REPORT

SAURABH PAREKH, TAPPAN AJMERA

1) Design:

In order to perform segmentation, merging the basic unit (*in our case strokes*) in every possible way would be computationally expensive and time consuming. To reduce the search space, we decided to use Minimum Spanning Tree(MST). MST will connect all the strokes with minimum possible edge weights[1]. The concept of MST along with the constraints that the strokes can be merged with only upcoming strokes and a symbol can be made up of at the most four strokes. The motive was to reduce the search space. Once different possible combination of the stokes are obtained, using a ranking method we can select the best possible combination as our segmented symbol. Duplicate removal, normalization and resampling was done as steps of preprocessing. A combination of offline and online features was used.

The trace files are generated using human handwriting on touch screen devices, which leads to creation of duplicate points because of the speed with which hand is moving. The repeated points might create errors during classification. Hence, we perform duplicate point removal.

As we are using offline features like fuzzy histograms, line crossing etc. The features require the data to be normalized between a certain range and hence we normalize the co-ordinates. At the end, we performed smoothening, to reduce pen jitter. In the project we have mainly used offline features such as global features, crossing features and 2d Fuzzy histograms. We also used online feature (pen-up and pen-down) for classification which sums up to 82 features. As we know that classifying symbols is multi class classification problem and RandomForest is inherently multi class. It also performs multi class classification faster than algorithms which use one-vs-all to classify. Hence, we chose RandomForest.

2) Preprocessing:

Before doing feature extraction, the data goes through following preprocessing steps.

1) Duplicate filtering:
    a. This removes duplicate points from the strokes, as they are not useful.
2) Normalization:
    a. All the x and y coordinates are normalized to the range of 0 and 1.
    b. $X_i = a+(X_i-X_{min})\ (b-a)/X_{max}-X_{min}$ where a = 0 and b = 1
    c. $Y_i = a+(Y_i-Y_{min})\ (b-a)/Y_{max}-Y_{min}$ where a = 0 and b = 1
3) Smoothening:
    a. The stroke is smoothened by taking the average of the current point, previous point and next point in a stroke.

3) Symbol Classifier

We used RandomForest classifier for this Project. No new features were added but the parameters of the model were changed. For this project we kept n_estimators = 100 where as in previous project it was 200. The reason for doing this was that we saw there was no accuracy gain between 100 and 200.This significantly reduced the model size and training time. We also experimented with the depth of trees, but the accuracy of our model dropped from 74% to 60% when depth was set to 10. The accuracy of our current model is 74%

4) Segmentation

We did not use any features for our segmentation. The approach was based on the concept of minimum spanning tree. A node represents a stroke and the weight on an edge represents the Euclidean distance between the centroids of two strokes. Pseudocode of algorithm for an inkml file can be given as follows.

Pseudocode:

1) For every stroke find centroid
2) Initialize a NxN matrix for Euclidean distances
3) For stroke i=0 till N:
    a. For stroke j = i+1 till N:
        i. calculate Euclidean distance between $stroke_i$ and $stroke_j$ and store at [i][j]
4) Create minimum spanning tree (MST) using Kruskal's algorithm
5) Create graph from adjacency matrix created for MST where nodes represent strokes
6) Initialize a data structure to keep track of taken strokes as {key:False} where key is the stroke ID in inkml file
7) For every node:
    a. Perform DFS to find 3 closest nodes
8) For every node:
    a. Merge closest nodes sequentially as $[stroke_i + stroke_{i+1}]$, $[stroke_i + stroke_{i+1}+stroke_{i+2}]$, $[stroke_i + stroke_{i+1}+stroke_{i+2}+ stroke_{i+3}]$, to form stroke combinations
9) For every combination:
    a. Do preprocessing at stroke level
    b. Perform feature extraction
10) Predict class probabilities for every combination
11) For every stroke:
    a. if the stroke is not taken by any segment
        i. Get its combinations with other strokes from step 8
        ii. Find combination with max class probability
        iii. Make it a segment
        iv. Update the taken data structure from step 6 with all the strokes taken by segment
12) The symbols are segmented

RandomForest classifier was trained for ranking system and with 100 trees.

Clarification for Algorithm:

1) In step 6, 'taken strokes' means a stroke id that is taken by a segment.
2) In step 7, we find only 3 closest nodes because we assume that any symbol can be represented by maximum 4 strokes. We find 3 closest nodes to i^{th} node. Hence, we consider total 4 nodes.
3) The DFS is performed based on stroke ID as we assume that any character is formed by consecutive strokes only.
4) In step 8, we are creating sequential combinations because of we assume that a symbol is formed only by sequential strokes.

Complexity analysis of Algorithm:

1) Step 1 runs in O(n) time where n is number of strokes.
2) Step 3 runs in $O(n^2)$ time.
3) Step 4 runs in O(E logV) time where E is the number of edges and V is the number vertices.
4) Complexity of step 7 is O(n)*O(n+m) where m=3 and hence can be ignored. So, complexity of step 7 becomes $O(n^2)$.
5) Step 8 is O(l) where, l = number of combinations.
6) Complexity of step 9 is O(l)*O(k) where, l = number of combinations and k is number of points in stroke.
7) The complexity of step 10 is not considered as it relies on the complexity of the model used.
8) Complexity of step 11 is O(l)

As we can see, the dominating complexity here becomes O(l)*O(k) as the number of combinations 'l' are greater than number of strokes 'n' and 'k' is points in strokes which can be very large.

5) Results

Data split: To perform data split an algorithm was designed which would work on idea similar to voting system. For each symbol in the ground truth we would obtain its current train to test ratio. If the current training split of the symbol is less than the 2/3 of the overall data seen until now then that symbol votes to place the file in the training set and if it is greater then 2/3 it votes for the testing set. The vote for each symbol is taken into consideration and depending on majority of the votes the file is classified in the training or testing data. If majority of the symbols have training split lower than 2/3 of the current data then the file becomes part of training set or else it becomes part of testing set.

Consider the example "x=a+b" and after reading 10 files the train split is as follows, x → 0.4 ; '=' → 0.5, a →0.7, + →0.3, b → 0.7.
So, 'x', '=' and '+' will vote in favor of training set and hence the file will be placed in training set.

Base line – Training

	Recall	Precision	F measure
objects	21.56	31.03	25.44
+classes	8.32	11.98	9.82

Base Line - Testing

	Recall	Precision	F measure
Objects	20.72	29.50	24.34
+Classes	6.0	8.54	7.05

MST-Random Forest – Training

	Recall	Precision	F measure
objects	91.33	83.97	87.49
+classes	91.23	83.88	87.40

MST-Random forest – Testing

	Recall	Precision	F measure
objects	60.41	57.13	58.72
+Classes	49.94	47.23	48.55

Figure-1

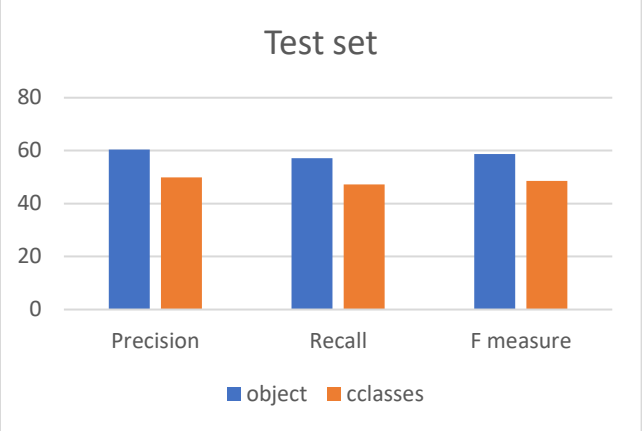

Figure-2

The F measure for object in the training set is 87.49 whereas the recall is 91.33, the lower pull to the F measure is because of the precision. The lower precision as compared to the recall states that the model is having fewer misses of the objects and classes, but it also has comparatively more false alarms. The F measure of the testing data drops sharply which tells us that the model is overfitting. A low recall of test set reveals that the model is missing many objects and classes, and lower precision tells us that it also has many false alarms and thus resulting in lower F measure. The F measure of the baseline is extremely low because it "only" consider four successive strokes while merging for symbols and it does not create MST thus, it lacks the measure of actual distance between the strokes.

Confusion Histogram

One of the major misclassification trend in our confusion histogram is that every symbol is misclassified with "\sin" within top 3 errors. The number of errors are more for symbols represented using straight lines like "+", "-", "x", "1". They are usually misclassified with symbols having curves in them like "(", "\sin", "c", ")". This supports our claim that smoothening is creating wrong curves between strokes causing misclassification with curved symbols. As we move to characters that are curved, the number of errors is reduced by half.

Errors in system

We analyzed the confHist file and came to know that symbols like 'x' were misclassified as '\sin' and 'pi'. One of the primary mistakes we found out was in our preprocessing task. Another problem is that after merging strokes, we send it for preprocessing. During preprocessing, when smoothening is performed, two strokes whose end and beginning are far apart will be averaged and a wrong curve between the two strokes will be generated. This may cause our model to learn wrong relationships between strokes and labels.

Currently this system assumes that a symbol is made up of consecutive strokes. However, this is not true. Some people tend to make some strokes at the end. For example, while writing words with character 't', some people may make the horizontal stroke of 't' the end of all the remaining strokes in that word. This can be improved by changing our algorithm where we consider only consecutive strokes to be merged. It can be changed to do all possible combinations between those 4 strokes.

Improvisation

To improvise our model further we would like to add the context information before merging the successive stroke pair. Features like **stoke pair shape context** that is considering current and following stroke, **Local neighborhood shape context** and **global shape context** as they did in [4] might help us in gaining better F measure. We would also like to change our approach of smoothening.

Time:

Model training time - 6.65 mins

Execution on Training set - 40.03 mins

6) Reference

[1] N. Matsakis, “Recognition of handwritten mathematical expressions,” Master’s thesis, Massachusetts Institute of Technology, Cambridge, MA, May1999.

[2] K. Davila, S. Ludi and R. Zanibbi, "Using Off-Line Features and Synthetic Data for On-Line Handwritten Math Symbol Recognition," *2014 14th International Conference on Frontiers in Handwriting Recognition*, Heraklion, 2014, pp. 323-328.

[3] “scikit-learn: machine learning in Python — scikit-learn 0.19.1 documentation.” [Online]. Available: http://scikit-learn.org/stable/

[4] L. Hu and R. Zanibbi, "Segmenting Handwritten Math Symbols Using AdaBoost and Multi-scale Shape Context Features," *2013 12th International Conference on Document Analysis and Recognition*, Washington, DC, 2013, pp. 1180-1184.