Math Expression Recognition - Parsing Report

Eric Falkenberg
Email: exf4789@rit.edu

I. Design

The design for my parser is based on the parser formulated in *MST-Based Visual Parsing of Online Handwritten Mathematical Expressions* by Hu, Zanibbi [1]. Despite the existence of some parsers that performed a bit better, this design was chosen because it did not rely on the creation or existence of a grammar and yet still performed pretty well. This design constructs a line-of-sight (LOS) graph between symbols, labels edge weights based on a relationship classifier's output, and then constructs a maximum spanning tree using Edmond's algorithm in order to produce the final output relationship graph. Said relationship classifier considers any two pairs of strokes connected via an edge in the LOS graph and classifies the relationship into one of seven categories via a random forest. The seven relationships described in the paper are as follows:

1) No Relationship
2) Right
3) Subscript
4) Superscript
5) Above
6) Below
7) Inside

The design for my segmenter is based on *Segmenting Handwritten Math Symbols Using AdaBoost and Multi-Scale Shape Context Features* by Hu, Zanibbi [2]. This design considers pairs of strokes in time order and utilizes geometric features, multi-scale shape context (MSSCF) features, and classification probabilities as input to an AdaBoost classifier in order to determine whether two successive strokes should be split or merged.

Finally, the design for my classifier has changed since the original submission. I've since migrated away from the use of an HMM and am now utilizing a random forest. The original features described in *HMM-Based Recognition of Online Handwritten Mathematical Symbols Using Segmental K-means Initialization and A Modified Pen-up/down Feature* by Hu, Zanibbi [3] are still used in this classifier but are truncated to 50 features. During the re-sampling of points, it is ensured that the points are re-sampled such that 50 points are evenly distributed along the original trajectories. Along with this, geometric and MSSCF features are utilized to improve classification scores.

II. Preprocessing and Features

The features to compare two symbols described in [1] were three sets of parzen window shape context (PSC) features. Along with this, a number of geometric features were utilized. I, however, used two sets of MSSCF and a subset of the original paper's described geometric features. There wasn't any intelligent reasoning for these omissions other than trouble with the implementation of those left out. The MSSCF is made up of 12 angle bins and 5 distance bins. These 5 distance bins are defined by the distances $\frac{1}{16}$, $\frac{1}{8}$, $\frac{1}{4}$, $\frac{1}{2}$, and 1. Two of these were formulated. The first of the two describes the MSSCF for the parent symbol and the second describes the MSSCF for the child symbol.

A number of geometric features were also used. These are as follows:

1) Distance between bounding box centers.
2) Distance between the centers formulated by averaging points in a symbol.
3) Maximum distance of any two points between the two symbols.
4) Horizontal offset between the end of the first symbol and the beginning of the second symbol.
5) Vertical offset between bounding box centers.
6) Writing slope.

These then were normalized to the $[0, 1]$ range.

The features described above were also used in segmentation. One set of MSSCF and 6 geometric features were generated based on the two candidate strokes. The segmenter also utilized raw class probability values outputted by the random forest. One vector was produced by feeding just the first stroke of the pair into the classifier and another vector was produced by feeding both strokes in together to be interpreted as the same symbol. Given 101 classes that means the number of features required by the segmenter was 268. No dimensionality reduction was run though it will be good to research into its benefits in the future.

III. Parsers

Algorithms 1 and 2 describe, in pseudo-code, how the two parsers provided with my submission function. They perform the act of parsing similarly but deal with their inputs differently. The first of the two reads in *inkml* files directly and performs parsing based on the grouping data provided by the files. The second reads in raw stroke data, performs segmentation with the segmenter formulated above, and finally parses based on the output of the segmenter.

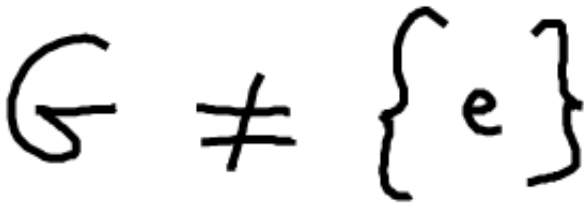
Luckily, due to some design decisions early on, it is pretty trivial to pass data back and forth between the segmenter and parser. A group class was formulated to store information associated with a group read from a crohme file so the segmenter packs segmented strokes into a group object and

Data: type of dataset to train / test on
Result: directory of .lg files
load pre-trained parser;
determine files to evaluate;
read files;
while *files left to process* **do**
    get next file;
    obtain group info;
    create line of sight graph for list of groups;
    label edge weights with parsers relationship classifier;
    run Edmonds algorithm to get MST;
    for *edge in MST.edges* **do**
        write relation to output directory under the name specified by the input file
    end
end

Algorithm 1: Parsing algorithm using ground truth segmentation data as input. As described, the maximum spanning tree is formulated to prune away unnecessary edges. The parser's relationship classifier is used to produce a vector of class probabilities. The maximal one is chosen and the probability output is used as the edge weight. Thus, relationships that have higher confidences are more likely to make it into the final MST.

Data: input files
Result: directory of .lg files
load pre-trained parser;
read files;
while *files left to process* **do**
    get next file;
    use segmenter to create group info from raw stroke data;
    create line of sight graph for formulated groups;
    label edge weights with parser's relationship classifer;
    run Edmonds algorithm to get MST;
    for *edge in MST.edges* **do**
        write relation to output directory under the name specified by the input file
    end
end

Algorithm 2: Parsing algorithm using raw stroke data as input. Once called upon to segment a group of symbols, the segmenter will consider them in stroke order. It will then use geometric features / msscf / classifier values to determine whether to split / merge the strokes with AdaBoost. Finally, once groups have been formed, it will run the stored classifier in order to identify the now isolated symbol.



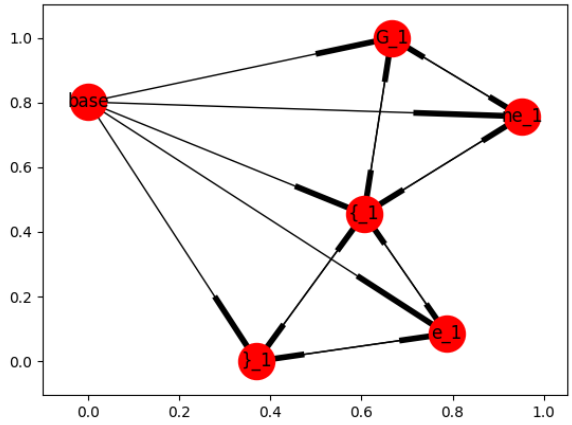


Fig. 1: Line of sight graph

passes it to the parser. Once there, the parser can interpret the group object ignorant of where it actually came from.

Unfortunately, the parser has a very obvious bottle-neck. The line-of-sight graph generation algorithm is extremely slow. It is possible that there is an error in implementation but I think the more likely scenario is that run-time slows down exponentially as the number of symbols increases and slows down even more if there is a large number of symbols that do not have line of sight with one another. I am currently using a variant of Bresenham's line algorithm to draw traces from every point in one symbol to every point in another. If none of these reach their target, then there is no line of sight. This makes the worst case for the algorithm the one where the two symbols being considered do not have line of sight with each-other. This seems to be a large problem with equations where there exist fractions with complicated numerators / denominators given that no symbol below the fraction line can see above it and vice-versa. Since we do this for every pair of symbols in the expression, the time complexity of this algorithm is n^2 for every pair in g^2 where n is the number of points in a symbol and g is the number of groups. Further research should be done on how to better implement this. It is possible that reducing the resolution (i.e. number of points in a stroke / symbol) is preferable

IV. Results and Discussion

In terms of segmentation, Figure 2 describes the results the Adaboost MSSCF segmenter obtains on a $2/3|1/3$ train|test

split. This segmenter used the described random forest isolated symbol classifier which was trained on the entire crohme dataset from project 1. When trained on the $2/3$ $1/3$ split, this classifier achieved a 75% classification rate which while not great is still an improvement on the original submission. Looking over segmentation errors, it is clear that the segmenter will often mis-classify merge as split if it is given two strokes that look like symbols on their own. For instance, the symbol F might be split into r and $-$. Another common one I found was C and $-$ for the symbol $\in$.

In terms of parsing, Figure 3 describes the results of the Random Forest MST parser on ground truth data and Figure 4 describes the results of said parser on the segmentation data described above. As a fair warning, the results specified were generated from a random sample of 100 files from the original test set. This was done because I was hitting runtimes that weren't going to help me hit my deadline (15 hours for the full test set). The results may not be fully accurate to how my parser performs on the full test set but I can still hypothesize that something is wrong with it from these results. It is certainly possible that this random sample is unlucky and my parser actually performs much better on the overall set but I am willing to hazard a guess that that is not the case.

A common error found when looking through results was that something like $\{x\}$ would give two relations $\{$ right x and $\{$ right $\}$. I thought about adding rules in to disallow certain ambiguous interpretations but it started to seem like a stochastic context free grammar would be the solution to that particular problem and accepted it as a shortcoming of a grammar-less system. That said, my parser definitely doesn't perform as well as the one described in [1] so there are many things that I can improve upon. I think the biggest shortcoming of my classifier is that I mistakenly didnt inject any no relation samples into my training set. This shortcoming means that there is no reason for my parser to identify something as no relationship. Without a rejection class, it is likely that many relationships were chosen just due to the fact that it couldn't be rejected. The recall percentages seem to back up this claim but further research is required.

V. References

References

[1] L. Hu and R. Zanibbi, "Mst-based visual parsing of online handwritten mathematical expressions."

[2] ——, "Segmenting handwritten math symbols using adaboost and multi-scale shape context features."

[3] ——, "Hmm-based recognition of online handwritten mathematical symbols using segmental k-means initialization and a modified pen-up/down feature."

	Recall	Precision	F-Measure
Objects	81.21	76.19	78.67
+ Classes	72.68	68.11	70.32
Class/Det	89.39		

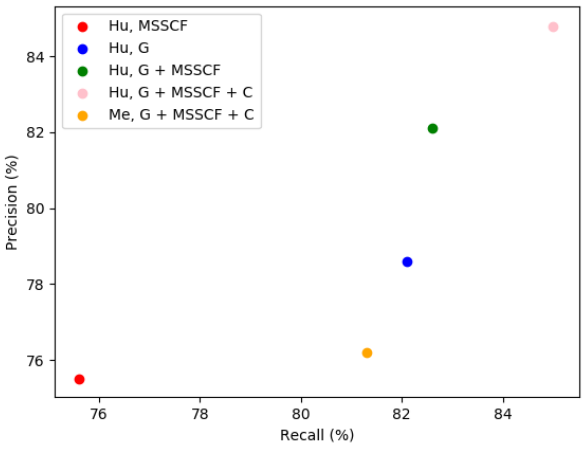


Fig. 2: Segmentation Results

	Recall	Precision	F-Measure
Objects	15.28	43.04	22.55
+ Classes	13.58	38.26	20.05
Class/Det	88.89		

Fig. 3: Parsing Results based on ground truth

	Recall	Precision	F-Measure
Objects	N/A	N/A	N/A
+ Classes	N/A	N/A	N/A
Class/Det	N/A		

Fig. 4: Parsing Results based on segmeter output