Introduction to Human Language Technologies
3. Morphology

Natural Language Research Group

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

Course 2018/19

Outline

Session requirements

skearn:

- Linux (via shell)
  - > `pip3 install sklearn`
- Windows (via cmd)
  - > `pip install sklearn`

PoS tagger & lemmatizer:

- Both Linux & Windows (via python shell)
  - > `import nltk`
  - > `nltk.download('averaged_perceptron_tagger')`
  - > `nltk.download('wordnet')`

Attached resources:

- `trial.tgz`: trial set of the project
- `wordsEn.txt`: list of english words
- `smsspamcollection.zip`: sms data set

Outline

Lexical level in nltk library

- Words

  Tokenization achieves words. Multiwords (e.g. "Even though") are not recognized. MWETokenizer is not useful.

- POS tags

  ```
  t_POS_list = nltk.pos_tag(t_list)
  ```

- Lemmas

  ```
  from nltk.stem import WordNetLemmatizer
  wnl = WordNetLemmatizer()
  wnl.lemmatize(token, pos=[POS])
  ```

  POS can be: 'n','v', ...

- Senses

  We will see in the sessions related to WSD and WordNet.

Example in NLTK

Part of Speech:

In [1]:
```
from nltk import pos_tag

words = ['Women','want','children']
pairs = pos_tag(words)
pairs
```

Out[1]:
```
[('Women', 'NNP'), ('want', 'VBP'), ('children', 'NNS')]
```

WordNet lemmatizer:

In [2]:
```
from nltk.stem import WordNetLemmatizer

wnl = WordNetLemmatizer()

def lemmatize(p):
    if p[1][0] in {'N','V'}:
        return wnl.lemmatize(p[0].lower(), pos=p[1][0].lower())
    return p[0]
```

In [3]:
```
[lemmatize(pair) for pair in pairs]
```

Out[3]:
```
['woman', 'want', 'child']
```

Mandatory exercise

Statement:

1. Read all pairs of sentences of the trial set within the evaluation framework of the project.
2. Compute their similarities by considering lemmas and Jaccard distance.
3. Compare the results with those in session 2 (document) in which words were considered.
4. Compare the results with gold standard by giving the pearson correlation between them.
5. Which is better: words or lemmas? Do you think that could perform better for any pair of texts? Justify the answer.

Outline

Levenshtein edit distance

- The edit distance is the number of characters that need to be substituted, inserted, or deleted, to transform the first string to the second one.
- Example:
  "rain" $\rightarrow$ "sain" $\rightarrow$ "shin" $\rightarrow$ "shine"
  It is needed at least 3 steps.
- NLTK example:

```
In [1]: from nltk.metrics.distance import edit_distance

        edit_distance('something', 'soothing')

Out[1]: 2
```

Optional exercise

The first approach to perform *spelling correction* needs a list of words such as the attached file `wordsEn.txt:`
(`http://www-01.sil.org/linguistics/wordlists/english/`)
The input of the process should a *word* to be corrected.
The output of the process should be the same word if it is included in the previous word list; or the word in the list with minimum edit distance to the input word if it do not belong to the list.

- Read the word list from the attached file
- Implement this basic approach
- Use the approach to correct the words: *something*, *soemthing* and some other of your choose.

towards a real approach

In order to implement a real system, some issues should be studied:

- *edit distance* is a high time consuming function.
  generating the candidates for only search among them
- In an interactive solution the output should be those words with minimum distance.
  `sat` $\rightarrow$ `set, sit, sad`
- For an automatic solution we need some kind of language model
  Resource: *Birkbeck spelling error corpus* from *the Oxford Text Archive*.
- New approaches use context information

See the Peter Norvig's article *How to Write a Spelling Corrector* (`http://www.norvig.com/spell-correct.html`) for delving into the topic.

Outline

Text categorization & machine learning

Spam filtering is a type of *text categorization*, such as *language identification* and *sentiment analysis*.

- Classification algorithm in Machine Learning
- How we represent text by means of features?
  - *bag of words*: boolean codifying the presence of the indexed words (`nltk.FreqDist`)
  - *term frequency (tf)*: frequency of the indexed word in the sentence (`nltk.text.tf`)
  - *term-frequency times inverse document-frequency (tf/idf)*: as above re-weighting to avoid effect of too common words such as english word *the* (`nltk.text.tf_idf`)
- `sklearn`: great Machine Learning library for python
  `http://scikit-learn.org/stable/modules/feature_extraction.html#text-feature-extraction`

SMS Spam Collection Data Set

- source: UCI repository
  `https://archive.ics.uci.edu/ml/datasets/SMS+Spam+Collection`
  - 5574 examples
  - 2 classes: *ham* and *spam*
  - Example:
    `ham    What you doing?how are you?`
- experiment design
  - single validation (50% - 50%)
  - randomly shuffle
  - punctuation removed
  - strings lowered

Text categorization in sklearn (I)

Bag of words:

In [43]:

```
from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer()
Xtrn = cv.fit_transform([' '.join(ex[1]) for ex in train])
Xtst = cv.transform([' '.join(ex[1]) for ex in test])
Ytrn = [ex[0] for ex in train]
Ytst = [ex[0] for ex in test]
```

k Nearest Neighbors:

In [46]:

```
from sklearn.neighbors import KNeighborsClassifier

clf = KNeighborsClassifier(1)
clf.fit(Xtrn, Ytrn)
preds = clf.predict(Xtst).tolist()
round(accuracy(refs, preds), 3)
```

Out[46]:

```
0.94
```

Text categorization in sklearn (II)

Confusion matrix:

```
In [47]:  1  print(ConfusionMatrix(refs, preds).pretty_format())

     |         s |
     |    h    p |
     |    a    a |
     |    m    m |
-----+-----------+
 ham |<2416>   . |
spam |  167 <204>|
-----+-----------+
(row = reference; col = test)
```

Text categorization in sklearn (III)

Support Vector Machines:

In [48]:
```
from sklearn.svm import SVC

clf = SVC(kernel='linear')
clf.fit(Xtrn, Ytrn)
preds = clf.predict(Xtst).tolist()
round(accuracy(refs, preds), 3)
```

Out[48]: 0.973

Confusion matrix:

In [49]:
```
print(ConfusionMatrix(refs, preds).pretty_format())
```

```
     |         s |
     |    h    p |
     |    a    a |
     |    m    m |
-----+-----------+
 ham |<2410>   6 |
spam |   68 <303>|
-----+-----------+
(row = reference; col = test)
```

Adapting ML algorithms to the problem

- *k Nearest Neighbors*: a distance such as *jaccard* should be used on kNN and all the methods based on distances:

In [29]:
```python
def kNN(ex, d):
    return min(train,key=lambda x: d(ex[1], x[1]))[0]

from nltk.metrics.distance import jaccard_distance
def jaccard(a, b):
    return jaccard_distance(set(a), set(b))
```

- *Support Vector Machines*
  1. Define a user kernel for sets:

$$\kappa(s_1, s_2) = 2^{|s_1 \cap s_2|}$$

  2. Define a user kernel for tf/idf:

$$\kappa(d_1, d_2) = \sum_t w(t)^2 \mathrm{tf}(t, d_1)\mathrm{tf}(t, d_2)$$

where tf is the term frequency, $w(t) = \frac{l}{\mathrm{df}(t)}$, l the number of documents and $\mathrm{df}(t)$ is the number of documents which contains the term t.

Optional exercise

Statement:

- Implement some of the approaches above and apply then to the *SMS Spam Collection* data set.
    - preprocessing text components and ML algorithms in sklearn
    - kNN with jaccard distance
    - set kernel for SVMs (requires advanced skills in *Machine Learning*)
    - tf/idf kernel for SVMs (requires advanced skills in *Machine Learning*)
- Extend the solution to the use of lemmas and other preprocess issues.