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

[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',
```
```

````
....
]
```

from nltk.corpus.reader import ConllCorpusReader
### Adapt the path to point to the CONLL2003 folder on your local machine
#train = ConllCorpusReader('C:/Users/coolm/OneDrive - Vrije Universiteit Amsterdam/University documents/Third year/Text Mining/Assignments/ba-text-mining-master/ba-text-mining-master/lab_sessions/lab4/CONLL2003', 'train.txt', ['words', 'pos', 'ignore', 'chunk'])
train = ConllCorpusReader('C:/Users/aewse/Documents/GitHub/ba-text-mining/lab_sessions/lab4/CONLL2003/CONLL2003', 'train.txt', ['words', 'pos', 'ignore', 'chunk'])
training_features = []
training_gold_labels = []

for token, pos, ne_label in train.iob_words():
    a_dict = {
      'words': token,
      'pos': pos
    }
    training_features.append(a_dict)
    training_gold_labels.append(ne_label)



### Adapt the path to point to the CONLL2003 folder on your local machine
#train = ConllCorpusReader('C:/Users/coolm/OneDrive - Vrije Universiteit Amsterdam/University documents/Third year/Text Mining/Assignments/ba-text-mining-master/ba-text-mining-master/lab_sessions/lab4/CONLL2003', 'test.txt', ['words', 'pos', 'ignore', 'chunk'])
train = ConllCorpusReader('C:/Users/aewse/Documents/GitHub/ba-text-mining/lab_sessions/lab4/CONLL2003/CONLL2003', 'test.txt', ['words', 'pos', 'ignore', 'chunk'])


test_features = []
test_gold_labels = []
for token, pos, ne_label in train.iob_words():
    a_dict = {
        'words': token,
        'pos': pos
    }
    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:

```
from collections import Counter

#test = ConllCorpusReader('C:/Users/coolm/OneDrive - Vrije
Universiteit Amsterdam/University documents/Third year/Text
Mining/Assignments/ba-text-mining-master/ba-text-mining-master/lab_ses
sions/lab4/CONLL2003', 'test.txt', ['words', 'pos', 'ignore',
'chunk'])
test = ConllCorpusReader('C:/Users/aewse/Documents/GitHub/ba-text-
mining/lab_sessions/lab4/CONLL2003/CONLL2003', 'test.txt', ['words',
'pos', 'ignore', 'chunk'])


#for _, _, ne_label in train.iob_words():
#    training_gold_labels.append(ne_label)

#for _, _, ne_label in test.iob_words():
#    test_gold_labels.append(ne_label)

train_instances = len(training_gold_labels)
test_instances = len(test_gold_labels)

train_label_counts = Counter(training_gold_labels)
test_label_counts = Counter(test_gold_labels)

print(f"Number of instances in training data: {train_instances}")
print(f"Number of instances in test data: {test_instances}")

print("Training data NERC label distribution:")
for label, count in train_label_counts.items():
    print(f"{label}: {count}")

print("Test data NERC label distribution:")
for label, count in test_label_counts.items():
    print(f"{label}: {count}")

print("Train vs Test:")
for label in
set(train_label_counts.keys()).union(test_label_counts.keys()):
    train_count = train_label_counts.get(label, 0)
    test_count = test_label_counts.get(label, 0)
```

```
    print(f"{label}: Train = {train_count}, Test = {test_count}")
```

```
Number of instances in training data: 203621
Number of instances in test data: 46435
Training data NERC label distribution:
B-ORG: 6321
O: 169578
B-MISC: 3438
B-PER: 6600
I-PER: 4528
B-LOC: 7140
I-ORG: 3704
I-MISC: 1155
I-LOC: 1157
Test data NERC label distribution:
O: 38323
B-LOC: 1668
B-PER: 1617
I-PER: 1156
I-LOC: 257
B-MISC: 702
I-MISC: 216
B-ORG: 1661
I-ORG: 835
Train vs Test:
I-ORG: Train = 3704, Test = 835
B-PER: Train = 6600, Test = 1617
I-LOC: Train = 1157, Test = 257
B-MISC: Train = 3438, Test = 702
I-PER: Train = 4528, Test = 1156
B-LOC: Train = 7140, Test = 1668
O: Train = 169578, Test = 38323
I-MISC: Train = 1155, Test = 216
B-ORG: Train = 6321, Test = 1661
```

The training and test data are balanced in terms of label distribution so they have an equal number of NERC label instances.

The class '0' is the most frequent class making which causes an imbalance in the overall dataset and as a result can lead a bias towards predicting 0.

There aren't significant differences between the training and testing datasets so the model is unlikely to show any distribution shifts with the entity types.

[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.

```
from sklearn.feature_extraction import DictVectorizer

vec = DictVectorizer()
the_array = vec.fit_transform(training_features + test_features)
train_instances = len(training_features)
X_train = the_array[:train_instances]
X_test = the_array[train_instances:]

y_train = training_gold_labels
y_test = test_gold_labels
```

[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?

```
from sklearn import svm

lin_clf = svm.LinearSVC()

##### [ YOUR CODE SHOULD GO HERE ]
# lin_clf.fit( # your code here

lin_clf.fit(X_train, y_train) #train

c:\Users\aewse\anaconda3\Lib\site-packages\sklearn\svm\_base.py:1249:
ConvergenceWarning: Liblinear failed to converge, increase the number
of iterations.
  warnings.warn(

LinearSVC()

y_pred = lin_clf.predict(X_test)


from sklearn.metrics import classification_report


print(classification_report(y_test, y_pred))

              precision    recall  f1-score   support

       B-LOC       0.81      0.77      0.79      1668
```

```
      B-MISC       0.78      0.66      0.71       702
       B-ORG       0.79      0.52      0.62      1661
       B-PER       0.86      0.44      0.58      1617
       I-LOC       0.62      0.53      0.57       257
      I-MISC       0.59      0.59      0.59       216
       I-ORG       0.66      0.48      0.55       835
       I-PER       0.33      0.87      0.48      1156
           O       0.99      0.98      0.98     38323

    accuracy                           0.92     46435
   macro avg       0.71      0.65      0.65     46435
weighted avg       0.94      0.92      0.92     46435
```

[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.

```
import gensim
import numpy as np
from sklearn.svm import LinearSVC
from sklearn.metrics import classification_report
from nltk.corpus.reader import ConllCorpusReader

# Load train data
train = ConllCorpusReader('C:/Users/aewse/Documents/GitHub/ba-text-
mining/lab_sessions/lab4/CONLL2003/CONLL2003',
                          'train.txt', ['words', 'pos', 'ignore',
'chunk'])

# Load test data
test = ConllCorpusReader('C:/Users/aewse/Documents/GitHub/ba-text-
mining/lab_sessions/lab4/CONLL2003/CONLL2003',
                         'test.txt', ['words', 'pos', 'ignore',
'chunk'])

train_sentences = list(train.sents())[:1000]
test_sentences = list(test.sents())[:200]     #cliped samples bcs it
was taking too long

word2vec_model = gensim.models.Word2Vec(sentences=train_sentences,
vector_size=100, window=5, min_count=1, workers=4)

def get_word_embedding(word, model, vector_size=100):
    if word in model.wv:
        return model.wv[word]
    return np.zeros(vector_size)  #0 if unknown

X_train_embed = np.array([get_word_embedding(word, word2vec_model) for
word, _, _ in train.iob_words()])
```

```
X_test_embed = np.array([get_word_embedding(word, word2vec_model) for 
word, _, _ in test.iob_words()]) #embedded rep

y_train = [label for _, _, label in train.iob_words()]
y_test = [label for _, _, label in test.iob_words()] #get target 
labels


from sklearn.preprocessing import StandardScaler

#normalize
scaler = StandardScaler()
X_train_embed_scaled = scaler.fit_transform(X_train_embed)
X_test_embed_scaled = scaler.transform(X_test_embed)


svm_model = LinearSVC(class_weight='balanced')
svm_model.fit(X_train_embed_scaled, y_train)
y_pred_embed = svm_model.predict(X_test_embed_scaled)

print(classification_report(y_test, y_pred_embed, 
zero_division='warn'))
```

```
              precision    recall  f1-score   support

       B-LOC       0.28      0.30      0.29      1668
      B-MISC       0.12      0.25      0.16       702
       B-ORG       0.09      0.01      0.02      1661
       B-PER       0.17      0.05      0.08      1617
       I-LOC       0.08      0.29      0.12       257
      I-MISC       0.09      0.47      0.15       216
       I-ORG       0.07      0.02      0.04       835
       I-PER       0.02      0.00      0.00      1156
           O       0.85      0.88      0.86     38323

    accuracy                           0.75     46435
   macro avg       0.20      0.25      0.19     46435
weighted avg       0.72      0.75      0.73     46435
```

The first model achieved an accuracy of 92%, showing higher performance over the second classifier's 72%. However, this value was largely influenced by the majority class ("O") present in the dataset. While the second model shows a lower accuracy, data imbalance should be considered.

The first classifier shows good precision (0.99) and recall (0.98) for the majority class again, whereas it struggles for the minority classes. B-LOC, B-MISC, B-ORG, B-PER, I-LOC, I-MISC, I-ORG all show decent performance in terms of precision but poor recall, suggesting that the model incorrectly classifies them as part of the majority class. Meanwhile, I-PER has a low precision but high recall score, indicating a high rate of false positives.

The second model has a lower precision and recall for the majority class compared to the first, however the scores themselves are still high (0.85 precision and 0.88 recall). Performance seems quite poor for minority classes however, with B-LOC, B-MISC, B-ORG, and I-PER all having precision and recall scores of 0.3 or lower, with I-PER even having a 0.02 precision and 0.00 recall. This struggle with minority classes is possibly due to the model being overfitted on the majority class, or insufficiently trained on the minority classes.

The better performance of the first model with relation to precision and recall is echoed by its higher f1 score, suggesting that it also does a better job at balancing these metrics than the second model. It also shows better f1 scores across the minority classes, suggesting it handles imbalanced data better.

The first classifier is the better performing of the two, as it seems to outperform the second on every metric, and balances precision and recall better (as shown through the f1 score) across both majority and minority classes. By comparison, the second model seems to struggle significantly with minority classes, hindering its overall performance for this task.

[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

```
import pandas

##### Adapt the path to point to your local copy of NERC_datasets
path =
'D:/antal/text_mining_collab/ba-text-mining/lab_sessions/lab4/ner_data
set.csv'
kaggle_dataset = pandas.read_csv(path,
on_bad_lines='skip',encoding="ISO-8859-1")

len(kaggle_dataset)

df_train = kaggle_dataset[:100000]
df_test = kaggle_dataset[100000:120000]
print(len(df_train), len(df_test))

100000 20000
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report
from sklearn.svm import LinearSVC
from sklearn.feature_extraction import DictVectorizer

train_features = 
df_train.drop(columns=["Tag"]).fillna("UNKNOWN").to_dict(orient="recor
ds")
test_features = 
df_test.drop(columns=["Tag"]).fillna("UNKNOWN").to_dict(orient="record
s")


y_train = df_train["Tag"].fillna("UNKNOWN").tolist()
y_test = df_test["Tag"].fillna("UNKNOWN").tolist()


vectorizer = DictVectorizer(sparse=True) #was too big if not sparse
X_train = vectorizer.fit_transform(train_features)
X_test = vectorizer.transform(test_features)

scaler = StandardScaler(with_mean=False)
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)


svm_model = LinearSVC(class_weight='balanced', max_iter=5000)
svm_model.fit(X_train_scaled, y_train)
```

```
c:\Users\antal\anaconda3\envs\text_mining\Lib\site-packages\sklearn\
svm\_base.py:1249: ConvergenceWarning: Liblinear failed to converge, 
increase the number of iterations.
  warnings.warn(

LinearSVC(class_weight='balanced', max_iter=5000)
```

[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.

```
y_pred = svm_model.predict(X_test_scaled)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

       B-art       0.01      0.50      0.02         4
       B-eve       0.00      0.00      0.00         0
       B-geo       0.80      0.73      0.76       741
```

```
         B-gpe       0.95      0.92      0.93       296
         B-nat       0.70      0.88      0.78         8
         B-org       0.60      0.48      0.53       397
         B-per       0.73      0.54      0.62       333
         B-tim       0.74      0.79      0.76       393
         I-art       0.00      0.00      0.00         0
         I-eve       0.00      0.00      0.00         0
         I-geo       0.64      0.53      0.58       156
         I-gpe       0.04      0.50      0.08         2
         I-nat       0.80      1.00      0.89         4
         I-org       0.30      0.60      0.40       321
         I-per       0.60      0.53      0.56       319
         I-tim       0.19      0.45      0.27       108
             O       0.99      0.96      0.97     16918

      accuracy                           0.91     20000
     macro avg       0.48      0.55      0.48     20000
  weighted avg       0.94      0.91      0.92     20000


c:\Users\antal\anaconda3\envs\text_mining\Lib\site-packages\sklearn\
metrics\_classification.py:1565: UndefinedMetricWarning: Recall is
ill-defined and being set to 0.0 in labels with no true samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
c:\Users\antal\anaconda3\envs\text_mining\Lib\site-packages\sklearn\
metrics\_classification.py:1565: UndefinedMetricWarning: Recall is
ill-defined and being set to 0.0 in labels with no true samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
c:\Users\antal\anaconda3\envs\text_mining\Lib\site-packages\sklearn\
metrics\_classification.py:1565: UndefinedMetricWarning: Recall is
ill-defined and being set to 0.0 in labels with no true samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
```

The model performs well for frequent categories and is reasonably effective for people, time expressions, and organizations. Performance drops significantly for rarer labels, likely due to limited training examples. Some categories don't appear at all, making prediction impossible. In general it works, but is still limited for a lot of targets.

B-art: Low performance, likely due to rarity of occurrence. B-eve: No examples, so the model can't predict it. B-geo: Strong performance, but misses some. B-gpe: Strong performance, predicts well. B-nat: Low performance, likely due to rarity of occurrence. B-org: Decent, but misses a lot. B-per: Decent, but misses a lot. B-tim: Decent, predicts well. I-art: No examples, so the model can't predict it. I-eve: No examples, so the model can't predict it. I-geo: Low performance, misses a lot. I-gpe: Low performance, likely due to rarity of occurrence. I-nat: Low performance, likely due to rarity of occurrence. I-org: Low performance, misses a lot. I-per: Decent, but misses a lot. I-tim: Low performance, misses a lot. O: Very good performance, makes sense for the model to adjust to these very well because it occurs so much.

End of this notebook