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',
....
]
```
```

In [40]:
```
```
from nltk.corpus.reader import ConllCorpusReader
import os
```

```
### Adapt the path to point to the CONLL2003 folder on your local machine
train = ConllCorpusReader(os.path.abspath('CONLL2003'), 'train.txt', ['words'
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)
```

In [41]:
```
training_gold_labels[:10]
```

Out[41]: ['B-ORG', 'O', 'B-MISC', 'O', 'O', 'O', 'B-MISC', 'O', 'O', 'B-PER']

In [42]:
```
### Adapt the path to point to the CONLL2003 folder on your local machine
test = ConllCorpusReader(os.path.abspath('CONLL2003'), 'test.txt', ['words',

test_features = []
test_gold_labels = []
for token, pos, ne_label in test.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:

In [43]:
```
from collections import Counter

print("Number of instances in training set: ", len(training_features), len(tr
print("Number of instances in test set: ", len(test_features), len(test_gold_

counter_train = Counter(training_gold_labels)
counter_test = Counter(test_gold_labels)

counter_train_dict = dict(counter_train)
counter_test_dict = dict(counter_test)

# Sort counter_train_dict
counter_train_dict = dict(sorted(counter_train_dict.items(), key=lambda x: x[
```

```
counter_test_dict = dict(sorted(counter_test_dict.items(), key=lambda x: x[1]

print("Frequency distribution of NERC labels")
print("(Training)")
print(counter_train_dict)
print("(Test)")
print(counter_test_dict)

print("Frequency distribution adjusted for size of the sets for easy comparis

multiple = len(training_gold_labels) / len(test_gold_labels)
print("multiple", multiple)

counter_test_dict_adjusted = {k: round(v * multiple, 0) for k, v in counter_t

print("(Training)")
print(counter_train_dict)

print("(Test)")
print(counter_test_dict_adjusted)
```

```
Number of instances in training set:  203621 203621
Number of instances in test set:  46435 46435
Frequency distribution of NERC labels
(Training)
{'O': 169578, 'B-LOC': 7140, 'B-PER': 6600, 'B-ORG': 6321, 'I-PER': 4528, 'I-O
RG': 3704, 'B-MISC': 3438, 'I-LOC': 1157, 'I-MISC': 1155}
(Test)
{'O': 38323, 'B-LOC': 1668, 'B-ORG': 1661, 'B-PER': 1617, 'I-PER': 1156, 'I-OR
G': 835, 'B-MISC': 702, 'I-LOC': 257, 'I-MISC': 216}
Frequency distribution adjusted for size of the sets for easy comparison purpo
ses
multiple 4.385075912565952
(Training)
{'O': 169578, 'B-LOC': 7140, 'B-PER': 6600, 'B-ORG': 6321, 'I-PER': 4528, 'I-O
RG': 3704, 'B-MISC': 3438, 'I-LOC': 1157, 'I-MISC': 1155}
(Test)
{'O': 168049.0, 'B-LOC': 7314.0, 'B-ORG': 7284.0, 'B-PER': 7091.0, 'I-PER': 50
69.0, 'I-ORG': 3662.0, 'B-MISC': 3078.0, 'I-LOC': 1127.0, 'I-MISC': 947.0}
```

For the purposes of comparing how balanced the data is I multiplied the test dataset by a number so that the test and train set are the same proportion size thing for easy comparison purposes.

In both the training and test set there are much more O labels than all other labels combined (but that is to be expected since most words aren't named entities)

The other labels are not very balanced, they range from about 7300 instances having the label to about 1000. The three most common named entities (LOC, PER, ORG) are quite balanced, randing from about 7200 to 6300 in the training set and about 7300 to 7000 in the test set. The "inner" (I-) version of those named entities all have lower frequency and B-MISC and I-MISC have significantly lower frequency than the most common named entities.

Between the training and test set the corresponding labels are balanced. They differ by at most around 15% and generally differ by around 5%.

[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 [44]:
```
from sklearn.feature_extraction import DictVectorizer
```

In [45]:
```
train_test_set = training_features + test_features
```

In [46]:
```
vec = DictVectorizer()
train_test_vector = vec.fit_transform(train_test_set).toarray()
```

In [47]:
```
train_vector = train_test_vector[:len(training_features)]
test_vector = train_test_vector[len(training_features):]
```

[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 [48]:
```
from sklearn import svm
```

In [49]:
```
lin_clf = svm.LinearSVC()
```

In [50]:
```
lin_clf.fit(train_vector, training_gold_labels)
```

Out[50]:
```
LinearSVC()
```

[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 [37]:
```
import gensim
import os
##### Adapt the path to point to your local copy of the Google embeddings mod
word_embedding_model = gensim.models.KeyedVectors.load_word2vec_format(os.pat
```

```
/opt/anaconda3/envs/TextMining/lib/python3.8/site-packages/gensim/similaritie
s/__init__.py:15: UserWarning: The gensim.similarities.levenshtein submodule i
s disabled, because the optional Levenshtein package <https://pypi.org/projec
t/python-Levenshtein/> is unavailable. Install Levenhstein (e.g. `pip install
python-Levenshtein`) to suppress this warning.
  warnings.warn(msg)
```

In [52]:
```
train_vector_embedding = []
```

```
train_gold_labels_embedding = []

for token, pos, ne_label in train.iob_words():
    if token == '' or token == 'DOCSTART':
        continue

    if token in word_embedding_model:
        vector = word_embedding_model[token]
    else:
        vector = [0] * 300

    train_vector_embedding.append(vector)
    train_gold_labels_embedding.append(ne_label)

test_vector_embedding = []
test_gold_labels_embedding = []

for token, pos, ne_label in test.iob_words():
    if token == '' or token == 'DOCSTART':
        continue

    if token in word_embedding_model:
        vector = word_embedding_model[token]
    else:
        vector = [0] * 300

    test_vector_embedding.append(vector)
    test_gold_labels_embedding.append(ne_label)
```

In [54]:
```
lin_clf_embedding = svm.LinearSVC()
lin_clf_embedding.fit(train_vector_embedding, train_gold_labels_embedding)
```

Out[54]:
```
LinearSVC()
```

In [55]:
```
lin_clf_prediction = lin_clf.predict(test_vector)
lin_clf_embedding_prediction = lin_clf_embedding.predict(test_vector_embeddin
```

In [57]:
```
from sklearn.metrics import classification_report

print("Without embedding")
print(classification_report(test_gold_labels, lin_clf_prediction))
```

```
Without embedding
              precision    recall  f1-score   support

       B-LOC       0.81      0.78      0.79      1668
      B-MISC       0.78      0.66      0.72       702
       B-ORG       0.79      0.52      0.63      1661
       B-PER       0.86      0.44      0.58      1617
       I-LOC       0.62      0.53      0.57       257
      I-MISC       0.57      0.59      0.58       216
       I-ORG       0.70      0.47      0.56       835
       I-PER       0.33      0.87      0.48      1156
           O       0.98      0.98      0.98     38323

    accuracy                           0.92     46435
   macro avg       0.72      0.65      0.65     46435
weighted avg       0.94      0.92      0.92     46435
```

In [58]:
```
print("With embedding")
```

```
print(classification_report(test_gold_labels_embedding, lin_clf_embedding_pre
```

```
With embedding
              precision    recall  f1-score   support

       B-LOC       0.76      0.80      0.78      1668
      B-MISC       0.72      0.70      0.71       702
       B-ORG       0.69      0.64      0.66      1661
       B-PER       0.75      0.67      0.71      1617
       I-LOC       0.51      0.42      0.46       257
      I-MISC       0.60      0.54      0.57       216
       I-ORG       0.48      0.33      0.39       835
       I-PER       0.59      0.50      0.54      1156
           O       0.97      0.99      0.98     38323

    accuracy                           0.93     46435
   macro avg       0.68      0.62      0.64     46435
weighted avg       0.92      0.93      0.92     46435
```

The models with and without embeddings are approximately equally good. Some metrics are higher in one and lower in the other model. Generally the metrics are within a few percentage points of each other. There is probably no statistically significant difference.

[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 [61]:
```
import pandas as pd
```

In [64]:
```
##### Adapt the path to point to your local copy of NERC_datasets
kaggle_dataset = pd.read_csv(os.path.abspath('ner_dataset.csv'), encoding='la
```

In [65]:
```
len(kaggle_dataset)
```

Out[65]:
```
1048575
```

In [66]:
```
df_train = kaggle_dataset[:100000]
df_test = kaggle_dataset[100000:120000]
print(len(df_train), len(df_test))
```

```
100000 20000
```

In [70]:
```
training_X = []
training_Y = []

for _, row in df_train.iterrows():
    a_dict = {
        'words': row['Word'],
        'pos': row['POS']
    }

    training_X.append(a_dict)
    training_Y.append(row['Tag'])


test_X = []
test_Y = []

for _, row in df_test.iterrows():
    a_dict = {
        'words': row['Word'],
        'pos': row['POS']
    }

    test_X.append(a_dict)
    test_Y.append(row['Tag'])
```

In [73]:
```
combined_X = training_X + test_X
```

In [74]:
```
dict_vectorizer = DictVectorizer()

combined_X_vector = dict_vectorizer.fit_transform(combined_X).toarray()
```

In [75]:
```
training_X = combined_X_vector[:len(training_X)]
test_X = combined_X_vector[len(training_X):]
```

[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 [77]:
```
lin_svm = svm.LinearSVC()
lin_svm.fit(training_X, training_Y)
```

Out[77]:
```
LinearSVC()
```

In [78]:
```
predictions = lin_svm.predict(test_X)
```

In [79]:
```
print(classification_report(test_Y, predictions))
```

```
              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       0.80      0.76      0.78       741
       B-gpe       0.96      0.92      0.94       296
       B-nat       1.00      0.50      0.67         8
       B-org       0.64      0.51      0.57       397
       B-per       0.81      0.53      0.64       333
       B-tim       0.91      0.76      0.83       393
       I-art       0.00      0.00      0.00         0
       I-eve       0.00      0.00      0.00         0
       I-geo       0.74      0.50      0.60       156
       I-gpe       1.00      0.50      0.67         2
       I-nat       0.80      1.00      0.89         4
       I-org       0.65      0.44      0.53       321
       I-per       0.42      0.90      0.57       319
       I-tim       0.41      0.08      0.14       108
           O       0.98      0.99      0.99     16918

    accuracy                           0.94     20000
   macro avg       0.60      0.49      0.52     20000
weighted avg       0.95      0.94      0.94     20000
```

```
/opt/anaconda3/envs/TextMining/lib/python3.8/site-packages/sklearn/metrics/_cl
assification.py:1318: UndefinedMetricWarning: Precision and F-score are ill-de
fined and being set to 0.0 in labels with no predicted samples. Use `zero_divi
sion` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/opt/anaconda3/envs/TextMining/lib/python3.8/site-packages/sklearn/metrics/_cl
assification.py:1318: UndefinedMetricWarning: Recall and F-score are ill-defin
ed and being set to 0.0 in labels with no true samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/opt/anaconda3/envs/TextMining/lib/python3.8/site-packages/sklearn/metrics/_cl
assification.py:1318: UndefinedMetricWarning: Precision and F-score are ill-de
fined and being set to 0.0 in labels with no predicted samples. Use `zero_divi
sion` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/opt/anaconda3/envs/TextMining/lib/python3.8/site-packages/sklearn/metrics/_cl
assification.py:1318: UndefinedMetricWarning: Recall and F-score are ill-defin
ed and being set to 0.0 in labels with no true samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/opt/anaconda3/envs/TextMining/lib/python3.8/site-packages/sklearn/metrics/_cl
assification.py:1318: UndefinedMetricWarning: Precision and F-score are ill-de
fined and being set to 0.0 in labels with no predicted samples. Use `zero_divi
sion` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/opt/anaconda3/envs/TextMining/lib/python3.8/site-packages/sklearn/metrics/_cl
assification.py:1318: UndefinedMetricWarning: Recall and F-score are ill-defin
ed and being set to 0.0 in labels with no true samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Several of the NERC labels have 0.0 on all metrics, probably because those labels aren't present in the test set.

The weighted average is high because the precision, recall and F1 score for the label O is high and that class is highly overrepresented, so the macro average is a more true measure of the model's performance.

The macro average precision is 0.6, and the macro aveerage recall and F1 score are around 0.5. Overall that's not very good but it's not terrible.

The different metrics for the different labels are all over the place. They vary greatly from label to label and within a label the precision, recall and F1 score are often very different.

For example B-gpe has above 90% for all three metrics, and B-org has around 60% for all three metrics. I-per has 0.41 precision and 0.90 recall.

End of this notebook