Text Analytics 2017
Homework 2

Submission

Submit a Jupyter notebook file by email to attardi@di.unipi.it, with your solution, including code and results.

Spell Checker

A spell checker tries to identify misspelled words and to suggest the most likely corrections. As training data, you can use the following file:

```
http://www.gutenberg.org/cache/epub/1661/pg1661.txt
```

Write a `spell_checker` function that takes as input a word and returns a list of correct form suggestions. If the word is correct, the first word in the list should be the word itself. An empty list is considered an error.

For example:

```
> spell_checker('wordd')
['word','world']
> spell_checker('pinting')
['painting','printing','pointing','pinning','pinging']
> spell_checker('itsself')
['itself']
```

HINT: use the edit distance to compare the input word with candidate correct words (e.g. present in the training data), and use the word frequencies for sorting the list of suggested words.

Test your corrector on this sample file:

http://aspell.net/test/cur/batch0.tab

Evaluate your results using this function:

```
def test(spell_checker, test_data):
    score = 0
    for wrong, correct in test_data:
        suggestions = spell_checker(wrong)
        if correct in suggestions:
            position = suggestions.index(correct)
            if position >= 0:
                score += 1/(position+1)
    return score / len(test_data)
```

POS Tagger

The following URLs contain the train and test set for the Evalita 2009 POS task:

http://medialab.di.unipi.it/evalita/agreement.php?corpus=PoSTaggingCorpus.tgz
http://medialab.di.unipi.it/evalita/corpus/PoSTest.tanl.tar

The input consists of sentences, one token per line, separated by one empty line.

Each line contains three, tab separated, fields:

```
FORM	POSTAG
```

For example:

```
Il	RDms
presidente	Sms
Sandro	SP
Pertini	SP
rifiuta	Vip3s
anche	B
la	RDfs
sola	Afs
idea	Sfs
di	E
un	RIms
accordo	Sms
preventivo	Ams
```

Train a POS tagger on these data using the nltk module classify.maxent.

This involves defining a set of features to provide to the classifier representing a training event for each token. For example, such features may include orthographic features, such as whether the word starts with a capital letter, or contains a number, or any hyphenation, or position of the token the sentence. You can use regular expressions to check patterns in the words. You can also use feature conjunctions, such as whether the word is capitalized AND the value of the previous label.

To train the classifier, you should create a vector of training examples. Each example consists in a pair:

```
(features, label)
```

where `features` is a dictionary mapping strings to `boolean` representing the presence or absence of that feature. For example, you may represent the feature for the current word being capitalized by the string '`CurCap`', and the previous word being capitalized as '`PrevCap`'.

Create a classifier on those examples:

```
classifier = trainer(examples)
```

Use the same feature extractor for implementing the tagger, and check its accuracy using this program:

```
http://medialab.di.unipi.it/evalita/poseval.py
```

Sequence Tagger

For sequence tagging you should train a maximum entropy model as before, including among the features the correct label for the previous word. For example, the string '`PrevPos=SP`' represents the feature of a word whose preceding token has POS tag '`SP`'.

At test time, the tagger will use the predicted tag for the previous word and exploit a Viterbi decoder to find the best possible sequence of labels, instead of just choosing the best label at each step.

Write such decoder. Compare the accuracy of the sequence tagger with the one from the previous exercise.