Student: Ankan Mookherjee
Advisor: Dr. Richard Zanibbi
Machine Learning and AI Labs, Rochester Institute of Technology
Handwritten Math Recognition, Pattern Recognition

Design

Our segmentation method is heavily based off of the Ada-Boost and multi-scale shape context based method outlined in Hu et. al. [2] and its references to Winkler et. al. We chose to model this method because it produces impressive results with a simple yet effective method and interesting features.

A lack of proper preprocessing and weak features resulted in very poor classification rates in our previous project, so much of the improvements in this round were focused on those areas. Our symbol classification is still based around a basic random forest classifier.

We added several preprocessing steps including stroke smoothing and expression size normalization in order to make the data more uniform.

Our previous set of features relied heavily on a simple 9 bin square distribution. Unfortunately, this did not provide enough information about the shape of symbols to successfully classify them. By replacing that distribution with the shape context distribution - a similar but much more descriptive feature - in addition to adding the internal angle and a few other features, our system should be better able to discriminate between symbols.

Our parsing method is heavily based on the baseline extraction method outlined in *Recognizing Mathematical Expressions Using Tree Transformation,* by you et. al.[1] We chose this method because of its straightforward yet effective approach.

Little was done to the previous segmentation and classification systems. The random forest classifiers in the symbol classification were changed from 100 trees with a max depth of 15 to 1000 trees with a max depth of 10. In addition, one of the classification features, the vector from the bounding box center to the average center, was changed from an angle and a magnitude to its x and y components. We feel this is a better description for the classifier to use.

Apart from these small changes, our system is unchanged apart from the addition of a parsing phase.

Preprocessing

Several methods were used to preprocess the data before segmenting and / or classifying.

All expressions were normalized to [0, 200] in the y direction while maintaining the aspect ratio.

All strokes were smoothed using the method mentioned in Hu et. al: each point (except for the first and last points) is averaged with the next.

Before segmentation, all strokes are normalized to 30 points. For strokes originally containing more than 30 points, random points are chosen and averaged with the next point in the stroke. This average point replaces the two points. For strokes containing fewer than 30 points, random points are duplicated until the stroke has 30 points.

PCA is performed on the segmentation features. The number of components is reduced from 206 features to 100.

Before symbol classification, the total number of points in the strokes of each *symbol* is normalized to 30 points. This is done using the same method as above. The relative ratio of points in each symbol is preserved as closely as possible.

The point normalization in both the segmentation and symbol classification steps are done in order to make several features more uniform, namely the shape context features and the internal angle features.

Features

Segmentation Features

Many of the features used in the segmentation classifier are outlined in Hu et al. The most interesting feature is the multi-scale shape context features.

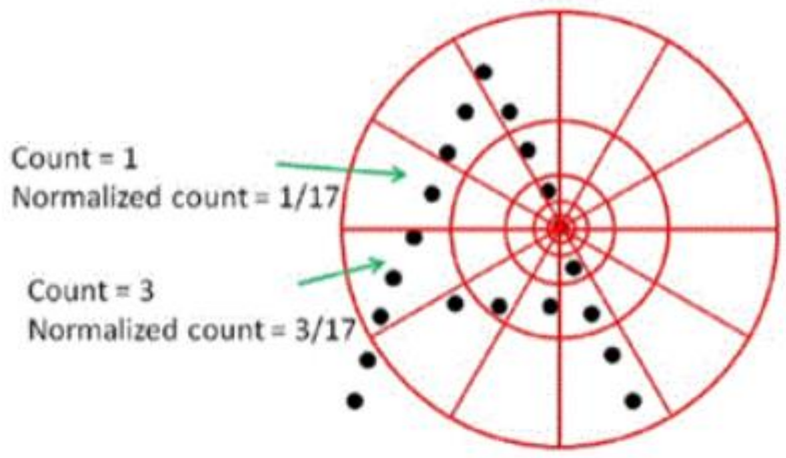


Fig. 1. Example: shape context feature computation.

This feature is a circle which encompasses an area of interest and breaks it up into a number of bins corresponding to both angle and distance from the center of the circle. Like Hu et. al, we used 12 equally spaced angle bins each with 5 exponentially increasing distance bins, for a total of 60 features per shape context.

For segmentations, three different shape context features are used. In all instance, the center of the circle is the center of the bounding box of the current stroke. Whereas the radii of these features are based upon maximal distances from the strokes to the circle center in Hu et. al., we used the radius of the bounding box of the strokes.

First, the **stroke pair shape context** encompasses the stroke of interest and the following stroke. The second, the **local neighborhood shape context** considers the current stroke and the three nearest neighbor strokes. Finally, the **global shape context** encompasses all of the strokes in the expression.

The additional features we used can be broken up into two groups: current stroke features, and stroke pair features.

The current stroke features are simple metrics relating to just the current stroke.

- The number of points in the stroke (before normalization)
- The distance between the first and last point in the stroke.
- The x and y components of that distance (Δx, Δy)

The stroke pair features are computed on both the current and previous stroke, and the current and next stroke. For each of these features, if there is no previous or next stroke, all features are -1. Otherwise, all features are positive values. The non-current stroke (previous or next) will be referred to as the pair stroke.

- Original stroke length of the pair stroke.
- The distance between the centers of bounding boxes of the current and pair stroke.
- The horizontal overlap of the bounding boxes.
- The area of the overlap of the bounding boxes.
- The ratios of width and height of the bounding boxes.
- The absolute difference in initial x and y values of the bounding boxes.
- The distance between the average centers of the current and pair stroke.
- The minimum and maximum distance between the current and pair stroke.
- The writing slope (angle between last point of first stroke and first point of next stroke).

In total, there are 206 features for each segmentation decision. After performing principal component analysis, that number decreases to 100.

<u>Symbol Classification Features</u>

Shape context features are also used in symbol classification. The context is the entire symbol, so the bounding box encompassing all of the strokes in the symbol is used.

We also added the internal angle and horizontal angle features. Both relate to the angles formed by consecutive points in the stroke. For the internal angle feature, for each three consecutive points (except the first and last points), calculate the absolute value of the cosine of the angle formed by these points. For the horizontal angle feature, for each two consecutive points, calculate the angle (0-360 degrees) from the horizontal formed by these points. Since all symbols are normalized to 30 points, the result is two feature vectors of length 27 for each symbol.

An additional interesting feature is the distance and angle between the bounding box center and the average center. This feature provides useful information about the symmetry of the symbol by pointing towards the center of gravity of the symbol. The distance is normalized by the radius of the bounding box to account for differences in symbol size.

Several additional features are used as well.

- The number of strokes in the symbol.
- The total number of points in the symbol before normalization.
- The aspect ratio of the symbol.
- The number of cusps in the symbol.
- The number of intersections in the symbol.
- The direction from initial point to the last point in the symbol (North, East, West, South)

Segmentation and Classification

Segmentation

Our segmentation method is primarily based on the Ada-Boost based segmenter in Hu et. al. When presented with the strokes in an expression and the stroke of interest, the classifier will spit out either “SPLIT” or “MERGE”.

To do the classification, the segmenter goes through the strokes of an unsegmented expression in order. The first stroke is added to the current symbol. For each stroke after the first in the expression, the classifier is used. If the verdict is “MERGE”, the stroke is simply added to the current symbol. If the classifier says to “SPLIT”, the current symbol is saved as a final symbol, a new current symbol is started, and the current stroke is added to that new symbol. When all strokes have been considered, the result is a list of segmented symbols.

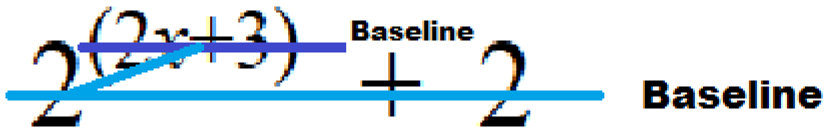
Symbol Classification

Our symbol classification is still based around the random forest classifier. The ensemble is comprised of 100 decision trees, each with a maximum depth of 15.

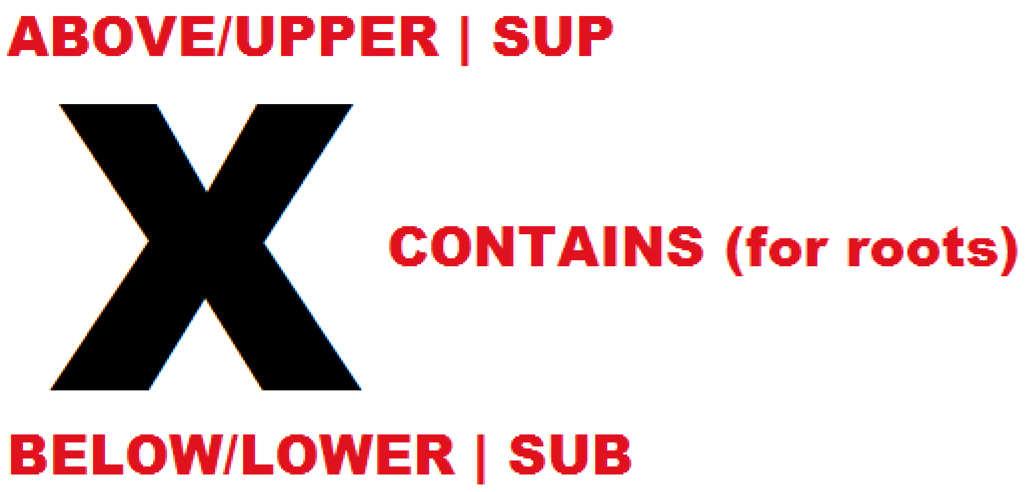
Parsing

As mentioned, our parsing system is based on the baseline extraction method. The method, as implemented, is nearly verbatim from the paper, with the exception of a few minor changes to partitioning parameters. Parsing takes place after segmentation and classification are performed, so the parser has information about each symbol, including what symbol it is.

The algorithm is straightforward in theory. The basic idea is to find the baseline in an expression, partition symbols off of the baseline into one of a number regions around nearby symbols, then recursively find baselines of those regions. A baseline in this context is an ordered list of the main symbols in the expression - those that will directly interact with each other after resolving any non-baseline factors.



The regions in this system are as follows: EXPRESSION, ABOVE, BELOW, UPPER, LOWER, SUP, SUB, and CONTAINS. EXPRESSION is for a root node - the nodes within EXPRESSION are baseline symbols. ABOVE and UPPER, and likewise BELOW and LOWER are similar, but UPPER and LOWER are used for variable range symbols such as **Σ**. SUP and SUB denote super- and sub-scripts. CONTAINS is strictly for the contents of roots. The system also contains temporary regions TLEFT and BLEFT. These regions are for symbols located to the left of the current symbol, and are later appropriated to the corresponding SUP or SUB region of the previous baseline symbol.



This system has only a basic understanding of the grammar of expressions. Symbols, and the corresponding logic for handling them, are broken up into several groups: open bracket, non-scripted, root, variable range, ascender, descender, centered, and closed bracket (included in centered for most purposes).

Brackets and roots are self explanatory. Non scripted symbols are those which can not have super- or sub-scripts. This group mostly includes operators and relations. Variable range symbols are operations which may have a variable range, such as a summation or an integral. ‘Ascender’ and ‘descender’ describe the shapes of symbols. Ascenders are symbols that are generally taller than others. Capital letters, numbers, and letters with high strokes such as ‘d’ or ‘l’ fall into this category. Descenders are those symbols with strokes that may go below the baseline, such as ‘g’, and ‘y’. All other symbols are classified as centered.

The purpose of these various classifications are to determine two things. First, what sort of regions are permissible for a particular symbol. It makes no sense for an open bracket or a plus sign to have a superscript, for example. Secondly, it determines where these regions are located relative to the symbol. A descender like ‘y’, for example, has a long tail below the baseline. A subscript of ‘y’ likely won’t be all the way down below the tail, but rather adjacent to the tail. The opposite is true for an ascender like ‘d’: a superscript probably won’t be above the high stroke, but rather alongside it.

Apart from these considerations, the system has no understanding of the grammar, and will happily create relations that make no sense. This gives the system the ability to parse incomplete or incorrect expressions, but at the expense of the ability to dismiss an incorrect parse based on its meaning.

Results

Fold 1

	Recall	**Precision**	**F-Measure**
Segmentation	76.29%	65.75%	70.63%
Segmentation + Classification	57.63%	49.66%	53.35%
Classification	~75.54%	~75.53%	75.53%
Relations	61.59%	61.64%	61.61%

Fold 2

	Recall	**Precision**	**F-Measure**
Segmentation	75.36%	63.22%	68.76%
Segmentation + Classification	56.55%	47.42%	51.58%
Classification	~75.04%	~75.00	75.02%

Relations	60.23%	60.54%	60.38%

<u>Fold 3</u>

	Recall	**Precision**	**F-Measure**
Segmentation	76.71%	66.23%	71.09%
Segmentation + Classification	58.03%	50.11%	53.78%
Classification	~75.65%	~75.66%	75.65%
Relations	61.72%	61.95%	61.83%

Discussion

Overall, the results were consistent and acceptable, but disappointing. At some point, we must have introduced a bug to the segmentation features, because the rates are significantly lower (around 10%) than previous, and the difference between precision and recall is much greater. Unfortunately, we lack the time to investigate.

The change to the random forest architecture in the symbol classifier did not seem to have a beneficial effect on the rates. The F-Measures for classification are 3-4% lower than before, though they remain very consistent, both across folds and between recall and precision rates.

The parser performed fairly well, correctly labelling approximately 61% of relations in the dataset. While not terrible, it is unlikely that a system with such a parser would be very useful, as it is very likely to mistake at least one relationship in an expression. Such a mistake, especially early in the expression, can lead to very bad results.

The main issue the system faced was correctly distinguishing baseline symbols from superscripts and subscripts. This is not surprising, since super- and sub-scripts are often very difficult to distinguish based on geometric features alone. In different contexts, identical pairs of symbols could be meant to have different relationships.

There are many possible additions and modifications that can improve upon this parser. Tweaking the individual partitioning parameters could help the system to better classify which region symbols belong to. It could be possible to automate this process, varying the parameters and finding the result with the best performance.

A major improvement could be possible with the addition of a grammar. With a grammar, the system could detect impossible relationships and avoid many possibilities. It is very likely that a well-designed and well implemented grammar system could improve the performance of this algorithm significantly.

References

[1] R.Zanibbi, D.Blostein and J.R.Cordy, "Recognizing mathematical expressions using tree transformation," IEEE Trans. PAMI, vol.24, no.11, pp.1455-1467, Nov. 2002

[2] Lei Hu; Zanibbi, R., "Segmenting Handwritten Math Symbols Using AdaBoost and Multi-scale Shape Context Features," Document Analysis and Recognition (ICDAR), 2013 12th International Conference on , vol., no., pp.1180,1184, 25-28 Aug. 2013
doi: 10.1109/ICDAR.2013.239

[3] Koschinski, M.; Winkler, H.-J.; Lang, M., "Segmentation and recognition of symbols within handwritten mathematical expressions," Acoustics, Speech, and Signal Processing, 1995. ICASSP-95., 1995 International Conference on , vol.4, no., pp.2439,2442 vol.4, 9-12 May 1995
doi: 10.1109/ICASSP.1995.479986

[4] F. Alvaro, J.A. Sanchez, and J.M. Bened. Recognition of on-line handwritten mathematical expressions using 2D stochastic context-free grammars and hidden Markov models. Pattern Recognition Letters, 2012