

# **TEXT CLASSIFICATION WITH BERT**



**Group 04**

Instructors: Mr. Le Thanh Tung

Ms. Tran Thi Thao Nhi

Danh sách thành viên:

20127674: LÊ ĐỨC ĐẠT

19127359: TRƯƠNG DIỆU ĐẠT

20127552: VƯƠNG HUỲNH TẤN LỘC



# TEXT CLASSIFICATION WITH BERT



## Phân công công việc:

Danh sách thành viên:

20127674: LÊ ĐỨC ĐẠT (80%)

Code, Làm slide chung(Doc to deck trên Canva)

19127359: TRƯƠNG DIỆU ĐẠT(10%)

Làm slide lí thuyết

20127552: VƯƠNG HUỲNH TẤN  
LỘC(10%)

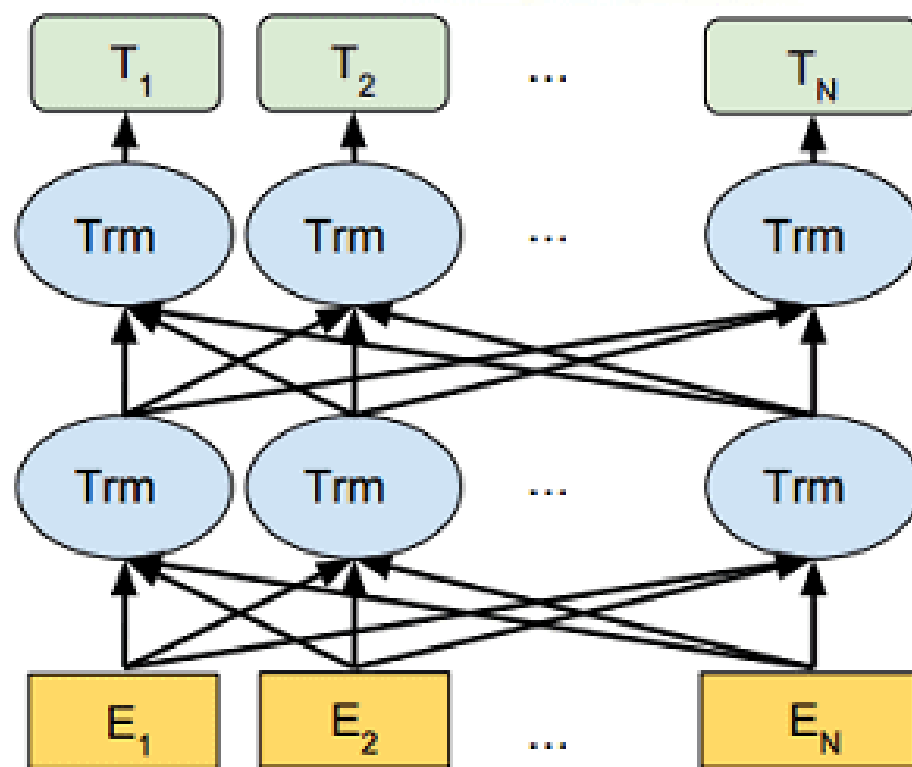
Làm slide lí thuyết

# 1. WHAT IS BERT

BERT (Bidirectional Encoder Representations from Transformers) is a language model developed by Google in 2018. BERT has achieved breakthrough results in many natural language processing (NLP) tasks such as text classification, predict the next sentence, answer questions, and understand semantics.



# BERT history:



BERT was originally published by Google researchers Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova in 2018.

On October 25, 2019, Google announced that they had started applying BERT models for English language search queries within the US. On December 9, 2019, it was reported that BERT had been adopted by Google Search for over 70 languages. In October 2020, almost every single English-based query was processed by a BERT model.

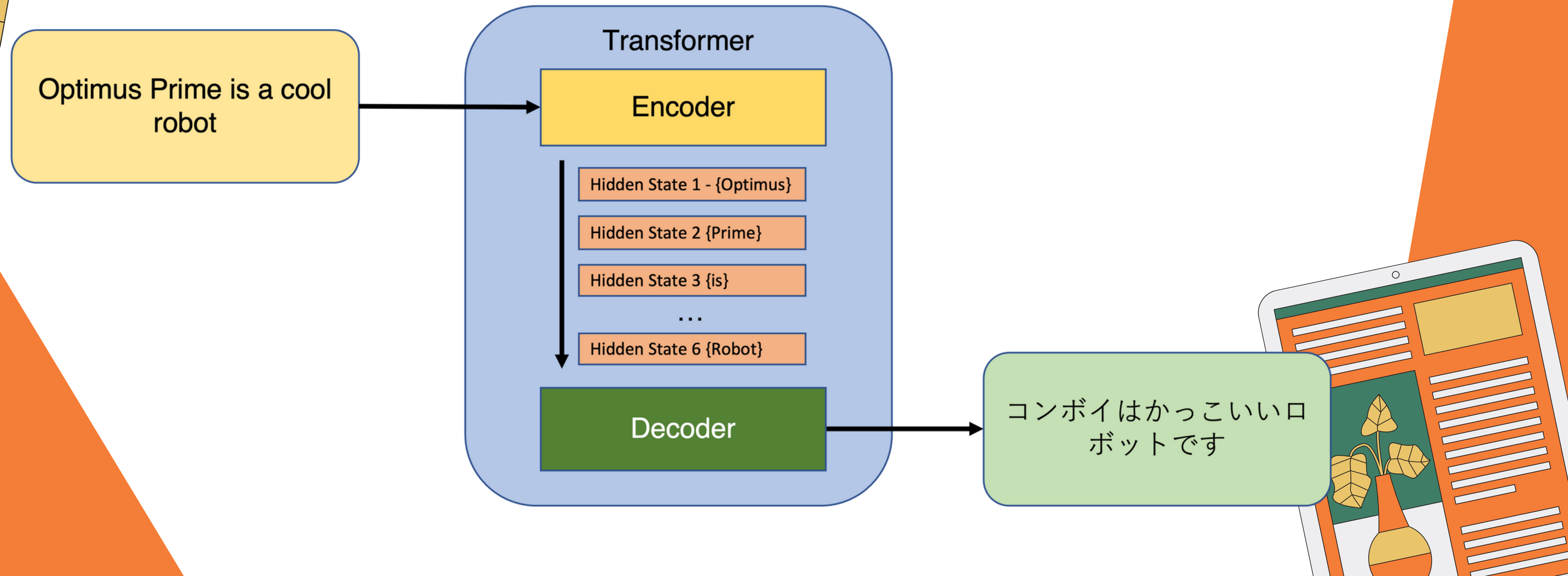
# Some of the key features of the BERT model





# INCLUDE

**Transformer Architecture: BERT uses the Transformer architecture, a self-attention-based structure that allows the model to focus on relationships between words in a sentence.**





**Bidirectional training: BERT is trained to understand the context in both directions (left-to-right and right-to-left), allowing better semantic capture than one-way training models.**

## BERT Explained

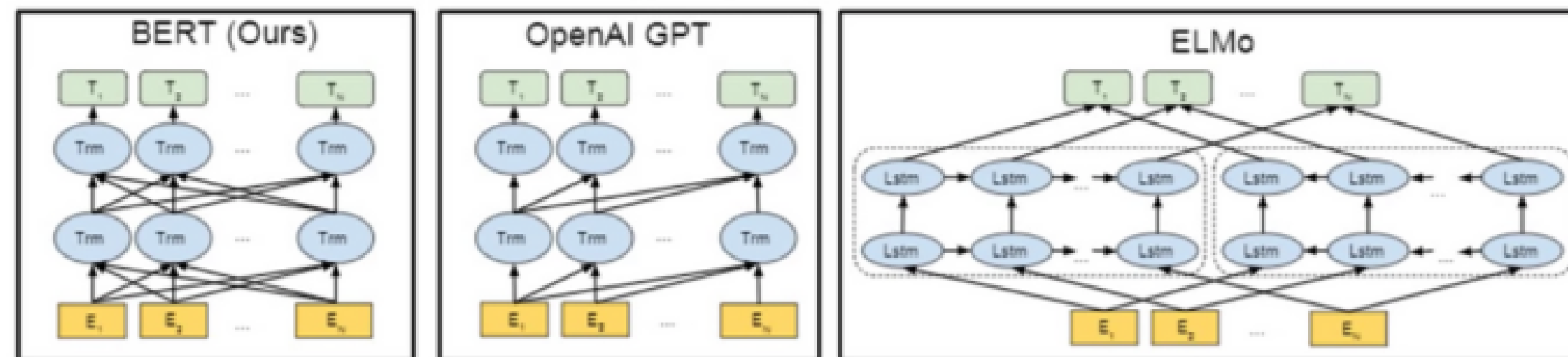

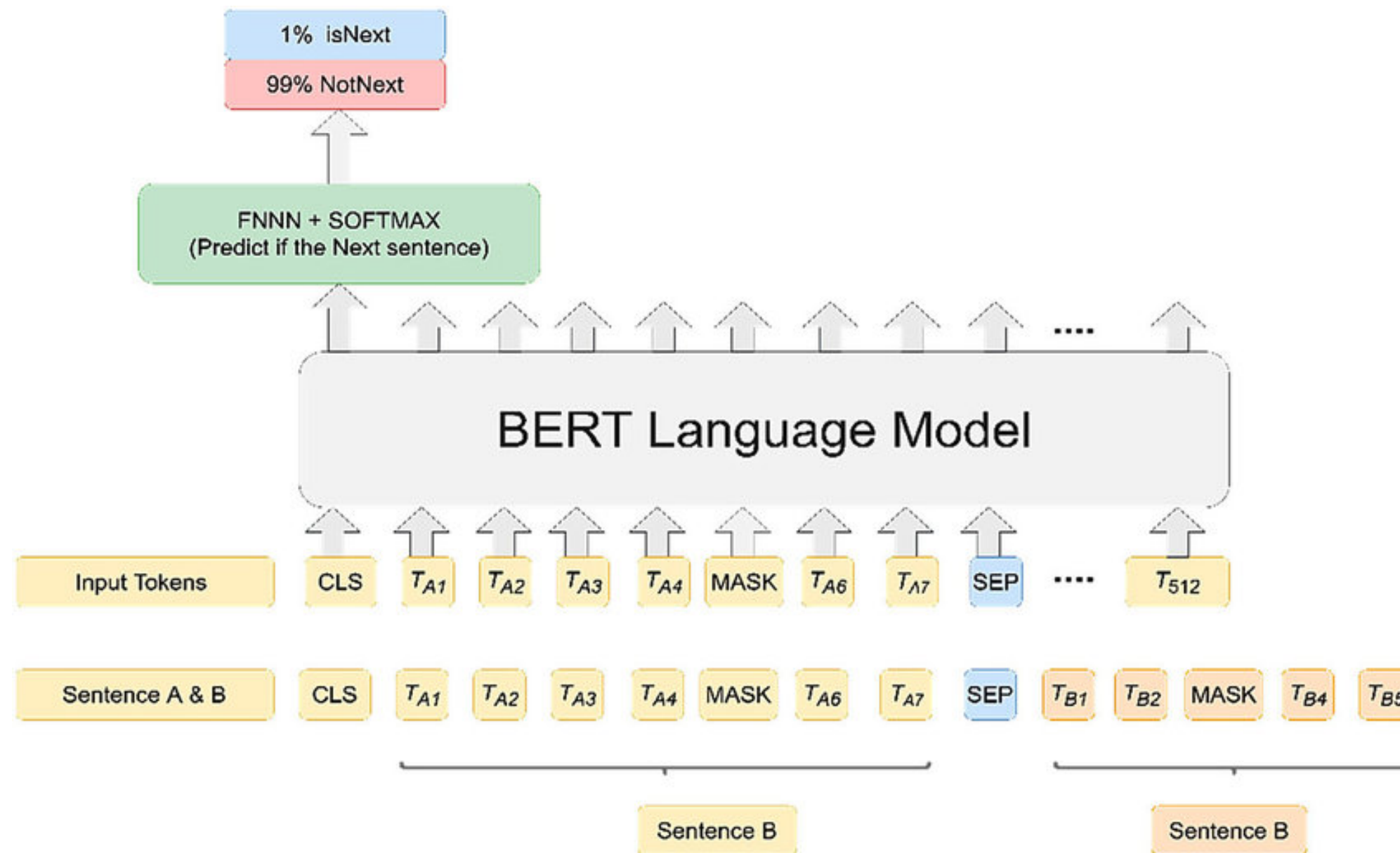


Figure 1: Differences in pre-training model architectures. BERT uses a bidirectional Transformer. OpenAI GPT uses a left-to-right Transformer. ELMo uses the concatenation of independently trained left-to-right and right-to-left LSTM to generate features for downstream tasks. Among three, only BERT representations are jointly conditioned on both left and right context in all layers.

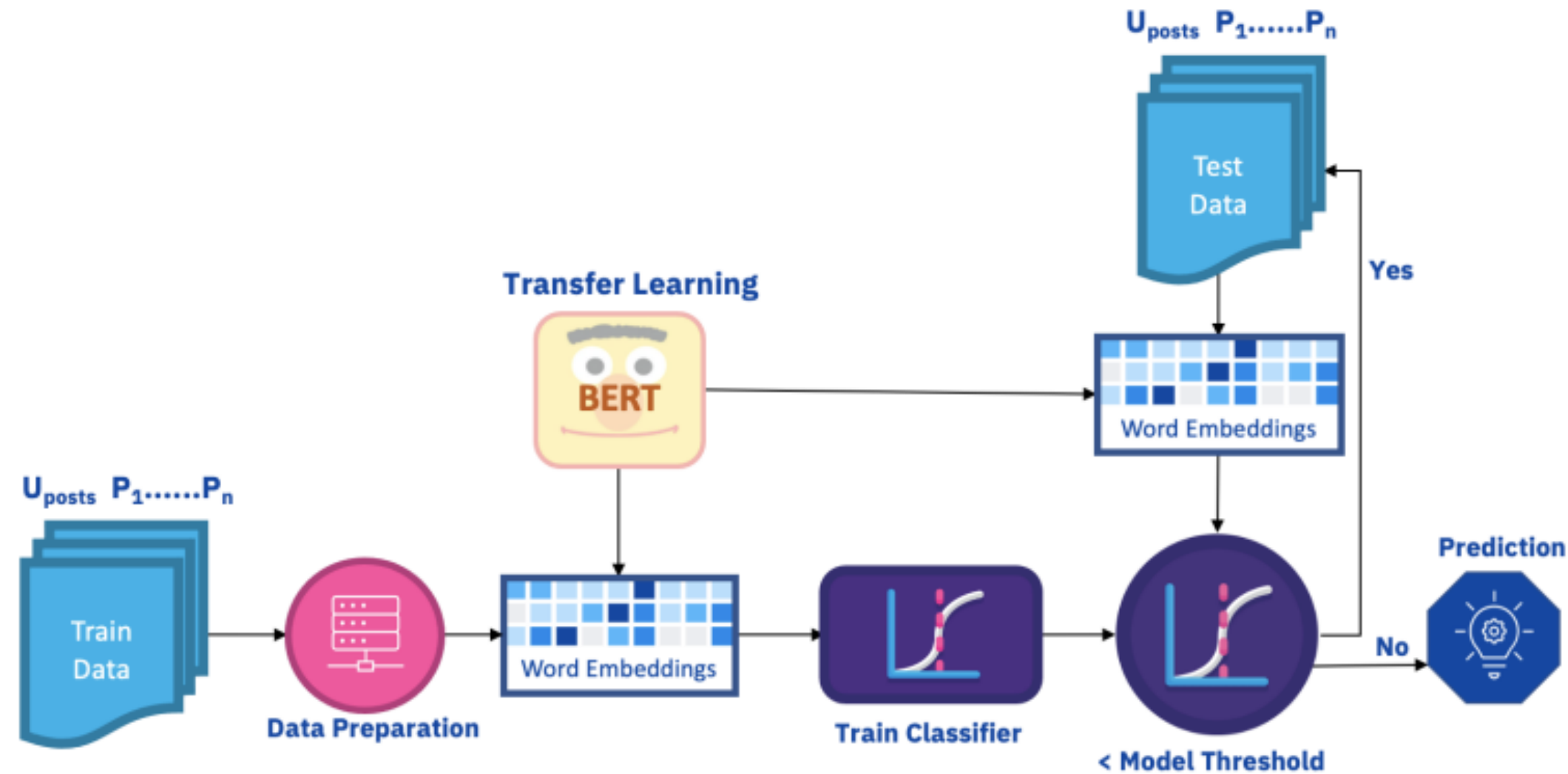


**Unsupervised Training: BERT is trained on unsupervised data using two techniques: Masked Language Model and Next Sentence Prediction. This allows BERT to learn from large amounts of textual data without the need for labels.**





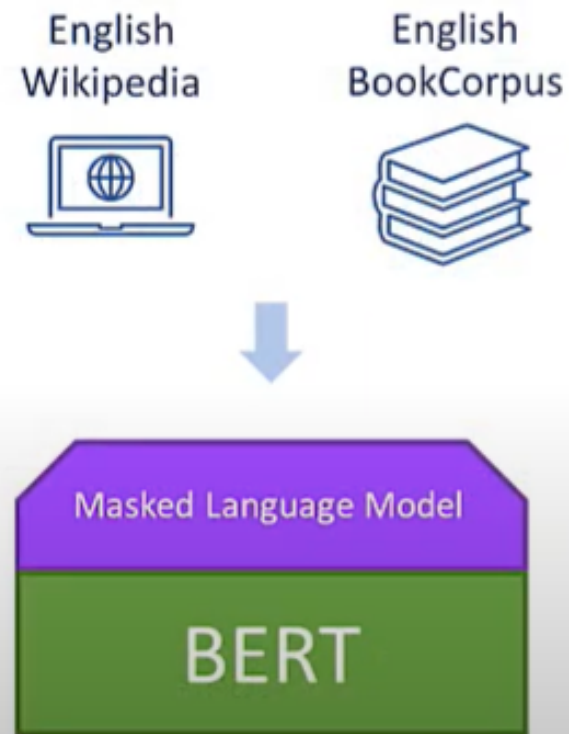
Transfer learning: BERT can be fine-tuned to solve specific tasks by transferring knowledge learned from training on unsupervised data. This improves performance and reduces training time for many NLP tasks.



**Figure 3:** High Level Flow Diagram / Architecture Diagram of the experiment

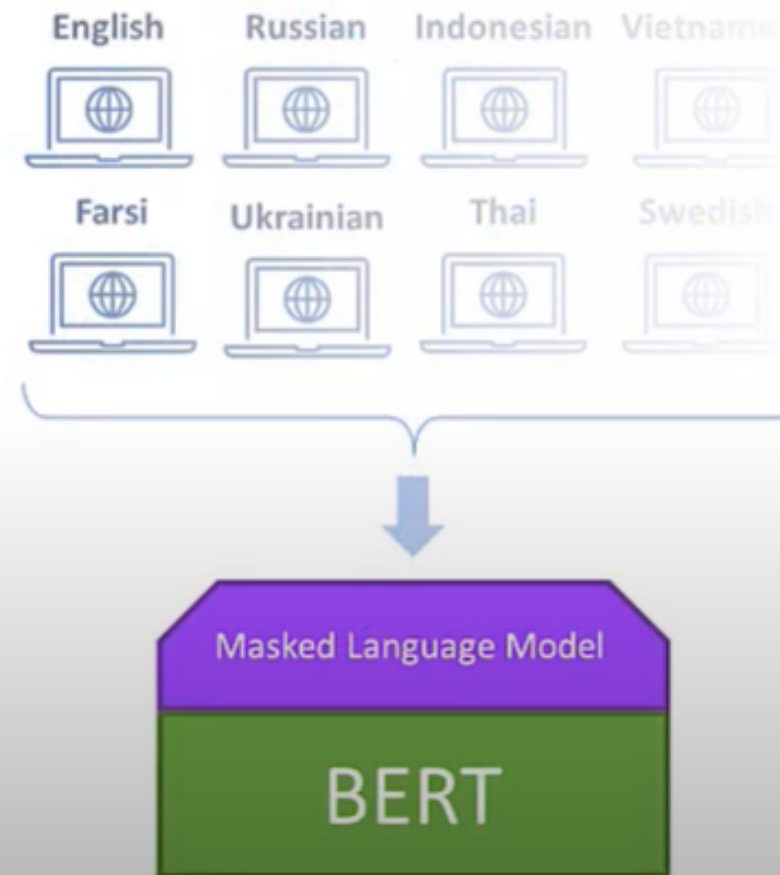
# Multilingual Models

## Original BERT Pre-Training



## XLM-R Pre-Training

CommonCrawl  
in 100 languages



Multilingual model: BERT is available in multiple versions that have been trained on multilingual data, allowing support for many different languages.





## 2. DATASET, INPUT AND OUTPUT



After you have downloaded the BERT model and created the tokenizer, you need to prepare the dataset, input and output data as follows:




**Dataset: Search or create a dataset consisting of Vietnamese documents/TEXT labeled as positive (1), neutral (0) and negative (2). This dataset will be used to train and test your models.**


```
data = {  
    "label": [1, 2, 0, 1, 2, 1, 2, 1, 2, 2, 0, 0, 0, 0, 0, 2, 1, 1, 0, 0, 2, 2, 1, 1, 2, 1, 0, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 2, 1, 1, 2, 1, 2, 2, 2, 0, 2],  
    "text": [  
        "Tôi rất thích sản phẩm này",  
        "Dịch vụ không tốt",  
        "Món ăn bình thường",  
        "Nhân viên thân thiện",  
        "Giá cả quá đắt",  
        "Tôi thực sự hạnh phúc",  
        "Tôi không thích sản phẩm này",  
        "Dịch vụ tốt",  
        "Món ăn không bình thường",  
        "Nhân viên không thân thiện",  
        "Giá cả không quá đắt",  
        "Tôi thực sự bình thường",  
        "Tôi thấy sản phẩm này bình thường",  
        "Dịch vụ bình thường",  
        "Món ăn không bình thường",  
    ]  
}
```







**Input data: Input data includes Vietnamese documents/TEXT from dataset. You need to split the dataset into two subsets: the training set and the test set. The training set will be used to train the model, while the test set helps to evaluate the performance of the model.**





**Output data: Output data are labels corresponding to the texts in the training and test sets. These labels are used to monitor the model training and evaluate the model's performance on the test set.**





# Steps to take



After preparing the dataset, input and output data, you can proceed to train and evaluate the BERT, TF-IDF, Word2Vec and Bag-of-words models





**Data preprocessing: Use BERT's tokenizer to encode the sentences in the training and test sets. For TF-IDF, Word2Vec and Bag-of-words, use "underthesea" library to separate Vietnamese words.**





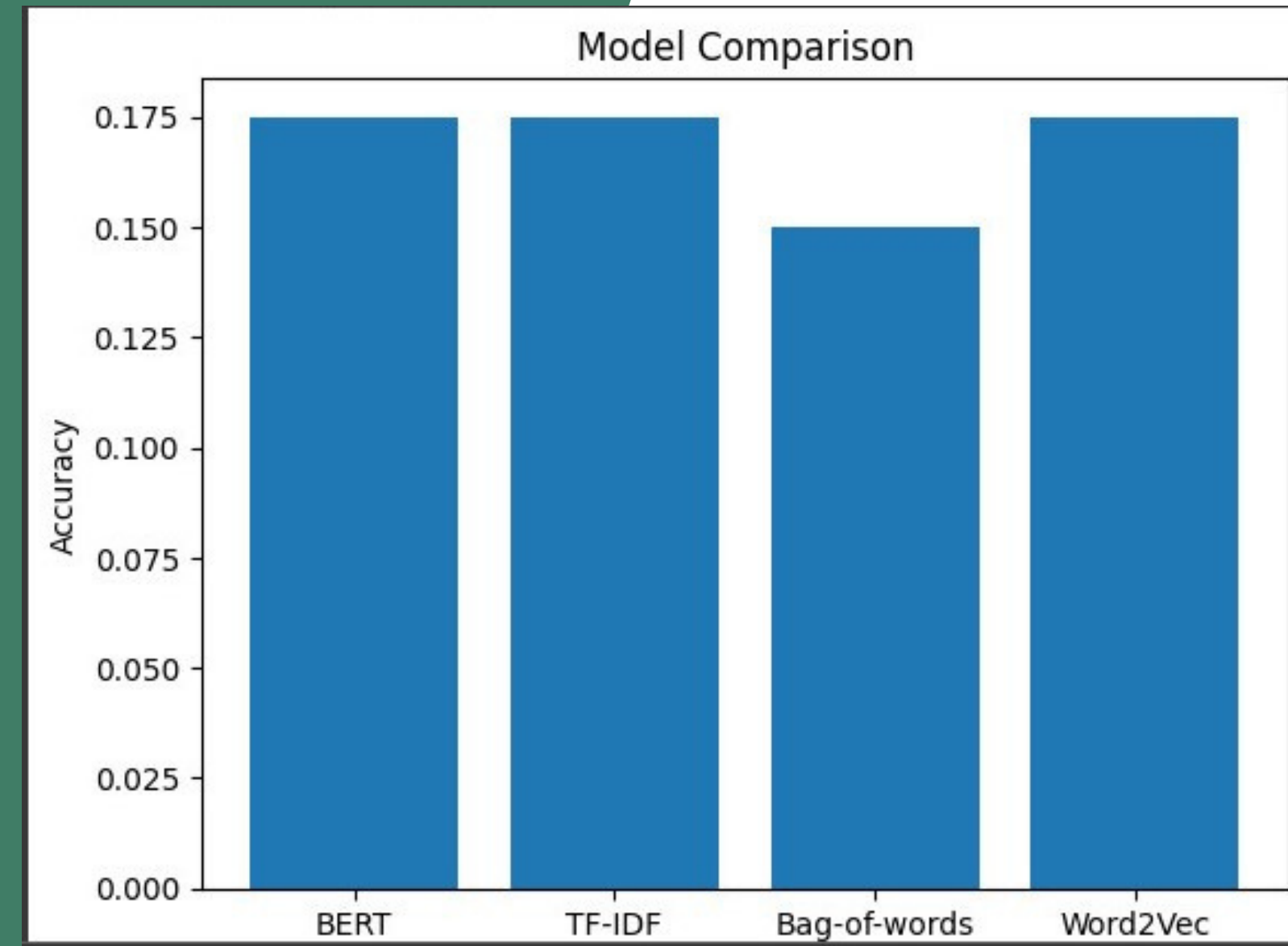
Model training: Train  
BERT, TF-IDF,  
Word2Vec and Bag-of-  
words models on the  
training set.



**Model evaluation: Use the trained models to predict the labels of the texts in the test set. Calculate the accuracy of each model by comparing the predicted results with the actual labels.**



Graphing: Plot a column graph to compare the accuracy of BERT, TF-IDF, Word2Vec and Bag-of-words models.



# EXAMPLES TO COMPARE ENGLISH AND VIETNAMESE TEXT CLASSIFICATION WITH BERT





# ENGLISH





# LIBRARIES

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score
from sklearn.pipeline import Pipeline
from gensim.models import Word2Vec
from gensim.models.keyedvectors import KeyedVectors
from transformers import BertTokenizer, TFBertForSequenceClassification
import tensorflow as tf
import gensim
import nltk
nltk.download("movie_reviews")

from nltk.corpus import movie_reviews
```



# LIBRARIES

## Các thư viện VD1 - Datasets review phim nước ngoài

`numpy`: Thư viện này giúp thực hiện các phép tính trên mảng nhiều chiều, là một thư viện cơ bản trong xử lý số liệu.

`pandas`: Thư viện hỗ trợ xử lý và phân tích dữ liệu dạng bảng, giúp dễ dàng thao tác với dữ liệu.

`matplotlib.pyplot`: Thư viện vẽ đồ thị và biểu đồ, trong trường hợp này, được sử dụng để vẽ biểu đồ so sánh độ chính xác của các mô hình.

`CountVectorizer` và `TfidfVectorizer`: Cung cấp các phương pháp để chuyển đổi văn bản thành đại diện vector dựa trên Bag-of-words và TF-IDF.

`train_test_split`: Hàm chia dữ liệu thành tập huấn luyện và tập kiểm tra.

`LogisticRegression`: Mô hình hồi quy logistic, được sử dụng như một bộ phân loại trong ví dụ này.

`classification_report` và `accuracy_score`: Cung cấp báo cáo phân loại và đánh giá độ chính xác của mô hình.

`Pipeline`: Giúp tổ chức một chuỗi các bước xử lý và mô hình học máy.

`Word2Vec` và `KeyedVectors`: Các lớp của thư viện Gensim, giúp huấn luyện và sử dụng mô hình Word2Vec.

`BertTokenizer` và `TFBertForSequenceClassification`: Các lớp của thư viện Transformers, hỗ trợ sử dụng mô hình BERT cho bài toán phân loại văn bản.

`tensorflow`: Thư viện học sâu của Google, được sử dụng để xây dựng, huấn luyện và đánh giá mô hình BERT.

`gensim`: Thư viện xử lý ngôn ngữ tự nhiên, được sử dụng để xây dựng mô hình Word2Vec.

`nltk`: Thư viện xử lý ngôn ngữ tự nhiên, được sử dụng để tải và sử dụng tập dữ liệu `movie_reviews`.

Tập dữ liệu `movie_reviews` được tải từ thư viện NLTK và được xử lý thành danh sách các đánh giá, mỗi đánh giá được biểu diễn dưới dạng một chuỗi văn bản.

# SETUP DATASETS

```
documents = [(list(movie_reviews.words(fileid)), category)
              for category in movie_reviews.categories()
              for fileid in movie_reviews.fileids(category)]

# documents là một danh sách các bộ (tuple) gồm hai phần tử: một danh sách các từ trong mỗi đánh giá và nhãn của đánh giá đó (chuyên mục).
# Đoạn mã này sử dụng list comprehension, lặp qua tất cả các chuyên mục của movie_reviews (ví dụ: 'pos' và 'neg'),
# sau đó lặp qua tất cả các fileid trong mỗi chuyên mục.
# movie_reviews.words(fileid) trích xuất danh sách các từ trong mỗi đánh giá, và category là nhãn của đánh giá đó.
reviews = [" ".join(review) for review, _ in documents]

# reviews là một danh sách các đánh giá dưới dạng chuỗi văn bản. Đoạn mã này sử dụng list comprehension
# để lặp qua tất cả các bộ (tuple) trong documents, sau đó nối các từ trong mỗi đánh giá bằng dấu cách.
# Kết quả là một danh sách các đánh giá được biểu diễn dưới dạng một chuỗi văn bản, thích hợp cho việc tiếp tục xử lý và phân loại.

# Tạo tập dữ liệu tổng hợp với các nhãn ngẫu nhiên
# Đoạn mã này tạo một tập dữ liệu từ danh sách reviews và phân chia nó thành tập huấn luyện và kiểm tra.
sentiments = np.random.randint(0, 3, len(reviews))

# Tạo nhãn ngẫu nhiên: Đoạn mã này tạo một mảng sentiments chứa các giá trị ngẫu nhiên từ 0 đến 2 (bao gồm 0 và 2).
# Kích thước của mảng này bằng với độ dài của reviews.
data = {"review": reviews[:200], "sentiment": sentiments[:200]}

# Tạo tập dữ liệu tổng hợp: Đoạn mã này tạo một từ điển data chứa hai khóa: 'review' và 'sentiment'.
# Giá trị của khóa 'review' là một danh sách 200 đánh giá đầu tiên từ reviews, và giá trị của khóa 'sentiment' là 200 nhãn ngẫu nhiên đầu tiên từ sentiments.
df = pd.DataFrame(data)

# Tạo DataFrame: Đoạn mã này tạo một DataFrame df từ từ điển data. DataFrame này chứa hai cột: 'review' và 'sentiment'.
x = df['review'].tolist()
y = df['sentiment'].tolist()

# Tách dữ liệu và nhãn: Đoạn mã này tách cột 'review' và 'sentiment' của DataFrame thành hai danh sách riêng biệt, x và y.
# x chứa các đánh giá, trong khi y chứa các nhãn tương ứng.
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)

# Phân chia dữ liệu: Đoạn mã này sử dụng hàm train_test_split để chia dữ liệu và nhãn thành tập huấn luyện (x_train, y_train) và tập kiểm tra (x_test, y_test).
# 20% dữ liệu được sử dụng cho tập kiểm tra, và số liệu random_state=42 đảm bảo rằng phân chia được tái tạo giống nhau mỗi lần chạy mã.
```

# BERT MODEL

```
# BERT Model
# xây dựng và huấn luyện mô hình phân loại văn bản dựa trên BERT, sau đó tính toán độ chính xác trên tập kiểm tra.
tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")
bert_model = TFBertForSequenceClassification.from_pretrained("bert-base-uncased", num_labels=3)
# Khởi tạo tokenizer và mô hình BERT: Đoạn mã này tạo một tokenizer từ mô hình "bert-base-uncased" đã được huấn luyện
# trước và một mô hình BERT để phân loại chuỗi từ cùng mô hình đã được huấn luyện trước đó. Số lượng nhãn được đặt là 3 (tích cực, trung tính, tiêu cực).
train_encodings = tokenizer(X_train, truncation=True, padding=True, max_length=128)
test_encodings = tokenizer(X_test, truncation=True, padding=True, max_length=128)
# Mã hóa dữ liệu huấn luyện và kiểm tra: Tokenizer được sử dụng để mã hóa các đánh giá trong tập huấn luyện (X_train) và tập kiểm tra (X_test).
# Mã hóa bao gồm việc cắt xén (truncation), đệm (padding) và giới hạn độ dài tối đa là 128.
train_dataset = tf.data.Dataset.from_tensor_slices((
    dict(train_encodings),
    y_train
)).shuffle(1000).batch(8)

test_dataset = tf.data.Dataset.from_tensor_slices((
    dict(test_encodings),
    y_test
)).batch(8)
# Tạo tập dữ liệu TensorFlow: Đoạn mã này chuyển đổi các mã hóa và nhãn tương ứng thành tập dữ liệu TensorFlow (tf.data.Dataset)
# cho tập huấn luyện và tập kiểm tra. Tập huấn luyện được xáo trộn và chia thành các batch với kích thước là 8.
```

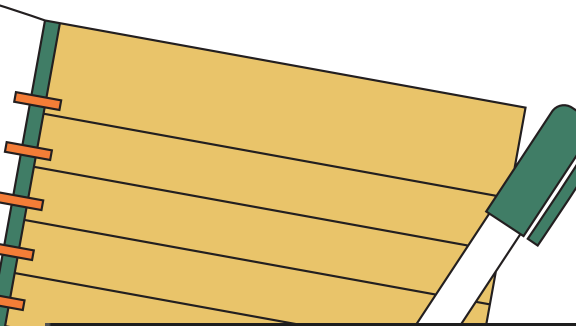


# BERT MODEL


```
optimizer = tf.keras.optimizers.Adam(learning_rate=5e-5)
loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
bert_model.compile(optimizer=optimizer, loss=loss, metrics=["accuracy"])
# Định nghĩa bộ tối ưu hóa, hàm mất mát và biên dịch mô hình: Đoạn mã này tạo một bộ tối ưu hóa Adam với tốc độ học 5e-5,
# Adam là một optimization algorithm được giới thiệu lần đầu bởi Diederik Kingma vào năm 2015, tại International Conference on Learning Representations.
# Theo như bài báo giới thiệu, Adam có khá nhiều điểm mạnh, đặc biệt là dễ sử dụng, tính toán hiệu quả, và phù hợp với các dữ liệu lớn và rời rạc.
# Đây cũng là thuật toán thường được sử dụng trong lĩnh vực computer vision và natural language processing.
# Khác với SGD (stochastic gradient descent), một phương pháp để tối ưu objective function trong deep learning,
# Adam áp dụng các learning rate khác nhau cho mỗi parameter dựa vào các tham số beta1 (first momentum of gradient) và beta2 (second momentum of gradient).
# Theo như tác giả, Adam có ưu điểm của hai optimization algorithm phổ biến khác là Adaptive gradient algorithm và Root mean square propagation.
# Đương nhiên, chúng ta không bắt buộc phải tự implement thuật toán để sử dụng. Hiện nay, Adam là một thuật toán phổ biến và được hỗ trợ trong nhiều framework
# khác nhau, chẳng hạn như TensorFlow, Keras, và Torch. Việc sử dụng Adam từ thư viện của các framework khá là tiện lợi.
# hàm mất mát là SparseCategoricalCrossentropy (tính từ logits) và biên dịch mô hình BERT với các thông số này cùng với độ chính xác ("accuracy") làm chỉ số đánh giá.
history = bert_model.fit(train_dataset, epochs=2, validation_data=test_dataset)
# Huấn luyện mô hình: Mô hình BERT được huấn luyện với tập dữ liệu huấn luyện trong 2 epochs và sử dụng tập dữ liệu kiểm tra làm dữ liệu xác thực.
bert_accuracy = history.history['val_accuracy'][-1]
# Tính toán độ chính xác của mô hình BERT: Sau khi huấn luyện, độ chính xác trên tập kiểm tra được lưu trữ trong biến bert_accuracy.
```



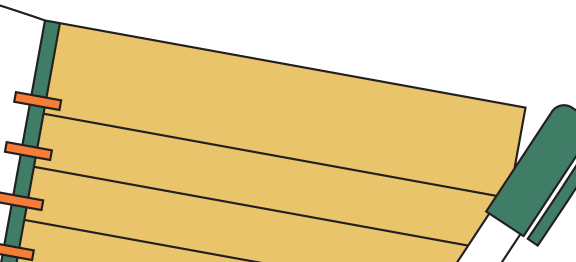
# TF-IDF MODEL




```
# TF-IDF Model - 1 mô hình phân loại văn bản khác.
tfidf_pipeline = Pipeline([
    ("tfidf_vectorizer", TfidfVectorizer()),
    ("classifier", LogisticRegression())
])
# Tạo một pipeline chứa hai bước: vector hóa TF-IDF (TfidfVectorizer) và phân loại logistic regression (LogisticRegression).
tfidf_pipeline.fit(X_train, y_train)
# Huấn luyện pipeline với tập dữ liệu huấn luyện (X_train, y_train).
tfidf_predictions = tfidf_pipeline.predict(X_test)
tfidf_accuracy = accuracy_score(y_test, tfidf_predictions)
# Dự đoán nhãn cho tập kiểm tra (X_test) và tính toán độ chính xác so với nhãn thực tế (y_test). Độ chính xác được lưu trữ trong biến tfidf_accuracy.
```



# BAG-OF-WORDS MODEL



```
# Bag-of-words Model
bow_pipeline = Pipeline([
    ("bow_vectorizer", CountVectorizer()),
    ("classifier", LogisticRegression())
])
# Tạo một pipeline chứa hai bước: vector hóa Bag-of-words (CountVectorizer) và phân loại logistic regression (LogisticRegression).
bow_pipeline.fit(X_train, y_train)
# Huấn luyện pipeline với tập dữ liệu huấn luyện (X_train, y_train).
bow_predictions = bow_pipeline.predict(X_test)
bow_accuracy = accuracy_score(y_test, bow_predictions)
# Dự đoán nhãn cho tập kiểm tra (X_test) và tính toán độ chính xác so với nhãn thực tế (y_test). Độ chính xác được lưu trữ trong biến bow_accuracy.
```



# WORD2VEC MODEL

```
# Word2Vec Model
def preprocess_text(text):
    return gensim.utils.simple_preprocess(text)
# Xác định hàm preprocess_text để chuyển đổi văn bản thành danh sách các từ đã được tiền xử lý.

tokenized_train = [preprocess_text(text) for text in X_train]
tokenized_test = [preprocess_text(text) for text in X_test]
# Tiền xử lý văn bản trong tập huấn luyện (X_train) và tập kiểm tra (X_test), lưu kết quả vào tokenized_train và tokenized_test.
word2vec_model = Word2Vec(tokenized_train, vector_size=100, window=5, min_count=1)
word_vectors = word2vec_model.wv
# Huấn luyện mô hình Word2Vec với các văn bản đã tiền xử lý trong tập huấn luyện và lưu trữ mô hình đã học vào biến word2vec_model. word_vectors chứa các vector từ đã học.
def vectorize_sentence(sentence, model):
    words = [word for word in sentence if word in model.index_to_key]
    return np.mean(model[words], axis=0) if words else np.zeros(model.vector_size)
# Định nghĩa hàm vectorize_sentence để chuyển đổi một câu thành một vector bằng cách lấy trung bình của các vector từ tương ứng trong mô hình đã học.
# Nếu không có từ nào trong câu có mặt trong mô hình, hàm trả về một vector 0.
X_train_w2v = np.array([vectorize_sentence(sentence, word_vectors) for sentence in tokenized_train])
X_test_w2v = np.array([vectorize_sentence(sentence, word_vectors) for sentence in tokenized_test])
# Vector hóa các câu trong tập huấn luyện và tập kiểm tra, lưu kết quả vào X_train_w2v và X_test_w2v.
w2v_classifier = LogisticRegression()
w2v_classifier.fit(X_train_w2v, y_train)
# Tạo và huấn luyện bộ phân loại logistic regression (w2v_classifier) với các vector câu đã học trong tập huấn luyện (X_train_w2v, y_train).
w2v_predictions = w2v_classifier.predict(X_test_w2v)
w2v_accuracy = accuracy_score(y_test, w2v_predictions)
# Dự đoán nhãn cho tập kiểm tra (X_test_w2v) và tính toán độ chính xác so với nhãn thực tế (y_test). Độ chính xác được lưu trữ trong biến w2v_accuracy.
```

# RESULT AS BAR-CHART

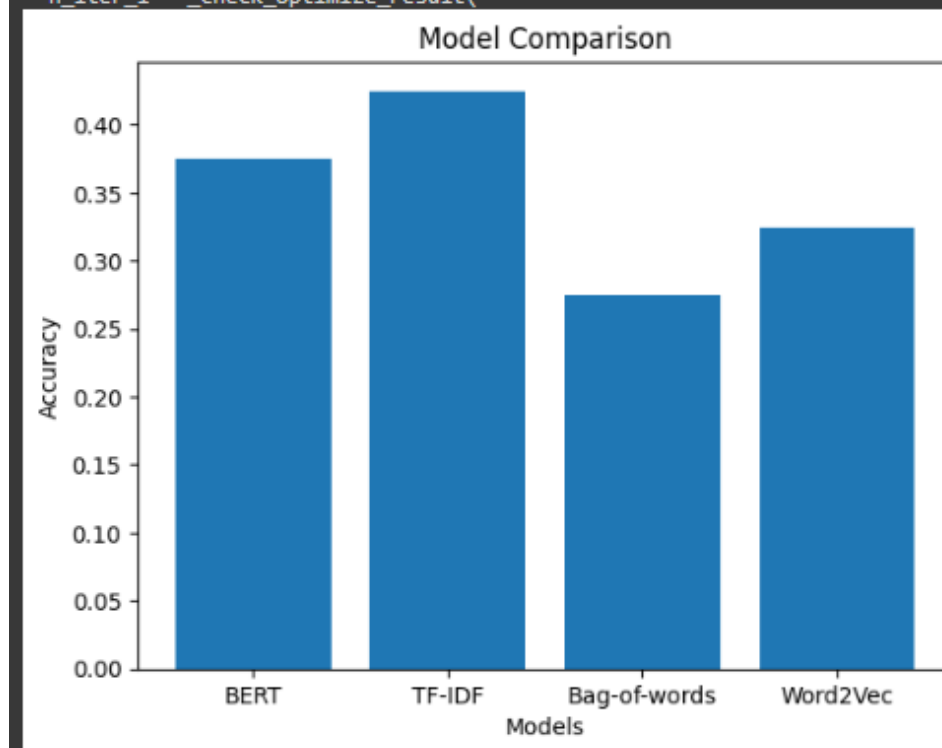
```
# Plot the results
labels = ["BERT", "TF-IDF", "Bag-of-words", "Word2Vec"]
accuracies = [bert_accuracy, tfidf_accuracy, bow_accuracy, w2v_accuracy]

plt.bar(labels, accuracies)
plt.xlabel("Models")
plt.ylabel("Accuracy")
plt.title("Model Comparison")
plt.show()
```

```
[nltk_data] Downloading package movie_reviews to /root/nltk_data...
[nltk_data] Package movie_reviews is already up-to-date!
All model checkpoint layers were used when initializing TFBertForSequenceClassification.

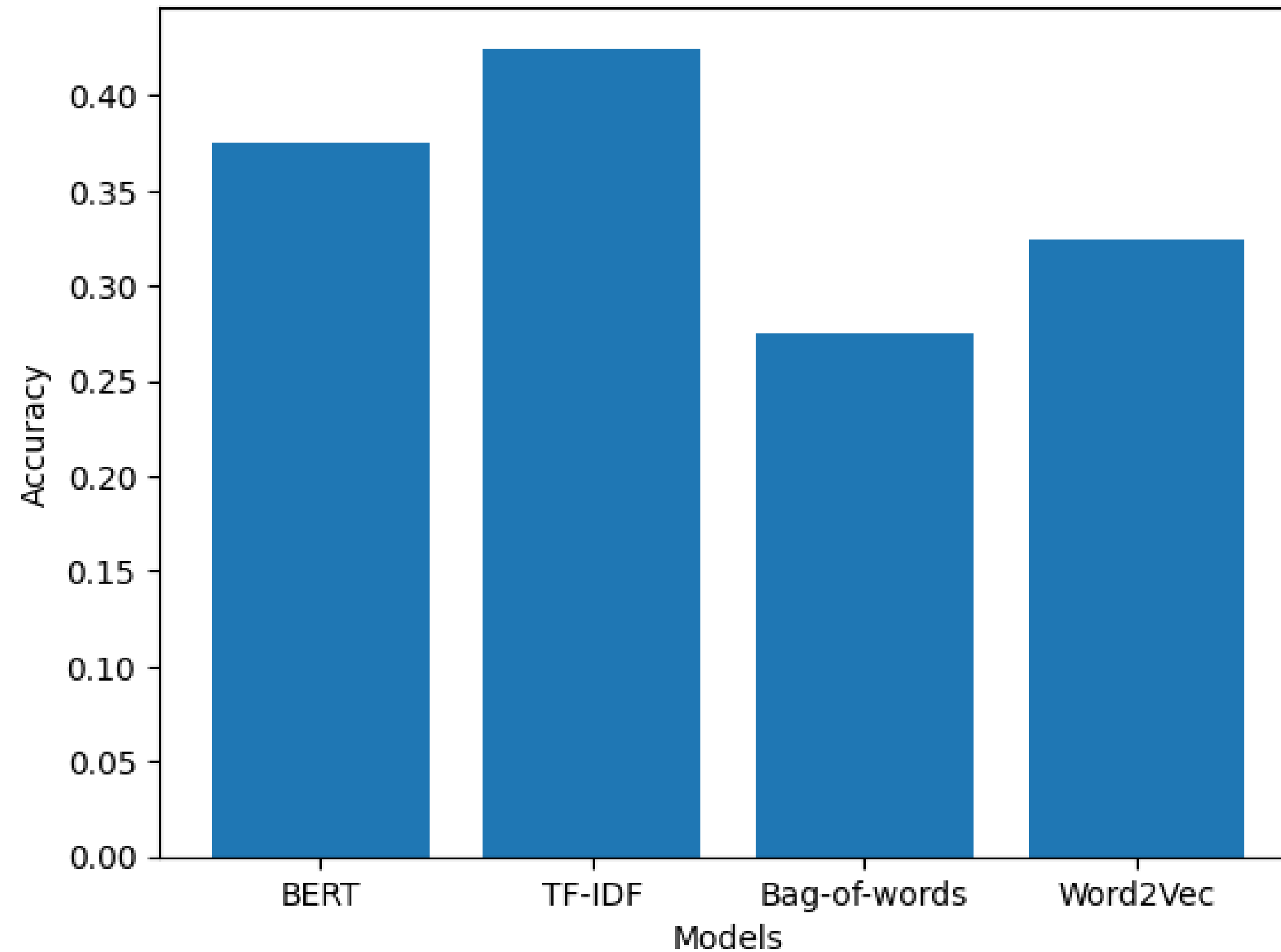
Some layers of TFBertForSequenceClassification were not initialized from the model checkpoint at bert-base-uncased and are newly initialized: ['classifier']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
Epoch 1/2
20/20 [=====] - 320s 14s/step - loss: 1.1060 - accuracy: 0.3688 - val_loss: 1.0684 - val_accuracy: 0.3750
Epoch 2/2
20/20 [=====] - 282s 14s/step - loss: 1.0914 - accuracy: 0.3875 - val_loss: 1.0752 - val_accuracy: 0.3750
/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
https://scikit-learn.org/stable/modules/linear\_model.html#logistic-regression
n_iter_i = _check_optimize_result(
```



# RESULT AS BAR-CHART

Model Comparison





# HOW ABOUT VIETNAMESE TEXT?





## LIBRARIES

Similar to English, but we add "Underthesea" module/Library in order to tokenize Vietnamese Words in sentences.



# Underthesea

```
from underthesea import word_tokenize  
#tách chữ tiếng Việt
```



# PREPARING DATASETS

We use 47 comments in Vietnamese - 47 labels

1 - positive

0 - neutral

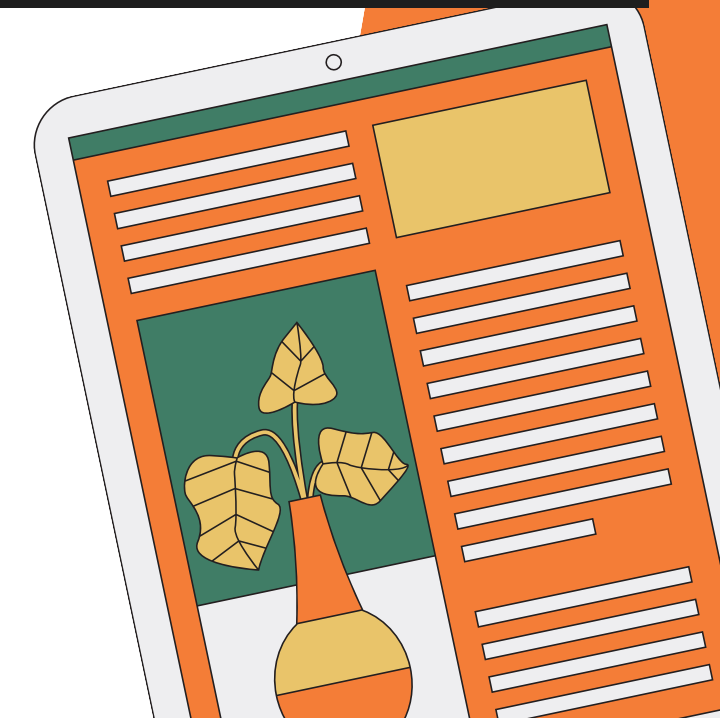
2 - negative

```
data = {  
  "text": [  
    "Tôi rất thích sản phẩm này",  
    "Dịch vụ không tốt",  
    "Món ăn bình thường",  
    "Nhân viên thân thiện",  
    "Giá cả quá đắt",  
    "Tôi thực sự hạnh phúc",  
    "Tôi không thích sản phẩm này",  
    "Dịch vụ tốt",  
    "Món ăn không bình thường",  
    "Nhân viên không thân thiện",  
    "Giá cả không quá đắt",  
    "Tôi thực sự bình thường",  
    "Tôi thấy sản phẩm này bình thường",  
    "Dịch vụ bình thường",  
    "Món ăn không bình thường",  
    "Nhân viên nhân nhó",  
    "Giá cả tuyệt vời",  
    "sản phẩm rất đẹp và chắc chắn Chất lượng sản phẩm tuyệt vời",  
  ],  
  "label": [1, 2, 0, 1, 2, 1, 2, 1, 2, 2, 0, 0, 0, 0, 0, 2, 1, 1, 0, 0, 2, 2, 1, 1, 2, 1, 0, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 2, 1, 1, 2, 1, 2, 2, 2, 0, 2],  
}
```



# SETUP DATASETS

```
df = pd.DataFrame([data])
# Tạo một DataFrame pandas từ dữ liệu đầu vào (biến data). DataFrame này bao gồm hai cột:
# "text" chứa các văn bản tiếng Việt và "label" chứa nhãn tương ứng (1: tích cực, 0: trung lập, 2: tiêu cực).
X = df["text"].tolist()
# Lấy cột "text" từ DataFrame và chuyển đổi thành danh sách các văn bản. Kết quả được lưu vào biến X.
y = df["label"].tolist()
# Lấy cột "label" từ DataFrame và chuyển đổi thành danh sách các nhãn. Kết quả được lưu vào biến y.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Sử dụng hàm train_test_split từ thư viện scikit-learn để chia dữ liệu thành hai tập con: tập huấn luyện (train) và tập kiểm tra (test).
# Phần tỷ lệ dữ liệu được sử dụng cho tập kiểm tra là 20% (test_size=0.2). Tham số random_state=42 đảm bảo kết quả chia dữ liệu là nhất quán mỗi khi chạy đoạn mã này.
def tokenize(text):
    return word_tokenize(text, format="text")
# Định nghĩa hàm tokenize nhận vào một chuỗi văn bản và trả về chuỗi văn bản đã được tách từ bằng cách sử dụng hàm word_tokenize từ thư viện underthesea.
# format="text" cho biết đầu ra sẽ là một chuỗi văn bản.
X_train_tokenized = [tokenize(text) for text in X_train]
# Áp dụng hàm tokenize cho từng văn bản trong X_train và lưu kết quả vào biến X_train_tokenized.
X_test_tokenized = [tokenize(text) for text in X_test]
# Tương tự như bước trên, áp dụng hàm tokenize cho từng văn bản trong X_test và lưu kết quả vào biến X_test_tokenized.
```



# BERT MODEL

```
# BERT Model
tokenizer = BertTokenizer.from_pretrained("bert-base-multilingual-uncased")
# Khởi tạo tokenizer BERT dựa trên mô hình "bert-base-multilingual-uncased", có hỗ trợ tiếng Việt.
bert_model = TFBertForSequenceClassification.from_pretrained("bert-base-multilingual-uncased", num_labels=3)
# Tải mô hình BERT dựa trên "bert-base-multilingual-uncased" và chỉnh sửa đầu ra để phân loại thành 3 nhãn (1: tích cực, 0: trung lập, 2: tiêu cực).
train_encodings = tokenizer(X_train_tokenized, truncation=True, padding=True, max_length=128)
# Mã hóa các văn bản huấn luyện đã được tách từ bảng tokenizer BERT. Cắt ngắn hoặc đệm các chuỗi để có độ dài tối đa là 128.
test_encodings = tokenizer(X_test_tokenized, truncation=True, padding=True, max_length=128)
# Tương tự như bước 3, mã hóa các văn bản kiểm tra đã được tách từ.
train_dataset = tf.data.Dataset.from_tensor_slices((
    dict(train_encodings),
    y_train
)).shuffle(1000).batch(8)
# Tạo tập dữ liệu huấn luyện từ các mã hóa và nhãn tương ứng. Xáo trộn dữ liệu với buffer size là 1000 và chia thành các batch với kích thước 8.
test_dataset = tf.data.Dataset.from_tensor_slices((
    dict(test_encodings),
    y_test
)).batch(8)
# Tạo tập dữ liệu kiểm tra từ các mã hóa và nhãn tương ứng, chia thành các batch với kích thước 8.
optimizer = tf.keras.optimizers.Adam(learning_rate=5e-5)
# Khởi tạo bộ tối ưu hóa Adam với learning rate là 5e-5.
loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
# Khởi tạo hàm mất mát (loss function) Sparse Categorical Crossentropy để sử dụng trong quá trình đào tạo mô hình.
bert_model.compile(optimizer=optimizer, loss=loss, metrics=["accuracy"])
# Biên dịch mô hình BERT với bộ tối ưu hóa, hàm mất mát và chỉ số đánh giá là độ chính xác (accuracy).
history = bert_model.fit(train_dataset, epochs=2, validation_data=test_dataset)
# Đào tạo mô hình BERT trên tập huấn luyện trong 2 kỳ (epochs) và đánh giá trên tập kiểm tra.
bert_accuracy = history.history['val_accuracy'][-1]
# Lấy độ chính xác (accuracy) của mô hình BERT trên tập kiểm tra từ lịch sử đào tạo.
# Độ chính xác này được lưu trong mảng 'val_accuracy', và ta chỉ quan tâm đến giá trị cuối cùng sau tất cả các kỳ đào tạo.
```

# TF-IDF MODEL

```
# TF-IDF Model
tfidf_pipeline = Pipeline([
    ("tfidf_vectorizer", TfidfVectorizer(tokenizer=tokenize)),
    # Bước đầu tiên trong pipeline là vector hóa dữ liệu văn bản sử dụng TfidfVectorizer, sử dụng hàm tokenize đã được định nghĩa ở trên.
    ("classifier", LogisticRegression())
    # Bước thứ hai trong pipeline là phân loại văn bản sử dụng hồi quy logistic.
])
# Tạo một pipeline chứa các bước để xử lý dữ liệu và xây dựng mô hình TF-IDF.
tfidf_pipeline.fit(X_train, y_train)
# Huấn luyện pipeline với dữ liệu huấn luyện (X_train và y_train).
tfidf_predictions = tfidf_pipeline.predict(X_test)
# Dự đoán nhãn cho dữ liệu kiểm tra (X_test) sử dụng mô hình TF-IDF.
tfidf_accuracy = accuracy_score(y_test, tfidf_predictions)
# Tính toán độ chính xác của mô hình TF-IDF trên dữ liệu kiểm tra.
```



# TF-IDF & BAG-OF-WORDS MODELS

```
# TF-IDF Model
tfidf_pipeline = Pipeline([
    ("tfidf_vectorizer", TfidfVectorizer(tokenizer=tokenize)),
    # Bước đầu tiên trong pipeline là vector hóa dữ liệu văn bản sử dụng TfidfVectorizer, sử dụng hàm tokenize đã được định nghĩa ở trên.
    ("classifier", LogisticRegression())
    # Bước thứ hai trong pipeline là phân loại văn bản sử dụng hồi quy logistic.
])
# Tạo một pipeline chứa các bước để xử lý dữ liệu và xây dựng mô hình TF-IDF.
tfidf_pipeline.fit(X_train, y_train)
# Huấn luyện pipeline với dữ liệu huấn luyện (X_train và y_train).
tfidf_predictions = tfidf_pipeline.predict(X_test)
# Dự đoán nhãn cho dữ liệu kiểm tra (X_test) sử dụng mô hình TF-IDF.
tfidf_accuracy = accuracy_score(y_test, tfidf_predictions)
# Tính toán độ chính xác của mô hình TF-IDF trên dữ liệu kiểm tra.
```

```
# Bag-of-words Model
bow_pipeline = Pipeline([
    ("bow_vectorizer", CountVectorizer(tokenizer=tokenize)),
    # Bước đầu tiên trong pipeline là vector hóa dữ liệu văn bản sử dụng CountVectorizer, sử dụng hàm tokenize đã được định nghĩa ở trên.
    ("classifier", LogisticRegression())
    # Bước thứ hai trong pipeline là phân loại văn bản sử dụng hồi quy logistic.
])

bow_pipeline.fit(X_train, y_train)
# Huấn luyện pipeline với dữ liệu huấn luyện (X_train và y_train).
bow_predictions = bow_pipeline.predict(X_test)
# Dự đoán nhãn cho dữ liệu kiểm tra (X_test) sử dụng mô hình Bag-of-words.
bow_accuracy = accuracy_score(y_test, bow_predictions)
# Tính toán độ chính xác của mô hình Bag-of-words trên dữ liệu kiểm tra.
```



# WORD2VEC MODEL

```
# Word2Vec Model
tokenized_train = [tokenize(text).split() for text in X_train]
# Tokenize các văn bản huấn luyện (X_train) và chuyển đổi chúng thành danh sách các từ.
tokenized_test = [tokenize(text).split() for text in X_test]
# Tokenize các văn bản kiểm tra (X_test) và chuyển đổi chúng thành danh sách các từ.
word2vec_model = Word2Vec(tokenized_train, vector_size=100, window=5, min_count=1)
# Huấn luyện mô hình Word2Vec với dữ liệu huấn luyện đã được tokenize, kích thước vector là 100,
# cửa sổ ngữ cảnh là 5 và số lần xuất hiện tối thiểu của một từ là 1.
word_vectors = word2vec_model.wv
# Lấy word vectors từ mô hình Word2Vec.
def vectorize_sentence(sentence, model):
    words = [word for word in sentence if word in model.index_to_key]
    return np.mean(model[words], axis=0) if words else np.zeros(model.vector_size)
# hàm để chuyển đổi một câu thành một vector bằng cách lấy trung bình của các vector từ trong câu.
# Nếu không có từ nào trong câu xuất hiện trong mô hình, hàm trả về một vector 0.
X_train_w2v = np.array([vectorize_sentence(sentence, word_vectors) for sentence in tokenized_train])
# Chuyển đổi các câu huấn luyện đã được tokenize thành các vector.
X_test_w2v = np.array([vectorize_sentence(sentence, word_vectors) for sentence in tokenized_test])
# Chuyển đổi các câu kiểm tra đã được tokenize thành các vector.
w2v_classifier = LogisticRegression()
# Khởi tạo một bộ phân loại hồi quy logistic.
w2v_classifier.fit(X_train_w2v, y_train)
# Huấn luyện bộ phân loại hồi quy logistic với dữ liệu huấn luyện đã được chuyển đổi sang các vector (X_train_w2v) và nhãn tương ứng (y_train).
w2v_predictions = w2v_classifier.predict(X_test_w2v)
# Dự đoán nhãn cho dữ liệu kiểm tra (X_test_w2v) sử dụng mô hình Word2Vec.
w2v_accuracy = accuracy_score(y_test, w2v_predictions)
# Tính toán độ chính xác của mô hình Word2Vec trên dữ liệu kiểm tra.
```

# TRAINING RESULT

Downloading (...)solve/main/vocab.txt: 100% 872k/872k [00:00<00:00, 6.52MB/s]  
Downloading (...)okenizer\_config.json: 100% 28.0/28.0 [00:00<00:00, 979B/s]  
Downloading (...)lve/main/config.json: 100% 625/625 [00:00<00:00, 21.7kB/s]  
Downloading tf\_model.h5: 100% 999M/999M [00:09<00:00, 87.1MB/s]  
All model checkpoint layers were used when initializing TFBertForSequenceClassification.  
  
Some layers of TFBertForSequenceClassification were not initialized from the model checkpoint at bert-base-multilingual-uncased and are newly initialized: ['classifier']  
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.  
Epoch 1/2  
5/5 [=====] - 80s 8s/step - loss: 1.0673 - accuracy: 0.4737 - val\_loss: 1.0810 - val\_accuracy: 0.4000  
Epoch 2/2  
5/5 [=====] - 33s 7s/step - loss: 0.9549 - accuracy: 0.5263 - val\_loss: 1.0393 - val\_accuracy: 0.6000  
/usr/local/lib/python3.9/dist-packages/sklearn/feature\_extraction/text.py:528: UserWarning: The parameter 'token\_pattern' will not be used since 'tokenizer' is not None'  
warnings.warn(  
/usr/local/lib/python3.9/dist-packages/sklearn/feature\_extraction/text.py:528: UserWarning: The parameter 'token\_pattern' will not be used since 'tokenizer' is not None'  
warnings.warn(  
/usr/local/lib/python3.9/dist-packages/sklearn/linear\_model/\_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):  
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

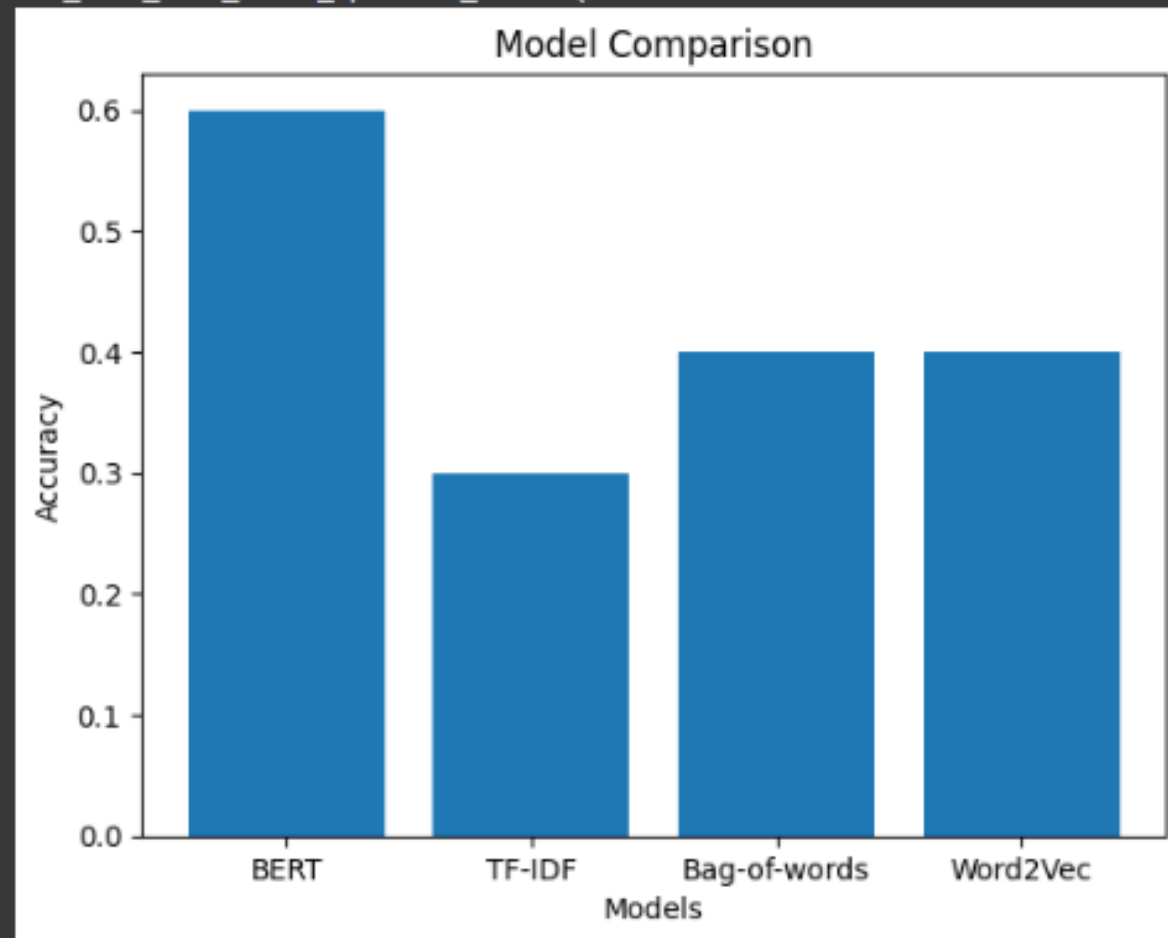
Increase the number of iterations (max\_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

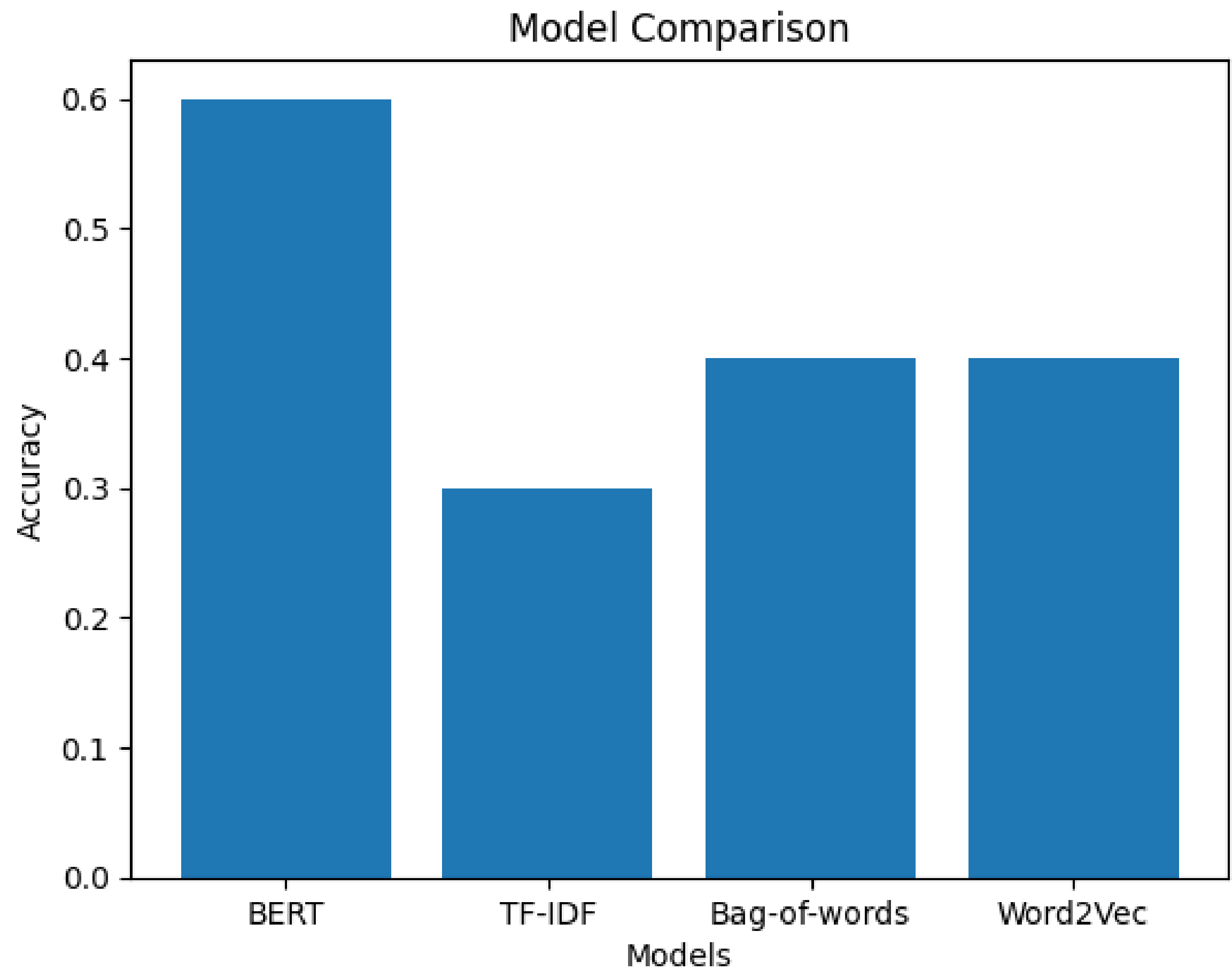
Please also refer to the documentation for alternative solver options:

[https://scikit-learn.org/stable/modules/linear\\_model.html#logistic-regression](https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)

```
n_iter_i = _check_optimize_result(
```



# BAR-CHART MODELS COMPARISON





**THANKS FOR LISTENING**

