Lab4-Assignment-nerc

March 10, 2025

1 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

1.1 Credits

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

1.2 [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',
....
]
```

[35]:
```python
from nltk.corpus.reader import ConllCorpusReader
### Adapt the path to point to the CONLL2003 folder on your local machine
train = ConllCorpusReader('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)
```

[36]:
```python
### Adapt the path to point to the CONLL2003 folder on your local machine
test = ConllCorpusReader('CONLL2003', 'test.txt', ['words', 'pos', 'ignore', 'chunk'])

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?
  - There are 203621 instances in the training set and 46435 instances in the test set.
- Provide a frequency distribution of the NERC labels, i.e., how many times does each NERC label occur?
  - See the output of the cells below
- 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?
  - The training and test data is fairly balanced. As can be seen in the cells below, the proportion of the training data for each NERC label is around 80% and that of the test

data is at around 20%.

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

[ ]:
```
from collections import Counter
# First subquestion
print(f"Total instances in train: {len(training_features)}")
print(f"Total instances in test: {len(test_features)}")

# Second subquestion
print("Frequency distribution of the whole dataset:")
Counter(training_gold_labels+test_gold_labels)
```

```
Total instances in train: 203621
Total instances in test: 46435
Frequency distribution of the whole dataset:
```

[ ]:
```
Counter({'O': 207901,
         'B-LOC': 8808,
         'B-PER': 8217,
         'B-ORG': 7982,
         'I-PER': 5684,
         'I-ORG': 4539,
         'B-MISC': 4140,
         'I-LOC': 1414,
         'I-MISC': 1371})
```

[ ]:
```
# Third subquestion
print("Frequency distribution of the training set:")
Counter(training_gold_labels)
```

```
Frequency distribution of the training set:
```

[ ]:
```
Counter({'O': 169578,
         'B-LOC': 7140,
         'B-PER': 6600,
         'B-ORG': 6321,
         'I-PER': 4528,
         'I-ORG': 3704,
         'B-MISC': 3438,
         'I-LOC': 1157,
         'I-MISC': 1155})
```

[ ]:
```
print("Frequency distribution of the test set:")
Counter(test_gold_labels)
```

```
Frequency distribution of the test set:
```

[ ]:
```
Counter({'O': 38323,
         'B-LOC': 1668,
```

```
          'B-ORG': 1661,
          'B-PER': 1617,
          'I-PER': 1156,
          'I-ORG': 835,
          'B-MISC': 702,
          'I-LOC': 257,
          'I-MISC': 216})
```

[ ]:
```python
train_proportion = {}
test_proportion = {}

all_counter = dict(Counter(training_gold_labels+test_gold_labels))
train_counter = dict(Counter(training_gold_labels))
test_counter = dict(Counter(test_gold_labels))

for label in all_counter:
    train_prop = train_counter[label]/all_counter[label]
    test_prop = test_counter[label]/all_counter[label]
    train_proportion[label] = train_prop
    test_proportion[label] = test_prop

print("Training data proportion")
train_proportion
```

```
Training data proportion
```

[ ]:
```
{'B-ORG': 0.7919067902781258,
 'O': 0.8156670723084545,
 'B-MISC': 0.8304347826086956,
 'B-PER': 0.8032128514056225,
 'I-PER': 0.796622097114708,
 'B-LOC': 0.8106267029972752,
 'I-ORG': 0.8160387750605861,
 'I-MISC': 0.8424507658643327,
 'I-LOC': 0.8182461103253182}
```

[ ]:
```python
print("Testing data proportion")
test_proportion
```

```
Testing data proportion
```

[ ]:
```
{'B-ORG': 0.20809320972187423,
 'O': 0.1843329276915455,
 'B-MISC': 0.16956521739130434,
 'B-PER': 0.19678714859437751,
 'I-PER': 0.20337790288529206,
 'B-LOC': 0.1893732970027248,
 'I-ORG': 0.18396122493941397,
```

```
 'I-MISC': 0.1575492341356674,
 'I-LOC': 0.18175388967468176}
```

[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()
all_features = training_features + test_features
the_array = vec.fit_transform(all_features)

len_training_features = len(training_features)
training_features = the_array[:len_training_features]
test_features = the_array[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?
  - The NERC labels that the classifier performs well on are O (f1 score = 0.98), B-LOC (f1 score = 0.79) and B-MISC (f1 score = 0.71). This is because these entities have a clear patterns and are frequent, so there is a lot of training data for the model.
- Which NERC labels does the classifier perform poorly on? Why do you think this is the case?
  - The NERC labels on which the classifier performs poorly are I-PER (f1 score = 0.48), I-ORG (f1 score = 0.55), I-LOC (f1 score = 0.57) and B-PER (f1 score = 0.58). In fact, I-PER and B-PER often involve person names, and they can be easily confused with other common nouns. I-ORG and I-LOC can be also challenging since the model has to detect when the name entity is composed of multiple words.

[ ]:
```
from sklearn import svm
from sklearn.metrics import classification_report
import numpy as np
```

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

[ ]:
```
##### [ YOUR CODE SHOULD GO HERE ]
lin_clf.fit(training_features,training_gold_labels)
```

```
c:\Users\amina\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()

[ ]:
```
y_pred = lin_clf.predict(test_features)

print(classification_report(test_gold_labels, 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       0.87      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 1d.

- B-LOC: the trained model from 1d (with an F1-score of 0.79) performs slightly better than the model of 1e (with an F1-score of 0.78). However, 1e has a better recall.
- B-MISC: the trained model from 1d (with an F1-score of 0.71) has the same performance as the model of 1e (with an F1-score of 0.71 too). However, 1d has a better precision rate, whereas 1e scores better for recall.
- B-ORG: the trained model from 1e (with an F1-score of 0.66) performs better than the model of 1d (with an F1-score of 0.62). 1d scores better for precision, while 1e scores better for recall.
- B-PER: the trained model from 1e (with an F1-score of 0.71) performs better than the model of 1d (with an F1-score of 0.58). However, 1d scores better for precision, while 1e scores better for recall.
- I-LOC: the trained model from 1d (with an F1-score of 0.57) performs better than the model of 1e (with an F1-score of 0.46). 1d scores better for both precision and recall.
- I-MISC: the trained model from 1d (with an F1-score of 0.59) performs slightly better than the model of 1e (with an F1-score of 0.57). 1d scores better for precision, while 1e scores better for recall.

- I-ORG: the trained model from 1d (with an F1-score of 0.55) performs better than the model of 1e (with an F1-score of 0.39). It also scores higher for both precision and recall.
- I-PER: the trained model from 1e (with an F1-score of 0.54) performs better than the model of 1d (with an F1-score of 0.48). However, 1d has a better recall rate, and 1e has a better precision rate.
- O: the trained model from 1d (with an impressive F1-score of 0.98) performs the same as the model from 1e (with an F1-score of 0.98 too). However, 1d scores slightly better for precision, and 1e scores slightly better for recall.

[ ]:
```
import gensim.downloader as api
word_embedding_model = api.load("word2vec-google-news-300")
```

[ ]:
```
input_vectors = []
labels = []

valid_tokens = [(token, ne_label) for token, pos, ne_label in train.iob_words()␣
↪if token and token != 'DOCSTART']

num_tokens = len(valid_tokens)
input_vectors = np.zeros((num_tokens, 300))
labels = np.empty(num_tokens, dtype=object)

for i, (token, ne_label) in enumerate(valid_tokens):
    if token in word_embedding_model:
        input_vectors[i] = word_embedding_model[token]
    labels[i] = ne_label
```

[ ]:
```
input_vectors_test = []
labels_test = []

valid_tokens_test = [(token, ne_label) for token, pos, ne_label in test.
↪iob_words() if token and token != 'DOCSTART']

num_tokens_test = len(valid_tokens_test)
input_vectors_test = np.zeros((num_tokens_test, 300))
labels_test = np.empty(num_tokens_test, dtype=object)

for i, (token, ne_label) in enumerate(valid_tokens_test):
    if token in word_embedding_model:
        input_vectors_test[i] = word_embedding_model[token]
    labels_test[i] = ne_label
```

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

lin_clf.fit(input_vectors, labels)
```

[ ]: LinearSVC()

[ ]:
```
testpredict = lin_clf.predict(input_vectors_test)

print(classification_report(labels_test, testpredict))
```

```
              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
```

1.3 [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

[19]:
```
import pandas
```

[20]:
```
##### Adapt the path to point to your local copy of NERC_datasets
path = 'ner_dataset.csv'
kaggle_dataset = pandas.read_csv(path, encoding='latin1')
```

[21]:
```
len(kaggle_dataset)
```

[21]: 1048575

[22]:
```
df_train = kaggle_dataset[:100000]
df_test = kaggle_dataset[100000:120000]
print(len(df_train), len(df_test))
```

```
100000 20000
```

[23]:
```
kaggle_training_features = []

pos_list = df_train["POS"].values
token_list = df_train["Word"].values

for token,pos in zip(token_list,pos_list):
    a_dict = {
        'words':token,
        'pos':pos
        }
    kaggle_training_features.append(a_dict)

kaggle_training_gold_labels = df_train["Tag"].values
```

[24]:
```
kaggle_test_features = []

pos_list = df_test["POS"].values
token_list = df_test["Word"].values

for token,pos in zip(token_list,pos_list):
    a_dict = {
        'words':token,
        'pos':pos
        }
    kaggle_test_features.append(a_dict)

kaggle_test_gold_labels = df_test["Tag"].values
```

[25]:
```
vec = DictVectorizer()
all_features = kaggle_training_features + kaggle_test_features
the_array = vec.fit_transform(all_features)

len_training_features = len(kaggle_training_features)
kaggle_training_features = the_array[:len_training_features]
kaggle_test_features = the_array[len_training_features:]
```

[26]:
```
lin_clf = svm.LinearSVC()
lin_clf.fit(kaggle_training_features,kaggle_training_gold_labels)
```

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

[26]:
```
LinearSVC()
```

[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.

- B-art
  - Precision: 0.00, Recall: 0.00, F1-score: 0.00, Support: 4
  - This label has poor performance and is failing to make any good predictions, likely because there are very limited instances.
- B-eve
  - Precision: 0.00, Recall: 0.00, F1-score: 0.00, Support: 0
  - There are no instances of this label.
- B-geo
  - Precision: 0.80, Recall: 0.76, F1-score: 0.78, Support: 741
  - A good performance, the model is good at identifying geographical entities.
- B-gpe
  - Precision: 0.96, Recall: 0.92, F1-score: 0.94, Support: 296
  - A very good performance, geopolitical entities are well identified.
- B-nat
  - Precision: 1.00, Recall: 0.50, F1-score: 0.67, Support: 8
  - A precision of 100% but recall is rather low.
- B-org
  - Precision: 0.63, Recall: 0.51, F1-score: 0.57, Support: 397
  - A moderate performance in both recall and precision, There is room for improvement in this part.
- B-per
  - Precision: 0.81, Recall: 0.53, F1-score: 0.64, Support: 333
  - The precision is good but recall is lower, this means that the model didn't identify all true entities.
- B-tim
  - Precision: 0.91, Recall: 0.76, F1-score: 0.83, Support: 393
  - A very good performance, time-related entities are well idnetified.
- I-art
  - Precision: 0.00, Recall: 0.00, F1-score: 0.00, Support: 0
  - There are no instances of this label.
- I-eve

- Precision: 0.00, Recall: 0.00, F1-score: 0.00, Support: 0
  - There are no instances of this label.
- I-geo
  - Precision: 0.74, Recall: 0.50, F1-score: 0.60, Support: 156
  - The precision is good but recall is lower, this means that the model didn't identify all true entities.
- I-gpe
  - Precision: 1.00, Recall: 0.50, F1-score: 0.67, Support: 2
  - A perfect precision but a low recall as well as a very few instances.
- I-nat
  - Precision: 0.80, Recall: 1.00, F1-score: 0.89, Support: 4
  - The precision is good and the recall is perfect, howver there are only very few instances
- I-org
  - Precision: 0.66, Recall: 0.44, F1-score: 0.53, Support: 321
  - Medium performance, there is more room for improvement.
- I-per
  - Precision: 0.42, Recall: 0.90, F1-score: 0.57, Support: 319
  - Low precision and a high recall, indicating a lot of false positives.
- I-tim
  - Precision: 0.41, Recall: 0.08, F1-score: 0.14, Support: 108
  - Very low performance, the model is failing to detect many of the time entities.
- O
  - Precision: 0.98, Recall: 0.99, F1-score: 0.99, Support: 16918
  - Very good performance, this label performs very good due to its high support.

[27]:
```
y_pred = lin_clf.predict(kaggle_test_features)

print(classification_report(kaggle_test_gold_labels, y_pred))
```

```
              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       0.63      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       0.66      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

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

1.4 End of this notebook