CSCI 727 - Pattern Recognition - Project Phase 03

Dharmendra Hingu(dph7305@g.rit.edu)
Sanjay Khatwani (sxk6714@g.rit.edu)

May 4, 2018

1 Design:

The design of the segmenter has been improved in this part of the project in order to get better accuracy on parsing task. As discussed in the improvement and future work in the previous project documentation, the detector part of the segmentation has been replaced by a binary detector that provides either a *merge* or *split* decision for two strokes rather than combining them using some score returned by the classifier. The binary detector is trained and tested on new geometric features described below.

The new set of features (20) were extracted for this new segmenter as described below:

1. **Distance between bounding box centers** [1 feature]: This feature computes and returns the smallest distance between the centers of bounding boxes of the traces under consideration. We normalize this by a factor that is maximum of height or width or 1 up to 4 decimal point precision.

2. **Bounding box overlap** [2 features]: This feature computes and returns the overlap between the bounding boxes of the pairs of strokes. We compute the vertical and horizontal overlap.

3. **Distance between the centroids** [1 feature]: This feature computes and returns the smallest distance between the centroids of the traces under consideration. We also normalize this feature value by a factor that is maximum of either height, width or 1 up to 4 decimal point precision.

4. **Distance between first points** [1 feature]: This feature computes and returns the largest Euclidean distance between the pair of first points of all the strokes under consideration normalized by the same factor as previously described.

5. **Distance between last points** [1 feature]: This is similar to above but considers last points as opposed to first points.

6. **Offsets** [4 features]: This feature computes and returns the 4 offset distances, *left horizontal offset*, *right horizontal offset*, *top vertical offset*, *bottom vertical offset*. The *left horizontal offset* is the distance between the left lines of the two bounding boxes, *right horizontal offset* is the distance between the right lines of the two bounding boxes. Similarly, *top vertical offset* and *bottom vertical offset* is calculated.

7. **Histogram of points** [10 features]: This feature computes and returns the histogram of points for given traces. 5 horizontal and 5 vertical. These features account for the density in $5x5$ region. The original symbol is divided into 5 parts horizontally and vertically. The number of points from traces is counted per grid cell and summed horizontally and vertically.

Improving the design of segmenter improved the F-score to 85.60 from 64.56 on the same training and testing set from *project 2*. There are two kinds of input for the parsers, symbols and strokes based on which we implement two parsers in this project, symbol level parser and stroke level parser. Both the parsers are built using a graph based approach, utilizing spatial relationships between symbols. None of the context free grammars have been used in the project, since graph based approaches using spatial properties are being explored and have shown promising results [3] and [.

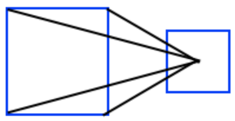
2 Preprocessing and Features:

In addition to adding features for segmenter, we also design and extract 72 features for the parser as well and the same features would be used by both symbol level and stroke level parser.

1. **Vertical bounding box distance** [1 feature]: This feature computes the vertical (y) distance between the bounding boxes of two traces/symbols under consideration. We normalize this by a factor that is maximum of height or width or 1 up to 4 decimal point precision.

2. **Horizontal bounding box distance** [1 feature]: This feature computes the horizontal (x) distance between the bounding box of two traces/symbols under consideration. We normalize this by a factor that is maximum of height or width or 1 up to 4 decimal point precision.

3. **Corner angle with center** [4 feature]: This feature computes the 4 angles made by a line drawn from each corner of the first bounding box to the center of other bounding boxes.



4. **Angle between bounding boxes** [1 feature]: This feature computes the angle between the two bounding boxes of traces/symbols under consideration.

5. **Distance between centers of bounding boxes** [1 feature]: This feature computes the Euclidean distance between the centers of the two bounding boxes of traces/symbols under consideration. We normalize this by a factor that is maximum of height or width or 1 up to 4 decimal point precision.

6. **Offsets** [4 features]: This feature is similar to one constructed for segmenter, described above, except that here it considers two symbols instead of strokes.

7. **Shape Context Feature [1]** [60 features]: Density-based features that count the number of points in an angular span starting from the bounding box center of the main symbol.

3 Parsers:

3.1 Symbol level Parser:

The input to symbol level parser is a set of symbols. The parser then finds the relationships between these symbols. To do this, the parser uses a relationship classifier which is used to obtain a relationship label and a score associated with that relationship between given two symbols. This classifier is trained on features described above to provide a relationship label and score. The set of possible relationship labels is the same as the ones defined in CROHME. These relationships include *Right, Inside, Sup, Sub, Above, Below*.

The parser builds a graph of 2-Nearest Neighbors for every input inkml file. The following algorithm was used to parse the input symbols.

```
for every input inkml file
    create a graph ($r$) of relationships by treating every symbol as a separate node
    for every node: $n$
        calculate Euclidean distances to the bounding box centers of all other nodes from the bounding box center of $n$
        order the nodes in increasing order of these distances and connect the current node to the two closest nodes
    for every edge: $e$
        obtain the features of the end point of $e$
        obtain a relationship prediction and confidence score from the relationship classifier
        label the edge with the relationship label and update the score of the edge as the relationship score
    compute Edmond's MST of the graph $r$
```

The overall complexity of the algorithm is $\mathcal{O}(n^3)$ for each inkml file. This can be explained since the running time of computing the Edmond's MST [5] is $\mathcal{O}(n^3)$, plus two extra step of building the relationship graph r which costs $\mathcal{O}(n^2 logn)$. The labeling of the edges takes $\mathcal{O}(n^2)$ time.

We start by building a relationship graph using the 2-Nearest Neighbors of every node. For every symbol s, an edge is added between s and 2 closest neighbors of s. Then for every pair of symbols connected in the graph, relationship classifier is used to obtain a relationship label and a score. This relationship classifier is trained using the relationships in the training set to predict the relationship between a pair of symbols. These labels and weights form the label and weight of the edge on the relationship graph. This graph gives the possible relationships between symbols, their labels and confidence/score for this relationship. This graph is then used to obtain a rooted Maximum Spanning Tree. This Maximum Spanning Tree is similar to the rooted label graph [2] where edges are labeled with the relationship. Edmond's Minimum Spanning Tree algorithm [5] is used for generating the MST. The weights of input graph are negated before giving it to Edmond's [5] algorithm since the algorithm computes a Minimum Spanning Tree and we need a Maximum Spanning Tree. The inverted weights ensure that a Maximum Spanning Tree is computed by the routine. The relationships are obtained from this Maximum Spanning Tree and each edge in the Maximum Spanning Tree represents a relationship between the symbols with the label same as the edge label. These relationships are then written to the output label graph file.

3.2 Stroke level Parser:

The input to the stroke level parser is a set of strokes in the expression. The parser first needs to segment and classify symbols before identifying the relationships between them.

The stroke level parser uses the new segmenter which uses the binary detector to identify symbols in the expression and then uses the symbol classifier to classify individual symbols. These symbols are then written into the output file in the label graph format. Then the symbol level parser is used with these symbol predictions to obtain relationships between the identified symbols. The Maximum Spanning Tree generated by the symbol level parser is then used to write out the relationships to the output file in the label graph format.

The time complexity of the stroke level parser is $\mathcal{O}(n^3)$ too. This is directly from the use of symbol level parser. The time complexity of the new segmenter is $\mathcal{O}(n^2)$.

4 Result and Discussion:

The data split has been reused from the *Segmentation phase* for this part of the project.

SYMBOL LEVEL PARSER

	Recall	Precision	F-Measure	Expression Recognition
Objects	100.00	100.00	100.00	100.00
Objects + Classes	100.00	100.00	100.00	100.00
Relations	70.07	95.05	80.67	32.61
Relations + Classes	66.87	90.70	76.98	29.88
Structure	-	-	-	32.61
Structure + Classes	-	-	-	29.88

STROKE LEVEL PARSER

	Recall	Precision	F-Measure	Expression Recognition
Objects	87.56	83.73	85.60	52.35
Objects + Classes	75.43	72.13	73.74	21.96
Relations	55.44	74.07	63.41	52.87
Relations + Classes	52.87	70.64	60.48	23.14
Structure	-	-	-	24.87
Structure + Classes	-	-	-	12.32

We see very high precision but drastic reduction in the expression recognition rate. One of the reason is, if at least one relationship is incorrect the whole expression is categorized as incorrect. Most of the errors are missing relationships between the symbols or strokes due to 2-NN implementation. In some cases, *above* and *below* relationships have been interchanged with the order of symbols also interchanged which is grammatically incorrect. For example, if there is an *above* relationship from *a* to *b*. Then the 2-NN model predicts *below* relationship from *b* to *a*. Similar confusion is observed for *sup* and *sub* relationships. Some of the segmentation errors are also propagating in the parsing phase, which is contributing the overall errors. We are considering spatial features and sometimes *right* relationship is treated as *sub* or *sup* due to variation in the location of the stroke.

4.1 Execution Time:

	Training time	Execution time
Classifier	44 mins	57 mins
Segmenter	24 mins	68 mins
Parser	27 mins	49 mins

4.2 Improvements and Future work:

The 2-NN approach has been used to generate the graph of relationships for every file. This approach is limited in the sense that relevant relationships might be missed and never even tried. Consider the example of $a^{(b+c)} + d$. In this case, the *Right* relationship between a and $+$ will never be discovered in the 2-NN case. For this reason, a Line of Sight graph [3] is preferred for parsing using spatial features. A Line of sight graph was tried and the results were observed to be degrading as compared to 2-NN for this design. The implementation is not yet finished, there is still some gap due to which the recognition rate is dropping. Once implementation correct it is proven to give better results. Another complementary feature to be extracted for Line of Sight graph is Parzen Shape Context features. Additionally, we have used lot of features in the parsing phase, we can use Principal Components Analysis to obtain fewer important features which may improve the performance. The good thing is segmentation results are really nice, a grammar-based approach would improve the performance.

5 References:

[1] Hu, L., & Zanibbi, R. (2016). Line-of-Sight Stroke Graphs and Parzen Shape Context Features for Handwritten Math Formula Representation and Symbol Segmentation. 2016 15th International Conference on Frontiers in Handwriting Recognition (ICFHR), 180-186.
[2] Mouchére, H., Zanibbi, R., Garain, U., & Viard-Gaudin, C. (2016). Advancing the state of the art for handwritten math recognition: the CROHME competitions, 20112014. International Journal on Document Analysis and Recognition (IJDAR), 19, 173-189.
[3] Hu, Lei & Zanibbi, Richard. (2016). MST-based Visual Parsing of Online Handwritten Mathematical Expressions. 337-342. 10.1109/ICFHR.2016.0070.
[4] Condon, Michael Patrick Erickson. (2017). Applying Hierarchical Contextual Parsing with Visual Density and Geometric Features to Typeset Formula Recognition (MS Thesis, Rochester Institute of Technology).
[5] J. Edmonds, Optimum branchings,Journal of Research of the National Bureau of Standards B, vol. 71, no. 4, pp. 233240, 1967.