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.
```
```
[
{'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',
....
]
```

[1]:
```python
from nltk.corpus.reader import ConllCorpusReader
### Adapt the path to point to the CONLL2003 folder on your local machine
train = ConllCorpusReader('C:/Users/lisam/OneDrive/Year 3/Period 4/Text Mining/
↪ba-text-mining-gr43/lab_sessions/lab4/CONLL2003/CONLL2003', 'test.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)
```

[2]:
```python
### Adapt the path to point to the CONLL2003 folder on your local machine
train = ConllCorpusReader('C:/Users/lisam/OneDrive/Year 3/Period 4/Text Mining/
↪ba-text-mining-gr43/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:

[3]:
```python
from collections import Counter

my_list=[1,2,1,3,2,5]
Counter(my_list)
```

[3]:
```
Counter({1: 2, 2: 2, 3: 1, 5: 1})
```

[4]:
```python
from collections import Counter

path = 'C:/Users/lisam/OneDrive/Year 3/Period 4/Text Mining/ba-text-mining-gr43/
↪lab_sessions/lab4/CONLL2003/CONLL2003'
```

```python
train = ConllCorpusReader(path, 'train.txt', ['words', 'pos', 'ignore',␣
  ↪'chunk'])
test = ConllCorpusReader(path, 'test.txt', ['words', 'pos', 'ignore', 'chunk'])

def count_labels(data):

    labels = []
    for token, pos, ne_label in data.iob_words():
        labels.append(ne_label)
    return labels, Counter(labels)

def calculate_percentages(label_counts, total):
    print("\nDstribution of NERC labels (in percentage):")
    for label, count in label_counts.items():
        percentage = (count / total) * 100
        print(f"{label}: {percentage:.2f}%")

def get_statistics(data, name):
    print(f"\n{name}:")

    labels, label_counts = count_labels(data)
    num_instances = len(labels)

    print(f"Number of instances: {num_instances}")

    print("\nFrequency distribution of NERC labels:")
    for label, count in label_counts.items():
        print(f"{label}: {count}")

    calculate_percentages(label_counts, num_instances)

get_statistics(train, "Train")
get_statistics(test, "Test")
```

```
Train:
Number of instances: 203621

Frequency distribution of NERC labels:
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

Dstribution of NERC labels (in percentage):
B-ORG: 3.10%
O: 83.28%
B-MISC: 1.69%
B-PER: 3.24%
I-PER: 2.22%
B-LOC: 3.51%
I-ORG: 1.82%
I-MISC: 0.57%
I-LOC: 0.57%

Test:
Number of instances: 46435

Frequency distribution of NERC labels:
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

Dstribution of NERC labels (in percentage):
O: 82.53%
B-LOC: 3.59%
B-PER: 3.48%
I-PER: 2.49%
I-LOC: 0.55%
B-MISC: 1.51%
I-MISC: 0.47%
B-ORG: 3.58%
I-ORG: 1.80%
```

Answers:

- There are 203621 instances of training data, and 46435 of testing, as it can be seen from the output of the cell below.
- The frequency distibution can also be seen in the output of the cell below, which for training data is: 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

and for test data is:

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

- The part with percentage count in the output of the cell below clearly indicates the imbalance of training and test data when looking at them separately. For both test and train data, the "O" label, which is used to label the words that are not a part of any category of named entity, dominates with 83.28% and 82.53%, respectively. This can lead to biases in the model and incorrect labeling. And looking at percentage of labels that start with I or B and comparing them, we can see that in general, NERC labels that start with B ahave higher percentages, which means that the most named entities are shorter rather than longer. If comparing test and training data with each other, they do have similar percentages for the same labels, not having differences of more than ~1%, some of them having a difference only of 0.02% (for example I-LOC, with 0.57% and 0.55%).

[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.

[5]:
```
from sklearn.feature_extraction import DictVectorizer
```

[6]:
```
path = 'C:/Users/lisam/OneDrive/Year 3/Period 4/Text Mining/ba-text-mining-gr43/
  ↪lab_sessions/lab4/CONLL2003/CONLL2003'
reader = ConllCorpusReader(path, ['train.txt', 'test.txt', 'valid.txt'],␣
  ↪['words', 'pos', 'ignore', 'chunk'])
```

```
training_features = []
training_labels = []

for token, pos, ne_label in reader.iob_words('train.txt'):
    training_features.append({'words': token, 'pos': pos})
    training_labels.append(ne_label)

test_features = []
test_labels = []

for token, pos, ne_label in reader.iob_words('test.txt'):
    test_features.append({'words': token, 'pos': pos})
    test_labels.append(ne_label)

print(f"Training examples: {len(training_features)}")
print(f"Test examples: {len(test_features)}")
```

```
Training examples: 203621
Test examples: 46435
```

[7]:
```
all_features = training_features + test_features

vec = DictVectorizer()
the_array = vec.fit_transform(all_features)

num_train = len(training_features)
train_array = the_array[:num_train]
test_array = the_array[num_train:]

print("Train array shape:", train_array.shape)
print("Test array shape:", test_array.shape)
```

```
Train array shape: (203621, 27361)
Test array shape: (46435, 27361)
```

[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?

[8]:
```
from sklearn import svm
from sklearn.metrics import classification_report
```

[9]:
```
lin_clf = svm.LinearSVC()
```

```
[11]: all_features = training_features + test_features
      all_labels = training_labels + test_labels

      vec = DictVectorizer()
      X_all = vec.fit_transform(all_features)

      X_train = X_all[:len(training_features)]
      X_test = X_all[len(training_features):]

      y_train = training_labels
      y_test = test_labels

      lin_clf.fit(X_train, y_train)

      y_pred = lin_clf.predict(X_test)

      print(classification_report(y_test, y_pred))
```

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

1.2.1 Answers:

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

  The classifier performs well on O labels, probably getting a high F1-score because there is a lot of them available and they're easy to tell from named entities. It also performs well on B-LOC and B-MISC, likely because locations and miscellaneous entities follow recognizable patterns.

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

However, it struggles with inside entity labels, especially I-PER, which has high recall but low precision, indicating over-prediction. B-PER also has low recall, meaning many person names are missed. This is likely due to the variability of personal names and insufficient training data for certain entities.

[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.

[12]:
```
import numpy as np
import gensim.downloader as api
from sklearn.svm import LinearSVC
from sklearn.metrics import classification_report
from sklearn.preprocessing import LabelEncoder

print("Loading GloVe embeddings...")
glove_model = api.load("glove-wiki-gigaword-50")

def get_embedding(word):
    return glove_model[word] if word in glove_model else np.zeros(50)

X_train = X_all[:len(training_features)]
X_test = X_all[len(training_features):]

y_encoder = LabelEncoder()
y_train = y_encoder.fit_transform(training_labels)
y_test = y_encoder.transform(test_labels)

print("Training SVM...")
svm_clf = LinearSVC()
svm_clf.fit(X_train, y_train)

y_pred = svm_clf.predict(X_test)

print("Classification Report:")
print(classification_report(y_test, y_pred, target_names=y_encoder.classes_))
```

```
Loading GloVe embeddings…
Training SVM…
Classification Report:
              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


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

1.2.2 Answers

- compare the results with the classifier you built in 2d.

  The results are identical and have no differences between them which could mean that these traditional features already captured enough information for NER, so adding word embedding didn't help. GloVe embedding might also not be well-suited for entity recognition because word embeddings focus on general semantic meaning but NER requires context information. Maybe the SVM model might not effectively leverage dense word embeddings compared to deep learning model like BiLTSMs or Transformers.

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

[1]:
```python
import pandas
```

[2]:
```python
##### Adapt the path to point to your local copy of NERC_datasets
path = 'C:/Users/lisam/OneDrive/Year 3/Period 4/Text Mining/ba-text-mining-gr43/
↪lab_sessions/lab4/ner_dataset.csv'
kaggle_dataset = pandas.read_csv(path, encoding='latin1',      ␣
↪on_bad_lines='skip')
```

[3]:
```python
len(kaggle_dataset)
```

[3]:
```
1048575
```

[4]:
```python
df_train = kaggle_dataset[:5000]
df_test = kaggle_dataset[5000:6000]
print(len(df_train), len(df_test))
```

```
5000 1000
```

[5]:
```
def create_features_and_labels(data):
    features = []
    labels = []

    for _, row in data.iterrows():
        word = row['Word']
        pos = row['POS']
        ne_label = row['Tag']


        feature_dict = {'word': word, 'pos': pos}
        features.append(feature_dict)
        labels.append(ne_label)

    return features, labels


train_features, train_labels = create_features_and_labels(df_train)
test_features, test_labels = create_features_and_labels(df_test)
```

[6]:
```
from sklearn.feature_extraction import DictVectorizer


all_features = train_features + test_features

vectorizer = DictVectorizer(sparse=False)
all_features_vectorized = vectorizer.fit_transform(all_features)
train_features_vectorized = all_features_vectorized[:len(train_features)]
test_features_vectorized = all_features_vectorized[len(train_features):]
```

[7]:
```
from sklearn.svm import SVC


svm_classifier = SVC(kernel='linear', verbose=True)
svm_classifier.fit(train_features_vectorized, train_labels)
```

```
[LibSVM]
```

[7]:
```
SVC(kernel='linear', verbose=True)
```

[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.

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


     test_predictions = svm_classifier.predict(test_features_vectorized)
     print(classification_report(test_labels, test_predictions))
```

```
              precision    recall  f1-score   support

       B-art       0.00      0.00      0.00         7
       B-eve       1.00      0.25      0.40         4
       B-geo       0.19      0.86      0.31        22
       B-gpe       0.89      0.46      0.60        35
       B-nat       1.00      0.33      0.50         3
       B-org       0.82      0.39      0.53        23
       B-per       1.00      0.36      0.53        14
       B-tim       0.92      0.86      0.89        14
       I-art       0.00      0.00      0.00         5
       I-eve       0.00      0.00      0.00         3
       I-geo       0.00      0.00      0.00         1
       I-gpe       1.00      0.80      0.89         5
       I-nat       0.00      0.00      0.00         2
       I-org       0.89      0.57      0.70        14
       I-per       0.67      0.11      0.18        19
           O       0.99      1.00      1.00       829

    accuracy                           0.91      1000
   macro avg       0.59      0.37      0.41      1000
weighted avg       0.94      0.91      0.91      1000

c:\Users\lisam\anaconda3\envs\TextMining\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\lisam\anaconda3\envs\TextMining\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\lisam\anaconda3\envs\TextMining\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))
```

1.3.1 Answser

- Evaluate the performance of the SVM on the test data

The performance of the SVM classifier can be evaluated using precision, recall, and F1-score for each NERC label. Considering the classification report, the classifier performs well on labels like B-tim and O, as they have high precision and recall. This is likely because location names are well-defined and frequently seen in training data.

However, the classifier struggles with labels like B-org and B-per, showing lower recall. This could be due to the variability in organization and person names, which may not follow strict patterns. The I-per label especially has poor precision but high recall, indicating that the model often makes mistakes with non-person continuation tokens as part of a person entity. These errors may be a reason for difficulty in distinguishing multi-token named entities, leading to incomplete entity recognition.

1.4 End of this notebook