CSCI 737 Pattern Recognition Project 2

Satwik Mishra Nikunj Kotecha

Contents

1 Design

Our objective is to design a model, such that given an inkml file from the CROHME dataset, extract the traces, apply preprocessing steps, perform segmentation, symbol classification and then finally output a labeled graph (.lg) file in object relation format. Figure 1, gives a high level overview of the entire pipeline, tracing the flow of information for a given inkml file (training/testing set), from start to end. Preprocessing, feature extraction and the design for a binary classifier (random forest) that serves as the basis for our segmentation model, was based on the paper by L. Hu and R. Zanibbi [1, 2]. The symbol classifier model, that is implemented using sklearns [3] Support Vector Machine (RBF kernel), was trained on the CROHME isolated symbol dataset, and is based on paper [4]. In comparison to the Project 1, We made minor changes to the symbol classification model by introducing multiprocessing in the feature extraction step for faster processing speed. Moreover, we included additional preprocessing steps like, interpolation, hook removal, duplicate filter and smoothing. All these steps are discussed in detail, in the sections below.

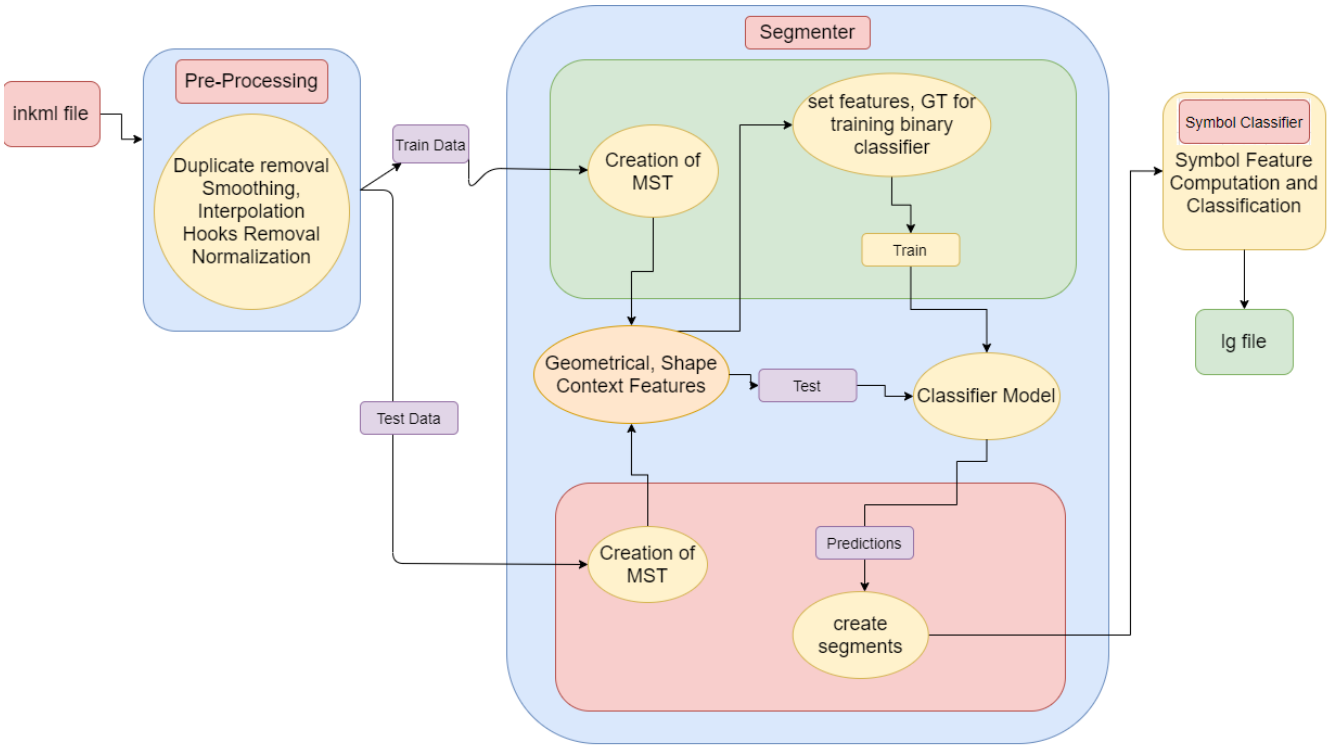

Figure 1: Overview of the design

2 Pre-Processing

2.1 Interpolation

The idea is to have a set of points in our stroke such that consecutive points are spaced apart by some constant distance. This distance d, is the average of the lengths of all the line segments in a stroke. For a point p_i, search the next point p_{i+n} such that the euclidean distance between those points is more than d. Then compute the (x_{new}, y_{new}), such that it is d distance away from p_i, using equations (1) and (2) [5] . All the previous points from p_i till (x_{new}, y_{new}) are removed. Start the search again from the new point added. Eventually we will have a list of points in the stroke where the consecutive points are are spaced almost equally apart. This helps reduce the variations caused due to different writing velocities.

$$x_{new} = \begin{cases} x_i + \sqrt{\frac{d^2}{k^2+1}} & \text{if } x_i < x_{i+1} \\ x_i - \sqrt{\frac{d^2}{k^2+1}} & \text{if } x_i > x_{i+1} \\ x_i & \text{if } x_i = x_{i+1} \end{cases} \tag{1}$$

$$y_{new} = \begin{cases} kx_{new} + y_i - kx_i & \text{if } x_i \neq x_{i+1} \\ y_i + d & \text{if } y_i < y_{i+1} \quad x_i \neq x_{i+1} \\ y_i - d & \text{if } y_i > y_{i+1} \quad x_i \neq x_{i+1} \end{cases} \tag{2}$$

2.2 Sharp Points

Authors of the paper [2] suggest that the hooks occur at very sharp points and are accompanied by a large change in angle in the stroke, due to erratic hand motion [5]. The beginning and the end points of a stroke are sharp points. Apart from these, in order to get the other sharp points, firstly compute the slope angles between consecutive points and then compute the ***changed angles (angles between consecutive lines)***. If the **turn angle** between consecutive lines is negative, it implies that there was a change in the writing direction, that point becomes a sharp point and a candidate for hooks. Turn angle is given by the product of the change angle between two consecutive lines.

2.3 Hook Removal

Hooks are basically the those parts in a stroke which are a result of pen down and pen up actions or random hand movements, and usually occur at the beginning or end of a stroke. These are characterized by three properties, the position, the size (small in length), and the change in angle [5]. Once the computation of sharp points is done, we can begin finding the hooks. If there are more than two sharp points we continue to find the hooks. In order to detect the hooks, line segments have to be defined between the first two and last two sharp points. Lets us name them seg_b and seg_e respectively. Let the slopes be β_b and β_e for the segments. There are two threshold values, one for the angle and one for the length. We compare the length of the line segmentsseg_b and seg_e and the angle difference between the slopes of segments seg_b and seg_{b+1}, and seg_e and seg_{e-1}. These values are then compared with threshold angle value (set at 90 degrees) and a threshold length

value i.e set at 3% of the diagonal length of the bounding box of the stroke. A segment is a hook if the condition following is satisfied: $\lambda_{b/e} \leq threshold_{angle}$ and $l_{b/e} \leq threshold_{length}$ [5].

(a) Stroke with out any preprocessing

(b) Stroke after subjected to interpolation, and removal of hooks

Figure 2: Preprocessing

2.4 Duplicate Point Filtering

We filter out the unique trace points getting rid of the duplicates. As those duplicate points don't add any extra information [1].

2.5 Smoothing

Smoothing helps to reduce the finger jitters caused while writing. Smoothing a stroke by moving an averaging window over the list of trace points, where, for every point we replace it with the average of the current previous and next point. For the first and last point we consider the next and previous only respectively. [1]

2.6 Normalization

To reduce the effect of different writing speeds, and to bring down the difference in range of strokes normalization is an important step. The y coordinate is normalized between $[0, 1]$ and the x coordinate is adjusted such that the aspect ratio is maintained [1]. For a given inkml file, while normalizing a stroke, we considered the min and max values of x, y coordinates across all the strokes, so that final normalized strokes maintain the same structure in the expression as before normalization. If one performs normalization to fit each stroke in a bounding box without considering the overall min and max of coordinates, then the overall structure is lost as all the segments will overlap each other in a bounded box of the same range.

3 Segmentation

3.1 Baseline Segmentation

In the baseline model, every stroke is considered as a single segment in itself, and then the classification is performed on it using our trained classifier model. The results for baseline segmentaion+classification is shown in Table 4.

3.2 Feature Selection for Binary Segmenter

To create a classifier that decides whether to merge two strokes or not, we need to first compute features for those two strokes. The features are discussed below.

Geometrical features: [1] For all the nodes that share an edges in the MST, the following features are computed. *Parallelity* (the angle between the vectors formed by the first and the second strokes), *distance between the centres* of the bounding boxes of the strokes, *horizontal and vertical distance* between the centres of the bounding boxes, *horizontal offset, vertical offset* (last point of first stroke and start point of second stroke forms the offset), the *distance between the centroids* of the bounding boxes and finally the *writing slope*, i.e. the angle made by the vector formed by the last point of the first stroke and first point of the second stroke, with the horizontal axis. Hence a total of 8 geometrical features.

Shape Features: [6] For two strokes that share an edge in the MST we first merged them into one bounding box, computed the new centres of the bounding box, then after shifting the origin to the centre of the bounding box, we recomputed the new coordinate points of the strokes. Then calculated the angles and distance between every pair of points in the bounding box. We tried to encode this information in the form of a *distance histogram* and an *angle histogram*. For the distance histogram, binning was done such that there were 5 bins in the range of $[log_{10}0.1, log_{10}2]$, and for the angle histogram, the range [0, 360] was divided into 13 bins with a step size of 30 degrees. Using the previously computed angles and distance values between every pair, these histograms were populated. We created another histogram that captures the log polar information of points in the bounding box, i.e. the distance and angle made by a point with the centre of the bounding box. The log-polar histogram is of size (5x13). On flattening out the histograms, we get a total of 65(log-polar histograms), 8(distance histograms) and 13(angle histograms). That is a total of 83 features capturing the information of the shape. Hence, for our binary classifier, geometrical and shape features combined, we compute a total of $83 + 8 = 91$ features.

3.3 MST and Binary Classifier

Algorithm 1 Modeling of graph, and creating feature set for training a binary model

```
# create mapping for trace id to trace group id
tgid_map = ast.literal_eval(tgid_map)
traces = get_traces(traces)
# perform preprocessing steps
traces = pre_process(traces)

# create Adjacency matrix graph with all the strokes as nodes
traces_graph = create_graph(traces)
# generate the Minimum Spanning Tree
mst = traces_graph.Kruskal_MST()
tid_to_tgid = get_map(tgid_map)

features, GT = None, None
for i in range(len(mst)):
  for j in range(len(mst)):
     if mst[i][j] > 0:
        # get the group id for a trace
        tg_id1, tg_id2 = tid_to_tgid[i], tid_to_tgid[j]

        # compute features for the nodes
        features = get_features(traces[i], traces[j])

        # If the group id is same for both nodes then GT=1 else GT=0
        # GT=1 means merge two nodes/strokes GT=0, means don't merge
        GT = 0 if tg_id1 != tg_id2 else 1

        # dump final set of features (later used training the model)
        write_to_csv(out, features, GT)
```

3.3.1 MST and feature vetor generation for Binary Classifier

As shown in Algorithm 1, individual traces in a given inkml input file, will first be converted to a graph, where each node corresponds to the stroke (referred by trace id), and the edges encode the euclidean distance between the centres of the bounding box of the two strokes. In the initial graph we create an undirected edge between every pair of strokes. Inorder to reduce the search space, we have to reduce the graph size, and to bring in this **constraint**, a minimum spanning tree (MST) is generated for the given graph (using Kruskal's algorithm). An MST is a spanning tree of a graph whose aggregation of the edge weights (i.e. euclidean distance in our implementation), is less than or equal to that of all the other possible spanning trees of that graph [7]. Because of this new constraint, instead of considering all the edges in the original graph we will only consider the edges

that belong to the MST, thereby reducing the search space. After creating the MST, for all the pairs in the MST, features are computed as mentioned in section 5.1 and the ground truth (GT) is set depending upon the mapping that can be derived from the given .inkml training file. If two nodes under consideration have the same group id, then the GT is set to 1 else 0. Binary classifier is then trained using the features generated. This model will serve as the binary classifier/detector, which is responsible for deciding if given two strokes, whether they should be merged or not. For a test input, we follow the same procedure as defined in Algorithm 1, however the only change is in the line 23 to 25, as we set the **prediction of the binary classifier** instead. Figure 4, gives a visualization of how the MST would be structured for an input .inkml file. In the given example, 0 and 1 on the edges show the classifier prediction, and finally the strokes would divided into 3 segments such that [1,2], [3, 4] and [5, 6] are together, and each segment would then be classified into a symbol using the improved model from Project 1.

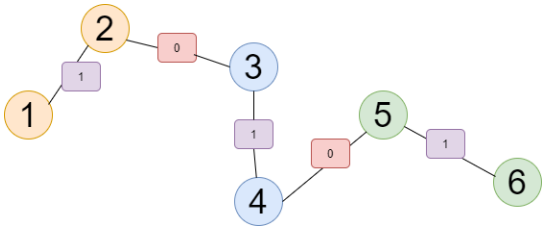

Figure 3: Example of an MST generated for an input set of strokes with trace ids 1, 2, 3, 4, 5 and 6. The edge label, 0 and 1 reflects the binary classifier prediction. 1 means merge and 0 means don't merge.

3.3.2 Binary Classifier

We trained our segmenter model using sklearns random forest classifier. We did a grid search over 'n_estimators': [50, 100, 200, 500, 1000], 'max_depth': [2, 6, 10, 20] and got the best parameters as [n_estimators: 500, max_depth:20]. The model was trained on the 70/30 split of the training data provided.

Parameter	Value
n_estimators	500
max_depth	20
class_weight	balanced

Table 1: Binary Classifier Final Parameters for Random Forest

4 Symbol Classifier

After segmentation, next step is to classifiy the segmented expression. We have incorporated additional preprocoessing steps i.e. interpolation, hook removal, smoothing, duplicate removal as mentioned in section 2, which were previously not a part of our Project 1. Another important addition was that of combining the preprocessing and feature extraction process with multiprocessing,

Which significantly decreases the overall time taken in the entire training process. After including the new changes, the model was retrained on the isolated symbol CROHME data set.

Parameter	Value
C	100
gamma	auto
class_weight	balanced
kernel	rbf

Table 2: Classifier Final Parameters for SVM (RBF Kernel)

5 Results and discussion

5.1 Test/Train Split

Initially we are counting the number of symbols across all the files. Then sorting the symbols according to its frequency of occurrence in ascending order. All the files which have that symbol are filtered out and then split into 70/30 ratio into training, testing set respectively. This process of sorting, filtering and splitting is repeated again for the remaining symbols in the remaining files. The objective of doing this is to minimize the KL divergence (D_{KL}) and get a value close to 0 (the lower the value, lower is the divergence) as shown Table 3. D_{KL} gives an idea of how different one probability distribution is from the other and is given by equation (3), where P and Q are the discrete probability distributions. Table 3, first row shows the scores considering training as P and testing as Q and second row considering test as P and training as Q.

$$D_{KL}(P||Q) = \sum P(x) \log \frac{P(x)}{Q(x)} \tag{3}$$

Data	Sample Distribution	KL-Divergence Score
Train	5752	0.00358
Test	2826	0.00359

Table 3: KL-Divergence score for train/test split on Training data set

5.2 Results

Data	Classifiers	Recall	Precision	F-Score
Train	Object	65.91	46.74	54.71
	+Classes	22.87	19.78	23.14
Test	Object	65.11	45.85	53.81
	+Classes	27.15	19.12	22.44

Table 4: Results of Baseline model on the 70-30% split of training data

Data	Classifiers	Recall	Precision	F-Score
Train	Object	97.07	97.45	97.26
	+Classes	60.34	60.59	60.47
Test	Object	87.10	82.08	84.52
	+Classes	43.72	41.20	42.42

Table 5: Results of Random Forest binary segmenter model on the 70-30% split of training data

Bonus Data	F score	Recall	Precision
Object	86.28	82.23	84.26
+Class	42.15	40.22	41.17

Table 6: F1 score for Random Forest segmenter on bonus data

We used the inkml file provided in the CROHME data set for our evaluation with the helps of the tool provided lgeval and CrohmeLib. The distribution of the training and testing split is shown in Table 3. On comparing Table 4 and 5, we can see that the sophisticated segmentor performs significantly better than the baseline model, where we were considering every stroke as an independent symbol. For segmentation we obtain an F1 score of 97.26% and 84.52% on the training and test respective for our sophisticated data as compared to baseline model 54.71% and 53.81% for train and test set. Important point to note is the low precision(46.74%) for a baseline segmentatiom although there is a decent recall(65.91%). The reason for a decent recall could be, that in the given data set, a majority of segments are single strokes, however the precision is low, as the strokes that should be grouped together, get partitioned into different segments, thereby increasing the number of False Positives. Since, $precision=\frac{TP}{TP+FP}$, where TP and FP are true positives and false positives respectively, the increase in FP, decreases precision. Using our sophisticated model, on test data, we see a consistent result across recall(87.10%), precision(82.08%) and F measure(84.52%), and solves the issue of large number of false positives we had in a baseline model. Although our sophisticated segmentation model using Random forest as a binary classifier does well, our symbol classifier doesn't perform well in making good predictions for the classes the symbols belong to. We hope to improve upon the classifier, possible ways would be use an ensemble of classifiers and do an intensive grid search over a larger range.

References

[1] L. Hu and R. Zanibbi. Line-of-sight stroke graphs and parzen shape context features for handwritten math formula representation and symbol segmentation. In *2016 15th International Conference on Frontiers in Handwriting Recognition (ICFHR)*, pages 180–186, 2016.

[2] L. Hu and R. Zanibbi. Mst-based visual parsing of online handwritten mathematical expressions. In *2016 15th International Conference on Frontiers in Handwriting Recognition (ICFHR)*, pages 337–342, 2016.

[3] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: experiences from the scikit-learn project. pages 108–122, 2013.

[4] K. Davila, S. Ludi, and R. Zanibbi. Using off-line features and synthetic data for on-line handwritten math symbol recognition. pages 323–328, Sep. 2014.

[5] Bing Quan Huang, Y. B. Zhang, and M. T. Kechadi. *Preprocessing Techniques for Online Handwriting Recognition*, pages 25–45. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[6] Belongie and Malik. Matching with shape contexts. In *2000 Proceedings Workshop on Content-based Access of Image and Video Libraries*, pages 20–26, 2000.

[7] Nicholas E. Matsakis, A. Viola, and Nicholas E. Matsakis. Recognition of handwritten mathematical expressions. Technical report, 1999.