POS_tagging_with_classical_models_handout_marthiensen

November 24, 2023

1 Task: Part of speech tagging

In this task we try to recreate a very rudimentary POS tagger “from scratch” using SpaCy and CRF models.

(We disregard the fact, that SpaCy has a built in POS tagger for the moment for demonstration purposes.)

The input is a tokenized English sentence. The task is to label each word with a part of speech (POS) tag. The tag set, which is identical the Universal Dependencies project’s basic tag set is the following:

- NOUN: noun
- VERB: verb
- DET: determiner
- ADJ: adjective
- ADP: adposition (e.g., prepositions)
- ADV: adverb
- CONJ: conjunction
- NUM: numeral
- PART: particle (function word that cannot be inflected, has no meaning in itself and doesn’t fit elsewhere, e.g., “to”)
- PRON: pronoun
- .: punctuation
- X: other

The code in this task is an adaptation of the NER code in the sklearn-crfsuite documentation.

2 The data set

Brown corpus: “The Brown University Standard Corpus of Present-Day American English (or just Brown Corpus) was compiled in the 1960s by Henry Kučera and W. Nelson Francis at Brown University, Providence, Rhode Island as a general corpus (text collection) in the field of corpus linguistics. It contains 500 samples of English-language text, totaling roughly one million words, compiled from works published in the United States in 1961” (Wikpedia: Brown Corpus)

Let’s download and inspect the data!

[1]:
```
%%capture
!pip install nltk
```

[2]:
```
import nltk

from nltk.corpus import brown
nltk.download('brown')

brown.words()
```

```
[nltk_data] Downloading package brown to
[nltk_data]     /Users/nilsmart96/nltk_data…
[nltk_data]   Package brown is already up-to-date!
```

[2]:
```
['The', 'Fulton', 'County', 'Grand', 'Jury', 'said', …]
```

[3]:
```
nltk.download('universal_tagset')
brown.tagged_words(tagset='universal')
```

```
[nltk_data] Downloading package universal_tagset to
[nltk_data]     /Users/nilsmart96/nltk_data…
[nltk_data]   Package universal_tagset is already up-to-date!
```

[3]:
```
[('The', 'DET'), ('Fulton', 'NOUN'), …]
```

[4]:
```
brown.sents()
```

[4]:
```
[['The', 'Fulton', 'County', 'Grand', 'Jury', 'said', 'Friday', 'an',
'investigation', 'of', "Atlanta's", 'recent', 'primary', 'election', 'produced',
'``', 'no', 'evidence', "''", 'that', 'any', 'irregularities', 'took', 'place',
'.'], ['The', 'jury', 'further', 'said', 'in', 'term-end', 'presentments',
'that', 'the', 'City', 'Executive', 'Committee', ',', 'which', 'had', 'over-
all', 'charge', 'of', 'the', 'election', ',', '``', 'deserves', 'the', 'praise',
'and', 'thanks', 'of', 'the', 'City', 'of', 'Atlanta', "''", 'for', 'the',
'manner', 'in', 'which', 'the', 'election', 'was', 'conducted', '.'], …]
```

[5]:
```
len(brown.words())
```

[5]:
```
1161192
```

From the brown the object provided by NLTK we will work with the tagged sentence list:

[6]:
```
sents = brown.tagged_sents(tagset="universal")

sents[:2]
```

[6]:
```
[[('The', 'DET'),
  ('Fulton', 'NOUN'),
  ('County', 'NOUN'),
  ('Grand', 'ADJ'),
  ('Jury', 'NOUN'),
  ('said', 'VERB'),
  ('Friday', 'NOUN'),
  ('an', 'DET'),
  ('investigation', 'NOUN'),
  ('of', 'ADP'),
  ("Atlanta's", 'NOUN'),
  ('recent', 'ADJ'),
  ('primary', 'NOUN'),
  ('election', 'NOUN'),
  ('produced', 'VERB'),
  ('``', '.'),
  ('no', 'DET'),
  ('evidence', 'NOUN'),
  ("''", '.'),
  ('that', 'ADP'),
  ('any', 'DET'),
  ('irregularities', 'NOUN'),
  ('took', 'VERB'),
  ('place', 'NOUN'),
  ('.', '.')],
 [('The', 'DET'),
  ('jury', 'NOUN'),
  ('further', 'ADV'),
  ('said', 'VERB'),
  ('in', 'ADP'),
  ('term-end', 'NOUN'),
  ('presentments', 'NOUN'),
  ('that', 'ADP'),
  ('the', 'DET'),
  ('City', 'NOUN'),
  ('Executive', 'ADJ'),
  ('Committee', 'NOUN'),
  (',', '.'),
  ('which', 'DET'),
  ('had', 'VERB'),
  ('over-all', 'ADJ'),
  ('charge', 'NOUN'),
  ('of', 'ADP'),
  ('the', 'DET'),
  ('election', 'NOUN'),
  (',', '.'),
  ('``', '.'),
  ('deserves', 'VERB'),
  ('the', 'DET'),
  ('praise', 'NOUN'),
  ('and', 'CONJ'),
  ('thanks', 'NOUN'),
  ('of', 'ADP'),
   ('the', 'DET'),
   ('City', 'NOUN'),
   ('of', 'ADP'),
   ('Atlanta', 'NOUN'),
   ("''", '.'),
   ('for', 'ADP'),
   ('the', 'DET'),
   ('manner', 'NOUN'),
   ('in', 'ADP'),
   ('which', 'DET'),
   ('the', 'DET'),
   ('election', 'NOUN'),
   ('was', 'VERB'),
   ('conducted', 'VERB'),
   ('.', '.')]]
```

[7]:
```
len(sents)
```

[7]:
```
57340
```

We divide our data set into a train and a valid part:

[8]:
```
valid_sents = sents[:5734]
train_sents = sents[5734:]
```

3 Feature template

Since the plan is to build a CRF model, we need a **feature template**, which generates features for a word in a sentence (our sequence in the sequence tagging task). We use spaCy for feature extraction.

[9]:
```
#Spacy install, load and such stuff
# Running locally
# !pip install spacy
# python -m spacy download en_core_web_sm

#Import
import spacy
from spacy.tokens import Doc

#By model load, please deactivate unnecessary pipeline elements!
en = spacy.blank("en")
```

We write a function which generates features for a token in a sentence, which is already a spaCy document. The feature vector is represented as a `dict` mapping feature names to their values.

The desired **feature set for a token is**:

- `bias`: A constant value of 1 as an input

- `token.lower`: the lowercased textual form of the token
- `token.suffix`: the textual form of the token's suffix as defined by SpaCy,
- `token.prefix`: the textual form of the token's prefix as defined by SpaCy,
- `token.is_upper`: boolean value indicating if the token is uppercase,
- `token.is_title`: boolean value indicating if the token is a title,
- `token.is_digit`: boolean value indicating if the token consists of numbers.

These are only the `Token`'s own properties, but they represent no context.

We would like to include information about the previous and next words, as well as indicating if the `Token` is the beginning or the end of sentence.

The **contextual features** should be:

- `-1:token.lower`: What is the lowercase textual form of the previous token?,
- `-1:token.is_title`: Is the previous token a title?,
- `-1:token.is_upper`: Is the previous token uppercase?,
- `+1:token.lower`: What is the lowercase textual form of the next token?,
- `+1:token.is_title`: Is the next token a title?,
- `+1:token.is_upper`: Is the next token uppercase?,
- `BOS`: Boolean value indicating if the token is the beginning of a sentence,
- `EOS`: Boolean value indicating if the token is the end of a sentence

[10]:
```python
def token2features(sent, i):
    """Return a feature dict for a token.
    sent is a spaCy Doc containing a sentence, i is the token's index in it.
    """
    token = sent[i]
    features = {
        'bias': 1.0,
        'token.lower': token.text.lower(),
        'token.suffix': token.suffix_,
        'token.prefix': token.prefix_,
        'token.is_upper': token.is_upper,
        'token.is_title': token.is_title,
        'token.is_digit': token.is_digit,
        '-1:token.lower': sent[i-1].text.lower() if i > 0 else '',
        '-1:token.is_title': sent[i-1].is_title if i > 0 else False,
        '-1:token.is_upper': sent[i-1].is_upper if i > 0 else False,
        '+1:token.lower': sent[i+1].text.lower() if i < len(sent)-1 else '',
        '+1:token.is_title': sent[i+1].is_title if i < len(sent)-1 else False,
        '+1:token.is_upper': sent[i+1].is_upper if i < len(sent)-1 else False,
        'BOS': i == 0,
        'EOS': i == len(sent)-1,
    }

    return features
```

For training, we will also need functions to generate feature dict and label lists for sentences in our training corpus:

[11]:
```
def sent2features(sent):
    "Return a list of feature dicts for a sentence in the data set."
    # Create a doc by instantiating a Doc class and iterating through the␣
 ↪sentence token by token.
    # Please bear in mind, that Brown has token-POS pairs, latter one we don't␣
 ↪need here...
    sent_text = [entry[0] for entry in sent]
    doc = Doc(en.vocab, words=sent_text)
    # Plese use the above defined token2features function on each token to␣
 ↪generate the features
    # For the whole sentence!
    sent_features = [token2features(doc, i) for i in range(len(doc))]

    return sent_features

def sent2labels(sent):
    #Please create / filter only the labels for given sentence!
    labels = [entry[1] for entry in sent]

    return labels
```

Sanity check: let's see the values for the first 2 tokens in the corpus:

[12]:
```
print(sent2features(sents[0])[:2])
print(sent2labels(sents[0])[:2])
```

```
[{'bias': 1.0, 'token.lower': 'the', 'token.suffix': 'The', 'token.prefix': 'T',
'token.is_upper': False, 'token.is_title': True, 'token.is_digit': False,
'-1:token.lower': '', '-1:token.is_title': False, '-1:token.is_upper': False,
'+1:token.lower': 'fulton', '+1:token.is_title': True, '+1:token.is_upper':
False, 'BOS': True, 'EOS': False}, {'bias': 1.0, 'token.lower': 'fulton',
'token.suffix': 'ton', 'token.prefix': 'F', 'token.is_upper': False,
'token.is_title': True, 'token.is_digit': False, '-1:token.lower': 'the',
'-1:token.is_title': True, '-1:token.is_upper': False, '+1:token.lower':
'county', '+1:token.is_title': True, '+1:token.is_upper': False, 'BOS': False,
'EOS': False}]
['DET', 'NOUN']
```

4 Putting the data into final form

Everything is ready to generate the training data in the form which is usable for the CRFsuite. Note that our inputs and labels will be 2-level representations, lists of lists, because we deal with token sequences (sentences).

[13]:
```
%%time
X_train = [sent2features(s) for s in train_sents]
y_train = [sent2labels(s) for s in train_sents]
```

```
X_valid = [sent2features(s) for s in valid_sents]
y_valid = [sent2labels(s) for s in valid_sents]
```

```
CPU times: user 7.27 s, sys: 288 ms, total: 7.56 s
Wall time: 7.56 s
```

[14]:
```
print("Feature dict for the first token in the first validation sentence:")
print(X_valid[0][0])
print("Its label:")
print(y_valid[0][0])
```

```
Feature dict for the first token in the first validation sentence:
{'bias': 1.0, 'token.lower': 'the', 'token.suffix': 'The', 'token.prefix': 'T',
'token.is_upper': False, 'token.is_title': True, 'token.is_digit': False,
'-1:token.lower': '', '-1:token.is_title': False, '-1:token.is_upper': False,
'+1:token.lower': 'fulton', '+1:token.is_title': True, '+1:token.is_upper':
False, 'BOS': True, 'EOS': False}
Its label:
DET
```

5 Training and evaluation

We use the super-optimized CRFsuite via the scikit-learn compatible sklearn-crfsuite wrapper to train a CRF model on the data.

[15]:
```
%%capture # only to avoid ugly printouts during install
!pip install sklearn_crfsuite
```

[16]:
```
# Please import and train an averaged perceptron model from CRFsuite and use␣
↪it's custom metrics,
# especially the multiple forms of accuracy score to evaluate the model!

import sklearn_crfsuite
from sklearn.metrics import classification_report, f1_score

# Initialize the CRF model
crf = sklearn_crfsuite.CRF(
    algorithm='ap',  # Averaged Perceptron
    max_iterations=100,
    all_possible_transitions=True
)

# Train the model
crf.fit(X_train, y_train)

# Make predictions on the test data
y_pred = crf.predict(X_valid)
```

```
# Flatten the true and predicted labels for evaluation (Tokens)
y_test_token = [label for sent_labels in y_valid for label in sent_labels]
y_pred_token = [label for sent_preds in y_pred for label in sent_preds]

# Evaluate the model using custom metrics
# Token Classification Report
print("Token-wise classification report:")
print(classification_report(y_test_token, y_pred_token))

# Sentence-wise F1 Score for comparison
sent_f1_sum = 0
for true_labels_sentence, pred_labels_sentence in zip(y_valid, y_pred):
    f1_sentence = f1_score(true_labels_sentence, pred_labels_sentence,␣
 ↪average="micro")
    sent_f1_sum += f1_sentence
avg_sent_f1 = round(sent_f1_sum / len(y_valid), 2)
print(f"Average Sentence-wise F1 Score: {avg_sent_f1}")
```

```
Token-wise classification report:
              precision    recall  f1-score   support

           .       1.00      1.00      1.00     14377
         ADJ       0.91      0.92      0.92      8525
         ADP       0.98      0.98      0.98     15138
         ADV       0.93      0.93      0.93      4438
        CONJ       0.99      1.00      1.00      3435
         DET       1.00      1.00      1.00     14128
        NOUN       0.98      0.97      0.98     36341
         NUM       0.99      0.98      0.98      2386
        PRON       0.99      0.99      0.99      3427
         PRT       0.94      0.93      0.94      2877
        VERB       0.97      0.98      0.97     18229
           X       0.87      0.59      0.70       100

    accuracy                           0.98    123401
   macro avg       0.96      0.94      0.95    123401
weighted avg       0.98      0.98      0.98    123401

Average Sentence-wise F1 Score: 0.97
```

[17]:
```
# Please draw some conclusion if this model is "good enough"
# in your view if you take token level and sentence level metrics into account!
```

Based on the provided results for POS tagging, the model appears to perform quite well. Let's analyze the key metrics and draw some conclusions:

1. **Accuracy:**
    - The overall accuracy of the model is 98%, which is high. This suggests that the model correctly predicts the part-of-speech tags for the majority of tokens in the dataset.

2. **Macro and Weighted Averages:**
    - There is a noticable difference between the macro and the weighted average values, with the weighted average being a few percentage points stronger for Precision, Recall and F1 respectively. This indicates strong overall performance across all classes, with more emphasis on the larger classes.
3. **Token Level and Sentence Level Metrics:**
    - At the token level, the precision, recall, and F1-scores are provided for each POS tag, giving a detailed view of the model's performance on individual tokens.
    - Some classes, such as "X", "PRT", "ADJ" or "ADV" have lower scores. All of these appear less often in the dataset, as can be seen in the support column. It could be the case that these metrics are improved with more training data. This is supported by the high weighted average values for Precision, Recall and F1 as previously mentioned.
    - The average sentence-wise F1 score is also high (0.97), indicating that the model performs well at the sentence level. This metric is particularly important as it assesses how well the model captures the sequential dependencies and overall structure of sentences.
4. **Consideration of "Good Enough":**
    - The model seems to be performing well, especially considering the high precision, recall, and F1-scores across most POS tags.
    - Whether the model is "good enough" depends on the specific requirements of your application. In many NLP tasks, achieving an accuracy of around 98% would be considered quite good.
    - It's essential to consider the practical implications of the errors made by the model. For example, if misclassifying certain POS tags has a significant impact on downstream tasks, further improvements may be necessary.
    - If possible, a larger dataset could help mitigate the worse performance of some of the less frequently used tags.

In conclusion, the provided POS tagging model seems to be quite effective. Sentence and token level performance is on a similarly high level. Whether it is good enogh depends on the specific use case, but I deem it good enogh for this assignment.

Let's instantiate and fit our model. CRFsuite implements several learning methods, here we use "ap", i.e., averaged perceptron.

6 Demonstration

Just for the fun, we can try out the model.

[18]:
```python
def predict_tags(sent):
    """Predict tags for a sentence.
    sent is a string.
    """
    doc = en(sent)
    return crf.predict([[token2features(doc, i) for i in range(len(doc))]])
```

[19]:
```python
while True:
    sent = input("\nEnter a sentence to tag or press return to quit:\n")
    if sent:
        print(predict_tags(sent))
    else:
        print("\nEmpty input received -- bye!")
        break
```

```
[['DET', 'NOUN', 'VERB', 'DET', 'NOUN', 'ADP', 'DET', 'NOUN', 'ADP', 'PRON',
'.']]
[['ADV', 'PRON', 'PRON', 'ADV', 'VERB', '.']]
[['NOUN', 'NUM', 'NOUN', 'VERB', 'ADV', 'ADJ', 'CONJ', 'ADJ', '.']]

Empty input received -- bye!
```