Group Report of Final Project

Group 1

Ruiqi Li, Yixi Liang, He Huang, Yuan Dang

Department of Data Science, The George Washington University

DATS6312: NLP for Data Science

Amir Jafari

Dec 11, 2022

Introduction

Our group project focuses on evaluating middle school students' writings. The original dataset is published on Kaggle by Georgia State University. The aim is to use the NLP techniques we have learned to present the EDA and predict the type of discourse elements for each text.

The discourse element is the label, and there are 7 discourse elements: lead, position, claim, counterclaim, rebuttal, evidence, and concluding statement. The information is given in Figure 1.

Figure 1

Categories of the discourse type.

- **Lead** - an introduction that begins with a statistic, a quotation, a description, or some other device to grab the reader's attention and point toward the thesis
- **Position** - an opinion or conclusion on the main question
- **Claim** - a claim that supports the position
- **Counterclaim** - a claim that refutes another claim or gives an opposing reason to the position
- **Rebuttal** - a claim that refutes a counterclaim
- **Evidence** - ideas or examples that support claims, counterclaims, or rebuttals.
- **Concluding Statement** - a concluding statement that restates the claims

For a general outline of shared work, we first handled data preprocessing and EDA, then we used the rule-based models of Logistic Regression and Naïve Bayes to make predictions. After rule-based models, we built and trained deep neural network models. We tried different transformers with heads of MLP, CNN, and LSTM to predict the labels. Finally, we generated the summary of each discourse text and then used that to predict the labels with the transformers using different heads.

Description of the dataset

The dataset contains argumentative essays written by U.S students in grades 6-12. Essays are automatically segmented into discrete discourse elements. The number of data rows in this dataset reaches 144,280. Figure 2 is an overview of the dataset sample.

Figure 2

Dataset Sample

	discourse_text	discourse_type	label
0	Modern humans today are always on their phone....	Lead	0
1	They are some really bad consequences when stu...	Position	2
2	Some certain areas in the United States ban ph...	Evidence	4
3	When people have phones, they know about certa...	Evidence	4
4	Driving is one of the way how to get around. P...	Claim	3
5	That's why there's a thing that's called no te...	Evidence	4

Data Preprocessing

We then preprocess the text data to do EDA by decapitalization, punctuation removal, stopwords removal, stemming, lemmatization, tokenization, etc. We also check if there is any missing data. Figure 3 is a report of no missing data.

Figure 3

Missing Value in each column

	missing value exists
id	False
discourse_id	False
discourse_start	False
discourse_end	False
discourse_text	False
discourse_type	False
discourse_type_num	False
predictionstring	False

EDA

We perform the EDA to better understand the dataset. Firstly, we make a table to list out the count of each discourse type with its label. In Figure 4, we can see that Claim has the most count while rebuttal has the least count.

Figure 4

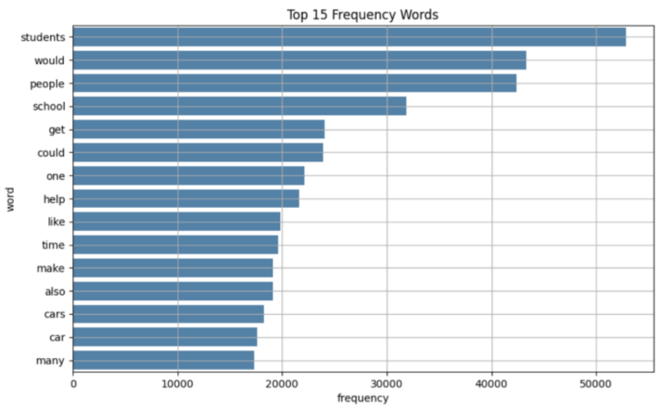
Number of Samples in Each Category

discourse_type	label	count
Claim	3	50204
Evidence	4	45702
Position	2	15417
Concluding Statement	6	13505
Lead	0	9305
Counterclaim	5	5817
Rebuttal	1	4334

We can find the top frequency words after performing tokenization. Total number of words in dataset is 57,414. The most frequent word is "students", then comes with nouns and objects like "people" and "school". Figure 6 is the plot.

Figure 6

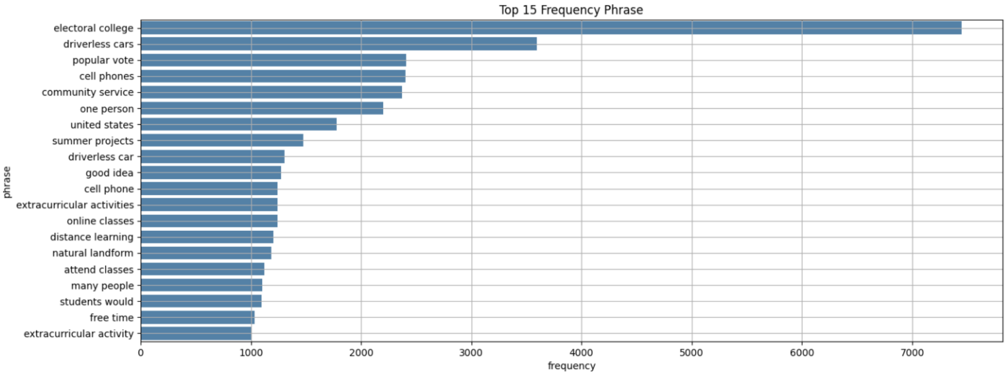
Most Frequent Words in Dataset Text



Besides the words, we also find the total number of phrases in dataset is 328,811, and the top frequency phrases are "electoral college", "driverless cars", etc. From the histrogram in Figure 7, it is obvious that "electoral college" has an extremely high count.

Figure 7

Most Frequent Phrase in Dataset Text

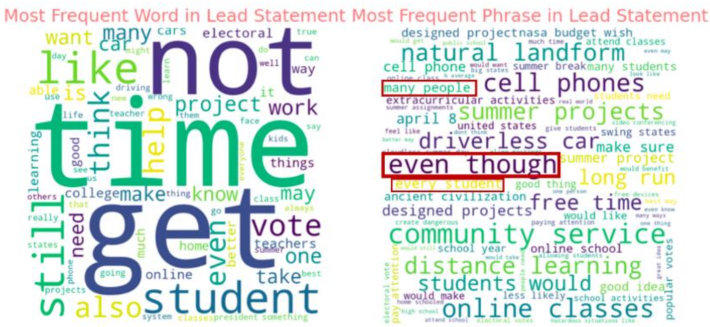


We also use Word Cloud to take a better view of word frequency in each discourse element category. The frequency of a word is considered as its importance and the importance of each word is shown with font size. In every category, the most used words are nouns that also appear in previous top frequency words among whole dataset.

In lead statement, nouns are used a lot, and some noun phrases such as "many people", 'every student", "many students" are common in beginning of paragraph. The most common conjunction is "even though" (Figure 8).

Figure 8

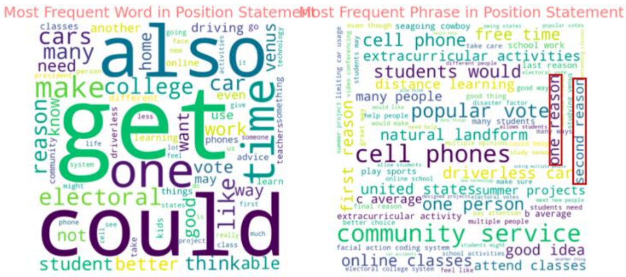
Text Lead Argumentative Type Word Cloud



Position is defined as an opinion on the main question. Some frequently used word phrases here are “one reason”, “second reason”, etc (Figure 9).

Figure 9

Position Argumentative Type Word Cloud



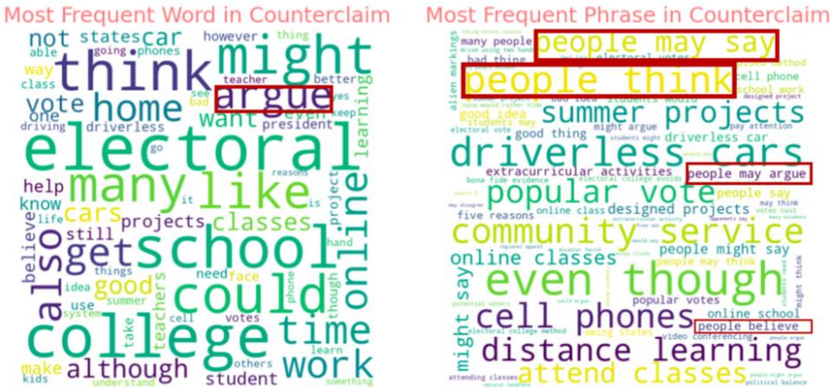
Claim supports the main opinion, and counterclaim always gives an opposite opinion. For most of the time, counterclaim starts with phrase “people may argue”. This word cloud shows an obvious attribute of counterclaim that is different than other elements (Figure 10 & Figure 11).

Figure 10

Claim Argumentative Type Word Cloud

Figure 11

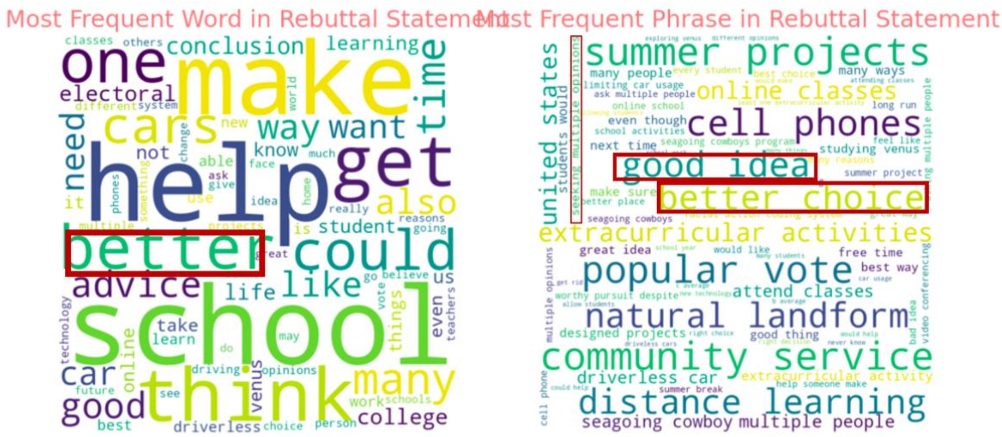
Counterclaim Argumentative Type Word Cloud



Rebuttal elements always follow the counterclaim and refute the counterclaim to support the author’s main opinion. The most common words in rebuttal statements are “better”, “better choice” (Figure 12).

Figure 12

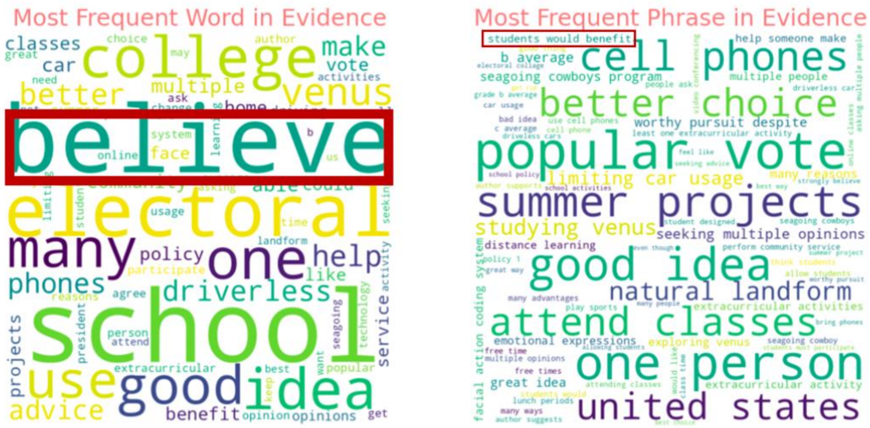
Rebuttal Argumentative Type Word Cloud



Evidence gives ideas or examples that support any claims, they often use words “believe”, “benefit”.

Figure 13

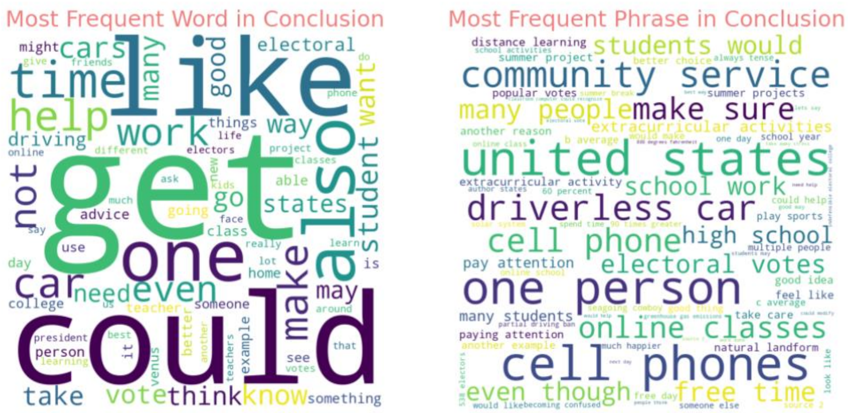
Evidence Argumentative Type Word Cloud



The concluding statement restates the claims, therefore, the bag of words here are very similar to it is for claim statement.

Figure 14

Conclusion Argumentative Type Word Cloud



The word clouds of the discourse types have helped us understand the data in a better way. Next step is to set up for the experiments.

Experimental Setup and Results

Rule based model

After the EDA, we begin the setup for the experiments, such as setting up the models. The first category is rule-based models.

Here is some context information about the Naïve Bayes and Logistic Regression. Naïve Bayes is a generative classifier while logistic regression is a discriminative classifier. The Naïve Bayes model calculates the probability of classes, which is based on Bayes Rule shown in Figure 15. The Bayes Rule calculates the posterior probability of an event given the probability of another event that has happened. A and B are events and P(B) must not be undefined. P(A|B) is posterior probability of A given B. P(A) is prior probability of A. P(B|A) is the likelihood probability of B given A. P(B) is the prior probability of B. The formula is shown in Fig 1.

Figure 15

Bayes Rule

$$P(A|B) = \frac{P(A)P(B|A)}{P(B)}$$

For the basic logistic regression, there is the input vector of [x1,x2…xn] and corresponding weight of [w1,w2…wn]. The output is binary. We calculate the sum of weighted features and bias. Figure 16 explains these two operations. Then input to function of z that ranges from 0 to 1, which is in sigmoid form (Figure 17). After that, calculate the probability and then classify it (Figure 18).

Figure 16

Logistic Operations

$$x = [x_1, x_1, ..., x_n]$$

$$W = [w_1, w_2, ..., w_n]$$

$$\hat{y} \in \{0, 1\}$$

$$Z = (\sum_{i=1}^{n} w_i x_i) + b$$

$$Z = w \cdot x + b$$

Figure 17

Logistic Plot

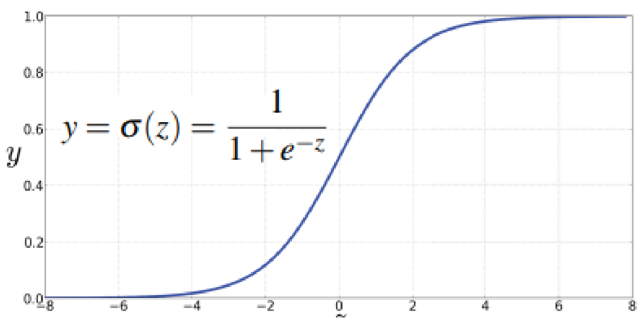


Figure 18

Further Operations of Logistic

$$P(y=1) = \sigma(w \cdot x + b) = \frac{1}{1 + exp(-(w \cdot x + b))}$$
$$P(y=0) = 1 - \sigma(w \cdot x + b) = 1 - \frac{1}{1 + exp(-(w \cdot x + b))}$$
$$P(y=0) = 1 - \sigma(w \cdot x + b) = \frac{exp(-(w \cdot x + b)}{1 + exp(-(w \cdot x + b))}$$

$$\hat{y} = 1 if P(y=1|x) > 0.5 otherwise 0$$

We also apply LSA to the data before logistic regression. In Figure 19 from the lecture, each word can be shown as a score of importance in each row, which is an array. These arrays of numbers are singular values.

Figure 19

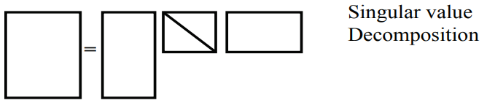
LSA Table

	algorithms	computers	data	energy	family	food	fun	games	health	home	java	kids	learning	love	machine	money	programming	science	structures
0	0.00	0.00	0.36	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.54	0.00	0.54	0.00	0.00	0.54	0.00
1	0.00	0.00	0.00	0.00	0.46	0.00	0.37	0.00	0.00	0.46	0.00	0.46	0.00	0.00	0.00	0.46	0.00	0.00	0.00
2	0.00	0.00	0.36	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.54	0.00	0.00	0.00	0.00	0.00	0.54	0.00	0.54
3	0.00	0.00	0.00	0.42	0.00	0.42	0.34	0.42	0.42	0.00	0.00	0.00	0.00	0.42	0.00	0.00	0.00	0.00	0.00
4	0.64	0.64	0.43	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

The data in this form will be transformed to a lower-dimensional matrix table of data which is a truncated SVD (Figure 20). The idea is to improve computational efficiency when training the model. Then we build the naïve bayes and logistic models. We also train the logistic model with LSA-processed input.

Figure 20

LSA-SVD



For the experimental results of rule-based models, Figure 22 shows a sample prediction produced by the logistic and naïve bayes.

Figure 22

Sample Prediction

```
                                                text  label  log_prediction
0  many advantage limiting car usage people many ...      6               0
1  shouldnt phone driving could cause different t...      0               5
2  recently cutting back usage car could extend l...      2               2
3  cant believe would think thi ignorantly known ...      3               1
4  designing summer assignment somethung student ...      6               5

                                                text  label  log_lsa_prediction
0  many advantage limiting car usage people many ...      6                   4
1  shouldnt phone driving could cause different t...      0                   4
2  recently cutting back usage car could extend l...      2                   5
3  cant believe would think thi ignorantly known ...      3                   5
4  designing summer assignment somethung student ...      6                   5

                                                text  label  bayes_prediction
0  many advantage limiting car usage people many ...      6                 5
1  shouldnt phone driving could cause different t...      0                 5
2  recently cutting back usage car could extend l...      2                 5
3  cant believe would think thi ignorantly known ...      3                 4
4  designing summer assignment somethung student ...      6                 5
```

We then evaluate the models with the classification report. Figure 23 is the report of logistic model and Figure 24 is the report of bayes naïve model. The comparison shows logistic regression has better accuracy and credibility than the naïve bayes does. The average f1 scores of labels is more than 0.5 and general accuracy is 0.65.

Figure 23

Classification Report of Logistic Model Prediction

	precision	recall	f1-score	support
0	0.652985	0.559105	0.602410	313.000000
1	0.604478	0.532895	0.566434	304.000000
2	0.644444	0.530488	0.581940	328.000000
3	0.512953	0.372180	0.431373	266.000000
4	0.710162	0.790488	0.748175	778.000000
5	0.638104	0.840336	0.725389	833.000000
6	0.715054	0.407975	0.519531	326.000000
accuracy	0.653748	0.653748	0.653748	0.653748
macro avg	0.639740	0.576210	0.596464	3148.000000
weighted avg	0.652199	0.653748	0.642334	3148.000000

Figure 24

Classification Report of Naïve Bayes Prediction

	precision	recall	f1-score	support
0	0.675676	0.159744	0.258398	313.000000
1	0.619403	0.273026	0.378995	304.000000
2	0.845070	0.182927	0.300752	328.000000
3	0.609375	0.146617	0.236364	266.000000
4	0.407797	0.847044	0.550543	778.000000
5	0.501296	0.696279	0.582915	833.000000
6	0.625000	0.061350	0.111732	326.000000
accuracy	0.473634	0.473634	0.473634	0.473634
macro avg	0.611945	0.338141	0.345671	3148.000000
weighted avg	0.564694	0.473634	0.415479	3148.000000

For the LSA-logistic model, the number of components of 500 and number of iterations of 1000 make it have scores close to previous ones. Figure 25 is the report. This shows that LSA hardly improves the model performance.

Figure 25

Classification Report of Logistic + LSA

	precision	recall	f1-score	support
0	0.605166	0.523962	0.561644	313.000000
1	0.610169	0.473684	0.533333	304.000000
2	0.631579	0.512195	0.565657	328.000000
3	0.479452	0.394737	0.432990	266.000000
4	0.679267	0.762211	0.718353	778.000000
5	0.626606	0.819928	0.710348	833.000000
6	0.663212	0.392638	0.493256	326.000000
accuracy	0.630559	0.630559	0.630559	0.630559
macro avg	0.613636	0.554194	0.573654	3148.000000
weighted avg	0.627776	0.630559	0.619453	3148.000000

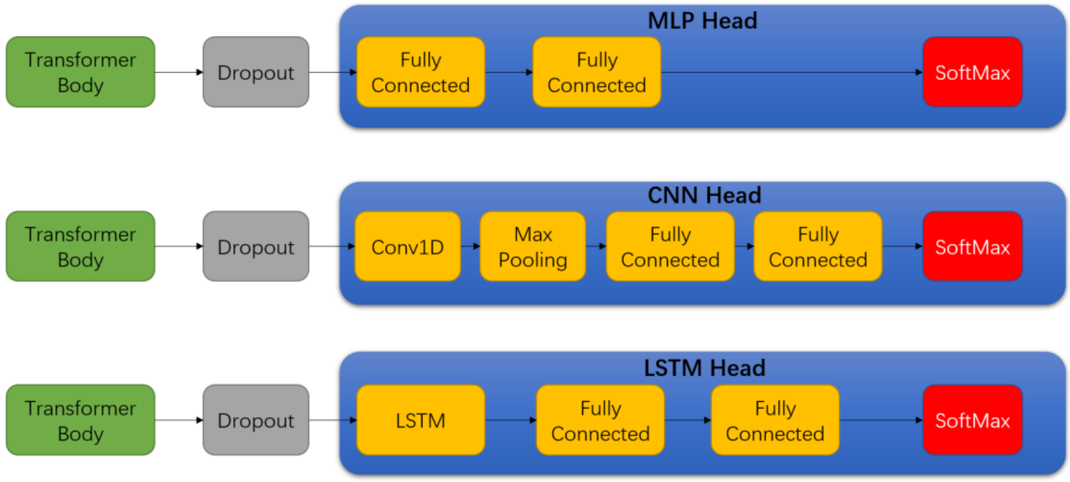
Transformer with heads

After checking those rule-based models' performances, we start using some statistical models on this text classification task. We try to design our own MLP, CNN or LSTM models, but their performances are so poor. Thus, we implement transfer learning with pre-trained models, in other words, transformers. Because transformers have already been trained on plenty of text, they work well than normal simple deep neural networks that we designed. However, just using transformers is not enough, and we need to adjust them for our task. So, we use transformers as models' body, and add some special heads for our text classification task.

We use 3 kinds of heads, MLP, CNN and LSTM, and their structures are shown in figure below. We add dropout layers right after the transformers body and between head layers to avoid overfitting. Also, according to Hendrycks and Gimpel (2016), GELU activation function is mathematically better than RELU activation function in nonlinearity task. So, we use GELU function in most fully connected layers. In the output layer, because this task is multiclass classification, we use SoftMax function.

Figure 26

Structures of Models

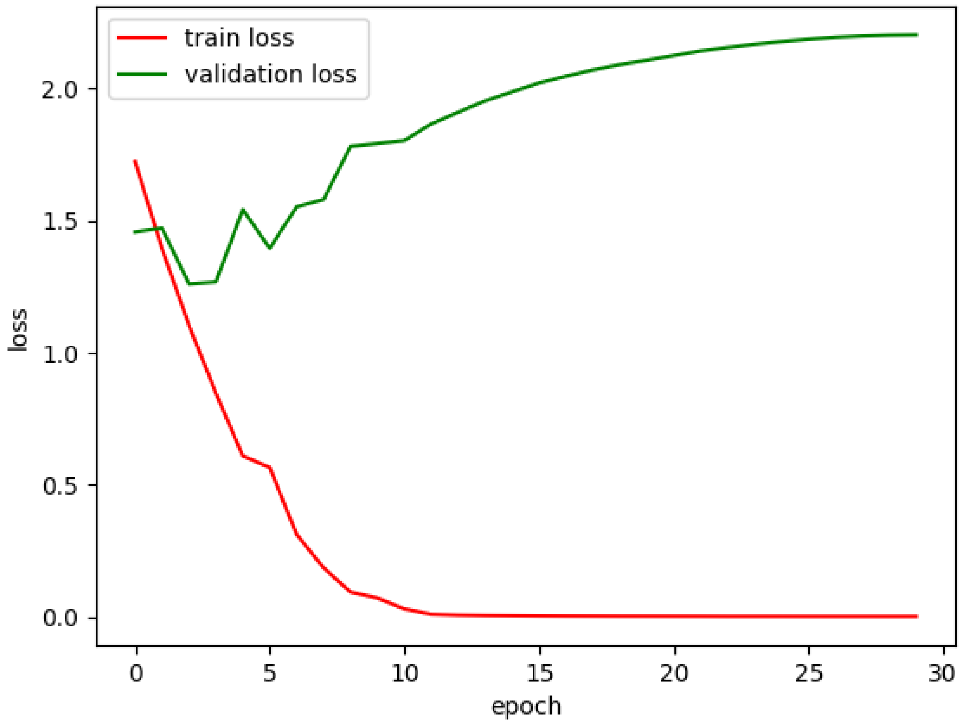


Before training, we tokenize the text data. In this training data, we calculated the distribution of number of tokens in each text data. Because most of the number of tokens are less than 150, we choose 150 as the max length.

To avoid overfitting, we have collected the loss changes of training data and validation data, and one of the changes in early training attempt is shown below. In this training, the train loss keeps decreasing, but the validation loss oscillates and reduced in the first 4 epochs, and then validation loss keeps increasing. This change of loss shows that overfitting occurred. Thus, we reduce the learning rate, add more dropout layers in the head and increase the dropout rate of the transformers.

Figure 27

Change of Loss in One Training Attempt



First, we try the base BERT model as transformer head. After around 10 – 15 epochs, these models have their best performance. The best perdition results of each model are shown in Table below. Considering both accuracy and F1-macro score, MLP head and CNN head perform very closely, MLP has a little better F1 score, and CNN has a little better accuracy. In these models using BERT body, LSTM head performed best, which we guess is because LSTM has memory to use previous sentences or tokens to make prediction. What's more, we have tried different transformer bodies such as RoBerta, XLM-RoBerta, and BigBird, but their performances are not good. The best one XLM-RoBerta with LSTM head only has an accuracy below 0.7.

Table 1

Results on Test Data

Transformer body	**Custom Head**	**Accuracy**	**F1-macro**
bert-base-cased	MLP	0.759847522	0.733070241
bert-base-cased	CNN	0.764612452	0.729890715
bert-base-cased	LSTM	0.783989835	0.760746121
xlm-roberta-base	LSTM	0.643348561	0.620621515

Summary + Transformer

In this section, we then use transformer pipeline ‘Summarization’ to summarize text and use the result them to improve the performance of the model. The reason why we try to use summarization is that we think summary can extract the core meaning and structure of the sentence, and we guess that might work for text classification. For instance, evidence is one of the seven classes in this dataset, and after using summarization the 200 length paragraph left only few sentences of 30 length may help model to classify them.

We use this code to initialize the summarizer ‘summarizer = pipeline("summarization",   model="t5-base", tokenizer="t5-base")’ , then set the parameter like this ‘summarizer(text, min_length=5, max_length=30)[0]['summary_text']’, and run them on the ‘discourse_text’ column.

Figure 28

Script of generating the summary.

```
summarizer = pipeline("summarization",  model="t5-base", tokenizer="t5-base")
res = []
for i in tqdm(range(len(df_train))):
    text = df_train.iloc[i]['text']
    res.append(summarizer(text, min_length=5, max_length=30)[0]['summary_text'])

df_train['summary'] = res
```

Figure 29

Example of summary.

text	label	summary
Technology is everywhere these days. In our pockets, on our tables and counters, everywhere we turn. You're using technology right now, as you read this. One of the most important types of technology we have is our cell phones. Everyone from 10-year-olds to 100-year-olds have them. To some people, it's just to connect with friends, to others, it's what their whole career depends on it. Either way, cell phones are a key element in today's society. Teenagers, out of everyone, are typically the age group that most depends on their cell phones. Whether it's just for talking to friends, communicating with their boss, or speaking with their family, there's no denying it. Teenagers love their cell phones.	0	cell phones are a key element in today's society . teens, out of everyone, are typically the age group that

As we can see, summarization works, it can reduce the length of the sentence, but it also has some problems of destructed the structure of whole paragraph and sometime the result of summarization is meaningless, since summarizer are not work very well; also, it costs plenty of time.

After generating them, we try several combinations, such as only putting the summary to train or add text and summary to train together. But we did not see much difference between them. Finally, we chose summary and text to train, and test on text only. And we mostly try them on BERT + LSTM. Here is the table.

Table 2

The result of training.

Pretrained	Model	Accuracy	F1 macro
bert-base-cased	BERT+LSTM	0.7820838627700127	0.7575876668982763
bert-base-uncased	BERT+LSTM	0.7916137229987293	0.7685482955944065

And we also use some pretrained like 'bert-large-uncased-whole-word-masking', 'bert-large-cased', 'bert-large-uncased', but they did not have good performance.

Summary Conclusion

In this final project, we preprocess the data and generate EDA visualizations to help us understand the data at the beginning. We have gained good understanding about the discourse types as the labels in the dataset. Then we prepare the experimental models.

Firstly, we build rule-based models of naïve bayes and logistic models. Logistic mode performs better than naïve bayes. LSA does not quite enhance the logistic model. The input data's dimensionality may not need a necessary reduction.

Secondly, of all the transformer body plus custom head models, BERT-LSTM model is the best one. The difference between different transformers body is larger than the difference between different custom head.

Finally, we try to use summarization to improve the result of the model. But there are still some problems remaining. For instance, we do not know whether adding original text and summary together is right or not to train the model. In addition, the summary of the texts is sometime not good enough. Moreover, it takes plenty of time to use summary transformer.

If improvements can be made, we can refine the text handling for the model. Some words can be handled better; therefore, trying other NLP packages or built-in methods could be useful. For the application of summary with transformers, maybe we can change the maximum length of summaries to see if there is any improvement.

References

Feedback prize - evaluating student writing. Kaggle. (n.d.). Retrieved December 11, 2022, from https://www.kaggle.com/competitions/feedback-prize-2021/data

Saxena, N. (2020, September 12). *Extracting keyphrases from text: Rake and gensim in python*. Medium. Retrieved December 11, 2022, from https://towardsdatascience.com/extracting-keyphrases-from-text-rake-and-gensim-in-python-eefd0fad582f

Generating Word Cloud in Python, https://www.geeksforgeeks.org/generating-word-cloud-python/#:~:text=For%20generating%20word%20cloud%20in,from%20UCI%20Machine%20Learning%20Repository.

Hendrycks, D., & Gimpel, K. (2016). Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*.

Ioana. (2020, November 22). *Latent semantic analysis: Intuition, math, implementation*. Medium. Retrieved December 11, 2022, from https://towardsdatascience.com/latent-semantic-analysis-intuition-math-implementation-a194aff870f8

Sangani, R. (2022, January 26). Adding custom layers on top of a hugging face model. Medium. Retrieved December 11, 2022, from https://towardsdatascience.com/adding-custom-layers-on-top-of-a-hugging-face-model-f1ccdfc257bd

Sklearn.decomposition.truncatedsvd. scikit. (n.d.). Retrieved December 11, 2022, from https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html

Appendix

```
# ------------------------preprocessing-----------------------
# ----------load data
# dataset = pd.read_csv('train.csv')
dataset = pd.read_csv('train.csv.zip')
data = dataset[['discourse_text', 'discourse_type']].copy()
# ----------check null values
print('missing values exists:\n', data.isnull().any())
# ----------set numeric label
labels = enumerate(set(data['discourse_type']))
label2int = {}
for i, l in labels:
    label2int[l] = i

data['label'] = data['discourse_type'].apply(lambda x: label2int[x])
print(data.head())
```

```python
# ----------clean text
print(f'Text contains non-ASCII characters, for example: {data.discourse_text.iloc[144290].encode()}')
def clean_text(x):
    # ----------convert text to ASCII form
    x = unidecode(x)
    # ----------lowercase
    x = x.lower()
    # ----------remove consecutive letter 3ormore
    x = re.sub(r'([^\W\d_])\1{2,}', r'\1\1', x)
    # ----------remove url(not-well formatted)
    # match_url = re.compile(r'http\S+')
    match_url = re.compile(r'https?://(www\.)?([-_\w\s\.\/]*)')
    x = re.sub(match_url, "", x)
    # ----------remove parenthesis
    # x = re.sub(re.compile(r'\([^\)]*\)'), "", x)
    x = re.sub(re.compile(r'[()]'), "", x)
    return x

data['text'] = data['discourse_text'].astype(str).apply(clean_text)


print(f'after cleaning: {data.text.iloc[144290]}')
# ----------remove stop words
stop_words = nltk.corpus.stopwords.words('english')

def remove_stop_words(corpus):
    result = []
    corp = corpus.split(' ')
    result = [w for w in corp if w not in stop_words]
    result = " ".join(result).strip()

    return result

data['text'] = data['text'].apply(remove_stop_words)

# ----------lemmatize
lemma = nltk.WordNetLemmatizer()
data['text'] = data.text.apply(lemma.lemmatize)

# ----------remove non-sense text
print('Dataset has text with no sense:')
print(data[data.text==""])
data = data[data.text!=""].reset_index()

# ----------savedata
df = data.filter(['text', 'discourse_text', 'discourse_type', 'label'])
path = os.getcwd()
df.to_csv(f'{path}/clean_train.csv')
```

```python
    dataset = pd.read_csv('Dataset/clean_train.csv', index_col=[0])
    data = dataset.filter(['text', 'discourse_type', 'label'])

    # ----------word tokenization
    def nltk_tokenization(text, remove_punc=False):
        # remove contraction
        text = contractions.fix(text)
        # remove punc
        if remove_punc:
            text = re.sub(r'[^\w\s]','',text)
        text = nltk.word_tokenize(text)

        return text

    data['token'] = data.text.apply(nltk_tokenization, remove_punc=True)


# ----------words frequency

# Lead(label 0)
df_lead = data.token[data.label==0]
# Position(label 1)
df_pos = data.token[data.label==1]
# Claim(label 3)
df_claim = data.token[data.label==3]
# Counter Claim(6)
df_counter = data.token[data.label==6]
# Rebuttal(5)
df_rebut = data.token[data.label==5]
# Evidence(2)
df_evidence = data.token[data.label==2]
# Concluding Statement(4)
df_conclude = data.token[data.label==4]

def get_bag_of_words(list_of_words, counter):
    counter.update(list_of_words)

def get_most_n(df, n):
    ct = Counter()
    df.apply(get_bag_of_words, counter=ct)
    return ct.most_common(n)
```

```python
lead = dict(get_most_n(df_lead, 80)[6:])
position = dict(get_most_n(df_pos, 80)[5:])
claim = dict(get_most_n(df_claim, 80)[4:])
counterc = dict(get_most_n(df_counter, 80)[5:])
rebut = dict(get_most_n(df_rebut, 80)[3:])
evidence = dict(get_most_n(df_evidence, 80)[5:])
conclusion = dict(get_most_n(df_conclude, 80)[4:])

# ----------phrase detection
# nlp = spacy.load('en_core_web_sm')
#
# def phrase_tokenization(text):
#     spacy_text = nlp(text)
#     return [chunk for chunk in spacy_text.noun_chunks]
#
# data['word_chunk'] = data.text.apply(phrase_tokenization)

def phrase_tokenization(text):
    r = Rake()
    r.extract_keywords_from_text(text)
    return r.get_ranked_phrases()

data['word_chunk'] = dataset.discourse_text.apply(phrase_tokenization)
```

```python
# ----------phrase frequency

# Lead(label 0)
df_lead_ph = data.word_chunk[data.label==0]
# Position(label 1)
df_pos_ph = data.word_chunk[data.label==1]
# Claim(label 3)
df_claim_ph = data.word_chunk[data.label==3]
# Counter Claim(6)
df_counter_ph = data.word_chunk[data.label==6]
# Rebuttal(5)
df_rebut_ph = data.word_chunk[data.label==5]
# Evidence(2)
df_evidence_ph = data.word_chunk[data.label==2]
# Concluding Statement(4)
df_conclude_ph = data.word_chunk[data.label==4]


def get_bag_of_phrase(list_of_words, counter, phrase_len):
    phrase = [p for p in list_of_words if len(p.split())>=phrase_len]
    counter.update(phrase)


def get_most_n_ph(df, n, phrase_len):
    ct = Counter()
    df.apply(get_bag_of_phrase, counter=ct, phrase_len=phrase_len)
    return ct.most_common(n)
```

```python
lead_ph = dict(get_most_n_ph(df_lead_ph, 80, 2)[3:])
position_ph = dict(get_most_n_ph(df_pos_ph, 80, 2)[3:])
claim_ph = dict(get_most_n_ph(df_claim_ph, 80, 2)[4:])
counterc_ph = dict(get_most_n_ph(df_counter_ph, 80, 2)[3:])
rebut_ph = dict(get_most_n_ph(df_rebut_ph, 80, 2)[3:])
evidence_ph = dict(get_most_n_ph(df_evidence_ph, 80, 2)[3:])
conclusion_ph = dict(get_most_n_ph(df_conclude_ph, 80, 2)[3:])

# ------------------------------Visualization------------------------------
# ----------tables
print(pd.DataFrame(dataset.filter(['discourse_text', 'discourse_type', 'label'])).head(5))
print('number of samples with each label:')
print(data[['discourse_type', 'label']].value_counts())

# ----------target distribution
temp_df = pd.DataFrame(data['discourse_type'].value_counts())
fig = plt.figure(figsize=(16, 6))
plt.barh(temp_df.index, temp_df.discourse_type)
plt.title('Discourse Element Type Distribution')
plt.xlabel('count')
plt.ylabel('discourse types')
plt.grid()
plt.show()

# ----------frequent word in dataset
df_all_word = pd.DataFrame(get_most_n(data.token, 15), columns=['word', 'frequency'])
fig = plt.figure(figsize=(10, 6))
sns.barplot(data=df_all_word, x='frequency', y='word', color='steelblue')
plt.title('Top 15 Frequency Words')
plt.grid()
plt.show()

# # ----------frequent phrase in dataset
df_all_phrase = pd.DataFrame(get_most_n_ph(data.word_chunk, 20, 2), columns=['word', 'frequency'])
fig = plt.figure(figsize=(16, 6))
sns.barplot(data=df_all_phrase, x='frequency', y='word', color='steelblue')
plt.title('Top 15 Frequency Phrase')
plt.ylabel('phrase')
plt.grid()
plt.show()

# ----------word cloud for each discourse element
word_all_df = [lead, position, claim, counterc, rebut, evidence, conclusion]
phrase_all_df = [lead_ph, position_ph, claim_ph, counterc_ph, rebut_ph, evidence_ph, conclusion_ph]
df_label = ['Lead Statement', 'Position Statement', 'Claim', 'Counterclaim',
            'Rebuttal Statement', 'Evidence', 'Conclusion']

for z in range(len(df_label)):
    fig, ax = plt.subplots(1, 2, figsize=(12, 6))
    wordcloud1 = WordCloud(width=900, height=900,
                           background_color='white',
                           min_font_size=10, random_state=12).generate_from_frequencies(word_all_df[z])
    wordcloud2 = WordCloud(width=900, height=900,
                         background_color='white',
                         min_font_size=10, random_state=12).generate_from_frequencies(phrase_all_df[z])

    plt.figure(figsize=(10, 10), facecolor=None)
    ax[0].imshow(wordcloud1)
    ax[0].set_title(f'Most Frequent Word in {df_label[z]}', fontsize=18, color='lightcoral')
    ax[1].imshow(wordcloud2)
    ax[1].set_title(f'Most Frequent Phrase in {df_label[z]}', fontsize=18, color='lightcoral')
    ax[0].axis('off')
    ax[1].axis('off')
    plt.tight_layout(pad=0)
    plt.show()
```

```python
### Vectorizer TFIDF
tfidf_vector = TfidfVectorizer()
tfidf_vector.fit(df_train_new['text'])
X_train_tfidf = tfidf_vector.transform(df_train_new['text'])
X_test_tfidf  = tfidf_vector.transform(df_test_new['text'])



### Logistic no LSA
clf_lo = LogisticRegression().fit(X_train_tfidf, df_train_new['label'])
predict_lo = clf_lo.predict(X_test_tfidf)
report_lo = classification_report(df_test_new['label'], predict_lo,
                                  target_names = sorted([str(i) for i in df_train_new['label'].unique()]),
                                  output_dict=True)
### Logistic + LSA
lsa = TruncatedSVD(n_components = 500, n_iter = 100)
lsa.fit(X_train_tfidf)
X_train_lsa = lsa.transform(X_train_tfidf)
X_test_lsa = lsa.transform(X_test_tfidf)
clf_l = LogisticRegression().fit(X_train_lsa, df_train_new['label'])
predict_l = clf_l.predict(X_test_lsa)
report_l_lsa = classification_report(df_test_new['label'], predict_l,
                                     target_names = sorted([str(i) for i in df_train_new['label'].unique()]),
                                     output_dict=True)
### Naive Bayes
clf_n = MultinomialNB().fit(X_train_tfidf, df_train_new['label'])
predict_n = clf_n.predict(X_test_tfidf)
report_nb = classification_report(df_test_new['label'], predict_n,
                                  target_names = sorted([str(i) for i in df_train_new['label'].unique()]),
                                  output_dict=True)

# da = pd.DataFrame({'text': df_test_new.text, 'label': df_test_new['label'],'log_prediction': predict_lo})
# print(da.head())
# print()
# db = pd.DataFrame({'text': df_test_new.text, 'label': df_test_new['label'],'log_lsa_prediction': predict_l})
# print(db.head())
# print()
# dc = pd.DataFrame({'text': df_test_new.text, 'label': df_test_new['label'],'bayes_prediction': predict_n})
# print(dc.head())
# print()

df_lo= pd.DataFrame(report_lo).transpose()
print(df_lo)
print()
df_l_lsa = pd.DataFrame(report_l_lsa).transpose()
print(df_l_lsa)
print()
df_nb = pd.DataFrame(report_nb).transpose()
print(df_nb)
```

```python
# tokenizer and data loader
tokenizer = AutoTokenizer.from_pretrained(checkpoint)
tokenizer.model_max_len = 150

def tokenize_train(batch):
  return tokenizer(batch['text'], batch['summary'], truncation=True, max_length=150)

def tokenize_test(batch):
  return tokenizer(batch['text'], truncation=True, max_length=150)

tokenized_dataset_train = data['train'].map(tokenize_train, batched=True)
tokenized_dataset_valid = data['valid'].map(tokenize_train, batched=True)
tokenized_dataset_test = data['test'].map(tokenize_train, batched=True)

tokenized_dataset = DatasetDict({'train':tokenized_dataset_train, 'valid':tokenized_dataset_valid, 'test':tokenized_dataset_test})

tokenized_dataset.set_format("torch", columns=["input_ids", "attention_mask", "label"])

data_collator = DataCollatorWithPadding(tokenizer=tokenizer)

train_dataloader = DataLoader(
    tokenized_dataset["train"], shuffle=True, batch_size=BATCH_SIZE, collate_fn=data_collator
)
eval_dataloader = DataLoader(
    tokenized_dataset["valid"], batch_size=BATCH_SIZE, collate_fn=data_collator
)
test_dataloader = DataLoader(
    tokenized_dataset["test"], batch_size=BATCH_SIZE, collate_fn=data_collator
)
```

```python
# model definition
class MLPCustomModel(nn.Module):
    def __init__(self, checkpoint, num_labels):
        super(MLPCustomModel, self).__init__()
        self.num_labels = num_labels

        # Load Model with given checkpoint and extract its body
        self.model = AutoModel.from_pretrained(checkpoint,
                                               config=AutoConfig.from_pretrained(
                                                   checkpoint,
                                                   output_attentions=True,
                                                   output_hidden_states=True))
        self.dropout = nn.Dropout(0.1)
        # Add MLP custom layers
        self.classifier1 = nn.Linear(768, 384)  # load and initialize weights
        self.act = nn.GELU()
        self.classifier2 = nn.Linear(384, num_labels)  # load and initialize weights

    def forward(self, input_ids=None, attention_mask=None, labels=None):
        # Extract outputs from the body
        outputs = self.model(input_ids=input_ids, attention_mask=attention_mask)

        # Add MLP custom layers
        sequence_output = self.dropout(outputs[0])  # outputs[0]=last hidden state

        x = sequence_output[:, 0, :]
        x = x.view(-1, 768)
        logits = self.classifier1(x)
        logits = self.classifier2(self.dropout(self.act(logits)))  # calculate losses
```

```python
        loss = None
        if labels is not None:
            loss_fct = nn.CrossEntropyLoss()
            loss = loss_fct(logits.view(-1, self.num_labels), labels.view(-1))

        return TokenClassifierOutput(loss=loss, logits=logits,
                                     hidden_states=outputs.hidden_states,
                                     attentions=outputs.attentions)

class CNNCustomModel(nn.Module):
    def __init__(self, checkpoint, num_labels):
        super(CNNCustomModel, self).__init__()
        self.num_labels = num_labels

        self.model = AutoModel.from_pretrained(checkpoint,
                                               config=AutoConfig.from_pretrained(
                                                   checkpoint,
                                                   output_attentions=True,
                                                   output_hidden_states=True))
        # add CNN layers
        self.conv = nn.Conv1d(in_channels=1, out_channels=256, kernel_size=7)
        self.relu = nn.ReLU()
        self.pool = nn.MaxPool1d(kernel_size=3)

        self.dropout = nn.Dropout(0.1)

        self.clf1 = nn.Linear(256 * 254, 256)
        self.clf2 = nn.Linear(256, num_labels)


    def forward(self, input_ids=None, attention_mask=None, labels=None):
        # Extract outputs from the body
        # outputs = self.bert(input_ids=input_ids, attention_mask=attention_mask)
        batch_size = len(input_ids)
        outputs = self.model(input_ids=input_ids, attention_mask=attention_mask)

        # Add CNN custom layers
        x = self.dropout(outputs[0])
        x = x[:, 0, :]
        # x = x.permute(0, 2, 1)
        x = x.reshape(batch_size, 1, 768)
        x = self.conv(x)
        x = self.relu(x)
        x = self.pool(x)
        x = self.dropout(x)
        # x = x.view(-1,)
        x = x.reshape(batch_size, 256 * 254)
        x = self.clf1(x)
        x = self.relu(x)
        x = self.dropout(x)
        x = self.clf2(x)

        loss = None
        if labels is not None:
            loss_fct = nn.CrossEntropyLoss()
            loss = loss_fct(x.view(-1, self.num_labels), labels.view(-1))

        return TokenClassifierOutput(loss=loss, logits=x,
                                     hidden_states=outputs.hidden_states,
                                     attentions=outputs.attentions)
```

```
class LSTMCustomModel(nn.Module):
    def __init__(self, checkpoint, num_labels):
        super(LSTMCustomModel, self).__init__()
        self.num_labels = num_labels

        self.model = AutoModel.from_pretrained(checkpoint,
                                               config=AutoConfig.from_pretrained(
                                                   checkpoint,
                                                   output_attentions=True,
                                                   output_hidden_states=True))
        # add LSTM layers
        self.dropout = nn.Dropout(0.1)
        # self.hidden_size = self.model.config.hidden_size
        self.lstm = nn.LSTM(768, 256, batch_first=True, bidirectional=True)
        self.clf1 = nn.Linear(256*2, 384)
        self.act = nn.GELU()
        self.clf2 = nn.Linear(384, num_labels)

  def forward(self, input_ids=None, attention_mask=None, labels=None):
      # Extract outputs from the body
      # outputs = self.bert(input_ids=input_ids, attention_mask=attention_mask)
      outputs = self.model(input_ids=input_ids, attention_mask=attention_mask)

      # add LSTM layers
      sequence_output = outputs[0]
      lstm_output, (h, c) = self.lstm(sequence_output)  ## extract the 1st token's embeddings
      hidden1 = lstm_output[:, -1, :256]
      hidden2 = lstm_output[:, 0, 256:]
      hidden = torch.cat((hidden1, hidden2), dim=-1)
      linear_output = self.clf1(hidden.view(-1, 256 * 2))
      linear_output = self.clf2(self.dropout(self.act(linear_output)))


      loss = None
      if labels is not None:
          loss_fct = nn.CrossEntropyLoss()
          loss = loss_fct(linear_output.view(-1, self.num_labels), labels.view(-1))

      return TokenClassifierOutput(loss=loss, logits=linear_output,
                                   hidden_states=outputs.hidden_states,
                                   attentions=outputs.attentions)

if head == 'MLP':
    model = MLPCustomModel(checkpoint=checkpoint, num_labels=number_labels)
elif head == 'CNN':
    model = CNNCustomModel(checkpoint=checkpoint, num_labels=number_labels)
else:
    model = LSTMCustomModel(checkpoint=checkpoint, num_labels=number_labels)
```

```python
model = model.to(device)

optimizer = torch.optim.AdamW(model.parameters(), lr=LR)

num_training_steps = num_epochs * len(train_dataloader)
lr_scheduler = get_scheduler(
    "linear",
    optimizer=optimizer,
    num_warmup_steps=0,
    num_training_steps=num_training_steps,
)
# ---------------------------------------------------------------
# train model
progress_bar_train = tqdm(range(num_training_steps))
progress_bar_eval = tqdm(range(num_epochs * len(eval_dataloader)))

print('\n')
print('start training')

hist_val_loss = []
hist_train_loss = []

for epoch in range(num_epochs):
    print('*' * 100)
    print(f'epoch : {epoch}\n')
    model.train()
    hist_train_loss_epoch = []
    hist_val_loss_epoch = []
    for batch in train_dataloader:
        batch = {k: v.to(device) for k, v in batch.items()}
        outputs = model(**batch)
        loss = outputs.loss
        hist_train_loss_epoch.append(loss.item())
        loss.backward()

        optimizer.step()
        lr_scheduler.step()
        optimizer.zero_grad()
        progress_bar_train.update(1)
    hist_train_loss.append(mean(hist_train_loss_epoch))
    model.eval()
    for batch in eval_dataloader:
        batch = {k: v.to(device) for k, v in batch.items()}
        with torch.no_grad():
            outputs = model(**batch)
        hist_val_loss_epoch.append(outputs.loss.item())
        logits = outputs.logits

        predictions = torch.argmax(logits, dim=-1)
        metric.add_batch(predictions=predictions, references=batch["labels"])
        progress_bar_eval.update(1)
    hist_val_loss.append(mean(hist_val_loss_epoch))
    print('*' * 100)
    acc = metric.compute()['accuracy']
```

```python
    if acc > max_acc:
        max_acc = acc
        # torch.save(model.state_dict(), "model_{}.pt".format(head))
        torch.save(model.state_dict(), f"model_final_{head}_{max_acc}.pt")
        print('Model has been saved!')
    print(f'Epoch : {epoch}')
    print(f'Accuracy: {acc}')
    print('validation finished')
    print(f'epoch {epoch} finished')
    print('*'*100)

print('training over')

plt.figure(figsize=(20,8))
plt.plot(hist_val_loss)
plt.plot(hist_train_loss)
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['val', 'train'], loc='upper left')
plt.show()
```

```python
# ---------------------------------------------------------------
# test the model
print('*'*100)
print('test start')

metric1 = load_metric('accuracy')
model.eval()

total_predictions = []
true_results = []

for batch in tqdm(test_dataloader):
    batch = {k: v.to(device) for k, v in batch.items()}
    with torch.no_grad():
        outputs = model(**batch)

    logits = outputs.logits
    predictions = torch.argmax(logits, dim=-1)
    metric1.add_batch(predictions=predictions, references=batch["labels"])
    predictions = predictions.detach().cpu().numpy()
    predictions = list(predictions)
    true_result = batch['labels'].detach().cpu().numpy()
    true_result = list(true_result)
    total_predictions.extend(predictions)
    true_results.extend(true_result)
print(metric1.compute())

metric2 = load_metric("f1")
print(metric2.compute(predictions=total_predictions, references=true_results, average="macro"))
```