[COM4513-6513] Lab 3: Named Entity Recognition with the Structured Perceptron

Instructor: Nikos Aletras

Teaching Assistants: George Chrysostomou, Hardy and Zeerak Waseem

Aim

The goal of this lab session is to learn a named entity recogniser (NER) using the structured perceptron. For each word in a sequence, the named entity recogniser should predict one of the following labels:

- O: not part of a named entity
- PER: part of a person's name
- LOC: part of a location's name
- ORG: part of an organisation's name
- MISC: part of a name of a different type (miscellaneous, e.g. not person, location or organisation)

Load the Data

The training and the test data are available in this folder (use your university google ID to access it): data. It consists of sentences up to 5 words long each from the data used in this shared task on NER.

After downloading the data, you can obtain the word and tag sequences for each sentence using the following function:

```python
def load_dataset_sents(file_path, as_zip=True, to_idx=False, token_vocab=None, target_vocab=None):
    targets = []
    inputs = []
    zip_inps = []
    with open(file_path) as f:
        for line in f:
            sent, tags = line.split('\t')
            words = [token_vocab[w.strip()] if to_idx else w.strip() for w in sent.split()]
            ner_tags = [target_vocab[w.strip()] if to_idx else w.strip() for w in tags.split()]
            inputs.append(words)
            targets.append(ner_tags)
            zip_inps.append(list(zip(words, ner_tags)))
    return zip_inps if as_zip else (inputs, targets)
```

e.g., `train_data = load_dataset_sents('train-sents.txt')`

Feature Extraction

After having loaded the data, you need to write functions that extract and transform a sentence into different feature types (this is the $\Phi(x, y)$ in the Perceptron algorithm below) in the form of Python dictionaries (see lecture 4, part I slides) for a given sentence:

1. **Current word-current label, $\Phi_1(x, y)$ (1 mark):**
   - (1) Write a funtion that takes as input a corpus $c = [s_1, s_2, ..., s_n]$ where $s_i = [(x_1, y_1), ..., (x_2, y_2)]$ is a sentence consisting of pairs of words x and tags y, and returns a Python dictionary containing current word-current label counts for each pair of words and tags in the corpus, e.g. cw_cl_counts = {'x1_y1':103, 'x2_y2':12, ...}. **Tip:** Keep only the features (dict. keys) that have a frequency greater or equal to 3 (you can delete keys in a Python dictionary) if your feature space is too big to keep in memory.
   - (2) Write a function `phi_1(x, y, cw_cl_counts)` that takes as input a sentence and returns a dictionary with counts of the `cw_cl_counts` keys in the given sentence (x, y).

2. **Previous label-current label** $\Phi_2(x, y)$ (**1 mark**)**:** You need to write two functions following a similar two-step approach as you did for $\Phi_1(x, y)$ above. After you have the feature extraction functionality in place, you will need to combine Φ_2 with Φ_1 to see whether the new feature type improves performance. This can be done by merging the two Python dictionaries you obtain for each feature type.
3. **Optional Bonus** (**2 marks**) Add at least two more feature types ($\Phi_3(x, y)$ and $\Phi_4(x, y)$) of your choice following the same approach as for Φ_1 and combine them with Φ_1 and Φ_2 to test if you obtain improved performance. Ideas: sub word features, previous/next words, label trigrams, etc..

Tip: For the feature extraction part, you can use the `Counter` and Python dictionaries to store the feature names as keys with their corresponding value (see the slides of lecture 4.1). **Tip++:** The feature representation should be obtained by looking only into the training data, otherwise your model is cheating!

Implement the Perceptron

Finally after you have your feature extraction mechanism in place, you need to implement a structured perceptron (**2 marks**). This has to be done using two functions. One function is called `train` and takes as input the training data as in:

```
Input: $D_{train} = \{(\mathbf{x}^1, \mathbf{y}^1)...(\mathbf{x}^M, \mathbf{y}^M)\}$
set $\mathbf{w} = 0$
for $(\mathbf{x}, \mathbf{y}) \in D_{train}$ do
  predict $\hat{\mathbf{y}} = \arg\max_{\mathbf{y} \in \mathcal{Y}^{\mathcal{N}}} \mathbf{w} \cdot \Phi(\mathbf{x}, \mathbf{y})$
  if $\hat{\mathbf{y}} \neq \mathbf{y}$ then
    update $\mathbf{w} = \mathbf{w} + \Phi(\mathbf{x}, \mathbf{y}) - \Phi(\mathbf{x}, \hat{\mathbf{y}})$
return $\mathbf{w}$
```

The second function should be called `predict` that takes as input the current weights w, and a feature representation of a sentence ($\Phi(x, y)$) and returns the predicted tag sequence $\hat{y}$ using the argmax over all the possible tag sequences.

During training of the structured perceptron, you are advised to use multiple passes with randomised order and averaging.

Evaluate the Perceptron with Different Feature Types

Write a `test` funtion to evaluate the perceptron. The input should be the trained weights ($\mathbf{w}$) of your perceptron and test data. As the dataset is imbalanced (most words are not part of a named entity), the evaluation metric to use is the micro-averaged F1 score from scikit learn: http://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html. You should call it in your code as:

```
f1_micro = f1_score(correct, predicted, average='micro', labels=['ORG', 'MISC', 'PER', 'LOC'])
```

where `y_true` and `y_predicted` are arrays containing the correct and the predicted labels. These arrays are flatten and contain all the tags for all sentences in the test data (true and predicted respectively).

Tip: You should re-use the **predict** function to get the prediction for a given test sentence by iterating over all the possible tag sequences for its words.

For each feature set you use (Φ_1, Φ_1+Φ_2 and the bonus features if you opt in), you need to answer the following questions:

- What is the F1 score you obtain for each feature set?
- What are the most positively-weighted features for each feature set? Give the top 10 for each class (tag) and comment on whether they make sense. (If they don't you might have a bug.)
- Are the differences among the feature sets in micro-F1 score expected? Did the features you propose improve the results? Why?

Correct answers to questions count **2 marks for each feature set** you have implemented (**+1** for the bonus features).

Tip: You can use barplots (see Python's matplotlib library) and tables to visualise/summarise the results in your report.

Submission Details

You should submit a python file (lab3.py) that can be executed as:

```
python3 lab3.py train.txt test.txt
```

and returns the micro F1 for the two feature sest (three if you opt in for the bonus) versions of the structured peceptron, and the top-10 most positive features for each version. Use python's `random` library to fix the random seed so that your results are reproducible! Make sure your code is **Python 3** compatible.

You also need to accompany the code with a lab3.pdf (no more than two A4 pages) answering the questions above. Note that you can use an extra half page in you report if you decide to implement the optional bonus.

This lab will be marked out of 8. It is worth 8% of your final grade in the module. Note that bonus marks will be added to your final mark for the lab but that will be capped at 8.

The deadline for this assignment is the beginning of the Week 7 lab and it needs to be submitted via MOLE. Standard departmental penalties for lateness will be applied.