Multipipes: Exploring Disjuntive Classifications in Hyperpipes

Aaron Riesbeck
West Virginia University
100 Address Lane
Morgantown, WV 26505
ariesbeck@theriac.org

Adam Brady
West Virginia University
100 Fake St.
Morgantown, WV 26505
adam.m.brady@gmail.com

ABSTRACT

This paper explores classifiction with disjunctive sets using a modified form of HyperPipes [1] called MultiPipes. Rather than apply HyperPipes it's intended sparse datasets, we find that it's application to non-sparse, many-class datasets typically results in several tied classification scores which we then union into a disjunction. This union presents interesting possibilities in it's high accuracy in containing the target class. Although we initially cannot predict single classes, we find that these disjunctions often eliminate large portions of possible classes. Essentialy we aren't certain what the class is, but we are very certain of what the class is not. The rest of the paper explores two alternative strategies with MultiPipes. The first involves methods of reducing the disjunctive sets to single classifications. The second considers growing the disjunctive sets to optimize the accuracy of containment vs. set size.

Keywords

HyperPipes, disjoint sets, LaTeX, multiple classes, indecisive learners

1. INTRODUCTION TO HYPERPIPES

HyperPipes is a learner originally offered by Eisenstein et al [1] for extremely large, sparse datasets. Rather than maintain a large working memory of statistics on each row of data, HyperPipes maintains a small data structure for each class that merely "remembers" whether a particular attribute has been encountered before. For numerics, a range of maximum and minimum values encountered is kept. When classifying, original HyperPipes classifies a row based on which class most "contains" the current attributes. For numeric attributes, a new instance is "contained" if it falls within the maximum and minimum values see so far.

This is perfect for sparse datsets, as your working memory need only contain one HyperPipe for each possible class, and each pipe must maintain only a the unique symbols encountered so far along with two numeric bounds for each column. The result is a fast, dumb, scalable learner.

One caveat remains in the HyperPipes algorithm. For large, sparse datasets there are enough unique columns to promote a wide variance in which HyperPipe best "contains" a class. However, for more traditional datasets with fewer columns HyperPipes' accuracy breaks down. By nature HyperPipes is strongly succeptible to outlier data, as the frequency of an encountered attribute is ignored and extremely high or low numerics will stretch the bounds. In this paper we explore a means to maintain the speed and scalability of HyperPipes while extending it's application to a wider variety of datasets.

Program 1 HyperPipes Pseudo Code.

```
procedure RunHyperPipes($Nothing$)
    $HyperPipes := array[]$
    $Guessed := 0$
    $GuessedCorrect := 0$
    for all Line in DataFile do
        $Guessed++$
        $GuessedCorrect{+}{=}Classify(MyHyperPipes, Line)$
        $HyperPipes := AddExperience(Line, HyperPipes)$
    end for
    $Accuracy := GuessedCorrect/Guessed$
end procedure
```

1.1 The Problem with HyperPipes

1.1.1 Tied Classes

During development of the original HyperPipes. implementation it was noted that the code only kept track of the most recently seen class with a score equal to or greater than the largest score seen to that point. This means that when running HyperPipes if multiple classes tied in score the class tested last would be blindly chosen. We found that this anomoly caused our HyperPipes implementation to prefer the last classes discovered once it learns many rows.

1.1.2 Susceptible to Over Fitting

HyperPipes has the ability to have very low overhead costs in terms of memory. However, this benefit is overshadowed by HyperPipes susceptibility to overfitting. As more and more rows are added as experience to a HyperPipe its min and max values with slowly approach the min and max of the attribute alone regardless of class. This causes HyperPipes guessing ability to completely fall apart. As the min and

Program 2 HyperPipes Classify Pseudo Code.

```
procedure CLASSIFY($HyperPipes$,$Line$)
    $BestScore := 0$
    $BestClass := []$
    for all HPipe in HyperPipes do
        $HPipeScore := 0$
        for all Attr in Line do
            if $Attr <= HPipe.Attr.max$ && $Attr >= HPipe.Attr.min$ then
                $HPipeScore++$
            end if
        end for
        if $HPipeScore >= BestScore$ then
            if $HPipeScore := BestScore$ then
                $BestClass := BestClass + HPipe.class$
            else
                $BestClass := array(HPipe.class)$
                $BestScore := HPipeScore$
            end if
        end if
    end for
    if $BestClass$ contains $Line.class$ then
        return 1
    end if
    return 0
end procedure
```

Program 3 HyperPipes Classify Pseudo Code.

```
procedure ADDEXPERIENCE($HyperPipes$,$Line$)
    $CurrentPipe := FindPipe(Line.class, HyperPipes)$
    if null $CurrentPipe$ then
        $HyperPipes := HyperPipes + CreateHyperPipe(Line.class)$
        $CurrentPipe := FindPipe(Line.class, HyperPipes)$
    end if
    for all Attr in Line do
        $CurrentPipe.max := max(CurrentPipe.max, Attr)$
        $CurrentPipe.min := min(CurrentPipe.min, Attr)$
    end for
end procedure
```

max values for the attributes in every HyperPipe expand the scores become very high and the likelyhood of encountering the issue in the previous sub-section becomes greater. That is, as each of the HyperPipes expand their min and max attribute values start to become so similar to eachother that they all begin to acheive common scores.

1.1.3 Outliers Ruin Scoring

As shown in the AddExperience function (Program 3) a HyperPipe contains a set of bounds for each attribute. These bounds indicate the min and a max value ever seen for this attribute for this class. This becomes a problem when an outlier for that attribute occurs. Imagine after 10,000 rows your min is 23 and your max is 45. You encounter a new row where this attributes value is 431. This causes your new max value to be 431. This can be a major pitfall if this is the only time a number this large is encountered for this class. You have now expanded this HyperPipe to a max so large that this attribute loses its ability to classify (It always matched on this attribute for this class).

1.1.4 Memory Management

The original HyperPipes required prior knowledge of the dataset. In other words it created all of the necessary HyperPipes at the beginning and modified them as it went through the dataset. This means that if a new class is created the entire algorithm would have to start from the beginning with this knowledge that this new class existed. If you were able to control the outliers and the over fitting for this algorithm you could use this type of classifier in a real world situation where new classes are constantly being discovered.

1.2 Patching HyperPipes

1.2.1 Fixing Tied Classes

When it was discovered that HyperPipes tended to choose classes towards the end of the classes list we investigated further to find that many classes were tieing against other classes. We then decided that this was unacceptable. Overwriting a class with another class of the same score puts a large emphasis on the order in which you score the classes. For this issue we decided to temporarily throw away the idea of classification and modified the code to return any classes who tied. This simple modification proved to us that this issue alone was a major factor in HyperPipes demise when put up against other classifiers. To recap the original hyperpipes said that if the current score is equal to or greater than the best score set the best class to the current class. After modification hyperpipes now states that if the score is greater than the best score set the best class to an array containing only the current class. If the current score is equal to the best score then append the current class to the list of best classes.

1.2.2 Fixing Over Fitting

As described previously Hyperpipes has an over fitting issue when it learns too much. While we have no implemented these potential fixes our possible solutions are described below:

1. Limit Number of Rows To Be Learned: It might be effective to simply limit the number of rows that Hy-

perPipes will use when doing its learning. This number of rows might be calculated based on the number classes and it may also require that this limit be evenly distributed across all classes.

2. Detect Overfitting by Class Overlap: It may be possible to detect over-fitting by determining the amount of overlap between classes. In other words, if the number of attributes in a HyperPipe reaches a certain level we could say that we should not modify our HyperPipes with the information learned in this new line as it would cause too much overlap between classes.

1.2.3 Fixing Outliers

We have implemented some fixes to the problem of outliers. However, there are many other outlier fixing strategies that have not been implemented. Below is a list of our outlier fixing strategies and thier implemtation status:

1. Weighted Means: This fix has been implemented. What this fix does is it changes the scoring mechanism. In the pseudocode for Classify (Program 2) you will notice that it returns 1 if the current attribute value falls within the range of that given by the HyperPipe for that Attribute. The Weighted Means changes the 1 to a number between 0 and 1 which represents the distance from the mean within the range. Two versions of this are explained further in Section 5.

2. Slowly Changing Ranges: This fix has not been implemented. This fix involves setting up some way of expanding the ranges more slowly for instance, if you encounter a value lower than the current min instead of setting the min to the current value set the min according to some calculation that accounts for the frequency of numbers lower than the current min.

3. Outlier Detection Algorithms: This fix has not been implemented. There are numerous proven outlier detection algorithms which could be used to attempt to detect outliers as we encounter them.

1.2.4 Fixing Memory Management

As explained above the old HyperPipes implementation needs information on the entire dataset for intialization. We saw value in making HyperPipes work on an ever changing stream of data. We implemented this by intializing our set of HyperPipes to an empty array. As new classes are discovered a new HyperPipe is created for that class and appended to the HyperPipes array. This allows us to use a straight datafile with just columns of data and no header information.

2. NARROWING VS. CLASSIFYING

It may seem unfair to consider an algorithm that in many ways, cheats. While most classifiers are burdened with the task of predicting for a single class, it is possible for our algorithm to return all possible classes and claim victory in a meaningless fashion. While we will demonstrate that this extreme overclassifying is rarely the case, it stands to reason that comparisions cannot be made between the accuracy of our disjunctions and those of single classifiers.

However, just as HyperPipes had it's niche in sparse data, MultiPipes has a niche in reducing the space of possible classifications. Consider the task of matching a photo of a criminal to a database of millions. While returning a single match with prosecution strength accuracy would be ideal, reducing the matches to a handful of twenty or thirty would be nearly as useful. At that point the problem becomes tractable for a human, and our propensity for pattern matching can now supplement the learner. In other words, our best interest may not always lie in trusting the smartest learner in the room; the hubris of certainty ignores our own innate abilities of classification.

3. PRELIMINARY RESULTS

3.1 Disjoint Learning

Learning with disjoint sets introduces some caveats to data analysis. As seen in figure 1, our baseline disjunctive HyperPipes consistently outperforms both naive bayes and the original HyperPipes algorithm. However, our disjuntive learner doesn't exactly play fair, hedgings it's bets with multiple predicted classes. Note as well that when we use our centroid method for single classification, we fail to consistently beat naive bayes.

However, when disjunctive HyperPipes cheats, it does so with a conscious. Figure 2 plots the average size of the sets returned for the primary-tumor dataset as we incrementally learn across the data. As our baseline results demonstrate, we return on average no more than 20% of potential classes. As a result, we trade in single classification for a doubling of accuracy (figure 1).

3.2 Breaking the Ties

While we believe the mutliple class result of HyperPipes is an interesting discovery we understand the need for a complete classification system. We have come up with two methods for attempting to classify the result of MultiPipes into a single class and have come up with two different approaches. One approach was discovered when attempting to relieve outliers.

3.2.1 Weighted Distance from Mean

There are two methods we use for calculating the distance from the mean in our algorithm. As stated above in "Fixing Outliers" we noted that an attribute range scores a 1 if it falls within the min and max for that attribute in a class. Our modification, as stated above was to use a normalized calculation to determine a value between 0 and 1 based on the attribute values distance from the mean attribute value in that class. Two calculations were discussed and they are as follows:

1. Weighted Distance From Mean 1: In this calculation it was decided that the distance from mean should be calculated as:

$$WghtDist = \frac{(Max - Min) - |(Mean - Val)|}{(Max - Min)} \quad (1)$$

   where Max, Min, and Mean are the max, min, and mean values for the attribute in the HyperPipe for the class currenlty being scored and Val is the current attribute value in the row being classified.

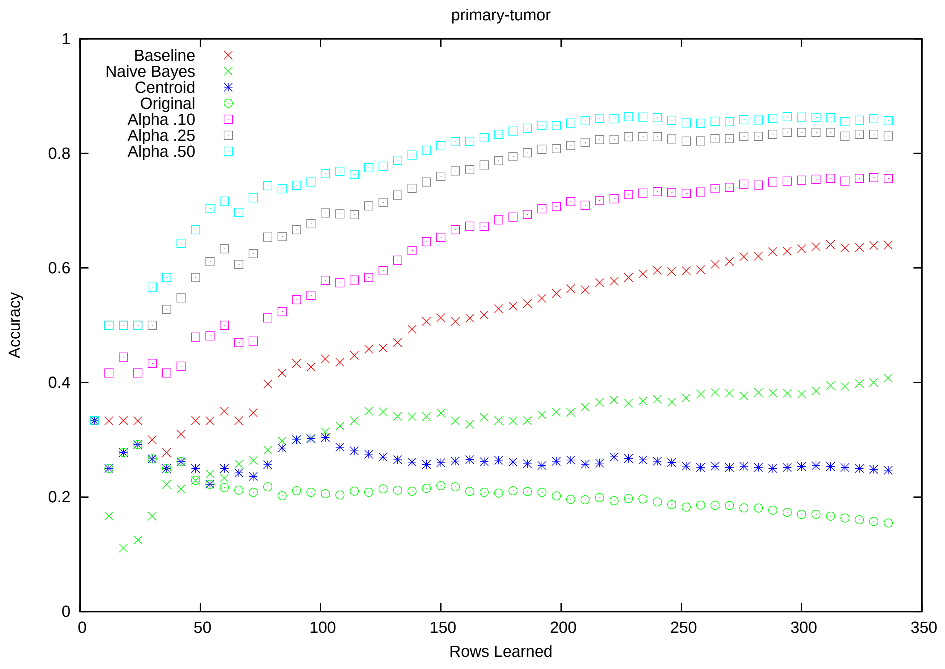

Figure 1: Incremental results from a discrete dataset with 22 class values

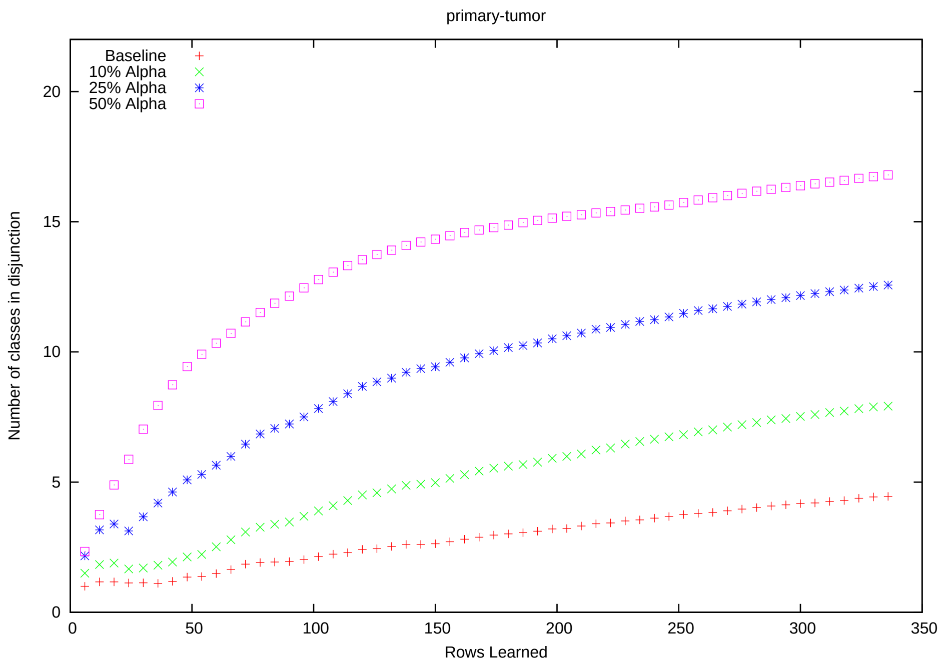

Figure 2: Average number of classes contained within disjunction for incremental learning

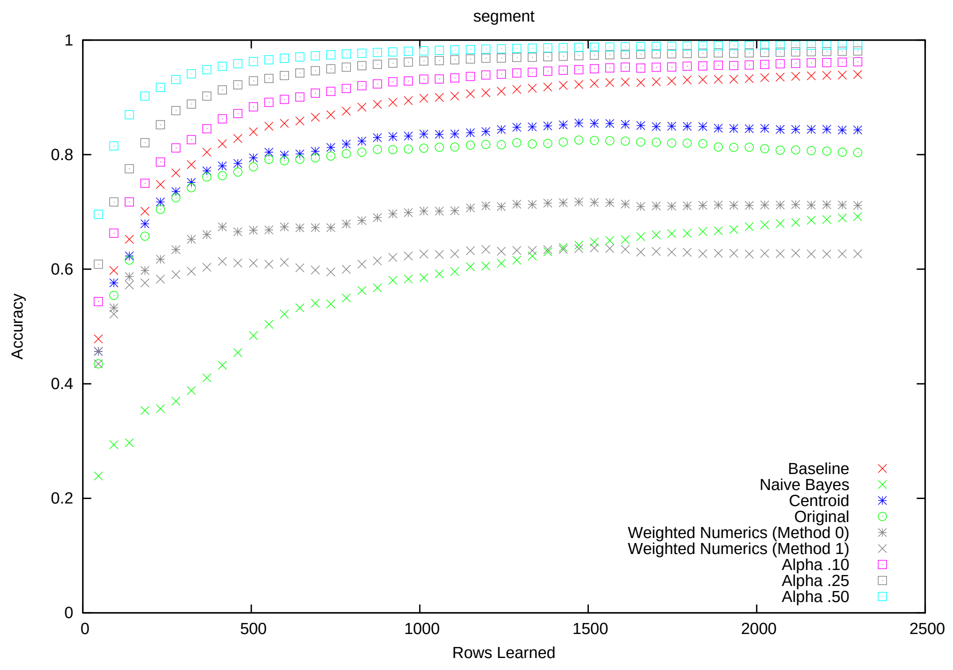

Figure 3: Incremental results from a numeric dataset with 7 discrete classes

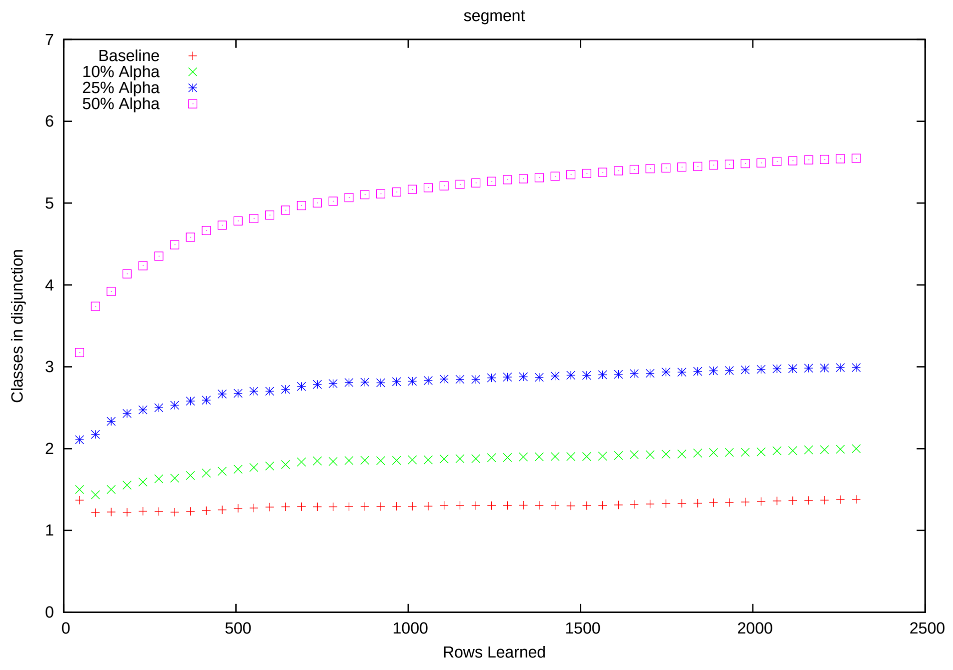

Figure 4: Average number of classes contained within disjunction for numeric attribute dataset

2. Weighted Distance From Mean 2: The calculation in this method is slightly different from the one above:

$$BigGap = max((Max - Mean), (Mean - Min)) \quad (2)$$

$$WghtDist = \frac{(BigGap) - |(Mean - Val)|}{(BigGap)} \quad (3)$$

where Max, Min, and Mean are the max, min, and mean values for the attribute in the HyperPipe for the class currenlty being scored and Val is the current attribute value in the row being classified.

graph of weighted distance classification accuracy

3.2.2 Centroid Assumption

One method we implemented for classifying to a single class was a centroid assumption function we created. The idea behind this algorithm is that when a tie is formed we calculate the class which is believed to be at the center of the returned classes. This center class is determined by scoring each HyperPipe's attribute overlap with other HyperPipes. For each HyperPipe we sum the number of attributes in that HyperPipe whose min and max overlap with another classes min and max. In other words, for a single attribute in a HyperPipe the maximum score for that attribute is the number of HyperPipes minus 1. The maximum score for the HyperPipe as a whole is the number of attributes times the number of hyperpipes minus 1. After calculating this sum for each HyperPipe the class within HyperPipe with the highest score is chosen as the classification. Pseudocode: graph of centroid learning results

Program 4 HyperPipes FindCentroid Pseudo Code.

```
procedure FindCentroid($HyperPipes$)
    $BestScore := 0$
    for all MainPipe in HyperPipes do
        $CurrentPipeScore := 0$
        for all ( doTestPipe in HyperPipes)
            if $MainPipe! = TestPipe$ then
                for all Attr in MainPipe do
                    if $((MainPipe.Attr.min > TestPipe.Attr.min) \&\& (MainPipe.Attr.min < TestPipe.Attr.min)) || ((MainPipe.Attr.max < TestPipe.Attr.max) \&\& (MainPipe.Attr.max > TestPipe.Attr.max))$ then
                        $CurrentPipeScore + +$
                    end if
                end for
            end if
        end for
    end for
end procedure
```

3.3 Casting a wider net

During testing it was determined that MultiPipes could easily be used as a reliable ???narrower??. We found that on some datasets we were able to acheive a very high reliability. That is, our resulting output set contained the actual class more than 90found that on some other datasets the reliability was quite less. When searching for a solution we realized we needed a way to expand the returned dataset. This would increase the chance that the desired class is included in the returned set. The reason to expand the returned set is so we can improve the reliability to say with greater confidence that the returned set is complete and accurate. Our next step is to train the learner as to which alpha value it should use. We would specify a goal reliability and expand the alpha slowly until that reliability is reached. In the Classify function (Program 2) above you will notice that we add anything to the BestClass list that has a score equal to the best score found. Each score is a fraction and the best possible score is 1. When you apply an Alpha of .1 it basically states that if the the current score is greater than the current best score minus .1 append it to the list. This requires one run through the list of HyperPipes to find the best score. It then calculates the score to beat as $ScoreToBeat = BestScore - Alpha$. On its second run through the HyperPipes any HyperPipe with a score greater than or equal to the $ScoreToBeat$ is included in the resulting set.

Results of expanding alpha (graph)

Analysis of growth in enclosure with alpha changes

4. CONCLUSIONS

We've demonstrated the potential for large improvements in accuracy in exchange for a narrowing of potential classes. We've also presented a possible means for single-classification with HyperPipes that performs better than the original algorithm but not as well as Naive Bayes in the currently explored datasets. However, further analysis is needed to better ascertain what improvements can be had when MultiPipes is run on data with a larger space of classes or columns. Also, there is much to be done involving the combination of numeric and discrete attributes and whether better outlier detection can be achieved.

5. REFERENCES

[1] J. Eisenstein and R. Davis. Visual and linguistic information in gesture classification. In *ICMI*, pages 113–120, 2004.