Chapter 4

Word Sense Representation and Disambiguation

> *But it must be recognized that the notion "probability of a sentence" is an entirely useless one, under any known interpretation of this term.* —Noam Chomsky (1969)

4.1 What is Word Sense Disambiguation?

The generally accepted definition of word sense disambiguation is the following. We are given as input:

- a content word W in *context* of a sentence $CONT$.
- all potential senses *senses*(W) from a sense inventory (e.g., the WordNet)

And we are supposed to identify the intended sense of W in $CONT$ from a *senses*(W).

If we could perform WSD reliably on a large *scale*, the results definitely will help improve many important NLP tasks, including Machine translation, Question Answering, and even determine the level of difficulty of words in a reader for learning a second language. For instance, according to $EnglishVocabularyProfile$, the word 'earn' has three distinctive meanings reflecting increasing language proficiency:

1. **A2** to receive money for doing work. *She earns about $50,000 a year.*
2. **B2** to have enough money for the things you need *You can't expect to earn a living from your painting.*
3. **C2** to get something that you deserve because of your work. *As a teacher you have to earn the respect of your students.*

This simple definition of WSD can be extended in many ways:

標記一個文章中所有單字的sense(?

- The context could be a **paragraph** or even **document**.
- The word senses in the inventory can be given a semantic **category**, so the WSD task can be tackled through classifying the given context into the category representing the intended senses.

There are four approaches to solving the WSD problem:

- **Supervised approach**: This approach requires a training corpus (e.g., *Semcor*, `http://www.cse.unt.edu/~rada/downloads.html#semcor`) of words tagged in context with intended senses. The training corpus can then be used to train a classifier that can tag words in a given new context. 訓練出來的model可以根據上下文來標記每個字的sense
- **Dictionary-based approach** (Lesk Algorithm): Choose sense with a definition that contains the most word overlap with the given context.
- **Semi-Supervised approach**: Start with a very small hand-labeled seed-set (sentences/collocations associated with each word sense). Then rely on the "one sense per collocation" constraint to tag some sentences. Subsequently, identify more collocations in order to extend the sized of tagged part of the training data, until no more sentences can be tagged.

- **Unsupervised approach**: combined the information in Roget's International Thesaurus with co-occurrence data from large corpora in order to learn disambiguation rules for Roget's classes, which could then be applied to words.

4.2 Lesk Algorithm

The most simple WSD algorithm is the well-known Lesk Algorithm (**LA**) based on word-level information in dictionary definitions of word meanings. In a nutshell, **LA** compares each word in a given sentence with defining words of each senses of the words in questions, and choose the sense that shares the most words with the given sentence.

Consider a sentence "*You may not have realized **pine cones** can move because they do so*
從這句話判斷出pine cones這兩個字的sense
slowly, and only in response to humidity change.", to illustrating this algorithm. For that, we use the following dictionary definitions are used as the sense inventory for PINE and CONE:

1. **PINE.1** kinds of **evergreen tree** with needle-shaped leaves
2. **PINE.2** waste away through sorrow or illness
3. **CONE.1** solid body which narrows to a point
4. **CONE.2** something of this shape whether solid or hollow
5. **CONE.3** fruit of certain **evergreen trees**

Kilgarriff and Rosenzweig (2000) propose a simplest version of the algorithm, often called the Simplified Lesk algorithm. As an example, consider disambiguating the word **bank** in "*The bank can guarantee **deposits** will eventually cover future tuition costs because it invests in adjustable-rate **mortgage** securities.*", using WordNet as the sense inventory:

1. **BANK.1** sloping land (especially the slope beside a body of water) *"they pulled the canoe up on the bank"; "he sat on the bank of the river and watched the currents"*

2. **BANK.2** a financial institution that accepts **deposits** and channels the money into lending activities *"he cashed a check at the bank"; "that bank holds the **mortgage** on my home"*

Therefore, we determine the intended sense is **BANK.2** based on the overlapped words, **deposits** and **mortgage**.

把一些太常見的字從document拿掉，讓他們不要影響判斷的結果
用tf–idf來決定要拿掉哪些字

Yet, another extension, called **Corpus Lesk Algorithm**, uses a sense-tagged corpus of annotated sentences in the place of dictionary definitions and examples. However, we have to filter and weight context words to obtain better results. In addition to removing function words, we have to weigh each word by its salience across documents. In other words, we have to down-weights words that occur in almost every document (example sentences for a certain word sense) according to **term frequency** (tf) and **inverse document frequency** (idf) commonly used in information retrieval.

4.3 WSD based on Corpus and Thesaurus Categories

統計字跟類別的關係

Yarowsky (1992) describes a program that disambignates English word senses using statistical models of context words appearing near a word belonging to each of 1,000 Roget's Thesaurus category.

Roget's categories correspond to typical sense distinction in WSD studies. Therefore, selecting the most likely category provides a useful level of sense disambiguation. Yarowsky (1992) identifies and weights context words that are indicative of each category based on a Bayesian framework. The method involves training on untagged text without human intervention. Applied to the 10 million word Grolier's Encyclopedia, the system correctly disambiguated 92% of the instances of 12 polysemous words.

More specifically, this WSD method uses classes of words (untagged) to derive models useful for disambignating individual words in context. For the purposes of this method, the senses of each (content) word are defined as the categories listed for that word in Roger's International Thesaurus (Fourth Edition - Chapman, 1977). Sense disambiguation will constitute selecting the listed category which is most probable given the surrounding context.

The Roget categories are very effective in partitioning the major senses Yarowsky (1992) focus on sense distinctions within nouns. For this purpose. a stochastic part-of-speech tagging should be done as a preprocessing step.

4.4 Work Sheet

Write a program to perform word sense disambiguation based roughly on Yarowsky (1992). For this, you will be given the follow datasets and seed program:

- A **training dataset (wn.in,evp.cat,txt)** contain training instances from the WordNet. Each line contain a training instance, with WordNet id, answer, sense definition, and target (candidate) answers.
- A **test dataset (evp.in,wn.cat.txt)** contain test cases from the English Vocabulary Profile. Each line contain a test case id, answer, sense definition, and target (candidate) answers.
- Two seed programs:
  - Lesk Algorithm (**simple.lesk.py**)
    - * **Input:** The training and test datasets
    - * **Output:** The intended WordNet sense of each EVP items (**simple.lesk.results.py**)

      Note that we use categories are based on Basic Level Concepts available at (`http://adimen.si.ehu.es/web/BLC`. BLC were automatically derived from WordNet 3.0

在Path Similarity Tree一直往上走
直到找到BLC (?

  - **Gender Classifier** (**gender.classifier.py**)
    - Traing data: a list of names annotated with gender
    - **Input**: **name**
    - **Output**: **gender**
    - **Method**: Logistical Regression and Maximum Entropy

4.4.1 Lesk Algorithm – Disambiguating EVP entries to WordNet

Recall that LA relies on the number of shared words to determine the intended sense of a given context. It is not uncommon to have a **tie** with two or more senses having **the same number of shared words** with the context. In order to break the tie, people have used the **weight** such as **idf** (**inverse document frequency**) of word to **break a tie** and even to improve the performance. Therefore, in the training phrase (preprocessing), we read the dataset file and compute the following information:

- **Document frequency (df)**: the number of word sense categories where a certain word appears in. For instance, the word *money* appears in 80 categories including *get.v.*01. Let D be the total number of categories. Then $idf = df/D$.
- **Within document term frequency**: For each defining/example word, compute the WordNet sense categories the word appears in.

```
import re
from collections import defaultdict, Counter
from nltk.stem.wordnet import WordNetLemmatizer
lmtzr = WordNetLemmatizer()
import nltk, random
def words(text): return re.findall(r'\w+', text.lower())
TF, DF = defaultdict(lambda: defaultdict(lambda: 0)), defaultdict(lambda: [])
def wnTag(pos): return {'noun': 'n', 'verb': 'v', 'adjective': 'a', 'adverb': 'r'}[pos]

def trainLesk():
    training = [  line.strip().split('\t') for line in open('wn.in.evp.cat.txt', 'r') \
                                                    if line.strip() != '' ]
    def isHead(head, word, tag):
```

```
    for wnid, wncat, senseDef, target in training:
        head, pos = wnid.split('-')[:2]
        for word in words(senseDef):
            if word != head and not isHead(head, word, pos):
                TF[word][wncat] += 1
                DF[word] += [] if wncat in DF[word] else [wncat]

def testLesk():
    testdata = [  line.strip().split('\t') for line in open('evp.in.wn.cat.txt', 'r')
                                                    if line.strip() != '' ]

    def df(word): return len(DF[word])

    def leskOverlap(senseDef, target):
        wnidCount = [ (wncat, tf, word, len(DF[word])+1) for word in senseDef \
                                                for wncat, tf in TF[word].items() \
                                                if wncat in target.values() ]
        res = sorted( [ (wnid, tf*int(1000/df)) for wnid, tf, word, df in wnidCount],
                key = lambda x: -x[1])[:7]
        if not res: return 'Not found'
        counter = Counter()
        for wnid, tfidf in res: counter[wnid] += tfidf
        return counter.most_common(1)[0]

    for evpid, wncat, senseDef, target in testdata:
        print( '%s\t%s\t%s' %(evpid, leskOverlap(words(senseDef), eval(target)), \
                    senseDef.split('||')[0] ) )

if __name__ == '__main__':
    trainLesk()
    testLesk()
```

4.4.2 Statistical Classifer for Predicting Gender from Name

To provide a quick introduction to programming a statistical classifier, we use the classification of names into two classes, MALE and FEMAL as an example.

This example program reads in a name corpus available in NLTK (nltk.corpus.names). Each name instance (with gender information) of this corpus is **augmented with a set of features** (e.g., starting and ending letters) to generate two *featuresets*. The *featureset* (both training and test datasets) are then used to **train** and **evaluate** a classifier using classifier modules (LogisticRegression and MaxentClassifier) in NLTK.

```
from __future__ import division
import nltk, random

def gender_features(word):
    import string
    features = {char: char in word for char in word.lower()}
    features.update({'count({})'.format(char): word.count(char) for char in word.lower()})
    features.update({'startswith': word[0], 'endswith': word[-1]})
    return features

def LG_gender(train_set, test_set):
    print('== SkLearn MaxEnt ==')
    from nltk.classify import SklearnClassifier
    from sklearn.linear_model import LogisticRegression
    sklearn_classifier = SklearnClassifier(LogisticRegression(C=10e5)).train(train_set)
    print(sklearn_classifier.prob_classify(gender_features('mark'))._prob_dict)
    print(nltk.classify.accuracy(sklearn_classifier, test_set))

def ME_gender(train_set, test_set):
    print('== NLTK MaxEnt ==')
    from nltk.classify import MaxentClassifier
    nltk_classifier = MaxentClassifier.train(train_set,\
        nltk.classify.MaxentClassifier.ALGORITHMS[0])
    print(nltk_classifier.prob_classify(gender_features('mark'))._prob_dict)
    print(nltk.classify.accuracy(nltk_classifier, test_set))

if __name__ == '__main__':
    names_gender = ([(name.lower(), 'male')
        for name in nltk.corpus.names.words('male.txt')]
       + [(name.lower(), 'female') \
        for name in nltk.corpus.names.words('female.txt')])
    random.shuffle(names_with_gender)
    featuresets = [(gender_features(name), gender) for name, gender in names_gender]
    split_point = len(featuresets)*9//10
```

```
train_set, test_set = featuresets[:split_point], featuresets[split_point:]
LG_gender(train_set, test_set)
ME_gender(train_set, test_set)
```

4.4.3 Exercise

Write a WSD program to assign each EVP entry with a WordNet synset based on both simple.lesk.py and gender.classifier.py. Use *gender.classifier.py* to help you design your program.

References

1. Daniel Jurafsky and James H. Martin. 2017. Speech and Language Processing An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition, 3rd Edition. Chapter 17. Computing with Word Senses. (draft available at `https://web.stanford.edu/~jurafsky/slp3/`

2. Izquierdo, RubÃľn, Armando SuÃąrez, and German Rigau. "Word vs. class-based word sense disambiguation." Journal of Artificial Intelligence Research 54 (2015): 83-122. `http://adimen.si.ehu.es/~rigau/publications/jair15-isr.pdf`