A Hybrid Rule Induction Classifier for Real-Time Classification and Incremental Learning

Kenrick Mock
Jan 10, 1999
Business Applications Components
Intel Architecture Labs
kenrick.j.mock@intel.com

Abstract

Intel® has created an Evaluator Toolkit (ET) as a way to easily integrate information management technologies into a myriad of applications. In the initial report (Mock, 1998) a vector-based classifier and a rule induction algorithm were described. While the rule induction algorithm was effective, it suffers from a slow induction process that requires tens of minutes to generate rules. This report describes a modification to the rule induction algorithm that is intended to support the incremental learning of rules and the generation of more meaningful relevance rankings for matching rules. The modified system is called the *Hybrid Rule Induction* classifier because it incorporates a shortened vector of words into an inducted profile of rules. To support incremental learning, all negative training examples are remembered by each profile. Performance results on the Reuters-21578 data set indicate that the incremental training is almost as effective as block training, and more effective than if no incremental training is done at all. However, induction time is often too slow for real-time usage on current processors but may be acceptable for background usage.

1. Rule Induction Background

One technique that is used to classify documents is through a list of decision rules. Decision rules take up very little memory, usually a few kilobytes, and are in a format easier for people to understand and modify than TF-IDF frequencies used in vector-based classifiers. We will pose the rules to be unordered (we could apply the rule in any order, not an if-then-else order as is generated by decision trees) in disjunctive normal form (DNF). For example, the rules could look like the following:

1. if AGENT then Class=Agents
2. if INTELLIGENT and MOBILE then Class=Agents
3. if NOT AGENT and ADMIN then Class=Administrative
4. if PARTY then Class=Personal

In DNF, we have only AND's within each rule, but an OR of all rules.

Developing a learning system for this representation is somewhat difficult. In ET, if multiple rules fire, then we will category the document into all categories matching a rule. However, in general we may have problems if we just OR together all of the rules.

The problem with OR'ing together rules is that, surprisingly, we might end up lowering our performance. Consider if you have two rules and 1000 cases. Rule 1 is activated (covers) 100 cases, and is correct on 90 of them. Rule 2 also covers 100 cases, and it is correct on 90 of them. What happens when we OR these rules together? In the best case, the 90 correct cases are different, but the incorrect cases are identical. The accuracy is now (90+90) / (90 + 90 + 10) = 0.95. However, in the worst case the two rules are correct on the same cases but wrong on different cases. The combined classifier is now 90 / (90+10+10) = 0.82.

The end result is that we need to be careful inducing the rules, and that the vector classifier may be better in some cases than the rule induction algorithm.

CN2 Induction Algorithm

ET implements a modified version of the CN2 rule induction algorithm. CN2 is one unordered rule induction algorithm designed by Peter Clark.

The idea of CN2 and other rule induction algorithms is to search the space of decision rules from the general to the specific. It employs a beam search to determine what rules to generate within the search space. This search space is exponentially large. If we only have 3 features, X, Y, and Z (in our text classification domain, these features would probably be specific words), then we could generate the following possible rules:

If X then…
If X and Y then…
If X and Y and Z then…
If X and Z then …
If Y then …
If Y and Z then …
If Z then…

The number of rules grows exponentially, 2^n-1. There are even more rules (2^{2n}-1) if we consider NOT X, NOT Y, etc. as features. With features that are typically in the tens or hundreds, search through this space can be extremely difficult. Essentially we will be searching through this space of possible rules for the best rule for the training data.

There are three procedures in the algorithm, which is described in general terms below from Clark and Boswell (1991):

```
CN2Unordered(allexamples, allclasses)
        Ruleset ← {}
        For each class in allclasses
                Generate rules by CN2ForOneClass(allexamples, class)
                Add rules to ruleset
        Return ruleset
```

```
CN2ForOneClass(examples, class)
    Rules ← {}
    Repeat
        Bestcond ← FindBestCondition(examples, class)
        If bestcond <> null then
            Add the rule "IF bestcond THEN PREDICT class"
            Remove from examples all cases in class covered by bestcond
    Until bestcond = null
    Return rules
```

The first procedure simply loops through all possible classes (categories) and generates a set of rules for each class.

The second procedure repeatedly calls "FindBestCondition" which will find the best rule, according to a heuristic, that covers the current set of examples. This rule is added to the set of rules. Then, all example cases that are positively matched by the rule are removed from the example set and the process repeats.

In the rule induction algorithm, it is important that we keep negative examples of the rule around so that future rules stand out from the negatives. We must remove the positive examples to prevent us from finding the same rule again. The task remains to implement the FindBestCond routine:

```
FindBestCondition(examples, class)
    MGC ← true     ' most general condition
    Star ← MGC
    Newstar ← {}
    Bestcond ← null
    While Star is not empty  (or loopcount < MAXCONJUNCTS)
        For each rule R in Star
            For each possible feature F
                R' ← specialization of Rule formed by adding F as an
                    Extra conjunct to Rule (i.e. Rule' = Rule AND F)
                    Removing null conditions (i.e. A AND NOT A)
                    Removing redundancies (i.e. A AND A)
                    And previously generated rules.
                If LaPlaceHeuristic(R',class) better than
                LaPlaceHeuristic (Bestcond, class)
                    Bestcond ← R'
                Add R' to Newstar
                If size(NewStar) > MAXRULESIZE then
                    Remove worst in Newstar
                    until Size=MAXRULESIZE
            Star ← Newstar
    Return Bestcond
```

The LaPlace heuristic is simply:

$$LaPlaceHeuristic = \frac{\#ExamplesInClass\Pr edictedByRule + 1}{\#TotalExamplesCoveredByRule + \#ClassesInDomain}$$

ET considers each class individually (e.g., Class or NOT Class) so the # of classes in the domain is set to 2.

A larger heuristic value is better and indicates how many examples are correctly predicted by a given rule. To compute the heuristic, we must apply the rule to all training examples and count the number of cases in the numerator and denominator. This is an expensive process, since the computation is buried with the loop that generates the features to test.

Initially, FindBestCondition will start with Star containing only the default “true” rule. To this we add specializations by testing the rules with one feature. For example if our features were X, Y and Z, then on the first pass we would generate “IF X then Class”, “IF Y then Class” and “IF Z then Class”. All of these rules would be tested to see which has the best LaPlace heuristic.

Next, we keep track of the best rule so far and only remember the MAXRULESIZE best rules. If MAXRULESIZE = 2, then we might only keep the rules “IF X then Class” and “IF Y then Class”. By limiting the rule size, we enforce a narrow “beam” in which we are searching through the rule space. To the top rules, we specialize them further, AND’ing each rule with all features. We would now have the rules:

> IF X and Y then class
> IF X and Z then class
> IF Y and Z then class

The process repeats. The next loop we would generate rules with three conjuncts, until we run out of features or reach the MAXCONJUNCTS limit.

Finally, when iterated over all classes, we will have a set of rules that predict each class. ET also stores the LaPlace heuristic value along with each rule as an indicator of the confidence of the rule. While this confidence value may be used as a relevance metric for visualization, the distribution of values tends to clump around 1 or less than 0.5. Consequently, the values will not result in a smooth or normal distribution curve like the vector evaluator’s relevance values. We have observed that LaPlace heuristic values > 0.5 are associated with fairly relevant rules, while rules with a heuristic confidence < 0.5 are often poor enough to be thrown out. The entire rule induction routine is quite compute intensive, especially for a large number of features and sample cases. In practice, we place limits on the MAXCONJUNCTS, MAXRULES, and MAXRULESIZE so that computation can be done within a reasonable time.

2. Rule Induction Weaknesses

A variant of the CN2 rule induction algorithm was originally implemented in the Intel Evaluator Toolkit (ET). The advantages of the induction algorithm include a small profile, excellent classification performance, and rules that can be easily understood by humans. The disadvantages include poor relevance ranking and difficulties in incrementally training the profile.

The only relevance values that the original rule profile returned are rule confidence values. These values are zero if the document does not match a rule, or a semi-discrete number (typically 2/3 or 3/4) if the rule does match. This discrete distribution does not give a clear indication of how relevant a document is with respect to the profile. In contrast, vector comparisons typically result in fine resolution relevance values in a normal distribution between 0 and 1.

In terms of incremental training, the rule induction algorithm requires a large batch of negative and positive examples to be provided before rules can be inducted. After rules are generated, then the examples are discarded. Any additional training requires the complete regeneration of rules from new examples. No facility was provided for new rules to be generated in addition to the old rules.

3. Rule Induction Modifications

To address the weaknesses of the algorithm, a number of modifications were performed. To provide better relevance values, a vector comparison is made with a small word vector that is created from the most frequently occurring words in the positive examples. The resulting value is then combined with the largest heuristic value from matching rules. To incrementally train the profile, negative examples are retained along with some number of positive examples. New examples are combined with a subset of existing positive examples before new rules are inducted and then added to the rule set.

Hybrid Vector Relevance Values

In the original algorithm, rules are associated with their corresponding matching LaPlace heuristic. The LaPlace heuristic is simply:

$$LaPlaceHeuristic = \frac{\# ExamplesInClass\,\mathrm{Pr}\,edictedByRule + 1}{\# TotalExamplesCoveredByRule + \# ClassesInDomain}$$

ET considers each class individually (e.g., Class or NOT Class) so the # of classes in the domain is set to 2.

A larger heuristic value is better and indicates how many examples are correctly predicted by a given rule. To compute the heuristic, we must apply the rule to all training examples and count the number of cases in the numerator and denominator. Note that this heuristic favors a rule that covers more examples than fewer examples. For example, if the rule correctly covers one example, the value is 2/3 = 0.667. However, if the rule correctly covers two examples, the value is 3/4 = 0.75. After the majority of cases have been covered by general rules, rules that cover 1 or 2 examples (and with heuristic values of 0.667 and 0.75 respectively) are commonly generated to cover outlying cases. Although these rules will generally have poor predictive power, they may be necessary to cover certain types of examples.

The resulting LaPlace heuristic values do provide some indication of relevance for a new document. If a document matches a rule with a heuristic value > 0.8, then that document is probably quite relevant. Conversely, a document that only matches a rule with a 0.667 value is probably not as relevant. However, the LaPlace heuristic values typically occupy only a limited number of discrete values between 0.667 and 1 (but may be 0-1). This range is not suitable for a variety of visualization techniques that require a finer mesh of discrimination between relevance values.

To flesh out the relevance values, a vector comparison is made with the most frequent words collected from the positive examples. Currently limited to the 50 most frequent words from all of the positive training examples, these words are normally generated by the rule induction algorithm and used as features for the rules. Since these words are only taken from positive examples, they do not provide much help in determining if something is not a member of the profile. However, they can help determine how strongly a new article matches the profile. If all of the words (i.e. features) in a new article match a profile rule and also match all words in the profile, then that article is likely to be highly relevant. However, if a new article matches a rule but only match one word in the profile, it is probably not as relevant as the previous example.

The vector comparison is a simple binary comparison. For a new article containing features $F_1 \ldots F_i$ and a profile containing words $P_1 \ldots P_j$ the relevance value is computed as:

```
NumMatches ← 0
For I ← 1 to NumFeaturesInArticle
    For J ← 1 to NumFeaturesInProfile
        If F(I) = P(J) then NumMatches++
VectorRelevance ← NumMatches / I
```

The resulting relevance value reflects the number of terms that match with respect to the size of the input document. This is only a rough estimate of relevance; a better value could be determined if frequencies were retained and tf-idf values calculated, as is done in the normal vector profile. Further work is necessary to determine if the extra statistics significantly improve the relevance values. However, this rough relevance computation may be sufficient since the vector value is ultimately combined and averaged out with the LaPlace heuristic value. Consequently, the impact of the relevance value alone is not as great since it is modified by the LaPlace value.

Two experiments were performed with different methods for combining the heuristic values. The "split-value" method separates cases where a rule matches at 0.5:

```
If BestLaPlaceMatch > MatchThreshold Then
        Classify = 0.5 + ((BestLaPlaceMatch +VectorRelevance) / 2) * 0.5
Else
        Classify = VectorRelevance * 0.5
End If
```

The MatchThreshold variable is a parameter that is used to control precision vs. recall. A low threshold such as 0.66 will allow most rules to fire. As a result, recall will be maximized but precision may be lower. A higher threshold such as 0.8 will restrict the system to only more general rules. For ultra-precision, a value above 0.9 can be used. As a result, precision will be maximized but recall may be lower. This parameter can be mapped to a user-controlled knob so that the appropriate data may be retrieved based upon the desired application context.

The split-value classification combination algorithm selects 0.5 as the cutoff point. Any value less than 0.5 means that no rule was found that matched the input text, and the return value is the vector relevance scaled between 0 and 0.5. If a rule does match, the algorithm returns 0.5 + the vector relevance added to the LaPlace heuristic, divided by their maximum combined value of 2 and then further scaled to a maximum of 0.5. The end result in the case of a match is a value between 0.5 and 1.

The second experiment simply averaged together the two heuristic values, using a value of 0 for BestLaPlaceMatch if the rule heuristic is not greater than the threshold:

```
If BestLaPlaceMatch > MatchThreshold Then
        Classify = (BestLaPlaceMatch +VectorRelevance) / 2)
Else
        Classify = VectorRelevance  / 2
End If
```

This version of the heuristic should result in a smoother distribution of relevance values than the split method, but will not as clearly delineate cases where a rule does or does not match.

Incremental Learning

The simplest and likely the most accurate form of incremental learning is to retain all positive and negative examples. As new training examples are provided they are simply added to the existing set and an entire new batch of rules is generated. While this method will probably generate the best rules, it is computationally expensive to generate an entire set of new rules each time that a new training example is presented. One solution is to wait until “enough” new cases have been collected, and then do a new generation of rules all at once.

For some applications, it is sufficient to induct rules once and then repeatedly use the generated rules. However, other applications require incremental modifications to the profile. As user interests change or additional training data becomes available after the initial training period, there should be a method to automatically update the profile to reflect the additional data. Furthermore, the update process should complete in a reasonable amount of time.

Instead of training an entire new set of rules, which is typically a time-consuming process, the profile was modified so that a single rule or set of rules could be generated and added to the existing rules. This requires that all of the negative examples and some of the positive examples be retained in the profile. At a minimum, the negative examples must be kept so that newly generated rules will cover as few negative cases as possible. To speed up the rule generation process, only a subset of the positive examples is used. At least one of the positive examples should be a new training case.

By generating new rule(s) based upon the negative examples and only a subset of positive examples, the hope is that the rule induction process will run much faster than if all rules were generated from all examples. Furthermore, if only a subset of positive examples is required, then the profile can discard many of the positive examples instead of remembering all examples. This can potentially save considerable amounts of memory.

Consider the case where positive and negative examples have been provided and rules have just been generated. Now, the user has a single new positive example and would like to update the rule set to include this new example. A number of possibilities exist:

1) The new example is already covered by an existing rule. If this is the case, then no new rules need to be inducted.
2) A new rule can be inducted that covers the new example using existing features.
3) A new rule cannot be inducted that covers the new example using existing features, and new features must be added and a new rule generated.

The hybrid rule profile addresses all of these possibilities in order. If the example is already covered, then no action is taken. Otherwise, the system attempts to generate a new rule using the single positive example and all of the negative examples. If a rule cannot be found, then the most frequently occurring feature in the new example that is not already in the profile’s feature set is selected. This new feature is added to the profile’s feature set. The process is repeated to induct a new rule and the next most frequently occurring feature added if necessary.

By retaining all of the negative examples, the new rule that is inducted should have little impact on existing rules. That is, it should not cause a large contribution to false positives since all known negative examples were considered in generating the rule.

However, the new rule is likely to be highly specific and may only cover the new positive example and not any other examples. Under this scenario, the largest LaPlace heuristic that will be generated for new rules is 0.667 since each time we induct rules, we will only have one positive example.

To induct more general rules requires the addition of more positive examples. However, instead of using all positive examples, a small set of the last N positive examples can be used, where N is a number such as 3 or 10. This is a type of "sliding window" technique where a set of old positive examples may be reused with new positive examples to generate new rules. In this manner, N new examples may be added at once and then all N examples used to induct rules, or 1 new example can be added but the last N positive examples used to induct rules. In the last case, we are looking for just one additional rule that covers the single new example. However, if this rule also covers other positive examples (that already happen to be covered by other rules) then the new rule will likely be more general than a rule that covers only the new example.

The implementation of this process requires a few modifications to the induction algorithm:

```
CN2Unordered2(allExamples, numPositive, Ruleset, knownClass)
    tempRuleset ← {}
    tempExamples ← {}
    For each negative example in allExamples
        Add negative example to tempExamples
    For most recent numPositive positive examples in allExamples
        Add positive example to tempExamples
    If Ruleset covers all positive examples in tempExamples then
        Return Ruleset
    else
        Do
          Rule R ← CN2ForOneClass(tempexamples, knownClass)  ' Induct rule
          If R is not in Ruleset then Add R to Ruleset
        While R <> empty
    Return Ruleset
```

The modified procedure takes all known examples, the existing ruleset, and the number of positive examples to use, and the class that we want to induct rules for. All negative examples and the most recent positive examples, up to the desired threshold, are used for generating rules. If the existing ruleset already covers all known positive examples then no work needs to be done. Otherwise, rules are inducted using the normal procedure and any duplicates are discarded.

4. Reuters-21578 Test Runs

The Reuters-21578 test collection that was used in the original experiments was also used to test the performance of the modified hybrid rule profile. To simulate how a user may train their profiles in practice, experiments were conducted in two phases: an initial training phase, and then an incremental training phase. In the initial training phase, a large set of positive and negative examples was selected for rule induction. In the incremental training phase, additional rules were inducted as small batches of new positive examples were supplied. After both phases were complete, the resulting rules were run on the test set using various cutoff thresholds for the rule heuristic value. This process intends to model the way a user will initially generate rules, use the results, and then train the profile further in real-time using the results of the generated rules.

In the initial training phase, rules were inducted using a set of positive and negative examples. 400 negative examples were randomly selected for training. If the total number of positive examples available was less than 40, then all positive examples were used for training. Otherwise, the first 33% of the positive examples were used. The positive examples were selected in order, i.e., if there were 300 positive examples total, then the first 100 positive examples were used.

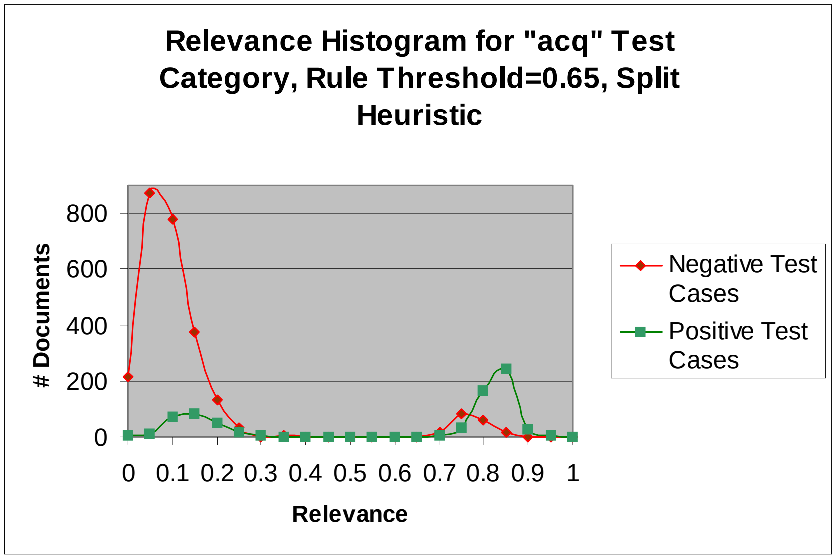
In the incremental training phase, up to100 positive examples were selected in order for training using the remaining positive examples. As iteration proceeded through these remaining positive examples, they were added to the profile in order and rules were inducted in batches of N in different experimental runs. These rules were added to the existing rules that were generated in the initial training phase.

Two parameters were varied during these runs. First, the batch size for incremental induction was varied. This parameter was set to 0, 1, 4, and 10. At a value of 0, no incremental induction was used and testing proceeded only on the rules inducted in the initial training phase. At a setting of 1, only the current new message was used as a positive training example. At a setting of 4, the current new message plus the 3 most recent positive messages were used as positive training examples, etc. Note that the batch size refers to the number of positive messages used for inducing rules, not the number of messages that are added for training before rules are inducted (although certainly a viable procedure). Instead, rules were inducted on every single incremental message using the specified batch size. An additional run was conducted using all incremental messages in one batch induction. This was expected to give the best results, while the run that ignores the incremental messages should be the worst.

The second parameter varied was the rule confidence threshold. This parameter was set to 0.65, 0.69, and 0.79. Any rules that have a LaPlace heuristic smaller than the threshold are discarded. A value of 0.65 will retain rules that cover only a single positive example. A value of 0.69 will discard rules that cover only a single value, but retain rules that cover two examples. Similarly, a value of 0.79 will discard rules that cover less than three positive examples. A larger threshold results in increased precision, but lower recall. Since there are discrete threshold for performance, this parameter is an excellent candidate for a “knob” that the user can adjust for desired performance.

Hybrid Profile Results

After profiles were generated, statistics were gathered for the "acq" category while the profile classified test data. The heuristic values were collected and a histogram plotted for test data that belonged to the category and test data that did not belong to the category. The results using the "split-value" combination scheme are shown in the figure below:

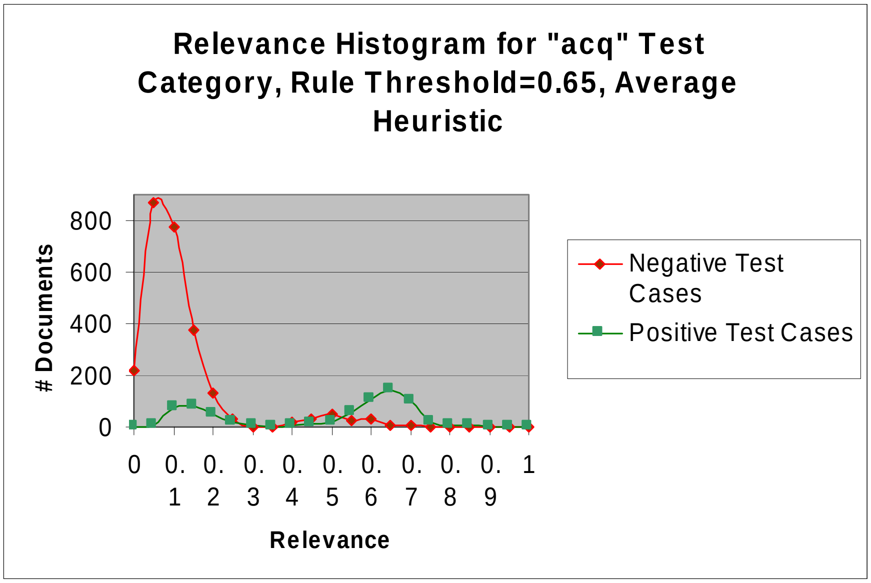


The histogram plots the number of documents that were classified with a particular relevance value. The green line depicts the relevance values for test set documents that belong to the category, while the red line depicts the relevance values for test set documents that did not belong to the category.

Due to the nature of the heuristic, which splits the heuristics around 0.5, the result is a double distribution centered on positives and negatives. The green hump centered at 0.8 represents the rules the true positives, while the green hump centered at 0.15 represents the positives that were missed. Similarly, the red hump centered on 0.75 represents the false positives, while the red hump centered at 0.1 represents the true negatives.

Overall, the split heuristic appears effective at separating the positives from the negatives, but does result in a fairly narrow range (0.7-0.9 in this case) of values for positive cases.

The results using the average heuristic scheme, which simply averages together the LaPlace and Vector values, is shown below:



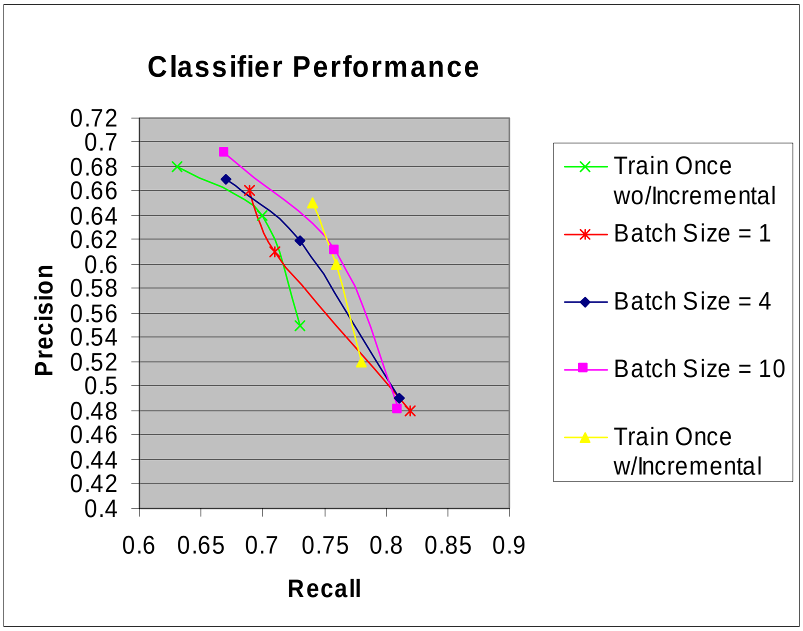
While the average heuristic scheme still results in two distinct humps for true positives and missed positives, the false negatives have been smoothed out compared to the split scheme. Similarly, the standard deviation for the positives is greater, resulting in slightly more discriminating relevance values.

While both schemes have a limited distribution range, the humps are typical of a normal distribution. This facilitates visualization systems that would otherwise fail if only discrete, binary rule thresholds were available.

Incremental Rule Induction Results

If the incremental rule induction strategy is successful, then the precision/recall performance should be superior in the incremental induction trials than in the trials where no incremental induction is performed. However, the strategy may be unsuccessful if the rules inducted during incremental induction are poor and trigger a large number of false hits. This is a possibility since the incremental rules are generated with a small set of positive examples.

Five different test runs were performed. For each run, the rule confidence threshold parameter was set to 0.65, 0.69, and 0.79 to explore the tradeoff between precision and recall. The plots shown are the average between two separate trials. To smooth the curves, additional trials should be run, but have been postponed due to time constraints. The precision/recall curves for the runs are shown in the figure below. Note that these results are not directly comparable to the results from the previous report since fewer training examples are being used and in a different distribution.



The first run in green depicts the performance without any incremental training. As expected, it performs the worst out of all runs with a precision and recall curve underneath all others and a breakeven point of about 0.66. This is expected since the trial does not use any of the data that the incremental training algorithms use. The results do show that the rules generated by incremental training actually help improve classification performance than if the rules are not generated at all. However, care must be taken: rules with low heuristic values tend towards high recall but low precision. For better precision, larger rule thresholds should be selected.

The second run in red shows the performance when the batch size is 1; that is, only the one single positive example is used when inducing new rules during the incremental training phase. The rules generated using only a single positive example are not very general, and the graph reflects this problem through improved recall but lower precision at low thresholds (bottom right end of the graph). Since the rules generated during the incremental phase will all be 2/3 or lower due to the single positive example, performance mirrors the training examples when no incremental training is done as the rule activation threshold increases.

The blue and purple lines indicate the performance when the batch size is set to 4 or 10. While the precision is still low at small threshold settings, the precision moves up as the threshold is increased. This indicates that useful rules are being generated during the incremental training process, and that the most useful rules cover more than one positive example (and therefore continue to fire as the threshold setting is increased). Using a batch size of 10, the breakeven point is approximately 0.68.

Finally, the curve in yellow depicts the performance when the same messages used in the incremental training are added to the profile without inducting rules, and one single batch of rules is inducted at the end. The performance of this run appears best for high precision and has a projected breakeven point of approximately 0.70. As expected, this is the best breakeven point of all the runs and is intended to show an upper bound on performance if incremental training is not necessary.

5. Conclusions

These experiments focused on two issues: providing a smooth distribution of relevance values and providing a mechanism for incremental learning. The hybrid relevance profile that consists of vector evaluations and rule heuristics do result in normal relevance distributions lumped around positive and negative classifications. This should be useful for visualization and ranking documents in finer detail than the previous system that only provided discrete relevance values.

Although not dramatic, the incremental learning modifications also showed improvements over the lack of incremental learning. However, performance was still slow to induct the incremental rules, 5-10 minutes per 500 document profile on a computer system with a single 200 MHz Pentium® Pro processor. While this is too slow to be performed in real time, it is an operation that could be performed as a background task or may be suitable for real time as processor performance increases. The results also show that the rule threshold is a simple knob that can be turned to adjust precision vs. recall. Finally, at some point a completely new set of inducted rules appears to perform better for higher precision than the incremental rules, and this could also be factored in if the processor time is available.

6. References

Apte, C., Damerau, F., & Weiss, S. (1994). Automated learning of decision rules for text categorization. ACM Transactions on Information Systems, 12, 233-251.

Clark, P. & Boswell, R. (1991). Rule Induction with CN2: Some Recent Improvements. Proceedings of the Fifth European Conference on Machine Learning. http://www.cs.utexas.edu/users/pclark/

Salton, G. (1991). Developments in Automatic Text Retrieval. *Science*, 253, pp. 974-980.

Cohen, W. (1996). Learning Trees and Rules with Set-valued Features. AAAI-96. http://www.research.att.com/~wcohen/

Lewis, D. (1992). An Evaluation of Phrasal and Clustered Representations on a Text Categorization Task. Fifteenth Annual International ACM SIGIR. Copenhagen.

Lewis, D. (1997). The Reuters-21578 Test Collection, Distribution 1.0 http://www.research.att.com/~lewis/reuters21578.html

Lewis, D. & Ringuette, M. (1994). A comparison of two learning algorithms for text categorization. Symposium on Document Analysis and Information Retrieval, Las Vegas, NV.

Mock, K (1998). “Evaluating the Evaluators: Performance Measures for the Evaluation Toolkit”. Intel Technical Report