In [ ]:
```
# BERT with Keras

import pandas as pd

df_balanced = pd.read_csv("dataframe_edit.tsv", sep = '\t')

df.head()

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(df_balanced['text'],df_b
                                                    stratify=df_balanced['hy


!pip install tensorflow_text

import tensorflow as tf
import tensorflow_hub as hub
import tensorflow_text as text

bert_preprocess = hub.KerasLayer("https://tfhub.dev/tensorflow/bert_en_uncas
bert_encoder = hub.KerasLayer("https://tfhub.dev/tensorflow/bert_en_uncased_

# DistilBERT
#https://tfhub.dev/jeongukjae/distilbert_en_uncased_L-6_H-768_A-12/1
#

# Bert layers
text_input = tf.keras.layers.Input(shape=(), dtype=tf.string, name='text')
preprocessed_text = bert_preprocess(text_input)
outputs = bert_encoder(preprocessed_text)

# Neural network layers
l = tf.keras.layers.Dropout(0.1, name="dropout")(outputs['pooled_output'])
l = tf.keras.layers.Dense(1, activation='sigmoid', name="output")(l)

# Use inputs and outputs to construct a final model
model = tf.keras.Model(inputs=[text_input], outputs = [l])

METRICS = [
      tf.keras.metrics.BinaryAccuracy(name='accuracy'),
      tf.keras.metrics.Precision(name='precision'),
      tf.keras.metrics.Recall(name='recall')
]

model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=METRICS)


model.fit(X_train, y_train, epochs=5)
```

```
model.evaluate(X_test, y_test)

y_predicted = model.predict(X_test)
y_predicted = y_predicted.flatten()

import numpy as np

y_predicted = np.where(y_predicted > 0.5, 1, 0)
y_predicted



from sklearn.metrics import confusion_matrix, classification_report

cm = confusion_matrix(y_test, y_predicted)
cm

from matplotlib import pyplot as plt
import seaborn as sn
sn.heatmap(cm, annot=True, fmt='d')
plt.xlabel('Predicted')
plt.ylabel('Truth')

print(classification_report(y_test, y_predicted))
```

In [ ]:

```
## **GPT-3 Zero shot**

!pip install openai

import os
import openai

import pandas as pd
ds = pd.read_csv("dataframe_edit.tsv", sep = '\t')

ds_65 = ds.tail(65)

y_true = ds_65["hyperpartisan"].tolist()

y_true

ds_65["text"].head()

len(ds_20)

import os
import openai

OPENAI_API_KEY = "sk-8zJhEQJenl8Qsq6WReemfj4Z3"
openai.api_key = OPENAI_API_KEY

start_sequence = "\nA:"
```

```
restart_sequence = "\n\nQ: "

alist= []

for i in ds_65["text"]:
  response = openai.Completion.create(
    model="text-davinci-003",
    prompt="Text: "+ i+ """\nQ: Does the above text contain hyperpartisan el
    Answer Only Yes or No.\nA:""",
    temperature=0,
    max_tokens=100,
    top_p=1,
    frequency_penalty=0,
    presence_penalty=0,
    stop=["\n"]
  )
  alist.append(1 if response["choices"][0]["text"]==" Yes" else 0)

response["choices"][0]["text"]

type(alist[0])

# Classical ML

import sklearn.metrics as metrics

print(metrics.confusion_matrix(y_true, alist))

# Print the precision and recall, among other metrics
print(metrics.classification_report(y_true, alist, digits=3))

prompt="Text: "+ i+ """\nAdditional Information -  Hyperpartisan argument is
prejudiced, or unreasoning allegiance to one party, faction, cause, or persc
information, does the above text contain hyperpartisan elements to it. Answe
prompt="Text: "+ i+ "\nQ: Does the above text contain hyperpartisan elements



import random

random_list = [random.randint(0, 1) for _ in range(65)]

len(random_list)

import sklearn.metrics as metrics

print(metrics.confusion_matrix(y_true,random_list))

# Print the precision and recall, among other metrics
print(metrics.classification_report(y_true, random_list, digits=3))
```

In [ ]:
```
import pandas as pd
import numpy as np
```

```
import collections
import csv
import os
import re
from nltk.tokenize import word_tokenize
from sklearn.preprocessing import LabelEncoder
from collections import defaultdict
from nltk.corpus import wordnet as wn
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn import model_selection, naive_bayes, svm
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score, f1_score, precision_score, recal
from nltk import pos_tag
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer

Corpus = pd.read_csv("dataframe_edit.tsv", sep = "\t", names = ['text','labe
Corpus.head()


df.groupby("label").count()

import nltk
nltk.download('wordnet')

# change it to str
Corpus.text = Corpus.text.astype(str)
Corpus['text'] = Corpus['text'].str.lower()
Corpus = Corpus.dropna()

Corpus.head()

Corpus.info()

#tokenizing our hyperpartisan text column here
Corpus['text'] = Corpus['text'].apply(nltk.word_tokenize)

# Tagging to understand if the word is a noun, verb, adverb etc

tag_map = defaultdict(lambda : wn.NOUN)
tag_map['J'] = wn.ADJ
tag_map['V'] = wn.VERB
tag_map['R'] = wn.ADV

import nltk
nltk.download('averaged_perceptron_tagger')

for index,entry in enumerate(Corpus['text']):

    Final_words = []

    word_Lemmatized = WordNetLemmatizer()
    for word, tag in pos_tag(entry):
```

```
        # check for Stop words and consider only alphabets
        if word not in stopwords.words('english') and word.isalpha():
            word_Final = word_Lemmatized.lemmatize(word,tag_map[tag[0]])
            Final_words.append(word_Final)
    # The final processed set of words for each iteration will be stored in
    Corpus.loc[index,'text_final'] = str(Final_words)

Corpus.head()

#Train, test split
Train_X, Test_X, Train_Y, Test_Y = model_selection.train_test_split(Corpus['
                                                                    Corpus['

#Encoding our labels
Encoder = LabelEncoder()
Train_Y = Encoder.fit_transform(Train_Y)
Test_Y = Encoder.fit_transform(Test_Y)

Tfidf_vect = TfidfVectorizer()

Tfidf_vect.fit(Corpus['text_final'])

Train_X_Tfidf = Tfidf_vect.transform(Train_X)
Test_X_Tfidf = Tfidf_vect.transform(Test_X)

print(len(Tfidf_vect.vocabulary_))

# fit the NB classifier
Naive = naive_bayes.MultinomialNB()
naive_model = Naive.fit(Train_X_Tfidf,Train_Y)
predictions_NB = Naive.predict(Test_X_Tfidf)
print("Naive Bayes Accuracy Score -> ",accuracy_score(predictions_NB, Test_Y

f1_score(predictions_NB, Test_Y)

#SVM classifier
SVM = svm.SVC(C=2.0, kernel='poly',degree=2, gamma='scale')
SVM.fit(Train_X_Tfidf,Train_Y)
predictions_SVM = SVM.predict(Test_X_Tfidf)
print("SVM Accuracy Score -> ",accuracy_score(predictions_SVM, Test_Y)*100)

f1_score(predictions_SVM, Test_Y)
```

In [ ]:

In [ ]:

In [ ]: