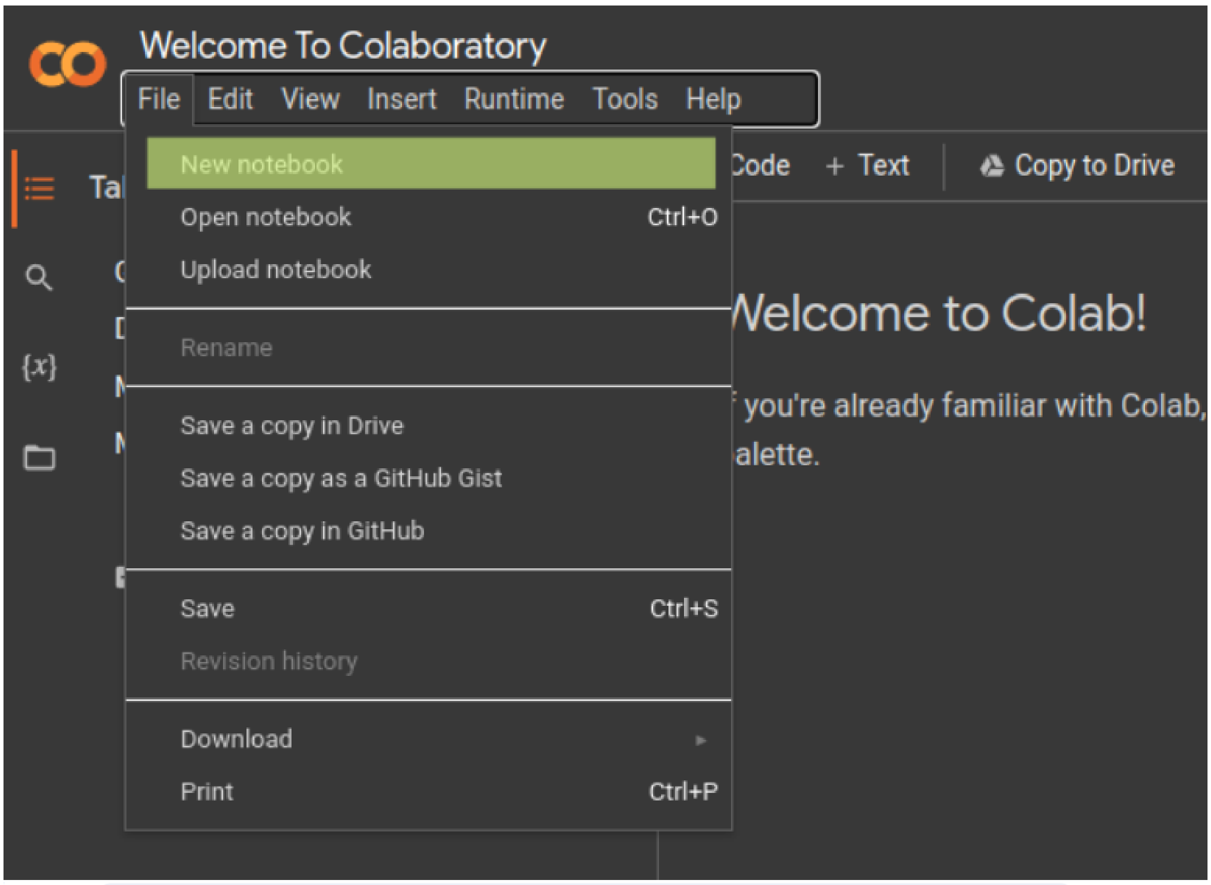
What is Google Colab?

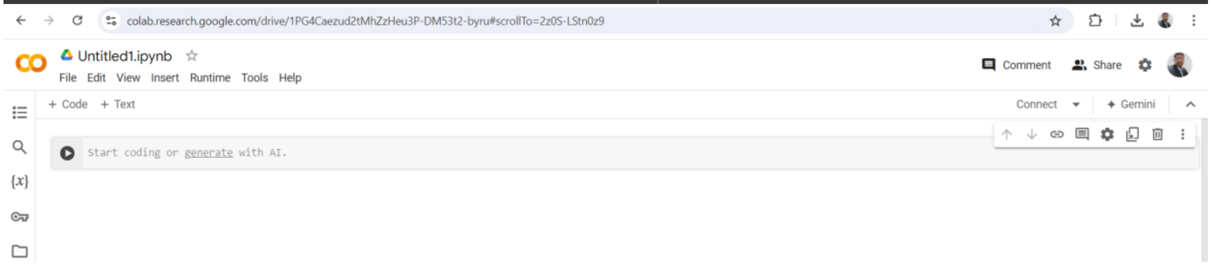
Google Colab, short for Colaboratory, is a free cloud-based platform provided by Google that allows users to write and execute Python code collaboratively in a Jupyter Notebook environment. Google Collaboratory notebook, is designed to facilitate machine learning (ML) and data science tasks by providing a virtual environment, Google colab python with access to free GPU resources [1].

Work with Google Colab

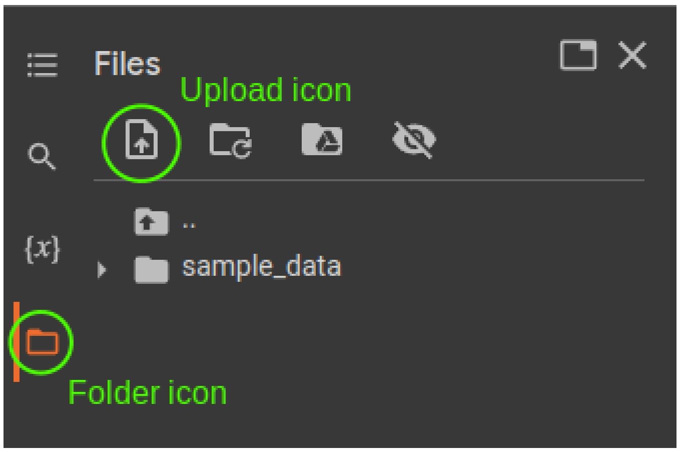
1. Open google colab using below link and sign in with your Google account. https://colab.research.google.com/

2. Create a New Notebook: Once you're on the Google Colab interface, click on File > New notebook to create a new notebook.




colab.research.google.com/drive/1PG4Caezud2tMhZzHeu3P-DM53t2-byru#scrollTo=2z0S-LStn0z9
Untitled1.ipynb
File Edit View Insert Runtime Tools Help
Comment
Share
+ Code + Text
Connect
Gemini
Start coding or generate with AI.

3. Use the File Browser: On the left-hand side, click on the folder icon to open the file browser. You can upload files by clicking on the upload icon.



4. The hotkeys on Colab and that on Jupyter notebooks are similar. These are some of the useful ones [2]:
   - Run cell: Ctrl + Enter
   - Run cell and add new cell below: Alt + Enter
   - Run cell and goto cell below: Shift + Enter
   - Indent line by two spaces: Ctrl +]
   - Unindent line by two spaces: Ctrl + [

References

1. https://www.geeksforgeeks.org/how-to-use-google-colab/
2. https://machinelearningmastery.com/google-colab-for-machine-learning-projects/

Presented Information

Dr.A.Saleem Raja MCA.,M.Phil.,M.Tech.,Ph.D.,

- Cisco Certified Network Associate,
- Cisco Certified Academy Instructor,
- Huawei Certified Academy Instructor,
- Certifier Ethical Hacker

Faculty, IT Department,

College of Computing and Information Sciences,

University of Technology and Applied Sciences-Shinas,

Sultanate of Oman

Mobile: 00968-96113496, Email: asaleemrajasec@gmail.com

Google Scholar: https://scholar.google.com/citations?user=CXfdHt0AAAAJ&hl=en

Orcid ID: https://orcid.org/0000-0002-7203-1426

Scopus ID: https://www.scopus.com/authid/detail.uri?authorId=58187646200

Segmentation

```
import nltk
from nltk.tokenize import sent_tokenize, word_tokenize

# Download the necessary resources from NLTK
nltk.download('punkt')

# Sample text for segmentation
text = """Natural Language Processing (NLP) is a sub-field of artificial intelligence (AI).
It focuses on the interaction between computers and humans through natural language.
The ultimate objective of NLP is to enable computers to understand, interpret, and generate human language."""

# Sentence Segmentation
sentences = sent_tokenize(text)
print("Sentence Segmentation:")
for i, sentence in enumerate(sentences):
    print(f"Sentence {i+1}: {sentence}")
```

```
Sentence Segmentation:
Sentence 1: Natural Language Processing (NLP) is a sub-field of artificial intelligence (AI).
Sentence 2: It focuses on the interaction between computers and humans through natural language.
Sentence 3: The ultimate objective of NLP is to enable computers to understand, interpret, and generate human language.
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

Tokenization

```
# Word tokenization
print("\nWord Tokenization:")
for i, sentence in enumerate(sentences):
    words = nltk.word_tokenize(sentence)
    print(f"Words in Sentence {i+1}: {words}")
```

```

Word Tokenization:
Words in Sentence 1: ['Natural', 'Language', 'Processing', '(', 'NLP', ')', 'is', 'a', 'sub-field', 'of', 'artificial', 'intelligen
Words in Sentence 2: ['It', 'focuses', 'on', 'the', 'interaction', 'between', 'computers', 'and', 'humans', 'through', 'natural', 'l
Words in Sentence 3: ['The', 'ultimate', 'objective', 'of', 'NLP', 'is', 'to', 'enable', 'computers', 'to', 'understand', ',', 'inte
```

Stemming

```
# Importing the necessary libraries
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize

# Initialize the Porter Stemmer
stemmer = PorterStemmer()

# Sample text
text = "running runs easily cared caring"

# Tokenize the text into words
words = word_tokenize(text)

# Apply stemming to each word
stemmed_words = [stemmer.stem(word) for word in words]

# Print the results
print("Original Words:", words)
print("Stemmed Words:", stemmed_words)
```

```
Original Words: ['running', 'runs', 'easily', 'cared', 'caring']
Stemmed Words: ['run', 'run', 'easili', 'care', 'care']
```

Lemmatization

```
import spacy

# Load the spaCy model
nlp = spacy.load('en_core_web_sm')

# Define a function for lemmatization
def spacy_lemmatize(text):
    doc = nlp(text)
    return [token.lemma_ for token in doc]

# Example usage
text = "running runs easily cared caring"
lemmatized_words = spacy_lemmatize(text)
print(lemmatized_words)
```

```
['run', 'run', 'easily', 'care', 'care']
```

Stop word removal

```
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

# Download the stopwords
nltk.download('stopwords')
#nltk.download('punkt')

# Initialize the stop words
stop_words = set(stopwords.words('english'))

# Example text
text = "This is an example sentence, showing off the stop words filtration."

# Tokenize the text
word_tokens = word_tokenize(text)

# Remove stop words
filtered_sentence = [word for word in word_tokens if word.lower() not in stop_words]

print(filtered_sentence)
```

```
['example', 'sentence', ',', 'showing', 'stop', 'words', 'filtration', '.']
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

```
import spacy

# Load the spaCy model
nlp = spacy.load('en_core_web_sm')

# Example text
text = "This is an example sentence, showing off the stop words filtration."

# Process the text
doc = nlp(text)

# Remove stop words
filtered_sentence = [token.text for token in doc if not token.is_stop]

print(filtered_sentence)
```

```
['example', 'sentence', ',', 'showing', 'stop', 'words', 'filtration', '.']
```

Character Level Embedding

```
import numpy as np

def text_to_one_hot(text):
    vocab = sorted(set(text))  # Get unique characters
    char_to_idx = {char: idx for idx, char in enumerate(vocab)}
    one_hot_vectors = np.eye(len(vocab))[list(map(char_to_idx.get, text))]
    return char_to_idx, one_hot_vectors

# Example usage
text = "hello world"
char_to_idx, one_hot_vectors = text_to_one_hot(text)

print("Vocabulary:", char_to_idx)
print("One-Hot Encoded Vectors:\n", one_hot_vectors)
```

```
Vocabulary: {' ': 0, 'd': 1, 'e': 2, 'h': 3, 'l': 4, 'o': 5, 'r': 6, 'w': 7}
One-Hot Encoded Vectors:
 [[0. 0. 0. 1. 0. 0. 0. 0.]
 [0. 0. 1. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0. 0. 0.]
 [0. 0. 0. 0. 0. 1. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 0. 1. 0. 0.]
 [0. 0. 0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 1. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0.]]
```

Bag of Words or Count Vectorizer

```
from sklearn.feature_extraction.text import CountVectorizer

# Sample documents
documents = [
    "This is the first document.",
    "This document is the second document.",
    "And this is the third one.",
    "Is this the first document?"
]

# Initialize the CountVectorizer
vectorizer = CountVectorizer()

# Fit and transform the documents to BoW representation
bow_matrix = vectorizer.fit_transform(documents)

# Get the feature names (words)
feature_names = vectorizer.get_feature_names_out()

# Convert the BoW matrix to an array for easy viewing
bow_array = bow_matrix.toarray()

# Display the results
print("Feature Names (Vocabulary):\n", feature_names)
print("\nBoW Matrix:\n", bow_array)
```

```
Feature Names (Vocabulary):
 ['and' 'document' 'first' 'is' 'one' 'second' 'the' 'third' 'this']

BoW Matrix:
 [[0 1 1 1 0 0 1 0 1]
 [0 2 0 1 0 1 1 0 1]
 [1 0 0 1 1 0 1 1 1]
 [0 1 1 1 0 0 1 0 1]]
```

TF-IDF Vectorizer

```
from sklearn.feature_extraction.text import TfidfVectorizer

# Sample documents
documents = [
    "This is the first document.",
    "This document is the second document.",
    "And this is the third one.",
    "Is this the first document?"
]

# Initialize the TF-IDF Vectorizer
vectorizer = TfidfVectorizer()

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

# Get feature names
feature_names = vectorizer.get_feature_names_out()

# Convert to dense array for better readability
dense_array = tfidf_matrix.todense()

# Print the TF-IDF matrix
print("Feature Names:", feature_names)
print("TF-IDF Matrix:\n", dense_array)
```

```
Feature Names: ['and' 'document' 'first' 'is' 'one' 'second' 'the' 'third' 'this']
TF-IDF Matrix:
 [[0.         0.46979139 0.58028582 0.38408524 0.         0.
  0.38408524 0.         0.38408524]
 [0.         0.6876236  0.         0.28108867 0.         0.53864762
  0.28108867 0.         0.28108867]
 [0.51184851 0.         0.         0.26710379 0.51184851 0.
  0.26710379 0.51184851 0.26710379]
 [0.         0.46979139 0.58028582 0.38408524 0.         0.
  0.38408524 0.         0.38408524]]
```

N-gram with Count Vectorizer

```
from sklearn.feature_extraction.text import CountVectorizer

# Sample documents
documents = [
    "This is the first document.",
    "This document is the second document.",
    "And this is the third one.",
    "Is this the first document?"
]

# Initialize the Count Vectorizer with n-gram range for bigrams (2-grams)
vectorizer = CountVectorizer(ngram_range=(2, 2))

# Fit and transform the documents
ngram_matrix = vectorizer.fit_transform(documents)

# Get feature names (bigrams)
feature_names = vectorizer.get_feature_names_out()

# Convert to dense array for better readability
dense_array = ngram_matrix.todense()

# Print the bigram feature names and matrix
print("Bigram Feature Names:", feature_names)
print("Bigram Count Matrix:\n", dense_array)
```

```
Bigram Feature Names: ['and this' 'document is' 'first document' 'is the' 'is this'
 'second document' 'the first' 'the second' 'the third' 'third one'
 'this document' 'this is' 'this the']
Bigram Count Matrix:
 [[0 0 1 1 0 0 1 0 0 0 0 1 0]
 [0 1 0 1 0 1 0 1 0 0 1 0 0]
 [1 0 0 1 0 0 0 0 1 1 0 1 0]
 [0 0 1 0 1 0 1 0 0 0 0 0 1]]
```

Word Embedding

```python
from gensim.models import Word2Vec
from gensim.utils import simple_preprocess

# Sample text data
sentences = [
    "The quick brown fox jumps over the lazy dog.",
    "Natural language processing involves understanding and generating human language.",
    "Machine learning models can be used to analyze large amounts of text data.",
    "Word embeddings help in capturing the semantic meaning of words.",
    "The cat sat on the mat and watched the world outside.",
    "Deep learning techniques are often employed in modern NLP applications.",
    "The weather today is sunny and warm, perfect for a walk in the park.",
    "Artificial intelligence is transforming many industries, including healthcare.",
    "Text classification can be used to categorize documents into different topics.",
    "The financial market is influenced by a variety of economic factors and trends."
]


# Preprocess text: tokenize and clean
def preprocess_text(text):
    return simple_preprocess(text, deacc=True)  # Tokenize and remove punctuation

# Create list of tokenized sentences
tokenized_sentences = [preprocess_text(sentence) for sentence in sentences]

# Initialize and train the Word2Vec model
model = Word2Vec(sentences=tokenized_sentences, vector_size=100, window=5, min_count=1, sg=0)

# Save the model
model.save("word2vec.model")

# Load the model (for demonstration purposes)
model = Word2Vec.load("word2vec.model")

# Retrieve vector for a specific word
word_vector = model.wv['fox']  # Replace 'fox' with any word in the vocabulary
print(f"Vector for 'fox':\n{word_vector}")
```

```
Vector for 'fox':
[ 0.00964945  0.0073213   0.00126067 -0.00340082 -0.00045071  0.00042016
 -0.00640356  0.00574766  0.00236941  0.00377386 -0.00725581  0.00852384
  0.00050821 -0.00020378 -0.0090677   0.00405151  0.00676771  0.00735438
 -0.00641844 -0.0078644  -0.00552221 -0.00059865 -0.0083388  -0.00824333
 -0.00191341  0.00113763 -0.00950915 -0.00373417  0.00064255  0.00681259
  0.00173903 -0.00062908 -0.00747372 -0.00675038 -0.00069807  0.00747163
  0.00544178 -0.00148573  0.00117139 -0.00961065 -0.00137902 -0.00462999
  0.00581255 -0.0023367  -0.00476188 -0.00947624 -0.00120633 -0.00719842
 -0.00168181 -0.00407126 -0.00237292 -0.00324923 -0.00815889 -0.00124956
  0.00168666 -0.00404628 -0.00763635 -0.0035872  -0.00904696 -0.00075365
  0.00588639 -0.0029606   0.00316343  0.00499701  0.00846423  0.00562427
  0.00950492 -0.00964354 -0.00796523 -0.00675872 -0.0074708  -0.00796421
 -0.00778837 -0.00294116  0.00140502 -0.00288235 -0.00881478  0.00498468
  0.00089823  0.00458875  0.00719684  0.00764926 -0.00080829  0.00366422
 -0.00512551  0.00191182  0.00453904  0.00988942 -0.00318535  0.00284082
 -0.00572644 -0.0022101   0.00812467 -0.00390112 -0.0011866  -0.0092857
 -0.00947755  0.00887981 -0.00570017  0.00504886]
```

CASE STUDY -1 (Social media comments classification)

Word2Vec Embeedings

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
from gensim.models import Word2Vec
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import nltk
from sklearn.preprocessing import LabelEncoder

# Ensure you have downloaded the necessary NLTK resources
nltk.download('punkt')
nltk.download('stopwords')
```

```
# Sample dataset of 100 social media comments with "Happy" or "Sad" labels
# For demonstration purposes, a subset is shown here. You should replace it with a full dataset of 100 records.
data = {
    'comment': [
        "I love this! It's fantastic and works perfectly.",
        "Terrible experience. I hate it.",
        "The product is okay, not great but not terrible either.",
        "Absolutely amazing! Will buy again.",
        "This is the worst purchase I’ve ever made.",
        "I really enjoyed using this. It was very helpful.",
        "Not good, very disappointed with the quality.",
        "Excellent service and support. Highly recommend!",
        "It didn't meet my expectations. Not satisfied.",
        "Great value for money. Very pleased with the purchase."
        # Add more comments up to 100 records
    ] * 10,  # Repeated for simplicity
    'label': ['Happy', 'Sad', 'Neutral', 'Happy', 'Sad', 'Happy', 'Sad', 'Happy', 'Sad', 'Happy'] * 10  # Repeated for simplicity
}

# Convert to DataFrame
df = pd.DataFrame(data)

# Filter to only "Happy" or "Sad" labels
df = df[df['label'].isin(['Happy', 'Sad'])]

# Preprocess the comments
def preprocess(text):
    tokens = word_tokenize(text.lower())
    stop_words = set(stopwords.words('english'))
    tokens = [word for word in tokens if word.isalpha() and word not in stop_words]
    return tokens

df['processed'] = df['comment'].apply(preprocess)

# Train Word2Vec model
word2vec_model = Word2Vec(sentences=df['processed'], vector_size=100, window=5, min_count=1, sg=0)

# Create feature vectors for each comment
def get_feature_vector(tokens):
    vectors = [word2vec_model.wv[word] for word in tokens if word in word2vec_model.wv]
    if len(vectors) == 0:
        return np.zeros(word2vec_model.vector_size)
    return np.mean(vectors, axis=0)

df['feature_vector'] = df['processed'].apply(get_feature_vector)

# Prepare data for classification
X = np.array(df['feature_vector'].tolist())
y = df['label']

# Encode labels as numbers
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)

# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, test_size=0.3, random_state=42)

# Train Random Forest classifier
clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train, y_train)

# Predict and evaluate
y_pred = clf.predict(X_test)

# Get target names that were actually predicted
unique_labels = np.unique(np.concatenate((y_test, y_pred)))
target_names = label_encoder.inverse_transform(unique_labels)

# Generate classification report with zero_division=0 to handle undefined metrics
print(classification_report(y_test, y_pred, target_names=target_names, zero_division=0))
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
              precision    recall  f1-score   support

       Happy       1.00      1.00      1.00        17
         Sad       1.00      1.00      1.00        10

    accuracy                           1.00        27
   macro avg       1.00      1.00      1.00        27
```

```
weighted avg       1.00      1.00      1.00        27
```

Transfer Learning

```
import pandas as pd
import numpy as np
import gensim.downloader as api
from gensim.models import KeyedVectors
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
from sklearn.preprocessing import LabelEncoder
from sklearn.feature_extraction.text import CountVectorizer

# Load pre-trained Word2Vec model
print("Loading pre-trained Word2Vec model...")
wv = api.load('word2vec-google-news-300')

# Sample dataset of social media comments and their labels
data = {
    'comment': [
        "I love this product, it's amazing!",
        "This is the best thing I've ever bought!",
        "I am so happy with my purchase.",
        "This was a terrible experience.",
        "I hate this product, it's awful.",
        "I'm really disappointed with this purchase.",
        "Fantastic! I'm thrilled with the result.",
        "This was a waste of money.",
        "Absolutely horrible, would not recommend.",
        "I'm so pleased with the service.",
        "I will never buy this again.",
        "This is perfect, I'm so satisfied.",
        "I'm not happy with this at all.",
        "The product is great, I'm very happy.",
        "This did not meet my expectations.",
        "Really happy with how this turned out.",
        "The worst product I've ever bought.",
        "This is exactly what I needed.",
        "I'm very unhappy with this purchase.",
        "What a great buy! I'm so happy.",
        # Add more entries to make up a total of 100 records
        # Here, for demonstration, I'll repeat the pattern.
    ] * 10,  # Repeat the sample 10 times to get 100 records
    'label': [
        'Happy', 'Happy', 'Happy', 'Sad', 'Sad', 'Sad',
        'Happy', 'Sad', 'Sad', 'Happy',
        'Sad', 'Happy', 'Sad', 'Happy', 'Sad',
        'Happy', 'Sad', 'Happy', 'Sad', 'Happy',
        # Repeat the labels pattern to match 100 records
    ] * 10
}

# Convert to DataFrame
df = pd.DataFrame(data)

# Function to convert comments to Word2Vec embeddings
def comment_to_vec(comment, model):
    words = comment.split()
    word_vecs = [model[word] for word in words if word in model]
    if len(word_vecs) == 0:
        return np.zeros(model.vector_size)
    return np.mean(word_vecs, axis=0)

# Convert comments to Word2Vec vectors
X = np.array([comment_to_vec(comment, wv) for comment in df['comment']])

# Prepare target variable
y = df['label']

# Encode labels as numbers
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)

# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, test_size=0.3, random_state=42)

# Train Logistic Regression classifier
clf = LogisticRegression(max_iter=1000)
clf.fit(X_train, y_train)
```

```
# Predict and evaluate
y_pred = clf.predict(X_test)

# Get target names that were actually predicted
unique_labels = np.unique(np.concatenate((y_test, y_pred)))
target_names = label_encoder.inverse_transform(unique_labels)

# Generate classification report with zero_division=0 to handle undefined metrics
print(classification_report(y_test, y_pred, target_names=target_names, zero_division=0))
```

```
Loading pre-trained Word2Vec model...
[==================================================] 100.0% 1662.8/1662.8MB downloaded
              precision    recall  f1-score   support

       Happy       1.00      0.75      0.86        28
         Sad       0.82      1.00      0.90        32

    accuracy                           0.88        60
   macro avg       0.91      0.88      0.88        60
weighted avg       0.90      0.88      0.88        60
```

Transformer

```
!pip install numpy==1.26.4
```

```
!pip install transformers
```

```
#sentiment-analysis
from transformers import pipeline
sentiment_pipeline = pipeline("sentiment-analysis")
data = ["I love you", "I hate you"]
sentiment_pipeline(data)
```

```
No model was supplied, defaulted to distilbert/distilbert-base-uncased-finetuned-sst-2-english and revision af0f99b (https://huggin
Using a pipeline without specifying a model name and revision in production is not recommended.
[{'label': 'POSITIVE', 'score': 0.9998656511306763},
 {'label': 'NEGATIVE', 'score': 0.9991129040718079}]
```

```
sentiment_pipeline(
    ["I've been waiting for a HuggingFace course my whole life.", "I hate this so much!"]
)
```

```
[{'label': 'POSITIVE', 'score': 0.9598048329353333},
 {'label': 'NEGATIVE', 'score': 0.9994558691978455}]
```

```
from transformers import pipeline

generator = pipeline("text-generation")
generator("In this course, we will teach you how to")
```

```
No model was supplied, defaulted to openai-community/gpt2 and revision 6c0e608 (https://huggingface.co/openai-community/gpt2).
Using a pipeline without specifying a model name and revision in production is not recommended.
config.json: 100%                                   665/665 [00:00<00:00, 7.63kB/s]
model.safetensors: 100%                             548M/548M [00:07<00:00, 87.3MB/s]
generation_config.json: 100%                        124/124 [00:00<00:00, 7.74kB/s]
tokenizer_config.json: 100%                         26.0/26.0 [00:00<00:00, 1.48kB/s]
vocab.json: 100%                                    1.04M/1.04M [00:00<00:00, 4.55MB/s]
merges.txt: 100%                                    456k/456k [00:00<00:00, 17.9MB/s]
tokenizer.json: 100%                                1.36M/1.36M [00:00<00:00, 4.80MB/s]
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
[{'generated_text': 'In this course, we will teach you how to build your own game software or an application for a computer. You
```

```python
#Text Genereation
from transformers import pipeline

generator = pipeline("text-generation", model="distilgpt2")
generator(
    "In this course, we will teach you how to",
    max_length=30,
    num_return_sequences=2,
)
```

```
config.json:  100%                                        762/762  [00:00<00:00,  11.8kB/s]
model.safetensors:  100%                                  353M/353M  [00:03<00:00,  102MB/s]
generation_config.json:  100%                             124/124  [00:00<00:00,  1.63kB/s]
tokenizer_config.json:  100%                              26.0/26.0  [00:00<00:00,  396B/s]
vocab.json:  100%                                         1.04M/1.04M  [00:00<00:00,  2.98MB/s]
merges.txt:  100%                                         456k/456k  [00:00<00:00,  4.26MB/s]
tokenizer.json:  100%                                     1.36M/1.36M  [00:00<00:00,  10.4MB/s]
Truncation was not explicitly activated but `max_length` is provided a specific value, please use `truncation=True` to explicitly tr
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
[{'generated_text': 'In this course, we will teach you how to use the \u202a-word\u202a in order to learn how to use the \u202a-
word'},
 {'generated_text': "In this course, we will teach you how to read, understand and use these skills in the classroom. You'll learn
```

```python
#Question answering
from transformers import pipeline

question_answerer = pipeline("question-answering")
question_answerer(
    question="Where do I work?",
    context="My name is Sylvain and I work at Hugging Face in Brooklyn",
)
```

```
No model was supplied, defaulted to distilbert/distilbert-base-cased-distilled-squad and revision 626af31 (https://huggingface.co/d
Using a pipeline without specifying a model name and revision in production is not recommended.
config.json:  100%                                        473/473  [00:00<00:00,  8.02kB/s]
model.safetensors:  100%                                  261M/261M  [00:02<00:00,  125MB/s]
tokenizer_config.json:  100%                              49.0/49.0  [00:00<00:00,  1.40kB/s]
vocab.txt:  100%                                          213k/213k  [00:00<00:00,  1.92MB/s]
tokenizer.json:  100%                                     436k/436k  [00:00<00:00,  6.92MB/s]
{'score': 0.6949766278266907, 'start': 33, 'end': 45, 'answer': 'Hugging Face'}
```

```python
from transformers import pipeline

summarizer = pipeline("summarization")
summarizer(
    """
    America has changed dramatically during recent years. Not only has the number of
    graduates in traditional engineering disciplines such as mechanical, civil,
    electrical, chemical, and aeronautical engineering declined, but in most of
    the premier American universities engineering curricula now concentrate on
    and encourage largely the study of engineering science. As a result, there
    are declining offerings in engineering subjects dealing with infrastructure,
    the environment, and related issues, and greater concentration on high
    technology subjects, largely supporting increasingly complex scientific
    developments. While the latter is important, it should not be at the expense
    of more traditional engineering.

    Rapidly developing economies such as China and India, as well as other
    industrial countries in Europe and Asia, continue to encourage and advance
    the teaching of engineering. Both China and India, respectively, graduate
    six and eight times as many traditional engineers as does the United States.
    Other industrial countries at minimum maintain their output, while America
    suffers an increasingly serious decline in the number of engineering graduates
    and a lack of well-educated engineers.
"""
)
```

```
No model was supplied, defaulted to sshleifer/distilbart-cnn-12-6 and revision a4f8f3e (https://huggingface.co/sshleifer/distilbart
Using a pipeline without specifying a model name and revision in production is not recommended.
config.json:  100%                                            1.80k/1.80k  [00:00<00:00,  39.0kB/s]
pytorch_model.bin:  100%                                      1.22G/1.22G  [00:14<00:00,  19.2MB/s]
tokenizer_config.json:  100%                                  26.0/26.0  [00:00<00:00,  1.34kB/s]
vocab.json:  100%                                             899k/899k  [00:00<00:00,  3.93MB/s]
merges.txt:  100%                                             456k/456k  [00:00<00:00,  17.4MB/s]
[{'summary_text': ' The number of engineering graduates in the United States has declined in recent years . China and India
graduate six and eight times as many traditional engineers as the U.S. does . Rapidly developing economies such as China continue
```

```
#Translation

from transformers import pipeline

translator = pipeline("translation", model="Hemanth-thunder/english-tamil-mt")
translator("Hardwork never fail", src_lang="en", tgt_lang="ta")
```

```
[{'translation_text': 'கடின உழைப்பு ஒருபோதும் தோல்வியை ஏற்படுத்தவில்லை.'}]
```

```
#Translation

from transformers import pipeline

translator = pipeline("translation", model="Helsinki-NLP/opus-mt-fr-en")
translator("Ce cours est produit par Hugging Face.")
```

```
config.json:  100%                                            1.42k/1.42k  [00:00<00:00,  20.7kB/s]
pytorch_model.bin:  100%                                      301M/301M  [00:02<00:00,  131MB/s]
generation_config.json:  100%                                 293/293  [00:00<00:00,  5.32kB/s]
tokenizer_config.json:  100%                                  42.0/42.0  [00:00<00:00,  585B/s]
source.spm:  100%                                             802k/802k  [00:00<00:00,  15.1MB/s]
target.spm:  100%                                             778k/778k  [00:00<00:00,  3.45MB/s]
vocab.json:  100%                                             1.34M/1.34M  [00:00<00:00,  12.6MB/s]
/usr/local/lib/python3.10/dist-packages/transformers/models/marian/tokenization_marian.py:175: UserWarning: Recommended: pip instal
  warnings.warn("Recommended: pip install sacremoses.")
```

```
!pip install google-generativeai
!pip install python-dotenv
```

```
Requirement already satisfied: google-generativeai in /usr/local/lib/python3.10/dist-packages (0.7.2)
Requirement already satisfied: google-ai-generativelanguage==0.6.6 in /usr/local/lib/python3.10/dist-packages (from google-generati
Requirement already satisfied: google-api-core in /usr/local/lib/python3.10/dist-packages (from google-generativeai) (2.19.1)
Requirement already satisfied: google-api-python-client in /usr/local/lib/python3.10/dist-packages (from google-generativeai) (2.13
Requirement already satisfied: google-auth>=2.15.0 in /usr/local/lib/python3.10/dist-packages (from google-generativeai) (2.27.0)
Requirement already satisfied: protobuf in /usr/local/lib/python3.10/dist-packages (from google-generativeai) (3.20.3)
Requirement already satisfied: pydantic in /usr/local/lib/python3.10/dist-packages (from google-generativeai) (2.8.2)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from google-generativeai) (4.66.5)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.10/dist-packages (from google-generativeai) (4.12.2)
Requirement already satisfied: proto-plus<2.0.0dev,>=1.22.3 in /usr/local/lib/python3.10/dist-packages (from google-ai-generativela
Requirement already satisfied: googleapis-common-protos<2.0.dev0,>=1.56.2 in /usr/local/lib/python3.10/dist-packages (from google-a
Requirement already satisfied: requests<3.0.0.dev0,>=2.18.0 in /usr/local/lib/python3.10/dist-packages (from google-api-core->googl
Requirement already satisfied: cachetools<6.0,>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from google-auth>=2.15.0->google-
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from google-auth>=2.15.0->google-g
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.10/dist-packages (from google-auth>=2.15.0->google-generativ
Requirement already satisfied: httplib2<1.dev0,>=0.19.0 in /usr/local/lib/python3.10/dist-packages (from google-api-python-client->
Requirement already satisfied: google-auth-httplib2<1.0.0,>=0.2.0 in /usr/local/lib/python3.10/dist-packages (from google-api-pytho
Requirement already satisfied: uritemplate<5,>=3.0.1 in /usr/local/lib/python3.10/dist-packages (from google-api-python-client->goo
Requirement already satisfied: annotated-types>=0.4.0 in /usr/local/lib/python3.10/dist-packages (from pydantic->google-generativea
Requirement already satisfied: pydantic-core==2.20.1 in /usr/local/lib/python3.10/dist-packages (from pydantic->google-generativeai
Requirement already satisfied: grpcio<2.0dev,>=1.33.2 in /usr/local/lib/python3.10/dist-packages (from google-api-core[grpc]!=2.0.*
Requirement already satisfied: grpcio-status<2.0.dev0,>=1.33.2 in /usr/local/lib/python3.10/dist-packages (from google-api-core[grp
Requirement already satisfied: pyparsing!=3.0.0,!=3.0.1,!=3.0.2,!=3.0.3,<4,>=2.4.2 in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: pyasn1<0.7.0,>=0.4.6 in /usr/local/lib/python3.10/dist-packages (from pyasn1-modules>=0.2.1->google-
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests<3.0.0.dev0,>=2.18
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests<3.0.0.dev0,>=2.18.0->google-a
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests<3.0.0.dev0,>=2.18.0->go
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests<3.0.0.dev0,>=2.18.0->go
Collecting python-dotenv
  Downloading python_dotenv-1.0.1-py3-none-any.whl.metadata (23 kB)
Downloading python_dotenv-1.0.1-py3-none-any.whl (19 kB)
Installing collected packages: python-dotenv
Successfully installed python-dotenv-1.0.1
```

```
!pip install streamlit
```

```
Collecting streamlit
  Downloading streamlit-1.37.1-py2.py3-none-any.whl.metadata (8.5 kB)
Requirement already satisfied: altair<6,>=4.0 in /usr/local/lib/python3.10/dist-packages (from streamlit) (4.2.2)
Requirement already satisfied: blinker<2,>=1.0.0 in /usr/lib/python3/dist-packages (from streamlit) (1.4)
Requirement already satisfied: cachetools<6,>=4.0 in /usr/local/lib/python3.10/dist-packages (from streamlit) (5.4.0)
Requirement already satisfied: click<9,>=7.0 in /usr/local/lib/python3.10/dist-packages (from streamlit) (8.1.7)
Requirement already satisfied: numpy<3,>=1.20 in /usr/local/lib/python3.10/dist-packages (from streamlit) (1.26.4)
Requirement already satisfied: packaging<25,>=20 in /usr/local/lib/python3.10/dist-packages (from streamlit) (24.1)
Requirement already satisfied: pandas<3,>=1.3.0 in /usr/local/lib/python3.10/dist-packages (from streamlit) (2.1.4)
Requirement already satisfied: pillow<11,>=7.1.0 in /usr/local/lib/python3.10/dist-packages (from streamlit) (9.4.0)
Requirement already satisfied: protobuf<6,>=3.20 in /usr/local/lib/python3.10/dist-packages (from streamlit) (3.20.3)
Requirement already satisfied: pyarrow>=7.0 in /usr/local/lib/python3.10/dist-packages (from streamlit) (14.0.2)
Requirement already satisfied: requests<3,>=2.27 in /usr/local/lib/python3.10/dist-packages (from streamlit) (2.32.3)
Requirement already satisfied: rich<14,>=10.14.0 in /usr/local/lib/python3.10/dist-packages (from streamlit) (13.7.1)
Collecting tenacity<9,>=8.1.0 (from streamlit)
  Downloading tenacity-8.5.0-py3-none-any.whl.metadata (1.2 kB)
Requirement already satisfied: toml<2,>=0.10.1 in /usr/local/lib/python3.10/dist-packages (from streamlit) (0.10.2)
Requirement already satisfied: typing-extensions<5,>=4.3.0 in /usr/local/lib/python3.10/dist-packages (from streamlit) (4.12.2)
Collecting gitpython!=3.1.19,<4,>=3.0.7 (from streamlit)
  Downloading GitPython-3.1.43-py3-none-any.whl.metadata (13 kB)
Collecting pydeck<1,>=0.8.0b4 (from streamlit)
  Downloading pydeck-0.9.1-py2.py3-none-any.whl.metadata (4.1 kB)
Requirement already satisfied: tornado<7,>=6.0.3 in /usr/local/lib/python3.10/dist-packages (from streamlit) (6.3.3)
Collecting watchdog<5,>=2.1.5 (from streamlit)
  Downloading watchdog-4.0.2-py3-none-manylinux2014_x86_64.whl.metadata (38 kB)
Requirement already satisfied: entrypoints in /usr/local/lib/python3.10/dist-packages (from altair<6,>=4.0->streamlit) (0.4)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from altair<6,>=4.0->streamlit) (3.1.4)
Requirement already satisfied: jsonschema>=3.0 in /usr/local/lib/python3.10/dist-packages (from altair<6,>=4.0->streamlit) (4.23.
Requirement already satisfied: toolz in /usr/local/lib/python3.10/dist-packages (from altair<6,>=4.0->streamlit) (0.12.1)
Collecting gitdb<5,>=4.0.1 (from gitpython!=3.1.19,<4,>=3.0.7->streamlit)
  Downloading gitdb-4.0.11-py3-none-any.whl.metadata (1.2 kB)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas<3,>=1.3.0->streamli
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas<3,>=1.3.0->streamlit) (2024.1
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-packages (from pandas<3,>=1.3.0->streamlit) (2024
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.27->strea
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.27->streamlit) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.27->streamlit)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.27->streamlit)
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.10/dist-packages (from rich<14,>=10.14.0->streamli
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.10/dist-packages (from rich<14,>=10.14.0->stream
Collecting smmap<6,>=3.0.1 (from gitdb<5,>=4.0.1->gitpython!=3.1.19,<4,>=3.0.7->streamlit)
  Downloading smmap-5.0.1-py3-none-any.whl.metadata (4.3 kB)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: attrs>=22.2.0 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in /usr/loca
Requirement already satisfied: referencing>=0.28.4 in /usr/local/lib/python3.10/d
Requirement already satisfied: rpds-py>=0.7.1 in /usr/local/lib/python3.10/dist-packages (from jsonschema>=3.0->altair<6,>=4.0->s
```

McAfee WebAdvisor

Your download's being scanned.
We'll let you know if there's an issue.

```
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.10/dist-packages (from markdown-it-py>=2.2.0->rich<14,>=10.14
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.2->pandas<3,>=1.3.0
Downloading streamlit-1.37.1-py2.py3-none-any.whl (8.7 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 8.7/8.7 MB 67.0 MB/s eta 0:00:00
Downloading GitPython-3.1.43-py3-none-any.whl (207 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 207.3/207.3 kB 12.8 MB/s eta 0:00:00
Downloading pydeck-0.9.1-py2.py3-none-any.whl (6.9 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 6.9/6.9 MB 70.9 MB/s eta 0:00:00
Downloading tenacity-8.5.0-py3-none-any.whl (28 kB)
Downloading watchdog-4.0.2-py3-none-manylinux2014_x86_64.whl (82 kB)
```

```
!wget -q -O - ipv4.icanhazip.com
```

```
35.230.37.213
```

```
%%writefile app.py
from dotenv import load_dotenv

load_dotenv()  # take environment variables from .env.

import streamlit as st
import os
import pathlib
import textwrap

import google.generativeai as genai
GOOGLE_API_KEY=''
os.getenv("GOOGLE_API_KEY")
genai.configure(api_key=os.getenv("GOOGLE_API_KEY"))

## Function to load OpenAI model and get respones

def get_gemini_response(question):
    model = genai.GenerativeModel('gemini-pro')
    response = model.generate_content(question)
    return response.text

##initialize our streamlit app

st.set_page_config(page_title="Q&A Demo")

st.header("Gemini Application")

input=st.text_input("Input: ",key="input")


submit=st.button("Ask the question")

## If ask button is clicked

if submit:

    response=get_gemini_response(input)
    st.subheader("The Response is")
    st.write(response)
```

```
Writing app.py
```

```
!streamlit run app.py &
```

```
Collecting usage statistics. To deactivate, set browser.gatherUsageStats to false.


  You can now view your Streamlit app in your browser.

  Local URL: http://localhost:8501
  Network URL: http://172.28.0.12:8501
  External URL: http://35.230.37.213:8501
```

Start coding or generate with AI.

McAfee WebAdvisor

Your download's being scanned.
We'll let you know if there's an issue.