A Lucene and Maximum Entropy Model Based Hedge Detection System

Lin Chen
University of Illinois at Chicago
Chicago, IL, USA
`lin@chenlin.net`

Barbara Di Eugenio
University of Illinois at Chicago
Chicago, IL, USA
`bdieugen@uic.edu`

Abstract

This paper describes the approach to hedge detection we developed, in order to participate in the shared task at CoNLL-2010. A supervised learning approach is employed in our implementation. Hedge cue annotations in the training data are used as the seed to build a reliable hedge cue set. Maximum Entropy (MaxEnt) model is used as the learning technique to determine uncertainty. By making use of Apache Lucene, we are able to do fuzzy string match to extract hedge cues, and to incorporate part-of-speech (POS) tags in hedge cues. Not only can our system determine the certainty of the sentence, but is also able to find all the contained hedges. Our system was ranked third on the Wikipedia dataset. In later experiments with different parameters, we further improved our results, with a 0.612 F-score on the Wikipedia dataset, and a 0.802 F-score on the biological dataset.

1 Introduction

A hedge is a mitigating device used to lessen the impact of an utterance[1]. As a very important way to precisely express the degree of accuracy and truth assessment in human communication, hedging is widely used in both spoken and written languages. Detecting hedges in natural language text can be very useful for areas like text mining and information extraction. For example, in opinion mining, hedges can be used to assess the degree of sentiment, and refine sentiment classes from {positive, negative, objective} to {positive, somehow positive, objective, somehow objective, negative, somehow negative}.

[1] `http://en.wikipedia.org/wiki/Hedge(linguistics)`

Hedge detection related work has been conducted by several people. Light et al. (2004) started to do annotations on biomedicine article abstracts, and conducted the preliminary work of automatic classification for uncertainty. Medlock and Briscoe (2007) devised detailed guidelines for hedge annotations, and used a probabilistic weakly supervised learning approach to classify hedges. Ganter and Strube (2009) took Wikipedia articles as training corpus, used weasel words' frequency and syntactic patterns as features to classify uncertainty.

The rest of the paper is organized as follows. Section 2 shows the architecture of our system. Section 3 explains how we make use of Apache Lucene to do fuzzy string match and incorporate POS tag in hedge cues and our method to generate hedge cue candidates. Section 4 describes the details of using MaxEnt model to classify uncertainty. We present and discuss experiments and results in section 5, and conclude in section 6.

2 System Architecture

Our system is divided into training and testing modules. The architecture of our system is shown in Figure 1.

In the training module, we use the training corpus to learn a reliable hedge cue set with balanced support and confidence, then train a MaxEnt model for each hedge cue to classify the uncertainty for sentences matched by that hedge cue.

In the testing module, the learned hedge cues are used to match the sentences to classify, then each matched sentence is classified using the corresponding MaxEnt model. A sentence will be classified as uncertain if the MaxEnt model determines it is. Because of this design, our system is not only able to check if a sentence is uncertain, but also can detect the contained hedges.

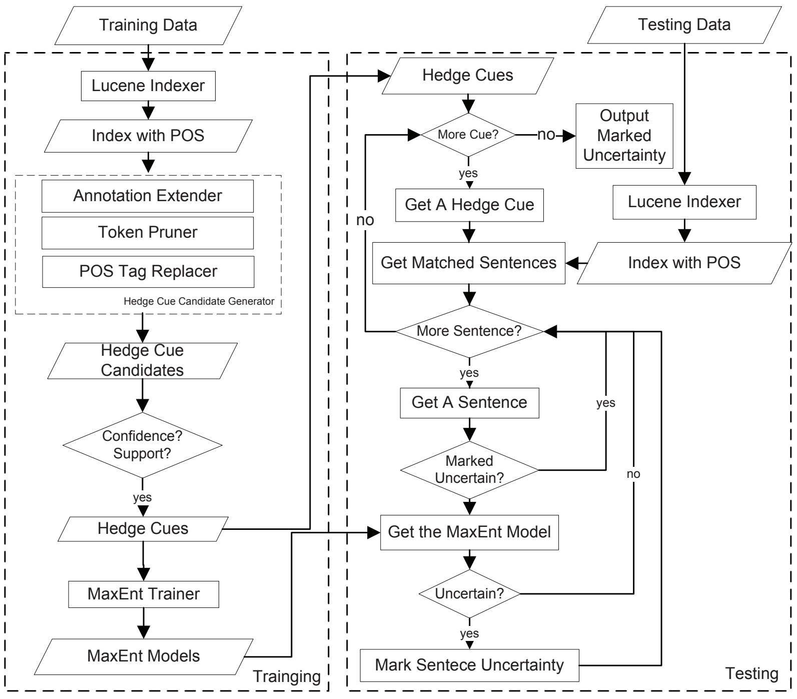

Figure 1: System Architecture

3 Learn Hedge Cues

The training data provided by CoNLL-2010 shared task contain "<ccue></ccue>" annotations for uncertain sentences. Most of the annotations are either too strict, which makes them hard to use to match other sentences, or too general, which means that most of the matched sentences are not uncertain.

Similar to how Liu (2007) measures the usefulness of association rules, we use support and confidence to measure the usefulness of a hedge cue.

Support is the ratio of sentences containing a hedge cue to all sentences. Because in a training dataset, the number of all the sentences is a fixed constant, we only use the number of sentences containing the hedge cue as support, see formula 1. In the other part of this paper, sentences matched by hedge cues means sentences contains hedge cues. We use support to measure the degree of generality of a hedge cue.

$$sup = count\,of\,matched\,sentences \tag{1}$$

Confidence is the ratio of sentences which contain a hedge cue and are uncertain to all the sentences containing the hedge cue, as formula 2. We use confidence to measure the reliability for a word or phrase to be a hedge cue.

$$conf = \frac{count\,of\,matched\,and\,uncertain}{count\,of\,matched\,sentences} \tag{2}$$

3.1 Usage of Apache Lucene

Apache Lucene[2] is a full text indexing Java library provided as an open source project of Apache Foundation. It provides flexible indexing and search capability for text documents, and it has very high performance. To explain the integration of Lucene into our implementation, we need to introduce several terms, some of which come from McCandless et al. (2010).

- Analyzer: Raw texts are preprocessed before being added to the index: text preprocessing components such as tokenization, stop words removal, and stemming are parts of an analyzer.

- Document: A document represents a collection of fields, it could be a web page, a text file, or only a paragraph of an article.

- Field: A field represents a document or the meta-data associated with that document, like the author, type, URL. A field has a name and a value, and a bunch of options to control how Lucene will index its value.

- Term: The very basic unit of a search. It contains a field name and a value to search.

[2]http://lucene.apache.org

- Query: The root class used to do search upon an index.

In our implementation, Lucene is used for the following 3 purposes:

- Enable quick counting for combinations of words and POS tags.
- Store the training and testing corpus for fast counting and retrieval.
- Allow gap between words or POS tags in hedge cues to match sentences.

Lucene provides the capability to build customized analyzers for complex linguistics analysis. Our customized Lucene analyzer employs tokenizer and POS tagger from OpenNLP tools[3] to do tokenization and POS tagging. For every word in the sentence, we put two Lucene tokens in the same position, by setting up the second token's PositionIncremental attribute to be 0.

For example, for sentence *it is believed to be very good*, our analyzer will make Lucene store it as Figure 2 in its index.

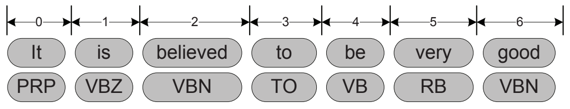

Figure 2: Customized Tokenizer Example

Indexing text in that way, we are able to match sentences cross words and POS tags. For example, the phrase *it is believed* will be matched by *it is believed*, *it is VBN*, *it VBZ believed*. This technique enables us to generalize a hedge cue.

In our implementation, all the data for training and testing are indexed. The indexing schema is: a sentence is treated as a Lucene document; the content of the sentence is analyzed by our customized analyzer; other information like sentence id, sentence position, uncertainty is stored as fields of the document. In this way, we can query all those fields, and when we find a match, we can easily get all the information out just from the index.

Lucene provides various types of queries to search the indexed content. We use SpanNearQuery and BooleanQuery to search the matched sentences for hedge cues. We rely on SpanNearQuery's feature of allowing positional restriction when matching sentences. When building a SpanNearQuery, we can specify the position gap allowed among the terms in the query. We build a SpanNearQuery from a hedge cue, put each token as a term of the query, and set the position gap to be 2. Take Figure 3 as an example, because the gap between token *is* and *said* is 1, is less than the specified gap setting 2, so *It is widely said to be good* will count as a match with hedge cue *is said*.

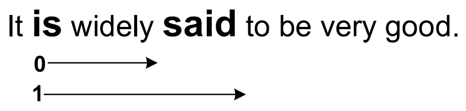

Figure 3: SpanNearQuery Matching Example

We use BooleanQuery with nested SpanNearQuery and TermQuery to count uncertain sentences, then to calculate the confidence of a hedge cue.

3.2 Hedge Cue Candidate Generation

We firstly tried to use the token as the basic unit for hedge cues. However, several pieces of evidence suggest it is not appropriate.

- Low Coverage. We only get 42 tokens in Wikipedia training data, using 20, 0.4 as the thresholds for support and confidence.
- Irrelevant words or stop words with lower thresholds. When we use 5, 0.3 as the thresholds for coverage and confidence, we get 279 tokens, however, words like *is*, *his*, *musical*, *voters*, *makers* appear in the list.

We noticed that many phrases with similar structures or fixed collocations appear very often in the annotations, like *it is believed*, *it is thought*, *many of them*, *many of these* and etc. Based on this observation, we calculated the support and confidence for some examples, see table 1.

Hedge Cue	**Sup.**	**Conf.**
it is believed	14	.93
by some	30	.87
many of	135	.65

Table 1: Hedge Cue Examples

We decided to use the phrase or collocation as the basic unit for hedge cues. There are two problems in using the original annotations as hedge cues:

[3]http://opennlp.sourceforge.net

- High confidence but low coverage: annotations that contain proper nouns always have very high confidence, usually 100%, however, they have very low support.
- High coverage but low confidence: annotations with only one token are very frequent, but only a few of them result in enough confidence.

To balance confidence and support, we built our hedge cue candidate generator. Its architecture is presented in Figure 4.

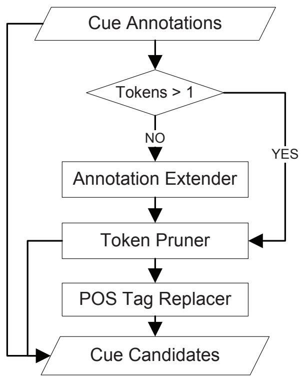

Figure 4: Hedge Cue Candidate Generator

The three main components of the hedge cue candidate generator are described below.

Annotation Extender: When the input hedge cue annotation contains only 1 token, this component will be used. It will generate 3 more hedge cue candidates by adding the surrounding tokens. We expect to discover candidates with higher confidence.

Token Pruner: According to our observations, proper nouns rarely contribute to the uncertainty of a sentence, and our Lucene based string matching method ensures that the matched sentences remain matched after we remove tokens from the original cue annotation. So we remove proper nouns in the original cue annotation to generate hedge cue candidates. By using this component, we expect to extract hedge cues with higher support.

POS Tag Replacer: This component is used to generalize similar phrases, by using POS tags to replace the concrete words. For example, we use the POS tag *VBN* to replace *believed* in *it is believed* to generate *it is VBN*. Hence, when a sentence contains *it is thought* in the testing dataset, even if *it is thought* never appeared in the training data set, we will still be able to match it and classify it against the trained MaxEnt model. We expect that this component will be able to increase support. Due to the O(2^n) time complexity, we did not try the brute force approach to replace every word, only the words with the POS tags in Table 2 are replaced in the process.

POS	Description	Example
VBN	past participle verb	it is **believed**
NNS	plural common noun	some **countries**
DT	determiner	some of **those**
CD	numeral, cardinal	**one** of the best

Table 2: POS Tag Replacer Examples

After hedge cue candidates are generated, we convert them to Lucene queries to calculate their confidence and support. We prune those that fall below the predefined confidence and support settings.

4 Learn Uncertainty

Not all the learned hedge cues have 100% uncertainty confidence, given a hedge cue, we need to learn how to classify whether a matched sentence is uncertain or not. The classification model is, given a tuple of (Sentence, Hedge Cue), in which the sentence contains the hedge cue, we classify it to the outcome set {Certain, Uncertain}.

MaxEnt is a general purpose machine learning technique, it makes no assumptions in addition to what we know from the data. MaxEnt has been widely used in Natural Language Processing (NLP) tasks like POS tagging, word sense disambiguation, and proved its efficiency. Due to MaxEnt's capability to combine multiple and dependent knowledge sources, we employed MaxEnt as our machine learning model. Features we used to train the model include meta information features and collocation features.

Meta Information Features include three features:

- Sentence Location: The location of the sentence in the article, whether in the title or in the content. We observed sentences in the title are rarely uncertain.
- Number of Tokens: The number of tokens in the sentence. Title of article is usually shorter, and more likely to be certain.

- Hedge Cue Location: The location of matched tokens in a sentence. We consider them to be in the beginning, if the first token of the matched part is the first token in the sentence; to be at the end, if the last token of the matched part is the last token of the sentence; otherwise, they are in the middle. We were trying to use this feature as a simplified version to model the syntactic role of hedge cues in sentences.

Collocation Features include the word and POS tag collocation features:

- Word Collocation: Using a window size of 5, extract all the word within that window, excluding punctuation.

- POS Tag Collocation: Using a window size of 5, extract all the POS tags of tokens within that window, excluding punctuation.

We use the OpenNLP MaxEnt[4] Java library as the MaxEnt trainer and classifier. For each hedge cue, the training is iterated 100 times, with no cut off threshold for events.

5 Experiments and Discussion

We first ran experiments to evaluate the performance of the entire system. We used official dataset as training and testing, with different confidence and support thresholds. The result on official Wikipedia dataset is presented in Table 3. Result on the biological dataset is listed in Table 4. In the result tables, the first 2 columns are the confidence and support threshold; "Cues" is the number of generated hedge cues; the last 3 columns are standard classifier evaluation measures.

Our submitted result used 0.35, 5 as the thresholds for confidence and support. We officially placed third on the Wikipedia dataset, with a 0.5741 F-score, and third from last on the biological dataset, with a 0.7692 F-score. In later experiments, we used different parameters, which resulted in a 0.03 F-score improvement. We believe the big difference of ranking on different datasets comes from the incomplete training. Due to incorrect estimation of running time, we only used the smaller training file in our submitted biological result.

From Table 3 and 4, we can see that a higher confidence threshold gives higher precision, and

Conf.	Sup.	Cues	Prec.	Recall	F
0.4	10	360	0.658	0.561	0.606
	15	254	0.672	0.534	0.595
	20	186	0.682	0.508	0.582
0.45	10	293	0.7	0.534	0.606
	15	190	0.717	0.503	0.591
	20	137	0.732	0.476	0.577
0.5	5	480	0.712	0.536	**0.612**
	10	222	0.736	0.492	0.590
	15	149	0.746	0.468	0.575
	20	112	0.758	0.443	0.559

Table 3: Evaluation Result on Wikipedia Dataset

Conf.	Sup.	Cues	Prec.	Recall	F
0.4	10	330	0.68	0.884	0.769
	15	229	0.681	0.861	0.76
	20	187	0.679	0.842	0.752
0.45	10	317	0.689	0.878	0.772
	15	220	0.69	0.857	0.764
	20	179	0.688	0.838	0.756
0.5	5	586	0.724	0.899	**0.802**
	10	297	0.742	0.841	0.788
	15	206	0.742	0.819	0.779
	20	169	0.74	0.8	0.769

Table 4: Evaluation Result on Biological Dataset

a lower support threshold leads to higher recall. Since the lower support threshold could generate more hedge cues, it will generate less training instances for hedge cues with both low confidence and support, which affects the performance of the MaxEnt classifier. In both datasets, it appears that 0.5 and 5 are the best thresholds for confidence and support, respectively.

Beyond the performance of the entire system, our hedge cue generator yields very promising results. Using the best parameters we just noted above, our hedge cue generator generates 52 hedge cues with confidence 100% on the Wikipedia dataset, and 332 hedge cues in the biological dataset. Some hedge cue examples are shown in Table 5.

We also ran experiments to verify the performance of our MaxEnt classifier. We used the same setting of datasets as for the system performance evaluation. Given a hedge cue, we extracted all the matched sentences from the training set to train a MaxEnt classifier, and used it to classify the matched sentences by the hedge cue in testing set.

[4]http://maxent.sourceforge.net

Hedge Cue	Sup.	Conf.	TestSize	Prec.	Recall	F
indicated that	63	0.984	6	1.0	1.0	1.0
by some	30	0.867	29	0.966	1.000	0.983
are considered	29	0.724	10	0.750	0.857	0.800
some of NNS	62	0.613	27	1.000	0.778	0.875
the most JJ	213	0.432	129	0.873	0.475	0.615

Table 6: MaxEnt Classifier Performance

Hedge Cue	Conf.	Sup.
probably VBN	1.0	21
DT probably	1.0	15
many NNS believe	1.0	10
NNS suggested DT	1.0	248
results suggest	1.0	122
has VBN widely VBN	1.0	10

Table 5: Generated Hedge Cue Examples

Table 6 shows the results, the hedge cues were manually chosen with relative higher support.

We can see that the performance of the MaxEnt classifier correlates tightly with confidence and support. Higher confidence means a more accurate detection for a phrase to be hedge cue, while higher support means more training instances for the classifier: the best strategy would be to find hedge cues with both high confidence and support.

While experimenting with the system, we found several potential improvements.

- Normalize words. Take the word *suggest* as an example. In the generated hedge cues, we found that its other forms are everywhere, like *it suggested*, *NNS suggests a*, and *DT suggesting that*. As we put POS tags into Lucene index, we can normalize words to their base forms using a morphology parser, and put base forms into index. After that, the query with *suggest* will match all the forms.

- Use more sophisticated features to train the MaxEnt classifier. Currently we only use shallow linguistics information as features, however we noticed that the role of the phrase could be very important to decide whether it indicates uncertainty. We can deep parse sentences, extract the role information, and add it to the feature list of classifier.

6 Conclusion

In this paper, we described the hedge detection system we developed to participate in the shared task of CoNLL-2010. Our system uses a heuristic learner to learn hedge cues, and uses MaxEnt as its machine learning model to classify uncertainty for sentences matched by hedge cues. Hedge cues in our system include both words and POS tags, which make them more general. Apache Lucene is integrated into our system to efficiently run complex linguistic queries on the corpus.

Acknowledgments

This work is supported by award IIS-0905593 from the National Science Foundation.

References

Viola Ganter and Michael Strube. 2009. Finding hedges by chasing weasels: Hedge detection using Wikipedia tags and shallow linguistic features. In *Proceedings of the ACL-IJCNLP 2009 Conference Short Papers*, pages 173–176, Suntec, Singapore, August. Association for Computational Linguistics.

Marc Light, Xin Ying Qiu, and Padmini Srinivasan. 2004. The language of bioscience: Facts, speculations, and statements in between. In *Proceedings of BioLink 2004 Workshop on Linking Biological Literature, Ontologies and Databases: Tools for Users*, pages 17–24, Boston, Mass, May.

Bing Liu. 2007. *Web data mining: exploring hyperlinks, contents, and usage data*. Springer.

Michael McCandless, Erik Hatcher, and Otis Gospodneti. 2010. *Lucene in action*. Manning Publications Co, 2nd edition.

Ben Medlock and Ted Briscoe. 2007. Weakly supervised learning for hedge classification in scientific literature. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 992–999. Association for Computational Linguistics, June.