



PROJECT REPORT

On

Similar Texts Identifying Tool

Course Title: Natural Language Processing

Submitted By:

Claudia Barbera

Haram EOM

Mohamed Hamas

Date of Submission: 10-06-2024

Table of Contents

1. Abstract
2. Introduction
 - 2.1 Aim
 - 2.2 Scope
 - 2.3 Methodology
3. Theoretical Part
4. Practical Part
5. Summary
6. Bibliography

1. Abstraction

Semantic similarity. A metric defined over a set of documents or terms, where the idea of distance between items is based on the likeness of their meaning or semantic content. Identifying similar texts is a fundamental task in various fields such as Natural Language Processing (NLP), information retrieval, plagiarism detection, and sentiment analysis, among others. This technique is used to find pairs of texts that express similar ideas, even if they are not identical word for word. This report presents the development and evaluation of Siamese BERT model to identify the semantic similarity between pairs of texts from Quora dataset.

2. Introduction

2.1 Aim

The aim of this project is to develop a tool capable of identify similarity between pair of texts. This tool is designed to help identify synthetically duplicate questions, which leads to improve the efficiency of questioning-answering systems and enhancing the user experience.

2.2 Scope

The scope of this project includes:

- Loading and preprocessing the question pairs dataset
- Performing data analysis to understand a characteristic of the given dataset
- Building a Siamese BERT model
- Training model for identifying text similarities

2.3 Methodology

- **Data loading and preprocessing:** Reading the dataset, handling missing values, text cleaning for effective analysis
- **Model building:** Using Siamese BERT model, encoding text data for BERT input, and compiling the model. Using L1 distance layer to capture the non-linearities
- **Model Evaluation:** Using model to predict then view results through confusion matrix

3. Theoretical Part

3.1 Theoretical Foundations

1. Text Preprocessing

- **Text Cleaning:** The first step in text analysis is cleaning, which involves removing unwanted characters, normalizing text (e.g., converting to lowercase), and removing stopwords. This reduces noise and improves the quality of the analysis.
 - **Tokenization:** Divides the text into smaller units called tokens (words, phrases, etc.), facilitating analysis.
 - **Lemmatization and Stemming:** Reduces words to their root form or lemma, helping to treat different words with the same meaning.
2. **Text Representation**
 - **Bag of Words (BoW):** Represents text as a collection of words, ignoring order but counting the frequency of each word.
 - **TF-IDF (Term Frequency-Inverse Document Frequency):** An improvement over BoW that weights words based on their frequency in a document and across the entire collection of documents.
 - **Word Embeddings:** Methods like Word2Vec, GloVe, and BERT represent words in vector spaces where words with similar meanings are closer together.
 3. **Similarity Measures**
 - **Euclidean Distance:** Measures the distance between two feature vectors in a multidimensional space.
 - **Cosine Similarity:** Measures the cosine of the angle between two vectors, useful for determining similarity in terms of direction.
 - **Jaccard Similarity:** Measures the similarity between two sets as the intersection divided by the union of the sets.
 4. **Machine Learning Models**
 - **Supervised Models:** Algorithms like Support Vector Machines (SVM), Random Forest, and neural networks trained with labeled data to predict similarity.
 - **Unsupervised Models:** Algorithms like K-means clustering that group similar texts without labeled data.
 5. **Deep Learning Models**
 - **Transformers:** Advanced models like BERT and GPT that capture the bidirectional context of words and are highly effective in NLP tasks.
 - **Siamese Networks:** Twin neural networks used to compute the similarity between pairs of texts by comparing their vector representations.

3.2 Tools and Libraries

1. **NLTK (Natural Language Toolkit):** Provides tools for text preprocessing, tokenization, lemmatization, and more.
2. **SpaCy:** A fast and robust NLP library that includes functionalities for tokenization, lemmatization, part-of-speech tagging, and more.
3. **Transformers (Hugging Face):** Offers pretrained models like BERT and GPT for advanced NLP tasks.
4. **TensorFlow and Keras:** Deep learning libraries that allow building and training neural network models.

4. Practical Part

1. Loading and Preprocessing Data

- Reading data from a CSV file.

```
train_data = pd.read_csv("train.csv")  
train_data.head(10)
```

	id	qid1	qid2	question1	question2	is_duplicate
0	0	1	2	What is the step by step guide to invest in sh...	What is the step by step guide to invest in sh...	0
1	1	3	4	What is the story of Kohinoor (Koh-i-Noor) Dia...	What would happen if the Indian government sto...	0
2	2	5	6	How can I increase the speed of my internet co...	How can Internet speed be increased by hacking...	0
3	3	7	8	Why am I mentally very lonely? How can I solve...	Find the remainder when 23^{24} is divided by 100	0
4	4	9	10	Which one dissolve in water quickly sugar, salt...	Which fish would survive in salt water?	0
5	5	11	12	Astrology: I am a Capricorn Sun Cap moon and c...	I'm a triple Capricorn (Sun, Moon and ascendan...	1
6	6	13	14	Should I buy tiago?	What keeps children active and far from phone ...	0
7	7	15	16	How can I be a good geologist?	What should I do to be a great geologist?	1
8	8	17	18	When do you use ㄣ instead of ㄵ?	When do you use "&" instead of "and"?	0
9	9	19	20	Motorola (company): Can I hack my Charter Moto...	How do I hack Motorola DCX3400 for free internet?	0

- Cleaning and preprocessing text (removing unwanted characters, converting to lowercase, lemmatization).

```
def text_cleaning(x):

    questions = re.sub('\s+\n+', ' ', x)
    questions = re.sub('[^a-zA-Z0-9]', ' ', questions)
    questions = questions.lower()

    return questions

train_data['question1_cleaned'] =
train_data['question1'].progress_apply(text_cleaning)
train_data['question2_cleaned'] =
train_data['question2'].progress_apply(text_cleaning)
train_data
```

	id	qid1	qid2	question1	question2	is_duplicate	question1_cleaned	question2_cleaned
0	0	1	2	What is the step by step guide to invest in sh...	What is the step by step guide to invest in sh...	0	what is the step by step guide to invest in sh...	what is the step by step guide to invest in sh...
1	1	3	4	What is the story of Kohinoor (Koh-i-Noor) Dia...	What would happen if the Indian government sto...	0	what is the story of kohinoor koh i noor dia...	what would happen if the indian government sto...
2	2	5	6	How can i increase the speed of my internet co...	How can Internet speed be increased by hacking...	0	how can i increase the speed of my internet co...	how can internet speed be increased by hacking...
3	3	7	8	Why am I mentally very lonely? How can i solve...	Find the remainder when $(\text{math}23^4(241)/\text{math})$ i...	0	why am i mentally very lonely how can i solve...	find the remainder when math 23 24 math i...
4	4	9	10	Which one dissolve in water quikly sugar, salt...	Which fish would survive in salt water?	0	which one dissolve in water quikly sugar salt...	which fish would survive in salt water

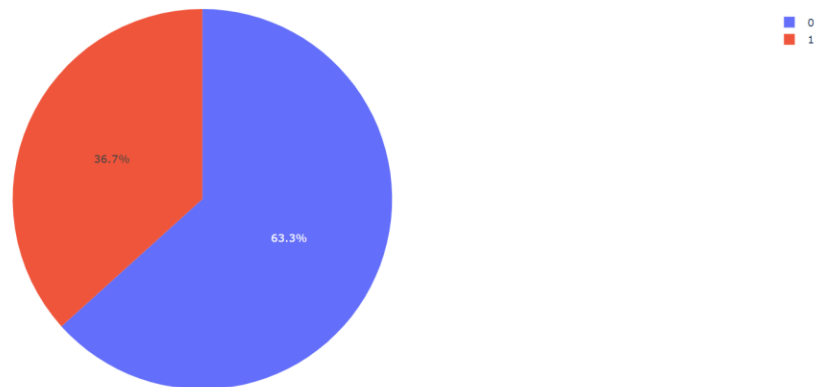
2. Data Exploration

- Analyzing the distribution of classes (similar vs. non-similar texts).

```
pio.renderers.default = 'colab'

fig = px.pie(train_data, values='id', names='is_duplicate', height=600,
title='Proportion of Duplicate and Non Duplicate Questions')
fig.show()
```

Proportion of Duplicate and Non Duplicate Questions



- Visualizing the distributions of question lengths.

```
## Compute ideal length of sentence
train_data['question1_lens'] =
train_data['question1_cleaned'].apply(lambda x: len(x.split()))
train_data['question2_lens'] =
train_data['question2_cleaned'].apply(lambda x: len(x.split()))

train_data['question1_lens'].describe()
train_data['question2_lens'].describe()

# Calculate Q1 and Q3 for question lengths
q1_q1 = train_data['question1_lens'].quantile(0.25)
q3_q1 = train_data['question1_lens'].quantile(0.75)
q1_q2 = train_data['question2_lens'].quantile(0.25)
q3_q2 = train_data['question2_lens'].quantile(0.75)

# Calculate upper outlier threshold for question lengths
upper_outlier_q1 = q3_q1 + 1.5 * (q3_q1 - q1_q1)
upper_outlier_q2 = q3_q2 + 1.5 * (q3_q2 - q1_q2)

# Determine the maximum upper outlier threshold between the two
max_upper_outlier = max(upper_outlier_q1, upper_outlier_q2)
```

```
# Since max_upper_outlier is 22 we should take a number close to and
higher than 22, so lets take ideal value as 50.
ideal_len = 50
```

```
count    404287.000000
mean      11.376792
std       6.480827
min       0.000000
25%       7.000000
50%      10.000000
75%      13.000000
max      248.000000
Name: question2_lens, dtype: float64
```

results of train_data['question2_lens'].describe()

3. Tokenization and Encoding

- Using a tokenizer (such as BERT's) to convert text into sequences of tokens.
- Ensuring all sequences have the same length through padding.

```
model_checkpoint = 'bert-base-uncased'
tokenizer = AutoTokenizer.from_pretrained(model_checkpoint)

def encode_text(text, tokenizer):

    encoded = tokenizer.batch_encode_plus(
        text,
        add_special_tokens=True,
        max_length=50,
        padding='max_length',
        truncation=True,
        return_attention_mask=True,
        return_tensors="tf",
    )

    input_ids = np.array(encoded["input_ids"], dtype="int32")
    attention_masks = np.array(encoded["attention_mask"],
dtype="int32")

    return {
        "input_ids": input_ids,
        "attention_masks": attention_masks
    }

final_df['encoded_question1'] =
final_df['question1_cleaned'].apply(lambda x: encode_text([x],
tokenizer))
final_df['encoded_question2'] =
final_df['question2_cleaned'].apply(lambda x: encode_text([x],
tokenizer))
```

4. Building the Model

- Using pretrained models like BERT to obtain vector representations of questions.
- Constructing a neural network to compare these representations and predict if two questions are similar.

```
class L1Dist(Layer):

    def __init__(self, **kwargs):
        super().__init__()

    def call(self, embedding1, embedding2):
        return tf.math.abs(embedding1 - embedding2)

with strategy.scope():
    transformer_model = TFBertModel.from_pretrained(model_checkpoint)

    input_ids_in1 = Input(shape=(None,), name='input_ids1',
dtype='int32')
    input_masks_in1 = Input(shape=(None,), name='attention_mask1',
dtype='int32')
    input_ids_in2 = Input(shape=(None,), name='input_ids2',
dtype='int32')
    input_masks_in2 = Input(shape=(None,), name='attention_mask2',
dtype='int32')

    embedding_layer1 = transformer_model(input_ids_in1,
attention_mask=input_masks_in1).last_hidden_state
    embedding_layer2 = transformer_model(input_ids_in2,
attention_mask=input_masks_in2).last_hidden_state

    embedding1 = GlobalAveragePooling1D()(embedding_layer1)
    embedding2 = GlobalAveragePooling1D()(embedding_layer2)
    l1_dist = L1Dist()(embedding1, embedding2)

    x = Dense(512, activation='relu')(l1_dist)
    output = Dense(1, activation='sigmoid')(x)

    model = Model(inputs=[input_ids_in1, input_masks_in1,
input_ids_in2, input_masks_in2], outputs = output)
    model.compile(loss='binary_crossentropy', optimizer=tf.keras.optimiz
ers.Adam(learning_rate=0.00001), metrics='accuracy')

model.summary()
```


model.safetensors: 100%  440M/440M [00:05<00:00, 105MB/s]

Some weights of the PyTorch model were not used when initializing the TF 2.0 model `TFBertModel`: [`'cls.seq_relationship.weight'`, `'cls.predictions.bias'`, `'cls.predictions.transform.LayerNorm.bias'`].

- This IS expected if you are initializing `TFBertModel` from a PyTorch model trained on another task or with another architecture (e.g. initializing a `TFBertForSequenceClassification` model from a PyTorch `BertForSequenceClassification` model).
- This IS NOT expected if you are initializing `TFBertModel` from a PyTorch model that you expect to be exactly identical (e.g. initializing a `TFBertForSequenceClassification` model from a `BertForSequenceClassification` model).

All the weights of `TFBertModel` were initialized from the PyTorch model.

If your task is similar to the task the model of the checkpoint was trained on, you can already use `TFBertModel` for predictions without further training.

```
Model: "model"
```

Layer (type)	Output Shape	Param #	Connected to
input_ids1 (InputLayer)	[(None, None)]	0	[]
attention_mask1 (InputLayer)	[(None, None)]	0	[]
input_ids2 (InputLayer)	[(None, None)]	0	[]
attention_mask2 (InputLayer)	[(None, None)]	0	[]
tf_bert_model (TFBertModel)	TFBaseModel(OutputWithPool ingAndCrossAt tentions(last_hidden_state=(None, None , 768), pooler_output=(None, 768), past_key_values=None, hidden_states=None, attentions=None, cross_attentions=None)	109482240	['input_ids1[0][0] ', 'attention_mask1[0][0] ', 'input_ids2[0][0] ', 'attention_mask2[0][0] ']
global_average_pooling1d (GlobalAveragePooling1D)	(None, 768)	0	['tf_bert_model[0][0] ']
global_average_pooling1d_1 (GlobalAveragePooling1D)	(None, 768)	0	['tf_bert_model[1][0] ']
l1_dist (L1Dist)	(None, 768)	0	['global_average_pooling1d[0][0] ', 'global_average_pooling1d_1[0][0] ']
dense (Dense)	(None, 512)	393728	['l1_dist[0][0] ']
dense_1 (Dense)	(None, 1)	513	['dense[0][0] ']

```
Total params: 109,876,481
Trainable params: 394,241
Non-trainable params: 109,482,240
```

5. Training and Evaluating the Model

- Splitting the data into training and test sets.
- Training the model using techniques like early stopping and learning rate reduction.

```
earlystopping = EarlyStopping(monitor='val_loss',min_delta = 0,  
patience = 5, verbose = 1, restore best weights=True)
```

[illegible]

```
# Reshape training input ids and attention masks
```

```
X1_train_input_ids = np.squeeze(X1_train_input_ids, axis=1)
```

```
X1_train_attention_masks = np.squeeze(X1_train_attention_masks, axis=1)
```

```
X2_train_input_ids = np.squeeze(X2_train_input_ids, axis=1)
```

```
X2_train_attention_masks = np.squeeze(X2_train_attention_masks, axis=1)
```

```
# Reshape test input ids and attention masks
```

```
X1 test input ids = np.squeeze(X1 test input ids, axis=1)
```

```
X1 test attention masks = np.squeeze(X1 test attention masks, axis=1)
```

```
X2 test input ids = np.squeeze(X2 test input ids, axis=1)
```

```
X2_test_attention_masks = np.squeeze(X2_test_attention_masks, axis=1)

# Train model
history = model.fit((X1_train_input_ids, X1_train_attention_masks,
X2_train_input_ids, X2_train_attention_masks),
                    y_train, batch_size=BATCH_SIZE, epochs=5,
                    validation_data=((X1_test_input_ids,
X1_test_attention_masks, X2_test_input_ids, X2_test_attention_masks),
y_test),
                    callbacks=[earlystopping, learning_rate_reduction])
```

- Evaluating the model using metrics like the confusion matrix and classification report.

```
y_pred = model.predict((X1_test_input_ids,
                        X1_test_attention_masks,
                        X2_test_input_ids,X2_test_attention_masks))

# Display confusion matrix to view results

from sklearn.metrics import classification_report, confusion_matrix

y_pred[y_pred>=0.5] = 1
y_pred[y_pred<0.5] = 0

sns.heatmap(confusion_matrix(y_test,
y_pred), cmap='viridis', annot=True, fmt='.5g',
            xticklabels=['Dissimilar', 'Similar'], yticklabels=['Dissimila
r', 'Similar'])
plt.xlabel('Predicted Class')
plt.ylabel('Actual Class')
plt.show()
```



```
# Print other evaluation metrics
print(classification_report(y_test,y_pred))
```

	precision	recall	f1-score	support
0	0.90	0.84	0.87	12674
1	0.75	0.84	0.79	7326
accuracy			0.84	20000
macro avg	0.82	0.84	0.83	20000
weighted avg	0.84	0.84	0.84	20000

5. Summary

Identifying similar texts is a complex task involving multiple stages of processing and analysis. The techniques and tools described allow this task to be approached efficiently and accurately, using traditional methods as well as advanced deep learning models. The project can be developed by further optimizing the model and exploring additional datasets for broader applicability. The correct application of these methods can significantly improve applications in various domains such as information retrieval, online assistance, and more.

6. Bibliography

Quora dataset: <https://www.kaggle.com/c/quora-question-pairs/>

Jupyter notebook for project (Full Code):

<https://colab.research.google.com/drive/1sADkeSRsUBSNyavYncneIPyqlv31AS5x?usp=sharing>

Reference for project: <https://www.kaggle.com/code/quadeer15sh/siamese-bert-for-quora-question-similarity>