Failing to Find Paraphrases Using PNrule

Benjamin Bartlett*
Department of Computer Science, University of Toronto
Toronto, Ontario, Canada M5S 3G4
bb@cs.toronto.edu

January 14, 2007

Abstract

In this paper, we attempt to detect clause-level paraphrases in cases where they are extremely rare, using a combination of lexical and syntactic measures along with a machine learning algorithm designed specifically for detecting rare classes: PNrule. When our method fails, we examine the probable causes of this failure, and what they mean for future work.

1 Introduction

A paraphrase is a restatement of a text, giving the same meaning as the original text in a different form. Recently, there has been a lot of work in computational linguistics towards developing systems that can automatically detect paraphrases. There are a number of ways in which such a system could be useful; some examples include: as part of an automatic summarization system, removing multiple instances of the same information; looking for the recurrence of an idea within a particular body of work; detecting talking points; or creating a dataset that can be used by a system that automatically learns how to generate paraphrases.

Most of the work up to this point has dealt primarily with paraphrases that occur within parallel or comparable corpora—situations in which paraphrases occur relatively frequently, and are often lexically similar to the text unit they paraphrase. Shinyama et al. (2002) use a combination of information extraction techniques and a similarity value based on named entities to find paraphrases in two news articles about the same event. Barzilay and Lee (2003) also look at news articles about the same event; they replace dates and proper nouns with generic tokens, then cluster sentences and induce word lattices that can be used to generate paraphrases. Barzilay and Elhadad (2003) first match similar paragraphs in comparable corpora, and then use sentence alignment between those paragraphs to find sentences containing clauses that paraphrase one another. Quirk et al. (2004) also use sentence alignment, and then use a phrasal decoder to generate new paraphrases.

*Current Address: Language Weaver, Marina Del Rey, CA 90292

It's also worth mentioning a related, but different, research area: that of entailment detection. Textual entailment is a directional relationship between two units of text: given two text units, t_1 and t_2, we say that t_1 *entails* t_2 if we can infer t_2 from the information given to us by t_1. This is relevant to us because, by the definition of entailment, if a text t_2 is a paraphrase of a text t_1, it must also be entailed by t_1.

Much of the recent work on entailment has come about due to the Recognizing Textual Entailment Challenge, put out by the PASCAL Network of Excellence (Dagan et al., 2005). This challenge consists of a dataset of 567 development examples and 800 test examples that has been designed to reflect the occurrence of entailment within several common NLP tasks. These tasks are: question answering, information extraction, machine translation, paraphrase acquisition, information retrieval, comparable documents (this was the authors' term; "sentence alignment of comparable documents" might be more accurate), and reading comprehension. An example consists of a text unit t and a hypothesis unit h; the text is one or two sentences (usually one), while the hypothesis is a shorter sentence. Systems submitted to the challenge must determine whether t entails h. This challenge is different from our goal not only in that it attempts to detect a different, albeit related, relationship, but also in that the examples are chosen so that there is a 50/50 split between positive and negative examples; thus rare classes do not come into play.

While these approaches have led to useful systems, we feel the next obvious step is to attempt to detect paraphrases in situations where they occur rarely, and are not necessarily lexically similar to the text unit they paraphrase. We attempt to do just that, using a machine learning technique in combination with a set of simple lexical and syntactic measures to detect clause-level paraphrases.

2 Motivation for this approach

2.1 Why machine learning?

There are several reasons we believe machine learning is a good approach to detecting paraphrases. First, there's the fact that machine-learning approaches have been used successfully to solve many other natural-language-processing problems. Màrquez (2000) lists many of these problems, including: sense discrimination, word sense disambiguation, text classification, speech recognition, part-of-speech tagging, text summarization, dialog act tagging, coreference resolution, cue phrase identification, machine translation, homograph disambiguation in speech synthesis, accent restoration, PP-attachment disambiguation, phonology, morphology, and spelling correction.

Second, and more importantly, machine learning has been used, with some success, to solve a very similar problem: Hatzivassiloglou et al. (1999) use it to match *similar* pairs of small textual units (in particular, paragraphs, although in theory their technique could be applied to sentences as well). They consider two textual units to be similar if they share the same focus on a common concept, actor, object, or action, and if that common actor or concept either performs or is the subject of the same action. This definition includes paraphrases, but it includes other sorts of relationships as well.

Now we can explain why machine learning *should* be used. Our reasoning is basically this: while ideally, we'd like to simply look at the semantics of two textual units (in our case, clauses) and decide whether or not they are the same, at this time, we simply do not have the resources to do so. Instead, we must rely on syntactic and lexical clues to decide whether or not one textual unit is a paraphrase of another. While some clues may be the same across the English language, others likely vary depending upon the corpus, both due to different styles of writing, and due to different contexts. In addition, even if a clue does exist across the entire language, it may be more important within some corpora than within others. Because we don't believe that it is currently possible to develop a system that can, out of the box, detect paraphrases in any corpus, we feel that it is best to develop a system that can be easily trained to deal with a new situation.

This leads us to our final point: simplicity. Machine-learning algorithms are much easier to adapt to new situations than are finely-tailored solutions. While the latter may be preferable in some cases, what we'd like to see is a machine-learning algorithm combined with a toolbox of syntactic and lexical features which could easily be trained on new corpora. This has the added benefit of allowing people to easily add new features to the toolbox (there are any number of features we have not tried yet, some of which could prove quite useful). There is, of course, one weakness to our approach: because it's a supervised method, we need annotated corpora, the production of which can be quite time consuming. Nevertheless, we firmly believe the benefits of our approach outweigh this weakness.

Why rule-based machine learning?

The main reason we feel that rule-based machine learning should be used for this task is that, as mentioned by Mitchell (1997) and Joshi (2002), a rule-based method is easily interpretable by humans. We believe this to be particularly important in the domain of paraphrase identification. Human beings have some intuition as to what attributes are likely to indicate the occurrence of paraphrases. Using a rule-based approach, they can see how well their intuition fits with what's discovered by the machine learning algorithm. This is advantageous for several reasons: first, we are more likely to catch errors in the software we use to obtain the attribute-value pairs, since such errors will cause situations where the results do not match our expectations. For example, if we find that the presence of matching trigrams between two sentences indicates that they are not paraphrases of one another, we know that we should take a second look at our trigram-matching algorithm. Second, in the case where the differences are not caused by an error, we may learn something new about the problem. Finally, because we can associate a record with the rule that classified it, we can look at false positives and discover both what attributes are indicating that these examples are positive, and, by looking at the sentences themselves, what additional attributes might allow us to distinguish between false and true positives. These are opportunities that less easily understood classification methods might not afford us.

A second reason for focusing on rule-based machine learning is that, compared to some other approaches, the parameters to the algorithm are relatively intuitive. One has a sense of what it means to have an upper bound on the length of a rule, or a lower bound on its accuracy. Compare this with, for example, support vector machines, where one has to decide between using a radial or an exponential kernel space. It's much more difficult to understand

what the latter "means" in terms of the problem one is trying to solve.

2.2 Why simple lexical and syntactic measures?

There are two reasons to try simple lexical and syntactic measures. First, they have been used with success in solving similar problems. Hatzivassiloglou et al. (1999) have used simple lexical and syntactic measures quite successfully in the system mentioned above. In addition, Barzilay and Elhadad (2003) have rather successfully detected paraphrases at the clausal level in comparable corpora using sentence alignment based on the cosine similarity of sentences and their containing paragraphs. While we anticipated that our task would be more difficult than theirs—they were able to obtain a precision of 57.9% at 55.8% recall just using a threshold on the cosine similarity between the sentences—nevertheless, our hypothesis was that paraphrases in our corpus would be similar enough to paraphrases in comparable corpora that this approach would work.

The other reason to try simple lexical and syntactic measure is the very fact they are simple. As a general rule, it is best to start with simple measures: if they work, then one avoids a lot of unnecessary work; if they fail, then at least one has learned something about the problem.

3 The problem with rare classes

Joshi (2002) lists three issues that arise in the context of a rare class that make learning difficult (note that we have simply borrowed Joshi's terms for these problems):

Low Separability Occasionally, one encounters a case where the data is noise-free and the examples can be classified using only one or two attributes. In this case, the data has a high degree of *separability*, and only a very simple classification model is needed. Usually, however, the records of class C are distributed in small clusters throughout various subspaces of the attribute space. This makes it difficult to find a large cluster containing records of class C that does not also contain a number of examples of class NC. This low degree of separability makes classification more difficult, and thus a more complicated model is required.

Multi-modality of the classes Related to separability is the fact that C and, in particular, NC may consist of different subclasses with different characteristics. It may be easy to separate C from some of those subclasses, but very difficult to separate it from others. We will see an example of this later in the chapter.

Rarity The rarity of the class is itself a problem. In particular, it's very difficult to avoid over-fitting the model to C: since the set of records in C is so much smaller than the set of records in NC, it's easy to find a model that fits those few positive examples exactly, but then doesn't generalize well. Thus, it performs well on the training data, but not on the testing data.

In addition to these problems, which arise regardless of the machine learning method used, rare classes pose two other problems to the most common method of rule induction, sequential covering (Agarwal and Joshi, 2000; Mitchell, 1997). In brief, sequential covering works as follows: there are two loops. In the inner loop, the algorithm begins with an empty rule, and then at each iteration tries all possible attribute tests, picks the one that most improves the performance of the rule over the training data, and adds it to the rule. It does this until the rule reaches an acceptably high level of accuracy. In the outer loop, the algorithm begins with an empty set of rules. Each time a new rule comes out of the inner loop, it removes from the training set all examples covered by the rule. The next rule is then learned over the remaining training set. This is repeated until the model has reached a satisfactory overall accuracy, or until there are no positive examples left. (Mitchell, 1997)

The two problems that arise when this method is used on rare classes are the *problem of small disjuncts* and the *problem of splintered positives*. The problem of small disjuncts was first identified by Holte et al. (1989). A small disjunct is one that correctly classifies only a few training cases. They are much more error-prone than large disjuncts (Holte et al., 1989). However, they collectively cover a significant percentage of examples, and thus cannot simply be ignored (Weiss, 1995). This is particularly true when dealing with rare classes, where even a few missed examples can lead to far lower recall for a particular class. Unfortunately, because there are few positive training examples, a rare class tends to create small disjuncts—because there aren't that many examples to begin with, a rule targeting a rare class probably will correctly classify only a few training cases, even if it is highly accurate. For example, if there are only 10 records in class A, out of perhaps a total of 1000 records, a rule R which predicts A and covers 4 of those examples and no other examples is highly accurate and covers a significant percentage of the examples of class A, but is still a small disjunct. To make matters worse, sequential covering removes some of those already-few examples at each iteration (Weiss, 1995; Joshi, 2002).

The second problem, that of splintered positives, was identified by Agarwal and Joshi (2000) and Joshi (2002). This problem arises when the signature for the target class is a composite of attributes that indicate the presence of the target class, and attributes that indicate the absence of the non-target class. Since we're dealing with paraphrases in this paper, let us consider an example in that domain: imagine that we have a number of pairs of sentences, most of which have nothing to do with one another. On the other hand, a few pairs of sentences will look like this:

a. *The elephant sitting on the dock was very large.*

b. *The elephant sitting on the dock was huge.*

It would thus seem that a high lexical similarity between a pair of sentences would be a good indication of the presence of our target class. However, remember that earlier we mentioned the multi-modality of classes. Although pairs of unrelated sentences make up a large subclass of the non-target class, there also could be a subclass that contains pairs of sentences with opposite meanings. This subclass might contain such pairs as:

a. *The horse next to the barn was very large.*

b. The horse next to the barn was very small.

Like the first pair, this pair would have a very high lexical similarity. However, clearly it is not a paraphrase. Instead, we need some other attribute, such as the presence of an antonym, to indicate the presence of the non-target class.

We should emphasize that the problem is not that an algorithm using sequential covering couldn't create a model that accounted for the presence of such an attribute; the problem is with how it would do so. Because of its high accuracy constraints, a sequential covering approach would refine its current rule, adding another conjunctive condition to it (in the case of our example, it might refine the rule "HighLexicalSimilarity $\Longrightarrow$ Paraphrase" to "HighLexicalSimilarity $\wedge$ $\neg$Antonym $\Longrightarrow$ Paraphrase"). However, if the current rule didn't cover very many negative examples, the algorithm might not correctly learn the signature of the non-target class—the additional condition might only apply to the few negative examples covered by the rule, instead of to the non-target class as a whole—and again we'd end up with generalization error (Joshi, 2002).

For prevalent classes, these two problems are not very likely to arise. During its early iterations, the algorithm is likely to discover rules that cover large numbers of positive examples, which means that the problem of small disjuncts will only arise in later iterations, when less-significant rules are discovered; these can simply be dropped. In addition, because the early-discovered rules would cover many positive examples, the rules which had to be refined to account for the presence of a non-target class would cover a large number of negative examples as well; if they did not, their accuracy would be high enough that they would not need to be refined. Thus the problem described in the previous paragraph would be avoided.

However, for rare classes, it is incredibly likely that both problems will occur. For one thing, there aren't very many positive examples to begin with, so the problem of small disjuncts will pop up very early, even in the most significant rules. For another, the signatures of rare classes tend to be very impure, and because of the high accuracy constraints, the sequential covering method is likely to create a large number of very detailed rules that cover very few examples.

Because in most situations paraphrases will be rare phenomena, sequential-covering rule-induction machine-learning algorithms are not very useful to us. Fortunately, Agarwal and Joshi have developed a rule-induction machine-learning algorithm called PNrule (Joshi, 2002; Agarwal and Joshi, 2000; Joshi et al., 2001) meant specifically to model cases where the target class is rare. Although they had in mind more systems-oriented uses such as data mining for network-intrusion detection (Joshi et al., 2001), we believe PNrule could be equally useful for cases within computational linguistics where one is trying to model rare phenomena, such as paraphrases. We will give a brief overview of PNrule below.

4 Detecting rare classes with PNrule

PNrule uses two innovations to deal with the problem of modeling rare classes: two-phase rule induction, and a scoring mechanism. Two-phase rule induction takes the place of sequential covering as the method by which the model is built, while the scoring mechanism is a post-processing step that adds additional flexibility to the model. We will discuss both of these

innovations, beginning with two-phase rule induction.

4.1 Two-Phase Rule Induction

Agarwal and Joshi's hypothesis is that sequential-covering algorithms have the problems mentioned above because they attempt to achieve both high recall and high precision at the same time. While this works well when the target class is prevalent, it does not work as well in situations where the target class is rare, for the reasons mentioned previously. Their solution is to model the class in two stages: in the first stage, they attempt to obtain high recall, and in the second, they attempt to obtain high precision (Joshi, 2002).

In order to understand these stages, it's important to understand two concepts: *rule-accuracy* and *support*. Let us say we have a rule R. A rule is a set of conditions on some set of attributes. If the attributes of a particular record meet the conditions of that rule, then we can say that the record satisfies that rule. Each rule predicts that the records that satisfy it will be of a particular class. Now, let there be some training set T, let $S \subseteq T$ be the set of records that satisfy R, and let $S' \subseteq S$ be the set of records in S that are from the class that R is trying to predict. Then the support of R is $|S|$, and the rule-accuracy of R is $|S'|/|S|$. In other words, support is the number of records that a rule covers, whereas rule-accuracy is the percentage of the records covered by a rule that are in the class that rule predicts (Joshi, 2002).

Now we can explain the two stages. The first stage is called the P-stage, and the rules discovered within it are called P-rules. In some ways, it works much the same way as sequential covering: at each iteration, a rule is formed, and the records covered by that rule are then removed from consideration. However, instead of simply using a high-accuracy constraint, P-rules are discovered using a fitness function that balances accuracy with support. This leaves us with more false positives (and thus lower accuracy) than in a sequential-covering algorithm, but without rules that cover only a tiny number of examples. The idea is that in this stage, the algorithm should maximize recall at the expense of precision.

The second stage is called the N-stage, and the rules discovered within it are called N-rules. This stage operates on the union of the sets of records covered by the P-rules. The point of this stage is to learn rules that will remove some of the false positives from the last stage; that is, the N-rules predict the absence of the target class. The problem is that these rules can also remove true positives; Agarwal and Joshi call this *introducing false negatives* (Joshi, 2002; Agarwal and Joshi, 2000). Aside from the fact that the training data is now the records covered by the P-rules, and that the target class is now in effect the non-target class, the N-stage works basically the same way the P-stage did. The idea is that in this stage, the algorithm should attempt to maximize precision.

We should take a moment to describe what, precisely, a rule looks like in PNrule. PNrule allows for two types of attributes: categorical and continuous. Given a categorical attribute A_{cat} and a value v, PNrule allows for two conditions: $A_{cat} = v$ and $A_{cat} \neq v$. For a continuous attribute A_{cont} and two values, vl and vr, where $vl \leq vr$, PNrule allows for three conditions: $A_{cont} > vl$, $A_{cont} \leq vr$, and $vl < A_{cont} \leq vr$. A rule is a conjunction of these conditions.

Each stage, then, can be thought of as an outer and inner loop. The inner loop finds

the best rule it can, according to some evaluation measure and input parameters. The outer loop looks at the rule produced by the inner loop, and, again according to certain parameters, decides whether or not to add it to the set of rules in the model. In other words, the inner loops refine the rules, while the outer loops add new rules; it's similar to running two sequential covering algorithms, with the second algorithm being fed as input the set of data covered by the rules output from the first algorithm.

Each of the four loops has its own stopping criteria. The P-stage rule refinement loop stops when further refinement of the rule would drop the support of that rule below an acceptable level, or when adding a new condition to the current rule will not yield an improvement according to some evaluation metric. The N-stage rule refinement loop stops when the model, including the current rule, has a high-enough recall, and when adding a new condition to the current rule will not yield an improvement according to some evaluation metric. Note that the stopping criteria for P-stage rule refinement form a disjunction—either no further improvements can be made, or to do so would drop support below an acceptable level—while those for N-stage rule refinement form a conjunction—the model's recall has reached an acceptable level *and* no further improvements to the rule can be made.

Like the N-stage rule-refinement loop, the stopping criteria for the P-stage rule-adding loop form a conjunction. First, the current set of rules must cover a minimum number of examples. Once this is satisfied, the accuracy of the rule is checked: if it is above a certain threshold, it is added to the set of P-rules; if it is below that threshold, it is discarded and the P-stage ends. The N-stage rule-adding loop has only one stopping condition. It checks to see if adding a new N-rule would increase the minimum description length of the model—the number of bits it would take to encode the model, plus the number of bits it would take to encode the errors made by the model—by more than some threshold. If it would, the rule is discarded and the N-stage is ended.

All of the thresholds that are used in the stopping conditions are parameters set by the user. Through some experimentation we set them to what seemed to be reasonable values.

Evaluation metric

As mentioned above, rule refinement in both stages relies on some evaluation metric to determine whether a refinement is an improvement over a current rule or not. According to Joshi (2002), this metric should capture three things: the ability of a rule to distinguish members of the target class from those records not in the target class, the support of that rule, and the accuracy of that rule. Naturally, a rule with high support and high accuracy should be given a high score by the metric. Joshi notes that there are several possible metrics, including Gini index, information gain, gain-ratio, and chi-square statistics. However, the one that Agarwal and Joshi define, and that we implement, is called Z-number, and is based on the z-test in statistics.

Let there be some rule R, with accuracy a_R and support s_R. Let S be the current training data (which, as mentioned earlier, changes over each iteration) and let $S_C \subseteq S$ be the examples in S that are in the target class. Then $a_C = \frac{|S_C|}{|S|}$ is the mean of the target class. Since this is a binary problem, $\sigma_C = \sqrt{a_C(1 - a_C)}$ gives us the standard deviation of the target class.

Using this, Z-number is:

$$Z_R = \frac{\sqrt{s_R}(a_R - a_C)}{\sigma_C}$$

There are two things going on in this measure. First, it's measuring the number of standard deviations the mean of the rule is away from the mean of the target class. A large positive Z-number indicates that the rule predicts the presence of the target class with high confidence, while a large negative Z-number indicates the rule predicts the absence of the target class with high confidence. Second, the Z-number is weighted by the square root of the rule's support. This gives preference to rules with high support, and also allows for a trade-off between accuracy and support.

4.2 Scoring mechanism

The second important innovation in PNrule is the scoring mechanism. Without the scoring mechanism, the model would simply predict that a record is in the target class if it satisfies some P-rule while satisfying none of the N-rules. However, recall that the N-rules were trained on the union of the examples covered by the P-rules. Thus, while we have a good sense of the effect an N-rule has on the overall number of false positives, we do not have any sense on how well it works towards removing the false positives caused by a particular P-rule. Let's assume we have a P-rule and two N-rules. One N-rule does an excellent job of removing the types of false positives the P-rule causes; the other does not. It stands to reason that a record that is covered by the P-rule and the first N-rule is less likely to be in the target class than one that is covered by the P-rule and the second N-rule.

Furthermore, it's possible that for a particular P-rule, a particular N-rule introduces a large number of false negatives. If one could give such a P-rule/N-rule combination a low score, one could recover those false negatives.

Thus, Agarwal and Joshi develop a scoring mechanism that estimates the posterior probability of a record belonging to the target class given the particular P-rule and the particular N-rule it satisfies, and assigns a score to each record accordingly. The algorithm then determines a threshold th, where a record is in the target class if its score is $\geq th$, that maximizes F_1.

Due to space constraints, we won't go into detail about the scoring method. In brief, for each P-rule/N-rule pair (P_i, N_j), a score is assigned, which is the Laplacian-smoothed accuracy of the pair according to the examples it covers in the test set. However, if (P_i, N_j) covers too few examples, or if its accuracy does not differ significantly from that of (P_i, N_{j-1}), then it is given the same score as (P_i, N_{j-1}). In addition, each P-rule is assigned a score in the case that no N-rules apply. The scoring mechanism then goes through the training set, testing each P-rule, and, for those examples covered by a P-rule, each N-rule, in order, and assigns each example a score accordingly. It then uses the labeled examples to find the threshold that gives the highest F_1 score. The minimum coverage required and the minimum amount of difference between accuracy scores are both parameters set by the user; again, we set these through experimentation to some reasonable values.

4.3 Changes to PNrule

During development, we noticed that, despite its focus on rare cases, PNrule was vastly overfitting its model to the training data: during the rule-refinement stage, it would often discover rules that covered only a single positive example. The problem was that, despite their low support, these rules would often have a higher Z-number than the alternatives. We also noticed that these rules usually consisted of a single range condition. In order to combat this overfitting problem, we made two changes to PNrule.

First, we removed PNrule's ability to treat a range as a single condition of a rule. You'll recall that in its original formation, PNrule was capable of discovering conditions such as $0 \leq v_1 < 1$, where v_1 is some attribute. However, we noticed that this allowed PNrule to discover highly accurate rules by using range conditions to cover a very small number of examples. It was our hypothesis that the problem was that, because ranges were treated as a single condition, PNrule was examining these before it could find potentially better combinations of conditions. Note that we did *not* remove the ability of PNrule to discover ranges—only that now a range is treated as two conditions of a rule instead of a single condition (i.e., $0 \leq v_1 < 1$ becomes $v_1 \geq 0 \wedge v_1 < 1$). Thus, PNrule compares a given range to all other possible pairs of conditions, instead of comparing it to unary conditions.

Second, we added Laplacian smoothing during the calculation of the Z-number. The advantage of using Laplacian smoothing in this manner is that it significantly lessens the accuracy value assigned to rules with low coverage without significantly affecting the accuracy value assigned to rules with high coverage. For instance, if a rule covered one positive example and no negative examples, its accuracy value would be reduced from 1.0 to $\frac{1+1}{1+2} = 0.66$, whereas if a rule covered 100 positive examples and no negative examples, its accuracy value would only be reduced to $\frac{100+1}{100+2} = 0.99$.

Although, as we will see in the results, PNrule was still unable to find a particularly good model, it did begin to make reasonable choices given the training data, and ceased to drastically overfit the data. Note that this reflects one of the great advantages of rule-based machine learning: because we were able to understand the model that PNrule was outputting, we were able to make adjustments to improve its performance.

In addition to these changes, we also created a second version of PNrule, wherein we substituted a new measure, *signature clarity*, for the Z-number during the rule-building stage, and changed the stopping condition for rule building during both the P-stage and the N-stage to a less than 0.05 increase in the clarity value of the rule. We call this measure signature clarity, because it reflects the amount of noise added to the class signature to create the hypothesis signature. A clear signature (which would have a value of 1 with this measure) would be one that 1) covered all of the target class examples, and 2) covered none of the non-target class examples. Thus, we felt our measure should reflect two things: 1) how strongly the presence of the target class indicates the presence of the signature, and 2) to what degree the signature has been polluted by the signature of the non-target class. We refer to 1) as the signature's *strength*, and to 2) as the signature's *purity*. Recall is an excellent measure of the former. But how do we measure the latter? It seems to us that this should be determined by how well the signature covers the target class, versus how well it covers the non-target class.

Thus, we arrive at the following measure for purity:

$$P_S = \frac{R_C}{R_C + R_{NC}}$$

where R_C is the recall of the target class, and R_{NC} is the recall of the non-target class. A signature's clarity is the product of its purity and its recall; thus

$$Cl_S = R_C \cdot \frac{R_C}{R_C + R_{NC}} = \frac{R_C^2}{R_C + R_{NC}}$$

Since in effect a model is simply a hypothesis signature, we can measure the signature clarity of a model.

Signature clarity differs from F_1 in that it measures how well a model identifies a target class, but is unaffected by the relative sizes of the target and non-target classes. For example, in a data set where there are 99 positive examples and 1 negative example, simply picking the null hypothesis and covering all of the examples gives a very high F_1 measure, while if there are 99 negative examples and 1 positive example, the null hypothesis gives a very low F_1 measure. In contrast, the signature clarity of the null hypothesis in each case would be 0.5, because in each case it perfectly covers both classes.

Signature clarity has a property that makes it particularly useful for PNrule. When the target class is significantly smaller than the non-target class, the purity of the rule quickly approaches 1. Thus, in order to obtain a high clarity in this situation, an algorithm will focus on maximizing recall. By contrast, if the target class is significantly larger than the non-target class, covering only a few of the non-target examples will cause the purity value—and thus the clarity value—to be very low; in such a situation, an algorithm will focus more on precision. This is precisely the behavior we'd like to see in PNrule. Note, however, that while a large disparity between the sizes of the target and non-target class can cause the purity value to approach 1 very quickly, the same is *not* true for recall. Thus, at the very least, PNrule must always balance recall with precision, while in some cases it can nearly ignore the latter in favor of the former. This asymmetry is important; if it were possible to nearly ignore recall, we'd end up with a model containing a large number of rules, each of which would cover only a small number of examples; this would mean the model was overfitting the data.

5 Our corpus

For our corpus, we used a set of reviews from Epinions.com (`www.epinions.com`). In particular, we used reviews of the movie *Spider-Man 2*. While we used all 99 reviews available for latent semantic indexing (which we'll get to later), we used five of these articles to create our training and testing sets. First, we paired the articles with each other, giving us a total of 10 pairs of articles, with a total of 74,204 pairs of sentences. Then, we had two judges annotate each article pair, marking those sentence pairs that had a clause-paraphrase relationship. Originally, the kappa agreement was only 0.388. According to Landis and Koch (1977), this indicates only fair agreement. However, due to the annotation method we used, it was entirely possible that some of the "disagreement" was actually caused by one annotator spotting a paraphrase that the other had missed. Because paraphrases are very rare in our corpus, it would both be easy

to overlook them, and would severely affect the kappa score were a few to be overlooked. To distinguish between actual disagreement and overlooked paraphrases, we had the annotators take a second look at the pairs upon which they disagreed. The new labels resulted in a kappa value is 0.792, which indicates substantial agreement. This indicates that much of the original disagreement was indeed due to overlooked paraphrases, as opposed to actual disagreement over what constitutes a paraphrase.

Discarding the examples the annotators disagreed on, this gave us a total of 90 positive examples and 74,067 negative examples: we are indeed dealing with a very rare class.

The rareness of this class made determining our testing and training sets somewhat difficult. We wanted to keep the article pairs either entirely in the training set, or entirely in the testing set; this way, it would be as though the system were seeing entirely new pairs of articles during training.[1] There were two reasons for this: first, we use information about the paragraphs in which the sentences are embedded in our system, and thus it's more realistic to assume the system is looking at pairs of articles; second, and more importantly, it may be that several types of paraphrasing occur within an article, and thus, we want a sample that would include all such types. In theory, were we simply to treat all of the pairs as one big bag, it might be possible that we'd only end up with one type of paraphrasing in our test set—particularly since there are so few positive examples.

However, while using entire articles, we still wanted to make sure we ended up with something close to 80% of the positive examples in the training set, and 20% of the positive examples in the testing set. This was potentially difficult, as the article pairs ranged from having 23 sentence pairs sharing a clause-paraphrase relationships, to having only 1 sentence pair sharing such a relationship. We managed to find two pairs of articles, one with 7 positive examples, and the other with 12, and we used these two pairs for our testing set. This gave us a total of 71 positive examples in the training set, and 19 in the testing set, which means we ended up with approximately 79% of the positive examples in the training set, and 21% of the positive examples in the testing set.

Preprocessing

Our preprocessing consists of the following steps:

1. Expanding contractions
2. Part-of-speech tagging
3. Removing auxiliary verbs
4. Removing certain parts of speech

We use a small set of heuristics to expand contractions back into their component words. For instance, we replace *can't* with *can not*, and *won't* with *will not*. We do this so that phrases

[1] Although our system would've seen both of these articles before individually, we don't see this as an issue, as our method looks at each pair of sentences individually, without retaining any information about a particular article. Thus, whether the system has trained on a particular article is not an issue; it's whether the system has trained on a particular *pair* of articles that matters.

such as *can't go* will match with phrases such as *can not go*.

In order to tag our data, we use the rule-based part-of-speech tagger developed by Eric Brill (Brill, 1994), and trained on the Brown corpus. Although it would've been preferable to retrain the tagger on data more similar to our own, we simply did not have a large enough tagged corpus available to do so. We examined our data after it had been tagged, however, and the Brill tagger seemed to do a reasonable job; its most common mistake was to mis-tag adjectives as nouns.

Because it's possible for auxiliary verbs to cause a pair of sentences to have a higher similarity measure than that pair ought to have—for example, the pair of sentences *I am running* and *I am dancing* will have a higher similarity measure than the pair of sentences *I dance* and *I run*—we want to remove any auxiliary verbs from our corpus. We do this as best we can by removing any form of the verbs *to have* or *to be* that is followed by another verb. The reason we don't simply remove all *to have* or *to be* verbs is because in cases where they are not auxiliary, they are potentially important indicators of paraphrasing.

In many applications similar to this one, function words are removed from the corpus. However, we found that in many lists of function words, words that were useful to our measures (such as forms of the verb *to be*) were included. Thus, instead of removing function words, we decided to try to remove words with certain parts of speech that seemed to indicate that they did not carry much semantic meaning. With that in mind, we remove words with the following parts of speech from the tagged versions of our corpus: determiner (*all*, *an*, etc.), preposition or subordinating conjunction (*astride*, *among*, etc.), modal auxiliary (*can*, *cannot*, etc.), pre-determiner (*all*, *both*, etc.), and interjection (*golly*, *gosh*, *jeepers*, etc.).

6 Measures

There were two overarching, and conflicting, concerns when we were developing our measures. The first was that for any measure we might use that relies on matching lexical units, two long sentences are much more likely to have a number of matches, regardless of what we are trying to match, than are two shorter sentences. Thus, it would appear that regularization is key. However, we are interested in matching *clauses*, not sentences. So, for instance, take two pairs of sentences:

1. a. *The boy jumped the fence.*
   b. *The boy hopped the barrier.*
2. a. *Susan walked to the store, and the boy jumped the fence.*
   b. *As impossible as it may seem, the boy hopped the barrier.*

Both of these should be positive examples, as they both contain clauses with a paraphrase relationship. However, due to regularization, the latter pair would score lower than the former.

We thus use a series of measures, of two types: continuous and binary. We try to combine measures that look at the whole sentence with others that we hope will help to pinpoint clauses with a paraphrase relationship.

Method	Recall	Precision	F_1
Cosine Threshold	0.042	0.1	0.059
PNrule	0.211	0.268	0.236
PNrule (clarity)	0.253	0.122	0.165
PNrule (3 measures)	0.127	0.130	0.129

Table 1: Precision, recall, and F_1 of the models learned on the training data.

The continuous measures we tried included: matching word stems; matching hypernyms using WordNet; matching synsets using WordNet; again matching hypernyms and synsets, but matching verbs based on their Levin classes from VerbNet; matching bigrams and trigrams of word stems, hypernyms, synsets, and Levin classes; matching bigrams but allowing every second word to be skipped in the creation of those bigrams (skip-one bigrams); matching word stems, hypernyms, synsets, and Levin classes within windows of 4–7 words; latent semantic indexing; cosine similarity of the paragraphs containing the sentences; and latent semantic indexing of the same. All continuous measures except for LSI and cosine similarity were regularized using a method taken from Hatzivassiloglou et al. (2001): given two textual units, A and B, where A contains $|A|$ features and B contains $|B|$ features, we divide these measures by $\sqrt{|A| \cdot |B|}$.

Our binary measures primarily check to see if the two sentences share some combination of lexical and syntactic features. These features take the form of pairs of words with different parts of speech, within a range of 1–5 words of one another. These pairs include: noun-verb, verb-noun, verb-adverb, be-adj, noun-noun, and noun-adjective. The noun-verb, verb-noun, and be-adj pairs must appear in that order; the others are order-independent. In addition, we look for cases where a word from each sentence has the same stem, but a different part of speech from the other word. The idea is to look for transformations from one part of speech to another. The matches we attempt are noun-to-verb, noun-to-adjective, verb-to-adjective, and adjective-to-verb.

7 Results

First, we had to create a baseline against which our results could be compared. Since (Barzilay and Elhadad, 2003) were also looking for sentences sharing clause-paraphrase relationships, we decided that, like them, we would use a cosine similarity threshold to create our baseline. To do so, we found the cosine similarity of all of our examples, then took the threshold that gave us the best F_1 value on the training data and applied it to the testing data. The results can be seen in Figure 1 (for the training data) and Figure 2 (for the testing data). We then collected the measures listed in the previous section and ran PNrule on them.

As we can see from Table 1, the model found by PNrule over our lexical measures was predicted to do better than the model developed using a cosine threshold. It's worth discussing the model that PNrule came up with, as it had some interesting qualities. An example is potentially labeled as positive if one of two P-rules apply to it, and none of the N-rules do.

Method	Recall	Precision	F_1
Cosine Threshold	0	0	0
PNrule	0	0	0
PNrule (clarity)	0.053	0.045	0.049
PNrule (3 measures)	0.210	0.235	0.222

Table 2: Precision, recall, and F_1 of the models applied to the testing data.

The two P-rules were:

1. The skip-one-bigram word-stem-matching measure is greater than 0.115 and the latent-semantic-indexing measure is greater than 0.586.

2. The latent-semantic-indexing measure is greater than 0.949.

There were other P-rules, but none of them had a confidence score over the threshold, so in effect we can ignore them. The N-rules that could apply to the two P-rules are listed in Figure 1. The N-rules are complicated and do not show any clear pattern, which demonstrates that there isn't any obvious way to divide the true positives from the false positives.

The confidence levels for rules covered by one of the two P-rules, and by none of the N-rules, were 0.421 and 0.428, respectively. This is clearly what gives us our precision: if one of these two rules covers an example, there's a nearly 50% chance that the example will be positive (not bad, given the difficulty of our problem). Also interesting to note: even when covered by an N-rule, an example covered by the first P-rule would still have a confidence level of 0.4; by contrast, one covered by the second P-rule could have a confidence level of as low as 0.182. This demonstrates that a combination of measures is (perhaps unsurprisingly) likely to lead to a better result than use of a single measure.

Unfortunately, in the training set, P-rule 1 covered only 7 positive examples, and P-rule 2 covered only 4—even worse, as we can see in Table 2, they covered no positive examples in the testing data, which means that our method did no better than cosine threshold. Thus while we have a couple of rules that appear to be reasonable indicators, relatively speaking, of a clause-paraphrase relationship, they cover far too few examples to be useful in and of themselves.

We include the results of our clarity-measure-based version of PNrule as well. As we can see, it unfortunately doesn't perform any better than PNrule. However, the model it arrived at was far simpler. This model consisted of a single one-condition P-rule, and two one-condition N-rules. In fact, the entire model can be summed up in one line:

$$\texttt{F\_HYP\_PoS} > 0.137 \wedge \neg(\texttt{LSI} \leq 0.531) \wedge \neg(\texttt{F\_HYP\_WIN}_5 \leq 0.5)$$

where F_HYP_PoS is the regularized most-frequent hypernym count, where the hypernyms must have the tagged part-of-speech; LSI is the latent semantic indexing measure; and $\texttt{F\_HYP\_WIN}_5$ is the regularized most-frequent hypernym count taken within a 5-word window. The P-rule covers 63 out of 90 of the positive examples, giving it a fairly high recall, but it also covers

1. $\texttt{F\_HYP\_PoS} \leq 0.137 \wedge \texttt{SYN\_PoS} \leq 0.213 \wedge \texttt{MATCH\_N\_ADJ} = false \wedge \texttt{ORDERED\_N\_V} = false \wedge \texttt{WORD} \leq 0.133 \wedge \texttt{PCOS} \leq 0.181$
2. $\texttt{SYN} \leq 0.26 \wedge \texttt{WORD\_WIN}_4 \leq 0.25 \wedge \texttt{HYP\_POS} > 0.146 \wedge \texttt{F\_HYP} > 0.118 \wedge \texttt{LSI} \leq 0.789 \wedge \texttt{PCOS} > 0$
3. $\texttt{F\_HYP\_PoS} \leq 0.166 \wedge \texttt{F\_SYN\_WIN}_7 \leq 0.286 \wedge \texttt{RANGED}_3\texttt{\_BE\_ADJ} = false \wedge \texttt{SYN\_PoS\_WIN}_7 \leq 0.429 \wedge \texttt{F\_HYP} \leq 0.218 \wedge \texttt{F\_SYN\_PoS} \leq 0.144$
4. $\texttt{PCOS} \leq 0.081 \wedge \texttt{ORDERED\_N\_V} = false \wedge \texttt{LSI} \leq 0.991 \wedge \texttt{HYP} > 0.171$
5. $\texttt{BISKIP\_SYN} \leq 0.02 \wedge \texttt{PCOS\_NOPROP} > 0.072 \wedge \texttt{ORDERED\_RANGED}_5\texttt{\_N\_V} = false \wedge \texttt{F\_HYP\_PoS} > 0.141 \wedge \texttt{PCOS} \leq 0.225$
6. $\texttt{PCOS} \leq 0.122 \wedge \texttt{HYP} \leq 0.476 \wedge \texttt{ORDERED\_RANGED}_5\texttt{\_V\_N} = false \wedge \texttt{SYN\_PoS} > 0.178 \wedge \texttt{PCOS\_NOPROP} > 0 \wedge \texttt{HYP\_PoS} \leq 0.6 \wedge \texttt{PCOS} > 0.056$

Figure 1: The N-rules discovered by PNrule over our measures. SYN is the regularized synonym count, HYP is the regularized hypernym count, and WORD is the regularized word count. If preceded by F_, only the most-frequent synonyms or hypernyms are matched. If followed by _PoS, only synonyms or hypernyms with the tagged part of speech are matched. If followed by $\_\texttt{WIN}_\#$, the count was obtained within a window of size #. If preceded by BI_, TRI_, or BISKIP_, the measure is of regularized bigrams, trigrams, or skip-one bigrams, respectively. PCOS is the cosine similarity of the containing paragraphs, PCOS_NOPROP is the same measure without proper nouns, and LSI is the latent semantic indexing measure. MATCH_, ORDERED_, and $\texttt{RANGED}_\#$ followed by two parts of speech indicate our stem-matching, ordered-matching, and ranged-matching measures, along with the parts-of-speech that have to be matched.

15674 negative examples, giving it a terrible precision. The N-rules remove 24 and 12 positive examples, respectively, but also remove 14248 and 1292 negative examples. This gives us 18 out of 90 positive examples, and 134 negative examples, which, while not a stunning result, is the sort of behavior we'd want to see in PNrule. In particular, the use of the LSI measure, which in our data has low coverage but is highly precise, as an N-rule instead of a P-rule makes considerably more sense. Thus, while the performance of this version of PNrule on our data is not ultimately any better than the performance of the original version, it does leave us with a much simpler model, and behaves in a manner closer to what we'd expect in PNrule.

As an experiment for further analysis, we decided to try running PNrule with small subsets of the measures. Of these, the best performance on the testing data was obtained by training on three measures: the cosine similarity of the containing paragraphs (ignoring proper nouns), the noun-noun ranged measure with a range of 3, and noun-adjective ranged measure with a range of 4. We label this PNrule (3 measures) in Figure 1 and Figure 2. As we can see, despite doing better on the testing data, it's still not a strong result—in fact, because it does no better on the training data, it's likely that it does better on the testing data due purely to chance.

8 Results on RTE dataset

Because its goal was similar to ours, we decided to also try our method on the first PASCAL RTE challenge dataset. In order to do so, we dropped all paragraph similarity measures (since the data didn't contain paragraphs), retrained the LSI model on the Reuters-21578 corpus

(Lewis, 1999) (since we're now dealing with news articles, not movie reviews), and added a *task* category to our measures. In addition, we set the minimum recall for the N-stage to 0.02—since we're not dealing with a rare class, there's no real need to worry about minimum recall. For the purpose of comparison, we've included the results of the systems submitted to the challenge, along with our own results, in Table 3.

As we can see, our results are in line with those of the other systems, with the comparable documents task causing most of the overall accuracy improvement. The F_1 score we obtained was 0.67, a basically insignificant improvement over simply labeling every example as positive, which obtains a score of 0.666. We don't know precisely what the F_1 scores of the other systems were, but we do know from Dagan et al. (2005) that none of them do significantly better than this.

9 Analysis

Now that we have discussed our results, we are left with two questions: how reliable are these results, and, if they are reliable, why didn't our approach work?

We anticipate that one criticism of these results might be the small number of positive examples in our training and testing data. While it's true that the number of positive examples is small, given the proportion of positive to negative examples, we'd need a very clear signature to achieve reasonable results—clear enough that it should be obvious even given those few positive examples. Our method is clearly unable to find such a clear signature.

The question that remains, then, is why was our method unable to find a clear signature? Two possibilities exist, and the answer may in fact be a combination of these: our measures are inadequate, or rule-based models are inadequate.

Barzilay and Elhadad (2003) demonstrate that when using comparable corpora, it is possible to classify such sentence pairs with a good deal of success, using simple lexical measures. Assuming, as we did, that the paraphrases within our corpus were like those found in comparable corpora, the problem is that lexical measures at best provide only an approximation of semantic measures. In cases where the two classes are close in size, a reasonably good approximation will suffice. However, in rare cases, the approximation must be nearly perfect, because, as we discussed earlier, even a small amount of noise can lead to a large amount of error. It is probably not a coincidence that the most accurate rules were those using the LSI measure—of all our measures, this comes closest to approximating the underlying semantics of the sentences. Unfortunately, those rules do not have a particularly high recall.

All of that said, our results both on our own corpus and on the RTE dataset, in combination with the results of the other systems, suggest two things: first, that the paraphrases found in our corpus are *not* necessarily of the same sort found in comparable corpora, and second, that simple lexical measures are not sufficient when dealing with paraphrases outside of the comparable documents domain. To see why this is so, let's look at a few examples. First, here's an example of the sort of paraphrase we anticipated would be prevalent:

a. *The red dog ran across the street.*

System	Coverage	Accuracy							
		By Task							Overall
		IR	CD	RC	QA	IE	MT	PP	
Akhmatova	100%	0.511	0.587	0.521	0.477	0.508	0.492	0.520	0.519
Andreevskaia, et al. (thresh=1)	100%	0.52	0.64	0.51	0.45	0.51	0.47	0.50	0.52
Andreevskaia, et al. (thresh=3)	100%	0.53	0.63	0.48	0.45	0.52	0.47	0.50	0.52
Bayer, et al. (system 1)	73%								0.516
Bayer, et al. (system 2)	100%								0.586
Bos, et al. (system 1)	100%								0.555
Bos, et al. (system 2)	100%								0.563
Braz, et al.	100%	0.522	0.773	0.514	0.500	0.500	0.533	0.500	0.561
Delmonte, et al.	100%	0.622	0.687	0.521	0.585	0.583	0.467	0.800	0.593
Fowler, et al.	100%	0.478	0.780	0.514	0.485	0.483	0.542	0.450	0.551
Glickman, et al.	100%	0.500	0.833	0.529	0.492	0.558	0.567	0.520	0.586
Herrera, et al.	100%	≤ 0.558	0.787	≤ 0.558	≤ 0.558	≤ 0.558	≤ 0.558	≤ 0.558	0.548
Jijkoun, et al.	100%	0.533	0.847	0.493	0.423	0.550	0.467	0.420	0.553
Newman, et al. (w/ task)	100%	0.446	0.747	0.571	0.515	0.558	0.475	0.520	0.563
Newman, et al. (w/out task)	100%	0.544	0.740	0.529	0.539	0.492	0.508	0.560	0.565
Pazienza, et al. (manual params)	100%	0.444	0.765	0.486	0.395	0.467	0.521	0.540	0.525
Pazienza, et al. (SVM-tuned params)	100%	0.489	0.644	0.521	0.457	0.492	0.479	0.500	0.518
Raina, et al. (single weight)	100%	0.567	0.793	0.529	0.485	0.475	0.467	0.580	0.562
Raina, et al. (multiple weights)	100%	0.556	0.840	0.507	0.439	0.550	0.475	0.540	0.552
Wu (w/ stoplist)	100%	0.478	0.700	0.450	0.446	0.517	0.392	0.520	0.505
Wu (w/out stoplist)	100%	0.467	0.713	0.471	0.408	0.550	0.400	0.560	0.513
PNrule	**100%**	**0.511**	**0.793**	**0.507**	**0.500**	**0.500**	**0.533**	**0.500**	**0.563**

Table 3: The performance of the various systems submitted for the PASCAL RTE challenge, plus PNrule's performance, included in bold. Where results were unavailable, we have left the cells blank.

b. *The dog ran across the street.*

As we can see, the only difference is that an adjective, *red*, has been dropped; this does not fundamentally change the meaning of the sentence. However, let us look at another example:

a. *Osaka is the gastronomic capital of Japan.*

b. *Osaka is the capital of Japan.*

Like the first example, the two sentences are lexically and syntactically very similar, and only an adjective, *gastronomic* has been dropped; however, this has *fundamentally altered* the meaning of the sentence. The difference is that *gastronomic* completely changes the meaning of the word *capital*, whereas *red* has no effect whatsoever on the meaning of the word *dog*. In cases such as this, measures such as mutual information may be able to help.

However, let's look at another pair of examples. For the first example, while the second text unit is both lexically and syntactically similar to the first, it is not a paraphrase of the first:

a. *The event was a gastronomic delight.*

b. *The event was a delight.*

Now, let's compare that to a second example:

a. *The event was a gastronomic delight.*

b. *The food at the event was absolutely delicious.*

The second sentence in this example is far less similar both lexically and, particularly, syntactically to the first sentence than is the second sentence in the previous example; however, we would argue that this *is* a paraphrase of the first sentence. What's more, outside of the comparable document domain, there are likely to be a number of negative examples that are equally or more lexically and syntactically similar to this first sentence than is our paraphrase. This turns out to be a far more prevalent problem than we had anticipated, particularly in our own corpus—due to the prevalence of negative examples—but also in the RTE corpus, where due to the annotation approach, it's more immediately obvious.[2] We hypothesize that due to the fact that two comparable documents are, in effect, descriptions of the same object or event, lexical similarity has a far higher correspondence with paraphrasing than in other domains.

This is not to say that lexical measures are entirely useless—in fact, when detecting paraphrases in cases where they are rare, most negative examples *can* be weeded out using simple lexical measures. For example, using our testing set, if we use a model where examples are labeled as *positive* if their LSI values are greater than or equal to 0.4, over half of the positive

[2]Due to the annotation process, many of the negative hypotheses in the RTE corpus are still very lexically similar to the texts.

examples (57%) are marked as true, whereas only around 900 of the negative examples (less than 10%) are marked as true. However, that still leaves us with a 90-to-1 ratio of negative to positive examples, and the remaining negative examples cannot be easily distinguished from the positive ones using simple lexical measures. Thus it's clear that, whatever the weaknesses of PNrule, we need more sophisticated measures for this task.

The other possibility as to why PNrule couldn't find a clear signature in the hypothesis space is the limits of using a rule-based model: it has a difficult time discovering relationships between measures. For example, imagine we have two measures, $\mathtt{m}_1$ and $\mathtt{m}_2$, as well as a target class C. The signature for the target class could be something like $\mathtt{m}_1 + \mathtt{m}_2 > 1$. While a number of statistical machine-learning techniques could model this signature directly, a rule-based approach can only approximate it. For instance, it could come up with the following series of rules:

1. $\mathtt{m}_1 > 1$
2. $\mathtt{m}_2 > 1$
3. $\mathtt{m}_2 > 0.75 \wedge \mathtt{m}_1 > 0.25$
4. $\mathtt{m}_1 > 0.5 \wedge \mathtt{m}_2 > 0.5$
5. $\mathtt{m}_1 > 0.75 \wedge \mathtt{m}_2 > 0.25$

Even in cases where the target class was not rare, there's a difficulty in finding this approximation: in order to find one of the rules with two conditions, the first condition must do better than all other possible first conditions. If, for instance, there was some measure, $\mathtt{m}_3$, and $\mathtt{m}_3 > 0.3$ gave better results than $\mathtt{m}_2 > 0.75$, rule 2 would not be discovered. This problem arises because the rule-refinement stage uses a greedy search; replacing it with, for example, a beam search might ameliorate this problem, but there's no computationally-friendly way to eliminate it entirely.

In addition, there is another difficulty that is specific to cases in which the target class is rare. By approximating the signature using several different rules, we split up an already-small pool of positive examples; each rule will, at best, cover only a tiny number of these examples. What's worse, we almost inevitably add error, either because the rules cover examples they shouldn't, or because they miss examples. For example, imagine we had an example where $\mathtt{m}_1 = 0.35$ and $\mathtt{m}_2 = 0.7$. This example would be covered under the original signature, as $0.35 + 0.7 > 1$. However, it's not covered by any of the rules in our approximation.

So, then, we are attempting to detect a semantic relationship with a set of measures that can at best only approximate semantics, and with a model that, in all likelihood, is only able to approximate the target-class signature. This, combined with the fact that, when dealing with a rare target class, even a few errors can lead to terrible results, likely explains why our method failed. The only question that remains is to what degree each of the two problems contributed to our method's failure—a question that will have to be answered in future work.

10 Future Work

Clearly, a lot of work still has to be done toward solving this problem. Roughly speaking, we can separate future work into four groups: preprocessing, methodology, method, and measures.

10.1 Preprocessing

One obvious thing to try is to use a partial parser (Abney, 1996; Abney, 1997) to split the sentences in our corpus into simplex clauses, and then attempting to match those clauses. The danger here is that misparsing sentences might add an unacceptable amount of error; on the other hand, if the parsing is good, it would reduce noise.

Another preprocessing step that would be worth trying would be to identify any multiword WordNet entries that might appear in the corpus (e.g., *German shepherd*. Of course, then one might want to deal with such multiword entries both as a single unit and as separate words (since, in theory, a *German shepherd* could refer to a German who herds sheep).

10.2 Methodology

In hindsight, and in particular after reviewing the systems submitted for the RTE challenge, it has become clear to us that one mistake we have made in our methodology is to treat an asymmetric relationship as though it were a symmetric one. Imagine we have two sentences, s_1 and s_2. Currently, we use symmetric measures in an attempt to detect whether one sentence contains a paraphrase of the other. It would be better if instead we dealt with each pair twice: first check to see if s_1 contained a paraphrase of a clause in s_2, and then vice versa. This would allow us to use asymmetric measures: for example, if s_1 contains a hypernym of a word in s_2, that increases its chance of containing a paraphrase—however, it does *not* increase the chance that s_2 contains a paraphrase of a clause in s_1, and in fact may *decrease* that chance. Currently, our measures cannot reflect that distinction; this change would fix that.

10.3 Methods

As we can see from the submissions to the Pascal RTE challenge, there are a number of potential approaches to the problem of entailment (and thus paraphrasing). Limiting ourselves to approaches using machine learning, there are several things that could be tried. First, we could attempt to make changes to PNrule that might improve its performance. For example, we could try fitness functions other than Z-number or signature clarity. Another idea would be to start with a pseudo N-stage: this stage would find strong indications that examples were *negative*, and then create N-rules accordingly. Why focus on negative examples first? Three reasons:

1. because there's a large number of negative examples, noise is less likely to be a factor;
2. the strongest indicator that an example is negative is not necessarily the complement of the strongest indicator that an example is positive; and

3. it's possible for the strongest indicator that an example is negative to be obscured once examples have been chosen based on the likelihood that they are positive.

Of course, the pseudo N-stage would be required to keep recall at one: the idea would be to remove *obvious* negative examples; less obviously labeled examples would be dealt with in the normal P- and N-stages.

Second, we could adapt a different machine-learning technique to deal with rare classes, either by redesigning the technique, or by penalizing false negatives much more severely than false positives. Preferably, we'd pick a method that does not have the weaknesses of a rule-based system mentioned earlier. Alternatively, if we gathered far more data, we might be able to use something like k-nearest-neighbor unaltered; one advantage of that method is that it can handle small "clusters" of positive examples surrounded by negative examples within a vector space.

Finally, related to this second option, we could combine multiple learners, using a method such as a *mixture of experts*. These learners could be different machine-learning systems trained on the same measures, copies of the same machine-learning system trained on different measures, or different systems trained on different measures. We'd want each system to obtain near-perfect recall, but not necessarily good precision. The idea would be that each system would cover the same true positives, but *different* false positives. Assuming this is, in fact, the case, the combination of systems will outperform the individual systems.

In addition, any future system should be modified to handle negation and numbers; some of the systems submitted to the Pascal RTE provide ideas as to how this can be done.

10.4 Measures

The Pascal RTE challenge has provided a number of other measures to try; in particular, we feel that measures based on lexical chains might be helpful, given that this would allows us to add more contextual information to our measures. Another measure we'd like to try is one based on latent Dirichlet allocation (Blei et al., 2002), since it is similar in some ways to LSI (which was one of our strongest measures), but may offer some advantages over the former. That said, given the results of both our and other systems on the Pascal RTE data, it seems as though paraphrase detection (as well as entailment detection) will require innovative new measures; a major part of any future work will be finding these measures—for example, using mutual information to decide which adjectives are important and which can be dropped. In addition, it would be worthwhile to research methods to select which measures to use, since including some measures does nothing more than introduce noise.

Clearly, there's still a great deal of work to be done, both on our specific task, and on detecting entailment in general. We hope that this paper contributes some ideas to that future research, as well as perhaps highlighting some pitfalls to avoid.

Acknowledgements

I'd like to thank my supervisor, Graeme Hirst, and the Natural Sciences and Research Council of Canada. Much of this paper is taken from my Master's thesis (Bartlett, 2006), to which the reader can refer for further details.

References

Steven Abney. 1996. Partial parsing via finite-state cascades. *Natural Language Engineering*, 2(4):337–344.

Steven Abney. 1997. The SCOL manual, version 0.1b. `http://www.vinartus.net/spa/`.

Ramesh Agarwal and Mahesh V. Joshi. 2000. PNrule: A new framework for learning classifier models in data mining. Technical Report 00-015, Department of Computer Science, University of Minnesota.

Benjamin Bartlett. 2006. Finding Paraphrases Using PNrule. Master's thesis, University of Toronto.

Regina Barzilay and Noemie Elhadad. 2003. Sentence alignment for monolingual comparable corpora. In *Proceedings of the 2003 Conference on Empirical Methods in Natural Language Processing (EMNLP-03)*.

Regina Barzilay and Lillian Lee. 2003. Learning to paraphrase: An unsupervised approach using multiple-sequence alignment. In *Proceedings of the 2003 Conference of Human Language Technology/North American Association for Computational Linguistics (HTL/NAACL)*.

D. Blei, A. Ng, and M. Jordan. 2002. Latent Dirichlet allocation. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, Cambridge, MA. MIT Press.

E. Brill. 1994. Some advances in rule-based part of speech tagging. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 722–727, Seattle, WA. American Association for Artificial Intelligence.

Ido Dagan, Oren Glickman, and Bernando Magnini. 2005. The pascal recognizing textual entailment challenge. In *Proceedings of the Pattern Analysis, Statistical Modelling, and Computational Learning (PASCAL) Challenges Workshop on Recognising Textual Entailment*.

Vasileios Hatzivassiloglou, Judith L. Klavans, Melissa L. Holcombe, Regina Barzilay, Min-Yen Kan, and Kathleen R. McKeown. 2001. Simfinder: A flexible clustering tool for summarization. In *Workshop on Automatic Summarization, North American Association for Computational Linguistics*, Pittsburg (PA), USA.

V. Hatzivassliloglou, J. Klavans, and E. Eskin. 1999. Detecting text similarity over short passages: exploring linguistic feature combinations via machine learning. In *Proceedings of Empirical Methods in Natural Language Processing*, MD, USA.

R. C. Holte, L. E. Acker, and B. W. Porter. 1989. Concept learning and the problem of small disjuncts. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 813–818, Detroit, Michigan.

Mahesh V. Joshi, Ramesh C. Agarwal, and Vipin Kumar. 2001. Mining needles in a haystack: classifying rare classes via two-phase rule induction. *SIGMOD Record (Association for Computing Machinery Special Interest Group on Management of Data)*, 30(2):91–102.

Mahesh Vijaykumar Joshi. 2002. *Learning Classifier Models for Predicting Rare Phenomena*. Ph.D. thesis, University of Minnesota.

R.J. Landis and G.G. Koch. 1977. The measurement of observer agreement for categorical data. *Biometrics*, 133:159–174.

D. D. Lewis. 1999. Reuters-21578 text categorization test collection distribution 1.0. `http://www.research.att.com/~lewis`.

Lluís Màrquez. 2000. Machine learning and natural language processing. Technical Report LSI-00-45-R, Departament de Llenguatges i Sistemes Informàtics (LSI), Universitat Politecnica de Catalunya (UPC), Barcelona, Spain.

Tom M. Mitchell. 1997. *Machine Learning*. McGraw-Hill.

Chris Quirk, Chris Brockett, and William Dolan. 2004. Monolingual machine translation for paraphrase generation. In Dekang Lin and Dekai Wu, editors, *Proceedings of Empirical Methods in Natural Language Processing (EMNLP) 2004*, pages 142–149, Barcelona, Spain, July. Association for Computational Linguistics.

Y. Shinyama, S. Sekine, K. Sudo, and R. Grishman. 2002. Automatic paraphrase acquisition from news articles. In *Proceedings of the Human Language Technology (HLT) Conference*, San Diego, USA.

G. M. Weiss. 1995. Learning with rare cases and small disjuncts. In *Proceedings of the Twelfth Joint International Conference on Artificial Intelligence*, pages 558–565.