REPORT

Tappan Ajmera, Saurabh Parekh

1) Design

The primary goal that we are trying to achieve is to classify the given trace file into a particular class. The first step to achieve this task is preprocessing. To achieve good classification results, it is important to preprocess the data. The x and y co-ordinates of the trace point do not lie in the same range. Moreover, the number of points for trace points for representation of different characters differ largely and hence to have a common basis of comparison data preprocessing is an important step. Our choice to preprocessing steps are duplicate point removal, normalization or x and y coordinates, and smoothing.

Since trace files are generated by tracing human handwriting on touch screen devices, duplicate points get registered because of the speed with which hand is moving. Slow speed might register more points whereas faster might register less points. These points can create errors during classification. If duplicates are not removed, the later stages for feature extraction will give erroneous features. Hence, we perform duplicate point removal, and this is the first step of preprocessing.

Second step in preprocessing is normalization of coordinates. This is required because we are using offline features like fuzzy histograms, line crossing, etc. These features require the data to be normalized between a certain range. Finally, smoothing is performed to remove pen jitter.

As non-baseline classifier, we choose Random Forest. The reason for choosing this classifier is that it is inherently multiclass classifier that is it does not require one-vs-rest or one-vs-all. Another advantage is that it relies on probabilities and majority voting. This means that at each node of the tree probabilities for the class is calculated and to take the final call, it takes performs majority voting. This is very intuitive, if 4/10 trees from the forest say that sample belongs to class '(' and 6/10 say it belongs to class '1' then vote is given to class '1'. However, every tree makes its decisions based on probabilities. Hence, we assume that a Random Forest will give us good classification rate.

Overall design of our system can be seen as a pipeline where the data is first preprocessed, then features are extracted and finally prediction is done by KD-tree and Random Forest classifiers.

2) Preprocessing and Features

In order to have an apple to apple comparison, it is very important to preprocess the data.

The first step towards data preprocessing was removing duplicates. These points have the same x and Y coordinates as their previous trace points and are not useful. Duplicates were removed by finding co-ordinates having same X and Y value for consecutive co-ordinates.

Once duplicate trace points are removed then data normalization is performed. All the x and y coordinates are normalized to the range of 0 and 1. Normalization is achieved using the formula

$$X_i = a+(X_i-X_{min})\,(b-a)/X_{max}-X_{min} \text{ where a and b are range for normalization}$$

Using the above formula experiments were performed to check how different normalizing range affects the accuracy of the model.

Lastly, in order to reduce the noise caused by the pen's jitter, we performed smoothing. Each point is replaced by the average value of three points. The point under consideration, the next point, and the previous point is considered for calculating the average.

Smoothing is performed by the following formula

$$X_i = (X_{i-1} + X_i + X_{i+1})/3$$

2.1) Feature extraction

The basic goal of our feature extraction was to make the model robust to new an unseen data. Also, while deciding upon the feature care was taken that the features are successfully able to reject the junk class even though the data is imbalanced. We initially tried to rely upon online feature to get information about how the trace was drawn. We believed the information of how the different trace points are related can be very useful for the classification part. However, one of the important preprocessing steps was to resample the data points so that all the input traces have the same number of data points. Resampling was done using interpolation but as we were not able to interpolate correctly, it was unable to create smooth curves and identical original character trace which resulted in a low accuracy of the models and we eventually had to shift to offline features for identifying the characters. The big take away with this entire process was how important is it to have data preprocess properly with no loss of information of data and the importance of uniformity among all the data entries in the data set was well.

Another approach we used for feature extraction was using a combination of global and local features. The global features are the features that describe an image as whole. They generalize entire image with a single feature. Local features concentrate on a particular region of the image in our case it will derive information from a particular region of the character. The extracted features can be classified into three broad categories global features, crossing features and 2d fuzzy Histogram.

1) **Global features:** These features account for a total of 7 features. The features are Number of traces, line length (total length of lines between consecutive trace points) , number of sharp points (sharp edges in characters for example the caps of 'M') , aspect ratio (width/height without normalization), mean of x coordinates, mean of y coordinates and covariance of X and Y.
2) **Crossing features:** Consider an arbitrary line passing through a given character. This can be given as number of times the line crosses the character and the start and end points where the line crosses the figure. This property is utilized to gain more idea about the shape of the character. It gives insights of the curves in a character. The x and y-axis are divided into 5 regions and each region further has 9 lines. The count of the number of times each of these lines crosses the character, the first point and the last point of crossing with the character is considered. Average of each of these features is taken. Hence, in every region we get 3 average values. We have 10 such regions hence we get 30 features from crossing lines.
3) **2d Fuzzy histogram:** In virtual 2d Fuzzy Histogram a grid is created which contains the character in it. We have created a 4 X 4 grid for which we have 25 corner points. Each trace point will belong to one of the cell. The membership value between each point and all the four corners of the grid where it lies is calculated

$$m_p = \frac{w - |x_p - x_C|}{w} \times \frac{h - |y_p - y_c|}{h}$$

Where m_p is the value for membership function, w is the width of cell and h is height of cell. In our case h and w were 0.25.
Once the membership values of each trace point are calculated the total membership values of the corner is added and divided by the total number of trace points. The membership value of each corner point was considered as one of the feature. So, in total it sums up to 25 features.

The total number of features sum up to 25+30+7 = 62 features.

Since the features from crossing contain X and Y coordinates, the total size of feature vector becomes 50 for crossing itself. Hence the final feature vector generated for a sample has 82 columns.

3) Classifier

3.1) KDTREE

A KD tree is basically an extension of the binary search tree. KD tree divides the data space into horizontal and vertical space. The node of the KD tree represents this space where only those data points are present which have closer dimensional/ attribute values. For simplicity, we will call this horizontal or vertical data space as a bounding box. Once the KD tree is constructed the class value of the new point is predicted by applying the concept of KNN classifier on the structure of the KD tree. We begin by traversing the tree looking for the bounding box where the new point should lie. Once the bounding box where the query point (value of the point to be predicted) lies is found we calculate the nearest distance to this query point by finding the Euclidean distance to all the points in the bounding box and select the smallest one. Once the nearest distance is returned, search each subtree where a closer point compared to the current shortest distance can be found. However, while searching other subtrees if the distance to the new node (in this case distance to bounding box) is greater than the shortest distance we simply prune that node. By doing so the computation time for the distance reduces greatly.

For this project, we have used scikit-learn's KNN classifier with k=1 and algorithm="kd-tree". Since KDTree is just another version of KNN, it is provided under KNN library in scikit-learn.

3.2) RandomForest

Random forest is created from multiple decision trees. A single decision tree has root node, child nodes and leaf nodes. The root node is where the training starts. Child nodes contains a split function and the goal is to optimize this split function. Finally, the leaf node contains a classifier (meaning: a function that decides the class of the sample depending upon the value of optimized split function) which classifies the input sample. During the training phase, the input dataset is given and the split function in child nodes is learned. The job of this split function is to decide which part of the input belongs to which branch. It splits the input data and sends each split of data further in its classified branch. The algorithm does this till it reaches a leaf node which contains the decision. In case of Decision trees, the parameters that we can control to tune our model is the depth of the decision tree.

A Random forest, as the name suggests is a forest of these Decision trees and we induce randomness to these trees. Randomness can be induced in multiple ways that is by bagging, randomized node optimization. The Random Forest algorithm can be summarized as

1) Randomly select k features where k << n (number of features)
2) From k features calculate the split function and create a split
3) Perform above steps until leaf nodes are reached

4) Build the forest for given ‘z’ number of trees

Due to the randomness introduced, the trees are de-correlated (they will make decisions based upon different set of features available to them). This leads to better results. This also ensures that different trees predict results based of different features and hence we can perform majority voting. For random forest the parameter that needs to be tuned is number of trees and maximum depth.

For our experiment, we used a forest with 200 decision trees. We tried a forest with 100 decision trees but using 200 trees gave a bump of approximately 1%. It also performed faster on such a big dataset.

We tried using different models like GaussianNB, SVM, KNN with (3,5,15,17) models. But RandomForest gave us better accuracy on our dataset with mentioned features.

5) Results

Random Forest accuracy without Junk data: 79.2%

Random Forest accuracy with Junk data: 76.41%

Random Forest accuracy with Junk data TOP 5: 96.45%

Random Forest accuracy without Junk data TOP 5: 93.63%

KDtree accuracy without the Junk data set: 59.45%

KDtree accuracy with the Junk data set: 63.56%

The sudden rise in the KD tree can be explained by reviewing the Accuracy formula

Accuracy = True Positive + False negative/(True positives + False positive + False negative +False Positive)

So, in this case, the number of the reject class (JUNK) is too high which makes the model to predict JUNK class more often. If we consider junk as reject class, the false negatives of the system increase which in turn increases the accuracy. The better representation of the model's performance can be understood by precision and recall.

The recall or true positive rate is the ratio of the number of objects that are correctly identified upon the total number of the object i.e (tp / (tp + fn)). In other words, it gives an idea that "of all cases that were actually true, what fraction did you report".

The precision, on the other hand, tells us that “of the fraction you said were true what fraction was actually true" i.e (tp / (tp + fp)).

Both precision score (0.792) and the Recall score (0.792) of the random forest model without the junk data is high and closer to the accuracy this shows that the model does actually predict the values accurately. Neither does it have false alarms nor does it misses on many of the actual symbols. However, when the JUNK data is induced, it has a low false alarm rate which is 7.53% hence it makes very less false alarms. But the false reject rate is 28.35% which is a bit high that means it is missing on actual symbols.

5.1 Confusion matrix summary

If we take an in-depth look at the models we found out that the letter 'o' has pretty less accuracy the reason is that the model is wrongly classifying it as zero-''0". Also, the letter 'C' was confused with the opening parenthesis -'(' and comma: ',' was also confused with opening parathesis-'('. Looking at the numbers we found that the TOP 1 accuracy was comparatively better for numbers as compared to letters.

We feel that using global and local features together did help in achieving the current accuracy. Also pre-processing the data, normalizing all the traces help used in achieving common grounds of comparison and better accuracy.

However, the accuracy of the model does not match the current state of the art one of the reasons we think is that we did not include any attribute that gives information about angles of a character. For example, curves in letter 'S'.

We think our classifier can be improved by adding online features and also use the features that would retain the information about the angles in character.

References

[1] "scikit-learn: machine learning in Python — scikit-learn 0.19.1 documentation." [Online]. Available: http://scikit-learn.org/stable/. [Accessed: 06-Mar-2018].

[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*, pp. 323–328, 2014.

[3] L. Hu and R. Zanibbi, "HMM-Based Recognition of Online Handwritten Mathematical Symbols Using Segmental K-Means Initialization and a Modified Pen-Up/Down Feature," in *2011 International Conference on Document Analysis and Recognition*, 2011, pp. 457–462.

[4] F. M. G. França and A. F. de Souza, Eds., *Intelligent Text Categorization and Clustering*. Berlin Heidelberg: Springer-Verlag, 2009.