Natural Language Processing

Assignment 3
Word Sense Disambiguation (WSD)
Using Vector Space Model (VSM)

Due November 22, 2015, at 11:59:59 pm

Introduction

In this assignment, we will:

1. Briefly introduce WSD and the Vector Model.
2. Go through the basics of WordNet and scikit-learn.
3. Develop a WSD system based on the Vector Model with supervised learning algorithms.
4. Use feature extraction methods to enhance your WSD system.

This document is structured in the following sequence:

WSD and Vector Model	Introduction to WSD and the Vector Space Model VSM.
Environment Setup	Set up everything to get you ready to start the assignment.
WordNet and scikit-learn	Quick intro to WordNet and scikit-learn Python libraries.
Assignment Part A: Basic WSD system	We will create, train, and evaluate a WSD system using the SENSEVAL corpus based on the vector space model VSM.
Assignment Part B: Feature Extraction	We will use feature extraction methods to enhance your WSD system.
Submission	Once you're done, follow the instructions to submit.

Introduction to WSD and VSM

What is word sense disambiguation (WSD)?

Word sense disambiguation (WSD) is the task of determining which sense of an ambiguous word is being used in a particular context. The solution to this problem impacts other NLP-related problems such as machine translation and document retrieval.

What is the lexical sample task?

The standard WSD (SENSEVAL) task has two variants: "lexical sample" and "all words." The former comprises disambiguating the occurrences of a small sample of target words which were previously selected, while in the latter all the words in a piece of running text need to be disambiguated. In this assignment, we will be working on the lexical sample task.

Algorithms for WSD:

There are several types of approaches to WSD, including dictionary-based methods, semi-supervised methods, supervised methods, and unsupervised methods. Supervised methods based on sense-labeled corpora are the best-performing methods for sense disambiguation. But such labeled data is expensive and limited. In contrast, dictionary-based methods and unsupervised methods require no labeled texts and are more efficient, but have lower accuracy. In the following part, you will see how two algorithms, a supervised algorithm based on the vector space model and the Lesk algorithm, work on WSD.

What is the vector space model (VSM)?

The vector space model (VSM) is a widely-used model in NLP. The basic idea is to represent each instance of an ambiguous word as a vector whose features (attributes) are extracted according to some rules. Usually classification or clustering methods are then applied to solve the problem.

The supervised approach to sense disambiguation can be based on the vector space model. Here is a skeleton supervised algorithm for the lexical sample task.

1. For each instance w_i of word w in a corpus, compute a context vector $\vec{c}$. Label the class of each vector by the sense of word w in the context.
2. Train a **classifier** with these labeled context vectors $\vec{c}$.
3. Disambiguate a particular token t of w by **predicting** the class of token t with the trained classifier.

Later in the assignment we will explain how to extract context vectors and what classification method you can use in order to implement this algorithm.

What is the Lesk algorithm?

The Lesk algorithm is a well-studied dictionary-based algorithm for word sense disambiguation. It is based on the hypothesis that words used together in text are related to each other and that the relation can be observed in the definitions of the words and their senses. Two (or more) words are disambiguated by finding the pair of dictionary senses with the greatest word overlap in their dictionary definitions.

The pseudo code of Lesk is shown below.

```
function SIMPLIFIED LESK(word, sentence) returns best sense of word
  best-sense ← most frequent sense for word
  max-overlap ← 0
  context ← set of words in sentence
  for each sense in senses of word do
    signature ← set of words in the gloss and examples of sense
    overlap ← COMPUTEOVERLAP(signature, context)
    if overlap > max-overlap then
      max-overlap ← overlap
      best-sense ← sense
    end
    return(best-sense)
```

You do not need to implement the Lesk algorithm in this assignment. But as an example you should get a sense of what an unsupervised algorithm is like.

For our lexical sample task, assume we have a Lesk Algorithm implementation in the following format:

lesk(context_sentence, ambiguous_word):
- param str context_sentence: The context sentence where the ambiguous word occurs.
- param str ambiguous_word: The ambiguous word that requires WSD.

- return: “lesk_sense” The Synset() object with the highest signature overlaps.

WordNet

WordNet is a lexical database (originally developed for English) that contains information about words, their senses, and the relationships between them. A word with multiple senses is called polysemous. A sense with multiple words is called a “synset” (synonym set). Synsets may be related to each other in different ways, e.g., “cat” is a hypernym (more general concept) of “lion.”

Environment Setup

You can copy the provided code and data by running the following commands:

```
cd ~/hidden/$PIN/
mkdir Homework3
cp /home/595/Homework3/student_files/* Homework3/
```

Basic WordNet and scikit-learn tutorial

How to use WordNet in NLTK for WSD tasks

- Import wordnet
>>> from nltk.corpus import wordnet as wn

- Get a word's synset
A synset is a set of words with the same sense. It is identified with a 3-part name of the form: word.pos.nn. Below are a description of the **synsets** function and examples of its usage.

def synsets(self, lemma, pos=None, lang='en'):
- Load all synsets with a given lemma (word) and part of speech tag.
- If no pos is specified, all synsets for all parts of speech will be loaded.
- If lang is specified, all the synsets in that language associated with the lemma will be returned. The default value of lang is 'en'.

>>> wn.synsets('dog')
[Synset('dog.n.01'), Synset('frump.n.01'), Synset('dog.n.03'), Synset('cad.n.01'), ...]
>>> wn.synsets('dog', pos='v', lang = 'en')
[Synset('chase.v.01')]

- **Get possible definitions for a word from its synsets**
>>> for ss in wn.synsets('bank'):
>>> print ss, ss.definition()
Synset('bank.n.01') sloping land (especially the slope beside a body of water)
Synset('depository_financial_institution.n.01') a financial institution that accepts deposits and channels the money into lending activities
Synset('bank.n.03') a long ridge or pile
Synset('bank.n.04') an arrangement of similar objects in a row or in tiers
…

- What languages can we use in NLTK?
The WordNet corpus reader gives access to the Open Multilingual WordNet, using ISO-639 language codes. For example, "cat" stands for Catalan (a language spoken in Spain, France and Andorra, which is related to Spanish, Italian, and the other Romance languages).
>>> sorted(wn.langs())
['als', 'arb', 'cat', 'cmn', 'dan', 'eng', 'eus', 'fas','fin', 'fra', 'fre', 'glg', 'heb', 'ind', 'ita', 'jpn', 'nno','nob', 'pol', 'por', 'spa', 'tha', 'zsm']

How to use scikit-learn for machine learning tasks

- **What is scikit-learn?**

scikit-learn (http://scikit-learn.org/stable/index.html) is an open source machine learning library for Python. It features various classification, regression, and clustering algorithms including support vector machines, k-nearest neighbors, logistic regression, naive Bayes, random forests, gradient boosting, k-means, and DBSCAN. For this class, we don't need to understand all the details of these algorithms.

- **Loading an example dataset**

scikit-learn comes with a few standard datasets. A dataset is a dictionary-like object that holds all the data and some metadata about the data. The data is stored in the .data member. In the case of supervised problems, one or more response variables are stored in the .target member. To load an example dataset,

```
>>> from sklearn import datasets
>>> iris = datasets.load_iris()
>>> digits = datasets.load_digits()
```

- **Learning and predicting**

Classification is a basic task in machine learning. The task is, given a set of data, usually represented as vectors, and several classes, predict which class each vector belongs to. To solve the problem, we will first train a classifier (estimator) using some data whose classes have already been given. Then we will predict the classes of unlabeled data with the classifier we just trained.

In scikit-learn, an estimator for classification is a Python object that implements the methods **fit(X,y)** and **predict(T)**. An example of an estimator is the class **sklearn.svm.SVC** that implements support vector classification.

```
>>> from sklearn import svm
>>> clf = svm.SVC(gamma=0.001, C=100.)
```

To train the model with all but the last item in our training dataset

```
>>> clf.fit(digits.data[:-1], digits.target[:-1])
```

Now you can predict new values by

```
>>> clf.predict(digits.data[-1])
```

Here is another example of how to use nearest neighbors algorithm to do the classification.

```
>>> from sklearn import neighbors
>>> clf = neighbors.KNeighborsClassifier(15, weights='uniform')
>>> clf.fit(iris.data, iris.target)
>>> clf.predict(iris.data)
```

Provided Files

Provided data

We will be using data from the Senseval 3 Lexical Sample Task on multilingual WSD, specifically the English, Catalan, and Spanish corpora.

Datasets:

All the data needed for this assignment are in this directory:
`/home/595/Homework3/data`

The data include:

`English-train.xml`	Training data for English
`English-dev.xml`	Development data for English
`English-dev.key`	Answer for the development data for English
`English.sensemap`	Sense map file needed for evaluation
`Catalan-train.xml`	Training data for Catalan
`Catalan-dev.xml`	Development data for Catalan
`Catalan-dev.key`	Answer for the development data for Catalan
`Spanish-train.xml`	Training data for Spanish
`Spanish-dev.xml`	Development data for Spanish
`Spanish-dev.key`	Answer for the development data for Spanish

The data files are in .xml form, where the tags are:

1. **lexelt/lemma**: Each lexelt/lemma represents a word to be disambiguated. The item attribute indicates the word and its POS tag. One lexelt could have several instances.

<lexelt item="difference.n">

2. **instance**: Each instance is a case where the target word appears in a context. An instance contains one context.

<instance id=" difference.n.bnc.00001061">

3. **context**: The context of the word from each instance. You can choose to use the entire context or just some words from the context (e.g., common collocations, words within a small window, words with high PMI values, etc).

<context>
In 1991/92 we shall need support even more . Every donation does help . That help makes all the <head>difference</head> to people sick with AIDS who want to stay at home , rather than spend time unnecessarily in hospital . Please help ! SIR JOHN FORD KCMG MG CHAIRMAN OF TRUSTEES
</context>

4. **answer**: The sense of the word in this instance. It only appears in training data.

```
<answer instance="difference.n.bnc.00001061" senseid="difference%1:24:00::"/>
```

Note: For Catalan and Spanish data, you will also see <previous>, <target>, <following> tags in a <context> tag. For this assignment we only use the text in the <target> tag.

Note2: If an instance contains multiple <answer> tags, we will only use the first tag for the purpose of training.

You can use the *xml* package in Python to parse the .xml file.

WSD systems are evaluated by computing the percentage of ambiguous words for which the WSD system has predicted the correct sense. We have provided an evaluator for you. You will use it to evaluate the performance of the algorithms you implement.

The .key files and .sensemap files are only used for evaluation. You do not need to know the meaning of these files. Later you will see how to use the evaluator with these files.

Provided code

```
baseline.py       a baseline WSD system implementation
scorer2           a script to evaluate WSD systems
scorer2.c         the source code for scorer2 written in C
A.py              skeleton code for part A
B.py              skeleton code for part B
main.py           runs the assignment
```

Before you start

You will be implementing a supervised algorithm based on the vector space model according to the description in the **Introduction** part. You will need to work on three languages: **English, Catalan, and Spanish**.

As a baseline, we explain the sense of a word in any context as the most frequent sense of this word. You can see the output of the baseline by running

```
python baseline.py language
```

The output file is `language.baseline`, with the same format in the format below:

```
lexelt_item instance_id sense_id
```

Your implementation should get a better performance than this baseline. You will be graded based on how good your implementation is.

Assignment Part A

For this part, you will be implementing a basic WSD system with two supervised algorithms based on the vector space mødel VSM using the Senseval data set.

Implementation

For each language (Catalan, English, and Spanish), you should follow the steps below.

1. Compute context vectors for each instance of a word. You will use `*Language*-train.xml` as training data.

   Suppose in a test instance T_i, the word you need to disambiguate is *w*. At the beginning you need to tokenize the sentences in the context with nltk.word_tokenize(). S_i is the set of words that are within *k* distance of word *w*. Let *S* be the union of set S_i of size *n* which may contain duplicate words. Then each test instance will correspond to a vector, where each attribute is the count of a word in set *S*. For this assignment, we set the window size *k=10*.

   Implement part A1 in the `build_s` and `vectorize` functions of `A.py`.

2. Train a classifier using the context vectors you obtained. You can use *scikit-learn* library to conduct this step. Here you need to try two different classifiers, k-nearest neighbors KNN (**neighbors.KNeighborsClassifier** class) and linear support vector machines SVM (**svm.LinearSVC** class).

   You will notice that there are several parameters for the classifiers. For this assignment you can just use the default settings (k=5 for KNN).

   Implement part A2 in the `classify` function of `A.py`.

You will disambiguate the target word in each test instance in the development data set. For each classifier, the output should be a single file corresponding to `*Language*-dev.xml`, with each line for a test instance in the format below:

```
lexelt_item instance_id sense_id
```

Note: The lines in the output file have to be sorted in alphabetical ascending order (A-Z) first by lexelt_item, then by instance_id. Also, please remove the accent of characters in the output file.

3. Use the SVM classifier to perform disambiguation for each test instance in `*Language*-dev.xml`. <u>Write the output for the SVM classifier for each language</u> to a file named by `SVM-*Language*.answer` formatted as described above.

4. Use the KNN classifier to perform disambiguation for each test instance in `*Language*-dev.xml`. <u>Write the output for the KNN classifier for each language</u> to a file named by `KNN-*Language*.answer` formatted as described above.

   Implement parts A3 and A4 in the `print_results` function of `A.py`.

5. In your report file, write and compare the performance of the two classifiers.

Assignment Part B

For this part, you will improve your WSD system. You will need to compare the performance of different features and classifiers, find the best combination, and describe your work in the report.

1. Try extracting better features than just taking all the words in the window, and then redo the classification. You should try the following approaches **(it is not necessarily that adding one of them will improve the system), also try to use different combinations**:

    a) Add collocational features such as: surrounding words w_{-2}, w_{-1}, w_0 w_1, w_2 and part-of-speech tags POS_{-2}, POS_{-1}, POS_0 POS_1, POS_2.

    b) Remove stop words, remove punctuations, do stemming, etc.

    c) Use the method described below.

    Suppose the word *w* that needs disambiguation has *k* senses. For a sense *s* of *w* and a word *c* in the window of *w*, we compute the "relevance score" of c with *s* using

    $$\log\left(\frac{p(s|c)}{p(\bar{s}|c)}\right)$$

    where $p(s|c)$ is the probability that the word *w* has sense *s* when *c* appears, and is computed using

    $$p(s|c) = \frac{N_{s,c}}{N_c}$$

    where $N_{s,c}$ is the number of instances where the context contains *c* and the sense of word *w* is *s*, and N_c is the number of instances where the context contains *c*. $p(\bar{s}|c)$ is the probability that the word *w* has sense other than *s* when *c* appears and can be computed similarly.

    For each sense *s*, we compute how relevant the words in the window are and select the top words as features. The final set of features is the union of all the top features for each sense.

    For example, in `English-train.xml`, the word **activate** has 3 senses, **38201**, **38202** and **38203**. We now compute the relevance score for the word **protein**. There are 10 test instances where the context contains **protein** and the sense is **38201**. There are 8 test instances for **38202** and 1 for **38203**. So the score for (**38201**, **protein**) is $\log\frac{10/19}{9/19}$. (**38202, protein)** gets $\log\frac{8/19}{11/19}$ and (**38203, protein)** gets $\log\frac{1/19}{18/19}$.

    d) Add more features by obtaining the synonyms, hyponyms and hypernyms of a word in WordNet; you should try different combination of these features rather than combining all of them. For instance, you might add hyponyms only, synonyms and hyperny, or combining all of these features.

e) Try good feature selection method. Hint: *Chi-square or pointwise mutual information (PMI)* may be useful for selection features.

Implement part B1a-d in the `extract_features` function of `B.py`. Implement part B1e in the `feature_selection` function of `B.py`.

In your report, describe how much each of the approaches above improves the performance.

2. Find the best combination of the feature extracting approaches. Also, use the classifier that gives better results. Submit your code with the best performance and the output files from that code; name those output files `Best-`*`Language`*`.answer`.
Implement part B2 in the `classify` function of `B.py`.

3. You are welcome to add more features and to adjust the window size *k* and the classifier parameters to get better results. If you use your own parameters or add different features than above, include in your report how you obtained the settings and what features you added. You can also try other classifiers if you want. We will compare the performance of all the students' classifiers, and give **bonus points** for the top performers.

Conclusions

Finally, write the conclusions you draw from this assignment. Also, include any interesting observations.
Discuss why are some languages are easier than other languages for WSD tasks (Extra points).

Evaluation

You will evaluate the result you obtained from the previous parts. Use scorer2 as shown below to evaluate your output.

```
./scorer2 answer_file key_file [sense_map_file]
```

where `answer_file` is the output of your algorithm and `key_file` is the gold standard for the test data.
Here is an example:

```
./scorer2 English.answer /home/595/Homework3/data/English-dev.key
          /home/595/Homework3/data/English.sensemap
```

Note: For Catalan and Spanish data, there is no sensemap file. So you do not need to include it in the command.

Note2: scorer2 is written in C, and we have given you a compiled binary. We have included the source code here in case you want to try it on other platforms.

Include the evaluation results in your report. Include any interesting findings and explain why they occurred (up to 2-3 paragraphs and a table with results).

Submission

As usual, you need to create a directory named `Homework3` in your hidden directory and copy all the files into it. The files you should submit are:

```
main.py
A.py
B.py
SVM-English.answer
SVM-Spanish.answer
SVM-Catalan.answer
KNN-English.answer
KNN-Spanish.answer
KNN-Catalan.answer
Best-English.answer
Best-Spanish.answer
Best-Catalan.answer
report.txt
```

As a final step, run a script that will setup the permissions to your homework files, so we can access and run your code to grade it:

```
/home/Homework3/hw3_set_permissions.sh <YOUR_PIN>
```

Note: You can create extra source code files, but make sure we can run your algorithm simply by running:

```
python main.py <input_training file> <input test file> <output KNN file> <output SVM file> <output best file> <language>
```

For instance:

```
python main.py /home/595/Homework3/data/English-train.xml /home/595/Homework3/data/English-dev.xml KNN-English.answer SVM-English.answer Best-English.answer English
```