In [1]:
```
!pip install -U bitsandbytes
!pip install --upgrade transformers
```

```
Requirement already satisfied: bitsandbytes in /opt/conda/lib/python3.10/s
ite-packages (0.43.3)
Requirement already satisfied: torch in /opt/conda/lib/python3.10/site-pac
kages (from bitsandbytes) (2.4.0)
Requirement already satisfied: numpy in /opt/conda/lib/python3.10/site-pac
kages (from bitsandbytes) (1.26.4)
Requirement already satisfied: filelock in /opt/conda/lib/python3.10/site-
packages (from torch->bitsandbytes) (3.15.1)
Requirement already satisfied: typing-extensions>=4.8.0 in /opt/conda/lib/
python3.10/site-packages (from torch->bitsandbytes) (4.12.2)
Requirement already satisfied: sympy in /opt/conda/lib/python3.10/site-pac
kages (from torch->bitsandbytes) (1.13.2)
Requirement already satisfied: networkx in /opt/conda/lib/python3.10/site-
packages (from torch->bitsandbytes) (3.3)
Requirement already satisfied: jinja2 in /opt/conda/lib/python3.10/site-pa
ckages (from torch->bitsandbytes) (3.1.4)
Requirement already satisfied: fsspec in /opt/conda/lib/python3.10/site-pa
ckages (from torch->bitsandbytes) (2024.6.1)
Requirement already satisfied: MarkupSafe>=2.0 in /opt/conda/lib/python3.1
0/site-packages (from jinja2->torch->bitsandbytes) (2.1.5)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /opt/conda/lib/python
3.10/site-packages (from sympy->torch->bitsandbytes) (1.3.0)
Requirement already satisfied: transformers in /opt/conda/lib/python3.10/s
ite-packages (4.44.2)
Requirement already satisfied: filelock in /opt/conda/lib/python3.10/site-
packages (from transformers) (3.15.1)
Requirement already satisfied: huggingface-hub<1.0,>=0.23.2 in /opt/conda/
lib/python3.10/site-packages (from transformers) (0.24.6)
Requirement already satisfied: numpy>=1.17 in /opt/conda/lib/python3.10/si
te-packages (from transformers) (1.26.4)
Requirement already satisfied: packaging>=20.0 in /opt/conda/lib/python3.1
0/site-packages (from transformers) (21.3)
Requirement already satisfied: pyyaml>=5.1 in /opt/conda/lib/python3.10/si
te-packages (from transformers) (6.0.2)
Requirement already satisfied: regex!=2019.12.17 in /opt/conda/lib/python
3.10/site-packages (from transformers) (2024.5.15)
Requirement already satisfied: requests in /opt/conda/lib/python3.10/site-
packages (from transformers) (2.32.3)
Requirement already satisfied: safetensors>=0.4.1 in /opt/conda/lib/python
3.10/site-packages (from transformers) (0.4.4)
Requirement already satisfied: tokenizers<0.20,>=0.19 in /opt/conda/lib/py
thon3.10/site-packages (from transformers) (0.19.1)
Requirement already satisfied: tqdm>=4.27 in /opt/conda/lib/python3.10/sit
e-packages (from transformers) (4.66.4)
Requirement already satisfied: fsspec>=2023.5.0 in /opt/conda/lib/python3.
10/site-packages (from huggingface-hub<1.0,>=0.23.2->transformers) (2024.
6.1)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /opt/conda/li
b/python3.10/site-packages (from huggingface-hub<1.0,>=0.23.2->transformer
s) (4.12.2)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /opt/conda/lib/
python3.10/site-packages (from packaging>=20.0->transformers) (3.1.2)
Requirement already satisfied: charset-normalizer<4,>=2 in /opt/conda/lib/
python3.10/site-packages (from requests->transformers) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /opt/conda/lib/python3.10/s
ite-packages (from requests->transformers) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in /opt/conda/lib/python
3.10/site-packages (from requests->transformers) (1.26.18)
Requirement already satisfied: certifi>=2017.4.17 in /opt/conda/lib/python
3.10/site-packages (from requests->transformers) (2024.7.4)
```

In [2]:
```
import numpy as np
import pandas as pd

import time
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM, BitsAndByte
import shap
import numpy as np
import torch.nn.functional as F
import pandas as pd
from time import time
import warnings

warnings.filterwarnings("ignore", message="The attention mask and the pad
```

In [3]:
```
class Llama:

    def __init__(self, model, tokenizer):

        self.model = model
        self.tokenizer = tokenizer


    def single_predict(self,token):
#         token = torch.tensor(train_v1[token_input_column][0:1].values)
#         mask = torch.tensor(train_v1[token_att_column][0:1].values)


        with torch.no_grad():
            output_ids = model.generate(
                token,
                max_length=token.shape[1] + 3,
                num_return_sequences=1,
                do_sample=False,
                return_dict_in_generate=True, output_scores=True
            )

        A_logit =F.softmax(output_ids.scores[0])[0][32]
        B_logit =F.softmax(output_ids.scores[0])[0][33]

        max_index = np.argmax(output_ids.scores[0])
#         return [A_logit,B_logit,max_index]
        return A_logit

    def predict(self, token_arr):

        output_list = []

        for i in range(token_arr.shape[0]):
            output_list.append(self.single_predict(token_arr[i:i+1,:]))

        return np.array(output_list)
```

In [4]:
```
def truncate_text(text, max_tokens=256):
    tokens = text.split()

    truncated_tokens = tokens[:max_tokens]
```

```
    truncated_text = ' '.join(truncated_tokens)

    return truncated_text

def prompt_generation(tweet):
    trucated_tweet = truncate_text(tweet)

    prompt = f'''Please select the option (A or B) that most closely desc
    (A) True
    (B) False
    Choice: ('''

    #     print(prompt)

    return prompt

def generate_token(input_text):
    # Tokenize the input text
    input_ids = tokenizer.encode(input_text, return_tensors="pt")

    # Generate the output
    with torch.no_grad():
        output_ids = model.generate(
            input_ids,
            max_length=80,
            num_return_sequences=1,
            do_sample=False,
            return_dict_in_generate=True, output_scores=True
        )

    index = np.argmax(F.softmax(output_ids.scores[0]))
    token_probability = F.softmax(output_ids.scores[0])[0][index]
    token = tokenizer.convert_ids_to_tokens([index])[0]
    return [index, token, token_probability, output_ids]

def predict_label(tweet):
    prompt = prompt_generation(tweet)
    print(prompt)
    lable = generate_token(prompt)
    return lable
```

In [5]:

```
def crossentropyloss(pred, target):
    '''Cross entropy loss that does not average across samples.'''
    if pred.ndim == 1:
        pred = pred[:, np.newaxis]
        pred = np.concatenate((1 - pred, pred), axis=1)

    if pred.shape == target.shape:
        # Soft cross entropy loss.
        pred = np.clip(pred, a_min=1e-12, a_max=1-1e-12)
        return - np.sum(np.log(pred) * target, axis=1)
    else:
        # Standard cross entropy loss.
        return - np.log(pred[np.arange(len(pred)), target])

class DatasetLossGame:
    '''
    Cooperative game representing the model's loss over a dataset.

    TODO: this implementation is slower than SAGE because it averages
```

```
    loss over entire dataset for each S. Need to reimplement as a stochas
    game (with caching) to accelerate convergence.

    Args:
      extension: model extension (see removal.py).
      data: array of model inputs.
      labels: array of corresponding labels.
      loss: loss function (see utils.py).
    '''
    def __init__(self, extension, data, labels, loss):
        # Convert labels dtype if necessary.
        if loss is crossentropyloss:
            # Make sure not soft cross entropy.
            if (labels.ndim == 1) or (labels.shape[1] == 1):
                # Only convert if float.
                if np.issubdtype(labels.dtype, np.floating):
                    labels = labels.astype(int)

        self.extension = extension
        self.data = data
        self.labels = labels
        self.loss = loss
        self.players = data.shape[1]
        self.data_tile = self.data
        self.label_tile = self.labels

    def __call__(self, S):
        # Return scalar is single subset.
        single_eval = (S.ndim == 1)
        if single_eval:
            S = S[np.newaxis]

        # Prepare data.
        if len(self.data_tile) != len(self.data) * len(S):
            self.data_tile = np.tile(self.data, (len(S), 1))
            self.label_tile = np.tile(
                self.labels,
                (len(S), *[1 for _ in range(len(self.labels.shape[1:]))]))
        S = S.repeat(len(self.data), 0)

        # Evaluate.
        output = - self.loss(self.extension(self.data_tile, S), self.labe
        output = output.reshape((-1, self.data.shape[0]))
        output = np.mean(output, axis=1)
        if single_eval:
            output = output[0]
        return output

def default_batch_size(game):
    '''
    Determine batch size.

    TODO maybe consider the number of features, or the type of model exte
    '''
    if isinstance(game, DatasetLossGame):
        return 32
    else:
        return 512

class MarginalExtension:
```

```
    '''Extend a model by marginalizing out removed features using their
    marginal distribution.'''

    def __init__(self, data, model):
        self.model = model
        self.data = data
        # self.data_repeat = data
        self.samples = len(data)
        # self.x_addr = None
        # self.x_repeat = None

    def __call__(self, x, S):
        # Prepare x and S.
        n = len(x)
        x = x.repeat(self.samples, 0)
        S = S.repeat(self.samples, 0)
        # if self.x_addr != id(x):
        #     self.x_addr = id(x)
        #     self.x_repeat = x.repeat(self.samples, 0)
        # x = self.x_repeat

        # Prepare samples.
        # if len(self.data_repeat) != self.samples * n:
        self.data_repeat = np.tile(self.data.values[:,0:S.shape[1]], (n,

        # Replace specified indices.
        x_ = x.copy()
        x_[~S] = self.data_repeat[~S]

#         return x_

        # Make predictions.
        pred = self.model.predict(torch.tensor(x_))

        return pred;
        pred = pred.reshape(-1, self.samples, *pred.shape[1:])
        return np.mean(pred, axis=1)

class PredictionGame:
    '''
    Cooperative game for an individual example's prediction.

    Args:
      extension: model extension (see removal.py).
      sample: numpy array representing a single model input.
    '''

    def __init__(self, extension, sample, input_col, att_col):
        # Add batch dimension to sample.
        if sample.ndim == 1:
            sample = sample[np.newaxis]
        elif sample.shape[0] != 1:
            raise ValueError('sample must have shape (ndim,) or (1,ndim)'

        self.extension = extension
        self.sample = sample
        self.input_col = input_col
        self.att_col = att_col
```

```
        self.players = sample[att_col].sum(axis = 1)[0]

        # Caching.
        self.sample_repeat = sample

    def __call__(self, S):
        # Return scalar if single subset.
        single_eval = (S.ndim == 1)
        if single_eval:
            S = S[np.newaxis]
            input_data = self.sample
        else:
            # Try to use caching for repeated data.
            if len(S) != len(self.sample_repeat):
                self.sample_repeat = self.sample[self.input_col].values[:
            input_data = self.sample_repeat

        # Evaluate.
        output = self.extension(input_data, S)
        if single_eval:
            output = output[0]
        return output

def RemoveIndividual(game, batch_size=None):
    '''Calculate feature attributions by removing individual
    players from the grand coalition.'''
    if batch_size is None:
        batch_size = default_batch_size(game)

    # Setup.
    n = game.players
    S = np.ones((n + 1, n), dtype=bool)
    for i in range(n):
        S[i + 1, i] = 0

    # Evaluate.
    output_list = []
    for i in range(int(np.ceil(len(S) / batch_size))):
        output_list.append(game(S[i * batch_size:(i + 1) * batch_size]))
    output = np.concatenate(output_list, axis=0)

#     return output

    return output[0] - output[1:]
```

In [6]:

```
torch.cuda.is_available()

from kaggle_secrets import UserSecretsClient
user_secrets = UserSecretsClient()
access_token = user_secrets.get_secret("hf_access_code")

print(access_token)
tokenizer = AutoTokenizer.from_pretrained("meta-llama/Meta-Llama-3.1-8B-I

quantization_config = BitsAndBytesConfig(load_in_4bit=True)

model = AutoModelForCausalLM.from_pretrained(
    "meta-llama/Meta-Llama-3.1-8B-Instruct",
    token=access_token,
```

```
    quantization_config=quantization_config
)
```

```
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
```

```
`low_cpu_mem_usage` was None, now set to True since model is quantized.
model.safetensors.index.json:   0%|          | 0.00/23.9k [00:00<?, ?B/s]
Downloading shards:   0%|          | 0/4 [00:00<?, ?it/s]
model-00001-of-00004.safetensors:   0%|          | 0.00/4.98G [00:00<?, ?
B/s]
model-00002-of-00004.safetensors:   0%|          | 0.00/5.00G [00:00<?, ?
B/s]
model-00003-of-00004.safetensors:   0%|          | 0.00/4.92G [00:00<?, ?
B/s]
model-00004-of-00004.safetensors:   0%|          | 0.00/1.17G [00:00<?, ?
B/s]
Loading checkpoint shards:   0%|          | 0/4 [00:00<?, ?it/s]
generation_config.json:   0%|          | 0.00/184 [00:00<?, ?B/s]
```

In [7]:
```
import warnings
warnings.filterwarnings('ignore')

import logging
logging.getLogger("transformers").setLevel(logging.ERROR)
```

In [26]:
```
text_to_analyse = 50

sample_to_replace = [5,2]


llama_model = Llama(model, tokenizer)

train = pd.read_csv("/kaggle/input/fake-new-covid/Constraint_Train.csv")
train['prompt'] = train["tweet"].apply(lambda x: prompt_generation(x))
sentences = list(train["prompt"])


tokenizer.pad_token = tokenizer.eos_token

# Tokenize the sentences with padding
encoded_inputs = tokenizer(
    sentences,
    padding=True,  # Pads all sequences to the same length
    return_tensors="pt"  # Returns PyTorch tensors
)


token_input_ids = encoded_inputs["input_ids"]
token_attention_mask = encoded_inputs["attention_mask"]

token_input_column  = [f"col_input_{i}" for i in range(token_input_ids.sh
token_att_column  = [f"col_att_{i}" for i in range(token_attention_mask.s

token_input_df = pd.DataFrame(token_input_ids, columns = token_input_colu
token_att_df = pd.DataFrame(token_attention_mask, columns = token_att_col

train_v1 = pd.concat([train,token_input_df,token_att_df], axis = 1)

Llama_v1 = Llama(model, tokenizer)

removal = MarginalExtension(
```

```
    train_v1[token_input_column +token_att_column][9:13].reset_index(drop
    Llama_v1
)
behavioural = PredictionGame(removal,
                             train_v1[token_input_column +token_att_colum
                             token_input_column,
                             token_att_column)
summary = RemoveIndividual(behavioural)
```

In [27]:
```
lenght_of_sentence = train_v1[text_to_analyse:text_to_analyse+1][token_at

importance = {}
for i in range(lenght_of_sentence):
    importance[tokenizer.decode(train_v1[token_input_column].values[text_
```

Explaining the prediction

In [29]:
```
import random
from IPython.core.display import display, HTML
import matplotlib.pyplot as plt


def highlighter(word):
    # Default color if the word is not in the importance dictionary
    default_color = "#FFFFFF"  # White background

    # Get the importance value (with a default of 0 if not found)
    importance_value = importance.get(word, 0)

    # Normalize the importance value to range [-1, 1] for a diverging col
    norm = plt.Normalize(vmin=-1, vmax=1)

    # Use a diverging colormap (e.g., coolwarm) to get the color based on
    color = plt.cm.coolwarm(norm(importance_value))

    # Convert the color from RGBA to Hex
    color_hex = "#{:02x}{:02x}{:02x}".format(int(color[0] * 255), int(col

    # Highlight the word with the corresponding color
    word = f'<span style="background-color:{color_hex}">' + word + '</spa

    return word


print("Here Red color suggest positive importance and Blue color suggest

text = ''.join([ highlighter(tokenizer.decode(train_v1[token_input_column

display(HTML(text))
```

```
Here Red color suggest positive importance and Blue color suggest negetive
importance
```

<|begin_of_text|>Please select the option (A or B) that most closely describes the following claim: Masks can help prevent the spread of #COVID19 when they are widely used in public. When you wear a mask you can help protect those around you. When others wear one they can help protect people around them incl. you. https://t.co/jkW

Prediction

In [16]:
```
token_for_text = torch.tensor(train_v1[token_input_column].values[text_to
with torch.no_grad():
    output_ids = model.generate(
        token_for_text,
        max_length=token_for_text.shape[1] + 3,
        num_return_sequences=1,
        do_sample=False,
        return_dict_in_generate=True, output_scores=True
    )
```

In [17]:
```
tokenizer.decode(output_ids.sequences[0])
```

Out[17]:
```
'<|begin_of_text|>Please select the option (A or B) that most closely de
scribes the following claim: States reported 1121 deaths a small rise fr
om last Tuesday. Southern states reported 640 of those deaths. https://
t.co/YASGRTT4ux.\n    (A) True\n    (B) False\n    Choice: (A) True'
```

In [ ]: