

Similar Texts Identifying Tool

Claudia Barbera | Haram Eom | Mohamed Hamas

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



Natural Language
Processing

Information
Retrieval



Plagiarism
Detection

Sentiment
Analysis



Aim

To develop a tool capable of identify similarity
between pair of texts



Scope

Loading and preprocessing the question pairs dataset
Performing data analysis
Building a Siamese BERT model
Training model

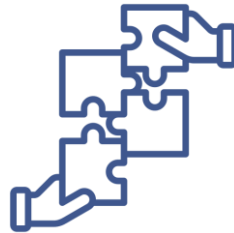
Methodology

Data loading, Preprocessing



Reading the dataset
Handling missing values
Text cleaning

Model Building



Encoding text for BERT
Compiling the model
Using L1 Distance Layer

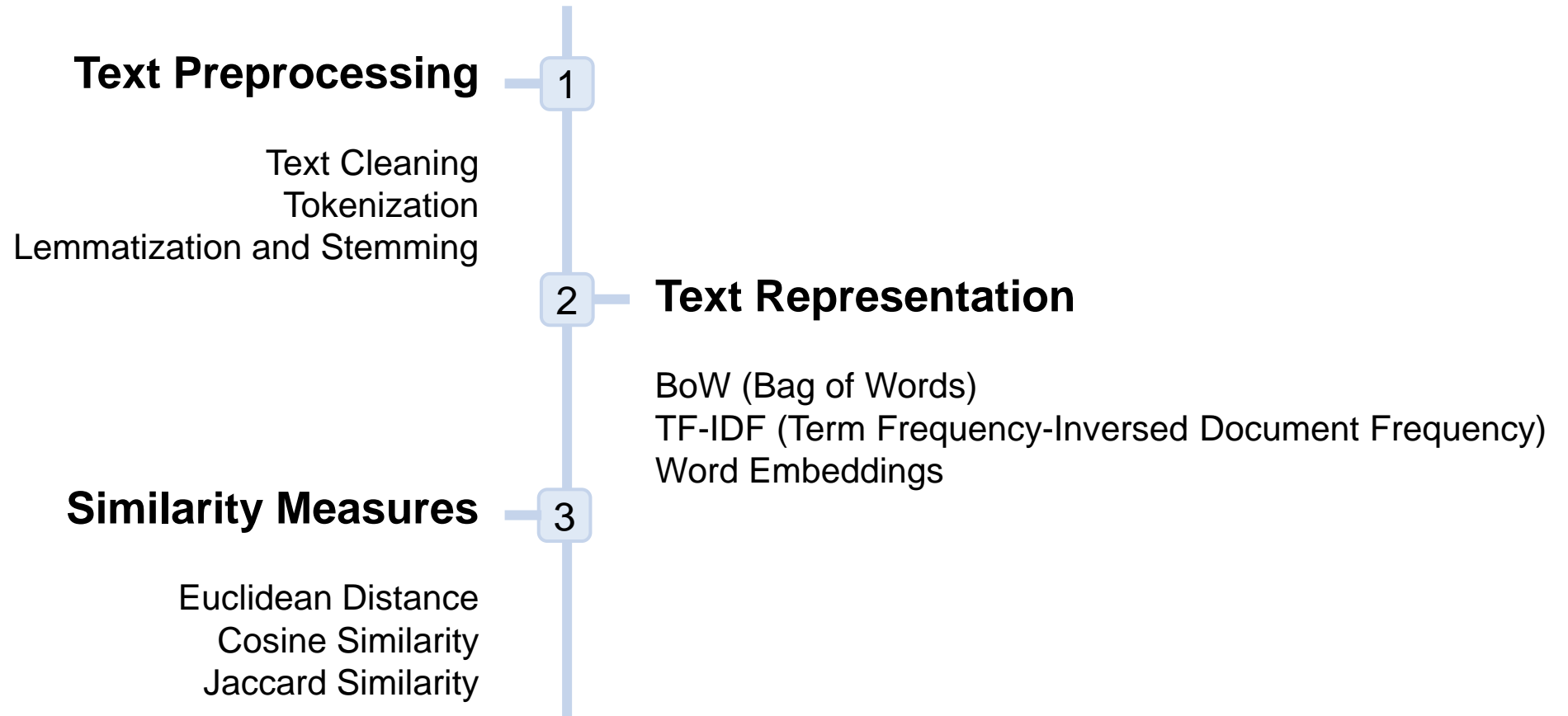
Model Evaluation



Predict similarity
View results – Confusion Matrix

Theoretical Part

Theoretical Foundations



Theoretical Foundations

Machine Learning Models

Supervised Models
Unsupervised Models

4

5

Deep Learning Models

Transformers
Siamese Networks

Tools and Libraries



NLTK

Text Preprocessing,
Tokenization,
Lemmatization



spaCy

Tokenization,
Lemmatization,
Part-of-speech
tagging



Transformers

Pretrained Models
(BERT and GPT)



TensorFlow and Keras

Deep learning libraries
- Build and Train
Neural Network Model

Practical Part

1. Loading and Preprocessing Data

Reading data from a csv file

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

```
train_data.head(10)
```

Cleaning and preprocessing text

```
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)
```

2. Data Exploration

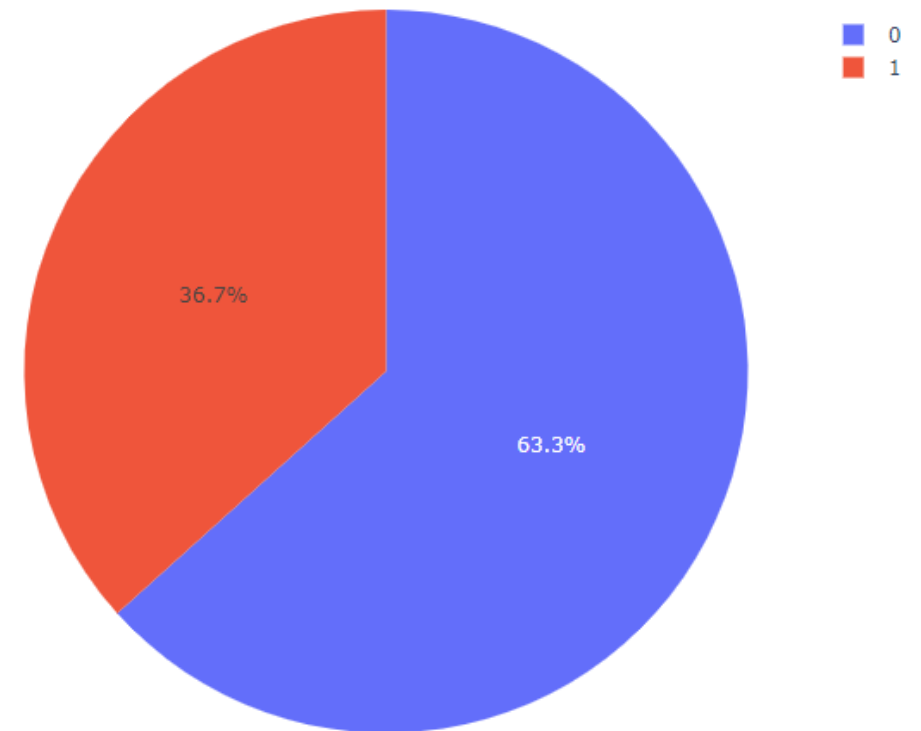
Analyzing the distribution of classes

```
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 distribution 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
```

3. Tokenization and Encoding

Using a tokenizer to convert text into sequence of tokens

```
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 a Model

```
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.opti
mizers.Adam(learning_rate=0.00001),metrics='accuracy')

    model.summary()
```

`resume_download` is deprecated and will be removed in version 1.0.0. Downloads always resume when possible. If you want to force a new download, use `force_download=True`.

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', 'cls.predictions.transform.dense.bias', 'cls.pre

- 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 BertForPreTraining 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)	TFBaseModelOutputWithPoolingAndCrossAttentions(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 and training the model

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

```
learning_rate_reduction = ReduceLROnPlateau(monitor='val_loss',  
                                             patience=3,  
                                             verbose=1,  
                                             factor=0.3,  
                                             min_lr=0.00000001)
```

```
# 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 (#1 Confusion Matrix)

```
y_pred = model.predict((X1_test_input_ids,  
                        X1_test_attention_masks,  
                        X2_test_input_ids,X2_test  
_attention_masks))
```

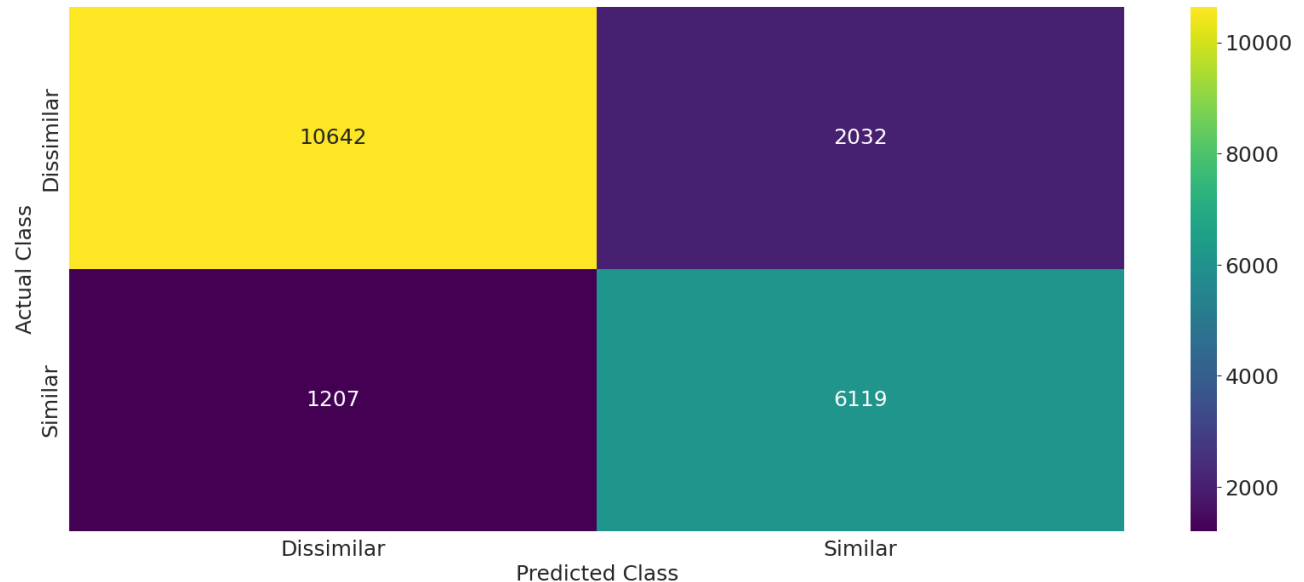
```
# Display confusion matrix to view results
```

```
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=['Dissimilar','Similar'])  
plt.xlabel('Predicted Class')  
plt.ylabel('Actual Class')  
plt.show()
```



Evaluating the model (#2 Classification Report)

```
# 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

Conclusion



Conclusion

Achieve the goal to identifying similar texts

Techniques and tools allow this task to be approached efficiently and accurately



Application

Improve applications in various domains -

Information retrieval

Online assistance

THANK YOU

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>