Key Feature extraction from classified summary of a Text file using BERT

Aastha Singh · Following

Published in Nerd For Tech · 6 min read · May 31, 2021

Harnessing the power of BERT embeddings

In this post, I'll show you how BERT solves a basic text summarization and categorization issue.

About BERT(Bidirectional Encoder Representations from Transformers)

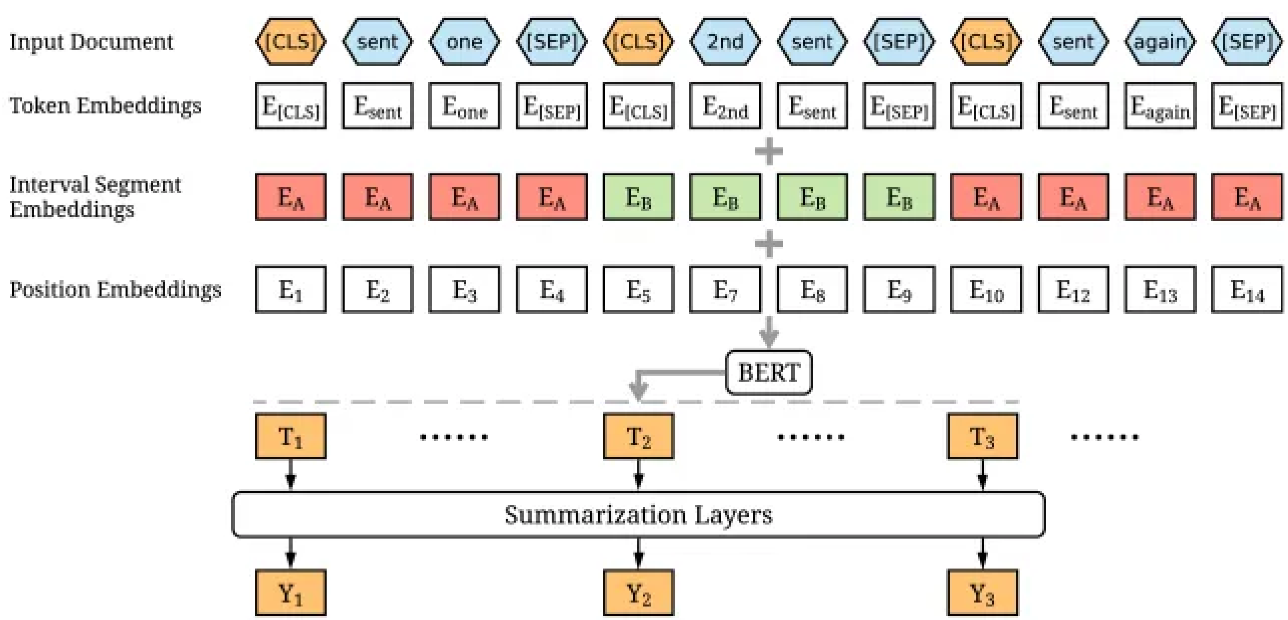
> *BERT, in a nutshell, is a model that understands how to represent text. You feed it a sequence, and it scans left and right a number of times before producing a vector representation for each word as an output.*
>
> *BERT and other Transformer encoder architectures have been wildly successful on a variety of tasks in NLP (natural language processing).*

Structure of BERT

1. The BERT summarizer

- It has 2 parts: a BERT encoder and a summarization classifier.
- In the encoder, we learn the interactions among tokens in our document while in the summarization classifier, we learn the interactions among sentences.

To assign each sentence a label , we need to add a token `[CLS]` before each sentence indicating whether the sentence should be included in the final summary.

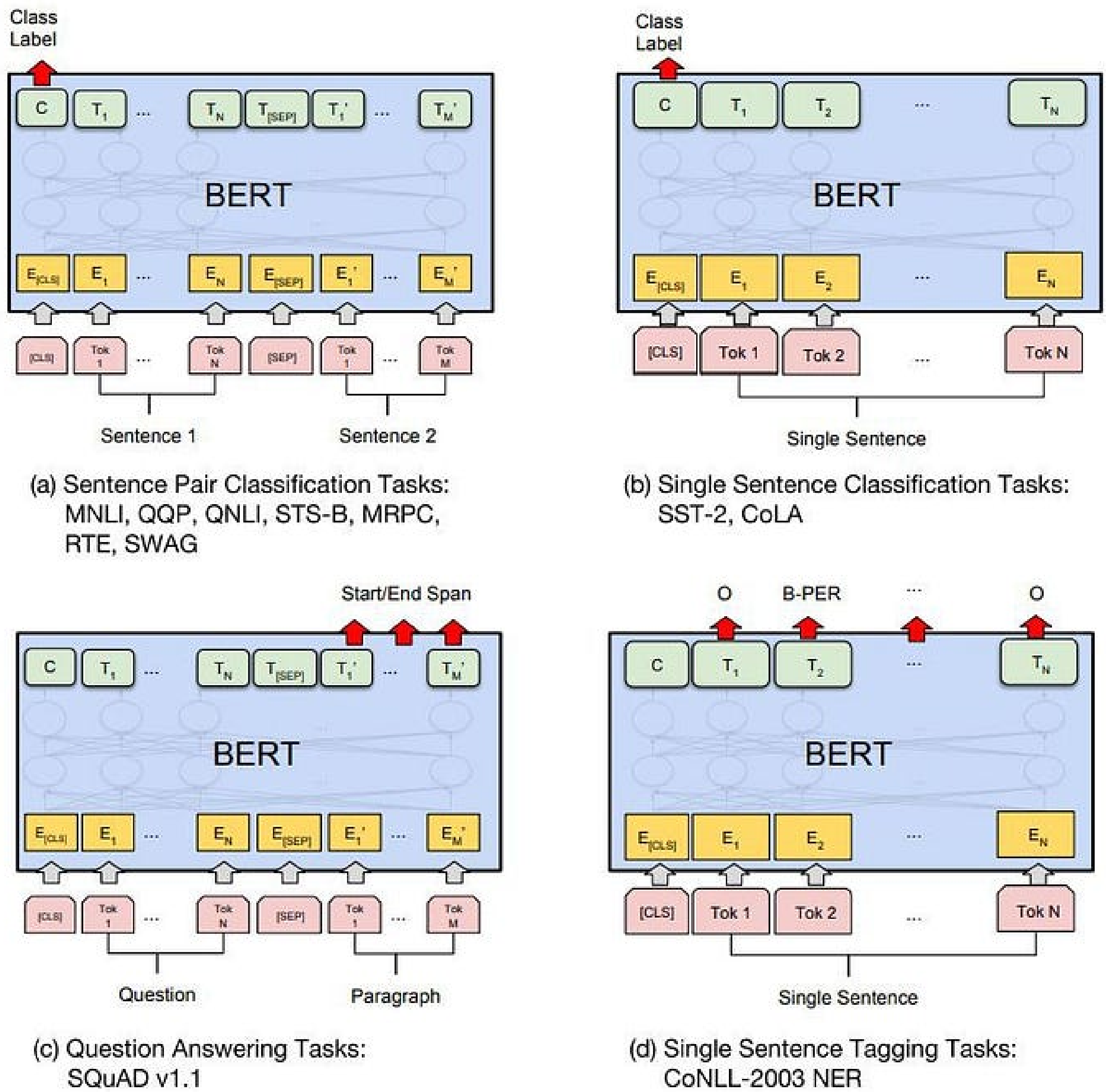


BERT structure for summarization

2. The BERT Classifier

Input — there's `[CLS]` token (classification) at the start of each sequence and a special `[SEP]` token that separates two parts of the input.

Output — for classification, we use the output of the first token (the `[CLS]` token). For more complicated outputs, we can use all the other tokens output.



Comparing BERT with XLNet & GPT-2, for Text Summarization based on performance

```
bert_model = Summarizer()
bert_summary = ''.join(bert_model(body, min_length=60))
print(bert_summary)
```

```
100%|██████████| 434/434 [00:00<00:00, 142793.22B/s]
100%|██████████| 1344997306/1344997306 [00:25<00:00, 53253939.01B/s]
100%|██████████| 231508/231508 [00:00<00:00, 10983089.36B/s]
Scientists say they have discovered a new species of orangutans on Indonesia's island of Sumatra. They say the animals are considered a new species because o
```

```
GPT2_model = TransformerSummarizer(transformer_type="GPT2",transformer_model_key="gpt2-medium")
full = ''.join(GPT2_model(body, min_length=60))
print(full)
```

```
100%|██████████| 718/718 [00:00<00:00, 456496.93B/s]
100%|██████████| 1520013706/1520013706 [00:28<00:00, 52676310.36B/s]
100%|██████████| 1042301/1042301 [00:00<00:00, 18217655.62B/s]
100%|██████████| 456318/456318 [00:00<00:00, 16614896.72B/s]
Scientists say they have discovered a new species of orangutans on Indonesia's island of Sumatra. The orangutans were found inside North Sumatra's Batang Tor
```

```
model = TransformerSummarizer(transformer_type="XLNet",transformer_model_key="xlnet-base-cased")
full = ''.join(model(body, min_length=60))
print(full)
```

```
100%|██████████| 760/760 [00:00<00:00, 647505.80B/s]
100%|██████████| 467042463/467042463 [00:09<00:00, 50831905.03B/s]
100%|██████████| 798011/798011 [00:00<00:00, 21278994.57B/s]
Scientists say they have discovered a new species of orangutans on Indonesia's island of Sumatra. They say the animals are considered a new species because o
```

Comparison after installing bert-extractive-summarizer, transformers==2.2.0, spaCy

Results :

- Terms of performance — GPT-2-medium is the best
- Terms of time taken — XLNet (11 s) GPT-2 medium (35s) Bert (30s)
- Terms of ease of use — BERT

Step 1: Choosing the BERT Model

There are multiple BERT models available.

- BERT-Base,
- Small BERTs
- ALBERT
- BERT Experts
- Electra

> *Final model used : DistilBERT*

> *It is a small, fast, cheap and light Transformer model trained by distilling BERT base.*
>
> *It has 40% less parameters than bert-base-uncased, runs 60% faster while preserving over 95% of BERT's performances as measured on the GLUE language understanding benchmark.*

Step 2: Text classification using BERT

> Your mind must be racing with all of the possibilities that BERT has opened up. We can use BERT's vast knowledge repository in a myriad of contexts for our NLP applications!

1. Let's Setup!

I have used the AdamW optimizer from tensorflow/models.

```
pip install bert-for-tf2
```

```
import tensorflow as tf
from tensorflow.keras.layers import Dense, Input, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.models import Model
from tensorflow.keras.callbacks import ModelCheckpoint
import tensorflow_hub as hub
from bert import bert_tokenization
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

2. Importing and Preprocessing the Dataset

Source : Kaggle

> *Dataset consistes of consumers' complaints sent by the CFPB about financial products and services to companies for response to help improve the financial marketplace.*

```
df=pd.read_csv("../input/us-consumer-finance-complaints/consumer_complaints.csv")
```

```
/opt/conda/lib/python3.7/site-packages/IPython/core/interactiveshell.py:3156: DtypeWarning: Columns (5,11) have mixed type
s.Specify dtype option on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

```
df.head()
```

date_received	product	sub_product	issue	sub_issue	consumer_complaint_narrative	company_public_response	company
08/30/2013	Mortgage	Other mortgage	Loan modification,collection,foreclosure	NaN	NaN	NaN	U.S. Bancorp
08/30/2013	Mortgage	Other mortgage	Loan servicing, payments, escrow account	NaN	NaN	NaN	Wells Fargo & Company
08/30/2013	Credit reporting	NaN	Incorrect information on credit report	Account status	NaN	NaN	Wells Fargo & Company
08/30/2013	Student loan	Non-federal student loan	Repaying your loan	Repaying your loan	NaN	NaN	Navient Solutions, Inc.
08/30/2013	Debt collection	Credit card	False statements or representation	Attempted to collect wrong amount	NaN	NaN	Resurgent Capital Services L.P.

Loading the dataset

2.1. Feature Selection

I have selected the columns that were directly related to resloving the issues and classifying them into the product classes

The output below shows that our dataset has 555,957 rows and 18 columns.

```
df.shape
```

```
(555957, 18)
```

```
df.isnull().sum()
```

```
date_received                       0
product                             0
sub_product                    158322
issue                               0
sub_issue                      343335
consumer_complaint_narrative   489151
company_public_response        470833
company                             0
state                            4887
zipcode                          4505
tags                           477998
consumer_consent_provided      432499
submitted_via                       0
date_sent_to_company                0
company_response_to_consumer        0
timely_response                     0
consumer_disputed?                  0
complaint_id                        0
dtype: int64
```

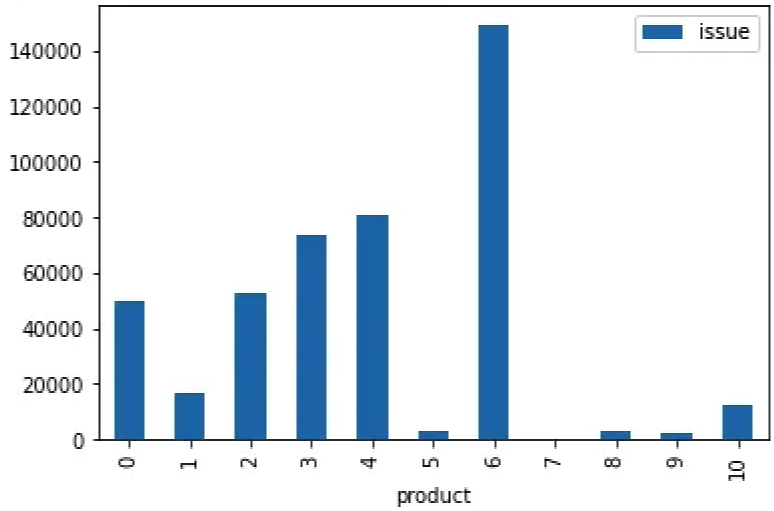
+ Code + Markdown

```
df=df[["product","issue"]]
```

Selected 2 out of 18 features.

```
df_train.groupby('product').count().plot.bar()
```

<AxesSubplot:xlabel='product'>



Issues Classified into 10 product categories

2.2. Label encoding

I have label encoded the ***Product*** column to convert the text format into label format using `LabelEncoder` .

> `LabelEncoder` *: It allows to assign ordinal levels to categorical data.*
>
> `fit_transform` *(y): Fit label encoder and return encoded labels.*

```
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
df["product"]=le.fit_transform(df["product"])
```

```
from sklearn.model_selection import train_test_split
df_train,df_test=train_test_split(df,test_size=0.2, random_state=42)
```

```
df_train.head()
```

	product	issue
514459	3	Incorrect information on credit report
119480	4	Cont'd attempts collect debt not owed
381413	6	Loan modification,collection,foreclosure
425165	0	Deposits and withdrawals
467163	2	Transaction issue

3. Creating a BERT Tokenizer

> *Text inputs need to be transformed to **numeric token ids** and arranged in several **Tensors** before being input to BERT.*
>
> *Tokenization refers to dividing a sentence into individual words. To tokenize our text, we will be using the BERT tokenizer.*

Importing the pre-trained model and tokenizer which is specific to BERT

- Create a BERT embedding layer by importing the BERT model from `hub.KerasLayer`
- Retrieve the BERT ***vocabulary file*** in the form a ***numpy array.***
- Set the text to lowercase and pass our `vocab_file` and `do_lower` variables to the `BertTokenizer` object.
- Initialise tokenizer_for_bert.

```
bert_layer=hub.KerasLayer("https://tfhub.dev/tensorflow/bert_en_uncased_L-12_H-768_A-12/1",trainable=True, name = 'keras_bert_layer')

vocab_file=bert_layer.resolved_object.vocab_file.asset_path.numpy()

do_lower_case=True

tokenizer_for_bert=bert_tokenization.FullTokenizer(vocab_file,do_lower_case)
```

```
print("The length of the vocab in our tokenizer is:",len(tokenizer_for_bert.vocab))
```

```
The length of the vocab in our tokenizer is: 30522
```

4. Defining helper function for text preprocessing

- The *encode_text* function is converting raw text data into encoded text('CLS'+token+ 'SEP')which is fitted and converted to token
- To create sentences of equal length, I have padded the ***token_ids, mask_ids, segment_ids*** to truncate the tokens with the provided batch size.

```
def encode_text(texts,tokenizer_for_bert, max_len=512):
    all_token_ids=[]
    all_masks=[]
    all_segments=[]

    for text in texts:
        tokens=tokenizer_for_bert.tokenize(text)
        tokens=tokens[:max_len-2]
        input_sequence=["[CLS]"]+tokens+["[SEP]"]
        pad_len=max_len-len(input_sequence)
        token_ids=tokenizer_for_bert.convert_tokens_to_ids(input_sequence)
        token_ids+=[0]*pad_len
        pad_masks=[1]*len(input_sequence)+[0]*pad_len
        segment_ids=[0]*max_len

        all_token_ids.append(token_ids)
        all_masks.append(pad_masks)
        all_segments.append(segment_ids)

    return np.array(all_token_ids),np.array(all_masks), np.array(all_segments)
```

- The model will take strings as input, and return appropriately formatted objects which can be passed to BERT.

```
test_text = "There was a blast in Lebanon the previous day. 130 people are reported to be dead. "

print ("Test text after tokenization: " , ["[CLS]"] + tokenizer_for_bert.tokenize( test_text) + ["[SEP]"] )

print ("Test text after encoding: " ,encode_text( [test_text], tokenizer_for_bert, 7 ) )
```

```
Test text after tokenization:  ['[CLS]', 'there', 'was', 'a', 'blast', 'in', 'lebanon', 'the', 'previous', 'day', '.', '13
0', 'people', 'are', 'reported', 'to', 'be', 'dead', '.', '[SEP]']
Test text after encoding:  (array([[ 101, 2045, 2001, 1037, 8479, 1999,  102]]), array([[1, 1, 1, 1, 1, 1, 1]]), array([[0
, 0, 0, 0, 0, 0, 0]]))
```

Passing text in **test_text** to **encode_text** function

> *Since this text preprocessor is a TensorFlow model, It can be included in any model directly.*

5. Defining the Model

- Create a very simple fine-tuned model, with the preprocessing model, the selected BERT model, one Dense and a Dropout layer for regularization.
- As you can see, there are 3 outputs from the preprocessing that a BERT model would use (`input_words_id` , `input_mask` and `segment_ids`).

Batch size = 40 implies that if the input is >than 40, it will be truncated to 40 tokens and if the input is <40 it will pad it to 40 tokens.

```
def bert_model(bert_layer,max_len=512):
    #Input to bert layer
    input_word_ids = Input(shape=(max_len,), dtype=tf.int32, name="input_word_ids")
    input_mask = Input(shape=(max_len,), dtype=tf.int32, name="input_mask")
    segment_ids = Input(shape=(max_len,), dtype=tf.int32, name="segment_ids")

    #Output from bert layer
    bert_layer_out = bert_layer([input_word_ids, input_mask, segment_ids]) # Python list of 2 tensors with shape (batch_size, 768) and (batch,

    #Extrating Embedding for CLS token comming out of bert layer. Note CLS is the first token
    cls_out = bert_layer_out[1][:,0,:] # Getting hidden-state of 1st tokens from second tensor in bert_layer_out, Tensor shape - (batch size,

    out = Dense(10, activation='softmax')(cls_out)


    #Model creation using inputs and output
    model = Model(inputs=[input_word_ids, input_mask, segment_ids], outputs=out, name='deeplearning_bert__model')



    learning_rate = 1e-6 # modify learning rate,as needed

    #Compiles Model depending on model type and number of classes. Loss function as well as metrics is used accordingly

    model.compile(Adam(lr= learning_rate), loss='sparse_categorical_crossentropy', metrics=['accuracy']) # ** For Binary classification

    return model
```

+ Code + Markdown

```
max_len=40
```

6. Converting the train text in encoded format

```
train_input=encode_text(df_train["issue"].values,tokenizer_for_bert,max_len=max_len)

y_train=df_train["product"].values #converted product column into array(3)
```

7. Fine-Tuning the model for text classification

Fine-tuning follows the optimizer set-up from BERT pre-training: It uses the *AdamW* optimizer

> *BERT was originally trained with: the "Adaptive Moments" (Adam). This optimizer minimizes the prediction loss and does regularization by weight decay.*

To increase the accuracy, increase the no. of epochs

```
#Model Training (Fine-tuning for text classification)
epochs = 2       #Modify as neded
batch_size = 32  #Modify as needed
train_history = model.fit(train_input, y_train ,epochs= epochs,batch_size= batch_size, verbose=1)
```

```
Epoch 1/2
13899/13899 [==============================] - 2248s 160ms/step - loss: nan - accuracy: 0.1124
Epoch 2/2
13899/13899 [==============================] - 2226s 160ms/step - loss: nan - accuracy: 0.1129
```

Building Pipeline

> Flow of Pipeline :
>
> Text Summarization using BERT>Text Classification using BERT >Name Entity Recognition using spaCy

For Text Summarization:

Extractive, abstractive, and mixed summarization strategies are most commonly used.

- *Extractive strategies* — It selects the top N sentences that best represent the article's important themes.
- *Abstractive summaries* — It attempts to rephrase the article's main ideas in new words.

1. **Installing bert-extractive-summarizer :**
2. **Installing spaCy :** The smallest English language model takes only a moment to download as it's around 11MB

> *This tool utilizes the* **HuggingFace** *Pytorch transformers library to run extractive summarizations.*

> *This works by first embedding the sentences, then running a clustering algorithm, finding the sentences that are closest to the cluster's centroids*

```
!pip install bert-extractive-summarizer
!pip install transformers==2.2.0
!pip install spacy
```

```
from summarizer import Summarizer,TransformerSummarizer
!python -m spacy download en_core_web_sm
```

```
import spacy
nlp=spacy.load("en_core_web_sm")
```

3. Defining the pipeline function

```
def text_summarization_classification(text,model):
    bert_model = Summarizer()
    bert_summary = ''.join(bert_model(text, min_length=60)) #one line
    print(bert_summary)
    prediction=model.predict(encode_text([text],tokenizer_for_bert,
                                         max_len=max_len)) #classification
    return prediction,summary
```

Testing the Model

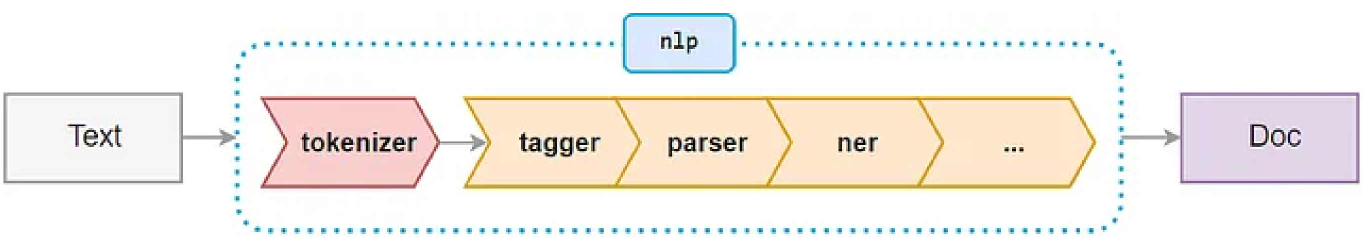
Passing Input to the trained model to summarize and then classify the text.

```
text="A mortgage is a loan that the borrower uses to purchase or maintain a
```

```
prediction,summary=text_summarization_classification(text,model)
```

Key Feature Extraction using spaCyNER

> About spaCy Named Entity Recognition
>
> spaCy’s Named Entity Recognition (NER) locates and identifies the named entities present in unstructured text into the standard categories such as person names, locations, organizations, time expressions, quantities, monetary values, percentage, codes etc.



Structure of spaCy NER

Accessing the Entity Annotations on the generated summary of the text

```
bert_model = Summarizer()
bert_summary = ''.join(bert_model(text, min_length=60))
```

```
doc=nlp(bert_summary)
for ent in doc.ents:
    print(ent.text, ent.start_char, ent.end_char, ent.label_)
```

`Doc.ents` are token spans with their own set of annotations

`ent.text`	The original entity text
`ent.label`	The entity type's hash value
`ent.label_`	The entity type's string description
`ent.start`	The token span's *start* index position in the Doc
`ent.end`	The token span's *stop* index position in the Doc
`ent.start_char`	The entity text's *start* index position in the Doc
`ent.end_char`	The entity text's *stop* index position in the Doc

Entity Annotations

Further Thoughts

For a much faster approach, I can directly extract the key features by extracting **noun phrases** from the generated text summary using spaCy.

This would help to get the **most common nouns, verbs, adverbs** and so on by counting frequency of all the tokens in the text file.

Feel free to play around with spaCy as there is a lot more built-in functionality available. I will be doing this in my next blog. Stay connected!

Artificial Intelligence | Machine Learning | Deep Learning | NLP | Bert

Written by Aastha Singh

Following

Shedding light to how our brains process, using AI !

More from Aastha Singh and Nerd For Tech

Aastha Singh in Analytics Vidhya

Evolving with BERT: Introduction to RoBERTa

No matter how good it is, it can always get better, and that's the exciting part.

8 min read · Jun 28, 2021

65 2

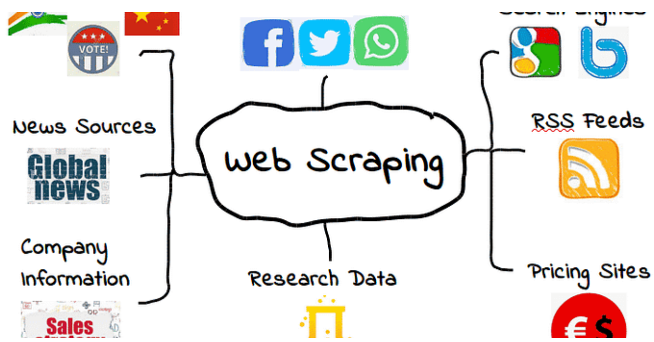
Gunjan Sahu in Nerd For Tech

Flipkart Data Engineer Interview

Flipkart Data Engineer Interview

6 min read · Jul 12

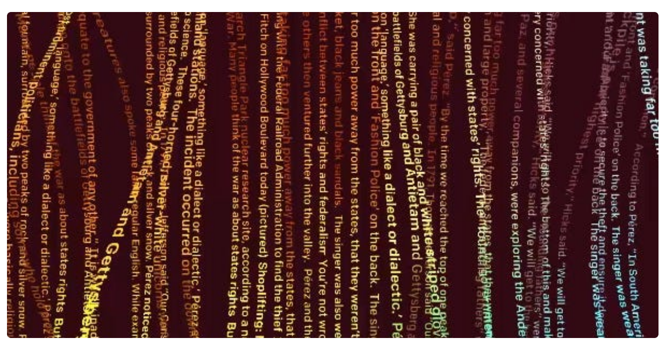
180 1



Yennhi95zz in Nerd For Tech

$4000 Freelance Income: Exploring More Than Just Web...

I've just been using Upwork for 3 months, but in that time I've completed various projects i...



Aastha Singh in Geek Culture

Auto-code generation using GPT-2

Let's code faster with AI

· 7 min read · Jul 17

374 2

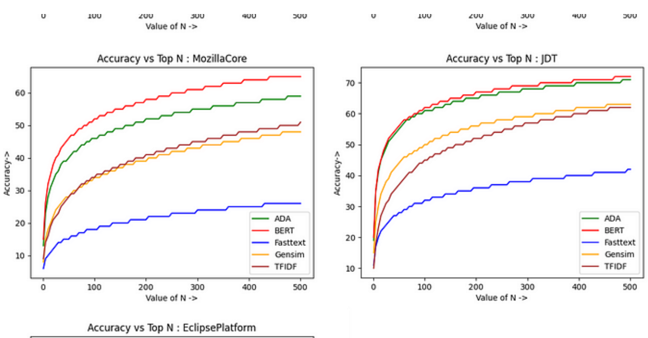
6 min read · Jun 16, 2021

100

See all from Aastha Singh

See all from Nerd For Tech

Recommended from Medium



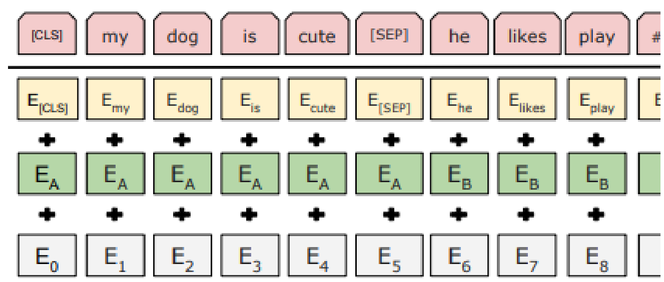
Avinash Patil

Embeddings: BERT better than ChatGPT4?

In this study, we compared the effectiveness of semantic textual similarity methods for…

4 min read · Sep 19

3 1



Zain ul Abideen

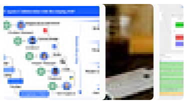
A Comparative Analysis of LLMs like BERT, BART, and T5

Exploring Language Models

6 min read · Jun 26

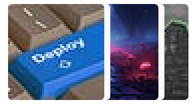
20 1

Lists



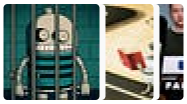
Natural Language Processing

669 stories · 283 saves



Predictive Modeling w/ Python

20 stories · 452 saves

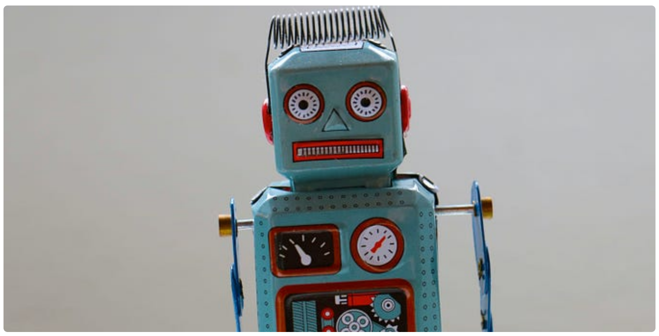


AI Regulation

6 stories · 138 saves

Practical Guides to Machine Learning

10 stories · 519 saves



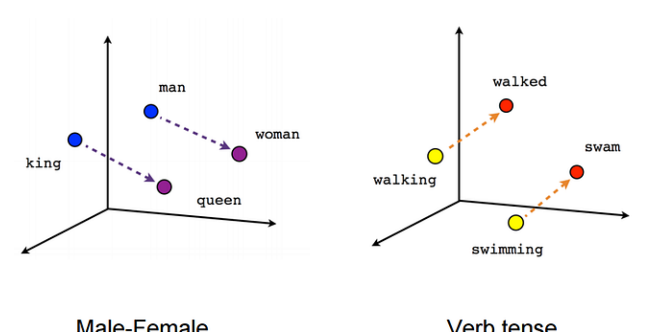
AR

BERT for Sequence Classification from Scratch—Code and Theory

Coding BERT for sequence classification from scratch serves as an exercise to better...

16 min read · Aug 3

3 2



Maninder Singh

Accelerate Your Text Data Analysis with Custom BERT Word...

One thing is for sure the way humans interact with each other naturally is one of the most...

4 min read · Apr 24

156

The Python Lab

Ahmet Taşdemir

How to Perform Sentiment Analysis using BERT in Python

Sentiment analysis, also known as opinion mining, is a field within natural language...

· 5 min read · May 23

26

Fine-Tuning DistilBERT for Emotion Classification

In this post, we will walk through the process of fine-tuning the DistilBERT model for...

8 min read · Jun 14

See more recommendations