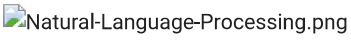
```
import re
import string
import numpy as np
import random
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
from plotly import graph_objs as go
import plotly.express as px
import plotly.figure_factory as ff
from collections import Counter

from PIL import Image
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator


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

from tqdm import tqdm
import os
import nltk
import spacy
import random
from spacy.util import compounding
from spacy.util import minibatch

from collections import defaultdict
from collections import Counter

import keras
from keras.models import Sequential
from keras.initializers import Constant
from keras.layers import (LSTM,
                          Embedding,
                          BatchNormalization,
                          Dense,
                          TimeDistributed,
                          Dropout,
                          Bidirectional,
                          Flatten,
                          GlobalMaxPool1D)
from keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.layers import Embedding
from keras.callbacks import ModelCheckpoint, ReduceLROnPlateau
from keras.optimizers import Adam

from sklearn.metrics import (
    precision_score,
    recall_score,
    f1_score,
    classification_report,
    accuracy_score
)
```

Natural Languague Processing 📝 A complete Guide

Natural Language Processing or NLP is a branch of Artificial Intelligence which deal with bridging the machines understanding humans in their Natural Language. Natural Language can be in form of text or sound, which are used for humans to communicate each other. NLP can enable humans to communicate to machines in a natural way.

Text Classification is a process involved in Sentiment Analysis. It is classification of peoples opinion or expressions into different sentiments. Sentiments include Positive, Neutral, and Negative, Review Ratings and Happy, Sad. Sentiment Analysis can be done on different consumer centered industries to analyse people's opinion on a particular product or subject.

Natural language processing has its roots in the 1950s. Already in 1950, Alan Turing published an article titled "Computing Machinery and Intelligence" which proposed what is now called the Turing test as a criterion of intelligence, a task that involves the automated interpretation and generation of natural language, but at the time not articulated as a problem separate from artificial intelligence.



In this kernel we are going to focus on text classification and sentiment analysis part. In the next lessons we will study Information retrival, Question answering, etc

Table of Content

1. Loading Data 💎

Just load the dataset and global variables for colors and so on.

```
# Defining all our palette colours.
primary_blue = "#496595"
primary_blue2 = "#85a1c1"
primary_blue3 = "#3f4d63"
primary_grey = "#c6ccd8"
primary_black = "#202022"
primary_bgcolor = "#f4f0ea"

primary_green = px.colors.qualitative.Plotly[2]


df = pd.read_csv("dataset.csv", encoding="latin-1")

df = df.dropna(how="any", axis=1)
df.columns = ['target', 'message' , 'lang' , 'sentiment']

df.head()
```

```
df['message_len'] = df['message'].apply(lambda x: len(x.split(' ')))
df.head()
```

	target	message	lang	sentiment	message_len
0	spam	No 1 POLYPHONIC tone 4 ur mob every week! Just...	english	Positive	31
1	spam	Avez-vous entendu parler de la nouvelle "Divor...	french	Neutral	16
2	ham	à¤¹à¤¾à¤¯ à¤¯à¤¹ à¤□à¤□à¥□ à¤□à¥□ à¤¤à¤°à¤¹ à¤¸...	hindi	Neutral	10
3	ham	Donc Ã§a veut dire que tu penses toujours au teju	french	Negative	10
4	spam	Senden Sie ein Logo 2 ur Liebhaber - 2 Namen d...	german	Positive	33

```
max(df['message_len'])
```

```
351
```

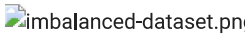
2. EDA

Now we are going to take a look about the target distribution and the messages length.

Balanced Dataset: — Let's take a simple example if in our data set we have positive values which are approximately same as negative values. Then we can say our dataset in balance.

Consider Orange color as a positive values and Blue color as a Negative value. We can say that the number of positive values and negative values in approximately same.

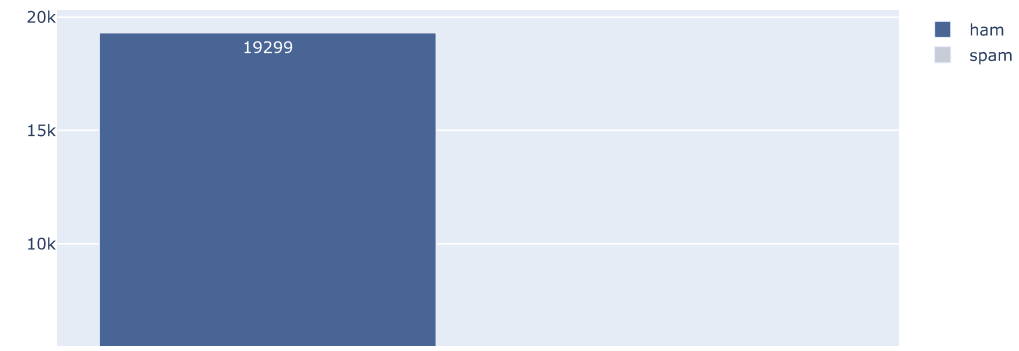
Imbalanced Dataset: — If there is the very high different between the positive values and negative values. Then we can say our dataset in Imbalance Dataset.




```
balance_counts = df.groupby('target')['target'].agg('count').values
balance_counts
```

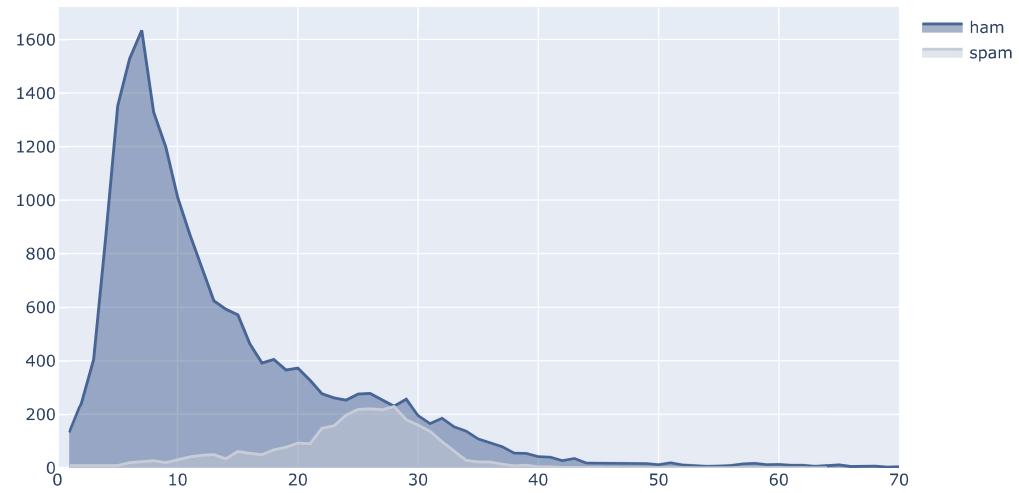
```
array([19299,  2988])
```

```
fig = go.Figure()
fig.add_trace(go.Bar(
    x=['ham'],
    y=[balance_counts[0]],
    name='ham',
    text=[balance_counts[0]],
    textposition='auto',
    marker_color=primary_blue
))
fig.add_trace(go.Bar(
    x=['spam'],
    y=[balance_counts[1]],
    name='spam',
    text=[balance_counts[1]],
    textposition='auto',
    marker_color=primary_grey
))
fig.update_layout(
    title='<span style="font-size:32px; font-family:Times New Roman">Dataset distribution by target</span>'
)
fig.show()
```



As we can see, the classes are imbalanced, so we can consider using some kind of resampling. We will study later. Anyway, it doesn't seem to be necessary.

```
ham_df = df[df['target'] == 'ham']['message_len'].value_counts().sort_index()
spam_df = df[df['target'] == 'spam']['message_len'].value_counts().sort_index()

fig = go.Figure()
fig.add_trace(go.Scatter(
    x=ham_df.index,
    y=ham_df.values,
    name='ham',
    fill='tozeroy',
    marker_color=primary_blue,
))
fig.add_trace(go.Scatter(
    x=spam_df.index,
    y=spam_df.values,
    name='spam',
    fill='tozeroy',
    marker_color=primary_grey,
))
fig.update_layout(
    title='<span style="font-size:32px; font-family:Times New Roman">Data Roles in Different Fields</span>'
)
fig.update_xaxes(range=[0, 70])
fig.show()
```



As we can see, the `ham` message length tend to be lower than `spam` message length.

3. Data Pre-processing 🛠

Now we are going to engineering the data to make it easier for the model to clasiffy.

This section is very important to reduce the dimensions of the problem.

3.1 Cleaning the corpus 🛠

```
# Special thanks to https://www.kaggle.com/tanulsingh077 for this function
def clean_text(text):
    '''Make text lowercase, remove text in square brackets,remove links,remove punctuation
    and remove words containing numbers.'''
    text = str(text).lower()
    text = re.sub('\[.*?\]', '', text)
    text = re.sub('https?://\S+|www\.\S+', '', text)
    text = re.sub('<.*?>+', '', text)
    text = re.sub('[%s]' % re.escape(string.punctuation), '', text)
    text = re.sub('\n', '', text)
    text = re.sub('\w*\d\w*', '', text)
    return text


df['message_clean'] = df['message'].apply(clean_text)
df.head()
```

	target	message	lang	sentiment	message_len	message_cle
0	spam	No 1 POLYPHONIC tone 4 ur mob every week! Just...	english	Positive	31	no polyphonic tone ur mob every week just t
1	spam	Avez-vous entendu parler de la nouvelle "Divor...	french	Neutral	16	avezvous entendu parler de la nouvelle divorce
2	ham	à¤¹à¤¾à¤ à¤¹à¤à¤¹à¤à¤¹à¤à¤ à¤¤à¤¬à¤¹à¤...	hindi	Neutral	10	à¤¹à¤¾à¤ à¤¹à¤à¤¹à¤à¤¹à¤à¤ à¤¤à¤¬à¤¹à¤ à¤
3	ham	Donc Ã§a veut dire que tu penses toujours au teju	french	Negative	10	donc ã§a veut dire que tu penses toujours au te
4	spam	Senden Sie ein Logo 2 ur Liebhaber - 2 Namen d...	german	Positive	33	senden sie ein logo ur liebhaber namen dur

Stopwords

Stopwords are commonly used words in English which have no contextual meaning in an sentence. So therefore we remove them before classification. Some examples removing stopwords are:

stopwords.png

```
import nltk
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
True
```

```
stop_words = stopwords.words('english')
more_stopwords = ['u', 'im', 'c']
stop_words = stop_words + more_stopwords

def remove_stopwords(text):
    text = ' '.join(word for word in text.split(' ') if word not in stop_words)
    return text

df['message_clean'] = df['message_clean'].apply(remove_stopwords)
df.head()
```

	target	message	lang	sentiment	message_len	message_cle
0	spam	No 1 POLYPHONIC tone 4 ur mob every week! Just...	english	Positive	31	polyphonic tone ur mob every week txt tor
1	spam	Avez-vous entendu parler de la nouvelle "Divor...	french	Neutral	16	avezvous entendu parler de la nouvelle divorce
2	ham	à¤¹à¤¾à¤ à¤¹à¥ à¤à¤¤ à¤¤à¤°à¤¹ à¤¸...	hindi	Neutral	10	à¤¹à¤¾à¤ à¤à¤¤ à¤¤à¤°à¤¹ à¤
3	ham	Donc Ã§a veut dire que tu penses toujours au teju	french	Negative	10	donc ã§a veut dire que tu penses toujours au te
4	spam	Senden Sie ein Logo 2 ur Liebhaber - 2 Namen d...	german	Positive	33	senden sie ein logo ur liebhaber namen dur

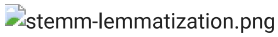
3.2 Stemming 🛠

Stemming/ Lematization

For grammatical reasons, documents are going to use different forms of a word, such as *write, writing and writes*. Additionally, there are families of derivationally related words with similar meanings. The goal of both stemming and lemmatization is to reduce inflectional forms and sometimes derivationally related forms of a word to a common base form.

Stemming usually refers to a process that chops off the ends of words in the hope of achieving goal correctly most of the time and often includes the removal of derivational affixes.

Lemmatization usually refers to doing things properly with the use of a vocabulary and morphological analysis of words, normally aiming to remove inflectional endings only and to return the base and dictionary form of a word



As far as the meaning of the words is not important for this study, we will focus on stemming rather than lemmatization.

Stemming algorithms

There are several stemming algorithms implemented in NLTK Python library:

1. **PorterStemmer** uses *Suffix Stripping* to produce stems. **PorterStemmer is known for its simplicity and speed**. Notice how the PorterStemmer is giving the root (stem) of the word "cats" by simply removing the 's' after cat. This is a suffix added to cat to make it plural. But if you look at 'trouble', 'troubling' and 'troubled' they are stemmed to 'trouble' because *PorterStemmer algorithm does not follow linguistics rather a set of 05 rules for different cases that are applied in phases (step by step) to generate stems*. This is the reason why PorterStemmer does not often generate stems that are actual English words. It does not keep a lookup table for actual stems of the word but applies algorithmic rules to generate stems. It uses the rules to decide whether it is wise to strip a suffix.
2. One can generate its own set of rules for any language that is why Python nltk introduced **SnowballStemmers** that are used to create non-English Stemmers!
3. **LancasterStemmer** (Paice-Husk stemmer) is an iterative algorithm with rules saved externally. One table containing about 120 rules indexed by the last letter of a suffix. On each iteration, it tries to find an applicable rule by the last character of the word. Each rule specifies either a deletion or replacement of an ending. If there is no such rule, it terminates. It also terminates if a word starts with a vowel and there are only two letters left or if a word starts with a consonant and there are only three characters left. Otherwise, the rule is applied, and the process repeats.

```
stemmer = nltk.SnowballStemmer("english")

def stemm_text(text):
    text = ' '.join(stemmer.stem(word) for word in text.split(' '))
    return text


df['message_clean'] = df['message_clean'].apply(stemm_text)
df.head()
```

	target	message	lang	sentiment	message_len	message_cle
0	spam	No 1 POLYPHONIC tone 4 ur mob every week! Just...	english	Positive	31	polyphon tone ur mob everi week txt tone
1	spam	Avez-vous entendu parler de la nouvelle "Divor...	french	Neutral	16	avezv entendu parler de la nouvell divorc barl

3.3 All together 🛠

```
def preprocess_data(text):
    # Clean puntuation, urls, and so on
    text = clean_text(text)
    # Remove stopwords
    text = ' '.join(word for word in text.split(' ') if word not in stop_words)
    # Stemm all the words in the sentence
    text = ' '.join(stemmer.stem(word) for word in text.split(' '))

    return text


df['message_clean'] = df['message_clean'].apply(preprocess_data)
df.head()
```

	target	message	lang	sentiment	message_len	message_cle
0	spam	No 1 POLYPHONIC tone 4 ur mob every week! Just...	english	Positive	31	polyphon tone ur mob everi week txt tone
1	spam	Avez-vous entendu parler de la nouvelle "Divor...	french	Neutral	16	avezv entendu parler de la nouvel divorc barb
2	ham	à¤¹à¤¾à¤¯ à¤¯à¤¹ à¤à¤•à¤¦à¤® à¤¸à¤¹à¥€ à¤¸à¤®à¤¯ à¤¹à¥ˆ à¤¸à¤¬...	hindi	Neutral	10	à¤¹à¤¾à¤¯ à¤¯à¤¹ à¤à¤•à¤¦à¤® à¤¸à¤¹à¥€ à¤¸à¤®à¤¯ à¤
3	ham	Donc Ã§a veut dire que tu penses toujours au teju	french	Negative	10	donc ã§a veut dire que tu pen toujour au te
4	spam	Senden Sie ein Logo 2 ur Liebhaber - 2 Namen d...	german	Positive	33	senden sie ein logo ur liebhab namen durch

3.4 Target encoding 🛠

```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
le.fit(df['target'])

df['target_encoded'] = le.transform(df['target'])
df.head()
```

	target	message	lang	sentiment	message_len	message_clea
0	spam	No 1 POLYPHONIC tone 4 ur mob every week! Just...	english	Positive	31	polyphon tone ur mob everi week txt tone ..
1	spam	Avez-vous entendu parler de la nouvelle "Divor...	french	Neutral	16	avezv entendu parler de la nouvel divorc barbi..
2	ham	à¤¹à¤¾à¤¯ à¤¯à¤¹ à¤à¤•à¤¦à¤® à¤¸à¤¹à¥€ à¤¸à¤®à¤¯ à¤¹à¥ˆ à¤¸à¤¬...	hindi	Neutral	10	à¤¹à¤¾à¤¯ à¤¯à¤¹ à¤à¤•à¤¦à¤® à¤¸à¤¹à¥€ à¤¸à¤®à¤¯ à¤¹ à¤¸à¤¬...
3	ham	Donc Ã§a veut dire que tu penses toujours au teju	french	Negative	10	donc ã§a veut dire que tu pen toujour au teju
4	spam	Senden Sie ein Logo 2 ur Liebhaber - 2 Namen d...	german	Positive	33	senden sie ein logo ur liebhab namen durch ..

5. Vectorization

Currently, we have the messages as lists of tokens (also known as lemmas) and now we need to convert each of those messages into a vector the SciKit Learn's algorithm models can work with.

We'll do that in three steps using the bag-of-words model:

1. Count how many times does a word occur in each message (Known as term frequency)
2. Weigh the counts, so that frequent tokens get lower weight (inverse document frequency)
3. Normalize the vectors to unit length, to abstract from the original text length (L2 norm)

Let's begin the first step:

Each vector will have as many dimensions as there are unique words in the SMS corpus. We will first use SciKit Learn's **CountVectorizer**. This model will convert a collection of text documents to a matrix of token counts.

We can imagine this as a 2-Dimensional matrix. Where the 1-dimension is the entire vocabulary (1 row per word) and the other dimension are the actual documents, in this case a column per text message.


```
# how to define X and y (from the SMS data) for use with COUNTVECTORIZER
x = df['message_clean']
y = df['target_encoded']

print(len(x), len(y))
```

```
22287 22287
```

```
# Split into train and test sets
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x, y, random_state=42)
print(len(x_train), len(y_train))
print(len(x_test), len(y_test))
```

```
16715 16715
5572 5572
```

```
from sklearn.feature_extraction.text import CountVectorizer

# instantiate the vectorizer
vect = CountVectorizer()
vect.fit(x_train)
```

```
CountVectorizer()
```

```
# Use the trained to create a document-term matrix from train and test sets
x_train_dtm = vect.transform(x_train)
x_test_dtm = vect.transform(x_test)
```

5.1 Tunning CountVectorizer

CountVectorizer has a few parameters you should know.

1. **stop_words**: Since CountVectorizer just counts the occurrences of each word in its vocabulary, extremely common words like 'the', 'and', etc. will become very important features while they add little meaning to the text. Your model can often be improved if you don't take those words into account. Stop words are just a list of words you don't want to use as features. You can set the parameter stop_words='english' to use a built-in list. Alternatively you can set stop_words equal to some custom list. This parameter defaults to None.
2. **ngram_range**: An n-gram is just a string of n words in a row. E.g. the sentence 'I am Groot' contains the 2-grams 'I am' and 'am Groot'. The sentence is itself a 3-gram. Set the parameter ngram_range=(a,b) where a is the minimum and b is the maximum size of ngrams you want to include in your features. The default ngram_range is (1,1). In a recent project where I modeled job postings online, I found that including 2-grams as features boosted my model's predictive power significantly. This makes intuitive sense; many job titles such as 'data scientist', 'data engineer', and 'data analyst' are 2 words long.
3. **min_df, max_df**: These are the minimum and maximum document frequencies words/n-grams must have to be used as features. If either of these parameters are set to integers, they will be used as bounds on the number of documents each feature must be in to be considered as a feature. If either is set to a float, that number will be interpreted as a frequency rather than a numerical limit. min_df defaults to 1 (int) and max_df defaults to 1.0 (float).

4. **max_features**: This parameter is pretty self-explanatory. The CountVectorizer will choose the words/features that occur most frequently to be in its' vocabulary and drop everything else.

You would set these parameters when initializing your CountVectorizer object as shown below.

```
vect_tunned = CountVectorizer(stop_words='english', ngram_range=(1,2), min_df=0.1, max_df=0.7, max_features=100)
```

5.2 TF-IDF

In information retrieval, tf–idf, **TF-IDF**, or TFIDF, **short for term frequency–inverse document frequency**, is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus It is often used as a weighting factor in searches of information retrieval, text mining, and user modeling. The tf–idf value increases proportionally to the number of times a word appears in the document and is offset by the number of documents in the corpus that contain the word, which helps to adjust for the fact that some words appear more frequently in general.

tf–idf is one of the most popular term-weighting schemes today. A survey conducted in 2015 showed that 83% of text-based recommender systems in digital libraries use tf–idf.


```
from sklearn.feature_extraction.text import TfidfTransformer

tfidf_transformer = TfidfTransformer()

tfidf_transformer.fit(x_train_dtm)
x_train_tfidf = tfidf_transformer.transform(x_train_dtm)

x_train_tfidf
```

```
<16715x16913 sparse matrix of type '<class 'numpy.float64'>'
        with 155353 stored elements in Compressed Sparse Row format>
```

5.3 Word Embeddings: GloVe

Thanks to: https://www.kaggle.com/mariapushkareva/nlp-disaster-tweets-with-glove-and-lstm

```
texts = df['message_clean']
target = df['target_encoded']
```

We need to perform **tokenization** - the processing of segmenting text into sentences of words. The benefit of tokenization is that it gets the text into a format that is easier to convert to raw numbers, which can actually be used for processing.


```
# Calculate the length of our vocabulary
word_tokenizer = Tokenizer()
word_tokenizer.fit_on_texts(texts)

vocab_length = len(word_tokenizer.word_index) + 1
vocab_length
```

```
24646
```

Pad_sequences

https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/sequence/pad_sequences

```
tf.keras.preprocessing.sequence.pad_sequences(
    sequences, maxlen=None, dtype='int32', padding='pre',
    truncating='pre', value=0.0
)
```

This function transforms a list (of length num_samples) of sequences (lists of integers) into a 2D Numpy array of shape (num_samples, num_timesteps). num_timesteps is either the maxlen argument if provided, or the length of the longest sequence in the list.

```
>>> sequence = [[1], [2, 3], [4, 5, 6]]
>>> tf.keras.preprocessing.sequence.pad_sequences(sequence, padding='post')
array([[1, 0, 0],
       [2, 3, 0],
       [4, 5, 6]], dtype=int32)
```

```
import nltk
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
True
```

```
def embed(corpus):
    return word_tokenizer.texts_to_sequences(corpus)

longest_train = max(texts, key=lambda sentence: len(word_tokenize(sentence)))
length_long_sentence = len(word_tokenize(longest_train))

train_padded_sentences = pad_sequences(
    embed(texts),
    length_long_sentence,
    padding='post'
)

train_padded_sentences
```

```
array([[3501,  340,   34, ...,    0,    0,    0],
       [ 902, 2549, 1089, ...,    0,    0,    0],
       [ 531,   50,  434, ...,    0,    0,    0],
       ...,
       [   5, 1330,    8, ...,    0,    0,    0],
       [   7,  440,  284, ...,    0,    0,    0],
       [2247,  220, 4606, ...,    0,    0,    0]], dtype=int32)
```

GloVe

GloVe method is built on an important idea,

> You can derive semantic relationships between words from the co-occurrence matrix.

To obtain a vector representation for words we can use an unsupervised learning algorithm called **GloVe (Global Vectors for Word Representation)**, which focuses on words co-occurrences over the whole corpus. Its embeddings relate to the probabilities that two words appear together.

Word embeddings are basically a form of word representation that bridges the human understanding of language to that of a machine. They have learned representations of text in an n-dimensional space where words that have the same meaning have a similar representation. Meaning that two similar words are represented by almost similar vectors that are very closely placed in a vector space.

Thus when using word embeddings, all individual words are represented as real-valued vectors in a predefined vector space. Each word is mapped to one vector and the vector values are learned in a way that resembles a neural network.

```
import requests
import numpy as np

# Define the URL of the GloVe 100D embeddings file
glove_url = 'https://nlp.stanford.edu/data/glove.6B.100d.txt'

# Initialize an empty dictionary to store word embeddings
embeddings_dictionary = dict()
embedding_dim = 100

# Open the GloVe embeddings file using the requests library
response = requests.get(glove_url)

# Check if the request was successful
if response.status_code == 200:
    lines = response.text.split('\n')
    for line in lines:
        records = line.split()
```

```
        if len(records) > 1:
            word = records[0]
            vector_dimensions = np.asarray(records[1:], dtype='float32')
            embeddings_dictionary[word] = vector_dimensions
else:
    print("Failed to download GloVe embeddings.")

# Now, embeddings_dictionary contains the loaded word embeddings
```

```
Failed to download GloVe embeddings.
```

```
# Now we will load embedding vectors of those words that appear in the
# Glove dictionary. Others will be initialized to 0.

embedding_matrix = np.zeros((vocab_length, embedding_dim))

for word, index in word_tokenizer.word_index.items():
    embedding_vector = embeddings_dictionary.get(word)
    if embedding_vector is not None:
        embedding_matrix[index] = embedding_vector

embedding_matrix
```

```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

6. Modeling

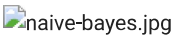
```
import plotly.figure_factory as ff

x_axes = ['Ham', 'Spam']
y_axes =  ['Spam', 'Ham']

def conf_matrix(z, x=x_axes, y=y_axes):

    z = np.flip(z, 0)

    # change each element of z to type string for annotations
    z_text = [[str(y) for y in x] for x in z]

    # set up figure
    fig = ff.create_annotated_heatmap(z, x=x, y=y, annotation_text=z_text, colorscale='Viridis')

    # add title
    fig.update_layout(title_text='<b>Confusion matrix</b>',
                      xaxis = dict(title='Predicted value'),
                      yaxis = dict(title='Real value')
                     )

    # add colorbar
    fig['data'][0]['showscale'] = True

    return fig


# Create a Multinomial Naive Bayes model
from sklearn.naive_bayes import MultinomialNB
nb = MultinomialNB()

# Train the model
nb.fit(x_train_dtm, y_train)
```

```
MultinomialNB
MultinomialNB()
```

6.1 Naive Bayes DTM

In statistics, naive Bayes classifiers are a family of simple "probabilistic classifiers" based on applying Bayes' theorem with strong (naïve) independence assumptions between the features. They are among the simplest Bayesian network models, but coupled with kernel density estimation, they can achieve higher accuracy levels.

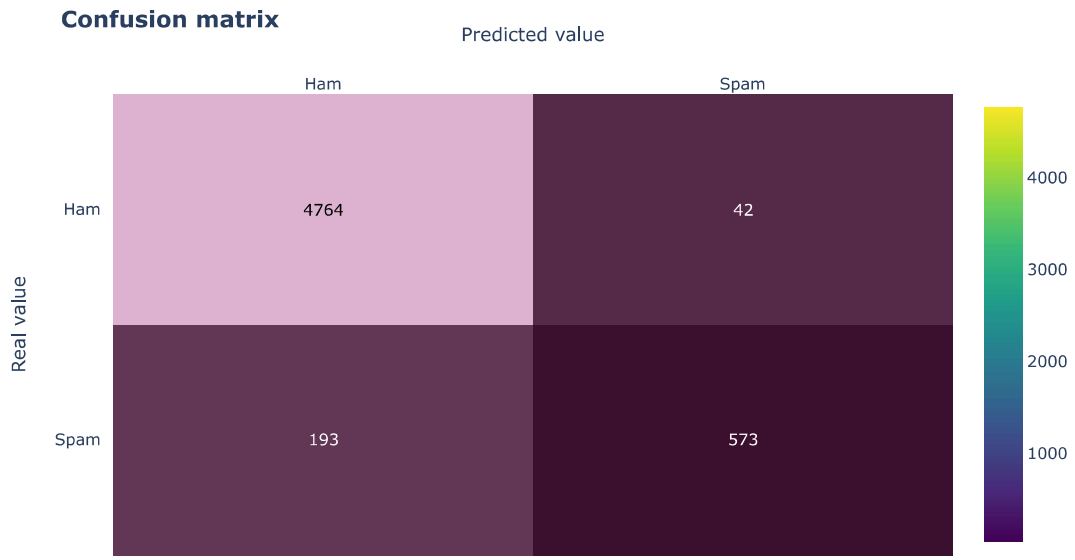
Naïve Bayes classifiers are highly scalable, requiring a number of parameters linear in the number of variables (features/predictors) in a learning problem. Maximum-likelihood training can be done by evaluating a closed-form expression, which takes linear time, rather than by expensive iterative approximation as used for many other types of classifiers.




```
# Make class anf probability predictions
y_pred_class = nb.predict(x_test_dtm)
y_pred_prob = nb.predict_proba(x_test_dtm)[:, 1]
```

```
# calculate accuracy of class predictions
from sklearn import metrics
print(metrics.accuracy_score(y_test, y_pred_class))

conf_matrix(metrics.confusion_matrix(y_test, y_pred_class))
```

```
0.9578248384781048
```




```
# Calculate AUC
metrics.roc_auc_score(y_test, y_pred_prob)
```

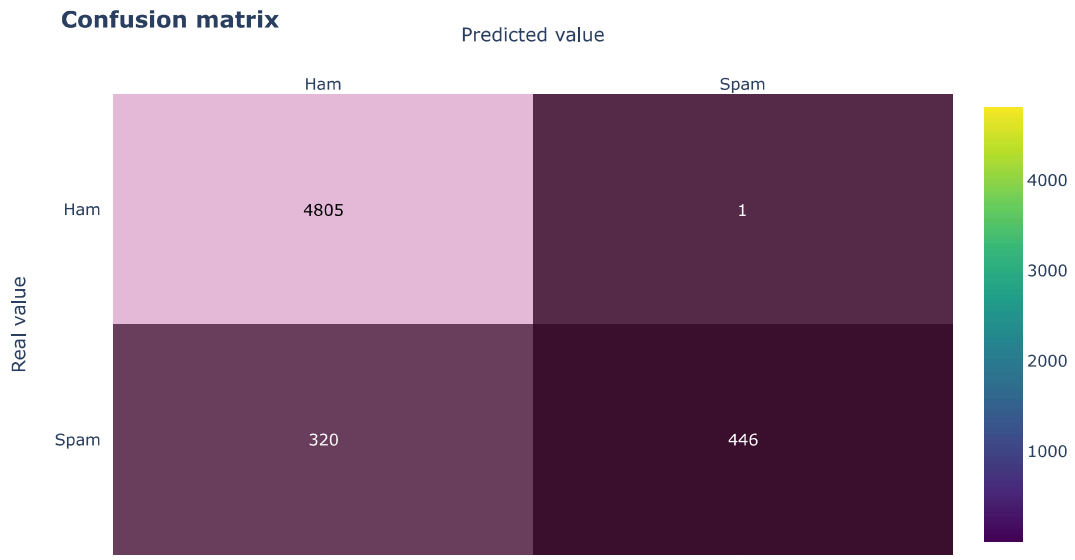
```
0.9463935691786485
```

6.2 Naive Bayes

```
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.pipeline import Pipeline

pipe = Pipeline([('bow', CountVectorizer()),
                 ('tfid', TfidfTransformer()),
                 ('model', MultinomialNB())])
```

```
# Fit the pipeline with the data
pipe.fit(x_train, y_train)

y_pred_class = pipe.predict(x_test)

print(metrics.accuracy_score(y_test, y_pred_class))

conf_matrix(metrics.confusion_matrix(y_test, y_pred_class))
```

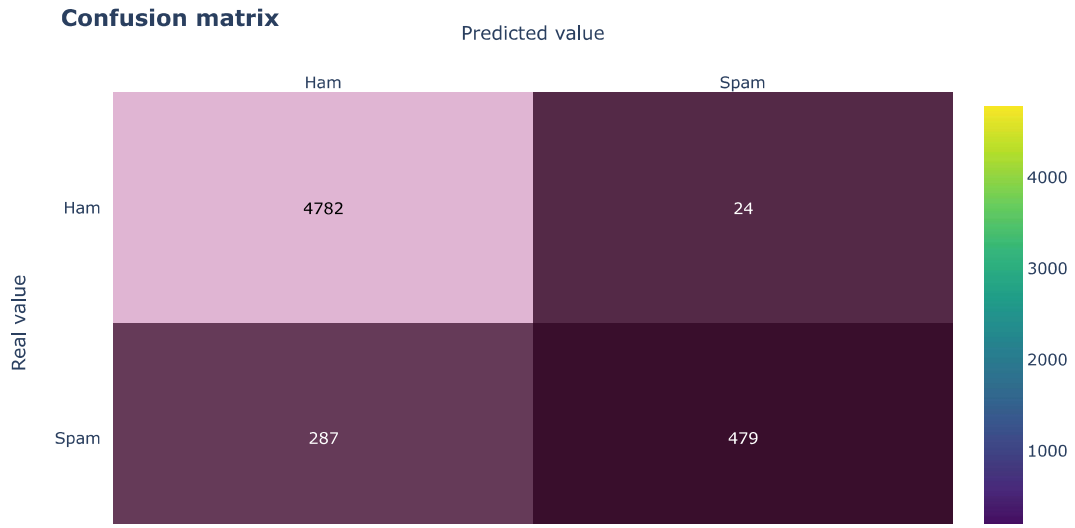
```
0.9423905240488155
```



6.3 XGBoost

```
import xgboost as xgb

pipe = Pipeline([
    ('bow', CountVectorizer()),
    ('tfid', TfidfTransformer()),
    ('model', xgb.XGBClassifier(
        learning_rate=0.1,
        max_depth=7,
        n_estimators=80,
        use_label_encoder=False,
        eval_metric='auc',
        # colsample_bytree=0.8,
        # subsample=0.7,
        # min_child_weight=5,
    ))
])


# Fit the pipeline with the data
pipe.fit(x_train, y_train)

y_pred_class = pipe.predict(x_test)
y_pred_train = pipe.predict(x_train)

print('Train: {}'.format(metrics.accuracy_score(y_train, y_pred_train)))
print('Test: {}'.format(metrics.accuracy_score(y_test, y_pred_class)))

conf_matrix(metrics.confusion_matrix(y_test, y_pred_class))
```

```
Train: 0.9518994914747233
Test: 0.9441852117731515
```



7. LSTM

```
# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(
    train_padded_sentences,
    target,
    test_size=0.25
)


# Model from https://www.kaggle.com/mariapushkareva/nlp-disaster-tweets-with-glove-and-lstm/data

def glove_lstm():
    model = Sequential()

    model.add(Embedding(
        input_dim=embedding_matrix.shape[0],
        output_dim=embedding_matrix.shape[1],
        weights = [embedding_matrix],
        input_length=length_long_sentence
    ))

    model.add(Bidirectional(LSTM(
        length_long_sentence,
        return_sequences = True,
        recurrent_dropout=0.2
    )))

    model.add(GlobalMaxPool1D())
    model.add(BatchNormalization())
    model.add(Dropout(0.5))
    model.add(Dense(length_long_sentence, activation = "relu"))
    model.add(Dropout(0.5))
    model.add(Dense(length_long_sentence, activation = "relu"))
    model.add(Dropout(0.5))
    model.add(Dense(1, activation = 'sigmoid'))
    model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['accuracy'])

    return model

model = glove_lstm()
model.summary()
```

```
WARNING:tensorflow:Layer lstm will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallba
WARNING:tensorflow:Layer lstm will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallba
WARNING:tensorflow:Layer lstm will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallba
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 510, 100)          2464600
```

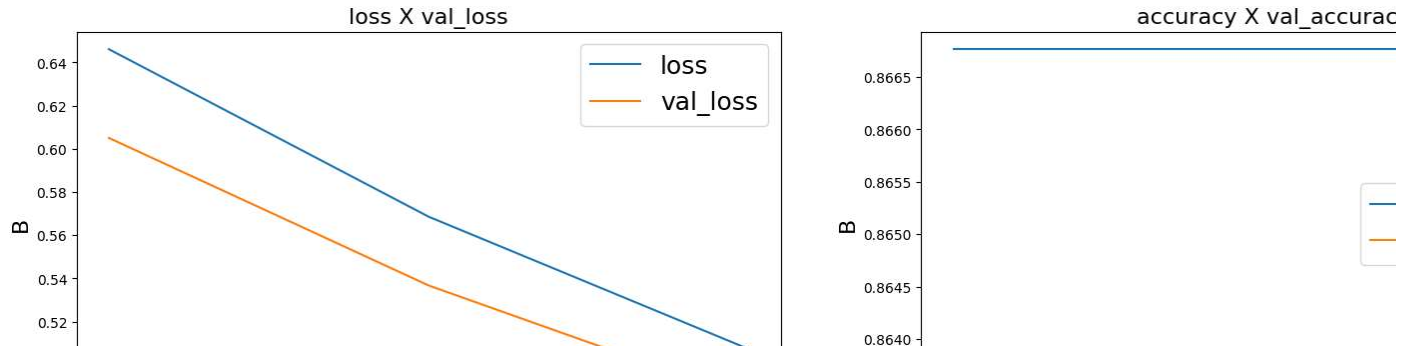
```
 bidirectional (Bidirection  (None, 510, 1020)         2492880
 al)

 global_max_pooling1d (Glob  (None, 1020)              0
 alMaxPooling1D)

 batch_normalization (Batch  (None, 1020)              4080
 Normalization)

 dropout (Dropout)           (None, 1020)              0

 dense (Dense)               (None, 510)               520710

 dropout_1 (Dropout)         (None, 510)               0

 dense_1 (Dense)             (None, 510)               260610

 dropout_2 (Dropout)         (None, 510)               0

 dense_2 (Dense)             (None, 1)                 511

=================================================================
Total params: 5743391 (21.91 MB)
Trainable params: 5741351 (21.90 MB)
Non-trainable params: 2040 (7.97 KB)
_________________________________________________________________
```

```
# Load the model and train!!

model = glove_lstm()

checkpoint = ModelCheckpoint(
    'model.h5',
    monitor = 'val_loss',
    verbose = 1,
    save_best_only = True
)
reduce_lr = ReduceLROnPlateau(
    monitor = 'val_loss',
    factor = 0.2,
    verbose = 1,
    patience = 5,
    min_lr = 0.001
)
history = model.fit(
    X_train,
    y_train,
    epochs = 3,
    batch_size = 64,
    validation_data = (X_test, y_test),
    verbose = 1,
    callbacks = [reduce_lr, checkpoint]
)
```

```
WARNING:tensorflow:Layer lstm_1 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fall
WARNING:tensorflow:Layer lstm_1 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fall
WARNING:tensorflow:Layer lstm_1 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fall
Epoch 1/3
262/262 [==============================] - ETA: 0s - loss: 0.6461 - accuracy: 0.8668
Epoch 1: val_loss improved from inf to 0.60504, saving model to model.h5
262/262 [==============================] - 969s 4s/step - loss: 0.6461 - accuracy: 0.8668 - val_loss: 0.6050 - val_accuracy: 0.8634 - l
Epoch 2/3
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3000: UserWarning:

You are saving your model as an HDF5 file via `model.save()`. This file format is considered legacy. We recommend using instead the nat

262/262 [==============================] - ETA: 0s - loss: 0.5686 - accuracy: 0.8668
Epoch 2: val_loss improved from 0.60504 to 0.53675, saving model to model.h5
262/262 [==============================] - 950s 4s/step - loss: 0.5686 - accuracy: 0.8668 - val_loss: 0.5367 - val_accuracy: 0.8634 - l
Epoch 3/3
262/262 [==============================] - ETA: 0s - loss: 0.5074 - accuracy: 0.8668
Epoch 3: val_loss improved from 0.53675 to 0.48413, saving model to model.h5
262/262 [==============================] - 943s 4s/step - loss: 0.5074 - accuracy: 0.8668 - val_loss: 0.4841 - val_accuracy: 0.8634 - l
```

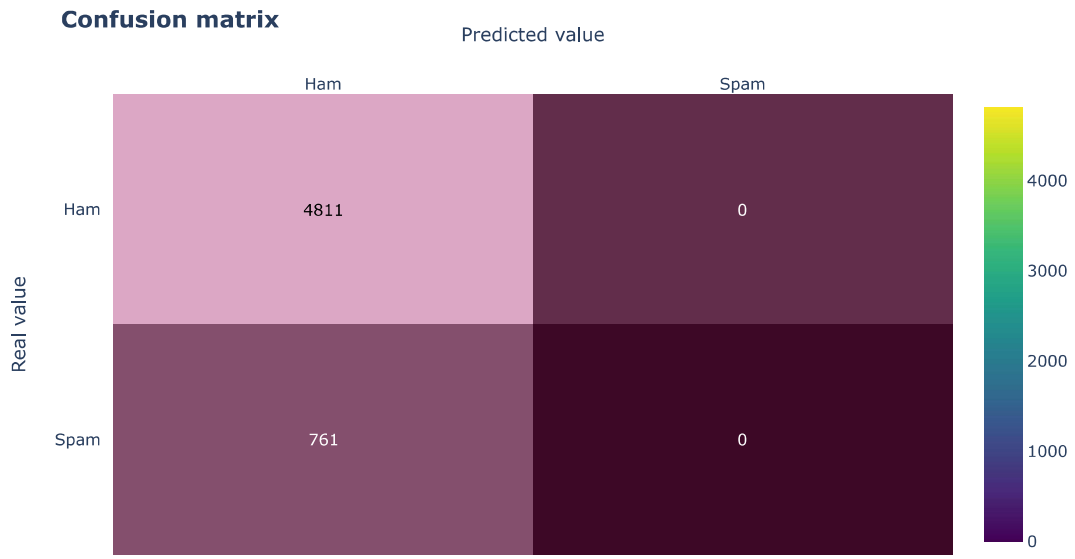
Lets see the results

```
def plot_learning_curves(history, arr):
    fig, ax = plt.subplots(1, 2, figsize=(20, 5))
    for idx in range(2):
        ax[idx].plot(history.history[arr[idx][0]])
        ax[idx].plot(history.history[arr[idx][1]])
        ax[idx].legend([arr[idx][0], arr[idx][1]],fontsize=18)
        ax[idx].set_xlabel('A ',fontsize=16)
        ax[idx].set_ylabel('B',fontsize=16)
        ax[idx].set_title(arr[idx][0] + ' X ' + arr[idx][1],fontsize=16)


plot_learning_curves(history, [['loss', 'val_loss'],['accuracy', 'val_accuracy']])
```




```
y_preds = (model.predict(X_test) > 0.5).astype("int32")
conf_matrix(metrics.confusion_matrix(y_test, y_preds))
```

```
175/175 [==============================] - 46s 261ms/step
```



8. BERT

BERT (Bidirectional Encoder Representations from Transformers) is a recent paper published by researchers at Google AI Language. It has caused a stir in the Machine Learning community by presenting state-of-the-art results in a wide variety of NLP tasks, including Question Answering (SQuAD v1.1), Natural Language Inference (MNLI), and others.

BERT's key technical innovation is applying the bidirectional training of Transformer, a popular attention model, to language modelling. This is in contrast to previous efforts which looked at a text sequence either from left to right or combined left-to-right and right-to-left training. The paper's results show that a language model which is bidirectionally trained can have a deeper sense of language context and flow than single-

direction language models. In the paper, the researchers detail a novel technique named Masked LM (MLM) which allows bidirectional training in models in which it was previously impossible.

Ref: https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270

```
# install transformers
!pip install transformers
```

```
Collecting transformers
  Downloading transformers-4.34.0-py3-none-any.whl (7.7 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7.7/7.7 MB 17.2 MB/s eta 0:00:00
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from transformers) (3.12.4)
Collecting huggingface-hub<1.0,>=0.16.4 (from transformers)
  Downloading huggingface_hub-0.17.3-py3-none-any.whl (295 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 295.0/295.0 kB 31.9 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-packages (from transformers) (1.23.5)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from transformers) (23.2)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-packages (from transformers) (6.0.1)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.10/dist-packages (from transformers) (2023.6.3)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from transformers) (2.31.0)
Collecting tokenizers<0.15,>=0.14 (from transformers)
  Downloading tokenizers-0.14.1-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (3.8 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.8/3.8 MB 40.2 MB/s eta 0:00:00
Collecting safetensors>=0.3.1 (from transformers)
  Downloading safetensors-0.4.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1.3 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.3/1.3 MB 47.0 MB/s eta 0:00:00
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.10/dist-packages (from transformers) (4.66.1)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.16.4->transformers) (202
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.16.4
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (3.3.0
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (2.0.6)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (2023.7.22)
Installing collected packages: safetensors, huggingface-hub, tokenizers, transformers
Successfully installed huggingface-hub-0.17.3 safetensors-0.4.0 tokenizers-0.14.1 transformers-4.34.0
```

```
import tensorflow as tf
from tensorflow.keras.layers import Dense, Input
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.models import Model
from tensorflow.keras.callbacks import ModelCheckpoint

import transformers
from tqdm.notebook import tqdm
from tokenizers import BertWordPieceTokenizer


try:
    tpu = tf.distribute.cluster_resolver.TPUClusterResolver()
    tf.config.experimental_connect_to_cluster(tpu)
    tf.tpu.experimental.initialize_tpu_system(tpu)
    strategy = tf.distribute.experimental.TPUStrategy(tpu)

except:
    strategy = tf.distribute.get_strategy()

print('Number of replicas in sync: ', strategy.num_replicas_in_sync)
```

```
Number of replicas in sync:  1
```

```
from transformers import BertTokenizer
tokenizer = BertTokenizer.from_pretrained('bert-large-uncased')

def bert_encode(data, maximum_length) :
    input_ids = []
    attention_masks = []

    for text in data:
        encoded = tokenizer.encode_plus(
            text,
            add_special_tokens=True,
            max_length=maximum_length,
            pad_to_max_length=True,

            return_attention_mask=True,
        )
```

```
        input_ids.append(encoded['input_ids'])
        attention_masks.append(encoded['attention_mask'])

    return np.array(input_ids),np.array(attention_masks)
```

```
Downloading (…)okenizer_config.json: 100%  28.0/28.0 [00:00<00:00, 1.08kB/s]
Downloading (…)solve/main/vocab.txt: 100%  232k/232k [00:00<00:00, 3.43MB/s]
Downloading (…)/main/tokenizer.json: 100%  466k/466k [00:00<00:00, 5.35MB/s]
Downloading (…)lve/main/config.json: 100%  571/571 [00:00<00:00, 33.9kB/s]
```

```
texts = df['message_clean']
target = df['target_encoded']

train_input_ids, train_attention_masks = bert_encode(texts,60)
```

```
Truncation was not explicitly activated but `max_length` is provided a specific value, please use `truncation=True` to explicitly trunc
/usr/local/lib/python3.10/dist-packages/transformers/tokenization_utils_base.py:2622: FutureWarning:

The `pad_to_max_length` argument is deprecated and will be removed in a future version, use `padding=True` or `padding='longest'` to pa
```

```
import tensorflow as tf
from tensorflow.keras.optimizers import Adam

def create_model(bert_model):

    input_ids = tf.keras.Input(shape=(60,),dtype='int32')
    attention_masks = tf.keras.Input(shape=(60,),dtype='int32')

    output = bert_model([input_ids,attention_masks])
    output = output[1]
    output = tf.keras.layers.Dense(32,activation='relu')(output)
    output = tf.keras.layers.Dropout(0.2)(output)
    output = tf.keras.layers.Dense(1,activation='sigmoid')(output)

    model = tf.keras.models.Model(inputs = [input_ids,attention_masks],outputs = output)
    model.compile(Adam(lr=1e-5), loss='binary_crossentropy', metrics=['accuracy'])
    return model


from transformers import TFBertModel
bert_model = TFBertModel.from_pretrained('bert-base-uncased')
```

```
Downloading (…)lve/main/config.json: 100%  570/570 [00:00<00:00, 9.49kB/s]
Downloading model.safetensors: 100%  440M/440M [00:05<00:00, 103MB/s]
Some weights of the PyTorch model were not used when initializing the TF 2.0 model TFBertModel: ['cls.predictions.transform.LayerNorm.we
- This IS expected if you are initializing TFBertModel from a PyTorch model trained on another task or with another architecture (e.g. i
- This IS NOT expected if you are initializing TFBertModel from a PyTorch model that you expect to be exactly identical (e.g. initializi
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
```

```
model = create_model(bert_model)
model.summary()
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimizers.le
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #   Connected to
==================================================================================================
 input_1 (InputLayer)        [(None, 60)]              0         []

 input_2 (InputLayer)        [(None, 60)]              0         []

 tf_bert_model (TFBertModel  TFBaseModelOutputWithPooli  109482240  ['input_1[0][0]',
 )                           ngAndCrossAttentions(last_  0          'input_2[0][0]']
                             hidden_state=(None, 60, 76
                             8),
                              pooler_output=(None, 768)
                             , past_key_values=None, hi
                             dden_states=None, attentio
                             ns=None, cross_attentions=
```

```
                                None)

 dense_6 (Dense)             (None, 32)                24608     ['tf_bert_model[0][1]']

 dropout_43 (Dropout)        (None, 32)                0         ['dense_6[0][0]']

 dense_7 (Dense)             (None, 1)                 33        ['dropout_43[0][0]']

==================================================================================================
Total params: 109506881 (417.74 MB)
Trainable params: 109506881 (417.74 MB)
Non-trainable params: 0 (0.00 Byte)
__________________________________________________________________________________________________
```

```
history = model.fit(
    [train_input_ids, train_attention_masks],
    target,
    validation_split=0.2,
    epochs=3,
    batch_size=128  # Adjust the batch size as needed
)
```

```
Epoch 1/3
140/140 [==============================] - 234s 1s/step - loss: 0.4344 - accuracy: 0.8553 - val_loss: 0.4075 - val_accuracy: 0.8701
Epoch 2/3
140/140 [==============================] - 211s 2s/step - loss: 0.4069 - accuracy: 0.8649 - val_loss: 0.3863 - val_accuracy: 0.8701
Epoch 3/3
140/140 [==============================] - 211s 2s/step - loss: 0.4082 - accuracy: 0.8649 - val_loss: 0.3913 - val_accuracy: 0.8701
```

```
plot_learning_curves(history, [['loss', 'val_loss'],['accuracy', 'val_accuracy']])
```