DACON - Judicial Precedent Prediction

Throwback

This was my first competition that I tried on Natural Language Processing task.
Even though the result was not that great, I learned a lot from this competition.
I have understood how the Language model
`BERT` is formed and how to use it.
Moreover, I have created a basic library for text processing.
It includes a Python Class that simplifies pandas functions, text processors with `regex` formats and `BERT Tokenizer` & `Auto Tokenizer`.
Later I found out that using PyTorch could have been way easier to use these tokenizer modules.

월간 데이콘 법원 판결 예측 AI 경진대회

출처 : DACON - Data Science Competition

https://dacon.io/competitions/official/236112/overview/description

This competition has just started right after finishing the last competition in Acoustic - Emotion Classification.

As there were no people competing yet, I tried my best to have the result as fast as I can to take the first place 🙂

Below is a brief requirements for this classification competition.

▼ Requirements

[Background]
Hello everyone! 😀 Welcome to the monthly Deacon court decision prediction AI competition.
The challenge for this Monthly Deacon is to develop an AI that predicts the outcome of a legal case.
We want you to create a model that utilizes the full power of AI to make accurate verdict predictions.
This will be an important step in exploring how AI can be effectively utilized in the legal field.

[Topics]
Develop an AI model to predict court decisions

[Description]

The provided dataset contains the case identifiers of US Supreme Court cases and the content of the case.
You need to develop an AI model that predicts whether the first party or the second party wins in a particular case.

[Evaluation]
Judging Criteria:
Accuracy
Primary Evaluation (Public Score): Scored on a randomly sampled 50% of the test data, publicly available during the competition.
Secondary Evaluation (Private Score): Scored with 100% of the test data, released immediately after the competition ends
The final ranking will be scored from the selected files, so participants must select the two files they would like to have scored in the submission window.
Final ranking is based on the highest score of the two selected files.
(If the final file is not selected, the first submitted file is automatically selected)
The Private Score ranking published immediately after the competition is not the final ranking; the final winners will be determined after code verification.

Determine the final ranking based on Private Score among the submitting teams that followed the competition evaluation rules.

We are given three csv files : `train`, `test`, `sample_submission`.

`train` file includes columns below:

`"ID"` : Index of each case

`"first_party"` : Name of the person / party competing against `second_party`

`"second_party"` : Name of the person / party competing against `first_party`

`"facts"` : Brief facts about each case

`"first_party_winner"` : Whether or not `first_party` won the case (0 or 1)

As I was now quite familiar with the competitions, I knew what I had to do.

1. Preprocess the data
2. Run the Machine Learning Model
3. Submit twice with different `train_accuracy_score` and `validation_accuracy_score` to check whether I should aim for higher accuracy_score or not.

▼ **Achieving the First Place (Time-Attack)**

To achieve the first place, I directly worked on preprocessing.
As it is a competition that we need to find either 0 or 1 for '`first_party_winner`', I simply divided the original data into `trian` and `val` dataset then ran XG Boost.

However, a crucial fact has blocked me on the way.
XG Boost does not allow object type in neither `train` nor `val`.
Then I quickly searched how to change them into numerical value.
What I found was LabelEncoder() that groups the identical string and number them.

▼ `Codes`

```
train = pd.read_csv('./train.csv')
test = pd.read_csv('./test.csv')
```

```
sample_submission = pd.read_csv('./sample_submission.csv')

label_encoder = LabelEncoder()
label_encoder.fit(train['ID'])
train['ID'] = label_encoder.transform(train['ID'])
label_encoder.fit(train['first_party'])
train['first_party'] = label_encoder.transform(train['first_party'])
label_encoder.fit(train['second_party'])
train['second_party'] = label_encoder.transform(train['second_party'])
label_encoder.fit(train['facts'])
train['facts'] = label_encoder.transform(train['facts'])

label_encoder.fit(test['ID'])
test['ID'] = label_encoder.transform(test['ID'])
label_encoder.fit(test['first_party'])
test['first_party'] = label_encoder.transform(test['first_party'])
label_encoder.fit(test['second_party'])
test['second_party'] = label_encoder.transform(test['second_party'])
label_encoder.fit(test['facts'])
test['facts'] = label_encoder.transform(test['facts'])


train_x = train.drop('first_party_winner', axis=1)
train_y = train['first_party_winner']
test_x = test
train_x
```

After encoding them into numerical values, I ran XG Boost right away then submitted it.

▼ `Codes`

```
X_train, X_val, y_train, y_val = train_test_split(train_x, train_y, test_size=
0.3, random_state=42)
print(X_train.shape, X_val.shape, y_train.shape, y_val.shape)
XGB = XGBClassifier(max_depth=10,
                    n_estimators=50,
                    grow_policy='depthwise',
                    n_jobs=-1,
                    random_state=42,
                    tree_method='auto'
                    )
XGB.fit(X_train, y_train)
print("-- XGB --")
print("Train ACC : %.3f" % accuracy_score(y_train, XGB.predict(X_train)))
print("Val ACC : %.3f" % accuracy_score(y_val, XGB.predict(X_val)))

print(test_x)
X_test = pd.get_dummies(data=test_x)
print(X_test)
XGB_pred = XGB.predict(test_x)
```

```
XGB_submission = pd.read_csv('./sample_submission.csv')
XGB_submission['first_party_winner'] = XGB_pred
XGB_submission.to_csv("./XGB_tryout_submission.csv", index=False)
```

This method has given me the first place, but I wanted to develop more and protect my place.
Moreover, I had to check the system on the metric.

Therefore, I have modified a bit of hyperparameter (=`n_estimator` : 50 ⇒ 25)

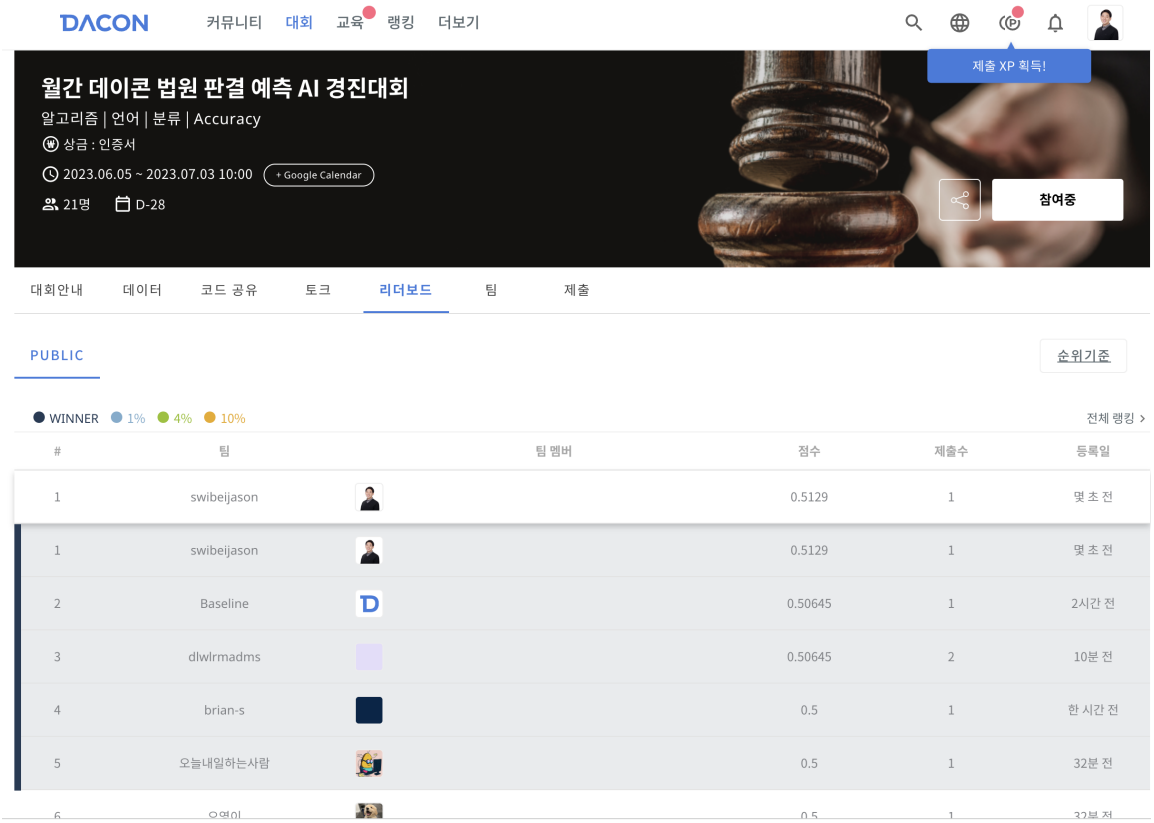
The result of the first set and the second score were as below.

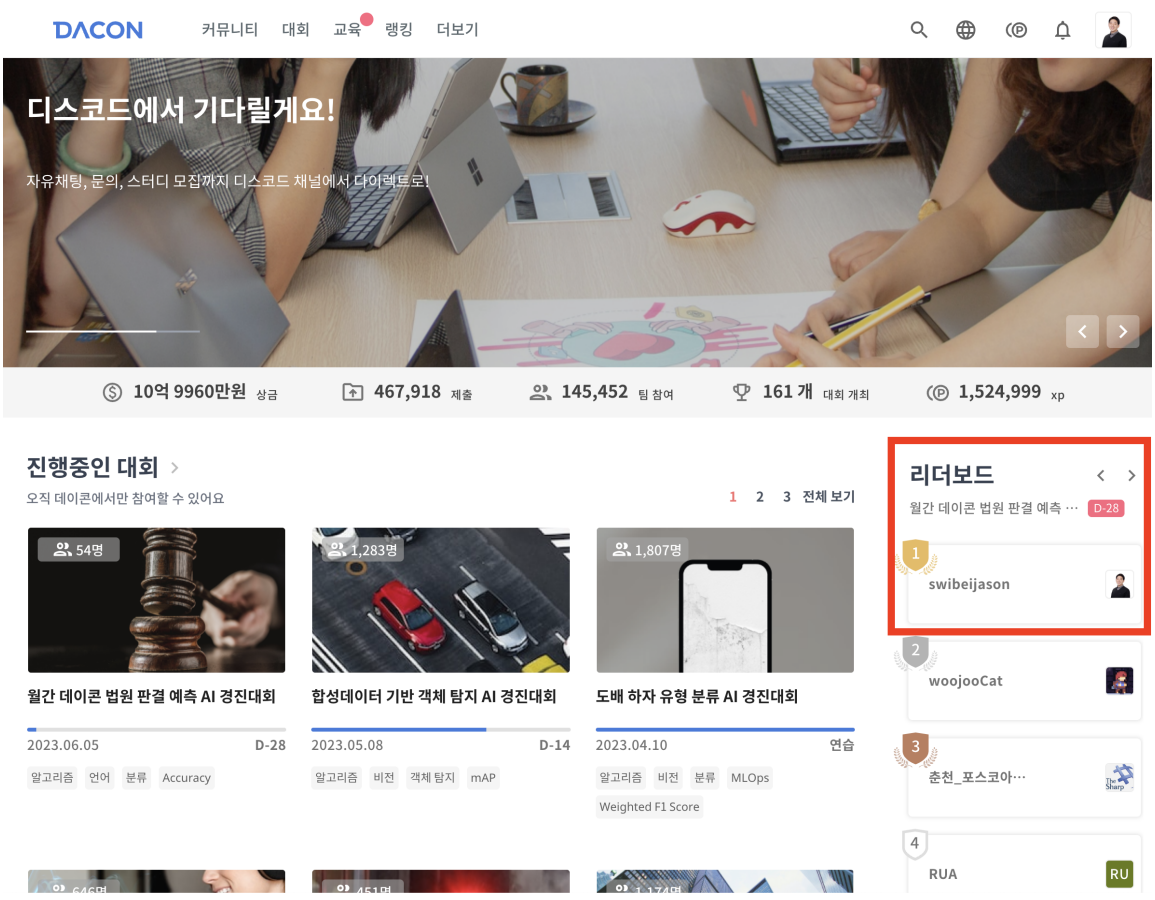
▼ **Codes**

```
-- First_Attempt XGB --
Train ACC : 0.997
Val ACC : 0.617

-- Second Attempt XGB --
Train ACC : 0.953
Val ACC : 0.636
```

I thought the result will be horrible, but I managed to take the first place and remain for 8 ~ 9 hours.


DACON
커뮤니티 대회 교육 랭킹 더보기
제출 XP 획득!
월간 데이콘 법원 판결 예측 AI 경진대회
알고리즘 | 언어 | 분류 | Accuracy
상금 : 인증서
2023.06.05 ~ 2023.07.03 10:00
+ Google Calendar
21명
D-28
참여중
대회안내 데이터 코드 공유 토크 리더보드 팀 제출
PUBLIC
순위기준
WINNER 1% 4% 10%
전체 랭킹 >
팀 팀 멤버 점수 제출수 등록일
1 swibeijason 0.5129 1 몇 초 전
1 swibeijason 0.5129 1 몇 초 전
2 Baseline 0.50645 1 2시간 전
3 dlwlrmadms 0.50645 2 10분 전
4 brian-s 0.5 1 한 시간 전
5 오늘내일하는사람 0.5 1 32분 전


DACON
커뮤니티 대회 교육 랭킹 더보기
디스코드에서 기다릴게요!
자유채팅, 문의, 스터디 모집까지 디스코드 채널에서 다이렉트로!
10억 9960만원 상금
467,918 제출
145,452 팀 참여
161 개 대회 개최
1,524,999 xp
진행중인 대회 >
오직 데이콘에서만 참여할 수 있어요
1 2 3 전체 보기
54명
월간 데이콘 법원 판결 예측 AI 경진대회
2023.06.05 D-28
알고리즘 언어 분류 Accuracy
1,283명
합성데이터 기반 객체 탐지 AI 경진대회
2023.05.08 D-14
알고리즘 비전 객체 탐지 mAP
1,807명
도배 하자 유형 분류 AI 경진대회
2023.04.10 연습
알고리즘 비전 분류 MLOps Weighted F1 Score
리더보드
월간 데이콘 법원 판결 예측 … D-28
1 swibeijason
2 woojooCat
3 춘천_포스코아…
4 RUA

While enjoying the glory of being the first place, I suddenly realized that this is not just a classification competition.
But this was a NLP (Natural Language Processing) competition.

There were only 2478 rows in the train dataset, and 1240 rows in the test dataset.

- Dataset Information

```
train.info()
print("\n ---------------------------------- \n")
test.info()
-------------------------------------------------------------

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2478 entries, 0 to 2477
Data columns (total 5 columns):
 #   Column              Non-Null Count  Dtype 
---  ------              --------------  ----- 
 0   ID                  2478 non-null   object
 1   first_party         2478 non-null   object
 2   second_party        2478 non-null   object
 3   facts               2478 non-null   object
 4   first_party_winner  2478 non-null   int64 
dtypes: int64(1), object(4)
memory usage: 96.9+ KB

 ----------------------------------

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1240 entries, 0 to 1239
Data columns (total 4 columns):
 #   Column        Non-Null Count  Dtype 
---  ------        --------------  ----- 
 0   ID            1240 non-null   object
 1   first_party   1240 non-null   object
 2   second_party  1240 non-null   object
 3   facts         1240 non-null   object
dtypes: object(4)
memory usage: 38.9+ KB
```

I decided to join this competition aiming to master my skills for classification…
But now I am destined to try NLP.

I have heard about GPT, BERT, but am not yet familiar with them.
So, I went on Google to first search about NLP, understood that what I am solving is a NLP Classification Problem.

Below is the first article I found online, and it well informs different models and where each of them are strong at.

1. 자연어 처리 모델 소개 (Introduction to NLP Model) — PseudoLab Tutorial Book

https://pseudo-lab.github.io/Tutorial-Book/chapters/NLP/Ch1-Introduction.html

It seemed to me that T5 and GPT were made for text-to-text conversations, while BERT seemed more suitable for predictions, classifications and such.

So I have searched a bit more to learn more about BERT.
As this was my first time actually looking into DeepLearning Models (except for the torch trial during the last competition), I thought there was one set of simple codes to activate the DeepLearning algorithms to work.
But it was a huge mistake to assume it that way.

There were hundreds of different models using BERT, and I had to choose which one to use.
I first looked into one scholarly article, that researched into Judicial Precedent Analysis with SBERT (Sentence BERT).

https://manuscriptlink-society-file.s3-ap-northeast-1.amazonaws.com/kips/conference/ack2022/presentation/KIPS_C2022B0013.pdf

'HolLaw' A Judicial Precedent Analysis Service using NLP and SBERT
S. Yoon, S. Kim, J. Lee, J. Oh, & N. Kim, 2022, Incheon National University.

After reviewing this article, I understood that I could also use BERT for preprocessing data.

Then, I found this article from TensorFlow about Text Classification using BERT, Sentence_Transformers.

BERT로 텍스트 분류 | Text | TensorFlow

https://www.tensorflow.org/text/tutorials/classify_text_with_bert?hl=ko

sentence-transformers/all-MiniLM-L6-v2 · Hugging Face

We're on a journey to advance and democratize artificial intelligence through open source and open science.

https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2

Had to do another set of DataFrame preprocessing to fit into model.

▼ Additional DF Preprocessing Codes

```
# definitions

def text_processor(s):
    """
    문장을 담고있는 variable을 넣어주면
    알파벳을 제외한 문장의 모든 기호, 숫자를 제거합니다.

    :param s: 문장을 담고있는 variable
    :return: 새로운 DataFrame안에 담긴 text_processor가 적용된 column
    """

    pattern = r'\([^)]*\)'  # ()
    s = re.sub(pattern=pattern, repl='', string=s)
    pattern = r'\[[^)]*\]'  # []
    s = re.sub(pattern=pattern, repl='', string=s)
    pattern = r'\<[^)]*\>'  # <>
    s = re.sub(pattern=pattern, repl='', string=s)
    pattern = r'\{[^)]*\}'  # {}
    s = re.sub(pattern=pattern, repl='', string=s)


    pattern = r'[^a-zA-Z]'
    s = re.sub(pattern=pattern, repl=' ', string=s)

    months = ['on january', 'on february', 'on march', 'on april', 'on may', 'on j
une', 'on july', 'on august', 'on september', 'on october', 'on november', 'on dec
ember', 'jan', 'feb', 'mar', 'apr', 'jun', 'jul', 'aug', 'sep', 'oct', 'nov', 'de
c']
    for month in months:
        s = s.lower()
        s = s.replace(month, '')


    units = ['mm', 'cm', 'km', 'ml', 'kg', 'g', 'th', 'st', 'rd', 'nd']
    for unit in units:
        s = s.lower()
        s = s.replace(unit, '')

    s_split = s.split()

    s_list = []
```

```
    for word in s_split:
        if len(word) != 1:
            s_list.append(word)

    s_list = " ".join(s_list)

    return s_list


def law_processor(df, column):
    '''
    입력한 df의 column에서
    알파벳을 제외한 모든 숫자, 기호를 제거합니다.

    :param df: 대상이 될 DataFrame
    :param column: df에서 대상이 될 Column
    :return: 새로운 DataFrame안에 담긴 text_processor가 적용된 column
    '''
    temp = []
    for i in range(len(df)):
        temp.append(text_processor(df[f'{column}'][i]))
    temp_dict = {f"{column1}": temp1}

    processed = pd.DataFrame(temp_dict)
    return processed
```

```
df = pd.DataFrame(train['facts'])
df = dlc.law_preprocessor(df, 'facts')
train['facts'] = df['facts']
df = pd.DataFrame(train['first_party'])
df = dlc.law_preprocessor(df, 'first_party')
train['first_party'] = df['first_party']
df = pd.DataFrame(train['second_party'])
df = dlc.law_preprocessor(df, 'second_party')
train['second_party'] = df['second_party']
train_cleansed = train.drop(columns='ID')
train_cleansed
```

Out[10]:

	ID	first_party	second_party	facts	first_party_winner
0	TRAIN_0000	Phil A. St. Amant	Herman A. Thompson	phil amant caidate for public office made tele...	1
1	TRAIN_0001	Stephen Duncan	Lawrence Owens	ramon nelson was ridin his bike when he suffer...	0
2	TRAIN_0002	Billy Joe Magwood	Tony Patterson, Warden, et al.	an alabama ate court convicted billy joe mawoo...	1
3	TRAIN_0003	Linkletter	Walker	victor linkletter was convicted in ate court o...	0
4	TRAIN_0004	William Earl Fikes	Alabama	in selma alabama an intruder broke into apartm...	1
...	...	...	...	...	...
2473	TRAIN_2473	HollyFrontier Cheyenne Refining, LLC, et al.	Renewable Fuels Association, et al.	conress ameed clean air act rouh enery policy ...	1
2474	TRAIN_2474	Grupo Mexicano de Desarrollo, S. A.	Alliance Bond Fund, Inc.	alliance bo fu inc an invement fu purchased ap...	1
2475	TRAIN_2475	Peguero	United States	in dirict court sentenced manuel peuero to mon...	0
2476	TRAIN_2476	Immigration and Naturalization Service	St. Cyr	enrico cyr lawful permanent resident pled uilt...	0
2477	TRAIN_2477	Markman	Westview Instruments, Inc.	herbert an owns patent to syem at tracks cloin...	0

I tried my best to understand, and implement it to my `jupyter notebook`.
Below is the first implementation, which I failed.

▼ BERT Implementation Codes (Failed)

```
# BERT definitions

def bert_tokenizer(df, column_name):
    '''
    입력한 df의 문자 벡터를 수치화 합니다.

    :param df:문자 벡터를 수치화하고 DataFrame
    :return:
    '''
    bert_model = 'bert-base-uncased'
    tokenizer = AutoTokenizer.from_pretrained(bert_model)


    ei_total_list = []
    for i in tqdm(df[column_name]):
        ei_list = []
        for j in range(1):
            encoded_input = tokenizer(i, padding='max_length', max_length=512, tru
ncation=True, return_tensors='pt')
            ei_list.append(encoded_input.items())
        ei_total_list.append(ei_list)
    df_1 = pd.DataFrame(ei_total_list, columns="tensors")
    return df_1

def auto_tokenizer(df, column_name):
    '''
    입력한 df의 문자 벡터를 수치화 합니다.

    :param df:문자 벡터를 수치화하고 DataFrame
    :return:
    '''
    model = 'sentence-transformers/all-MiniLM-L6-v2''bert-base-uncased'
    tokenizer = AutoTokenizer.from_pretrained(model)
    model = AutoModel.from_pretrained(model)

    ei_total_list = []
    for i in tqdm(df[column_name]):
        ei_list = []
        for j in range(1):
            encoded_input = tokenizer(i, padding='max_length', max_length=512, tru
ncation=True, return_tensors='pt')
            with torch.no_grad():
                model_output = model(**encoded_input)
            sentence_embeddings = mean_pooling(model_output, encoded_input['attent
ion_mask'])
            sentence_embeddings = F.normalize(sentence_embeddings, p=2, dim=1)
            ei_list.append(sentence_embeddings)
        ei_total_list.append(ei_list)
    df_1 = pd.DataFrame(ei_total_list)
    return df_1
```

```
## BERT Failed

train_facts = pd.DataFrame(train_cleansed['facts'])
test_cleansed = pd.DataFrame(test_cleansed['facts'])
model_input_df = dlc.bert_tokenizer(train_facts, 'facts')

df_part_1, df_part_2, df_part_3, df_part_4, df_part_5, df_part_6, df_part_7, df_pa
rt_8, df_part_9, df_part_10, df_part_11, df_part_12, df_part_13, df_part_14, df_pa
rt_15, df_part_16, df_part_17, df_part_18, df_part_19, df_part_20, df_part_21, df_
part_22, df_part_23, df_part_24, df_part_25, df_part_26 = so.df_divider(train_clea
nsed, 'facts')

df_part_1 = pd.DataFrame(df_part_1)
df_part_2 = pd.DataFrame(df_part_2)
df_part_3 = pd.DataFrame(df_part_3)
df_part_4 = pd.DataFrame(df_part_4)
df_part_5 = pd.DataFrame(df_part_5)
df_part_6 = pd.DataFrame(df_part_6)
df_part_7 = pd.DataFrame(df_part_7)
df_part_8 = pd.DataFrame(df_part_8)
df_part_9 = pd.DataFrame(df_part_9)
df_part_10 = pd.DataFrame(df_part_10)
df_part_11 = pd.DataFrame(df_part_11)
df_part_12 = pd.DataFrame(df_part_12)
df_part_13 = pd.DataFrame(df_part_13)
df_part_14 = pd.DataFrame(df_part_14)
df_part_15 = pd.DataFrame(df_part_15)
df_part_16 = pd.DataFrame(df_part_16)
df_part_17 = pd.DataFrame(df_part_17)
df_part_18 = pd.DataFrame(df_part_18)
df_part_19 = pd.DataFrame(df_part_19)
df_part_20 = pd.DataFrame(df_part_20)
df_part_21 = pd.DataFrame(df_part_21)
df_part_22 = pd.DataFrame(df_part_22)
df_part_23 = pd.DataFrame(df_part_23)
df_part_24 = pd.DataFrame(df_part_24)
df_part_25 = pd.DataFrame(df_part_25)
df_part_26 = pd.DataFrame(df_part_26)


embedded_df_1 = dlc.auto_tokenizer(train_cleansed, 'facts')
embedded_df_1
embedded_df_1 = embedded_df_1.rename(columns={0:'facts_berted'})
embedded_df_1.to_csv('./embeddings/facts_embedded.csv', index=False)
```

The main reason of the failure was that I did not have enough understanding of the NLP Preprocessing process.
As per my current understanding, NLP Preprocessing has to go through 3 main steps.

1. **Encoding**
2. **Tokenizing**
3. **Embedding**

However, the steps I followed on the first trial was as below.

1. **Encoding**
2. **Tokenizing**
3. **Encoding**
4. **Tokenizing**
5. **Embedding**

I went through Encoding and Tokenizing process again, not knowing that I have already done it.
This was why the attempt was keep failing, and I thought the problem was the amount of the data I had. And that is why I tried to separate the rows into 26, which seemed like it wouldn’t cause any trouble.

I was so shocked at the moment I realized the error.
Then I modified the code as below.

▼ **Codes**

```
def auto_tokenizer(df, column_name):
    '''
    입력한 df의 문자 벡터를 수치화 합니다.

    :param df:문자 벡터를 수치화하고 DataFrame
    :return:
    '''
    bert_model = 'bert-base-uncased'
    tokenizer = AutoTokenizer.from_pretrained(bert_model)
    model = AutoModel.from_pretrained(bert_model)

    ei_total_list = []
    for i in tqdm(df[column_name]):
        ei_list = []
        for j in range(1):
            encoded_input = tokenizer(i, padding='max_length', max_length=512, tru
ncation=True, return_tensors='pt')
```

```
            with torch.no_grad():
                model_output = model(**encoded_input)
            sentence_embeddings = mean_pooling(model_output, encoded_input['attent
ion_mask'])
            sentence_embeddings = F.normalize(sentence_embeddings, p=2, dim=1)
            ei_list.append(sentence_embeddings)
        ei_total_list.append(ei_list)
    df_1 = pd.DataFrame(ei_total_list)
    return df_1

first_party_berted = dlc.auto_tokenizer(train_cleansed, 'first_party')
second_party_berted = dlc.auto_tokenizer(train_cleansed, 'second_party')
facts_berted = dlc.auto_tokenizer(train_cleansed, 'facts')

test_second_party_berted = dlc.auto_tokenizer(test_cleansed, 'test_second_party_be
rted')
test_second_party_berted = test_second_party_berted.rename(columns={0:'second_part
y_berted'})
test_second_party_berted.to_csv('./embeddings/test_second_party_berted.csv', index
=False)

test_facts_berted = dlc.auto_tokenizer(test_cleansed, 'test_facts_berted')
test_facts_berted = test_facts_berted.rename(columns={0:'test_facts_berted'})
test_facts_berted.to_csv('./embeddings/test_facts_berted.csv', index=False)

# This process took so long, I had to export to csv and import it for later uses
:)

test_first_party_berted = pd.read_csv('./embeddings/test_first_party_berted.csv')
test_second_party_berted = pd.read_csv('./embeddings/test_second_party_berted.cs
v')
test_facts_berted = pd.read_csv('./embeddings/test_facts_berted.csv')
```

```
Some weights of the model checkpoint at bert-base-uncased were not used when initializing BertModel: ['cls.prediction
s.transform.dense.weight', 'cls.predictions.bias', 'cls.predictions.transform.LayerNorm.weight', 'cls.predictions.tra
nsform.dense.bias', 'cls.seq_relationship.weight', 'cls.predictions.transform.LayerNorm.bias', 'cls.seq_relationship.
bias', 'cls.predictions.decoder.weight']
- This IS expected if you are initializing BertModel from the checkpoint of a model trained on another task or with a
nother architecture (e.g. initializing a BertForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing BertModel from the checkpoint of a model that you expect to be exactly
identical (initializing a BertForSequenceClassification model from a BertForSequenceClassification model).
100%|██████████████████████████████████████████████████| 1240/1240 [06:20<00:00,  3.26it/s]
```

I was glad, it worked so well!!

This was the output for the next step.

	first_party_berted	second_party_berted	test_facts_berted
0	tensor([[-6.6722e-03, -8.0011e-02, -1.4580e-02...	tensor([[2.6328e-02, -1.6656e-02, -2.0892e-02...	tensor([[-4.1743e-02, -4.1057e-02, 3.7544e-02...
1	tensor([[-3.0531e-03, -1.1769e-02, -3.4803e-02...	tensor([[1.0108e-02, 1.5938e-03, 6.1594e-03...	tensor([[-6.2039e-02, -9.7709e-03, 3.8074e-02...
2	tensor([[-2.5638e-02, -1.6005e-02, -2.1478e-02...	tensor([[1.4963e-02, 1.2814e-02, -2.9757e-02...	tensor([[-5.1186e-02, -2.1976e-05, 2.2741e-02...
3	tensor([[2.8991e-02, 3.6599e-02, -5.9128e-02...	tensor([[2.6328e-02, -1.6656e-02, -2.0892e-02...	tensor([[-5.2218e-02, -2.7130e-02, 2.2865e-02...
4	tensor([[3.6941e-02, -3.9247e-02, -2.6675e-02...	tensor([[3.2124e-02, -1.3590e-02, -4.0433e-02...	tensor([[-3.7322e-02, 4.6000e-03, 2.6736e-02...
...	...	...	...
1235	tensor([[-1.7606e-02, -2.6507e-02, -3.8503e-02...	tensor([[-3.6597e-02, -2.9370e-02, -8.3051e-03...	tensor([[-5.7088e-02, -6.9000e-04, 2.8725e-02...
1236	tensor([[-1.2198e-03, -2.3640e-03, -2.5906e-02...	tensor([[-1.0735e-02, 1.2387e-02, -2.2711e-02...	tensor([[-5.7708e-02, 1.6911e-02, 1.7108e-02...
1237	tensor([[1.0257e-03, 2.3440e-02, 1.3377e-02...	tensor([[1.8597e-02, 2.6318e-02, -3.0361e-02...	tensor([[-5.3087e-02, -1.4558e-02, 2.6369e-02...
1238	tensor([[1.6344e-03, -8.1419e-04, -1.4695e-02...	tensor([[1.8647e-02, 2.5467e-03, -2.3004e-02...	tensor([[-4.9280e-02, -3.1140e-03, 6.9771e-03...
1239	tensor([[-5.8938e-02, 1.0668e-02, 4.9173e-03...	tensor([[-1.6965e-02, 2.8206e-03, -4.9479e-03...	tensor([[-6.2537e-02, 5.0910e-03, 2.8591e-02...

1240 rows × 3 columns

However, I realized that most of the Machine Learning Models wouldn't take this form as an input.

But what is TENSOR?!
After few minutes of research, I found out that these are multi-dimensional frame.
I was suffering to convert these into 2-dimensional frame, but it was very confusing and harder than I thought.

This was the first attempt on converting this 4-dimensional frame into 2-dimension.

▼ **Codes (Fail)**

```
def tensor_2_2d(df, n):
    df_renamed = df.rename(columns={0: 'tbd', 1: 'hmm'})
    tensors = pd.DataFrame(df_renamed.groupby(by="tbd"))
    tensors1 = tensors[1]
    tensors1_df = pd.DataFrame(tensors1)
    tensors1_1 = pd.DataFrame(tensors1_df[1][n])
    target_name_temp = tensors1_1['tbd']
    target = tensors1_1['hmm']
    target_name_df = pd.DataFrame(target_name_temp)
    target_name = target_name_df.iat[0, 0]
    target_df = pd.DataFrame(target)
    target_df = target_df.reset_index()
    target_df = target_df.drop(columns='index')
    target_final_df = target_df.rename(columns={'hmm': target_name})

    temp = []
    for i in tqdm(range(len(target_final_df))):
        units = ['[', ']', 'tensor', '(', ')']

        for unit in units:
            s = str(target_final_df[target_name][i]).replace(unit, '')
        temp.append(s)
```

```
    temp_dict = {target_name: temp}

    final_df = pd.DataFrame(temp_dict)

    return final_df
```

It didn't work well, and I still couldn't use it as Xs and y.

Then I thought, why don't I simply convert these into string, process it, and convert them back to numerics.

Here is the code I came up with, and it worked well!

▼ **Codes (Success)**

```
def tensor_separator(df, column_name):
    to_replace = ["t e n s o r", "[", "]", "(", ")", " "]
    full_tensor_list =[]
    for tensor in tqdm(df[column_name]):
        # tensor = tensor.astype(str) ## if tensor != str
        tensor = " ".join(tensor)
        list_per_row = []
        for i in to_replace:
            tensor = tensor.lower()
            tensor = tensor.replace(i, "")
        tensor_list = tensor.split(",")
        list_per_row.extend(tensor_list)
        full_tensor_list.append(list_per_row)
    full_tensor_df = pd.DataFrame(full_tensor_list)

    return full_tensor_df
```

`# tensor = tensor.astype(str) ## if tensor != str` ⇒ I put this line of code just incase if each of the tensor's data type was not read in `str`. But its type was `str`, so I skipped activating it.

```
train_to_ml_fp_pr = dlc.tensor_separator(train_to_ml, 'first_party_berted')
train_to_ml_sp_pr = dlc.tensor_separator(train_to_ml, 'second_party_berted')
train_to_ml_facts_pr = dlc.tensor_separator(train_to_ml, 'facts_berted')

test_to_ml_fp_pr = test_to_ml_fp_pr.astype('float64')
test_to_ml_sp_pr = test_to_ml_sp_pr.astype('float64')
test_to_ml_facts_pr = test_to_ml_facts_pr.astype('float64')

to_be_X = pd.concat([train_to_ml_fp_pr, train_to_ml_facts_pr], axis=1)
to_be_test_x = pd.concat([test_to_ml_fp_pr, test_to_ml_facts_pr], axis=1)

to_be_X.columns = np.arange(len(to_be_X.columns))
```

```
to_be_X = pd.concat([to_be_X, train_to_ml['first_party_winner']], axis =1)
to_be_X
```

In [11]:
```
train_to_ml_fp_pr
```

Out[11]:

click to scroll output; double click to hide

	0	1	2	3	4	5	6	7	8	9	...	758	759	760	761	
0	03	03	02	7.3032e-03	-1.7571e-02	\n-2.9552e-02	-6.3181e-03	2.5047e-02	-9.3935e-03	-1.5124e-02	...	-1.4542e-02	-1.0145e-02	\n-1.7571e-02	-1.0507e-02	8
1	-3.9016e-03	2.6781e-02	-4.4628e-02	-3.2344e-02	-1.7295e-03	\n-1.8735e-02	2.1107e-03	-1.8207e-02	-9.9187e-03	-7.5807e-02	...	2.9344e-02	-1.4786e-03	\n-1.7004e-02	-9.6405e-03	1
2	3.0499e-03	9.2587e-03	-1.7064e-02	-1.9458e-02	-6.2003e-03	\n-8.5528e-03	5.0010e-02	3.5903e-02	-3.0393e-02	-2.0106e-02	...	-4.1979e-02	8.1220e-03	\n5.7186e-03	-3.7114e-02	-7
3	-1.5234e-02	-8.7536e-04	1.5897e-02	-1.8534e-02	5.4608e-02	\n-3.0493e-02	4.1842e-02	5.7857e-02	-1.3273e-02	-1.9490e-02	...	6.1164e-02	-2.3714e-02	\n-8.1452e-03	-5.4743e-02	-4
4	-1.1956e-02	-9.9873e-03	2.0198e-02	9.5077e-03	2.4592e-02	\n-3.7784e-02	2.0635e-02	-2.3835e-02	-2.9138e-02	-3.1670e-02	...	1.9298e-02	-5.3003e-04	\n4.8818e-03	-5.0081e-02	3
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
2473	-2.0591e-02	-5.6972e-02	-3.3879e-02	3.1382e-02	7.9985e-02	\n2.5763e-02	-6.5687e-03	4.4251e-02	-8.7967e-03	-6.0979e-02	...	-1.9980e-02	1.3039e-02	\n-4.5310e-02	-3.0027e-02	6
2474	-2.3678e-02	-2.2937e-02	-2.2214e-02	1.3166e-02	5.4840e-02	\n1.1287e-02	1.6339e-02	9.0016e-02	3.1381e-04	1.4739e-03	...	5.4709e-02	1.0245e-02	\n1.9659e-02	6.0086e-03	-3
2475	-4.2952e-03	-2.9256e-02	-1.5316e-02	-1.0641e-02	7.1409e-03	\n2.1234e-02	5.1800e-03	6.4935e-02	-2.2453e-02	-3.4121e-02	...	1.6813e-02	2.4257e-02	\n-9.5996e-03	-2.8923e-02	-8
2476	2.1019e-02	-3.1200e-03	-7.0313e-03	7.9619e-03	4.5658e-02	\n1.4659e-02	2.4269e-02	-2.1894e-02	-3.2928e-02	-2.9230e-02	...	3.2089e-02	-1.2351e-02	\n1.0574e-02	-2.2969e-02	-2
2477	-6.3555e-03	-1.1619e-02	-7.4391e-03	-1.2634e-02	-8.0101e-03	\n-5.9471e-03	3.2153e-02	-1.6714e-02	3.9917e-02	-4.6451e-02	...	5.6200e-02	-7.7531e-03	\n-5.5107e-04	-9.0997e-03	-7

2478 rows × 768 columns

Out[14]:

	0	1	2	3	4	5	6	7	8	9	...	1527	1528	1529	1530
0	0.005036	0.006439	-0.012888	-0.007303	-0.017571	-0.029552	-0.006318	0.025047	-0.009394	-0.015124	...	-0.018174	0.001737	0.003518	-0.010968
1	-0.003902	0.026781	-0.044628	-0.032344	-0.001729	-0.018735	0.002111	-0.018207	-0.009919	-0.075807	...	-0.010969	0.006232	-0.012417	-0.020754
2	0.003050	0.009259	-0.017064	-0.019458	-0.006200	-0.008553	0.050010	0.035903	-0.030393	-0.020106	...	-0.011412	-0.006155	-0.010546	-0.024606
3	-0.015234	-0.000875	0.015897	-0.018534	0.054608	-0.030493	0.041842	0.057857	-0.013273	-0.019490	...	-0.019637	-0.005936	-0.003335	-0.026111
4	-0.011956	-0.009987	0.020198	0.009508	0.024592	-0.037784	0.020635	-0.023835	-0.029138	-0.031670	...	-0.033741	0.007560	-0.023639	-0.028438
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2473	-0.020591	-0.056972	-0.033879	0.031382	0.079985	0.025763	-0.006569	0.044251	-0.008797	-0.060979	...	-0.027422	0.000112	0.015122	0.009355
2474	-0.023678	-0.022937	-0.022214	0.013166	0.054840	0.011287	0.016339	0.090016	0.000314	0.001474	...	-0.017667	0.015822	-0.002527	0.004892
2475	-0.004295	-0.029256	-0.015316	-0.010641	0.007141	0.021234	0.005180	0.064935	-0.022453	-0.034121	...	-0.011819	0.019265	-0.019696	-0.024496
2476	0.021019	-0.003120	-0.007031	0.007962	0.045658	0.014659	0.024269	-0.021894	-0.032928	-0.029230	...	-0.023799	0.002132	-0.011824	-0.020932
2477	-0.006356	-0.011619	-0.007439	-0.012634	-0.008010	-0.005947	0.032153	-0.016714	0.039917	-0.046451	...	-0.031149	-0.003120	0.002196	-0.010829

2478 rows × 1537 columns

After making these tensors into 2D, I could finally insert them into ML models!
Also, unlikely to my previous attempts on hyperparameter tuning with `for` loop, I have used `optuna`, which I just learned from the course, to find the optimal parameters.
Here is the output.

▼ **Codes**

```
X = to_be_X.drop(columns='first_party_winner')
y = pd.DataFrame(to_be_X['first_party_winner'])

X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.3, random_stat
e=42)

test_x = to_be_test_x

def lgb_objective(trial):
    lgb_params = {
        'application': 'binary',
        'max_depth': -1,
        'metric': 'binary_logloss',
        'boosting_type': trial.suggest_categorical('boosting_type', ['gbdt',  'dar
t']),
        'num_leaves': trial.suggest_int('num_leaves', 10, 2000),
        'lambda' : trial.suggest_float('lambda', 0.01, 0.5),
        'num_iteration': 500,
        'n_jobs': -1,
        'learning_rate': trial.suggest_float('learning_rate', 0.05, 0.1),
        'feature_fraction': trial.suggest_uniform('feature_fraction', 0.7, 0.9),
        'bagging_fraction': trial.suggest_uniform('bagging_fraction', 0.1, 0.8),
        'random_state': 42
    }

    lgb_model = lgb.LGBMClassifier(**lgb_params)
    lgb_model.fit(X_train, y_train)
    lgb_preds = lgb_model.predict(X_val)

    return accuracy_score(y_val, lgb_preds)

xgb_study = optuna.create_study(direction='minimize')
xgb_study.optimize(xgb_objective, show_progress_bar=True)

lgb_study = optuna.create_study(direction='minimize')
lgb_study.optimize(lgb_objective, n_trials=5, show_progress_bar=True)
```

```
[I 2023-06-10 16:41:38,918] A new study created in memory with name: no-name-fb52e4f8-268c-4753-bffa-93c2b9a21b53

[LightGBM] [Warning] lambda_l2 is set with reg_lambda=0.0, will be overridden by lambda=0.3902423070465261. Current v
alue: lambda_l2=0.3902423070465261
[LightGBM] [Warning] num_iterations is set=500, num_iteration=500 will be ignored. Current value: num_iterations=500
[LightGBM] [Warning] feature_fraction is set=0.8695116589311459, colsample_bytree=1.0 will be ignored. Current value:
feature_fraction=0.8695116589311459
[LightGBM] [Warning] bagging_fraction is set=0.19815622730347354, subsample=1.0 will be ignored. Current value: baggi
ng_fraction=0.19815622730347354

[I 2023-06-10 16:41:57,822] Trial 0 finished with value: 0.6545698924731183 and parameters: {'boosting_type': 'dart',
'num_leaves': 310, 'lambda': 0.3902423070465261, 'learning_rate': 0.08734927999126854, 'feature_fraction': 0.86951165
89311459, 'bagging_fraction': 0.19815622730347354}. Best is trial 0 with value: 0.6545698924731183.

[LightGBM] [Warning] lambda_l2 is set with reg_lambda=0.0, will be overridden by lambda=0.2050951401342081. Current v
alue: lambda_l2=0.2050951401342081
[LightGBM] [Warning] num_iterations is set=500, num_iteration=500 will be ignored. Current value: num_iterations=500
[LightGBM] [Warning] feature_fraction is set=0.8094065342219346, colsample_bytree=1.0 will be ignored. Current value:
feature_fraction=0.8094065342219346
[LightGBM] [Warning] bagging_fraction is set=0.38349619593424245, subsample=1.0 will be ignored. Current value: baggi
```

```
print('Number of finished LGB trials: {}'.format(len(lgb_study.trials)))
print('LGB Best trial:')
```

```
lgb_trial = lgb_study.best_trial

print('  Value: {}'.format(lgb_trial.value))
print('  Params: ')

for key, value in lgb_trial.params.items():
    print('    {}: {}'.format(key, value))
```

```
Number of finished LGB trials: 5
LGB Best trial:
  Value: 0.6559139784946236
  Params:
    boosting_type: gbdt
    num_leaves: 1611
    lambda: 0.29749127298451106
    learning_rate: 0.06593896755655602
    feature_fraction: 0.7542207041219193
    bagging_fraction: 0.7099047625020872
```

```
lgb_best_params = lgb_study.best_params
lgb_best_params['random_state'] = 42
lgb_model = lgb.LGBMClassifier(**lgb_best_params)
lgb_model.fit(X_train, y_train)
```

```
Number of finished LGB trials: 5
LGB Best trial:
  Value: 0.6559139784946236
  Params:
    boosting_type: gbdt
    num_leaves: 1611
    lambda: 0.29749127298451106
    learning_rate: 0.06593896755655602
    feature_fraction: 0.7542207041219193
    bagging_fraction: 0.7099047625020872
```

```
lgb_preds = lgb_model.predict(X_val)
lgb_accuracy = accuracy_score(y_val, lgb_preds)

print("-- LGB --")
print("Train ACC : %.3f" % accuracy_score(y_train, lgb_model.predict(X_train)))
print("Val ACC : %.3f" % accuracy_score(y_val, lgb_model.predict(X_val)))
```

```
-- LGB --
Train ACC : 1.000
Val ACC : 0.656
```

```
print(classification_report(y_val, lgb_preds))
```

```
              precision    recall  f1-score   support

           0       0.38      0.07      0.12       245
           1       0.67      0.94      0.79       499

    accuracy                           0.66       744
   macro avg       0.53      0.51      0.45       744
weighted avg       0.58      0.66      0.57       744
```

```
X_test = pd.get_dummies(data=test_x)
print(X_test)
lgb_preds = lgb_model.predict(test_x)
```

```
          0         1         2         3         4         5         6     \
0    -0.006672 -0.080011 -0.014580 -0.038840  0.048455 -0.040639  0.039663
1    -0.003053 -0.011769 -0.034803 -0.041324 -0.026498 -0.014561  0.012917
2    -0.025638 -0.016005 -0.021478  0.014413  0.048054 -0.036683 -0.034054
3     0.028991  0.036599 -0.059128 -0.030722 -0.017587 -0.028222  0.052064
4     0.036941 -0.039247 -0.026675 -0.006587 -0.022472 -0.016113  0.046348
...        ...       ...       ...       ...       ...       ...       ...
1235 -0.017606 -0.026507 -0.038503 -0.013017  0.036128 -0.006750 -0.023582
1236 -0.001220 -0.002364 -0.025906 -0.023937  0.002847 -0.021732  0.076602
1237  0.001026  0.023440  0.013377  0.016586  0.024960 -0.001395  0.063505
1238  0.001634 -0.000814 -0.014695  0.003300  0.001420  0.028769 -0.025222
1239 -0.058938  0.010668  0.004917 -0.051615  0.033669  0.037903  0.019593

          7         8         9     ...      1526      1527      1528  \
0     0.068547  0.002239 -0.029425  ... -0.012600 -0.023597 -0.003138
1    -0.009658 -0.000563 -0.018914  ... -0.018383 -0.027521  0.004810
2     0.043429 -0.017806 -0.045973  ... -0.023804 -0.017397  0.008190
3    -0.009327  0.006705 -0.008745  ... -0.021711 -0.015069 -0.001111
4    -0.012971  0.029547 -0.051456  ... -0.016862 -0.034918  0.005033
...        ...       ...       ...  ...       ...       ...       ...
1235  0.004234 -0.010011 -0.024067  ... -0.010572 -0.034447 -0.002972
1236 -0.014772 -0.012813 -0.014318  ... -0.000953 -0.013990  0.002872
1237 -0.017101 -0.032935 -0.016052  ... -0.018592 -0.016234  0.018842
1238  0.065449 -0.011581 -0.044858  ...  0.004752  0.003640  0.013376
1239  0.016298 -0.019234 -0.045782  ... -0.027754 -0.020528  0.022163

          1529      1530      1531      1532      1533      1534      1535
0    -0.008353 -0.034796 -0.020804 -0.006913 -0.038063  0.008657 -0.021999
1    -0.006279 -0.019910 -0.044318 -0.049594 -0.022413  0.012250  0.003738
2    -0.010989 -0.014011 -0.041264 -0.009615 -0.012293 -0.000732  0.010219
3    -0.014473 -0.025876 -0.006428 -0.018803 -0.028281 -0.004001 -0.012030
4     0.000040 -0.037341 -0.040531 -0.014745 -0.042890 -0.000467 -0.010147
...        ...       ...       ...       ...       ...       ...       ...
1235 -0.012302 -0.004949 -0.046846 -0.039191 -0.034175 -0.013973  0.006327
1236 -0.030309 -0.013520 -0.053392 -0.012986 -0.014506 -0.011636  0.024630
1237 -0.002355 -0.015415 -0.033917 -0.057270 -0.028927 -0.005387  0.000249
1238 -0.006307 -0.005089 -0.044533 -0.016824 -0.018459 -0.004400  0.022938
1239 -0.013797 -0.022734 -0.028132 -0.043650 -0.025505 -0.002197 -0.011266

[1240 rows x 1536 columns]
```

```
sample_submission = pd.read_csv('./sample_submission.csv')
LGB_pred = pd.DataFrame(lgb_preds.astype(int))
LGB_pred = LGB_pred.rename(columns={0:'first_party_winner'})
sample_submission['first_party_winner'] = LGB_pred['first_party_winner']

print(sample_submission['first_party_winner'].value_counts())

sample_submission.to_csv("./results/LGB_submission_{Train:.03f}_{Val:.03f}.csv".fo
rmat(Train=accuracy_score(y_train, lgb_model.predict(X_train)), Val = accuracy_sco
re(y_val, lgb_model.predict(X_val))), index=False)
```

```
1.0    697
0.0     47
Name: first_party_winner, dtype: int64
```

I have exported the final submission csv for 50+ times , but no score could exceed the result of my test_submission `0.5161`.

While feeling helpless and being lazy on tuning hyperparameter for each model, I came up with a crazy idea.

It was to run one `optuna` code to run all Classification models to get the best result.
Simultaneously to when I thought of this idea, I thought of what we learned during the course.
We learned that some parameters exists to regularize the model, give limits to the model so that it wouldn't go overfitted.

XGBoost for example, we were taught that when our model takes high `n_estimators`, it usually is better to have low `col_sample_bytree` / `col_sample_bylevel`.
Then I thought, why don't I simply create the instances for the parameters not to go beyond certain points.

Below is my attempt. Even though I haven't added the `cat_boost` yet, the `function` itself is working well.

▼ **Codes**

```
def model_selector(trial):
            model_type = trial.suggest_categorical('model_type', ['xgb', 'lgbm'])
            random_state = 42


            if model_type == 'xgb':
                eval_metric = 'error'
                objective = trial.suggest_categorical('objective', ['binary:logist
ic', 'binary_hinge'])
                tree_method = trial.suggest_categorical('tree_method', ['exact',
'approx', 'hist'])

                if tree_method == 'exact':
                    sampling_method = 'uniform'
                    subsample = 0.5
                    booster = trial.suggest_categorical('booster', ['dart', 'gbtre
e'])
                    if booster == 'gbtree':
                        max_depth = trial.suggest_int('max_depth', 1, 16)
                        n_estimators = trial.suggest_int('n_estimators', 1, 1500)
                        if n_estimators < 1000:
                            colsample_bytree = trial.suggest_loguniform('colsample
_bytree', 0.1, 0.2)
                            learning_rate = trial.suggest_categorical('learning_ra
```

```
te', [1e-4, 5e-4, 1e-3, 5e-3, 1e-2, 5e-2, 1e-1])  # aka eta, from 0 to 1, logunifo
rm
                            if learning_rate in [1e-4, 5e-4]:
                                min_split_loss = trial.suggest_loguniform('min_spl
it_loss', 0.1, 500)  # aka gamma, from 0 to 1, loguniform
                                min_child_weight = trial.suggest_int('min_child_we
ight', 1, 20)  # from 0 to infinite, int
                                alpha = trial.suggest_loguniform('alpha', 0.1, 20)
# from 0 to infinite, loguniform
                                reg_lambda = trial.suggest_loguniform('reg_lambd
a', 0.01, 20)  # from 0 to infinite, loguniform
                                model = xgb(
                                    eval_metric=eval_metric,
                                    objective=objective,
                                    learning_rate=learning_rate,
                                    sampling_method=sampling_method,
                                    subsample=subsample,
                                    colsample_bytree=colsample_bytree,
                                    max_depth=max_depth,
                                    tree_method=tree_method,
                                    booster=booster,
                                    min_split_loss=min_split_loss,
                                    min_child_weight=min_child_weight,
                                    alpha=alpha,
                                    reg_lambda=reg_lambda,
                                    random_state=random_state
                                )

                                model.fit(X_train, y_train)
                                preds = model.predict(X_val)

                                return accuracy_score(y_val, preds)

                            elif learning_rate in [1e-3, 5e-3, 1e-2]:
                                min_split_loss = trial.suggest_loguniform('min_spl
it_loss', 200,
                                                                          700)  #
aka gamma, from 0 to 1, loguniform
                                min_child_weight = trial.suggest_int('min_child_we
ight', 10, 30)  # from 0 to infinite, int
                                alpha = trial.suggest_loguniform('alpha', 10, 30)
# from 0 to infinite, loguniform
                                reg_lambda = trial.suggest_loguniform('reg_lambd
a', 10, 30)  # from 0 to infinite, loguniform
                                model = xgb(
                                    eval_metric=eval_metric,
                                    objective=objective,
                                    learning_rate=learning_rate,
                                    sampling_method=sampling_method,
                                    subsample=subsample,
                                    colsample_bytree=colsample_bytree,
                                    max_depth=max_depth,
                                    tree_method=tree_method,
                                    booster=booster,
                                    min_split_loss=min_split_loss,
                                    min_child_weight=min_child_weight,
```

```
                                    alpha=alpha,
                                    reg_lambda=reg_lambda,
                                    random_state=random_state
                                )

                                model.fit(X_train, y_train)
                                preds = model.predict(X_val)

                                return accuracy_score(y_val, preds)

                            elif learning_rate in [5e-2, 1e-1]:
                                min_split_loss = trial.suggest_loguniform('min_spl
it_loss', 500,
                                                                          1000)  #
aka gamma, from 0 to 1, loguniform
                                min_child_weight = trial.suggest_int('min_child_we
ight', 20, 40)  # from 0 to infinite, int
                                alpha = trial.suggest_loguniform('alpha', 20, 40)
# from 0 to infinite, loguniform
                                reg_lambda = trial.suggest_loguniform('reg_lambd
a', 20, 40)  # from 0 to infinite, loguniform
                                model = xgb(
                                    eval_metric=eval_metric,
                                    objective=objective,
                                    learning_rate=learning_rate,
                                    sampling_method=sampling_method,
                                    subsample=subsample,
                                    colsample_bytree=colsample_bytree,
                                    max_depth=max_depth,
                                    tree_method=tree_method,
                                    booster=booster,
                                    min_split_loss=min_split_loss,
                                    min_child_weight=min_child_weight,
                                    alpha=alpha,
                                    reg_lambda=reg_lambda,
                                    random_state=random_state
                                )
                                model.fit(X_train, y_train)
                                preds = model.predict(X_val)

                                return accuracy_score(y_val, preds)

                        elif n_estimators > 500:
                            colsample_bytree = trial.suggest_loguniform('colsample
_bytree', 0.1, 0.7)
                            learning_rate = trial.suggest_categorical('learning_ra
te', [1e-4, 5e-4, 1e-3, 5e-3, 1e-2, 5e-2,

1e-1])  # aka eta, from 0 to 1, loguniform
                            if learning_rate in [1e-4, 5e-4]:
                                min_split_loss = trial.suggest_loguniform('min_spl
it_loss', 0.1,
                                                                          500)  #
aka gamma, from 0 to 1, loguniform
                                min_child_weight = trial.suggest_int('min_child_we
ight', 1, 20)  # from 0 to infinite, int
```

```
                                alpha = trial.suggest_loguniform('alpha', 0.1, 20)
# from 0 to infinite, loguniform
                                reg_lambda = trial.suggest_loguniform('reg_lambd
a', 0.01, 20)  # from 0 to infinite, loguniform
                                model = xgb(
                                    eval_metric=eval_metric,
                                    objective=objective,
                                    learning_rate=learning_rate,
                                    sampling_method=sampling_method,
                                    subsample=subsample,
                                    colsample_bytree=colsample_bytree,
                                    max_depth=max_depth,
                                    tree_method=tree_method,
                                    booster=booster,
                                    min_split_loss=min_split_loss,
                                    min_child_weight=min_child_weight,
                                    alpha=alpha,
                                    reg_lambda=reg_lambda,
                                    random_state=random_state
                                )

                                model.fit(X_train, y_train)
                                preds = model.predict(X_val)

                                return accuracy_score(y_val, preds)

                            elif learning_rate in [1e-3, 5e-3, 1e-2]:
                                min_split_loss = trial.suggest_loguniform('min_spl
it_loss', 200,
                                                                          700)  #
aka gamma, from 0 to 1, loguniform
                                min_child_weight = trial.suggest_int('min_child_we
ight', 10, 30)  # from 0 to infinite, int
                                alpha = trial.suggest_loguniform('alpha', 10, 30)
# from 0 to infinite, loguniform
                                reg_lambda = trial.suggest_loguniform('reg_lambd
a', 10, 30)  # from 0 to infinite, loguniform
                                model = xgb(
                                    eval_metric=eval_metric,
                                    objective=objective,
                                    learning_rate=learning_rate,
                                    sampling_method=sampling_method,
                                    subsample=subsample,
                                    colsample_bytree=colsample_bytree,
                                    max_depth=max_depth,
                                    tree_method=tree_method,
                                    booster=booster,
                                    min_split_loss=min_split_loss,
                                    min_child_weight=min_child_weight,
                                    alpha=alpha,
                                    reg_lambda=reg_lambda,
                                    random_state=random_state
                                )

                                model.fit(X_train, y_train)
                                preds = model.predict(X_val)
```

```
                                return accuracy_score(y_val, preds)

                            elif learning_rate in [5e-2, 1e-1]:
                                min_split_loss = trial.suggest_loguniform('min_spl
it_loss', 500,
                                                                          1000)  #
aka gamma, from 0 to 1, loguniform
                                min_child_weight = trial.suggest_int('min_child_we
ight', 20, 40)  # from 0 to infinite, int
                                alpha = trial.suggest_loguniform('alpha', 20, 40)
# from 0 to infinite, loguniform
                                reg_lambda = trial.suggest_loguniform('reg_lambd
a', 20, 40)  # from 0 to infinite, loguniform
                                model = xgb(
                                    eval_metric=eval_metric,
                                    objective=objective,
                                    learning_rate=learning_rate,
                                    sampling_method=sampling_method,
                                    subsample=subsample,
                                    colsample_bytree=colsample_bytree,
                                    max_depth=max_depth,
                                    tree_method=tree_method,
                                    booster=booster,
                                    min_split_loss=min_split_loss,
                                    min_child_weight=min_child_weight,
                                    alpha=alpha,
                                    reg_lambda=reg_lambda,
                                    random_state=random_state
                                )

                                model.fit(X_train, y_train)
                                preds = model.predict(X_val)

                                return accuracy_score(y_val, preds)

                    elif booster == 'dart':
                        max_depth = trial.suggest_int('max_depth', 1, 300)
                        colsample_bytree = trial.suggest_loguniform('colsample_byt
ree', 0.5, 0.9)
                        learning_rate = trial.suggest_categorical('learning_rate',
[1e-4, 5e-4, 1e-3, 5e-3, 1e-2, 5e-2,

1e-1])  # aka eta, from 0 to 1, loguniform
                        if learning_rate in [1e-4, 5e-4]:
                            rate_drop = trial.suggest_loguniform('rate_drop', 0.0
1, 0.5)
                            min_child_weight = trial.suggest_int('min_child_weigh
t', 1, 20)  # from 0 to infinite, int
                            model = xgb(
                                eval_metric=eval_metric,
                                objective=objective,
                                learning_rate=learning_rate,
                                sampling_method=sampling_method,
                                subsample=subsample,
                                colsample_bytree=colsample_bytree,
```

```
                                rate_drop=rate_drop,
                                max_depth=max_depth,
                                tree_method=tree_method,
                                booster=booster,
                                min_child_weight=min_child_weight,
                                random_state=random_state
                            )

                            model.fit(X_train, y_train)
                            preds = model.predict(X_val)

                            return accuracy_score(y_val, preds)

                        elif learning_rate in [1e-3, 5e-3, 1e-2]:
                            rate_drop = trial.suggest_loguniform('rate_drop', 0.3,
0.7)
                            min_child_weight = trial.suggest_int('min_child_weigh
t', 10, 30)  # from 0 to infinite, int
                            model = xgb(
                                eval_metric=eval_metric,
                                objective=objective,
                                learning_rate=learning_rate,
                                sampling_method=sampling_method,
                                subsample=subsample,
                                colsample_bytree=colsample_bytree,
                                rate_drop=rate_drop,
                                max_depth=max_depth,
                                tree_method=tree_method,
                                booster=booster,
                                min_child_weight=min_child_weight,
                                random_state=random_state
                            )

                            model.fit(X_train, y_train)
                            preds = model.predict(X_val)

                            return accuracy_score(y_val, preds)

                        elif learning_rate in [5e-2, 1e-1]:
                            rate_drop = trial.suggest_loguniform('rate_drop', 0.5,
1.0)
                            min_child_weight = trial.suggest_int('min_child_weigh
t', 20, 40)  # from 0 to infinite, int
                            model = xgb(
                                eval_metric=eval_metric,
                                objective=objective,
                                learning_rate=learning_rate,
                                sampling_method=sampling_method,
                                subsample=subsample,
                                colsample_bytree=colsample_bytree,
                                rate_drop=rate_drop,
                                max_depth=max_depth,
                                tree_method=tree_method,
                                booster=booster,
                                min_child_weight=min_child_weight,
                                random_state=random_state
```

```
                            )

                            model.fit(X_train, y_train)
                            preds = model.predict(X_val)

                            return accuracy_score(y_val, preds)

                    # elif booster == 'gblinear':
                    #     learning_rate = trial.suggest_categorical('learning_rat
e', [1e-4, 5e-4, 1e-3, 5e-3, 1e-2, 5e-2,
                    #
1e-1])  # aka eta, from 0 to 1, loguniform
                    #     if learning_rate in [1e-4, 5e-4]:
                    #         alpha = trial.suggest_loguniform('alpha', 0.1, 20)
# from 0 to infinite, loguniform
                    #         reg_lambda = trial.suggest_loguniform('reg_lambda',
0.01, 20)  # from 0 to infinite, loguniform
                    #         model = xgb(
                    #             eval_metric=eval_metric,
                    #             objective=objective,
                    #             learning_rate=learning_rate,
                    #             sampling_method=sampling_method,
                    #             subsample=subsample,
                    #             tree_method=tree_method,
                    #             booster=booster,
                    #             alpha=alpha,
                    #             reg_lambda=reg_lambda,
                    #             random_state=random_state
                    #         )
                    #
                    #         model.fit(X_train, y_train)
                    #         preds = model.predict(X_val)
                    #
                    #         return accuracy_score(y_val, preds)
                    #
                    #     elif learning_rate in [1e-3, 5e-3, 1e-2]:
                    #         alpha = trial.suggest_loguniform('alpha', 10, 30)  #
from 0 to infinite, loguniform
                    #         reg_lambda = trial.suggest_loguniform('reg_lambda',
10, 30)  # from 0 to infinite, loguniform
                    #         model = xgb(
                    #             eval_metric=eval_metric,
                    #             objective=objective,
                    #             learning_rate=learning_rate,
                    #             sampling_method=sampling_method,
                    #             subsample=subsample,
                    #             tree_method=tree_method,
                    #             booster=booster,
                    #             alpha=alpha,
                    #             reg_lambda=reg_lambda,
                    #             random_state=random_state
                    #         )
                    #
                    #         model.fit(X_train, y_train)
                    #         preds = model.predict(X_val)
                    #
```

```
                    #         return accuracy_score(y_val, preds)
                    #
                    #     elif learning_rate in [5e-2, 1e-1]:
                    #         alpha = trial.suggest_loguniform('alpha', 20, 40)  #
from 0 to infinite, loguniform
                    #         reg_lambda = trial.suggest_loguniform('reg_lambda',
20, 40)  # from 0 to infinite, loguniform
                    #         model = xgb(
                    #             eval_metric=eval_metric,
                    #             objective=objective,
                    #             learning_rate=learning_rate,
                    #             sampling_method=sampling_method,
                    #             subsample=subsample,
                    #             tree_method=tree_method,
                    #             booster=booster,
                    #             alpha=alpha,
                    #             reg_lambda=reg_lambda,
                    #             random_state=random_state
                    #         )
                    #
                    #         model.fit(X_train, y_train)
                    #         preds = model.predict(X_val)
                    #
                    #         return accuracy_score(y_val, preds)

                else:
                    sampling_method = 'uniform'
                    booster = trial.suggest_categorical('booster', ['dart', 'gbtre
e'])
                    if booster == 'gbtree':
                        subsample = trial.suggest_loguniform('subsample', 0.1, 0.
5)
                        max_depth = 0
                        n_estimators = trial.suggest_int('n_estimators', 1, 1000)
                        if n_estimators < 500:
                            colsample_bytree = trial.suggest_loguniform('colsample
_bytree', 0.3, 0.9)
                            learning_rate = trial.suggest_categorical('learning_ra
te', [1e-4, 5e-4, 1e-3, 5e-3, 1e-2, 5e-2,

1e-1])  # aka eta, from 0 to 1, loguniform
                            if learning_rate in [1e-4, 5e-4]:
                                min_split_loss = trial.suggest_loguniform('min_spl
it_loss', 0.1,
                                                                          400)  #
aka gamma, from 0 to 1, loguniform
                                min_child_weight = trial.suggest_int('min_child_we
ight', 1, 20)  # from 0 to infinite, int
                                alpha = trial.suggest_loguniform('alpha', 0.1, 20)
# from 0 to infinite, loguniform
                                reg_lambda = trial.suggest_loguniform('reg_lambd
a', 0.01, 20)  # from 0 to infinite, loguniform
                                model = xgb(
                                    eval_metric=eval_metric,
                                    objective=objective,
                                    learning_rate=learning_rate,
```

```
                                    sampling_method=sampling_method,
                                    subsample=subsample,
                                    colsample_bytree=colsample_bytree,
                                    max_depth=max_depth,
                                    tree_method=tree_method,
                                    booster=booster,
                                    min_split_loss=min_split_loss,
                                    min_child_weight=min_child_weight,
                                    alpha=alpha,
                                    reg_lambda=reg_lambda,
                                    random_state=random_state
                                )

                                model.fit(X_train, y_train)
                                preds = model.predict(X_val)

                                return accuracy_score(y_val, preds)

                            elif learning_rate in [1e-3, 5e-3, 1e-2]:
                                min_split_loss = trial.suggest_loguniform('min_spl
it_loss', 200,
                                                                          700)  #
aka gamma, from 0 to 1, loguniform
                                min_child_weight = trial.suggest_int('min_child_we
ight', 10, 30)  # from 0 to infinite, int
                                alpha = trial.suggest_loguniform('alpha', 10, 30)
# from 0 to infinite, loguniform
                                reg_lambda = trial.suggest_loguniform('reg_lambd
a', 10, 30)  # from 0 to infinite, loguniform
                                model = xgb(
                                    eval_metric=eval_metric,
                                    objective=objective,
                                    learning_rate=learning_rate,
                                    sampling_method=sampling_method,
                                    subsample=subsample,
                                    colsample_bytree=colsample_bytree,
                                    max_depth=max_depth,
                                    tree_method=tree_method,
                                    booster=booster,
                                    min_split_loss=min_split_loss,
                                    min_child_weight=min_child_weight,
                                    alpha=alpha,
                                    reg_lambda=reg_lambda,
                                    random_state=random_state
                                )

                                model.fit(X_train, y_train)
                                preds = model.predict(X_val)

                                return accuracy_score(y_val, preds)

                            elif learning_rate in [5e-2, 1e-1]:
                                min_split_loss = trial.suggest_loguniform('min_spl
it_loss', 400,
                                                                          1000)  #
aka gamma, from 0 to 1, loguniform
```

```
                                min_child_weight = trial.suggest_int('min_child_we
ight', 20, 40)  # from 0 to infinite, int
                                alpha = trial.suggest_loguniform('alpha', 20, 40)
# from 0 to infinite, loguniform
                                reg_lambda = trial.suggest_loguniform('reg_lambd
a', 20, 40)  # from 0 to infinite, loguniform
                                model = xgb(
                                    eval_metric=eval_metric,
                                    objective=objective,
                                    learning_rate=learning_rate,
                                    sampling_method=sampling_method,
                                    subsample=subsample,
                                    colsample_bytree=colsample_bytree,
                                    max_depth=max_depth,
                                    tree_method=tree_method,
                                    booster=booster,
                                    min_split_loss=min_split_loss,
                                    min_child_weight=min_child_weight,
                                    alpha=alpha,
                                    reg_lambda=reg_lambda,
                                    random_state=random_state
                                )

                                model.fit(X_train, y_train)
                                preds = model.predict(X_val)

                                return accuracy_score(y_val, preds)

                        elif n_estimators > 500:
                            colsample_bytree = trial.suggest_loguniform('colsample
_bytree', 0.1, 0.7)
                            learning_rate = trial.suggest_categorical('learning_ra
te', [1e-4, 5e-4, 1e-3, 5e-3, 1e-2, 5e-2,

1e-1])  # aka eta, from 0 to 1, loguniform
                            if learning_rate in [1e-4, 5e-4]:
                                min_split_loss = trial.suggest_loguniform('min_spl
it_loss', 0.1,
                                                                          400)  #
aka gamma, from 0 to 1, loguniform
                                min_child_weight = trial.suggest_int('min_child_we
ight', 1, 20)  # from 0 to infinite, int
                                alpha = trial.suggest_loguniform('alpha', 0.1, 20)
# from 0 to infinite, loguniform
                                reg_lambda = trial.suggest_loguniform('reg_lambd
a', 0.01, 20)  # from 0 to infinite, loguniform
                                model = xgb(
                                    eval_metric=eval_metric,
                                    objective=objective,
                                    learning_rate=learning_rate,
                                    sampling_method=sampling_method,
                                    subsample=subsample,
                                    colsample_bytree=colsample_bytree,
                                    max_depth=max_depth,
                                    tree_method=tree_method,
                                    booster=booster,
```

```
                                    min_split_loss=min_split_loss,
                                    min_child_weight=min_child_weight,
                                    alpha=alpha,
                                    reg_lambda=reg_lambda,
                                    random_state=random_state
                                )

                                model.fit(X_train, y_train)
                                preds = model.predict(X_val)

                                return accuracy_score(y_val, preds)

                            elif learning_rate in [1e-3, 5e-3, 1e-2]:
                                min_split_loss = trial.suggest_loguniform('min_spl
it_loss', 200,
                                                                          700)  #
aka gamma, from 0 to 1, loguniform
                                min_child_weight = trial.suggest_int('min_child_we
ight', 10, 30)  # from 0 to infinite, int
                                alpha = trial.suggest_loguniform('alpha', 10, 30)
# from 0 to infinite, loguniform
                                reg_lambda = trial.suggest_loguniform('reg_lambd
a', 10, 30)  # from 0 to infinite, loguniform
                                model = xgb(
                                    eval_metric=eval_metric,
                                    objective=objective,
                                    learning_rate=learning_rate,
                                    sampling_method=sampling_method,
                                    subsample=subsample,
                                    colsample_bytree=colsample_bytree,
                                    max_depth=max_depth,
                                    tree_method=tree_method,
                                    booster=booster,
                                    min_split_loss=min_split_loss,
                                    min_child_weight=min_child_weight,
                                    alpha=alpha,
                                    reg_lambda=reg_lambda,
                                    random_state=random_state
                                )

                                model.fit(X_train, y_train)
                                preds = model.predict(X_val)

                                return accuracy_score(y_val, preds)

                            elif learning_rate in [5e-2, 1e-1]:
                                min_split_loss = trial.suggest_loguniform('min_spl
it_loss', 400,
                                                                          1000)  #
aka gamma, from 0 to 1, loguniform
                                min_child_weight = trial.suggest_int('min_child_we
ight', 20, 40)  # from 0 to infinite, int
                                alpha = trial.suggest_loguniform('alpha', 20, 40)
# from 0 to infinite, loguniform
                                reg_lambda = trial.suggest_loguniform('reg_lambd
a', 20, 40)  # from 0 to infinite, loguniform
```

```
                                model = xgb(
                                    eval_metric=eval_metric,
                                    objective=objective,
                                    learning_rate=learning_rate,
                                    sampling_method=sampling_method,
                                    subsample=subsample,
                                    colsample_bytree=colsample_bytree,
                                    max_depth=max_depth,
                                    tree_method=tree_method,
                                    booster=booster,
                                    min_split_loss=min_split_loss,
                                    min_child_weight=min_child_weight,
                                    alpha=alpha,
                                    reg_lambda=reg_lambda,
                                    random_state=random_state
                                )

                                model.fit(X_train, y_train)
                                preds = model.predict(X_val)

                                return accuracy_score(y_val, preds)

                    elif booster == 'dart':
                        subsample = 0.5
                        max_depth = 0
                        colsample_bytree = trial.suggest_loguniform('colsample_byt
ree', 0.5, 0.9)
                        learning_rate = trial.suggest_categorical('learning_rate',
[1e-4, 5e-4, 1e-3, 5e-3, 1e-2, 5e-2,

1e-1])  # aka eta, from 0 to 1, loguniform
                        if learning_rate in [1e-4, 5e-4]:
                            rate_drop = trial.suggest_loguniform('rate_drop', 0.0
1, 0.5)
                            min_child_weight = trial.suggest_int('min_child_weigh
t', 1, 20)  # from 0 to infinite, int
                            model = xgb(
                                eval_metric=eval_metric,
                                objective=objective,
                                learning_rate=learning_rate,
                                sampling_method=sampling_method,
                                subsample=subsample,
                                colsample_bytree=colsample_bytree,
                                max_depth=max_depth,
                                tree_method=tree_method,
                                booster=booster,
                                rate_drop=rate_drop,
                                min_child_weight=min_child_weight,
                                random_state=random_state
                            )

                            model.fit(X_train, y_train)
                            preds = model.predict(X_val)

                            return accuracy_score(y_val, preds)
```

```
                        elif learning_rate in [1e-3, 5e-3, 1e-2]:
                            rate_drop = trial.suggest_loguniform('rate_drop', 0.3,
0.7)
                            min_child_weight = trial.suggest_int('min_child_weigh
t', 10, 30)  # from 0 to infinite, int
                            model = xgb(
                                eval_metric=eval_metric,
                                objective=objective,
                                learning_rate=learning_rate,
                                sampling_method=sampling_method,
                                subsample=subsample,
                                colsample_bytree=colsample_bytree,
                                max_depth=max_depth,
                                tree_method=tree_method,
                                booster=booster,
                                rate_drop=rate_drop,
                                min_child_weight=min_child_weight,
                                random_state=random_state
                            )

                            model.fit(X_train, y_train)
                            preds = model.predict(X_val)

                            return accuracy_score(y_val, preds)

                        elif learning_rate in [5e-2, 1e-1]:
                            rate_drop = trial.suggest_loguniform('rate_drop', 0.5,
1.0)
                            min_child_weight = trial.suggest_int('min_child_weigh
t', 20, 40)  # from 0 to infinite, int
                            model = xgb(
                                eval_metric=eval_metric,
                                objective=objective,
                                learning_rate=learning_rate,
                                sampling_method=sampling_method,
                                subsample=subsample,
                                colsample_bytree=colsample_bytree,
                                max_depth=max_depth,
                                tree_method=tree_method,
                                booster=booster,
                                rate_drop=rate_drop,
                                min_child_weight=min_child_weight,
                                random_state=random_state
                            )

                            model.fit(X_train, y_train)
                            preds = model.predict(X_val)

                            return accuracy_score(y_val, preds)


                    # elif booster == 'gblinear':
                    #     learning_rate = trial.suggest_categorical('learning_rat
e', [1e-4, 5e-4, 1e-3, 5e-3, 1e-2, 5e-2,
                    #
```

```
1e-1])  # aka eta, from 0 to 1, loguniform
                    #     if learning_rate in [1e-4, 5e-4]:
                    #         alpha = trial.suggest_loguniform('alpha', 0.1, 20)
# from 0 to infinite, loguniform
                    #         reg_lambda = trial.suggest_loguniform('reg_lambda',
0.01, 20)  # from 0 to infinite, loguniform
                    #         model = xgb(
                    #             eval_metric=eval_metric,
                    #             objective=objective,
                    #             learning_rate=learning_rate,
                    #             sampling_method=sampling_method,
                    #             tree_method=tree_method,
                    #             booster=booster,
                    #             alpha=alpha,
                    #             reg_lambda=reg_lambda,
                    #             random_state=random_state
                    #         )
                    #         model.fit(X_train, y_train)
                    #         preds = model.predict(X_val)
                    #
                    #         return accuracy_score(y_val, preds)
                    #
                    #     elif learning_rate in [1e-3, 5e-3, 1e-2]:
                    #         alpha = trial.suggest_loguniform('alpha', 10, 30)  #
from 0 to infinite, loguniform
                    #         reg_lambda = trial.suggest_loguniform('reg_lambda',
10, 30)  # from 0 to infinite, loguniform
                    #         model = xgb(
                    #             eval_metric=eval_metric,
                    #             objective=objective,
                    #             learning_rate=learning_rate,
                    #             sampling_method=sampling_method,
                    #             tree_method=tree_method,
                    #             booster=booster,
                    #             alpha=alpha,
                    #             reg_lambda=reg_lambda,
                    #             random_state=random_state
                    #         )
                    #         model.fit(X_train, y_train)
                    #         preds = model.predict(X_val)
                    #
                    #         return accuracy_score(y_val, preds)
                    #
                    #     elif learning_rate in [5e-2, 1e-1]:
                    #         alpha = trial.suggest_loguniform('alpha', 20, 40)  #
from 0 to infinite, loguniform
                    #         reg_lambda = trial.suggest_loguniform('reg_lambda',
20, 40)  # from 0 to infinite, loguniform
                    #         model = xgb(
                    #             eval_metric=eval_metric,
                    #             objective=objective,
                    #             learning_rate=learning_rate,
                    #             sampling_method=sampling_method,
                    #             tree_method=tree_method,
                    #             booster=booster,
                    #             alpha=alpha,
```

```
                    #             reg_lambda=reg_lambda,
                    #             random_state=random_state
                    #         )
                    #         model.fit(X_train, y_train)
                    #         preds = model.predict(X_val)
                    #
                    #         return accuracy_score(y_val, preds)

            elif model_type == 'lgbm':
                objective = 'binary'
                metric = 'accuracy'
                num_threads = 0
                max_depth = -1

                num_leaves = trial.suggest_int('num_leaves', 1, 1000)
                boosting_type = trial.suggest_categorical('boosting_type', ['gbd
t', 'dart', 'rf', 'goss'])
                if boosting_type == 'gbdt':
                    n_estimators = trial.suggest_int('n_estimators', 1, 500)
                    learning_rate = trial.suggest_categorical('learning_rate', [1e
-4, 5e-4, 1e-3, 5e-3, 1e-2, 5e-2,
                                                                                1e
-1])  # aka eta, from 0 to 1, loguniform
                    if learning_rate in [1e-4, 5e-4]:
                        min_data_in_leaf = trial.suggest_int('min_data_in_leaf',
1, 20)
                        feature_fraction = trial.suggest_uniform('feature_fractio
n', 0.5, 0.999)
                        bagging_fraction = trial.suggest_uniform('bagging_fractio
n', 0.5, 0.999)
                        bagging_freq = trial.suggest_int('bagging_freq', 1, n_esti
mators)
                        alpha = trial.suggest_loguniform('alpha', 0.1, 20)  # from
0 to infinite, loguniform
                        reg_lambda = trial.suggest_loguniform('reg_lambda', 0.01,
20)  # from 0 to infinite, loguniform
                        model = lgb.LGBMClassifier(
                            learning_rate=learning_rate,
                            boosting_type=boosting_type,
                            num_leaves=num_leaves,
                            num_threads=num_threads,
                            objective=objective,
                            metric=metric,
                            n_estimators=n_estimators,
                            min_data_in_leaf=min_data_in_leaf,
                            feature_fraction=feature_fraction,
                            bagging_fraction=bagging_fraction,
                            bagging_freq=bagging_freq,
                            max_depth=max_depth,
                            alpha=alpha,
                            reg_lambda=reg_lambda,
                            random_state=random_state
                        )
                        model.fit(X_train, y_train)
                        preds = model.predict(X_val)
```

```
                        return accuracy_score(y_val, preds)

                    elif learning_rate in [1e-3, 5e-3, 1e-2]:
                        min_data_in_leaf = trial.suggest_int('min_data_in_leaf', 1
0, 30)
                        feature_fraction = trial.suggest_uniform('feature_fractio
n', 0.3, 0.7)
                        bagging_fraction = trial.suggest_uniform('bagging_fractio
n', 0.3, 0.7)
                        bagging_freq = trial.suggest_int('bagging_freq', 1, n_esti
mators)
                        alpha = trial.suggest_loguniform('alpha', 10, 30)  # from
0 to infinite, loguniform
                        reg_lambda = trial.suggest_loguniform('reg_lambda', 10, 3
0)  # from 0 to infinite, loguniform
                        model = lgb.LGBMClassifier(
                            learning_rate=learning_rate,
                            boosting_type=boosting_type,
                            num_leaves=num_leaves,
                            num_threads=num_threads,
                            objective=objective,
                            metric=metric,
                            n_estimators=n_estimators,
                            min_data_in_leaf=min_data_in_leaf,
                            feature_fraction=feature_fraction,
                            bagging_fraction=bagging_fraction,
                            bagging_freq=bagging_freq,
                            max_depth=max_depth,
                            alpha=alpha,
                            reg_lambda=reg_lambda,
                            random_state=random_state
                        )
                        model.fit(X_train, y_train)
                        preds = model.predict(X_val)

                        return accuracy_score(y_val, preds)

                    elif learning_rate in [5e-2, 1e-1]:
                        min_data_in_leaf = trial.suggest_int('min_data_in_leaf', 2
0, 40)
                        feature_fraction = trial.suggest_uniform('feature_fractio
n', 0.1, 0.5)
                        bagging_fraction = trial.suggest_uniform('bagging_fractio
n', 0.1, 0.5)
                        bagging_freq = trial.suggest_int('bagging_freq', 1, n_esti
mators)
                        alpha = trial.suggest_loguniform('alpha', 20, 40)  # from
0 to infinite, loguniform
                        reg_lambda = trial.suggest_loguniform('reg_lambda', 20, 4
0)  # from 0 to infinite, loguniform
                        model = lgb.LGBMClassifier(
                            learning_rate=learning_rate,
                            boosting_type=boosting_type,
                            num_leaves=num_leaves,
                            num_threads=num_threads,
                            objective=objective,
```

```
                            metric=metric,
                            n_estimators=n_estimators,
                            min_data_in_leaf=min_data_in_leaf,
                            feature_fraction=feature_fraction,
                            bagging_fraction=bagging_fraction,
                            bagging_freq=bagging_freq,
                            max_depth=max_depth,
                            alpha=alpha,
                            reg_lambda=reg_lambda,
                            random_state=random_state
                        )
                        model.fit(X_train, y_train)
                        preds = model.predict(X_val)

                        return accuracy_score(y_val, preds)

                elif boosting_type == 'dart':
                    n_estimators = trial.suggest_int('n_estimators', 1, 500)
                    learning_rate = trial.suggest_categorical('learning_rate', [1e
-4, 5e-4, 1e-3, 5e-3, 1e-2, 5e-2,
                                                                                  1e
-1])  # aka eta, from 0 to 1, loguniform
                    if learning_rate in [1e-4, 5e-4]:
                        min_data_in_leaf = trial.suggest_int('min_data_in_leaf',
1, 20)
                        feature_fraction = trial.suggest_uniform('feature_fractio
n', 0.5, 0.999)
                        bagging_fraction = trial.suggest_uniform('bagging_fractio
n', 0.5, 0.999)
                        bagging_freq = trial.suggest_int('bagging_freq', 1, n_esti
mators)
                        drop_rate = trial.suggest_loguniform('drop_rate', 0.01, 0.
5)
                        model = lgb.LGBMClassifier(
                            learning_rate=learning_rate,
                            boosting_type=boosting_type,
                            num_leaves=num_leaves,
                            num_threads=num_threads,
                            n_estimators=n_estimators,
                            objective=objective,
                            metric=metric,
                            min_data_in_leaf=min_data_in_leaf,
                            feature_fraction=feature_fraction,
                            bagging_fraction=bagging_fraction,
                            bagging_freq=bagging_freq,
                            max_depth=max_depth,
                            drop_rate=drop_rate,
                            random_state=random_state
                        )
                        model.fit(X_train, y_train)
                        preds = model.predict(X_val)

                        return accuracy_score(y_val, preds)

                    elif learning_rate in [1e-3, 5e-3, 1e-2]:
                        min_data_in_leaf = trial.suggest_int('min_data_in_leaf', 1
```

```
0, 30)
                        feature_fraction = trial.suggest_uniform('feature_fractio
n', 0.3, 0.7)
                        bagging_fraction = trial.suggest_uniform('bagging_fractio
n', 0.3, 0.7)
                        bagging_freq = trial.suggest_int('bagging_freq', 1, n_esti
mators)
                        drop_rate = trial.suggest_loguniform('drop_rate', 0.3, 0.
7)
                        model = lgb.LGBMClassifier(
                            learning_rate=learning_rate,
                            boosting_type=boosting_type,
                            num_leaves=num_leaves,
                            num_threads=num_threads,
                            n_estimators=n_estimators,
                            objective=objective,
                            metric=metric,
                            min_data_in_leaf=min_data_in_leaf,
                            feature_fraction=feature_fraction,
                            bagging_fraction=bagging_fraction,
                            bagging_freq=bagging_freq,
                            max_depth=max_depth,
                            drop_rate=drop_rate,
                            random_state=random_state
                        )
                        model.fit(X_train, y_train)
                        preds = model.predict(X_val)

                        return accuracy_score(y_val, preds)

                    elif learning_rate in [5e-2, 1e-1]:
                        min_data_in_leaf = trial.suggest_int('min_data_in_leaf', 2
0, 40)
                        feature_fraction = trial.suggest_uniform('feature_fractio
n', 0.5, 0.999)
                        bagging_fraction = trial.suggest_uniform('bagging_fractio
n', 0.5, 0.999)
                        bagging_freq = trial.suggest_int('bagging_freq', 1, n_esti
mators)
                        drop_rate = trial.suggest_loguniform('drop_rate', 0.5, 1)
                        model = lgb.LGBMClassifier(
                            learning_rate=learning_rate,
                            boosting_type=boosting_type,
                            num_leaves=num_leaves,
                            num_threads=num_threads,
                            n_estimators=n_estimators,
                            objective=objective,
                            metric=metric,
                            min_data_in_leaf=min_data_in_leaf,
                            feature_fraction=feature_fraction,
                            bagging_fraction=bagging_fraction,
                            bagging_freq=bagging_freq,
                            max_depth=max_depth,
                            drop_rate=drop_rate,
                            random_state=random_state
                        )
```

```
                        model.fit(X_train, y_train)
                        preds = model.predict(X_val)

                        return accuracy_score(y_val, preds)

                elif boosting_type == 'rf':
                    n_estimators = trial.suggest_int('n_estimators', 1, 500)
                    feature_fraction_bynode = trial.suggest_loguniform('feature_fr
action_bynode', 0.01, 0.999)
                    if feature_fraction_bynode < 0.33:
                        bagging_fraction = trial.suggest_uniform('bagging_fractio
n', 0.5, 0.999)
                        bagging_freq = trial.suggest_int('bagging_freq', 1, n_esti
mators)
                        min_data_in_leaf = trial.suggest_int('min_data_in_leaf',
0, 20)
                        alpha = trial.suggest_loguniform('alpha', 0.1, 20)  # from
0 to infinite, loguniform
                        reg_lambda = trial.suggest_loguniform('reg_lambda', 0.01,
20)  # from 0 to infinite, loguniform
                        model = lgb.LGBMClassifier(
                            boosting_type=boosting_type,
                            num_leaves=num_leaves,
                            num_threads=num_threads,
                            feature_fraction_bynode=feature_fraction_bynode,
                            bagging_fraction=bagging_fraction,
                            bagging_freq=bagging_freq,
                            objective=objective,
                            metric=metric,
                            n_estimators=n_estimators,
                            min_data_in_leaf=min_data_in_leaf,
                            max_depth=max_depth,
                            alpha=alpha,
                            reg_lambda=reg_lambda,
                            random_state=random_state
                        )
                        model.fit(X_train, y_train)
                        preds = model.predict(X_val)

                        return accuracy_score(y_val, preds)

                    elif feature_fraction_bynode > 0.33 and feature_fraction_bynod
e < 0.66:
                        min_data_in_leaf = trial.suggest_int('min_data_in_leaf', 1
0, 30)
                        alpha = trial.suggest_loguniform('alpha', 10, 30)  # from
0 to infinite, loguniform
                        bagging_fraction = trial.suggest_uniform('bagging_fractio
n', 0.5, 0.999)
                        bagging_freq = trial.suggest_int('bagging_freq', 1, n_esti
mators)
                        reg_lambda = trial.suggest_loguniform('reg_lambda', 10, 3
0)  # from 0 to infinite, loguniform
                        model = lgb.LGBMClassifier(
                            boosting_type=boosting_type,
                            num_leaves=num_leaves,
```

```
                            num_threads=num_threads,
                            feature_fraction_bynode=feature_fraction_bynode,
                            bagging_fraction=bagging_fraction,
                            bagging_freq=bagging_freq,
                            objective=objective,
                            metric=metric,
                            n_estimators=n_estimators,
                            min_data_in_leaf=min_data_in_leaf,
                            max_depth=max_depth,
                            alpha=alpha,
                            reg_lambda=reg_lambda,
                            random_state=random_state
                        )

                        model.fit(X_train, y_train)
                        preds = model.predict(X_val)

                        return accuracy_score(y_val, preds)

                    elif feature_fraction_bynode > 0.66:
                        min_data_in_leaf = trial.suggest_int('min_data_in_leaf', 2
0, 40)
                        alpha = trial.suggest_loguniform('alpha', 20, 40)  # from
0 to infinite, loguniform
                        reg_lambda = trial.suggest_loguniform('reg_lambda', 20, 4
0)  # from 0 to infinite, loguniform
                        bagging_fraction = trial.suggest_uniform('bagging_fractio
n', 0.5, 0.999)
                        bagging_freq = trial.suggest_int('bagging_freq', 1, n_esti
mators)
                        model = lgb.LGBMClassifier(
                            boosting_type=boosting_type,
                            num_leaves=num_leaves,
                            num_threads=num_threads,
                            feature_fraction_bynode=feature_fraction_bynode,
                            objective=objective,
                            metric=metric,
                            bagging_freq=bagging_freq,
                            bagging_fraction=bagging_fraction,
                            n_estimators=n_estimators,
                            min_data_in_leaf=min_data_in_leaf,
                            max_depth=max_depth,
                            alpha=alpha,
                            reg_lambda=reg_lambda,
                            random_state=random_state
                        )

                        model.fit(X_train, y_train)
                        preds = model.predict(X_val)

                        return accuracy_score(y_val, preds)

                elif boosting_type == 'goss':
                    learning_rate = trial.suggest_categorical('learning_rate', [1e
-4, 5e-4, 1e-3, 5e-3, 1e-2, 5e-2,
                                                                                1e
```

```
-1])  # aka eta, from 0 to 1, loguniform
                    if learning_rate in [1e-4, 5e-4]:
                        min_data_in_leaf = trial.suggest_int('min_data_in_leaf',
0, 20)
                        top_rate = trial.suggest_float('top_rate', 0.1, 0.5)
                        model = lgb.LGBMClassifier(
                            learning_rate=learning_rate,
                            boosting_type=boosting_type,
                            num_leaves=num_leaves,
                            num_threads=num_threads,
                            objective=objective,
                            metric=metric,
                            top_rate=top_rate,
                            min_data_in_leaf=min_data_in_leaf,
                            max_depth=max_depth,
                            random_state=random_state
                        )
                        model.fit(X_train, y_train)
                        preds = model.predict(X_val)

                        return accuracy_score(y_val, preds)

                    elif learning_rate in [1e-3, 5e-3, 1e-2]:
                        min_data_in_leaf = trial.suggest_int('min_data_in_leaf', 1
0, 30)
                        top_rate = trial.suggest_float('top_rate', 0.3, 0.7)
                        model = lgb.LGBMClassifier(
                            learning_rate=learning_rate,
                            boosting_type=boosting_type,
                            num_leaves=num_leaves,
                            num_threads=num_threads,
                            objective=objective,
                            metric=metric,
                            top_rate=top_rate,
                            min_data_in_leaf=min_data_in_leaf,
                            max_depth=max_depth,
                            random_state=random_state
                        )
                        model.fit(X_train, y_train)
                        preds = model.predict(X_val)

                        return accuracy_score(y_val, preds)

                    elif learning_rate in [5e-2, 1e-1]:
                        min_data_in_leaf = trial.suggest_int('min_data_in_leaf',
2, 40)
                        top_rate = trial.suggest_float('top_rate', 0.5, 0.9)
                        model = lgb.LGBMClassifier(
                            learning_rate=learning_rate,
                            boosting_type=boosting_type,
                            num_leaves=num_leaves,
                            num_threads=num_threads,
                            objective=objective,
                            top_rate=top_rate,
                            min_data_in_leaf=min_data_in_leaf,
                            max_depth=max_depth,
```

```
                            random_state=random_state
                        )

                        model.fit(X_train, y_train)
                        preds = model.predict(X_val)

                        return accuracy_score(y_val, preds)
```

The limitation on this is that I am still not so clear about each parameter, and what their functions are.

I have studied about how `optuna` works, and how each parameters work on training. However, I realized that
`optuna` usually only helps a bit on improving the model performance; it wouldn't boost up the model performance.
As per many previous attempts of other people,
`optuna` increase the score for only 0.001 to 0.01.

> What impacts more on the model performance is how the dataset is preprocessed.

Therefore, I tried to give each party name `mask` for `bert`.
As there are many variances use cases of names in
`facts` column, I separated the people's name into Family and First name.
Also, as some parties are governments, I had to give them a separate
`mask`.

▼ **Codes**

```
def auto_tokenizer(df, column_name):
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    bert_model = 'nlpaueb/bert-base-uncased-contracts'
    tokenizer = AutoTokenizer.from_pretrained(bert_model)
    model = AutoModelForTokenClassification.from_pretrained(bert_model)
    model = model.to(device)
    nlp = pipeline('ner', model=model, tokenizer=tokenizer, device=0)

    ei_total_list = []
    encoded_input_list = []
    for text in tqdm(df[column_name]):
        text = text.lower()
        entities = nlp(text)
```

```
        party_names = {}
        for entity in entities:
            if 'entity_group' in entity and entity['entity_group'] == 'LABEL_1':
                if 'word' in entity:
                    party = entity['word']
                    if party not in party_names:
                        party_names[party] = {'first_name': '', 'family_name': ''}
                        names = re.findall(r'\b\w+\b', party)
                        if len(names) == 2:
                            party_names[party]['first_name'] = names[0]
                            party_names[party]['family_name'] = names[1]
                        elif len(names) == 1:
                            party_names[party]['first_name'] = names[0]
            else:
                if 'party' in entity:
                    party = entity['party']
                    if party not in party_names:
                        party_names[party] = {'first_name': '', 'family_name': ''}
                    if 'first_name' in entity:
                        party_names['party']['first_name'] = entity['first_name']
                    if 'family_name' in entity:
                        party_names[party]['family_name'] = entity['family_name']

        list_of_states = [
            'wyoming', 'wisconsin', 'west virginia', 'washington', 'virginia',
            'vermont', 'utah', 'texas', 'tennessee', 'south dakota',
            'south carolina', 'rhode island', 'pennsylvania', 'oregon', 'oklahom
a',
            'ohio', 'north dakota', 'north carolina', 'new york', 'new mexico',
            'new jersey', 'new hampshire', 'nevada', 'nebraska', 'montana',
            'missouri', 'mississippi', 'minnesota', 'michigan', 'massachusetts',
            'maryland', 'maine', 'louisiana', 'kentucky', 'kansas',
            'iowa', 'indiana', 'illinois', 'idaho', 'hawaii',
            'georgia', 'florida', 'delaware', 'connecticut', 'colorado',
            'california', 'arkansas', 'arizona', 'alaska', 'alabama'

        ]

        list_of_usa = ['usa', 'america', 'u.s.', 'united states', 'the states', 't
he us', 'the united states',
                       'the united states of america', 'the u.s.', 'the usa']

        masked_text = text
        for party, names in party_names.items():
            first_name = names['first_name']
            family_name = names['family_name']

            if first_name in list_of_states:
                first_name = '[MASK]'

            if family_name in list_of_states:
                family_name = '[MASK]'

            if first_name in list_of_usa:
                first_name = '[MASK]'
            if family_name in list_of_usa:
```

```
                family_name = '[MASK]'

            masked_text = masked_text.replace(first_name, '[MASK]')
            masked_text = masked_text.replace(family_name, '[MASK]')

        for state in list_of_states:
            masked_text = masked_text.replace(state, '[MASK]')

        for usa in list_of_usa:
            masked_text = masked_text.replace(usa, '[MASK]')

        encoded_input = tokenizer(masked_text, padding='max_length', max_length=51
2, truncation=True, return_tensors='pt')
        encoded_input = {key: value.to(device) for key, value in encoded_input.ite
ms()}
        encoded_input_list.append(encoded_input)

        for encoded_input in encoded_input_list:
            with torch.no_grad():
                model_output = model(**encoded_input)

            sentence_embeddings = mean_pooling(model_output, encoded_input['attent
ion_mask'])
            sentence_embeddings = F.normalize(sentence_embeddings, p=2, dim=1)
            ei_total_list.append(sentence_embeddings.squeeze().cpu().numpy())

    df_berted = np.array(ei_total_list)

    return df_berted
```

I tried to run this code on Google Colab and Kaggle notebook using `gpu`, it wasn't so successful and depleted all the limits.
So I could only try more on local environment without `gpu`.
However, as expected, estimated completion time was 999+ hours.

It was weird to see such estimation as the dataset is not that extremely huge.

> There should be something wrong with the code.

While searching for the not detected error in the code, I had to start another competition (`ICR - Identifying Age-Related Conditions`) as the course I am taking asked me to.
I will create another page for it. A lot to talk.

Anyways, while working on this new competition, I learned about `Autogluon`.
By using this
`Autogluon`, it tries all possible methods (including ML and DL) on the Train dataset and predicts on Test dataset.
So I decided to give it a try on this competition and not waste any submission chances.
I have done 3 tries with
`Autogluon`. (The length of each codes is surprisingly short)

One was with the raw Train and Test data.

▼ Autogluon Result

```
Warning: path already exists! This predictor may overwrite an existing predictor!
path="model"
Presets specified: ['best_quality']
Stack configuration (auto_stack=True): num_stack_levels=0, num_bag_folds=8, num_ba
g_sets=1
Beginning AutoGluon training ...
AutoGluon will save models to "model/"
AutoGluon Version:  0.8.0
Python Version:     3.10.9
Operating System:   Darwin
Platform Machine:   arm64
Platform Version:   Darwin Kernel Version 22.5.0: Mon Apr 24 20:53:44 PDT 2023; ro
ot:xnu-8796.121.2~5/RELEASE_ARM64_T8103
Disk Space Avail:   39.41 GB / 245.11 GB (16.1%)
Train Data Rows:    2478
Train Data Columns: 4
Label Column: first_party_winner
Preprocessing data ...
AutoGluon infers your prediction problem is: 'binary' (because only two unique lab
el-values observed).
  2 unique label values:  [1, 0]
  If 'binary' is not the correct problem_type, please manually specify the problem
_type parameter during predictor init (You may specify problem_type as one of: ['b
inary', 'multiclass', 'regression'])
Selected class <--> label mapping:  class 1 = 1, class 0 = 0
Using Feature Generators to preprocess the data ...
Fitting AutoMLPipelineFeatureGenerator...
  Available Memory:                    4150.98 MB
  Train Data (Original)  Memory Usage: 4.41 MB (0.1% of available memory)
  Inferring data type of each feature based on column values. Set feature_metadata
_in to manually specify special dtypes of the features.
  Stage 1 Generators:
    Fitting AsTypeFeatureGenerator...
  Stage 2 Generators:
    Fitting FillNaFeatureGenerator...
  Stage 3 Generators:
    Fitting CategoryFeatureGenerator...
      Fitting CategoryMemoryMinimizeFeatureGenerator...
```

```
    Fitting TextSpecialFeatureGenerator...
      Fitting BinnedFeatureGenerator...
      Fitting DropDuplicatesFeatureGenerator...
    Fitting TextNgramFeatureGenerator...
      Fitting CountVectorizer for text features: ['first_party', 'second_party',
'facts']
      CountVectorizer fit with vocabulary size = 2828
  Stage 4 Generators:
    Fitting DropUniqueFeatureGenerator...
  Stage 5 Generators:
    Fitting DropDuplicatesFeatureGenerator...
  Unused Original Features (Count: 1): ['ID']
    These features were not used to generate any of the output features. Add a fea
ture generator compatible with these features to utilize them.
    Features can also be unused if they carry very little information, such as bei
ng categorical but having almost entirely unique values or being duplicates of oth
er features.
    These features do not need to be present at inference time.
    ('object', []) : 1 | ['ID']
  Types of features in original data (raw dtype, special dtypes):
    ('object', ['text']) : 3 | ['first_party', 'second_party', 'facts']
  Types of features in processed data (raw dtype, special dtypes):
    ('category', ['text_as_category'])  :    2 | ['first_party', 'second_party']
    ('int', ['binned', 'text_special']) :   52 | ['first_party.char_count', 'first
_party.word_count', 'first_party.capital_ratio', 'first_party.lower_ratio', 'first
_party.digit_ratio', ...]
    ('int', ['text_ngram'])             : 2777 | ['__nlp__.000', '__nlp__.10', '__
nlp__.11', '__nlp__.12', '__nlp__.13', ...]
  7.7s = Fit runtime
  3 features in original data used to generate 2831 features in processed data.
  Train Data (Processed) Memory Usage: 13.9 MB (0.3% of available memory)
Data preprocessing and feature engineering runtime = 7.83s ...
AutoGluon will gauge predictive performance using evaluation metric: 'accuracy'
  To change this, specify the eval_metric parameter of Predictor()
User-specified model hyperparameters to be fit:
{
  'NN_TORCH': {},
  'GBM': [{'extra_trees': True, 'ag_args': {'name_suffix': 'XT'}}, {}, 'GBMLarg
e'],
  'CAT': {},
  'XGB': {},
  'FASTAI': {},
  'RF': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini', 'problem_types':
['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args': {'name_suffix': 'E
ntr', 'problem_types': ['binary', 'multiclass']}}, {'criterion': 'squared_error',
'ag_args': {'name_suffix': 'MSE', 'problem_types': ['regression', 'quantile']}}],
  'XT': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini', 'problem_types':
['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args': {'name_suffix': 'E
ntr', 'problem_types': ['binary', 'multiclass']}}, {'criterion': 'squared_error',
'ag_args': {'name_suffix': 'MSE', 'problem_types': ['regression', 'quantile']}}],
  'KNN': [{'weights': 'uniform', 'ag_args': {'name_suffix': 'Unif'}}, {'weights':
'distance', 'ag_args': {'name_suffix': 'Dist'}}],
}
Fitting 13 L1 models ...
Fitting model: KNeighborsUnif_BAG_L1 ...
  Warning: Potentially not enough memory to safely train model. Estimated to requi
```

```
re 0.101 GB out of 4.056 GB available memory (12.453%)... (20.000% of avail memory
is the max safe size)
  To avoid this warning, specify the model hyperparameter "ag.max_memory_usage_rat
io" to a larger value (currently 1.0, set to >=0.22 to avoid the warning)
    To set the same value for all models, do the following when calling predictor.
fit: `predictor.fit(..., ag_args_fit={"ag.max_memory_usage_ratio": VALUE})`
  0.6138   = Validation score   (accuracy)
  0.24s  = Training   runtime
  0.47s  = Validation runtime
Fitting model: KNeighborsDist_BAG_L1 ...
  Warning: Potentially not enough memory to safely train model. Estimated to requi
re 0.101 GB out of 4.091 GB available memory (12.345%)... (20.000% of avail memory
is the max safe size)
  To avoid this warning, specify the model hyperparameter "ag.max_memory_usage_rat
io" to a larger value (currently 1.0, set to >=0.21 to avoid the warning)
    To set the same value for all models, do the following when calling predictor.
fit: `predictor.fit(..., ag_args_fit={"ag.max_memory_usage_ratio": VALUE})`
  0.6134   = Validation score   (accuracy)
  0.24s  = Training   runtime
  0.32s  = Validation runtime
Fitting model: LightGBMXT_BAG_L1 ...
  Fitting 8 child models (S1F1 - S1F8) | Fitting with ParallelLocalFoldFittingStra
tegy
  0.6792   = Validation score   (accuracy)
  4.36s  = Training   runtime
  0.2s   = Validation runtime
Fitting model: LightGBM_BAG_L1 ...
  Fitting 8 child models (S1F1 - S1F8) | Fitting with ParallelLocalFoldFittingStra
tegy
  0.6764   = Validation score   (accuracy)
  4.32s  = Training   runtime
  0.13s  = Validation runtime
Fitting model: RandomForestGini_BAG_L1 ...
  0.6602   = Validation score   (accuracy)
  1.95s  = Training   runtime
  1.49s  = Validation runtime
Fitting model: RandomForestEntr_BAG_L1 ...
  0.6671   = Validation score   (accuracy)
  1.44s  = Training   runtime
  1.5s   = Validation runtime
Fitting model: CatBoost_BAG_L1 ...
  Fitting 8 child models (S1F1 - S1F8) | Fitting with ParallelLocalFoldFittingStra
tegy
  0.682  = Validation score   (accuracy)
  33.76s   = Training   runtime
  1.13s  = Validation runtime
Fitting model: ExtraTreesGini_BAG_L1 ...
  0.6578   = Validation score   (accuracy)
  2.03s  = Training   runtime
  1.52s  = Validation runtime
Fitting model: ExtraTreesEntr_BAG_L1 ...
  0.6618   = Validation score   (accuracy)
  2.08s  = Training   runtime
  1.57s  = Validation runtime
Fitting model: NeuralNetFastAI_BAG_L1 ...
  Fitting 8 child models (S1F1 - S1F8) | Fitting with ParallelLocalFoldFittingStra
```

```
tegy
  0.6457   = Validation score   (accuracy)
  6.04s  = Training   runtime
  0.11s  = Validation runtime
Fitting model: XGBoost_BAG_L1 ...
  Fitting 8 child models (S1F1 - S1F8) | Fitting with ParallelLocalFoldFittingStra
tegy
  0.678  = Validation score   (accuracy)
  30.11s   = Training   runtime
  0.1s   = Validation runtime
Fitting model: NeuralNetTorch_BAG_L1 ...
  Fitting 8 child models (S1F1 - S1F8) | Fitting with ParallelLocalFoldFittingStra
tegy
  0.6683   = Validation score   (accuracy)
  6.01s  = Training   runtime
  0.16s  = Validation runtime
Fitting model: LightGBMLarge_BAG_L1 ...
  Fitting 8 child models (S1F1 - S1F8) | Fitting with ParallelLocalFoldFittingStra
tegy
  0.6751   = Validation score   (accuracy)
  18.72s   = Training   runtime
  0.27s  = Validation runtime
Fitting model: WeightedEnsemble_L2 ...
  0.6828   = Validation score   (accuracy)
  0.56s  = Training   runtime
  0.0s   = Validation runtime
AutoGluon training complete, total runtime = 169.17s ... Best model: "WeightedEnse
mble_L2"
TabularPredictor saved. To load, use: predictor = TabularPredictor.load("model/")
```

The other one with Train and Test data with keywords `berted`.

▼ Autogluon Result

```
Warning: path already exists! This predictor may overwrite an existing predictor!
path="model"
Presets specified: ['best_quality']
Stack configuration (auto_stack=True): num_stack_levels=0, num_bag_folds=8, num_ba
g_sets=1
Beginning AutoGluon training ...
AutoGluon will save models to "model/"
AutoGluon Version:  0.8.0
Python Version:     3.10.9
Operating System:   Darwin
Platform Machine:   arm64
Platform Version:   Darwin Kernel Version 22.5.0: Mon Apr 24 20:53:44 PDT 2023; ro
ot:xnu-8796.121.2~5/RELEASE_ARM64_T8103
Disk Space Avail:   38.91 GB / 245.11 GB (15.9%)
Train Data Rows:    2478
Train Data Columns: 6
Label Column: first_party_winner
Preprocessing data ...
```

```
AutoGluon infers your prediction problem is: 'binary' (because only two unique lab
el-values observed).
  2 unique label values:  [1, 0]
  If 'binary' is not the correct problem_type, please manually specify the problem
_type parameter during predictor init (You may specify problem_type as one of: ['b
inary', 'multiclass', 'regression'])
Selected class <--> label mapping:  class 1 = 1, class 0 = 0
Using Feature Generators to preprocess the data ...
Fitting AutoMLPipelineFeatureGenerator...
  Available Memory:                    3660.45 MB
  Train Data (Original)  Memory Usage: 0.12 MB (0.0% of available memory)
  Inferring data type of each feature based on column values. Set feature_metadata
_in to manually specify special dtypes of the features.
  Stage 1 Generators:
    Fitting AsTypeFeatureGenerator...
  Stage 2 Generators:
    Fitting FillNaFeatureGenerator...
  Stage 3 Generators:
    Fitting IdentityFeatureGenerator...
  Stage 4 Generators:
    Fitting DropUniqueFeatureGenerator...
  Stage 5 Generators:
    Fitting DropDuplicatesFeatureGenerator...
  Types of features in original data (raw dtype, special dtypes):
    ('float', []) : 6 | ['0', '1', '2', '3', '4', ...]
  Types of features in processed data (raw dtype, special dtypes):
    ('float', []) : 6 | ['0', '1', '2', '3', '4', ...]
  0.0s = Fit runtime
  6 features in original data used to generate 6 features in processed data.
  Train Data (Processed) Memory Usage: 0.12 MB (0.0% of available memory)
Data preprocessing and feature engineering runtime = 0.03s ...
AutoGluon will gauge predictive performance using evaluation metric: 'accuracy'
  To change this, specify the eval_metric parameter of Predictor()
User-specified model hyperparameters to be fit:
{
  'NN_TORCH': {},
  'GBM': [{'extra_trees': True, 'ag_args': {'name_suffix': 'XT'}}, {}, 'GBMLarg
e'],
  'CAT': {},
  'XGB': {},
  'FASTAI': {},
  'RF': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini', 'problem_types':
['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args': {'name_suffix': 'E
ntr', 'problem_types': ['binary', 'multiclass']}}, {'criterion': 'squared_error',
'ag_args': {'name_suffix': 'MSE', 'problem_types': ['regression', 'quantile']}}],
  'XT': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini', 'problem_types':
['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args': {'name_suffix': 'E
ntr', 'problem_types': ['binary', 'multiclass']}}, {'criterion': 'squared_error',
'ag_args': {'name_suffix': 'MSE', 'problem_types': ['regression', 'quantile']}}],
  'KNN': [{'weights': 'uniform', 'ag_args': {'name_suffix': 'Unif'}}, {'weights':
'distance', 'ag_args': {'name_suffix': 'Dist'}}],
}
Fitting 13 L1 models ...
Fitting model: KNeighborsUnif_BAG_L1 ...
  0.5969   = Validation score   (accuracy)
  0.01s  = Training   runtime
```

```
  0.01s  = Validation runtime
Fitting model: KNeighborsDist_BAG_L1 ...
  0.5823   = Validation score   (accuracy)
  0.0s   = Training   runtime
  0.0s   = Validation runtime
Fitting model: LightGBMXT_BAG_L1 ...
  Fitting 8 child models (S1F1 - S1F8) | Fitting with ParallelLocalFoldFittingStra
tegy
  0.6687   = Validation score   (accuracy)
  0.52s  = Training   runtime
  0.01s  = Validation runtime
Fitting model: LightGBM_BAG_L1 ...
  Fitting 8 child models (S1F1 - S1F8) | Fitting with ParallelLocalFoldFittingStra
tegy
  0.6683   = Validation score   (accuracy)
  0.74s  = Training   runtime
  0.01s  = Validation runtime
Fitting model: RandomForestGini_BAG_L1 ...
  0.6065   = Validation score   (accuracy)
  0.41s  = Training   runtime
  0.08s  = Validation runtime
Fitting model: RandomForestEntr_BAG_L1 ...
  0.6174   = Validation score   (accuracy)
  0.55s  = Training   runtime
  0.12s  = Validation runtime
Fitting model: CatBoost_BAG_L1 ...
  Fitting 8 child models (S1F1 - S1F8) | Fitting with ParallelLocalFoldFittingStra
tegy
  0.6691   = Validation score   (accuracy)
  1.76s  = Training   runtime
  0.0s   = Validation runtime
Fitting model: ExtraTreesGini_BAG_L1 ...
  0.6186   = Validation score   (accuracy)
  0.46s  = Training   runtime
  0.09s  = Validation runtime
Fitting model: ExtraTreesEntr_BAG_L1 ...
  0.6239   = Validation score   (accuracy)
  0.25s  = Training   runtime
  0.09s  = Validation runtime
Fitting model: NeuralNetFastAI_BAG_L1 ...
  Fitting 8 child models (S1F1 - S1F8) | Fitting with ParallelLocalFoldFittingStra
tegy
  0.6667   = Validation score   (accuracy)
  3.64s  = Training   runtime
  0.07s  = Validation runtime
Fitting model: XGBoost_BAG_L1 ...
  Fitting 8 child models (S1F1 - S1F8) | Fitting with ParallelLocalFoldFittingStra
tegy
  0.6598   = Validation score   (accuracy)
  0.82s  = Training   runtime
  0.01s  = Validation runtime
Fitting model: NeuralNetTorch_BAG_L1 ...
  Fitting 8 child models (S1F1 - S1F8) | Fitting with ParallelLocalFoldFittingStra
tegy
  0.6663   = Validation score   (accuracy)
  2.92s  = Training   runtime
```

```
  0.04s  = Validation runtime
Fitting model: LightGBMLarge_BAG_L1 ...
  Fitting 8 child models (S1F1 - S1F8) | Fitting with ParallelLocalFoldFittingStra
tegy
  0.6683   = Validation score   (accuracy)
  1.47s  = Training   runtime
  0.01s  = Validation runtime
Fitting model: WeightedEnsemble_L2 ...
  0.6691   = Validation score   (accuracy)
  0.97s  = Training   runtime
  0.0s   = Validation runtime
AutoGluon training complete, total runtime = 43.61s ... Best model: "WeightedEnsem
ble_L2"
TabularPredictor saved. To load, use: predictor = TabularPredictor.load("model/")
```

The final one with the whole datasets `berted`.

▼ Autogluon Result

```
Warning: path already exists! This predictor may overwrite an existing predictor!
path="model"
Presets specified: ['best_quality']
Stack configuration (auto_stack=True): num_stack_levels=0, num_bag_folds=8, num_ba
g_sets=1
Beginning AutoGluon training ...
AutoGluon will save models to "model/"
AutoGluon Version:  0.8.0
Python Version:     3.10.9
Operating System:   Darwin
Platform Machine:   arm64
Platform Version:   Darwin Kernel Version 22.5.0: Mon Apr 24 20:53:44 PDT 2023; ro
ot:xnu-8796.121.2~5/RELEASE_ARM64_T8103
Disk Space Avail:   38.62 GB / 245.11 GB (15.8%)
Train Data Rows:    2478
Train Data Columns: 2304
Label Column: first_party_winner
Preprocessing data ...
AutoGluon infers your prediction problem is: 'binary' (because only two unique lab
el-values observed).
  2 unique label values:  [1, 0]
  If 'binary' is not the correct problem_type, please manually specify the problem
_type parameter during predictor init (You may specify problem_type as one of: ['b
inary', 'multiclass', 'regression'])
Selected class <--> label mapping:  class 1 = 1, class 0 = 0
Using Feature Generators to preprocess the data ...
Fitting AutoMLPipelineFeatureGenerator...
  Available Memory:                    4062.64 MB
  Train Data (Original)  Memory Usage: 45.67 MB (1.1% of available memory)
  Inferring data type of each feature based on column values. Set feature_metadata
_in to manually specify special dtypes of the features.
  Stage 1 Generators:
    Fitting AsTypeFeatureGenerator...
```

```
  Stage 2 Generators:
    Fitting FillNaFeatureGenerator...
  Stage 3 Generators:
    Fitting IdentityFeatureGenerator...
  Stage 4 Generators:
    Fitting DropUniqueFeatureGenerator...
  Stage 5 Generators:
    Fitting DropDuplicatesFeatureGenerator...
  Types of features in original data (raw dtype, special dtypes):
    ('float', []) : 2304 | ['0', '1', '2', '3', '4', ...]
  Types of features in processed data (raw dtype, special dtypes):
    ('float', []) : 2304 | ['0', '1', '2', '3', '4', ...]
  7.8s = Fit runtime
  2304 features in original data used to generate 2304 features in processed data.
  Train Data (Processed) Memory Usage: 45.67 MB (1.1% of available memory)
Data preprocessing and feature engineering runtime = 7.9s ...
AutoGluon will gauge predictive performance using evaluation metric: 'accuracy'
  To change this, specify the eval_metric parameter of Predictor()
User-specified model hyperparameters to be fit:
{
  'NN_TORCH': {},
  'GBM': [{'extra_trees': True, 'ag_args': {'name_suffix': 'XT'}}, {}, 'GBMLarg
e'],
  'CAT': {},
  'XGB': {},
  'FASTAI': {},
  'RF': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini', 'problem_types':
['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args': {'name_suffix': 'E
ntr', 'problem_types': ['binary', 'multiclass']}}, {'criterion': 'squared_error',
'ag_args': {'name_suffix': 'MSE', 'problem_types': ['regression', 'quantile']}}],
  'XT': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini', 'problem_types':
['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args': {'name_suffix': 'E
ntr', 'problem_types': ['binary', 'multiclass']}}, {'criterion': 'squared_error',
'ag_args': {'name_suffix': 'MSE', 'problem_types': ['regression', 'quantile']}}],
  'KNN': [{'weights': 'uniform', 'ag_args': {'name_suffix': 'Unif'}}, {'weights':
'distance', 'ag_args': {'name_suffix': 'Dist'}}],
}
Fitting 13 L1 models ...
Fitting model: KNeighborsUnif_BAG_L1 ...
  0.6106   = Validation score   (accuracy)
  0.26s  = Training   runtime
  0.44s  = Validation runtime
Fitting model: KNeighborsDist_BAG_L1 ...
  0.6094   = Validation score   (accuracy)
  0.27s  = Training   runtime
  0.28s  = Validation runtime
Fitting model: LightGBMXT_BAG_L1 ...
  Fitting 8 child models (S1F1 - S1F8) | Fitting with ParallelLocalFoldFittingStra
tegy
  0.6776   = Validation score   (accuracy)
  98.71s   = Training   runtime
  0.17s  = Validation runtime
Fitting model: LightGBM_BAG_L1 ...
  Fitting 8 child models (S1F1 - S1F8) | Fitting with ParallelLocalFoldFittingStra
tegy
  0.6776   = Validation score   (accuracy)
```

```
  121.09s  = Training   runtime
  0.16s  = Validation runtime
Fitting model: RandomForestGini_BAG_L1 ...
  0.6578   = Validation score   (accuracy)
  4.8s   = Training   runtime
  1.2s   = Validation runtime
Fitting model: RandomForestEntr_BAG_L1 ...
  0.6622   = Validation score   (accuracy)
  4.63s  = Training   runtime
  1.17s  = Validation runtime
Fitting model: CatBoost_BAG_L1 ...
  Fitting 8 child models (S1F1 - S1F8) | Fitting with ParallelLocalFoldFittingStra
tegy
  0.6836   = Validation score   (accuracy)
  630.59s  = Training   runtime
  0.25s  = Validation runtime
Fitting model: ExtraTreesGini_BAG_L1 ...
  0.6638   = Validation score   (accuracy)
  1.69s  = Training   runtime
  1.29s  = Validation runtime
Fitting model: ExtraTreesEntr_BAG_L1 ...
  0.6659   = Validation score   (accuracy)
  1.2s   = Training   runtime
  1.22s  = Validation runtime
Fitting model: NeuralNetFastAI_BAG_L1 ...
  Memory not enough to fit NNFastAiTabularModel folds in parallel. Will do sequent
ial fitting instead.  Consider decreasing folds trained in parallel by passing num
_folds_parallel to ag_args_ensemble when calling predictor.fit
  Fitting 8 child models (S1F1 - S1F8) | Fitting with SequentialLocalFoldFittingSt
rategy
No improvement since epoch 7: early stopping
No improvement since epoch 7: early stopping
No improvement since epoch 6: early stopping
No improvement since epoch 4: early stopping
  0.6279   = Validation score   (accuracy)
  26.38s   = Training   runtime
  0.14s  = Validation runtime
Fitting model: XGBoost_BAG_L1 ...
  Fitting 8 child models (S1F1 - S1F8) | Fitting with ParallelLocalFoldFittingStra
tegy
  0.6751   = Validation score   (accuracy)
  212.65s  = Training   runtime
  0.41s  = Validation runtime
Fitting model: NeuralNetTorch_BAG_L1 ...
  Fitting 8 child models (S1F1 - S1F8) | Fitting with ParallelLocalFoldFittingStra
tegy
  0.6683   = Validation score   (accuracy)
  11.86s   = Training   runtime
  0.38s  = Validation runtime
Fitting model: LightGBMLarge_BAG_L1 ...
  Fitting 8 child models (S1F1 - S1F8) | Fitting with ParallelLocalFoldFittingStra
tegy
  0.6768   = Validation score   (accuracy)
  1157.88s   = Training   runtime
  0.28s  = Validation runtime
Fitting model: WeightedEnsemble_L2 ...
```

```
  0.6836   = Validation score   (accuracy)
  0.53s  = Training   runtime
  0.0s   = Validation runtime
AutoGluon training complete, total runtime = 2310.37s ... Best model: "WeightedEns
emble_L2"
TabularPredictor saved. To load, use: predictor = TabularPredictor.load("model/")
```

However the score for all these attempts only remained within 0.502 ~ 0.512 range. Which is not the best score.

▼ `dacon_law_class` module

```
# !pip install pandas
# !pip install torch
# !pip install tqdm
# !pip install pytorch_transformers
# !pip install transformers
# !pip install catboost
# ! pip install spacy


import numpy as np
import pandas as pd
import re
from datetime import date
import torch
import torch.nn.functional as F
from tqdm import tqdm
from transformers import AutoTokenizer, AutoModelForTokenClassification, pipeline
from xgboost import XGBClassifier as xgb
import lightgbm as lgb
import optuna
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report



# def text_processor(s):
#     """
#     문장을 담고있는 variable을 넣어주면
#     알파벳을 제외한 문장의 모든 기호, 숫자를 제거합니다.
#
#     :param s: 문장을 담고있는 variable
#     :return: 새로운 DataFrame안에 담긴 text_processor가 적용된 column
#     """
#
#     pattern = r'\([^)]*\)'  # ()
```

```
#     s = re.sub(pattern=pattern, repl='', string=s)
#     pattern = r'\[[^)]*\]'  # []
#     s = re.sub(pattern=pattern, repl='', string=s)
#     pattern = r'\<[^)]*\>'  # <>
#     s = re.sub(pattern=pattern, repl='', string=s)
#     pattern = r'\{[^)]*\}'  # {}
#     s = re.sub(pattern=pattern, repl='', string=s)
#
#
#     pattern = r'[^a-zA-Z]'
#     s = re.sub(pattern=pattern, repl=' ', string=s)
#
#     months = ['on january', 'on february', 'on march', 'on april', 'on may', 'on
june', 'on july', 'on august', 'on september', 'on october', 'on november', 'on de
cember', 'jan', 'feb', 'mar', 'apr', 'jun', 'jul', 'aug', 'sep', 'oct', 'nov', 'de
c']
#     for month in months:
#         s = s.lower()
#         s = s.replace(month, '')
#
#
#     units = ['mm', 'cm', 'km', 'ml', 'kg', 'g', 'th', 'st', 'rd', 'nd']
#     for unit in units:
#         s = s.lower()
#         s = s.replace(unit, '')
#
#     s_split = s.split()
#
#     s_list = []
#     for word in s_split:
#         if len(word) != 1:
#             s_list.append(word)
#
#     s_list = " ".join(s_list)
#
#     return s_list

def text_processor_2(s):
    """
    문장을 담고있는 variable을 넣어주면
    알파벳을 제외한 문장의 모든 기호, 숫자를 제거합니다.

    :param s: 문장을 담고있는 variable
    :return: 새로운 DataFrame안에 담긴 text_processor가 적용된 column
    """

    pattern = r'\([^)]*\)'  # ()
    s = re.sub(pattern=pattern, repl='', string=s)
    pattern = r'\[[^)]*\]'  # []
    s = re.sub(pattern=pattern, repl='', string=s)
    pattern = r'\<[^)]*\>'  # <>
    s = re.sub(pattern=pattern, repl='', string=s)
    pattern = r'\{[^)]*\}'  # {}
    s = re.sub(pattern=pattern, repl='', string=s)

    pattern = r'[^a-zA-Z0-9]'
```

```
    s = re.sub(pattern=pattern, repl=' ', string=s)

    useless = ['et al', 'and', 'inc']
    for word in useless:
        s = s.lower()
        s = s.replace(word, '')

    s_split = s.split()

    s_list = []
    for word in s_split:
        if len(word) != 1:
            s_list.append(word)

    s_list = " ".join(s_list)

    return s_list


def law_preprocessor(df, column):
    '''
    입력한 df의 column에서
    알파벳을 제외한 모든 숫자, 기호를 제거합니다.

    :param df: 대상이 될 DataFrame
    :param column: df에서 대상이 될 Column
    :return: 새로운 DataFrame안에 담긴 text_processor가 적용된 column
    '''
    temp = []
    for i in range(len(df)):
        temp.append(text_processor_2(df[f'{column}'][i]))

    temp_dict = {f"{column}": temp}

    processed = pd.DataFrame(temp_dict)
    return processed

# def alpha_only_3_cols(df, column1, column2, column3):
#     '''
#     입력한 df의 column 3개에서 알파벳을 제외한 모든 숫자, 기호를 제거합니다.
#
#     :param df: 대상이 될 DataFrame
#     :param column1: df에서 대상이 될 Column 1
#     :param column2: df에서 대상이 될 Column 2
#     :param column3: df에서 대상이 될 Column 3
#     :return: 새로운 DataFrame안에 담긴 text_processor가 적용된 column
#     '''
#
#     temp1 = []
#     temp2 = []
#     temp3 = []
#     for i in range(len(df)):
#         temp1.append(text_processor(df[f'{column1}'][i]))
#         temp2.append(text_processor(df[f'{column2}'][i]))
#         temp3.append(text_processor(df[f'{column3}'][i]))
#     temp = pd.DataFrame({f"{column1}": temp1, f'{column2}': temp2, f'{column3}':
```

```
temp3})
#     df[f"{column1}"] = temp[f"{column1}"]
#     df[f"{column2}"] = temp[f"{column2}"]
#     df[f"{column3}"] = temp[f"{column3}"]
#
#     return df

def alpha_numeric_3_cols(df, column1, column2, column3):
    '''
    입력한 df의 column 3개에서 알파벳을 제외한 모든 숫자, 기호를 제거합니다.

    :param df: 대상이 될 DataFrame
    :param column1: df에서 대상이 될 Column 1
    :param column2: df에서 대상이 될 Column 2
    :param column3: df에서 대상이 될 Column 3
    :return: 새로운 DataFrame안에 담긴 text_processor가 적용된 column
    '''

    temp1 = []
    temp2 = []
    temp3 = []
    for i in range(len(df)):
        temp1.append(text_processor_2(df[f'{column1}'][i]))
        temp2.append(text_processor_2(df[f'{column2}'][i]))
        temp3.append(text_processor_2(df[f'{column3}'][i]))
    temp = pd.DataFrame({f"{column1}": temp1, f'{column2}': temp2, f'{column3}': t
emp3})
    df[f"{column1}"] = temp[f"{column1}"]
    df[f"{column2}"] = temp[f"{column2}"]
    df[f"{column3}"] = temp[f"{column3}"]

    return df


def mean_pooling(model_output, attention_mask):
    '''
    하단 tokenizer를 위한 definition
    '''
    token_embeddings = model_output[0]
    input_mask_expanded = attention_mask.unsqueeze(-1).expand(token_embeddings.siz
e()).float()
    return torch.sum(token_embeddings * input_mask_expanded, 1) / torch.clamp(inpu
t_mask_expanded.sum(1), min=1e-9)


# device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
#
# tokenizer = AutoTokenizer.from_pretrained("nlpaueb/bert-base-uncased-contracts")
# model = AutoModelForTokenClassification.from_pretrained("nlpaueb/bert-base-uncas
ed-contracts").to(device)


def auto_tokenizer(df, column_name):
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    bert_model = 'nlpaueb/bert-base-uncased-contracts'
    tokenizer = AutoTokenizer.from_pretrained(bert_model)
```

```
    model = AutoModelForTokenClassification.from_pretrained(bert_model)
    model = model.to(device)
    nlp = pipeline('ner', model=model, tokenizer=tokenizer, device=0)

    ei_total_list = []
    encoded_input_list = []
    for text in tqdm(df[column_name]):
        text = text.lower()
        entities = nlp(text)

        party_names = {}
        for entity in entities:
            if 'entity_group' in entity and entity['entity_group'] == 'LABEL_1':
                if 'word' in entity:
                    party = entity['word']
                    if party not in party_names:
                        party_names[party] = {'first_name': '', 'family_name': ''}
                        names = re.findall(r'\b\w+\b', party)
                        if len(names) == 2:
                            party_names[party]['first_name'] = names[0]
                            party_names[party]['family_name'] = names[1]
                        elif len(names) == 1:
                            party_names[party]['first_name'] = names[0]
            else:
                if 'party' in entity:
                    party = entity['party']
                    if party not in party_names:
                        party_names[party] = {'first_name': '', 'family_name': ''}
                    if 'first_name' in entity:
                        party_names['party']['first_name'] = entity['first_name']
                    if 'family_name' in entity:
                        party_names[party]['family_name'] = entity['family_name']

        list_of_states = [
            'wyoming', 'wisconsin', 'west virginia', 'washington', 'virginia',
            'vermont', 'utah', 'texas', 'tennessee', 'south dakota',
            'south carolina', 'rhode island', 'pennsylvania', 'oregon', 'oklahom
a',
            'ohio', 'north dakota', 'north carolina', 'new york', 'new mexico',
            'new jersey', 'new hampshire', 'nevada', 'nebraska', 'montana',
            'missouri', 'mississippi', 'minnesota', 'michigan', 'massachusetts',
            'maryland', 'maine', 'louisiana', 'kentucky', 'kansas',
            'iowa', 'indiana', 'illinois', 'idaho', 'hawaii',
            'georgia', 'florida', 'delaware', 'connecticut', 'colorado',
            'california', 'arkansas', 'arizona', 'alaska', 'alabama'

        ]

        list_of_usa = ['usa', 'america', 'u.s.', 'united states', 'the states', 't
he us', 'the united states',
                       'the united states of america', 'the u.s.', 'the usa']

        masked_text = text
        for party, names in party_names.items():
            print(party, names)
            first_name = names['first_name']
```

```
            family_name = names['family_name']

            if first_name in list_of_states:
                first_name = '[MASK]'

            if family_name in list_of_states:
                family_name = '[MASK]'

            if first_name in list_of_usa:
                first_name = '[MASK]'
            if family_name in list_of_usa:
                family_name = '[MASK]'

            masked_text = masked_text.replace(first_name, '[MASK]')
            masked_text = masked_text.replace(family_name, '[MASK]')

        for state in list_of_states:
            masked_text = masked_text.replace(state, '[MASK]')

        for usa in list_of_usa:
            masked_text = masked_text.replace(usa, '[MASK]')

        encoded_input = tokenizer(masked_text, padding='max_length', max_length=51
2, truncation=True, return_tensors='pt')
        encoded_input = {key: value.to(device) for key, value in encoded_input.ite
ms()}
        encoded_input_list.append(encoded_input)

        for encoded_input in encoded_input_list:
            with torch.no_grad():
                model_output = model(**encoded_input)

            sentence_embeddings = mean_pooling(model_output, encoded_input['[atten
tion_mask]'])
            sentence_embeddings = F.normalize(sentence_embeddings, p=2, dim=1)
            ei_total_list.append(sentence_embeddings.squeeze().cpu().numpy())

    df_berted = np.array(ei_total_list)

    return df_berted

def analyze_correlations(tokenized_data):
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    device = torch.device(device)
    bert_model = 'nlpaueb/bert-base-uncased-contracts'
    tokenizer = AutoModelForTokenClassification.from_pretrained(bert_model)
    model = AutoModelForTokenClassification.from_pretrained(bert_model)
    model = model.to(device)

    tensor_data = torch.cat(tokenized_data, dim=0)  # Concatenate the list of tens
ors into a single tensor
    with torch.no_grad():
        outputs = model(tensor_data.to(device))
        attention_weights = outputs.attention_weights[-1]

    correlations = []
```

```
    for i, tensor in enumerate(tokenized_data):
        masked_attention_weights = attention_weights[i][tensor == tokenizer.mask_t
oken_id]
        correlations.append(masked_attention_weights.cpu().numpy())

    return correlations

def rename_tokenized(df_1, df_2, column_1, column_2, column_3):
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

    df_1_list = []
    df_2_list = []
    df_list = [df_1, df_2]
    column_list = [column_1, column_2, column_3]

    for df in df_list:
        for col in column_list:
            df_berted = auto_tokenizer(df, col)

            if isinstance(df_berted, np.ndarray):
                column_names = [f'{col}_berted_{i}' for i in range(df_berted.shape
[1])]
                df_berted = pd.DataFrame(df_berted, columns=column_names)

            tokenized_data = []
            for _, row in df_berted.iterrows():
                tensor = torch.tensor(row.values, device=device)
                tokenized_data.append(tensor.tolist())

            if df is df_1:
                df_1_list.extend([tokenized_data])
            elif df is df_2:
                df_2_list.extend([tokenized_data])

    df_1_df = pd.DataFrame(df_1_list, index=column_list)
    df_2_df = pd.DataFrame(df_2_list, index=column_list)

    df_1_df = df_1_df.T
    df_2_df = df_2_df.T

    return df_1_df, df_2_df


def token_to_df(df):
    outer_temp_df = pd.DataFrame()
    count = 0
    for column in df:
        inner_temp_df = pd.DataFrame()
        temp_list = []
        for value in tqdm(df[column]):
            value = value.replace('[', '').replace(']', '')
            value = value.split(',')
            temp_list.append(value)
        temp_df = pd.DataFrame(temp_list)
        inner_temp_df = pd.concat([inner_temp_df, temp_df], axis=1)
        inner_temp_df_col = [f'{df.columns[count]}_1', f'{df.columns[count]}_2']
```

```
        inner_temp_df.set_axis(inner_temp_df_col, axis=1, inplace=True)
        count += 1
        outer_temp_df = pd.concat([outer_temp_df, inner_temp_df], axis=1)
    return outer_temp_df


def tensor_2_2d(df, n):
    df_renamed = df.rename(columns={0: 'tbd', 1: 'hmm'})
    tensors = pd.DataFrame(df_renamed.groupby(by="tbd"))
    tensors1 = tensors[1]
    tensors1_df = pd.DataFrame(tensors1)
    tensors1_1 = pd.DataFrame(tensors1_df[1][n])
    target_name_temp = tensors1_1['tbd']
    target = tensors1_1['hmm']
    target_name_df = pd.DataFrame(target_name_temp)
    target_name = target_name_df.iat[0, 0]
    target_df = pd.DataFrame(target)
    target_df = target_df.reset_index()
    target_df = target_df.drop(columns='index')
    target_final_df = target_df.rename(columns={'hmm': target_name})

    temp = []
    for i in tqdm(range(len(target_final_df))):
        units = ['[', ']', 'tensor', '(', ')']

        for unit in units:
            s = str(target_final_df[target_name][i]).replace(unit, '')
        temp.append(s)

    temp_dict = {target_name: temp}

    final_df = pd.DataFrame(temp_dict)

    return final_df


def tensor_separator(df, column_name):
    to_replace = ["t e n s o r", "[", "]", "(", ")", " ", "n", "/"]
    full_tensor_list =[]
    for tensor in tqdm(df[column_name]):
        # tensor = tensor.astype(str) ## if tensor != str
        tensor = " ".join(tensor)
        list_per_row = []
        for i in to_replace:
            tensor = tensor.lower()
            tensor = tensor.replace(i, "")
        tensor_list = tensor.split(",")
        list_per_row.extend(tensor_list)
        full_tensor_list.append(list_per_row)
    full_tensor_df = pd.DataFrame(full_tensor_list)

    return full_tensor_df

def new_tensor_separator(df, column_name):
    to_replace = ["t e n s o r", "[", "]", "(", ")", " ", "n", "/", "tensor", "=",
"device", "d e v i c e", "cuda:0", "c u d a = 0",]
```

```
    full_tensor_list =[]
    for tensor in tqdm(df[column_name]):
        # tensor = tensor.astype(str) ## if tensor != str
        tensor = " ".join(tensor)
        list_per_row = []
        for i in to_replace:
            tensor = tensor.lower()
            tensor = tensor.replace(i, "")
        tensor_list = tensor.split(",")
        list_per_row.extend(tensor_list)
        full_tensor_list.append(list_per_row)
    full_tensor_df = pd.DataFrame(full_tensor_list)

    return full_tensor_df

def X2_T2(df1, df2, column):
    X_temp = pd.DataFrame()
    temp_train = df1.drop(columns=column)
    for i in temp_train:
        to_be_X = pd.concat([X_temp, tensor_separator(temp_train, i)], axis=1)

    to_be_X = (pd.concat([to_be_X, df1[column]], axis=1)).astype('float64')

    X_test_temp = pd.DataFrame()
    for i in df2:
        to_be_test_X = (pd.concat([X_test_temp, tensor_separator(df2, i)], axis=
1)).astype('float64')

    return to_be_X, to_be_test_X


def test_val_separator(df1, df2, test_size):
    train_cols = df1.columns.values.tolist()
    test_cols = df2.columns.values.tolist()
    column_y = [i for i in train_cols if i not in test_cols]
    column_X = [i for i in train_cols if i not in column_y]
    X = df1[df1.columns[df1.columns.isin(column_X)]]
    y = df1[df1.columns[df1.columns.isin(column_y)]]

    X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=test_size, r
andom_state=42)
    test_X = df2

    return X_train, X_val, y_train, y_val, test_X

class SimpleOps():
    '''
    간단한 pandas 표 자르고 넣기
    매번 치기 귀찮아서 만듦
    '''
    def __init__(self, df):
        self.df = df
    def __len__(self):
        return len(self.df)

    def __getitem__(self, idx):
```

```
        row = self.df.iloc[idx]
        column = self.df.iloc[:, idx]

    def right_merger(self, df1, column_idx):
        '''
        입력한 두개의 df를 right merge합니다.
        column_idx에는 기준이 되는 컬럼 index를 입력해주면 됩니다.
        '''
        merged = pd.merge(self, df1, how='right', on=self.columns[column_idx])

        return merged

    def left_merger(self, df1, column_idx):
        '''
        right merger 만들었는데 left는 안만들면 섭섭할까봐 만듦
        '''
        merged = pd.merge(self, df1, how='left', on=self.columns[column_idx])
        return merged

    def ccd(self, start_number, end_number):
        '''
        Continuous_Column_Dropper
        연속되는 column을 삭제합니다.

        :param start_number: 시작 column index
        :param end_number: 종료 column index
        :return:
        '''
        df = self.drop(self.columns[start_number:(end_number + 1)], axis=1)
        return df

    def ocd(self, colnum1):
        '''
        One_Column_Dropper
        한 개의 column을 삭제합니다.
        위에꺼 만들고 안만들면 섭섭해서 그냥 만들었습니다.

        :param colnum1: Column number you want to drop
        :return: df with dropped column
        '''
        df = self.drop(columns=[colnum1])
        return df

    def law_train_clean_ccd(self, df):
        df = pd.concat([self.iloc[:, 0], df], axis=1)
        temp = SimpleOps.ccd(self, 3, 4)
        temp = SimpleOps.right_merger(temp, df, 0)
        temptemp = SimpleOps.ccd(self, 1, 3)
        train_cleansed = SimpleOps.right_merger(temp, temptemp, 0)
        return train_cleansed

    def law_train_clean1(self, df, colnum1):
        df = pd.concat([self.iloc[:, 0], df], axis=1)
        temp = SimpleOps.ocd(self, colnum1)
        temp = SimpleOps.right_merger(temp, df, 0)
        temptemp = SimpleOps.ocd(self, colnum1)
```

```
        train_cleansed = SimpleOps.right_merger(temp, temptemp, 0)
        return train_cleansed

    def df_divider(self, column):
        if len(self) % 2 == 0:
            divided_df = np.array_split(self[column], 26)

        else:
            divided_df = np.array_split(self[column][:-1], 25)
            divided_df.apend(self[column][-1:])

        return divided_df[0], divided_df[1], divided_df[2], divided_df[3], divided
_df[4], divided_df[5], divided_df[6], divided_df[7] , divided_df[8] , divided_df
[9], divided_df[10], divided_df[11], divided_df[12], divided_df[13], divided_df[1
4], divided_df[15], divided_df[16], divided_df[17] , divided_df[18] , divided_df[1
9], divided_df[20], divided_df[21], divided_df[22], divided_df[23], divided_df[2
4], divided_df[25]

today = date.today()
date = today.strftime("%d")
month = today.strftime('%b')
year = today.strftime('%Y')

if date in [1, 21, 31]:
    suffix = 'st'
elif date in [2, 22]:
    suffix = 'nd'
elif date in [3, 23]:
    suffix = 'rd'
else:
    suffix = 'th'


print(
" _____\n"
"|                            |\n"
"|======== YearDream ========|\n"
"|============================|\n"
"|==== DLC Well Imported ====|\n"
"|============================|\n"
"|========= BYJASON =========|\n"
f"|_______{date}{suffix}_{month}_{year}_______|\n"
)
```