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 [ ]:
```
from nltk.corpus.reader import ConllCorpusReader
### Adapt the path to point to the CONLL2003 folder on your local machine
train = ConllCorpusReader('nerc_datasets/CONLL2003', 'train.txt', ['words', 'pos',
training_features = []
training_gold_labels = []

for token, pos, ne_label in train.iob_words():
    training_features.append({
        'word': token,
        'pos': pos
    })
    training_gold_labels.append(ne_label)
```

In [ ]:
```
### Adapt the path to point to the CONLL2003 folder on your local machine
test = ConllCorpusReader('nerc_datasets/CONLL2003', 'test.txt', ['words', 'pos', '

testing_features = []
testing_gold_labels = []
for token, pos, ne_label in test.iob_words():
    testing_features.append({
        'word': token,
        'pos': pos
    })
    testing_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 [ ]:
```
from collections import Counter

for tt in ['train', 'test']:
    print(f"{tt.title()}ing instances: {len(globals()[f'{tt}ing_features'])}")
    print("Of which the label distribution is:")
    for l, i in Counter(globals()[f"{tt}ing_gold_labels"]).items():
        #store the label distribution in a dictionary
        globals()[f"{tt}_label_distribution"] = Counter(globals()[f"{tt}ing_gold_l
    #sort the dictionary by value and print it
    for l, i in sorted(globals()[f"{tt}_label_distribution"].items(), key=lambda x
        print(f"\t {l:7}: {i:7} ({i/len(globals()[f'{tt}ing_features'])*100:.2f}%)
    print()
```

```
Training instances: 203621
Of which the label distribution is:
         O      :  169578 (83.28%)
         B-LOC  :    7140 (3.51%)
         B-PER  :    6600 (3.24%)
         B-ORG  :    6321 (3.10%)
         I-PER  :    4528 (2.22%)
         I-ORG  :    3704 (1.82%)
         B-MISC :    3438 (1.69%)
         I-LOC  :    1157 (0.57%)
         I-MISC :    1155 (0.57%)

Testing instances: 46435
Of which the label distribution is:
         O      :   38323 (82.53%)
         B-LOC  :    1668 (3.59%)
         B-ORG  :    1661 (3.58%)
         B-PER  :    1617 (3.48%)
         I-PER  :    1156 (2.49%)
         I-ORG  :     835 (1.80%)
         B-MISC :     702 (1.51%)
         I-LOC  :     257 (0.55%)
         I-MISC :     216 (0.47%)
```

The training set has a very unbalanced distribution. The 'Other' (O) label occurs 2 orders of magnitude more than any other label (169578 instances). It has about 23 times more instances than the second most occurring label B-LOC (7140 instances). It does make sense why this is the case, because it is simply the most common part of regular text/speech. All the other labels in the training set still have large differences in the number of occurrences. However, the difference between label occurrences is relatively small after comparing to the difference to the O label. The biggest difference between occurrences for the other labels within the training set is between I-MISC (1155 instances) and B-LOC (7140 instances), which is almost a difference of a factor of 7.

When comparing this drastic imbalance to the test set, we can see a similar pattern. The most occurring instance is the O label, which occurs about 23 times as often as the second most occurring label, B-LOC (1668 instances). When looking beyond this most occurring label, the rest of the label are once again relatively balanced, having a similar distribution to the training set. The second most occurring label B-LOC (1668 instances), and the least occurring label, I-MISC (216), differ in number of occurrences by around a factor of 8, which is very similar to the factor of 7 in the training set.

Lastly, when looking at the complete distribution and observing the instance distribution in percentages, we can see it is very similar. The largest difference in distribution percentage between the training and test set is about 0.7%, for the O label. This shows that while the original distribution may not be very balanced between instances, the training and the test set do stay very consistent and have a very similar distribution.

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

In [ ]:
```
vec = DictVectorizer()
the_array = vec.fit_transform(training_features + testing_features)
```

In [ ]:
```
training_features = the_array[:len(training_features)]
testing_features = the_array[-len(testing_features):]
del(the_array)
```

[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 [ ]:
```
from sklearn import svm
from sklearn.metrics import classification_report
```

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

In [ ]:
```
lin_clf.fit(training_features, training_gold_labels)
print(classification_report(testing_gold_labels, lin_clf.predict(testing_features)))
```

```
              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
```

The dataset is quite imbalanced since some classes have around 200 instances wile others have around 1500 and label O has over 38000. This is because most words are not named entities. Since there is a class imbalance it is best to look at the macro average instead of the accuracy. Overall the f1-score is 0.65. The recall is also 0.65 with a higher precision of 0.72.

The B-LOC and B-MISC labels are the best performing having a high recall and high precision meaning the classifier returns a lot of those instances and assigns them the correct class.

Most of the labels have low recall but high precision. The labels in question are B-ORG, B-PER, I-LOC and I-ORG. It does not find a lot of those instances but the classifier is able to return the correct label of the instances. B-PER has the highest precision 0.86.

I-PER is the only class that has high recall (0.87) but low precision (0.33) meaning it does find a lot of instances of that class but most are incorrectly labeled compared to the training labels.

[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 [ ]:
```
import gensim
import numpy as np
from sklearn.preprocessing import LabelBinarizer

word_embedding_model = gensim.models.KeyedVectors.load_word2vec_format('../../Goog

def to_embedding(word):
    if word != '' and word != 'DOCSTART' and word in word_embedding_model:
        return word_embedding_model[word]
    else: return [0]*300
```

In [ ]:
```
training_features, testing_features = np.empty((training_features.shape[0], 300)),
training_gold_labels, testing_gold_labels = [], []
pos = []

for i, (w, p, l) in enumerate(train.iob_words()):
    training_features[i] = to_embedding(w)
    training_gold_labels.append(l)
    pos.append(p)

for i, (w, p, l) in enumerate(test.iob_words()):
    testing_features[i] = to_embedding(w)
    testing_gold_labels.append(l)
    pos.append(p)

pos = LabelBinarizer().fit_transform(pos)
training_features = np.concatenate([training_features, pos[:training_features.shap
testing_features = np.concatenate([testing_features, pos[-testing_features.shape[0
```

In [ ]:
```
lin_clf = svm.LinearSVC()
lin_clf.fit(training_features, training_gold_labels)
print(classification_report(testing_gold_labels, lin_clf.predict(testing_features)
```

```
              precision    recall  f1-score   support

       B-LOC       0.75      0.81      0.78      1668
      B-MISC       0.71      0.69      0.70       702
       B-ORG       0.68      0.63      0.66      1661
       B-PER       0.75      0.68      0.71      1617
       I-LOC       0.51      0.41      0.46       257
      I-MISC       0.62      0.55      0.58       216
       I-ORG       0.50      0.36      0.42       835
       I-PER       0.60      0.54      0.57      1156
           O       0.98      0.99      0.98     38323

    accuracy                           0.93     46435
   macro avg       0.68      0.63      0.65     46435
weighted avg       0.92      0.93      0.93     46435
```

Again looking at the macro average instead of the accuracy we see that the f1-score is 0.65. The recall is 0.63 with a precision of 0.68. Comparing this model against the model in 2b we conclude that the previous model is slightly better. The recall and precision are 0.65 and 0.72. The previous model returns slightly more labels and classifies them slightly better.

None of the labels have a higher recall but lower precision. While all of the labels have lower recall and higher precision meaning that the classifier does not find a lot of those instances but the classifier is able to return the correct label of the instances. B-PER has a recall high precision of B-PER.

The labels I-LOC and I-ORG are the worst classified with an f1-score of 0.46 and 0.42 respectively. All the other labels have a f1-score higher then 0.60. O, B-LOC and B-PER labels are the best performing.

[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 [ ]:
```
import pandas
```

In [ ]:
```
path = 'nerc_datasets/kaggle/ner_v2.csv'
kaggle_dataset = pandas.read_csv(path, on_bad_lines='skip')
features = kaggle_dataset[['word', 'pos']]#list(set(kaggle_dataset.columns).differe
labels = kaggle_dataset.tag
```

In [ ]:
```
len(features) == len(labels)
```

Out[ ]:
```
True
```

In [ ]:
```
encoded_features = DictVectorizer().fit_transform([i.to_dict() for _, i in features
```

In [ ]:
```
df_train_features, df_test_features = encoded_features[:100000], encoded_features[
df_train_labels, df_test_labels = labels[:100000], labels[100000:120000]
print(df_train_features.shape[0], df_test_features.shape[0])
print(len(df_train_labels), len(df_test_labels))
```

```
100000 20000
100000 20000
```

[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 [ ]:
```
lin_clf = svm.LinearSVC()
lin_clf.fit(df_train_features, df_train_labels)
print(classification_report(df_test_labels, lin_clf.predict(df_test_features), zer
```

```
              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
```

For some labels there were so few entities it's hard to rate its performance. The following labels had support lower than 10 where the performance is unclear:

- *B-art: There were only 4 named entities, these 4 were all incorrectly labeled.*
- *I-art: There were 0 named entities.*
- *B-eve: There were 0 named entities.*
- *I-eve: There were 0 named entities.*
- *B-nat: There were 8 named entities, were the precision was 1 but the recall 0.50.*
- *I-nat: There were 4 named entities were the precision was 0.80 and the recall 1.*
- *I-gpe: There were only 2 named entities were the precision was 1 but the recall 0.50.*

The other labels had sufficient support. B-geo: This label had by far the most entities after O. The precision was 0.80, recall 0.76, f1-score 0.78. Performance was good. I-geo: The precision was 0.74, recall 0.50, f1-score 0.60. It was quite accurate but ignored quite a few entities. B-gpe: The precision was 0.96, recall 0.92 and f1-score 0.94. So the performance was very good. B-org: The precision was 0.64, recall 0.51 and f1-score 0.57. So the performance was not quite good, although the precision was decent. I-org: The precision was 0.65, recall 0.44, f1 score 0.53. This is a low f1-score and a quite bad performance. B-per: Precision was 0.81, recall 0.53, f1-score 0.64. This is high precision but a not so good recall, making it quite accurate but ignoring a lot of entities. I-per: Precision was 0.42, recall 0.90, f1-score 0.57. Here the precision

is very low, but the recall is very high. Overall making the f1-score 0.57 which is not a great performance. B-tim: Precision 0.91, recall 0.76, f1-score 0.87. This is quite a good performance with a high f1-score. I-tim: Precision 0.41, recall 0.08, f1 score 0.14. So this NERC label has a bad performance with ignoring many entities. The precision is also low.

O: This has many more entities than all the other labels. The f1-score of 0.99 makes it have a really good performance.

The accuracy overall is 0.94. This is quite high, because of the O-label. This has more than 75% of all the entities. The macro avg f1-score however, is only 0.52. </i>