Hedge detection using a rewards and penalties approach

Ken Stahl
*University at Albany, State University of New York, kenualbany@gmail.com*

The University at Albany community has made this article openly available. Please share how this access benefits you.

Follow this and additional works at: [https://scholarsarchive.library.albany.edu/legacy-etd](https://scholarsarchive.library.albany.edu/legacy-etd)

Part of the Computer Sciences Commons

**Recommended Citation**
[https://scholarsarchive.library.albany.edu/legacy-etd/1018](https://scholarsarchive.library.albany.edu/legacy-etd/1018)

This Master's Thesis is brought to you for free and open access by the The Graduate School at Scholars Archive. It has been accepted for inclusion in Legacy Theses & Dissertations (2009 - 2024) by an authorized administrator of Scholars Archive.

Please see Terms of Use. For more information, please contact scholarsarchive@albany.edu.
Hedge Detection Using a Rewards and Penalties Approach

by

Ken Stahl

A Thesis
Submitted to the University at Albany, State University of New York
in Partial Fulfillment of
the Requirements for the Degree of
Master of Science

College of Computing & Information
Department of Computer Science
2013
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>1</td>
</tr>
<tr>
<td>Corpora Used and Ground Truth</td>
<td>3</td>
</tr>
<tr>
<td>Automatic Detection of Hedges in Game Chat</td>
<td>4</td>
</tr>
<tr>
<td>Cleaning Up the Text</td>
<td>4</td>
</tr>
<tr>
<td>Tokenization</td>
<td>6</td>
</tr>
<tr>
<td>Representation</td>
<td>7</td>
</tr>
<tr>
<td>Comparison</td>
<td>10</td>
</tr>
<tr>
<td>Applying a Genetic Algorithm to Find an Optimal Solution</td>
<td>13</td>
</tr>
<tr>
<td>Genetic Algorithm Settings</td>
<td>15</td>
</tr>
<tr>
<td>Weighting and Pruning</td>
<td>17</td>
</tr>
<tr>
<td>Performance Evaluation</td>
<td>18</td>
</tr>
<tr>
<td>Conclusion</td>
<td>20</td>
</tr>
<tr>
<td>References</td>
<td>21</td>
</tr>
</tbody>
</table>
Introduction

A hedge is a mitigating device, an added linguistic expression used to qualify or lessen the impact of an utterance. For example, hedges are used to express uncertainty in a statement to change an idea into a suggestion rather than a matter of fact. The following statements illustrate the impact hedges (in bold) have:

Example 1:
It will rain this afternoon.
Perhaps it will rain this afternoon.
It will probably rain this afternoon.

The bold words are hedges, and they lower the certainty of the statement. Consequences of being wrong are potentially worse without hedges because they explicitly reduce the confidence level in the statement. Writers thus reduce the “degree of liability” or responsibility that they might face in expressing referential information (Crismore & Van Kopple, 1988). An example, with the hedge in bold:

Example 2:
The current low price is an opportunity to buy gold.
The current low price may indicate an opportunity to buy gold.
In a situation where a person is attempting to convince another of an assertion, especially if the other is of higher status (or a stranger) or if the message is counter-attitudinal, it is seen as polite to give the person the feeling that the assertion is not a command. Example with hedge in bold:

Example 3:
You need to keep an eye on that trouble maker.
You **might** need to keep an eye on that trouble maker.

Hedges used in this way provide the listener the opportunity to reject or disagree with the assertion (Durik, Britt, Reynolds & Storey, 2008).

In addition, hedges are thought to be indicators of several real life higher-level classifications such as age. (Rosa Mikeal Martey, Jennifer Stromer-Galley, et. al., 2010)

The purpose of this project was to provide hedge data and statistics in a larger research project that examined players’ behavior in an interactive game. The data provided was a count of utterances containing a hedge per person; where an utterance is defined as a single entry in the chat log. The hedge distribution data was then combined with various other features in the chat text and analyzed for correlations with certain personal attributes of the players, such as age, gender, or leadership skills.
In order to deliver hedge use statistics, an automated hedge detection system was developed. This system and its performance are described below.

**Corpora Used and Ground Truth**

Two text corpora were used in this project. Both were chat logs from online games (World of Warcraft and Second Life). The Second Life corpus consists of 47 chat sessions with 38807 utterances (Small et. al. 2011). The World of Warcraft corpus consists of 5 chat sessions with 2041 utterances. The average vocabulary size used by a participant was 260 words. The average length per utterance was 6.32 words. Approximately 5% of the utterances contained one or more hedges.

In both Second Life and World of Warcraft, a “quest” was created and provided the players with a common objective that required a series of tasks and puzzles they had to try to solve together. The players were recruited from university students and the gaming community. The following snippet is from a World of Warcraft chat log containing a hedge in italics:

**Participant 1**: Yeah the tesla coil  
**Participant 2**: It fits all of the clues  
**Participant 3**: That *seems* most likely, I agree.
A subset of the chat logs were manually annotated for hedges to obtain a ground truth. Three annotators underwent training until they reached an 80% agreement (Krippendorff’s alpha) among each other and then the corpora were split into sessions and each session was annotated by a single annotator. The annotation identified the beginning and end word positions of each hedge within each utterance.

**Automatic Detection of Hedges in Game Chat**

We have designed a system that would automatically detect and mark hedges in the input text, thus obviating the need for manual annotation. The system is described below and it consists of the following steps:

1: Clean Up the Text

2: Tokenization

3: Representation

4: Comparison of Tokens

**Cleaning Up the Text**

We removed emoticons from the text before any processing for a few reasons. First, the emoticons were used in various parts of the sentences and it was unclear
whether they provided meaningful and consistent context. As an example, take the following sentences:

*How nice a day it is :) without that jackhammer making noise! :)*

The smiley face emoticon can be placed in multiple places in the sentence without it being conceived as weird or changing the meaning.

Second, spacing between emoticons and other words was inconsistent so sometimes it was difficult to separate an emoticon from a word since the emoticons often contain letters. Take the following for example:

**X-Domg I forgot to pick up the quest**

It is hard to determine if something is being misspelled or even where a word starts and stops.

Third, our annotators were not instructed to identify emoticons as hedges. This makes removing the emoticons reasonable, if the context they provide is inconsistent.

The emoticons were removed by using a lexicon provided by the social scientists, sorted by length descending, and then each emoticon was looked for in
an utterance. The reason for this is some emoticons contain other emoticons so the longer ones should be removed first.

**Tokenization**

After emoticons were removed, each utterance was tagged by Stanford’s Part of Speech (POS) Tagger, where each word was identified with a POS tag. A token was then created for each word with the previous two words (and their POS tags) as well as the subsequent two words (and their POS tags). The definition of a token in the context of this project is a collection of 10 strings representing a word and surrounding words and all of their POS values.

Example of a token created for the word “be” in the sentence:

I might be late for dinner tonight.

Output from Stanford’s Part of Speech Tagger:

I/PRP might/MD be/VB late/JJ for/IN dinner/NN tonight./NN

Token created for the word “be”:

<table>
<thead>
<tr>
<th>Word Value</th>
<th>Previous Word</th>
<th>2nd Previous Word</th>
<th>Following Word</th>
<th>2nd Following Word</th>
<th>Word POS</th>
<th>Previous POS</th>
<th>2nd Previous POS</th>
<th>Following POS</th>
<th>2nd Following POS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Be</td>
<td>might</td>
<td>I</td>
<td>late</td>
<td>for</td>
<td>VB</td>
<td>MD</td>
<td>PRP</td>
<td>JJ</td>
<td>IN</td>
</tr>
</tbody>
</table>

Using the human annotated training data we collected tokens that contained
hedges, that is where the central word was a hedge with the left and right context as shown above. Those tokens were organized into lists that contained all tokens sharing the same word value.

When a new token extracted from unmarked text is to be evaluated as a hedge or not, the word value is used as a key to retrieve the appropriate list of tokens from the hedge data. All tokens in that list are compared to the candidate token and the highest similarity score is used to evaluate the new token. When the score is high enough, the new token is classified as a hedge and incorporated into the hedge list.

**Representation**

A comparison function was devised to calculate similarity. A reward was added to the similarity score when one of the token’s features (word value, its POS tag, and its context) matched, and a penalty was subtracted when there was no match. In order to determine the optimal amount of these rewards and penalties, a genetic algorithm was used. The linear combination of these reward and penalty parameters is the basis of determining comparison between two tokens by establishing a similarity score. If the score was above a threshold of 0, then the token was found to be similar enough to one of the annotated hedge tokens and
marked as a hedge.

The similarity function can be represented by a series of integers representing the amounts of rewards and penalties to add to a token’s score when comparing two tokens. There are 10 strings to compare, so there are 10 rewards and 10 penalties that need to be determined. These 20 integers can be used by a tagger to compare new tokens with the database of annotated tokens. Hits, misses, and false alarms will be determined. Using these numbers, precision and recall can be calculated and then an f-score formed by the harmonic mean of precision and recall. This f-score is an evaluation of the performance of a set of rewards and penalties. The representation of the solution forms the genome of an individual in the genetic algorithm.
In the following example of an individual, each number represents a value to add to (reward) or subtract from (penalty) the similarity score when the string values match for each feature labeled in the column.

Rewards:

<table>
<thead>
<tr>
<th>Word Value</th>
<th>Previous Word</th>
<th>2nd Previous Word</th>
<th>Following Word</th>
<th>2nd Following Word</th>
<th>Word POS</th>
<th>Previous POS</th>
<th>2nd Previous POS</th>
<th>Following POS</th>
<th>2nd Following POS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>7</td>
<td>0</td>
<td>2</td>
<td>8</td>
<td>6</td>
</tr>
</tbody>
</table>

Penalties:

<table>
<thead>
<tr>
<th>Word Value</th>
<th>Previous Word</th>
<th>2nd Previous Word</th>
<th>Following Word</th>
<th>2nd Following Word</th>
<th>Word POS</th>
<th>Previous POS</th>
<th>2nd Previous POS</th>
<th>Following POS</th>
<th>2nd Following POS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

These values are stored in an individual as follows:

**Rewards:** 1 0 0 1 1 7 0 2 8 6

**Penalties:** 0 0 0 2 0 0 0 0 0 0
Comparison

When comparing two tokens, a string match between features adds the reward in that column while a mismatch subtracts the penalty in the same column. If the sum of the rewards and penalties exceeds a threshold (default is 0), then the token is judged to be a hedge.

Example of a comparison using an individual: token marked as hedge and stored in a list: (word is might)

I/PRP **might/**MD be/VB late/JJ for/IN dinner/NN tonight./NN

<table>
<thead>
<tr>
<th>Word Value</th>
<th>Previous Word</th>
<th>2nd Previous Word</th>
<th>Following Word</th>
<th>2nd Following Word</th>
<th>Word POS</th>
<th>Previous POS</th>
<th>2nd Previous POS</th>
<th>Following POS</th>
<th>2nd Following POS</th>
</tr>
</thead>
<tbody>
<tr>
<td>might</td>
<td>I</td>
<td>“”</td>
<td>be</td>
<td>late</td>
<td>MD</td>
<td>PRP</td>
<td>“”</td>
<td>VB</td>
<td>JJ</td>
</tr>
</tbody>
</table>

new token to compare: (word is might)

He/PRP said/VBD she/PRP **might/**MD be/VB wrong/JJ about/IN the/DT train/NN schedule./NN

<table>
<thead>
<tr>
<th>Word Value</th>
<th>Previous Word</th>
<th>2nd Previous Word</th>
<th>Following Word</th>
<th>2nd Following Word</th>
<th>Word POS</th>
<th>Previous POS</th>
<th>2nd Previous POS</th>
<th>Following POS</th>
<th>2nd Following POS</th>
</tr>
</thead>
<tbody>
<tr>
<td>might</td>
<td>she</td>
<td>said</td>
<td>be</td>
<td>wrong</td>
<td>MD</td>
<td>PRP</td>
<td>VB</td>
<td>VB</td>
<td>JJ</td>
</tr>
</tbody>
</table>

Calculation of similarity:
Individual being used:
Rewards: 1 0 0 1 1 7 0 2 8 6  
Penalties: 0 1 1 2 0 0 3 0 0 0

Main word “might” matches, so a reward of 1 is added to the similarity score.  
Rewards: 1 0 0 1 1 7 0 2 8 6  
Penalties: 0 1 1 2 0 0 3 0 0 0

The previous word does not match (“I” and “she”), so a penalty of 1 is subtracted for a total of 0.  
Rewards: 1 0 0 1 1 7 0 2 8 6  
Penalties: 0 1 1 2 0 0 3 0 0 0

The 2nd previous word does not match (“” and “said”), so a penalty of 1 is subtracted for a total of -1.  
Rewards: 1 0 0 1 1 7 0 2 8 6  
Penalties: 0 1 1 2 0 0 3 0 0 0

The subsequent word “be” matches so 1 more is added for a total of 0.  
Rewards: 1 0 0 1 1 7 0 2 8 6  
Penalties: 0 1 1 2 0 0 3 0 0 0

The main word’s Part of Speech “MD” is the same, so 7 is added; total of 7.  
Rewards: 1 0 0 1 1 7 0 2 8 6  
Penalties: 0 1 1 2 0 0 3 0 0 0

11
The previous word’s POS “PRP” matches, but the reward is 0; total stays 7.

**Rewards:** 1 0 0 1 1 7 0 2 8 6
**Penalties:** 0 1 1 2 0 0 3 0 0 0

The 2nd previous word’s POS (“” and “VB”) don’t match; total stays 7.

**Rewards:** 1 0 0 1 1 7 0 2 8 6
**Penalties:** 0 1 1 2 0 0 3 0 0 0

The subsequent word’s POS “VB” matches, so 2 is added for a total of 9.

**Rewards:** 1 0 0 1 1 7 0 2 8 6
**Penalties:** 0 1 1 2 0 0 3 0 0 0

The second subsequent word’s POS “JJ” matches; final total is 15.

**Rewards:** 1 0 0 1 1 7 0 2 8 6
**Penalties:** 0 1 1 2 0 0 3 0 0 0

With the default threshold being 0, the similarity score of 15 results in a positive classification as a hedge.
Applying a Genetic Algorithm to Find an Optimal Solution

When calculating similarity, the amounts to and and subtract from the score when comparing two tokens was unclear. In order to find optimal values, a search had to be conducted in a very large space representing possible combinations of rewards and penalties.

This task was very suited to a genetic algorithm. Genetic algorithms use a collection of potential solutions (called “individuals”) and are able to search high dimensional search spaces for multiple variables at the same time.

The two main components for the genetic algorithm are the representation and the fitness function. Representation is translating the problem into a series of parameters which can be manipulated. The fitness is how the solutions are evaluated, using the series of parameters as a solution. The fitness determines not only which solution is the best, but probabilities for solutions to be reused in the next generation as they improve.

The representation of a solution is as stated in the previous section. There are 20 integers representing the values used to reward and penalize tokens when they are being compared with one another. These integer values are stored together
in an individual, and many individuals are stored in a “population”.

The fitness is calculated by taking an individual being examined and using its set of values to evaluate a corpus. Each token, when compared, falls into one of the following categories: true positive, true negative, false positive, and false negative.

Precision and recall are calculated as follows:

\[
\text{Precision} = \frac{tp}{tp + fp}
\]

\[
\text{Recall} = \frac{tp}{tp + fn}
\]

\((tp = \text{true positive}, \; fp = \text{false positive}, \; fn = \text{false negative})\)

The F₁ score is then calculated by taking the harmonic mean of the precision and recall:

\[
\frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}}
\]

The F₁ score is used as the fitness value.
Genetic Algorithm Settings

A population of 5000 individuals were used, with each individual containing a genome comprised of 20 integers representing rewards and penalties when comparing tokens. The population was separated into sections that behaved differently.

After all individuals’ performance were evaluated, they were sorted by f-score descending. The top 1% formed an elite group, which means they were guaranteed to go to the next generation with their genome untouched.

The 25% of the population below the elite group in a new generation used a roulette wheel selection to select individuals to cross-breed and form new solutions for the next generation. All the f-scores were added (from the entire list) and then a random number between 0 and the total was chosen. Starting from the top, the scores were added until the random number was exceeded. That would determine the individual selected. This process was repeated as needed to find random individuals to breed.

The lower 75% of the population were also subjected to mutation. The chance and amount of mutation increased proportionally as lower ranks were reached. The mutation chance for every allele was 15% with the lowest scoring individual and the maximum amount the allele was altered was 10.
The population was initialized by setting all individuals’ alleles to 0 and then going through a round of mutation and crossover. This was the basis of “generation 0”.

Using these methods, it took approximately 20 generations for the population to converge, so 25 generations were used to confirm the convergence.

**Fitness Graph**

Graph showing the top fitness in the population (top) and the average fitness of the entire population (bottom). When both the top and average fitnesses converge on their own values, it is an indication that the top fitness is an optimal solution. The scale is from 0 to 0.65 here (y-axis) over 25 generations (x-axis).
Weighting and Pruning

Not all hedges occur equally, and the amount of annotation for each hedge varies. In order to take into account these differences, a weighting scheme was devised. The goal of these weights was to combine the frequency a token was found to be a hedge with the number of times a hedge was annotated. These weights were used as a default, and while they didn’t improve performance by a significant amount; they later became a tool by which we could fine tune the tagger for individual hedges. The default value was calculated as follows:

**frequency**: percentage of all occurrences a token is annotated a hedge

**inverted utterance frequency (IUF)**: \( \log(\text{utterance count}) / (\text{occurrence count}) \)

**weight**: frequency / (1 + max(IUF) - IUF)

Once results were generated, the social scientists went over the results and re-evaluated the hedges. A few were deemed unimportant because they were more often not a hedge or because they occurred so frequently that it was uncertain whether it was mentioned out of habit or if it was truly a hedge (such as “I think”). Overall agreement with annotators increased as a result and eventually a 90% agreement was reached; which then made the hedge data usable because 80% was
the cutoff for the social scientists’ standards. Note that this process of weighting and pruning occurred after the system was implemented and no further training was needed to increase the agreement by over 10%.

Performance Evaluation

Baseline used for measure of performance was a string match using a hedge lexicon. The baseline was calculated as follows:

<table>
<thead>
<tr>
<th>Tag</th>
<th>Actual</th>
<th>Hedge</th>
<th>Not Hedge</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hedge</td>
<td>1525</td>
<td>4647</td>
<td></td>
</tr>
<tr>
<td>Not Hedge</td>
<td>289</td>
<td>33049</td>
<td></td>
</tr>
</tbody>
</table>

Precision: 0.24708360337  
Recall: 0.840683572216  
f-score: 0.381851103

Each corpus was shuffled and split into 10 sections. Then using 9 of the sections to train and the last to test, 10 different performance scores were obtained. Second Life got a score of 0.746, World of Warcraft got a score of 0.790, and the combined corpora of Second Life and World of Warcraft got a score of 0.761.
10-Fold Cross Validation

<table>
<thead>
<tr>
<th>Second Life</th>
<th>World of Warcraft</th>
<th>Combined Corpora</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 - 0.857</td>
<td>1 - 0.801</td>
<td>1 - 0.709</td>
</tr>
<tr>
<td>2 - 0.749</td>
<td>2 - 0.772</td>
<td>2 - 0.784</td>
</tr>
<tr>
<td>3 - 0.818</td>
<td>3 - 0.814</td>
<td>3 - 0.778</td>
</tr>
<tr>
<td>4 - 0.8</td>
<td>4 - 0.740</td>
<td>4 - 0.809</td>
</tr>
<tr>
<td>5 - 0.9</td>
<td>5 - 0.782</td>
<td>5 - 0.785</td>
</tr>
<tr>
<td>6 - 0.774</td>
<td>6 - 0.729</td>
<td>6 - 0.808</td>
</tr>
<tr>
<td>7 - 0.631</td>
<td>7 - 0.809</td>
<td>7 - 0.750</td>
</tr>
<tr>
<td>8 - 0.533</td>
<td>8 - 0.822</td>
<td>8 - 0.689</td>
</tr>
<tr>
<td>9 - 0.736</td>
<td>9 - 0.783</td>
<td>9 - 0.766</td>
</tr>
<tr>
<td>10- 0.666</td>
<td>10- 0.762</td>
<td>10- 0.758</td>
</tr>
<tr>
<td>Avg = 0.746</td>
<td>Avg = 0.790</td>
<td>Avg = 0.761</td>
</tr>
</tbody>
</table>

The average of these scores were comparable to similar works using different methods in hedge detection. (Lin Chen, and Barbara Di Eugenio 2010 and Fernandes, Crestana, and Milidiú 2010.)
Conclusion

The main advantage to using this method is the ability to control the behavior of the tagger for individual hedges independent of other hedges and the training process. Because of this ability, hedges that bring down the performance can be controlled by themselves. In addition, no expertise is needed to change the weights if they are contained in an external file.

Another reason to consider this method is when the training occurs (genetic algorithm is run), many nearly-optimal solutions can be found in the population as well. The top solutions could be over-fitted to the corpus they were trained on, but some of the solutions further down the list may provide a more generalized solution. Adding a function that finds the minimal allele values required to achieve the same f-score can serve to show the relationship between various features and how indicative they are of being a hedge.

This method could potentially be used to detect other linguistic devices that can be represented as a series of rewards and penalties in a similarity function.
References


Fernandes, Crestana, and Milidiú 2010. Hedge Detection using the RelHunter Approach: International Conference on Computational Natural Language Learning
