Word Sense Disambiguation

CS 4740 - Introduction to NLP

Fall 2015

Proposal hardcopy to be submitted in class and CMS on 10/8
Project due via CMS on 10/19 @11:59pm

1 Introduction

Word Sense Disambiguation (WSD) is a task to find the correct meaning of a word given context. Because many words in natural language are polysemous, humans perform WSD based on various cues from the context including both verbal and non-verbal. In this assignment, you are going to implement a WSD system by using one of two different methods: a supervised method or a dictionary-based method. Since many problems in natural language understanding are related to knowing the sense of each word in a document, WSD can be utilized as a building block for a variety of high-level tasks in the future.

Through this assignment, you will use the English Lexical Sample task from Senseval for training and testing your system. Training and evaluating on these data, you will analyze the pros and cons of your selected WSD approach. We will also setup a Kaggle competition so that you can submit the prediction results of your WSD system.

2 Dataset

The data files for this project are available via CMS and consist of training data, test data, and a dictionary that describes commonly used senses for each word. All these files are in XML format. Every lexical item in the dictionary contains multiple sense items, and the examples in the training data are annotated with the correct sense of the target word in the given context.

The file training-data.data contains several $< lexelt >$ elements corresponding to each word in the training set. Each $< lexelt >$ element has an item attribute whose value is word.pos, where *word* is the target word for which we are to predict the sense and *pos* represents the part-of-speech of the target word where 'n', 'v', and 'a' stand for noun, verb, and adjective, respectively.

Each $< lexelt >$ element has several $< instance >$ elements, each corresponds to a training instance for the *word* that corresponds to the parent

$<$ *lexelt* $>$ element. Each $<$ *lexelt* $>$ element also has an *id* attribute and contains one or more $<$ *answer* $>$ and a $<$ *context* $>$ element.

Every $<$ *answer* $>$ element has two attributes, *instance* and *senseid*. The *senseid* attribute identifies the correct sense from the dictionary for the present *word* in the current *context*. A special value "U" is used to indicate that the correct sense is not clear.

A $<$ *context* $>$ element contains:
prev-context $<$ *head* $>$ target-word $<$ /*head* $>$ next-context

- prev-context is the actual text given before the target word for which we are predicting the sense.
- head is the actual appearance of the target word to be disambiguated. Note that it may be morphological variant of the target word. For example, the word "begin.v" could show up as "beginning" instead of "begin".
- next-context is the actual text that follows the target word.

The structure of the test-data.data file is the same as training-data.data except that it does not have $<$ *answer* $>$ elements inside $<$ *instance* $>$ elements. Instead, your system will predict the answer senses.

In the dictionary file, every sense item contains a gloss field to indicate the corresponding definition. Each gloss consists of commonly used definitions (It is possible to explain a certain sense in multiple ways inside a single sense item entry) delimited by a semicolon, and may have multiple real examples wrapped by quotation marks being also delimited by a semicolon. The format of the gloss field is not highly uniform, so, you may find some glosses that do not have real examples.

3 Supervised WSD

This section will show a simple probabilistic approach called the Naive-Bayes model to perform supervised WSD. It is also explained in the text book. This model takes a word in context as an input and outputs a probability distribution over predefined senses, indicating how likely each sense corresponds to the correct meaning of the target word within the given context. Specifically, it picks the best sense by the following:

$$\hat{s} = argmax_{s \in S(w)} P(s|\vec{f})$$

In the above equation, S(w) is the predefined set of senses for the target word w, and $\vec{f}$ is a feature vector extracted from the context surrounding w. Thus the equation says we are going to choose the most probable sense as the correct meaning of the target word w given the feature vector $\vec{f}$. By applying Bayes rule,

$$P(s|\vec{f}) = \frac{P(\vec{f}|s)P(s)}{P(\vec{f})}$$

As the denominator does not change with respect to s ∈ S(w), the best sense $\vec{s}$ is determined by

$$\vec{s} = argmax_{s \in S(w)} P(\vec{f}|s)P(s)$$

The model then naively assumes that features in the feature vector $\vec{f}$ are conditionally independent given the sense of the word s. This assumption yields the following decomposition:

$$P(\vec{f}|s) = \prod_{j=1}^{n} P(f_j|s) \quad \text{where} \quad f = (f_1, f_2, ..., f_n)$$

In other words, the probability of a feature vector given a particular sense can be es- timated by the product of the probabilities of its individual features given that sense. Hence the best sense is determined by

$$\vec{s} = argmax_{s \in S(w)} P(\vec{f}|s)P(s) = argmax_{s \in S(w)} P(s) \prod_{j=1}^{n} P(f_j|s)$$

What you have to implement for this model is given below. Read the following instructions carefully.

1. In order to train the above model, you should learn the model parameters: 1) the prior probability of each sense P(s) and 2) the individual feature probabilities $P(f_j|s)$. Those are computed by the Maximum Likelihood Estimation (MLE) which simply counts the number of occurrences in the training set. In particular for the i-th sense s_i of a word w,

   $$P(s_i) = \frac{count(s_i, w)}{count(w)} \quad P(f_j|s_i) = \frac{count(f_j, s_i)}{count(s_i)}$$

   For instance, let's assume there are 1,000 training examples corresponding to the word "bank". Among them, 750 occurrences stand for $bank_1$ which covers the financial sense, and 250 occurrences for $bank_2$ which covers the river sense. Then the prior probabilities are

   $$P(s_1) = \frac{750}{1000} = 0.75 \quad P(s_2) = \frac{250}{1000} = 0.25$$

   If the first feature "credit" occurs 195 times within the context of $bank_1$, but only 5 times within the context of $bank_2$,

   $$P(f_1 = "credit"|s_1) = \frac{195}{750} = 0.26 \quad P(f_1 = "credit"|s_2) = \frac{5}{250} = 0.02$$

2. Though the basic setup is given above, **the performance of your WSD system will mainly rely on how well you generate feature vectors from the context**. Note that target words to be disambiguated are always pro- vided within a sufficiently long sentence. As you have seen in the above toy example, extracting informative features from the surrounding context will have the model parameters discriminate the unlikely senses from the correct sense. In our model, this process of deciding model parameters corresponds to the training. Of course, **you have to train a separate model per each target word in the training data**.

3. When learning your model, be sure that you never use the test data for training. Instead of marking the predicted senses directly into the testing file, **you are going to generate a separate output file consisting of the predicted senses for test data**. If you upload it to Kaggle, you will get back your score until the submission deadline. You can find the output file specification in Section 5.

4. If you want to evaluate the performance of your system before submitting to Kaggle, or you designed multiple different models based on the Naive-Bayes model, you should reserve a certain portion of the training data as a validation set in order to measure the performance of your system (or models). Since you know the true senses for the training data, testing on the validation set will let you guess how well your system (or models) works for the test data. This process is called validation. Note that you must not train on the validation set if you use a validation step.

4 Dictionary-based WSD

This section will show a dictionary-based WSD approach which is different from the previous supervised setting. This technique was discussed in class and in the text book. Rather than training on the human-tagged dataset of true senses, dictionary-based approaches utilize definitions given in the dictionary. See the following example of disambiguating "pine cone".

- pine (the context)
    1. a kind of **evergreen tree** with needle-shaped leaves
    2. to waste away through sorrow or illness
- cone (the target word)
    1. A solid body which narrows to a point
    2. Something of this shape, whether solid or hollow
    3. Fruit of certain **evergreen trees**

As bold faced in the above, the 3rd sense of the target word best matches the 1st sense of the context word among all possible combinations. This process shows the original Lesk algorithm to disambiguate senses based only on the cross-comparing the definitions. However, it is able to be extended to utilize examples in the dictionary to get wider matching. **Read the following instructions carefully**.

1. Design a metric that rewards consecutive overlaps beyond treating them as two distinct overlaps of a single word. Note that there would be morphological varia- tions in the definitions and examples. In order to increase the matching, stemming or lemmatizing could be useful. (You can find such tools in the WordNet).

2. Implement a dictionary-based WSD system that disambiguates the sense by comparing the definitions of the target word to the definitions of relevant words in the context. In the same way that the performance of the

supervised system depends largely on the feature generation, here your design decision of finding relevant words will determine the performance of the dictionary- based system in combination with the metric you designed above.

3. In contrast to the supervised settings, there is no training process involved here because we mainly use definitions and examples in the dictionary to figure out the correct sense. As explained in the Section 2, the dictionary file contains a certain amount of glosses and examples. If you conclude those are not enough to perform the dictionary-based approach, using the WordNet dictionary could be an alternative. If it is hard to find the mapping between predefined senses of our dictionary and WordNet, you may first find the correct sense in WordNet, and then may choose the best sense in our dictionary based on the WordNet sense with its examples and definitions. Though you may have to perform the matching algorithm twice, incorporating WordNet may be helpful due to its richness and APIs.

4. Since there is no training process, you could verify the performance of your dictionary-based system on the entire training set. You could also compare the performance of two WSD systems via testing on the same validation set you reserved from the training set. Similar to supervised WSD, you have to submit prediction results for the test data to Kaggle.**Your final Kaggle submission can be the one with higher accuracy.**

5 Scoring

We use accuracy (= No. of correct predictions / No. of total predictions) as a score. Since a target word may contain multiple meanings at the same time, a single prediction could be counted as incorrect unless your system predicts exactly same senses with the ground-truth labels. Suppose that the target word in a test example has k senses in the given context. Then you should predict all k senses in your output file. The prediction file you will have to submit to Kaggle consists of a single line per test instance and it should always begin with the following line.

Id,Prediction

This line should be followed by several lines following the format below and there should be one line for each test example.

$< instanceidvalue >, < Senseid1 >< Senseid2 > ... < Senseidk >$

An example line in the correct format is shown below:

activate.v.bnc.00134704,38201 38203

6 Proposal

Describe your WSD system and implementation plan in 1 page: what kinds of features are you planning to extract from the context if you elect to focus on supervised WSD? Otherwise, What are you going to do for dictionary-based

WSD? Provide a clear and brief explanation of your planned system and algorithm. (You don't have to repeat the basic WSD models that described in this document) Rather than writing up every single detail, try to explain the motivation of your design decisions by providing the intuition via examples. Note that the more examples you brainstorm, the higher accuracy you will likely get in this assignment.

7 Report

You should submit a short document (5-6 pages will suffice) that consists of the following sections. (Of course, you can add more sections if you wish!)

1. **Approach:** Explain the approach for your WSD system. Try to justify your design decisions by providing intuitive examples.

2. **Software:** List any software that your system uses, but you did not write by yourself. Make clear which parts of system rely on the software if it is not obvious.

3. **Results:** Explain what kinds of experiments you perform for this assignment. Summarize the performance of your system on the test data. Minimally, you could compare your results to the baseline system that always predicts to the most frequent sense. Note that having no comparisons will not justify the experimental performance of your system. Please put the results into clearly labeled tables and diagrams including a written summary of the results.

4. **Discussion:** You should include observations that you make during the experiments. One essential discussion for the machine learning based system is to analyze why and which features are informative based on the real examples. Please report the top three features with the selected real examples. Which aspect(s) of your dictionary based approach are most important for good performance? Can you characterize the kinds of examples for which your system is particularly good(or bad)?

8 Guidelines

1. You should work in groups of 2 for this project.Working with people from various backgrounds will highly benefit both implementation and analysis for this assignment.

2. It is not allowed to use other pre-built WSD systems to implement or fortify your own system. In contrast, using tool-kits such as NLTK or OpenNLP is encouraged. Note that you should not use machine learning algorithms such as SVMs for this assignment unless they are used in addition to Naive Bayes.

3. Start early and try to discuss the training and dictionary data together before writing a proposal. Reserve enough time to provide an analysis of your results.

4. **Submitting to Kaggle is not optional**, but mandatory for every team. Detailed information about Kaggle competition will be updated via Piazza. Extra credit will be provided for top teams for each approach on Kaggle. Details about extra credit are explained in the grading guidelines.

5. Grading

   - Proposal: [10 pts]
   - Implementation: [45 points]
   - Report: [50 pts]
   - Kaggle Submission[5 pts]
   - Extra Credit by Kaggle Rank: #1 - 10 pts, #2 - 5pts, #3 - 3pts, #4&5 - 2 pts, #6-10 - 1pt

6. What to submit

   - Proposal (pdf into CMS, hardcopy at class)
   - Source code (only the code that YOUR TEAM wrote, not for any of the common packages that your team used).
   - Prediction output (the files you submit to Kaggle)
   - The report (pdf into CMS, hardcopy at class)
   - Archive all of these except the proposal, and upload it to CMS.

7. When to submit

   - Proposal due on Thursday 10/08. Submit hard copy in class and upload soft copy to CMS
   - Final project due on Monday 10/19 @ 11:59 pm. Upload project to CMS in a .zip file