IDS566: Advanced Text Analytics *Updated: 09/30/22*

Mini-Project #2

Instructor: Moontae Lee *Total Points: 200*

Policy

1. You are allowed to work as a group of up to three or four students.
2. Having wider discussions is not prohibited. Put all the names of students that you discuss with beyond your group members. However individual groups must write their own solutions.
3. Put your write-up and results from the coding questions in a single pdf file. Compress Python source codes into one zip file. Each student/group must submit only two files. (Your solution will not be graded if the answers for coding questions are not included the pdf report)
4. If you would include some graphs, be sure to include the source codes together that were used to generate those figures. Every result must be reproducible.
5. Maximally leverage Piazza to benefit other students by your questions and answers. Try to be updated by checking notifications in both Piazza and the class webpage.
6. No assignment will be accepted after its due date. The final report will be due by 10/17/2022 11:59pm

Problem 1: Word-Sense Disambiguation [100 points]

Word Sense Disambiguation (WSD) is a task to find the correct meaning of a word given context, which can be a building block for various high-level NLP tasks. As many words in languages have more than a single meaning, humans perform WSD with respect to various verbal and non-verbal signals. In this problem, you are going to implement a WSD system by using two different models: *ontological* model and *supervised* model. To start, read the English Lexical Sample Task written by Mihalcea, Chklovski and Kilgarriff in the following link.[1]

The data files are lightly preprocessed for the class project. They consist of training, validation, and test data provided with a XML formatted dictionary that describes commonly used senses for each word. Every lexical element in the dictionary contains multiple sense items, assigning one integer id per each sense. Briefly see the following example from our XML dictionary

[1] `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.153.8426`

```
<lexelt item="future.n" num="4">
<sense id="1" wordnet="1" gloss="time to come" examples="In the future we will drive flying cars.
| What will you do in the future." />
<sense id="2" wordnet="2" gloss="verb tense" examples="The paper was written in future tense." />
<sense id="3" wordnet="3" gloss="commodities" examples="He made his living trading in futures." />
<sense id="4" wordnet="" gloss="personal time to come" examples="My future is bright. | What will
you do with your future?" />
</lexelt>
```

It describes one lexical element: *future* (part-of-speech is noun) with its four different senses. Each sense has its own *gloss* (definition) and *examples* that are separated by | symbol. Each sense is also associated with the corresponding senses of WordNet 2.1. As our sense divisions and WordNet's are not identical, some of our senses could be mapped to multiple WordNet senses or possibly nothing (e.g., See the fourth sense item in the above example). Since the current NLTK is using WordNet 3, the sense mapping from 2.1 to 3.x could be useful.[2]

The training data specifies the correct sense of the target word providing its verbal context surrounding the target word. Each line of training data has the the following format. Note that sense-ids in the test data are all erased to 0 as those are what you should predict.

`word.pos | sense-id | prev-context %% target %% next-context`

- `word` is the original form of the target word for which we are to predict the sense. You will use it to lookup the XML dictionary.
- `pos` is the POS where 'n', 'v', and 'a' stand for noun, verb, and adjective, respectively.
- `sense-id` is the integer number for the correct sense id defined in our dictionary.
- `prev-context` is the text given earlier than each of the target word occurrence.
- `target` is the actual occurrence of the target word. Note that the `word` "begin.v" could occur as "beginning" instead of "begin" to denote a participle at the given position.
- `next-context` is the text given later than each of the target word occurrence.

Now we describe a simple probabilistic approach called the Naive-Bayes model. The model takes a word in context as an input and outputs a probability distribution over predefined senses, indicating how likely each sense would be the correct meaning of the target word within the given context. Specifically, it picks the best sense by the following equation:

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

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 that we are going to choose the most probable sense as the correct meaning of w. By Bayes rule,

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

[2] `https://stackoverflow.com/questions/46950379/how-to-fetch-a-specific-version-of-wordnet-when-doi`

As the denominator does not change with respect to $s \in S(w)$, the best sense $\hat{s}$ is given by

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

Here the model *naively* assumes[3] that each feature in the feature vector $\vec{f}$ is conditionally independent given the sense of the word s. The assumption allows us to evaluate $p(\vec{f}|s)$ by

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

In other words, the probability of a feature vector given sense can be estimated by the product of the probability of its individual features given that sense under our assumption. Hence,

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

What you have to implement for this model is given in the following instructions.

1. 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 purely counts the number of actual occurrences in the training set. Particularly for the i-th sense s_i of a word w,

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

   For instance, 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

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

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

$$P(f_1 = \text{“}credit\text{”}|s_1) = \frac{195}{750} = 0.26 \qquad P(f_1 = \text{“}credit\text{”}|s_2) = \frac{5}{250} = 0.02$$

2. The performance of your WSD system would rely more on how to generate feature vectors from the context. Note that target words are always provided within sufficiently long sentence(s). As the above example shows, extracting informative words from surrounding context allows the model parameters to discriminate unlikely senses from the correct sense. In our model, this process of deciding model parameters becomes *training*. **You have to train a separate model per each target word in the training data**.

[3]This is why the model is called Naive-Bayes.

3. When initially training your model, make sure that you never use the validation/test data. Note that the correct sense-ids given in the test data are deliberately erased to 0, which means those are no longer true labels. Instead of marking the predicted senses directly on the testing file, you must generate a separate output file consisting only of the predicted sense-ids, one id per line in each test data.

4. To achieve quality performance, smoothing is necessary. Implement either add-1 or add-λ smoothing. If you want to compare the performance of multiple models (e.g., different λ's), feel free to use a validation set, which is randomly reserved from the original test data. Since the true senses are alive in the validation data, testing on the validation set will let you guess the true performance on the unseen data. Note that you must not train on the validation set if you want to validate the performance. However, you can add the validation set to the training data for predicting the best senses of the test data.

Problem 2: Extension [100 points]

We have a list of topics for further extension. First, **dictionary-based WSD** utilizes definitions given in the dictionary. See the following example that tries to disambiguate *"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, 3rd sense of the target word matches the most with 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, rich examples given in the dictionary can be utilized to extend this model for better matching.

1. Design a metric that more rewards consecutive overlaps. For example, the 1st sense of *pine* and the 3rd sense of *cone* share two consecutive words in their definitions, which must earn higher score than the case when the two shared words are not consecutive in each definition. Note that there would be morphological variations in the definitions and examples. To increase the matching, stemming or lemmatizing could be useful.[4]

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. Your design decision of choosing relevant words will determine the performance of the dictionary-based system in combining with the metric you designed above.

[4]You can find relevant tools: stemmer and lemmatizer in NLTK and WordNet.

3. Because we mainly use glosses and examples in the dictionary to figure out the correct senses, no training is necessary for the Simple Lesk WSD. If you want to try the Corpus Lesk WSD (extension), try to augment the dictionary by the trainig data. If you think that those are not enough to achieve competitive accuracy, feel free to use the WordNet dictionary to further improve the performance.[5]

4. If no training process is involved, you could verify the performance of your Simple Lesk WSD system on the entire training set. If you want to compare the performance of various WSD systems like your Corpus Lesk or supervised WSD in the next section, test on the same validation set that we provide. You should also submit prediction results on the test data for every model that you would try.

We have the following **additional extensions** on top of the default option of doing Simple Lesk algorithm described earlier.

1. (10 pts) Implement the Corpus Lesk algorithm by augmenting the dictionary with the training data. Report your improved performance against the Simple Lesk algorithm.

2. (10 pts) Instead of hard-comparing to a single correct sense, you could design soft-scoring scheme that partially votes to each sense with respect to its confidence based on your model. For the supervised WSD using the Naive-Bayes model, it is easy to vote partially because the system guesses the correctness of each sense as a probability distribution, whereas you may have to do some normalization for soft-scoring in the dictionary-based method.[6] Note that this scoring is an expected score: for example, if your best answers are *sense-1* with 70% confidence and *sense-2* with 30% confidence, you gain only 0.7 (rather than 1.0) if *sense-1* is a right answer, whereas you gain only 0.3 (rather than 1.0) if sense-2 is a right answer. Evaluate the prediction result **on the validation set** and compute the average accuracy. Analyze the difference between the two scoring schemes and discuss which one seems more beneficial with supporting reasons.

3. (20 pts) Use embeddings via Spacy package in Python. Install Spacy and download the pre-trained embedding models by "`python -m spacy download en_core_web_md`". Then you can retrieve individual word-vectors given a sentence or its sentence-vector in terms of the average of the word vectors. Improve your ontological WSD and supervised WSD by incorporating these word-vector information. Feel free to use the following script for this open-ended extension.

```
import spacy
# Load the spacy model that you have installed.
model = spacy.load('en_core_web_md')
# Process a sentence given the pre-trained model.
embeddings = model("You are working on the second homework.")
# Extract a word-vector for the 7-th word homework.
embeddings[6].vector
# Get a sentence-vector as a mean of the individual word vectors.
embeddings.vector
```

[5] If you would use WordNet, be careful in the version difference as stated in the introduction.
[6] Recall 1-(d) that explains how to normalize the score, getting a probability distribution.

What to submit?

We use *accuracy*[7] as a score. Since each of possible senses is already specified by a dfferent sense-id, and no examples has multiple senses at the same time, a single prediction will be counted as incorrect one unless the prediction is equivalent to the ground-truth sense tag.

Assuming the given word in a test example has k different senses based on our XML dictionary, the prediction file must consist of a $1-k$ integer number per line. Concretely, if the test set consists of three examples where each example has 7, 3, and 5 different senses, your system should output one line for each of three test examples like the following.

```
7
1
4
```

Minimally you should implement the Simple Lesk algorithm for the ontological WSD and the Naive-Bayes algorithm with add-1 smoothing for the supervised WSD. After experiments, write a short PDF report (max 4 pages) that consists of the followings in addition to **your codes and predictions on the test data**. Add accordingly if you do some extensions. (List any software that you did not write by yourself. Note that using any pre-built WSD is not allowed)

(a) Explain all WSD systems that you have built. Ideally two systems: the Simple Lesk and the Corpus Lesk for ontological WSD. Another two systems: add-1 and add-λ smoothing Naive-Bayes for supervised WSD.

(b) Try various scoring functions and different feature engineering. Pick the best one for each WSD system, providing several intuitive real examples chosen from the training data that can justify your design decisions.

(c) Report the comparative performance among your ontological WSD systems with table/-graphs by testing on the validation set. Report the comparative performance similarly among your supervised WSD systems. Note that there must be a baseline WSD system that always predicts to the most frequent sense. No comparisons with the baseline cannot justify performance of your systems. Finally compare the entire WSD systems, reporting clearly labeled tables/graphs with a written summary of the results.

(d) Include observations that you achieve during the experiment. One essential discussion is to analyze informative features based on the real examples. In addition, Discuss the difference between the supervised and the dictionary-based WSD systems. Which system is more appropriate for which cases based on the real examples chosen from the data.

(e) Report your additional findings if you decide to implement some of the extensions.

[7] Accuracy = # of correct predictions / # of total predictions