Natural Language Processing

Team 40

Word Sense Disambiguation

March 2023

Interim Submission

Summary

1 Topic Understanding

1.1 Features Space

Features are helpful in WSD supervised algorithms because they provide information about the context in which the ambiguous word appears. By analyzing the surrounding words, part-of-speech tags, and other linguistic features, the algorithm can infer the most likely sense of the ambiguous word.

For example, consider the word "bank" in the sentence "I need to deposit my paycheck in the bank." Without any additional context, it is unclear whether "bank" refers to a financial institution or the edge of a river. However, if we look at the surrounding words, such as "deposit" and "paycheck," we can infer that "bank" most likely refers to a financial institution.

By incorporating features such as these into the algorithm's feature vector, we can improve its accuracy at predicting the correct sense of the ambiguous word in various contexts.

For any supervised algorithm in Word Sense Disambiguation (WSD), the following features can be helpful:

- **Word form:** The form of the target word can be an important feature, as different word forms can have different senses.
- **Part-of-Speech (POS) tag:** The POS tag of the target word and its neighboring words can help identify the sense of the target word.
- **Context words:** The words surrounding the target word can provide important information for identifying the sense of the target word.
- **Collocations:** The co-occurrence of certain words with the target word can also be an important feature.
- **Semantic features:** The semantic features of the target word, such as its hypernyms, hyponyms, synonyms, and antonyms, can provide useful information for identifying its sense.

- **Syntactic features:** The syntactic features of the target word and its neighboring words, such as their dependency relationships, can provide additional information for identifying the sense of the target word.
- **Word frequency:** The frequency of the target word and its neighboring words in a corpus can be used as a feature.
- **Domain-specific knowledge:** Domain-specific knowledge, such as knowledge about the topic or domain of the text, can provide additional information for identifying the sense of the target word.

1.2 Algorithms

1.2.1 Naive Bayes Classifier

Naive Bayes classifier is a probabilistic algorithm that can be used for Word Sense Disambiguation (WSD) by predicting the sense of a word given its context. Here is the logical implementation of Naive Bayes Classifier for WSD:

- Preprocess the training data: Extract the context and the sense label for each instance in the training set.
- Compute the prior probability of each sense label: Calculate the probability of each sense label in the training set, i.e., $P(senseLabel)$.
- Compute the likelihood of each feature given each sense label: For each feature in the training set, calculate the probability of observing that feature given a particular sense label, i.e., $P(feature|senseLabel)$.
- Compute the posterior probability of each sense label given the observed features: Using Bayes' theorem, compute the posterior probability of each sense label given the observed features, i.e., $P(senseLabel|features) = P(features|senseLabel)*P(senseLabel)/P(features)$.
- Choose the sense label with the highest posterior probability: Predict the sense label with the highest posterior probability as the correct sense of the target word.

Here is the mathematical implementation of Naive Bayes Classifier for WSD: Let x be a context and y be a sense label. Then, the goal is to compute the posterior probability P(y—x) of each sense label given the observed context.

Using Bayes' theorem, we have:

$$P(y|x) = \frac{P(x|y)P(y)}{P(x)}$$

where $P(x|y)$ is the likelihood of observing context x given the sense label y, P(y) is the prior probability of the sense label y, and P(x) is the marginal probability of observing context x.

Using the naive assumption that the features are conditionally independent given the sense label, we can write:

$$P(x|y) = P(x_1|y)P(x_2|y)...P(x_n|y)$$

where $x_1, x_2, ..., x_n$ are the individual features in the context x.

Assuming a multinomial distribution for the features, we can estimate the likelihood as:

$$P(x_i|y) = \frac{(count(x_i, y) + alpha)}{(count(y) + alpha|V|)}$$

where count(xi, y) is the count of feature xi in instances with sense label y, count(y) is the count of instances with sense label y, alpha is a smoothing parameter, and $|V|$ is the size of the vocabulary.

The prior probability can be estimated as:

$$P(y) = \frac{count(y)}{N}$$

where N is the total number of instances in the training set.

The marginal probability can be computed using the law of total probability:

$$P(x) = sum(P(x|y)P(y))$$

Finally, the sense label with the highest posterior probability can be chosen as the correct sense of the target word.

Basic Preprocessing : The clean function is a preprocessing function that takes in a list of tokens (words) and performs several operations to clean and transform them for further analysis. Here is a brief explanation of each operation:

- Part-of-speech (POS) tagging: The function applies POS tagging to each token using the pos-tag function of nltk library, which assigns a tag to each token indicating its grammatical category (e.g., noun, verb, adjective).
- Lemmatization: The function then lemmatizes each token using the WordNet lemmatizer (WordNetLemmatizer), which reduces each token to its base form (lemma). It also passes the POS tag from step 1 to the lemmatizer to improve accuracy.
- Stopword removal: The function removes stopwords from the tokens using a predefined list of stopwords from the NLTK library (nltk.corpus.stopwords). It also adds some additional punctuation marks to the stopword list.
- Number-to-word conversion: The function converts any numeric tokens to their corresponding word representations using the num2words function from the num2words library. This is done to avoid treating numbers as distinct vocabulary items.
- Output: The function returns a list of cleaned tokens, where each token is represented as a tuple of the form (word, tag), where word is the cleaned word and tag is its POS tag.

Implementation: Using the idea suggested in papers , First implementation is done using the local context of every word in the sentence and thus we use only one feature vector for Naive Bayes classification. The model do not perform very well in this case. The second idea is to use POS tag of every word in its surroundings (local context) . The accuracy found for this model is also very similar to previous means POS tags do not work very well in this case. Following idea from the paper "High WSD accuracy using Naive Bayesian classifier with rich features " to choose different features and finally choose the best set of features which make the model best. The features presented by the paper are:

- F1 is a set of unordered words in the large context, F1= $\{\ldots, w_{-2}, w_{-1}, w_1, w_2, ...\}$
- F2 is a set of words assigned with their positions in the local context, F2 = $\{..., (w_{-2}, -2), (w_{-1}, -1), (w_1, 1)$
- F3 is a set of part-of-speech tags assigned with their positions in the local context, $\{..., (p_{-2}, -2), (p_{-1}, -1), (p_1, 1), (p_2, 2), ...\}$

- F4 is a set of collocations of words, F4 = $\{..., w_{-1}w, w_{-2}w_{-1}w, ww_1, ww_1w_2, ...\}$
- F5 is a set of collocations of part-of-speech tags, F5 = $\{..., p_{-1}w, p_{-2}p_{-1}w, wp_1, wp_1p_2, ...$ $\}$

Using this idea we are trying to find the best set of features . The paper has shown the F1,F2 and F4 and we are trying to get these results for Semcor dataset. This is one of next work we are thinking.

We are trying one more baseline implementation using Naive Bayes only with the help of BERT Pretrained Model. We will complete two supervised algorithms Naive Bayes and SVM and will start working on Neural Models of Word Sense Disambiguations.

1.2.2 Support Vector Machine

The SVM approach helps us to find a hyperplane which separates the training examples into two classes. A test sample is classified based on which side of hyperplane it is present. In order to counter cases where the data is not linearly separable, we use a kernel function which maps the input features to a higher dimension in a way that it becomes linearly separable. We will also use a regularization parameter, in order to allow error up to a certain extent. This allows us to reduce over-fitting in certain cases. Note that SVM is a binary classifier. Hence, we construct a binary classifier for each sense class (output 1 if it belongs to the class else 0). For a given test sample, if multiple SVM's output positive result, we pick the one with the best SVM score among these. The SVM score could simply be the distance from the hyperplane.
In order to disambiguate a word w, the paper suggests to extract four features, viz Parts of Speech (POS) of neighboring words, single words in surrounding context , local collocations and syntactic relations. Apart from this, it was observed that some additional info on the language source helps us get an improved accuracy. We shall mainly be focusing on the first four features.

Parts-of-Speech (POS) of Neighboring Words:
We use 7 features, viz P_{-3}, P_{-2}, P_{-1}, P_0, P_1, P_2, P_3, where P_i denotes the POS of the i^{th} token relative to w. A token may be a word or a punctuation symbol, an d each of the these neighboring tokens must be in the same sentence as w.

Single Words in Surrounding Context: To extract this feature, we consider all unigrams in surrounding context of the token w, and these tokens may be in different sentence from w.

We convert all tokens in the surrounding context to lowercase and replace them to by their morphological form. Stop-words and tokens which contain no alphabetical characters such as numbers and punctuations are removed. All remaining tokens from all training contexts provided for token w are gathered. Each remaining token t contributes one feature. In a training (or test) example, the feature corresponding to & is set to 1 iff the context of in that training (or test) example contains it.

As an example, if w is the word "bars", and the set of selected unigrams is {chocolate, iron, beer}, then the feature vector of the sentece "Reid saw me looking at the iron bars." will be $< 0, 1, 0 >$ because, iron is the only word present in the context.

Local Collocations:

A local collocation $C_{i,j}$ refers to the ordered sequence of tokens in the local narrow context of w. For example, for the sentece, "Reid saw me looking at the iron bars.", the word bars, $C_{-2,-1}$ would be "the_iron".

In order to represent he knowledge source of local collocations, we extract 11 features corresponding to the following collocations: $C_{-1,-1}$, $C_{1,1}$, $C_{-2,-2}$, $C_{2,2}$, $C_{-2,-1}$, $C_{-1,1}$, $C_{1,2}$, $C_{-3,-1}$, $C_{-2,1}$, $C_{-1,2}$, $C_{1,3}$.

Syntactic Relations

We pass the sentence containing the word w through a statistical parser. The constituent tree is converted into a dependency tree in which every word points to a parent headword. Depending on the POS of w, we use different types of syntactic relations. For example, if w is a nound, we use four syntactic features viz : the parent headword h, the POS of h, the voice of h, and relative position of h from w. If it's a verb, we use six features viz nearest word l to the left of w such that w is the parent headword of l, the nearest word r, to the right similarly, POS of l , POS of r, POS of w and voice of w. For adjectives, we use two features viz the parent headword h and the POS of h.

2 Future Plan

- **Word Sense Disambiguation using LSTM or BiLSTM :** The model is trained end-to-end, directly from the raw text to sense labels, and makes effective use of word order.

- **WSD in Knowledge Based Algorithms:** We will try to implement 1-2 knowledge based algorithm as Lesk, Walker using concept of overlapping etc.
- **Analysis :** We are going to analyse/compare the baseline models, knowledge based and neural models on two different datasets SemCor and SemEval.

3 Problems Faced :

- BERT model is consuming huge memory due to which not able to produce good results for baseline model.