```
import numpy as np # linear algebra
import pandas as pd
```

```
# dowload dataset
test = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/dataset/kaggle dataset/
train= pd.read_csv("/content/drive/MyDrive/Colab Notebooks/dataset/kaggle dataset/
submit=pd.read_csv("/content/drive/MyDrive/Colab Notebooks/dataset/kaggle dataset/
```

```
import matplotlib.pyplot as plt
import seaborn as sns

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

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

```
# load the data
data = train
data.head()
```

```
print("Shape of dataset ", data.shape)
print("Columns ", data.columns)
```

```
Shape of dataset  (20800, 5)
Columns  Index(['id', 'title', 'author', 'text', 'label'], dtype='object')
```

```
# Let's do some statistics of the text columns
txt_len = data.text.str.split().str.len()
txt_len.describe()
```

```
count    20761.000000
mean       760.308126
std        869.525988
min          0.000000
25%        269.000000
```

```
50%        556.000000
75%       1052.000000
max      24234.000000
Name: text, dtype: float64
```

```
# Let's do some statistics of the title columns
title_len = data.title.str.split().str.len()
title_len.describe()
```

```
count    20242.000000
mean        12.420709
std          4.098735
min          1.000000
25%         10.000000
50%         13.000000
75%         15.000000
max         72.000000
Name: title, dtype: float64
```

```
# Class Distribution
# 1: Unreliable
# 2: Reliable
sns.countplot(x='label', data= data)
```

```
print(data.label.value_counts())
print()
print(round(data.label.value_counts(normalize=True),2)*100)
```

```
1    10413
0    10387
Name: label, dtype: int64

1    50.0
0    50.0
Name: label, dtype: float64
```

```
data.isnull().sum()
```

```
id           0
title      558
author    1957
text        39
label        0
dtype: int64
```

```
column_n = ['id', 'title', 'author', 'text', 'label']
remove_c = ['id','author']
categorical_features = []
target_col = ['label']
text_f = ['title', 'text']
```

```
# cleaning
import nltk
from nltk.corpus import stopwords
import re
from nltk.stem.porter import PorterStemmer
from collections import Counter

ps = PorterStemmer()
wnl = nltk.stem.WordNetLemmatizer()

stop_words = stopwords.words('english')
stopwords_dict = Counter(stop_words)

# remove unused columns
def remove_unused_c(df, column_n=remove_c):
    df = df.drop(column_n, axis=1)
    return df

# impute null values with none
def null_process(feature_df):
    for col in text_f:
        feature_df.loc[feature_df[col].isnull(),col] = "None"
    return feature_df

# clean_data
def clean_dataset(df):
    # remove unused column
    df = remove_unused_c(df)
    #impute null value
    df = null_process(df)

    return df

# Cleaning text from unused characters
def clean_text(text):
    text = str(text).replace(r'http[\w:/\.]+', ' ')  # removing urls
    text = str(text).replace(r'[^\.\w\s]', ' ')  # remove everything but characters
    text = str(text).replace('[^a-zA-Z]', ' ')
    text = str(text).replace(r'\s\s+', ' ')
    text = text.lower().strip()
    #text = ' '.join(text)
```

```
    return text

## Nltk Preprocessing include:
# Stop words, Stemming and Lemmetization
# For our project we use only Stop word removal
def nltk_preprocess(text):
    text = clean_text(text)
    wordlist = re.sub(r'[^\w\s]', '', text).split()
    text = ' '.join([wnl.lemmatize(word) for word in wordlist if word not in stopwo
    return  text
```

```
df = clean_dataset(data)
df['text'] = df.text.apply(nltk_preprocess)
df['title'] = df.title.apply(nltk_preprocess)
```

```
df.head()
```

```
from wordcloud import WordCloud, STOPWORDS

# initialize the word cloud
wordcloud = WordCloud(background_color='black', width=800, height=600)
# generate the word cloud
text_cloud = wordcloud.generate(" ".join(df['text']))
# plotting the word cloud
plt.figure(figsize=(20,30))
plt.imshow(text_cloud)
plt.axis('off')
plt.show()
```

```
# reliable news (0)
reliable_news = " ".join(df[df['label']==0]['text'])
wc = wordcloud.generate(reliable_news)
plt.figure(figsize=(20,30))
plt.imshow(wc)
plt.axis('off')
plt.show()
```

```
# unreliable news (1)
unreliable_news  = ' '.join(df[df['label']==1]['text'])
wc= wordcloud.generate(unreliable_news)
plt.figure(figsize=(20,30))
plt.imshow(wc)
plt.axis('off')
plt.show()
```

```
# Bigram

def plot_top_ngrams(corpus, title, ylabel, xlabel="Number of Occurenes", n =2):
    true_b = (pd.Series(nltk.ngrams(corpus.split(), n)).value_counts())[:20]
    true_b.sort_values().plot.barh(color='blue', width=.9, figsize=(12,8))
    plt.title(title)
    plt.ylabel(ylabel)
    plt.xlabel(xlabel)
    plt.show()


plot_top_ngrams(reliable_news, "Top 20 Frequently Occuring True News Bigrams", "Bi
```

```
plot_top_ngrams(unreliable_news, 'Top 20 Frequently Occuring Fake news Bigrams', "I
```

```
# Trigram
plot_top_ngrams(reliable_news, "Top 20 Frequently Occuring True News Bigrams", "Bi
```

```
plot_top_ngrams(unreliable_news, "Top 20 Frequently Occuring True News Bigrams", "
```

```
!pip install transformers
```

```
Collecting transformers
  Downloading transformers-4.19.1-py3-none-any.whl (4.2 MB)
     |████████████████████████████████| 4.2 MB 4.3 MB/s
Requirement already satisfied: filelock in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: importlib-metadata in /usr/local/lib/python3.7
Collecting pyyaml>=5.1
  Downloading PyYAML-6.0-cp37-cp37m-manylinux_2_5_x86_64.manylinux1_x86_64.ma
     |████████████████████████████████| 596 kB 45.2 MB/s
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-pack
Collecting huggingface-hub<1.0,>=0.1.0
  Downloading huggingface_hub-0.6.0-py3-none-any.whl (84 kB)
     |████████████████████████████████| 84 kB 3.0 MB/s
Collecting tokenizers!=0.11.3,<0.13,>=0.11.1
  Downloading tokenizers-0.12.1-cp37-cp37m-manylinux_2_12_x86_64.manylinux201
     |████████████████████████████████| 6.6 MB 38.2 MB/s
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.7/
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.7/di
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/p
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /usr/local/lib/pyt
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /us
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/
Installing collected packages: pyyaml, tokenizers, huggingface-hub, transform
  Attempting uninstall: pyyaml
    Found existing installation: PyYAML 3.13
    Uninstalling PyYAML-3.13:
      Successfully uninstalled PyYAML-3.13
Successfully installed huggingface-hub-0.6.0 pyyaml-6.0 tokenizers-0.12.1 tra
```

```python
import torch
from transformers.file_utils import is_tf_available, is_torch_available, is_torch_
from transformers import BertTokenizerFast, BertForSequenceClassification
from transformers import Trainer, TrainingArguments
from sklearn.model_selection import train_test_split
import random
```

```python
def set_seed(seed: int):
    """
    Helper function for reproducible behavior to set the seed in ``random``, ``num
    installed).

    Args:
        seed (:obj:`int`): The seed to set.
    """
    random.seed(seed)
    np.random.seed(seed)
    if is_torch_available():
        torch.manual_seed(seed)
        torch.cuda.manual_seed_all(seed)
        # ^^ safe to call this function even if cuda is not available
    if is_tf_available():
```

```
        import tensorflow as tf

        tf.random.set_seed(seed)

set_seed(123)
```

```
model_name = "bert-base-uncased"
max_length= 512
```

```
tokenizer = BertTokenizerFast.from_pretrained(model_name, do_lower_case=True)
```

```
data.head()
```

```
## Data Preparation
data = data[data['text'].notna()]
data = data[data['title'].notna()]
data = data[data['author'].notna()]
```

```
def prepare_data(df, test_size=0.2, include_title=True, include_author=True):
    texts = []
    labels = []

    for i in range(len(df)):
        text = df['text'].iloc[i]
        label = df['label'].iloc[i]

        if include_title:
            text = df['title'].iloc[i] + " - " + text
        if include_author:
            text = df['author'].iloc[i] + " - " + text

        if text and label in [0,1]:
            texts.append(text)
```

```
            labels.append(label)

    return train_test_split(texts, labels, test_size=test_size)

train_texts, valid_texts, train_labels, valid_labels = prepare_data(data)
```

```
print(len(train_texts), len(train_labels))
print(len(valid_texts), len(valid_labels))
```

```
14628 14628
3657 3657
```

```
# tokenizing the dataset
train_encodings = tokenizer(train_texts, truncation=True, padding=True, max_length=
valid_encodings = tokenizer(valid_texts, truncation=True, padding=True, max_length=
```

```
# converting the encoding into a PyTorch datset
class NewsGroupsDataset(torch.utils.data.Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels

    def __getitem__(self, idx):
        item = {k: torch.tensor(v[idx]) for k, v in self.encodings.items()}
        item['labels'] = torch.tensor([self.labels[idx]])
        return item

    def __len__(self):
        return len(self.labels)

# convert tokenize data into torch dataset
train_dataset = NewsGroupsDataset(train_encodings, train_labels)
valid_dataset = NewsGroupsDataset(valid_encodings, valid_labels)
```

```
model = BertForSequenceClassification.from_pretrained(model_name, num_labels=2)
```

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_recall_fscore_support

def computer_metrics(pred):
    labels = pred.label_ids
    preds = pred.predictions.argmax(-1)
    precision,recall,f1,_ =precision_recall_fscore_support(labels,preds,average='b
```

```
    acc = accuracy_score(labels, preds)

    return {
        'accuracy':acc,
        'f1':f1,
        'precision':precision,
        'recall':recall
        }
```

```
training_args = TrainingArguments(
    output_dir='./results',          # output directory
    num_train_epochs=1,              # total number of training epochs
    per_device_train_batch_size=10,  # batch size per device during training
    per_device_eval_batch_size=20,   # batch size for evaluation
    warmup_steps=100,                # number of warmup steps for learning rate sc
    logging_dir='./logs',            # directory for storing logs
    load_best_model_at_end=True,     # load the best model when finished training
    # but you can specify `metric_for_best_model` argument to change to accuracy o
    logging_steps=200,               # log & save weights each logging_steps
    save_steps=200,
    evaluation_strategy="steps",     # evaluate each `logging_steps`
)
```

```
using `logging_steps` to initialize `eval_steps` to 200
PyTorch: setting up devices
The default value for the training argument `--report_to` will change in v5 (
```

```
trainer = Trainer(
    model = model,
    args = training_args,
    train_dataset=train_dataset,
    eval_dataset=valid_dataset,
    compute_metrics=computer_metrics,
)
```

```
trainer.train()
```

```
/usr/local/lib/python3.7/dist-packages/transformers/optimization.py:309: Futu
  FutureWarning,
***** Running training *****
  Num examples = 14628
  Num Epochs = 1
  Instantaneous batch size per device = 10
  Total train batch size (w. parallel, distributed & accumulation) = 10
  Gradient Accumulation steps = 1
  Total optimization steps = 1463
```

[1463/1463 1:13:55, Epoch 1/1]

Step	Training Loss	Validation Loss	Accuracy	F1	Precision	Recall
200	0.010400	0.029362	0.996172	0.995628	0.997497	0.993766
400	0.039100	0.118688	0.973476	0.968840	0.999337	0.940150
600	0.067200	0.020631	0.996445	0.995944	0.996877	0.995012
800	0.021100	0.015175	0.996992	0.996581	0.993800	0.999377
1000	0.006400	0.014516	0.998086	0.997819	0.997508	0.998130
1200	0.029000	0.009180	0.998086	0.997819	0.997508	0.998130
1400	0.014100	0.005111	0.998906	0.998755	0.997512	1.000000

```
***** Running Evaluation *****
  Num examples = 3657
  Batch size = 20
Saving model checkpoint to ./results/checkpoint-200
Configuration saved in ./results/checkpoint-200/config.json
Model weights saved in ./results/checkpoint-200/pytorch_model.bin
***** Running Evaluation *****
  Num examples = 3657
  Batch size = 20
Saving model checkpoint to ./results/checkpoint-400
Configuration saved in ./results/checkpoint-400/config.json
Model weights saved in ./results/checkpoint-400/pytorch_model.bin
***** Running Evaluation *****
  Num examples = 3657
  Batch size = 20
Saving model checkpoint to ./results/checkpoint-600
Configuration saved in ./results/checkpoint-600/config.json
Model weights saved in ./results/checkpoint-600/pytorch_model.bin
***** Running Evaluation *****
  Num examples = 3657
  Batch size = 20
Saving model checkpoint to ./results/checkpoint-800
Configuration saved in ./results/checkpoint-800/config.json
Model weights saved in ./results/checkpoint-800/pytorch_model.bin
***** Running Evaluation *****
  Num examples = 3657
  Batch size = 20
Saving model checkpoint to ./results/checkpoint-1000
Configuration saved in ./results/checkpoint-1000/config.json
Model weights saved in ./results/checkpoint-1000/pytorch_model.bin
***** Running Evaluation *****
  Num examples = 3657
  Batch size = 20
Saving model checkpoint to ./results/checkpoint-1200
```

```
# evaluate the current model after training
```

```
# evaluate the current model after training
trainer.evaluate()
```

```
***** Running Evaluation *****
  Num examples = 3657
  Batch size = 20
[183/183 03:59]
{'epoch': 1.0,
 'eval_accuracy': 0.9989062072737216,
 'eval_f1': 0.9987546699875467,
 'eval_loss': 0.005110885016620159,
 'eval_precision': 0.9975124378109452,
 'eval_recall': 1.0,
 'eval_runtime': 241.0123,
 'eval_samples_per_second': 15.174,
 'eval_steps_per_second': 0.759}
```

```
# saving the fine tuned model & tokenizer
model_path = "fake-news-bert-base-uncased"
model.save_pretrained(model_path)
tokenizer.save_pretrained(model_path)
```

```
Configuration saved in fake-news-bert-base-uncased/config.json
Model weights saved in fake-news-bert-base-uncased/pytorch_model.bin
tokenizer config file saved in fake-news-bert-base-uncased/tokenizer_config.j
Special tokens file saved in fake-news-bert-base-uncased/special_tokens_map.j
('fake-news-bert-base-uncased/tokenizer_config.json',
 'fake-news-bert-base-uncased/special_tokens_map.json',
 'fake-news-bert-base-uncased/vocab.txt',
 'fake-news-bert-base-uncased/added_tokens.json',
 'fake-news-bert-base-uncased/tokenizer.json')
```

```
def get_prediction(text, convert_to_label=False):
    # prepare our text into tokenized sequence
    inputs = tokenizer(text, padding=True, truncation=True, max_length=max_length,
    # perform inference to our model
    outputs = model(**inputs)
    # get output probabilities by doing softmax
    probs = outputs[0].softmax(1)
    # executing argmax function to get the candidate label
    d = {
        0: "reliable",
        1: "fake"
    }
    if convert_to_label:
        return d[int(probs.argmax())]
    else:
        return int(probs.argmax())
```

```
real_news = """
Tim Tebow Will Attempt Another Comeback, This Time in Baseball - The New York Time
"""

get_prediction(real_news, convert_to_label=True)
```

```
# read the test set
test_df = test
# make a copy of the testing set
new_df = test_df.copy()
# add a new column that contains the author, title and article content
new_df["new_text"] = new_df["author"].astype(str) + " : " + new_df["title"].astype
# get the prediction of all the test set
new_df["label"] = new_df["new_text"].apply(get_prediction)
# make the submission file
final_df = new_df[["id", "label"]]
final_df.to_csv("submit_final.csv", index=False)
```

Executing (1m 57s) Cell > apply() > apply() > apply_standard() > get_prediction()