NEC Features exercise

1 Named-Entity Classification: Feature Templates

Think of named entities, like "Michael Jackson", "Barcelona", or "United Nations". We can distinguish between different types of entities. For example, "Michael Jackson" is a PERSON, "Barcelona" is a LOCATION and "United Nations" is an ORGANIZATION. Like in all NLP problems, we will frequently encounter difficult cases, such as

```
[ Jack London ] went to Paris.
```

where "Jack London" is a PERSON rather than a LOCATION despite containing a well known location name. There are cases of ambiguity, such as

```
[ Spain ] won the championship.
```

where "Spain" is an ORGANIZATION because in this case it refers to a team or country, not a geographical location. Hence we would like to use linear classification methods that predict the most appropriate class. Let Y be the set of named entity classes (three in our example). In this exercise we will assume that a previous process has identified the boundaries of named entities, and our goal is to classify them (later in the course we will see how to perform identification and classification jointly). We will make the following linear predictions

$$\text{ne\_type}(x_{1:n}, i, j) = \underset{y \in Y}{\text{argmax}}\, \mathbf{w} \cdot \mathbf{f}(x_{1:n}, i, j, y)$$

where

- $x_{1:n}$ is a sentence with n tokens (x_i is the i-th token)
- i and j are the first and last positions of the entity; thus $x_{i:j}$ is the complete entity
- y is one of the entity types
- $\mathbf{f}(x_{1:n}, i, j, y)$ is a function that returns a feature vector
- $\mathbf{w}$ is a vector of parameters, with one weight per feature

We are interested in creating a set of features for the classifier. For example, feature 1 may indicate if the current entity is the single-word entity "London" and is tagged as LOC (location):

$$\mathbf{f}_1(x_{1:n}, i, j, y) = \begin{cases} 1 & \text{if } i = j \text{ and } x_i = \text{"London" and } y = \text{LOC} \\ 0 & \text{otherwise} \end{cases}$$

Rather than specifying each dimension explicitly, we will create *feature templates* that generate a number of features mechanically. Each feature template is identified by a type. For example, the template with type 1 may be

$$\mathbf{f}_{1,l,a}(x_{1:n}, i, j, y) = \begin{cases} 1 & \text{if } i = j \text{ and } x_i = a \text{ and } y = l \\ 0 & \text{otherwise} \end{cases}$$

and will generate an actual feature for every $l \in Y$ and every word $a \in V$ (where V is the set of words in our language). If we set $l = \text{LOC}$ and $a = \text{London}$ we obtain the feature above. In general, this template will create feature dimensions identified by a tuple of the form $\langle 1, l, a \rangle$. We will assume that the parameter vector $\mathbf{w}$ is indexed with the same type of tuple dimensions. For example, $\mathbf{w}_{\langle 1,\text{LOC},\text{London} \rangle}$ is the parameter associated with the feature above.

1.1 Question 1: Feature Templates

Write feature templates that capture the following information. For each template, specify the tuple that identifies the feature dimensions. Justify your answers if necessary

- Type 1: The entity is word a
  Answer:

$$\mathbf{f}_{1,l,a}(x_{1:n}, i, j, y) = \begin{cases} 1 & \text{if } i = j \text{ and } x_i = a \text{ and } y = l \\ 0 & \text{otherwise} \end{cases}$$

- Type 2: All entity words are capitalized.
- Type 3: The entity contains word a.
- Type 4: The entity has at least three words, the first is a and the second is b.
- Type 5: Word a appears right before or right after the entity.

1.2 Question 2: Feature Vectors

Assume we define a feature function with the templates 1 to 5 above. Assume also that our vocabulary V includes all English words. For each case below, write the indices of the features that are non-zero:

1. $x_{1:5} =$ `I live in Barcelona .`
   $\mathbf{f}(x_{1:5}, 4, 4, \text{LOC}) = \langle 1, \text{LOC}, \text{Barcelona} \rangle, \ldots$
2. $x_{1:12} =$ `Smith wrote in the Journal of the Royal College of Physics .`
   $\mathbf{f}(x_{1:12}, 1, 1, \text{PER}) =$?
   $\mathbf{f}(x_{1:12}, 5, 11, \text{ORG}) =$?
3. $x_{1:6} =$ `Spain won the World Cup .`
   $\mathbf{f}(x_{1:6}, 1, 1, \text{ORG}) =$? $\mathbf{f}(x_{1:6}, 1, 1, \text{LOC}) =$?
   $\mathbf{f}(x_{1:6}, 4, 5, \text{ORG}) =$?

1.3 Question 3: Learning to weight features

We will now play the role of a learning algorithm by setting the weights of features. The main point is to show that, while feature templates generate a large number of actual features, a learning algorithm should be able to select a few features which are highly discriminative.

Let's start with a simple example. Assume our training set has two examples:

`[ Barcelona ]`$_{\text{LOC}}$ `is a beautiful city .`
`[ Barcelona ]`$_{\text{ORG}}$ `won the game .`

Assume we use feature templates 1–5 defined above. We need to set some weights such that the training set is perfectly classified, and such that the norm of the weight vector is not too large (in this exercise we will make training error to be zero, and try to set a few non-zero weights). A possible weight vector could be:

$$w_{\langle 1, \text{LOC}, \texttt{Barcelona} \rangle} = 1$$
$$w_{\langle 5, \text{ORG}, \texttt{won} \rangle} = 2$$

Thus, with these weights, we have *learned* that `Barcelona` is a location, but that words surrounded by `won` should have a higher positive score for being organizations. Note that this knowledge is acquired via learning: the feature templates do not have prior knowledge about what words are locations or surround organizations.

Assume the following training set:

[Maria]$_{\text{PER}}$ is beautiful .
[Barcelona]$_{\text{LOC}}$ is beautiful .
[Jack London]$_{\text{PER}}$ is nice .
[Milan]$_{\text{LOC}}$ is nice .
[Jack Smith]$_{\text{PER}}$ lives in [Great Yarmouth]$_{\text{LOC}}$.
[Maria Domenech]$_{\text{PER}}$ works in [Barcelona]$_{\text{LOC}}$.
[Maria Muschatta]$_{\text{PER}}$ attends school in [Santa Maria Della Mole]$_{\text{LOC}}$.
[Barcelona]$_{\text{ORG}}$ won [Milan]$_{\text{ORG}}$.

Question: Using feature templates 1–5, set a parameter vector with as few features as possible.