Lab4-Assignment about Named Entity Recognition and Classification

This notebook describes the assignment of Lab 4 of the text mining course. We assume you have succesfully completed Lab1, Lab2 and Lab3 as welll. Especially Lab2 is important for completing this assignment.

Learning goals

- going from linguistic input format to representing it in a feature space
- working with pretrained word embeddings
- train a supervised classifier (SVM)
- evaluate a supervised classifier (SVM)
- learn how to interpret the system output and the evaluation results
- be able to propose future improvements based on the observed results

Credits

This notebook was originally created by Marten Postma and Filip Ilievski and adapted by Piek vossen

In [1]:
```
## all imports
from nltk.corpus.reader import ConllCorpusReader

from collections import Counter
from typing import Dict

from sklearn.feature_extraction import DictVectorizer
from sklearn import svm
from sklearn.metrics import classification_report

import gensim

import pandas
```

[Points: 18] Exercise 1 (NERC): Training and evaluating an SVM using CoNLL-2003

[4 point] a) Load the CoNLL-2003 training data using the *ConllCorpusReader* and create for both *train.txt* and *test.txt*:

```
[2 points]  -a list of dictionaries representing the features for each training instances, e..g,
```
[
{'words': 'EU', 'pos': 'NNP'},
{'words': 'rejects', 'pos': 'VBZ'},
...
]
```

[2 points] -the NERC labels associated with each training instance, e.g.,
dictionaries, e.g.,
```
[
'B-ORG',
'O',
....
]
```
```

In [2]:
```
### Adapt the path to point to the CONLL2003 folder on your local machine
train = ConllCorpusReader(r'C:\Users\remia\Desktop\labs\ba-text-mining\lab_sessions\lab4\nerc_datasets\CONLL2003', 'train.txt', ['words', 'pos', 'ignore', 'chunk'])
training_features = []
training_gold_labels = []

for token, pos, ne_label in train.iob_words():
    a_dict = {
      'bias': 1.0,
      'words': token,  # original token
      'pos': pos,  # Part Of Speech tag of the token
      'word.lower()': token.lower(),  # lower case variant of the token
      'word[-3:]': token[-3:],  # suffix of 3 characters
      'word[-2:]': token[-2:],  # suffix of 2 characters
      'word.isupper()': token.isupper(),  # is the token in uppercase
      'word.istitle()': token.istitle(),  # does the token start with a capital letter
      'word.isdigit()': token.isdigit(),  # is the token a digit
      'postag': pos,  # Part Of Speech tag
      'postag[:2]': pos[:2],  # first two characters of the PoS tag
    }
    training_features.append(a_dict)
    training_gold_labels.append(ne_label)
```

In [3]:
```
### Adapt the path to point to the CONLL2003 folder on your local machine
test = ConllCorpusReader(r'C:\Users\remia\Desktop\labs\ba-text-mining\lab_sessions\lab4\nerc_datasets\CONLL2003', 'test.txt', ['words', 'pos', 'ignore', 'chunk'])

test_features = []
test_gold_labels = []
for token, pos, ne_label in test.iob_words():
    a_dict = {
      'bias': 1.0,
      'words': token,  # original token
      'pos': pos,  # Part Of Speech tag of the token
      'word.lower()': token.lower(),  # lower case variant of the token
      'word[-3:]': token[-3:],  # suffix of 3 characters
      'word[-2:]': token[-2:],  # suffix of 2 characters
      'word.isupper()': token.isupper(),  # is the token in uppercase
      'word.istitle()': token.istitle(),  # does the token start with a capital letter
      'word.isdigit()': token.isdigit(),  # is the token a digit
      'postag': pos,  # Part Of Speech tag
      'postag[:2]': pos[:2],  # first two characters of the PoS tag
    }
    test_features.append(a_dict)
    test_gold_labels.append(ne_label)
```

[2 points] b) provide descriptive statistics about the training and test data:

- How many instances are in train and test?
- Provide a frequency distribution of the NERC labels, i.e., how many times does each NERC label occur?
- Discuss to what extent the training and test data is balanced (equal amount of instances for each NERC label) and to what extent the training and test data differ?

Tip: you can use the following `Counter` functionality to generate frequency list of a list:

In [4]:
```
num_train:int = len(training_features) # number of instances in the training set
num_test:int = len(test_features) # number of instances in the test set

freq_dist_train = Counter(training_gold_labels)
freq_dist_test = Counter(test_gold_labels)

freq_dist_train: Dict[str, int] = dict(sorted(freq_dist_train.items(), key=lambda item: item[1], reverse=True)) # The sorted frequency distribution dictionary of the training set (high to low)
freq_dist_test: Dict[str, int] = dict(sorted(freq_dist_test.items(), key=lambda item: item[1], reverse=True)) # The sorted frequency distribution dictionary of the test set (high to low)

print(f'Total instances TRAINING SET: {num_train}, Total instances TEST SET: {num_test}.')
print()

for i,j in zip(freq_dist_train, freq_dist_test):
    print(f'Label: {i}, Absolute frequency TRAINING SET: {freq_dist_train[i]}, Relative frequency TRAINING SET: {(freq_dist_train[i]/num_train)*100:.2f}%.')
    print(f'Label: {j}, Absolute frequency TEST SET: {freq_dist_test[j]}, Relative frequency TEST SET: {(freq_dist_test[j]/num_test)*100:.2f}%.')
    print()
```

```
Total instances TRAINING SET: 203621, Total instances TEST SET: 46435.

Label: O, Absolute frequency TRAINING SET: 169578, Relative frequency TRAINING SET: 83.28%.
Label: O, Absolute frequency TEST SET: 38323, Relative frequency TEST SET: 82.53%.

Label: B-LOC, Absolute frequency TRAINING SET: 7140, Relative frequency TRAINING SET: 3.51%.
Label: B-LOC, Absolute frequency TEST SET: 1668, Relative frequency TEST SET: 3.59%.

Label: B-PER, Absolute frequency TRAINING SET: 6600, Relative frequency TRAINING SET: 3.24%.
Label: B-ORG, Absolute frequency TEST SET: 1661, Relative frequency TEST SET: 3.58%.

Label: B-ORG, Absolute frequency TRAINING SET: 6321, Relative frequency TRAINING SET: 3.10%.
Label: B-PER, Absolute frequency TEST SET: 1617, Relative frequency TEST SET: 3.48%.

Label: I-PER, Absolute frequency TRAINING SET: 4528, Relative frequency TRAINING SET: 2.22%.
Label: I-PER, Absolute frequency TEST SET: 1156, Relative frequency TEST SET: 2.49%.

Label: I-ORG, Absolute frequency TRAINING SET: 3704, Relative frequency TRAINING SET: 1.82%.
Label: I-ORG, Absolute frequency TEST SET: 835, Relative frequency TEST SET: 1.80%.

Label: B-MISC, Absolute frequency TRAINING SET: 3438, Relative frequency TRAINING SET: 1.69%.
Label: B-MISC, Absolute frequency TEST SET: 702, Relative frequency TEST SET: 1.51%.

Label: I-LOC, Absolute frequency TRAINING SET: 1157, Relative frequency TRAINING SET: 0.57%.
Label: I-LOC, Absolute frequency TEST SET: 257, Relative frequency TEST SET: 0.55%.

Label: I-MISC, Absolute frequency TRAINING SET: 1155, Relative frequency TRAINING SET: 0.57%.
Label: I-MISC, Absolute frequency TEST SET: 216, Relative frequency TEST SET: 0.47%.
```

Answer (b)

The training and test data show a distribution of labels similar between the two sets. For example, the most common label, 'O', which represents words not identified as named entities, makes up 83.28% of the training set and 82.53% of the test set, showing a slight variation but maintaining a consistent majority. Other labels such as 'B-LOC' (beginning of location), 'B-PER' (beginning of person), and 'B-ORG' (beginning of organization) also show similar proportions between the training and test sets, with 'B-LOC' appearing in 3.51% of the training data and 3.59% in the test data, 'B-PER' making up 3.24% of the training and slightly more, 3.48%, in the test set, and 'B-ORG' comprising 3.10% of the training set compared to 3.58% of the test set. When examining the balance of instances across the various NERC labels, There seems to be a significant imbalance, particularly with the overwhelming presence of the 'O' label compared to others. Such dominance suggests that instances of named entities like 'I-PER' (inside of person), 'I-ORG' (inside of organization), 'I-LOC' (inside of location), and 'I-MISC' (inside of miscellaneous) are far less frequent. For instance, 'I-PER' constitutes only 2.22% of the training set and sees a slight increase to 2.49% in the test set, while 'I-MISC' and 'I-LOC' are even less common, each making up less than 1% of the instances in both datasets.

[2 points] c) Concatenate the train and test features (the list of dictionaries) into one list. Load it using the *DictVectorizer*. Afterwards, split it back to training and test.

Tip: You've concatenated train and test into one list and then you've applied the DictVectorizer. The order of the rows is maintained. You can hence use an index (number of training instances) to split the_array back into train and test. Do NOT use: `from sklearn.model_selection import train_test_split` here.

In [5]:
```
vec = DictVectorizer()
features=training_features + test_features
the_array = vec.fit_transform(features)
# print(the_array)
# print(len(the_array))

n_train = len(training_features)

train_array = the_array[:n_train]
test_array = the_array[n_train:]
```

[4 points] d) Train the SVM using the train features and labels and evaluate on the test data. Provide a classification report (sklearn.metrics.classification_report). The train (*lin_clf.fit*) might take a while. On my computer, it took 1min 53s, which is acceptable. Training models normally takes much longer. If it takes more than 5 minutes, you can use a subset for training. Describe the results:

- Which NERC labels does the classifier perform well on? Why do you think this is the case?
- Which NERC labels does the classifier perform poorly on? Why do you think this is the case?

In [6]:
```
lin_clf = svm.LinearSVC()
```

In [7]:
```
lin_clf.fit(train_array, training_gold_labels)
test_predictions = lin_clf.predict(test_array)
report = classification_report(test_gold_labels, test_predictions)
print(report)
```

```
C:\Users\remia\anaconda3\Lib\site-packages\sklearn\svm\_classes.py:32: FutureWarning: The default value of `dual` will change from `True` to `'auto'` in 1.5. Set the value of `dual` explicitly to suppress the warning.
  warnings.warn(
              precision    recall  f1-score   support

       B-LOC       0.73      0.81      0.77      1668
      B-MISC       0.71      0.70      0.71       702
       B-ORG       0.68      0.58      0.63      1661
       B-PER       0.68      0.59      0.63      1617
       I-LOC       0.59      0.54      0.56       257
      I-MISC       0.56      0.59      0.58       216
       I-ORG       0.54      0.49      0.51       835
       I-PER       0.48      0.54      0.51      1156
           O       0.98      0.99      0.99     38323

    accuracy                           0.93     46435
   macro avg       0.66      0.65      0.65     46435
weighted avg       0.92      0.93      0.92     46435
```

Answer (d)

O: The high precision (0.98) and recall (0.99) values for the non-entity class suggest that the classifier performs well in identifying tokens that do not belong to any named entity category, indicating robust learning of non-entity patterns. This is likely due to their over-representation in the training set as evidenced by their provided data distribution.

Beginning entities typically mark the start of a named entity within a sequence of words. This often corresponds to the initial word of a named entity, which tends to have more distinctive naming patterns or keywords associated with specific entity types.

B-LOC: The relatively high precision (0.73) and recall (0.81) suggest that location entities often have distinct patterns or keywords, making them easier for the classifier to recognize accurately.
B-MISC: Miscellaneous entities cover a wide range of categories and can be ambiguous depending on the context. Regardless, the model manages to score a relatively high precision (0.71) and recall (0.70).
B-ORG: Organizations often have unique naming conventions or structures, contributing to the relatively high precision (0.68). However, the lower recall (0.58) suggests that some organization entities may have less distinct patterns.
B-PER: Person names may follow certain recognizable patterns, contributing to the high precision (0.68). However, the lower recall (0.59) suggests that some person entities may be less predictable in the data.

Inside entities often occur within the context of longer phrases or sentences, where the presence of multiple words and syntactic structures can introduce ambiguity and variability, thus resulting in lower precision and recall than beginning entities. They are also less represented in the data than their Beginning entities. This results in the following scores:
I-LOC: A precision of 0.59 and a recall of 0.54.
I-MISC: A precision of 0.56 and a recall of 0.59.
I-ORG: A precision of 0.54 and a recall of 0.49. The model seems to have the most difficulty with capturing all the Inside organization entities.
I-PER: A precision of 0.48 and a recall of 0.54. The model seems to have the most difficulty with correctly identifying this class.

[6 points] e) Train a model that uses the embeddings of these words as inputs. Test again on the same data as in 2d. Generate a classification report and compare the results with the classifier you built in 2d.

In [8]:
```
##### Adapt the path to point to your local copy of the Google embeddings model
word_embedding_model = gensim.models.KeyedVectors.load_word2vec_format(r'C:\Users\remia\Desktop\labs\ba-text-mining\lab_sessions\lab2\GoogleNews-vectors-negative300.bin', binary=True, limit=500000)

training_input= []

for token, pos, ne_label in train.iob_words():
    word=token #the next word from the tokenized text
    # we check if our word
    # is inside the model vocabulary (loaded with the Google word2vec embeddings)
    if word in word_embedding_model:
        # in this case the word was found and vector is assigned with its embedding vector as the value
        vector=word_embedding_model[word]
    else:
        # if the word does not exist in the embeddings vocabulary,
        # we create a vector with all zeros.
        # The Google word2vec model has 300 dimensions so we creat a vector with 300 zeros
        vector=[0]*300
        # print('This word is not in the word2vec vocabulary:', word)
    training_input.append(vector)
```

In [9]:
```
test_input = []
for token, pos, ne_label in test.iob_words():
    word=token #the next word from the tokenized text
    # we check if our word
    # is inside the model vocabulary (loaded with the Google word2vec embeddings)
    if word in word_embedding_model:
        # in this case the word was found and vector is assigned with its embedding vector as the value
        vector=word_embedding_model[word]
    else:
        # if the word does not exist in the embeddings vocabulary,
        # we create a vector with all zeros.
        # The Google word2vec model has 300 dimensions so we creat a vector with 300 zeros
        vector=[0]*300
        # print('This word is not in the word2vec vocabulary:', word)
    test_input.append(vector)
```

In [10]:
```
lin_clf_emb = svm.LinearSVC()
lin_clf_emb.fit(training_input, training_gold_labels)
```

```
C:\Users\remia\anaconda3\Lib\site-packages\sklearn\svm\_classes.py:32: FutureWarning: The default value of `dual` will change from `True` to `'auto'` in 1.5. Set the value of `dual` explicitly to suppress the warning.
  warnings.warn(
```

Out[10]:
```
▾ LinearSVC
LinearSVC()
```

In [11]:
```
pred_emb = lin_clf_emb.predict(test_input)
report_emb = classification_report(test_gold_labels, pred_emb ,digits = 2) # evaluation

# print(report_emb)
report_cols = report_emb.split('\n')[0]
report_emb_lines = report_emb.split('\n')[1:]
report_lines = report.split('\n')[1:]

print(report_cols)
for line_emb, line in zip(report_emb_lines, report_lines):
    if line_emb != '':
        print(line_emb, '(embedded)')
        print(line)
        print()
    else:
        print(line)
```

```
              precision    recall  f1-score   support

       B-LOC       0.76      0.80      0.78      1668 (embedded)
       B-LOC       0.73      0.81      0.77      1668

      B-MISC       0.72      0.69      0.71       702 (embedded)
      B-MISC       0.71      0.70      0.71       702

       B-ORG       0.70      0.63      0.67      1661 (embedded)
       B-ORG       0.68      0.58      0.63      1661

       B-PER       0.76      0.64      0.69      1617 (embedded)
       B-PER       0.68      0.59      0.63      1617

       I-LOC       0.52      0.41      0.46       257 (embedded)
       I-LOC       0.59      0.54      0.56       257

      I-MISC       0.59      0.52      0.56       216 (embedded)
      I-MISC       0.56      0.59      0.58       216

       I-ORG       0.50      0.34      0.41       835 (embedded)
       I-ORG       0.54      0.49      0.51       835

       I-PER       0.59      0.45      0.51      1156 (embedded)
       I-PER       0.48      0.54      0.51      1156

           O       0.97      0.99      0.98     38323 (embedded)
           O       0.98      0.99      0.99     38323


    accuracy                           0.92     46435 (embedded)
    accuracy                           0.93     46435

   macro avg       0.68      0.61      0.64     46435 (embedded)
   macro avg       0.66      0.65      0.65     46435

weighted avg       0.92      0.92      0.92     46435 (embedded)
weighted avg       0.92      0.93      0.92     46435
```

The comparison between the classification reports using crafted feature vectors and those obtained from embedding-based features does not show a notable improvement in the model's performance when leveraging word embeddings. However, for entity types such as B-LOC, B-MISC, B-ORG, and B-PER, there are various increases in precision, but a small decrease in recall for B-LOC, B-MISC, while B-ORG and B-PER show a relatively high increase in recall. This suggests that although the embedding model is slightly more precise in identifying the beginning of location, miscellaneous, organization, and person entities, the previous model is better at capturing more of the actual location and miscellaneous entities present in the data. The f1-score for B-MISC remains unchanged, however, the f1-scores for B-LOC, B-ORG, and B-PER show improvement. This improvement seems to highlight that utilizing embeddings may lead a model to better capture semantic and contextual information, to distinguish between these entity types more effectively.

For I-LOC, I-MISC, I-ORG, I-PER the performance varies, with the previous model generally achieving higher scores in most categories except for the precision of I-MISC and I-PER, where the embeddings model performs better. Meanwhile, the performance for the non-entity type 'O' remains high in both cases, with the previous model slightly outperforming the embeddings model

The previous model has a slightly higher overall accuracy compared to the embedding model (0.93 vs. 0.92) The macro avg, which considers the average performance across all categories, indicates that the previous model has a slightly better balance between precision and recall (f1 score) (0.65 vs 0.64), suggesting it may perform more consistently across different entity types. Both models have identical weighted averages for precision, recall, and f1-score, suggesting that, when taking into account the support (the number of true instances for each label), their performance is largely equivalent.

[Points: 10] Exercise 2 (NERC): feature inspection using the Annotated Corpus for Named Entity Recognition

[6 points] a. Perform the same steps as in the previous exercise. Make sure you end up for both the training part (*df_train*) and the test part (*df_test*) with:

- the features representation using **DictVectorizer**
- the NERC labels in a list

Please note that this is the same setup as in the previous exercise:

- load both train and test using:
  - list of dictionaries for features
  - list of NERC labels
- combine train and test features in a list and represent them using one hot encoding
- train using the training features and NERC labels

In [12]:
```
##### Adapt the path to point to your local copy of NERC_datasets
path = r'C:\Users\remia\Desktop\labs\ba-text-mining\lab_sessions\lab4\nerc_datasets\kaggle\ner_v2.csv'
kaggle_dataset = pandas.read_csv(path, on_bad_lines='skip')
```

In [13]:
```
len(kaggle_dataset)
```

Out[13]: 1050795

In [14]:
```
df_train = kaggle_dataset[:100000]
df_test = kaggle_dataset[100000:120000]

sentence_column=df_train['sentence_idx']

def string(x):
    return (str(x))

new=sentence_column.apply(string)
df_train['sentence_idx']=new
df_train.head()
```

```
C:\Users\remia\AppData\Local\Temp\ipykernel_7988\2272784935.py:10: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_train['sentence_idx']=new
```

Out[14]:

	id	lemma	next-lemma	next-next-lemma	next-next-pos	next-next-shape	next-next-word	next-pos	next-shape	next-word	...	prev-prev-lemma	prev-prev-pos	prev-prev-shape	prev-prev-word	prev-shape	prev-word	sentence_idx	shape	word	tag
0	0	thousand	of	demonstr	NNS	lowercase	demonstrators	IN	lowercase	of	...	__start2__	__START2__	wildcard	__START2__	wildcard	__START1__	1.0	capitalized	Thousands	O
1	1	of	demonstr	have	VBP	lowercase	have	NNS	lowercase	demonstrators	...	__start1__	__START1__	wildcard	__START1__	capitalized	Thousands	1.0	lowercase	of	O
2	2	demonstr	have	march	VBN	lowercase	marched	VBP	lowercase	have	...	thousand	NNS	capitalized	Thousands	lowercase	of	1.0	lowercase	demonstrators	O
3	3	have	march	through	IN	lowercase	through	VBN	lowercase	marched	...	of	IN	lowercase	of	lowercase	demonstrators	1.0	lowercase	have	O
4	4	march	through	london	NNP	capitalized	London	IN	lowercase	through	...	demonstr	NNS	lowercase	demonstrators	lowercase	have	1.0	lowercase	marched	O

5 rows × 25 columns

drop 'tag' column, the 'tag' column contains the NERC labels

drop 'id' column, the 'id' column does not offer any predictive value

In [15]:
```
#Split df into labels and features arrays

train_labels = df_train['tag'].values
features_train = df_train.drop('tag', axis=1)
features_train = features_train.drop('id', axis=1)
features_dict_train = features_train.to_dict(orient='records')

test_labels = df_test['tag'].values
features_test = df_test.drop('tag', axis=1)
features_test = features_test.drop('id', axis=1)
features_dict_test = features_test.to_dict(orient='records')
```

In [16]:
```
vec2 = DictVectorizer()
features2=features_dict_train + features_dict_test
print(features2[1])
the_array = vec2.fit_transform(features2)
print(the_array)

n_train = len(features_dict_train)

train_array = the_array[:n_train]
test_array = the_array[n_train:]

print(train_array.shape)
print(test_array.shape)
```

```
{'lemma': 'of', 'next-lemma': 'demonstr', 'next-next-lemma': 'have', 'next-next-pos': 'VBP', 'next-next-shape': 'lowercase', 'next-next-word': 'have', 'next-pos': 'NNS', 'next-shape': 'lowercase', 'next-word': 'demonstrators', 'pos': 'IN', 'prev-iob': 'O', 'prev-lemma': 'thousand', 'prev-pos': 'NNS', 'prev-prev-iob': '__START1__', 'prev-prev-lemma': '__start1_
_', 'prev-prev-pos': '__START1__', 'prev-prev-shape': 'wildcard', 'prev-prev-word': '__START1__', 'prev-shape': 'capitalized', 'prev-word': 'Thousands', 'sentence_idx': '1.0', 'shape': 'lowercase', 'word': 'of'}
  (0, 7329)	1.0
  (0, 13352)	1.0
  (0, 18437)	1.0
  (0, 24093)	1.0
  (0, 24122)	1.0
  (0, 29674)	1.0
  (0, 35348)	1.0
  (0, 35386)	1.0
  (0, 43763)	1.0
  (0, 46843)	1.0
  (0, 46883)	1.0
  (0, 47411)	1.0
  (0, 55054)	1.0
  (0, 55074)	1.0
  (0, 55577)	1.0
  (0, 62960)	1.0
  (0, 62973)	1.0
  (0, 67110)	1.0
  (0, 74411)	1.0
  (0, 78696)	1.0
  (0, 86270)	1.0
  (0, 90816)	1.0
  (0, 94729)	1.0
  (1, 5280)	1.0
  (1, 10446)	1.0
  :	:
  (119998, 90819)	1.0
  (119998, 99619)	1.0
  (119999, 4721)	1.0
  (119999, 11904)	1.0
  (119999, 21330)	1.0
  (119999, 24081)	1.0
  (119999, 24122)	1.0
  (119999, 32344)	1.0
  (119999, 35371)	1.0
  (119999, 35386)	1.0
  (119999, 42679)	1.0
  (119999, 46843)	1.0
  (119999, 46882)	1.0
  (119999, 52165)	1.0
  (119999, 55023)	1.0
  (119999, 55072)	1.0
  (119999, 60311)	1.0
  (119999, 62934)	1.0
  (119999, 62967)	1.0
  (119999, 71620)	1.0
  (119999, 74405)	1.0
  (119999, 83207)	1.0
  (119999, 86269)	5469.0
  (119999, 90819)	1.0
  (119999, 99202)	1.0
(100000, 102681)
(20000, 102681)
```

[4 points] b. Train and evaluate the model and provide the classification report:

- use the SVM to predict NERC labels on the test data
- evaluate the performance of the SVM on the test data

Analyze the performance per NERC label.

In [17]:
```
lin_clf = svm.LinearSVC()
lin_clf.fit(train_array, train_labels)
test_predictions = lin_clf.predict(test_array)
report = classification_report(test_labels, test_predictions, zero_division = 0)
print(report)
```

```
C:\Users\remia\anaconda3\Lib\site-packages\sklearn\svm\_classes.py:32: FutureWarning: The default value of `dual` will change from `True` to `'auto'` in 1.5. Set the value of `dual` explicitly to suppress the warning.
  warnings.warn(
              precision    recall  f1-score   support

       B-art       0.00      0.00      0.00         4
       B-eve       0.00      0.00      0.00         0
       B-geo       0.87      0.88      0.87       741
       B-gpe       0.90      0.93      0.92       296
       B-nat       0.80      0.50      0.62         8
       B-org       0.77      0.67      0.72       397
       B-per       0.81      0.83      0.82       333
       B-tim       0.95      0.84      0.89       393
       I-geo       0.97      0.96      0.97       156
       I-gpe       1.00      1.00      1.00         2
       I-nat       1.00      1.00      1.00         4
       I-org       0.95      0.93      0.94       321
       I-per       0.95      0.98      0.96       319
       I-tim       1.00      0.86      0.93       108
           O       0.99      0.99      0.99     16918

    accuracy                           0.97     20000
   macro avg       0.80      0.76      0.77     20000
weighted avg       0.97      0.97      0.97     20000
```

B-art shows very low performance with all metrics at 0.00, indicating that the model failed to correctly identify any 'Art' entities. This could be due to a very low sample size (support = 4) or a lack of relevant features for these entities in the training data.

B-eve does not seem to have any instances in the sample, which indicates that the model did not have a chance to predict this category in the dataset provided.

B-geo shows high precision (0.87) and recall (0.88) leading to a high f1-score (0.87), indicating strong performance in identifying geographical entities, likely due to a larger support (741).

B-gpe has a very good performance with precision at 0.90, recall at 0.93, and f1-score at 0.92, this could be due to the relatively large number of instances (296). This suggests the model is well-tuned for recognizing geopolitical entities.

B-nat shows moderate performance with precision at 0.80, recall at 0.50, and f1-score at 0.62. This suggests the model can recognize natural phenomena to some extent but struggles with consistency, likely due to very low support (8).

B-org seems to have decent performance with precision at 0.77, recall at 0.67, and f1-score at 0.72, indicating the model is fairly reliable at identifying organizations from the text.

B-per has a strong performance with precision at 0.81, recall at 0.83, and f1-score at 0.82. This shows the model is effective at identifying the names of individuals.

B-tim has excellent precision (0.95) and good recall (0.84) leading to a high f1-score (0.89), indicating strong performance in recognizing time-related expressions. This could be due to the relatively large number of instances (393)

I-geo, I-gpe, I-nat, I-org, I-per, and I-tim: These 'Inside' labels generally show high precision, recall, and f1-scores, indicating that the model is very effective at continuing to recognize entities once they have been identified as beginning. Notably, I-gpe and I-nat have perfect scores, but their support is very low, this could be due to limited testing instances.

O has almost perfect scores in precision, recall, and f1-score (all 0.99), supported by a very large number of instances (16918). This suggests the model is extremely effective at identifying tokens that are not named entities.

End of this notebook