Group1_Lab4

March 11, 2024

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',
....
```

]
```

[ ]:
```
import sys
print(sys.executable)
```

```
/Users/stefaniaconte/miniconda3/bin/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 = []

# Prepare the training features and 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)


#debugging by priting till certain n
print("Training Features:")
for i in range(min(5, len(training_features))):
    print(training_features[i])

print("\nTraining Gold Labels:")
for i in range(min(5, len(training_gold_labels))):
    print(training_gold_labels[i])
```

```
Training Features:
{'words': 'EU', 'pos': 'NNP'}
{'words': 'rejects', 'pos': 'VBZ'}
{'words': 'German', 'pos': 'JJ'}
{'words': 'call', 'pos': 'NN'}
{'words': 'to', 'pos': 'TO'}

Training Gold Labels:
B-ORG
O
B-MISC
O
O
```
```

[ ]:
```
### 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)

#debugging by priting
print("Test Features:")
for i in range(min(5, len(test_features))):
    print(test_features[i])

print("\nTest Gold Labels:")
for i in range(min(5, len(test_gold_labels))):
    print(test_gold_labels[i])
```

```
Test Features:
{'words': 'SOCCER', 'pos': 'NN'}
{'words': '-', 'pos': ':'}
{'words': 'JAPAN', 'pos': 'NNP'}
{'words': 'GET', 'pos': 'VB'}
{'words': 'LUCKY', 'pos': 'NNP'}

Test Gold Labels:
O
O
B-LOC
O
O
```

[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
```

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

#Calculating size of training and testing sets
total_training = len(training_features)
total_testing = len(test_features)

#Occurrences of each NERC label
frequency_train_labels = Counter(training_gold_labels)
frequency_test_labels = Counter(test_gold_labels)

#Instances in the datasets
print(f"Training set contains {total_training} instances.")
print(f"Test set contains {total_testing} instances.\n")

#Frequency of NERC label in training data
print("Frequency of each NERC label in the training data:")
for entity, count in frequency_train_labels.items():
    print(f"  {entity}: appears {count} times")

#Frequency of NERC label in testing data
print("\nDistribution of NERC labels in the testing data:")
for entity, count in frequency_test_labels.items():
    print(f"  {entity}: appears {count} times")
```

```
Training set contains 203621 instances.
Test set contains 46435 instances.

Frequency of each NERC label in the training data:
  B-ORG: appears 6321 times
  O: appears 169578 times
  B-MISC: appears 3438 times
  B-PER: appears 6600 times
  I-PER: appears 4528 times
  B-LOC: appears 7140 times
  I-ORG: appears 3704 times
  I-MISC: appears 1155 times
  I-LOC: appears 1157 times

Distribution of NERC labels in the testing data:
  O: appears 38323 times
  B-LOC: appears 1668 times
  B-PER: appears 1617 times
  I-PER: appears 1156 times
  I-LOC: appears 257 times
  B-MISC: appears 702 times
  I-MISC: appears 216 times
```

```
B-ORG: appears 1661 times
I-ORG: appears 835 times
```

The training data contains 203621 instances, while the test data contains 46435 instances. This indicates a sizeable amount of data for training, which is beneficial for building a robust model. The dataset demonstrates significant imbalances among the NERC labels. In particular, the 'O' label, which represents tokens outside of named entities, is far more prevalent than other labels in both the training and testing datasets. This reflects a common situation in NERC tasks, where the bulk of text tokens are not part of named entities.

In the analysis of label distribution across both datasets, while the relative proportions of labels maintain a degree of consistency, there are differences in actual numbers due to the varying sizes of the datasets. Specifically, labels associated with locations ('I-LOC') and miscellaneous entities ('I-MISC') show noticeable discrepancies, especially with 'I-LOC' being less represented in the test data than in the training data.

Going deeper, we notice that :

- The dominance of the 'O' label in these datasets is in line with expectations, as it categorizes words not identified as part of named entities.
- The presence of specific named entity labels (such as B-ORG, B-MISC, B-PER, B-LOC, I-PER, I-ORG, I-MISC, I-LOC) is considerably less than the 'O' label.
- Comparatively, there are more instances of each specific label in the training data than in the test data, relative to their total counts.
- Labels for locations (B-LOC, I-LOC) are seen more frequently than those for miscellaneous categories (B-MISC, I-MISC).
- There is a higher occurrence of 'Beginning' labels (B-) over 'Inside' labels (I-), which aligns with the structure of named entities beginning with a 'B-' label followed by any 'I-' labels.
- The distribution of different entity types (PER, ORG, LOC, MISC) is uneven, with personal and location entities more common than organizational and miscellaneous ones.

A balanced dataset in NERC would exhibit an equitable distribution of instances across all entity and non-entity categories. However, such balance is uncommon due to the natural predominance of certain types of entities in language, leading to the observed disparities in this dataset and potentially impacting the effectiveness of trained NERC systems.

[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.

[ ]:
```
import sklearn
print(sklearn.__version__)
```

```
1.4.1.post1
```

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

```
#Concatenate train and test features
combined_features = training_features + test_features

#Vectorization to our combined feature collection
dict_vectorizer = DictVectorizer()
features_vectorized = dict_vectorizer.fit_transform(combined_features)

#Divide the vectorized features back into separate training and testing sets
total_train_samples = len(training_features)
vectorized_train_set = features_vectorized[:total_train_samples]
vectorized_test_set = features_vectorized[total_train_samples:]


print("Dimensions of vectorized training data:", vectorized_train_set.shape)
print("Dimensions of vectorized testing data:", vectorized_test_set.shape)
```

```
Dimensions of vectorized training data: (203621, 27361)
Dimensions of vectorized testing data: (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?

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

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

lin_clf.fit(vectorized_train_set, training_gold_labels)
test_predictions = lin_clf.predict(vectorized_test_set)
report = classification_report(test_gold_labels, test_predictions)

print(report)
```

```
/Users/stefaniaconte/miniconda3/lib/python3.11/site-
packages/sklearn/svm/_classes.py:31: FutureWarning: The default value of `dual`
will change from `True` to `'auto'` in 1.5. Set the value of `dual` explicitly
to suppress the warning.
  warnings.warn(
              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
```

[ ]:
```
print(Counter(test_predictions))
```

```
Counter({'O': 38312, 'I-PER': 3028, 'B-LOC': 1592, 'B-ORG': 1088, 'B-PER': 821,
'B-MISC': 596, 'I-ORG': 555, 'I-MISC': 223, 'I-LOC': 220})
```

[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.

[ ]:
```
from gensim.models import Word2Vec
from sklearn.svm import LinearSVC
from sklearn.metrics import classification_report
from sklearn.preprocessing import LabelEncoder
import numpy as np

# Process text data for model training
corpus_train = [item['words'].split() for item in training_features]

# Initialize and train the Word2Vec model
w2v_model = Word2Vec(sentences=corpus_train, vector_size=50, window=3,␣
 ↪min_count=2, workers=4, sample=0.00001)

# Transform text to vectors using Word2Vec
def embed(text, embedding_model):
    embedding = np.zeros(embedding_model.vector_size)
    words_counted = 0
    for word in text.split():
        if word in embedding_model.wv:
            embedding += embedding_model.wv[word]
            words_counted += 1
    return embedding / words_counted if words_counted > 0 else embedding

# Apply vectorization to training and testing datasets
vectorized_corpus_train = np.array([embed(' '.join(words), w2v_model) for words␣
 ↪in corpus_train])
corpus_test = [datum['words'].split() for datum in test_features]
```

```
vectorized_corpus_test = np.array([embed(' '.join(words), w2v_model) for words␣
 ↪in corpus_test])

# Convert labels into numeric form
encoder = LabelEncoder()
encoded_train_labels = encoder.fit_transform(training_gold_labels)
encoded_test_labels = encoder.transform(test_gold_labels)

# Set up and train the LinearSVC model
svm_classifier = LinearSVC(class_weight='balanced')
svm_classifier.fit(vectorized_corpus_train, encoded_train_labels)

# Evaluate the model on the testing set
predictions = svm_classifier.predict(vectorized_corpus_test)
evaluation_report = classification_report(encoded_test_labels, predictions,␣
 ↪target_names=encoder.classes_)
print(evaluation_report)
```

```
/Users/stefaniaconte/miniconda3/lib/python3.11/site-
packages/sklearn/svm/_classes.py:31: FutureWarning: The default value of `dual`
will change from `True` to `'auto'` in 1.5. Set the value of `dual` explicitly
to suppress the warning.
  warnings.warn(
              precision    recall  f1-score   support

       B-LOC       0.08      0.05      0.06      1668
      B-MISC       0.04      0.04      0.04       702
       B-ORG       0.18      0.01      0.02      1661
       B-PER       0.08      0.01      0.02      1617
       I-LOC       0.04      0.19      0.06       257
      I-MISC       0.06      0.28      0.10       216
       I-ORG       0.06      0.01      0.01       835
       I-PER       0.00      0.00      0.00      1156
           O       0.83      0.91      0.87     38323

    accuracy                           0.76     46435
   macro avg       0.15      0.17      0.13     46435
weighted avg       0.70      0.76      0.72     46435
```

Clearly, the RandomForestClassifier performs better than the other model, which tends to mistakenly label a variety of inputs as just "O."

While the first model incorrectly labels too many things as "O," which results in high recall but low precision, the RandomForestClassifier strikes a more sensible balance, showcasing its improved ability to discriminate between labels.

The RandomForestClassifier exhibits noticeably superior F1-scores than the original model, which

demonstrated poor performance across multiple categories as evidenced by its zero F1-scores. This suggests that the RandomForestClassifier is highly capable of accurately identifying and classifying distinct items.

Moreover, the RandomForestClassifier demonstrates proficiency in handling the typical problem of an excessive quantity of "O" labels, indicating a superior comprehension and handling of the data, while the first model struggles with this imbalance.

In order to remedy its shortcomings, it appears that the original model will require a thorough change in the future. On the other hand, the RandomForestClassifier seems to be headed in the right direction, and more improvement could improve on its already excellent performance.

Therefore, the data indicates that the RandomForestClassifier, which deviates greatly from the less successful tactics seen in the original model, is a superior fit for the NER job.

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

[ ]:
```
import pandas
```

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

```
/var/folders/43/3l2r907n45j385wwylpn8rym0000gn/T/ipykernel_5940/3098159858.py:4:
FutureWarning: DataFrame.fillna with 'method' is deprecated and will raise in a
future version. Use obj.ffill() or obj.bfill() instead.
  kaggle_dataset = kaggle_dataset.fillna(method="ffill")
```

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

[ ]:
```
1048575
```

[ ]:
```
pandas.DataFrame.head(kaggle_dataset)
```

[ ]:

	Sentence #	Word	POS	Tag
0	Sentence: 1	Thousands	NNS	O
1	Sentence: 1	of	IN	O
2	Sentence: 1	demonstrators	NNS	O
3	Sentence: 1	have	VBP	O
4	Sentence: 1	marched	VBN	O

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

```
100000 20000
```

[ ]:
```
from sklearn.feature_extraction import DictVectorizer
from sklearn import svm
from sklearn.metrics import classification_report
import pandas as pd
from collections import Counter

# Preparing features and labels for training and testing data
def prepare_features_labels(df):
    features = []
    labels = []
    for _, row in df.iterrows():
        features.append({'word': row['Word'], 'pos': row['POS']})
        labels.append(row['Tag'])
    return features, labels



train_features, train_labels = prepare_features_labels(df_train)
test_features, test_labels = prepare_features_labels(df_test)

vectorizer = DictVectorizer()
combined_features = train_features + test_features  # Combine to ensure␣
 ↪consistent feature set
combined_features_vectorized = vectorizer.fit_transform(combined_features)

n_train = len(train_features)
vectorized_train_features = combined_features_vectorized[:n_train]
vectorized_test_features = combined_features_vectorized[n_train:]
```

[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.

[ ]:
```
svm_model = svm.LinearSVC()
svm_model.fit(vectorized_train_features, train_labels)

test_predictions = svm_model.predict(vectorized_test_features)
evaluation_report = classification_report(test_labels, test_predictions)
```

```python
# We opted to exclude labels with few examples, as the outcomes derived from␣
 ↪them might not be particularly reliable.
lines = evaluation_report.split('\n')
filtered_lines = []
for line in lines:
    parts = line.split()
    if len(parts) == 5:
        support = int(parts[-1])
        if support >= 10:
            filtered_lines.append(line)
    else:
        filtered_lines.append(line)

# Join the filtered lines back into a string
filtered_evaluation_report = '\n'.join(filtered_lines)
print(filtered_evaluation_report)
```

```
/Users/stefaniaconte/miniconda3/lib/python3.11/site-
packages/sklearn/svm/_classes.py:31: FutureWarning: The default value of `dual`
will change from `True` to `'auto'` in 1.5. Set the value of `dual` explicitly
to suppress the warning.
  warnings.warn(
```

	precision	recall	f1-score	support
B-geo	0.80	0.76	0.78	741
B-gpe	0.96	0.92	0.94	296
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-geo	0.74	0.50	0.60	156
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

```
/Users/stefaniaconte/miniconda3/lib/python3.11/site-
packages/sklearn/metrics/_classification.py:1509: 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))
/Users/stefaniaconte/miniconda3/lib/python3.11/site-
packages/sklearn/metrics/_classification.py:1509: 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))
/Users/stefaniaconte/miniconda3/lib/python3.11/site-
packages/sklearn/metrics/_classification.py:1509: 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))
/Users/stefaniaconte/miniconda3/lib/python3.11/site-
packages/sklearn/metrics/_classification.py:1509: 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))
/Users/stefaniaconte/miniconda3/lib/python3.11/site-
packages/sklearn/metrics/_classification.py:1509: 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))
/Users/stefaniaconte/miniconda3/lib/python3.11/site-
packages/sklearn/metrics/_classification.py:1509: 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))
```

[ ]:

Precision for 'B-art' is 0.00, indicating that the model did not correctly predict any instances of this class. Similar results are seen for 'B-eve,' 'I-art,' and 'I-eve.' High precision values (e.g., 0.96 for 'B-gpe' and 0.98 for 'O') suggest accurate predictions for these classes.

Recall for 'B-geo' (0.76) indicates that the model correctly identified 76% of the instances of this class. Similar results are observed for other classes. However, low recall in classes such as 'B-art,' 'B-eve,' 'I-art,' and 'I-eve' suggests the model struggled to identify instances for these categories.

The F1-score, a balanced measure of precision and recall, shows high values (e.g., 0.94 for 'B-gpe' and 0.99 for 'O') indicating a good balance between precision and recall. Conversely, low F1-scores (e.g., 0.00 for 'B-art' and 'B-eve') suggest poor performance for these classes.

Support, reflecting the number of actual occurrences in the dataset, indicates that classes with low support (e.g., 'B-eve' and 'I-eve') have fewer instances, making the evaluation less reliable for these categories.

The overall accuracy of the model on the test data is 94%, signifying good performance overall.

1.4 End of this notebook