Features for Detecting Hedge Cues

Nobuyuki Shimizu
Information Technology Center
The University of Tokyo
shimizu@r.dl.itc.u-tokyo.ac.jp

Hiroshi Nakagawa
Information Technology Center
The University of Tokyo
n3@dl.itc.u-tokyo.ac.jp

Abstract

We present a sequential labeling approach to hedge cue detection submitted to the biological portion of task 1 for the CoNLL-2010 shared task. Our main approach is as follows. We make use of partial syntactic information together with features obtained from the unlabeled corpus, and convert the task into one of sequential BIO-tagging. If a cue is found, a sentence is classified as uncertain and certain otherwise. To examine a large number of feature combinations, we employ a genetic algorithm. While some features obtained by this method are difficult to interpret, they were shown to improve the performance of the final system.

1 Introduction

Research on automatically extracting factual information from biomedical texts has become popular in recent years. Since these texts are abundant with hypotheses postulated by researchers, one hurdle that an information extraction system must overcome is to be able to determine whether or not the information is part of a hypothesis or a factual statement. Thus, detecting hedge cues that indicate the uncertainty of the statement is an important subtask of information extraction (IE). Hedge cues include words such as "may", "might", "appear", "suggest", "putative" and "or". They also includes phrases such as "...raising an intriguing question that..." As these expressions are sparsely scattered throughout the texts, it is not easy to generalize results of machine learning from a training set to a test set. Furthermore, simply finding the expressions listed above does not guarantee that a sentence contains a hedge. Their function as a hedge cue depends on the surrounding context.

The primary objective of the CoNLL-2010 shared task (Farkas et al., 2010) is to detect hedge cues and their scopes as are present in biomedical texts. In this paper, we focus on the biological portion of task 1, and present a sequential labeling approach to hedge cue detection. The following summarizes the steps we took to achieve this goal. Similarly to previous work in hedge cue detection (Morante and Daelemans, 2009), we first convert the task into a sequential labeling task based on the BIO scheme, where each word in a hedge cue is labeled as B-CUE, I-CUE, or O, indicating respectively the labeled word is at the beginning of a cue, inside of a cue, or outside of a hedge cue; this is similar to the tagging scheme from the CoNLL-2001 shared task. We then prepared features, and fed the training data to a sequential labeling system, a discriminative Markov model much like Conditional Random Fields (CRF), with the difference being that the model parameters are tuned using Bayes Point Machines (BPM), and then compared our model against an equivalent CRF model. To convert the result of sequential labeling to sentence classification, we simply used the presence of a hedge cue, i.e. if a cue is found, a sentence is classified as uncertain and certain otherwise.

To prepare features, we ran the GENIA tagger to add partial syntactic parse and named entity information. We also applied Porter's stemmer (Jones and Willet, 1997) to each word in the corpus. For each stem, we acquired the distribution of surrounding words from the unlabeled corpus, and calculated the similarity between these distributions and the distribution of hedge cues in the training corpus. Given a stem and its similarities to different hedge cues, we took the maximum similarity and discretized it. All these features are passed on to a sequential labeling system. Using these base features, we then evaluated the effects of feature combinations by repeatedly training the system and selecting feature combinations that increased the performance on a heldout set. To automate this process, we employed a genetic algorithm.

The contribution of this paper is two-fold. First, we describe our system, outlined above, that we submitted to the CoNLL-2010 shared task in more detail. Second, we analyze the effects of particular choices we made when building our system, especially the feature combinations and learning methods.

The rest of this paper is organized as follows. In Section 2, we detail how the task of sequential labeling is formalized in terms of linear classification, and explain the Viterbi algorithm required for prediction. We next present several algorithms for optimizing the weight vector in a linear classifier in Section 3. We then detail the complete list of feature templates we used for the task of hedge cue detection in Section 4. In order to evaluate the effects of feature templates, in Section 5, we remove each feature template and find that several feature templates overfit the training set. We finally conclude with Section 6.

2 Sequential Labeling

We discriminatively train a Markov model using Bayes Point Machines (BPM). We will first explain linear classification, and then apply a Markov assumption to the classification formalism. Then we will move on to BPM. Note that we assume all features are binary in this and upcoming sections as it is sufficient for the task at hand.

In the setting of sequential labeling, given the input sequence $\mathbf{x} = (x_1, x_2, x_3, ...x_n)$, a system is asked to produce the output sequence $\mathbf{y} = (y_1, y_2, y_3, ...y_n)$. Considering that $\mathbf{y}$ is a class, sequential labeling is simply a classification with a very large number of classes. Assuming that the problem is one of linear classification, we may create a binary feature vector $\phi(\mathbf{x})$ for an input $\mathbf{x}$ and have a weight vector $\mathbf{w_y}$ of the same dimension for each class $\mathbf{y}$. We choose a class $\mathbf{y}$ that has the highest dot product between the input vector and the weight vector for the class $\mathbf{y}$. For binary classification, this process is very simple: compare two dot product values. Learning is therefore reduced to specifying the weight vectors.

To follow the standard notations in sequential labeling, let weight vectors $\mathbf{w_y}$ be stacked into one large vector $\mathbf{w}$, and let $\phi(\mathbf{x}, \mathbf{y})$ be a binary feature vector such that $\mathbf{w}^\top \phi(\mathbf{x}, \mathbf{y})$ is equal to $\mathbf{w_y}^\top \phi(\mathbf{x})$. Classification is to choose $\mathbf{y}$ such that $\mathbf{y} = argmax_{\mathbf{y}'}(\mathbf{w}^\top \phi(\mathbf{x}, \mathbf{y}'))$.

Unfortunately, a large number of classes created out of sequences makes the problem intractable, so the Markov assumption factorizes $\mathbf{y}$ into a sequence of labels, such that a label y_i is affected only by the label before and after it (y_{i-1} and y_{i+1} respectively) in the sequence. Each structure, or label $\mathbf{y}$ is now associated with a set of the parts $parts(\mathbf{y})$ such that $\mathbf{y}$ can be recomposed from the parts. In the case of sequential labeling, parts consist of states y_i and transitions $y_i \rightarrow y_{i+1}$ between neighboring labels. We assume that the feature vector for an entire structure $\mathbf{y}$ decomposes into a sum over feature vectors for individual parts as follows: $\phi(\mathbf{x}, \mathbf{y}) = \sum_{r \in parts(\mathbf{y})} \phi(\mathbf{x}, r)$. Note that we have overloaded the symbol ϕ to apply to either a structure y or its parts r.

The Markov assumption for factoring labels lets us use the Viterbi algorithm (much like a Hidden Markov Model) in order to find

$$\begin{aligned} \mathbf{y} &= argmax_{\mathbf{y}'} \quad (\mathbf{w}^\top \phi(\mathbf{x}, \mathbf{y}')) \\ &= argmax_{\mathbf{y}'} \quad (\textstyle\sum_{j=1}^{n} \mathbf{w}^\top \phi(\mathbf{x}, y'_j) \\ &\qquad + \textstyle\sum_{j=1}^{n-1} \mathbf{w}^\top \phi(\mathbf{x}, y'_j \rightarrow y'_{j+1})). \end{aligned}$$

3 Optimization

We now turn to the optimization of the weight parameter $\mathbf{w}$. We compare three approaches – Perceptron, Bayes Point Machines and Conditional Random Fields, using our c++ library for structured output prediction [1].

Perceptron is an online update scheme that leaves the weights unchanged when the predicted output matches the target, and changes them when it does not. The update is:

$$\mathbf{w}_k := \mathbf{w}_k - \phi(\mathbf{x}^i, \mathbf{y}) + \phi(\mathbf{x}^i, \mathbf{y}^i).$$

Despite its seemingly simple update scheme, perceptron is known for its effectiveness and performance (Collins, 2002).

Conditional Random Fields (CRF) is a conditional model

$$P(\mathbf{y}|\mathbf{x}) = \frac{1}{Z_\mathbf{x}} exp(\mathbf{w}^\top \phi(\mathbf{x}, \mathbf{y}))$$

where w is the weight for each feature and $Z_\mathbf{x}$ is a normalization constant for each $\mathbf{x}$.

$$Z_\mathbf{x} = \sum_{\mathbf{y}} exp(\mathbf{w}^\top \phi(\mathbf{x}, \mathbf{y}))$$

[1] Available at http://soplib.sourceforge.net/

for structured output prediction. To fit the weight vector $\mathbf{w}$ using the training set $\{(\mathbf{x}^i, \mathbf{y}^i)\}_{i=1}^n$, we use a standard gradient-descent method to find the weight vector that maximizes the log likelihood $\sum_i^n logP(\mathbf{y}^i|x^i)$ (Sha and Pereira, 2003). To avoid overfitting, the log likelihood is often penalized with a spherical Gaussian weight prior: $\sum_i^n logP(\mathbf{y}^i|x^i) - \frac{C||\mathbf{w}||}{2}$. We also evaluated this penalized version, varying the trade-off parameter C.

Bayes Point Machines (BPM) for structured prediction (Corston-Oliver et al., 2006) is an ensemble learning algorithm that attempts to set the weight $\mathbf{w}$ to be the Bayes Point which approximates to Bayesian inference for linear classifiers. Assuming a uniform prior distribution over w, we revise our belief of w after observing the training data and produce a posterior distribution. We create the final w_{bpm} for classification using a posterior distribution as follows:

$$\mathbf{w}_{bpm} = E_{p(\mathbf{w}|D)}[\mathbf{w}] = \sum_{i=1}^{|V(D)|} p(\mathbf{w}_i|D)\mathbf{w}_i$$

where $p(\mathbf{w}|D)$ is the posterior distribution of the weights given the data D and $E_{p(\mathbf{w}|D)}$ is the expectation taken with respect to this distribution. $V(D)$ is the version space, which is the set of weights $\mathbf{w}_i$ that classify the training data correctly, and $|V(D)|$ is the size of the version space. In practice, to explore the version space of weights consistent with the training data, BPM trains a few different perceptrons (Collins, 2002) by shuffling the samples. The approximation of Bayes Point $\mathbf{w}_{bpm}$ is the average of these perceptron weights:

$$\mathbf{w}_{bpm} = E_{p(\mathbf{w}|D)}[\mathbf{w}] \approx \sum_{k=1}^{K} \frac{1}{K}\mathbf{w}_k.$$

The pseudocode of the algorithm is shown in Algorithm 3.1. We see that the inner loop is simply a perceptron algorithm.

Algorithm 3.1: BPM$(K, T, \{(\mathbf{x}^i, \mathbf{y}^i)\}_{i=1}^n)$

```
$\mathbf{w}_{bpm} := \mathbf{0}$;
for $k := 1$ to $K$
    Randomly shuffle the sequential order of
     samples $\{(\mathbf{x}^i, \mathbf{y}^i)\}_{i=1}^n$
    $\mathbf{w}_k := \mathbf{0}$;
    for $t := 1$ to $T$   # Perceptron iterations
        for $i := 1$ to $n$   # Iterate shuffled samples
            $\mathbf{y} := argmax_{\mathbf{y}'}(\mathbf{w}_k^\top \phi(\mathbf{x}^i, \mathbf{y}'))$
            if $(\mathbf{y} \neq \mathbf{y}^i)$
                $\mathbf{w}_k := \mathbf{w}_k - \phi(\mathbf{x}^i, \mathbf{y}) + \phi(\mathbf{x}^i, \mathbf{y}^i)$;
    $\mathbf{w}_{bpm} := \mathbf{w}_{bpm} + \frac{1}{K}\mathbf{w}_k$;
return $(\mathbf{w}_{bpm})$
```

4 Features

4.1 Base Features

For each sentence $\mathbf{x}$, we have state features, represented by a binary vector $\phi(\mathbf{x}, y'_j)$ and transition features, again a binary vector $\phi(\mathbf{x}, y'_j \to y'_{j+1})$.

For transition features, we do not utilize lexicalized features. Thus, each dimension of $\phi(\mathbf{x}, y'_j \to y'_{j+1})$ is an indicator function that tests a combination of labels, for example, O→B-CUE, B-CUE→I-CUE or I-CUE→O.

For state features $\phi(\mathbf{x}, y'_j)$, the indicator function for each dimension tests a combination of y'_j and lexical features obtained from $\mathbf{x} = (x_1, x_2, x_3, ...x_n)$. We now list the base lexical features that were considered for this experiment.

F^0 a token, which is usually a word. As a part of preprocessing, words in each input sentence are tokenized using the GENIA tagger [2]. This tokenization coincides with Penn Treebank style tokenization [3].

We add a subscript to indicate the position. F_j^0 is exactly the input token x_j. From x_j, we also create other lexical features such as F_j^1, F_j^2, F_j^3, and so on.

F^1 the token in lower case, with digits replaced by the symbol #.

F^2 1 if the letters in the token are all capitalized, 0 otherwise.

F^3 1 if the token contains a digit, 0 otherwise.

F^4 1 if the token contains an uppercase letter, 0 otherwise.

F^5 1 if the token contains a hyphen, 0 otherwise.

[2]Available at: http:// www-tsujii.is.s.u-tokyo.ac.jp/ GENIA/ tagger/

[3]A tokenizer is available at: http:// www.cis.upenn.edu/ treebank/ tokenization.html

F^6 first letter in the token.

F^7 first two letters in the token.

F^8 first three letters in the token.

F^9 last letter in the token.

F^{10} last two letters in the token.

F^{11} last three letters in the token.

The features F^0 to F^{11} are known to be useful for POS tagging. We postulated that since most frequent hedge cues tend not to be nouns, these features might help identify them.

The following three features are obtained by running the GENIA tagger.

F^{12} a part of speech.

F^{13} a CoNLL-2000 style shallow parse. For example, B-NP or I-NP indicates that the token is a part of a base noun phrase, B-VP or I-VP indicates that it is part of a verb phrase.

F^{14} named entity, especially a protein name.

F^{15} a word stem by Porter's stemmer [4]. Porter's stemmer removes common morphological and inflectional endings from words in English. It is often used as part of an information retrieval system.

Upon later inspection, it seems that Porter's stemmer may be too aggressive in stemming words. The word *putative*, for example, after being processed by the stemmer, becomes simply *put* (which is clearly erroneous).

The last nine types of features utilize the unlabeled corpus for the biological portion of shared task 1, provided by the shared task organizers. For each stem, we acquire a histogram of surrounding words, with a window size of 3, from the unlabeled corpus. Each histogram is represented as a vector; the similarity between histograms was then computed. The similarity metric we used is called the Tanimoto coefficient, also called extended/vector-based Jaccard coefficient.

$$\frac{\mathbf{v}_i \cdot \mathbf{v}_j}{||\mathbf{v}_i|| + ||\mathbf{v}_j|| - \mathbf{v}_i \cdot \mathbf{v}_j}$$

It is based on the dot product of two vectors and reduces to Jaccard coefficient for binary features.

[4]Available at: http://tartarus.org/ martin/PorterStemmer/

This metric is known to perform quite well for near-synonym discovery (Hagiwara et al., 2008). Given a stem and its similarities to different hedge cues, we took the maximum similarity and discretized it.

F^{16} 1 if similarity is bigger than 0.9, 0 otherwise.

...

F^{19} 1 if similarity is bigger than 0.6, 0 otherwise.

...

F^{24} 1 if similarity is bigger than 0.1, 0 otherwise.

This concludes the base features we considered.

4.2 Combinations of Base Features

In order to discover combinations of base features, we implemented a genetic algorithm (Goldberg, 1989). It is an adaptive heuristic search algorithm based on the evolutionary ideas of natural selection and genetics. After splitting the training set into three partitions, given the first partition as the training set, the fitness is measured by the score of predicting the second partition. We removed the feature sets that did not score high, and introduced mutations – new feature sets – as replacements. After several generations, surviving feature sets performed quite well. To avoid over fitting, occasionally feature sets were evaluated on the third partition, and we finally chose the feature set according to this partition.

The features of the submitted system are listed in Table 1. Note that Table 1 shows the dimensions of the feature vector that evaluate to 1 given $\mathbf{x}$ and y'_j. The actual feature vector is created by instantiating all the combinations in the table using the training set.

Surprisingly, our genetic algorithm removed features F^{10} and F^{11}, the last two/three letters in a token. It also removed the POS information F^{12}, but kept the sequence of POS tags $F^{12}_{j-1}, F^{12}_j, F^{12}_{j+1}, F^{12}_{j+2}, F^{12}_{j+3}$. The reason for longer sequences is due to our heuristics for mutations. Occasionally, we allowed the genetic algorithm to insert a longer sequence of feature combinations at once. One other notable observation is that shallow parses and NEs are removed. Between the various thresholds from F^{16} to F^{24}, it only kept F^{19}, discovering 0.6 as a similarity threshold.

State $\phi(\mathbf{x}, y'_j)$
y'_j
y'_j, F^0_{j-2}
y'_j, F^0_{j-1}
y'_j, F^0_j
y'_j, F^0_j, F^{19}_j
$y'_j, F^0_{j-1}, F^0_j, F^0_{j+1}, F^0_{j+2}, F^0_{j+3}, F^0_{j+4}$ –(1)
y'_j, F^0_{j+1}
y'_j, F^0_{j+2}
y'_j, F^1_j
y'_j, F^2_j –(2)
y'_j, F^3_j
y'_j, F^4_j
$y'_j, F^4_{j-2}, F^4_{j-1}, F^4_j, F^4_{j+1}, F^4_{j+2}$
y'_j, F^5_j
y'_j, F^5_j, F^7_{j-1}
y'_j, F^6_j
y'_j, F^7_j
y'_j, F^8_j
$y'_j, F^9_{j-1}, F^9_j, F^9_{j+1}, F^9_{j+2}, F^9_{j+3}$
$y'_j, F^{12}_{j-1}, F^{12}_j, F^{12}_{j+1}, F^{12}_{j+2}, F^{12}_{j+3}$
$y'_j, F^{15}_j, F^{15}_{j+1}, F^{15}_{j+2}, F^{15}_{j+3}$
$y'_j, F^{19}_{j-2}, F^{19}_{j-1}, F^{19}_j, F^{19}_{j+1}, F^{19}_{j+2}$

Table 1: Features for Sequential Labeling

5 Experiments

In order to examine the effects of learning parameters, we conducted experiments on the test data after it was released to the participants of the shared task.

While BPM has two parameters, K and T, we fixed $T = 5$ and varied K, the number of perceptrons. As increasing the number of perceptrons results in more thorough exploration of the version space $V(D)$, we expect that the performance of the classifier would improve as K increases. Table 2 shows how the number of perceptrons affects the performance.

TP stands for True Positive, FP for False Positive, and FN for False Negative. The evaluation metrics were precision P (the number of true positives divided by the total number of elements labeled as belonging to the positive class) recall R (the number of true positives divided by the total number of elements that actually belong to the positive class) and their harmonic mean, the F_1 score ($F_1 = 2PR/(P + R)$). All figures in this paper measure hedge cue detection performance at the sentence classification level, not word/phrase classification level. From the results, once the number of perceptrons hits 20, the performance stabilizes and does not seem to show any improvement.

K	TP	FP	FN	P (%)	R (%)	F_1 (%)
10	641	80	149	88.90	81.14	84.84
20	644	79	146	89.07	81.52	85.13
30	644	80	146	88.95	81.52	85.07
40	645	81	145	88.84	81.65	85.09
50	645	80	145	88.97	81.65	85.15

Table 2: Effects of K in Bayes Point Machines

Next, in order to examine whether or not we have overfitted to the training/heldout set, we removed each row of Table 1 and reevaluated the performance of the system. Reevaluation was conducted on the labeled test set released by the shared task organizers after our system's output had been initially evaluated. Thus, these figures are comparable to the sentence classification results reported in Farkas et al. (2010).

	TP	FP	FN	P (%)	R (%)	F_1 (%)
1	647	79	143	89.12	81.90	85.36
2	647	80	143	89.00	81.90	85.30
1,2	647	81	143	88.87	81.90	85.24

Table 3: Effects of removing features (1) or (2), or both

Table 3 shows the effect of removing (1), (2), or both (1) and (2), showing that they overfit the training data. Removing any other rows in Table 1 resulted in decreased classification performance. While there are other large combination features such as ones involving F^4, F^9, F^{12}, F^{15} and F^{19}, we find that they do help improving the performance of the classifier. Since these features seem unintuitive to the authors, it is likely that they would not have been found without the genetic algorithm we employed. Error analysis shows that inclusion of features involving F^9 affects prediction of "believe", "possible", "putative", "assumed", "seemed", "if", "presumably", "perhaps", "suggestion", "suppose" and "intriguing". However, as this feature template is unfolded into a large number of features, we were unable to obtain further linguistic insights.

In the following experiments, we used the currently best performing features, that is, all features except (1) in Table 1, and trained the classifiers using the formalism of Perceptron and Conditional Random Fields besides Bayes Point Machines as we have been using. The results in Table 4 shows that BPM performs better than Perceptron or Conditional Random Fields. As the training time for BPM is better than CRF, our choice of BPM helped us to run the genetic algorithm repeatedly as well. After several runs of empirical tuning and tweaking, the hyper-parameters of the algorithms were set as follows. Perceptron was stopped at 40 iterations ($T = 40$). For BPM, we fixed $T = 5$ and $K = 20$. For Conditional Random Fields, we compared the penalized version with $C = 1$ and the unpenalized version ($C = 0$). The results in Table 4 is that of the unpenalized version, as it performed better than the penalized version.

Perceptron

TP	FP	FN	P (%)	R (%)	F_1 (%)
671	128	119	83.98	84.94	84.46

Conditional Random Fields

TP	FP	FN	P (%)	R (%)	F_1 (%)
643	78	147	89.18	81.39	85.11

Bayes Point Machines

TP	FP	FN	P (%)	R (%)	F_1 (%)
647	79	143	89.12	81.90	85.36

Table 4: Performance of different optimization strategies

6 Conclusion

To tackle the hedge cue detection problem posed by the CoNLL-2010 shared task, we utilized a classifier for sequential labeling following previous work (Morante and Daelemans, 2009). An essential part of this task is to discover the features that allow us to predict unseen hedge expressions. As hedge cue detection is semantic rather than syntactic in nature, useful features such as word stems tend to be specific to each word and hard to generalize. However, by using a genetic algorithm to examine a large number of feature combinations, we were able to find many features with a wide context window of up to 5 words. While some features are found to overfit, our analysis shows that a number of these features are successfully applied to the test data yielding good generalized performance. Furthermore, we compared different optimization schemes for structured output prediction using our c++ library, freely available for download and use. We find that Bayes Point Machines have a good trade-off between performance and training speed, justifying our repeated usage of BPM in the genetic algorithm for feature selection.

Acknowledgments

The authors would like to thank the reviewers for their comments. This research was supported by the Information Technology Center through their grant to the first author. We would also like to thank Mr. Ono, Mr. Yonetsuji and Mr. Yamada for their contributions to the library.

References

Michael Collins. 2002. Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In *Proceedings of Empirical Methods in Natural Language Processing (EMNLP).*

Simon Corston-Oliver, Anthony Aue, Kevin Duh, and Eric Ringger. 2006. Multilingual dependency parsing using bayes point machines. In *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*, pages 160–167, June.

Richárd Farkas, Veronika Vincze, György Móra, János Csirik, and György Szarvas. 2010. The CoNLL-2010 Shared Task: Learning to Detect Hedges and their Scope in Natural Language Text. In *Proceedings of the Fourteenth Conference on Computational Natural Language Learning (CoNLL-2010): Shared Task*, pages 1–12, Uppsala, Sweden. ACL.

David E. Goldberg. 1989. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley Professional.

Masato Hagiwara, Yasuhiro Ogawa, and Katsuhiko Toyama. 2008. Context feature selection for distributional similarity. In *Proceedings of IJCNLP-08.*

Karen Spärk Jones and Peter Willet. 1997. *Readings in Information Retrieval.* Morgan Kaufmann.

Roser Morante and Walter Daelemans. 2009. Learning the scope of hedge cues in biomedical texts. In *BioNLP '09: Proceedings of the Workshop on BioNLP*, pages 28–36.

Fei Sha and Fernando Pereira. 2003. Shallow parsing with conditional random fields. In *Proceedings of the Human Language Technology Conference (HLT).*