ML for NLP2 2023-2024

Lab 03 Tokenization / BERT representation

22210802 Haeeul HWANG

Using the Sequoia and German GSD corpora:

- Extract all nouns and their gender

German GSD

	LEMMA	FEATS
0	Beratung	Fem
1	Behebung	Fem
2	Problem	Neut
3	Kundenservice	Neut
4	Rahmen	Masc
5	Gespräch	Neut
6	Ergebnis	Neut
7	Zeit	Fem
8	Behandlung	Fem
9	Leiden	Neut
10	Physiotherapieraxis	Fem
11	Positive	Neut
12	Terminvergabe	Fem
13	Behandlungsraum	Masc
14	Trainingsplan	Masc
15	Mitarbeiter	Masc
16	Sauberkeit	Fem
17	Ordnung	Fem
18	Freundlichkeit	Fem
19	Standard	Masc

French Sequoia

	LEMMA	FEATS
0	exposition	Fem
1	siècle	Masc
2	site	Masc
3	industrie	Fem
4	forge	Fem
5	emplacement	Masc
6	fourneau	Masc
7	fonderie	Fem
8	jour	Masc
9	maire	Masc
10	échevin	Masc
11	lettre	Fem
12	val	Masc
13	vallée	Fem
14	aval	Masc
15	forge	Fem
16	apogée	Masc
17	siècle	Masc
18	long	Masc
19	année	Fem

- Tokenize the nouns using BERT tokenizer

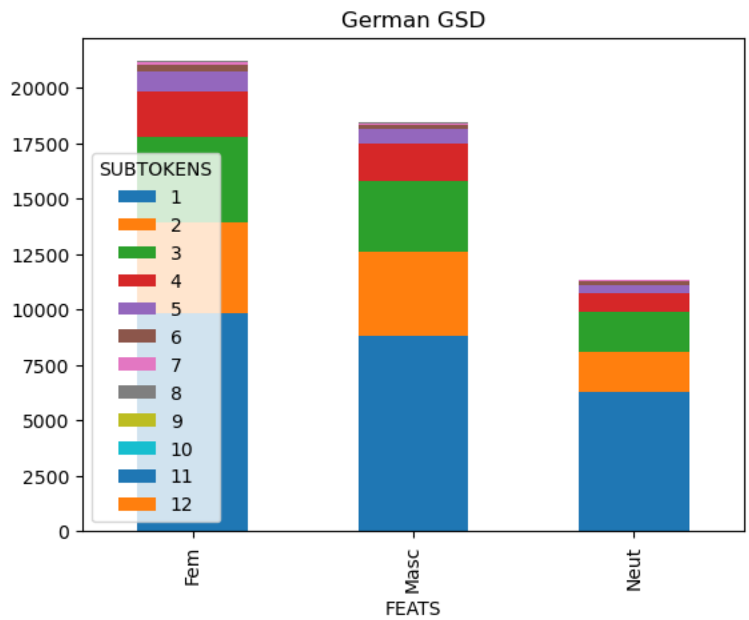
German GSD

	LEMMA	FEATS	TOKENS
0	Beratung	Fem	[Be, ##ratu, ##ng]
1	Behebung	Fem	[Be, ##hebung]
2	Problem	Neut	[Problem]
3	Kundenservice	Neut	[Kunden, ##ser, ##vice]
4	Rahmen	Masc	[Rahmen]

French Sequoia

	LEMMA	FEATS	TOKENS
0	exposition	Fem	[exposition]
1	siècle	Masc	[siècle]
2	site	Masc	[site]
3	industrie	Fem	[industrie]
4	forge	Fem	[for, ##ge]

- For each grammatical gender, determine the distribution of the number of subtokens resulting from the tokenization of a noun



Mean number of subtokens - GSD:

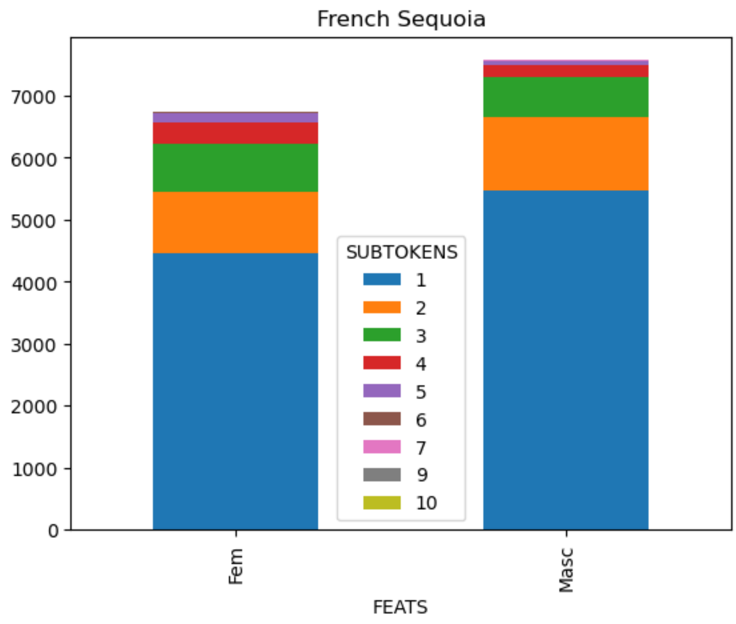
FEATS

Fem 2.137908

Masc 2.054168

Neut 1.942116

Name: SUBTOKENS, dtype: float64



Mean number of subtokens - Sequoia:

FEATS

Fem 1.629751

Masc 1.448567

Name: SUBTOKENS, dtype: float64

- Conclusion

In German, nouns are categorized into three genders: feminine, masculine, and neuter.
It can be observed from the plot that words with up to 5 subtokens are prominently displayed.
In the corpus, feminine nouns appear to be the most abundant, and overall, most words consist of a single token.
In French, nouns are classified as feminine or masculine, and in this corpus, masculine nouns seem to be more prevalent.
Most words consist of a single token, and it can be observed from the plot that words with up to 4 subtokens are prominently displayed.
The number of subtokens appears to be higher in German (2.06) compared to French (1.53), regardless of the gender of the nouns. In both languages, words with feminine nouns have the highest number of subtokens.

Using BERT features

- The embeddings of the *amazon_polarity* dataset
- Use this representations to train a logistic regression model

1. **Using the [CLS] token**

```
# get the embeddings of the dataset using [CLS] token with batch size 32

def get_embeddings(data, batch_size = 32):
    dataset = TensorDataset(data['input_ids'], data['attention_mask'])
    dataloader = DataLoader(dataset, batch_size = batch_size)
    embeddings = []
    with torch.no_grad():
        for input_ids, attention_mask in dataloader:
            input_ids = input_ids.to(device)
            attention_mask = attention_mask.to(device)
            outputs = model(input_ids, attention_mask)
            embeddings.append(outputs.last_hidden_state[:,0,:].cpu())
    return torch.cat(embeddings)
```

```
# train a logistic regression model

clf = LogisticRegression(random_state = 42)
clf.fit(train_data_embeddings.cpu().numpy(), df_train_sample['label'])

# predict the test data

y_pred = clf.predict(test_data_embeddings.cpu().numpy())
accuracy = accuracy_score(df_test_sample['label'], y_pred)
print(f'Accuracy: {accuracy}')
```

Accuracy: 0.795

```python
# test for different training sizes

train_sizes = [0.002, 0.003]
accuracies = []

for train_size in train_sizes:
    df_train_sample = df_train.sample(frac = train_size, random_state = 42)
    train_data = tokenize_data(df_train_sample)
    train_data_embeddings = get_embeddings(train_data)
    clf = LogisticRegression(random_state = 42)
    clf.fit(train_data_embeddings.cpu().numpy(), df_train_sample['label'])
    y_pred = clf.predict(test_data_embeddings.cpu().numpy())
    accuracy = accuracy_score(df_test_sample['label'], y_pred)
    accuracies.append(accuracy)
    print(f'Training size: {train_size}, Accuracy: {accuracy}')
```

Training size: 0.002, Accuracy: 0.8125

Training size: 0.003, Accuracy: 0.8325

2. Using mean-pooling

```python
# get the embeddings of the dataset using mean pooling with batch size 32

def get_embeddings_mean_pooling(data, batch_size = 32):
    dataset = TensorDataset(data['input_ids'], data['attention_mask'])
    dataloader = DataLoader(dataset, batch_size = batch_size)
    embeddings = []
    with torch.no_grad():
        for input_ids, attention_mask in dataloader:
            input_ids = input_ids.to(device)
            attention_mask = attention_mask.to(device)
            outputs = model(input_ids, attention_mask)
            mask = attention_mask.unsqueeze(2).expand(outputs.last_hidden_state.size())
            sum_embeddings = torch.sum(outputs.last_hidden_state * mask, 1)
            mask = mask.sum(1)
            embeddings.append(sum_embeddings / mask)
    return torch.cat(embeddings)

train_data_mean_embeddings = get_embeddings_mean_pooling(train_data)
test_data_mean_embeddings = get_embeddings_mean_pooling(test_data)
```

```
# train a logistic regression model

clf = LogisticRegression(random_state = 42)
clf.fit(train_data_mean_embeddings.cpu().numpy(), df_train_sample['label'])

# predict the test data

y_pred = clf.predict(test_data_mean_embeddings.cpu().numpy())
accuracy = accuracy_score(df_test_sample['label'], y_pred)
print(f'Accuracy: {accuracy}')
```

Accuracy: 0.83

```
# test for different training sizes

train_sizes = [0.002, 0.003]
accuracies = []

for train_size in train_sizes:
    df_train_sample = df_train.sample(frac = train_size, random_state = 42)
    train_data = tokenize_data(df_train_sample)
    train_data_mean_embeddings = get_embeddings_mean_pooling(train_data)
    clf = LogisticRegression(random_state = 42)
    clf.fit(train_data_mean_embeddings.cpu().numpy(), df_train_sample['label'])
    y_pred = clf.predict(test_data_mean_embeddings.cpu().numpy())
    accuracy = accuracy_score(df_test_sample['label'], y_pred)
    accuracies.append(accuracy)
    print(f'Training size: {train_size}, Accuracy: {accuracy}')
```

Training size: 0.002, Accuracy: 0.82

Training size: 0.003, Accuracy: 0.83

- compare the performance to the one you have achieved in the first lab

When increasing the size of the training set, the model's accuracy significantly improved when using [CLS] token embeddings.
However, when using mean-pooling for embeddings, the correlation with the size of the training set seemed weaker.
This might be because mean-pooling simply averages the embeddings of all tokens in the sentence.
Therefore, it seems that the mean-pooling method may not fully capture the essence of the data as effectively as the [CLS] token embeddings.

In conclusion, as observed in the first task, there is a higher likelihood of improved model accuracy when using embeddings generated by BERT, especially when using [CLS] token embeddings, compared to when using TfidfVectorizer for classification tasks.