Classification of Handwritten Math Symbols

Bhuta, Bhavin
bsb5375@rit.edu

Chauhan, Dhaval
dmc8686@rit.edu

1 Design

We start with reading the data files, and extracting labels and trace information from it. Trace information contains array of 2 dimensional co-ordinate points which we are cleaning and preprocessing before extracting features from them. Once the features are extracted and a feature vector is created, we fit them with two classification models that are KDtree, and Random Forest. Our processing chain and most choice of preprocessing techniques and features are inspired from the work done in [1]. This document is structured to explain every major step of the process in detail with results and general discussion following them.

2 Preprocessing

Some form of preprocessing is always required for any machine training task. This step is to ensure that data is compared on common ground. Just to see the difference in accuracies we first trained our classifiers with data that were not preprocessed at all. The accuracy of our model increased by 6 times once the preprocessing step was inserted in our process. Our preprocessing is based on some of the preprocessing techniques used in [3].

The first step in our preprocessing step was to clean the data. We removed the duplicate points that are consecutive in the timeline. Consecutive duplicate points are not desired because it increases the computation time for no reason. They do not contain any information anyway. Duplicate points may have been recorded from holding the pen in a particular spot for an unexpectedly long time. So if we come across a series of duplicate points we keep one and delete the rest.

Once the duplicate points are removed, we use cubic splines to smoothen the trace. This removes the aliasing artifacts that occur from shaky hands, slippery pen etc. It also gives us new set of interpolation points that when linked to each other, makes the trace look like one continuous curve. Interpolated points eventually help us to better compute one of the most important feature in our list of features. Figure 1 shows the trace of symbol '2' before smoothening and the same trace after smoothening. Smoothening also renders removal of hooks in [3] unnecessary.

The last and one of the most important preprocessing task in normalization. This is the step that puts every symbol on the same comparison ground. Our idea of normalization squishes or inflates every symbol such that it fits into a grid of size 2 by 2 in on -1 and 1 on both the axis. However, our implementation of normalization is only based on maintaining the original aspect ratio, it doesn't take into account the size of the symbol in relation with other symbols. The

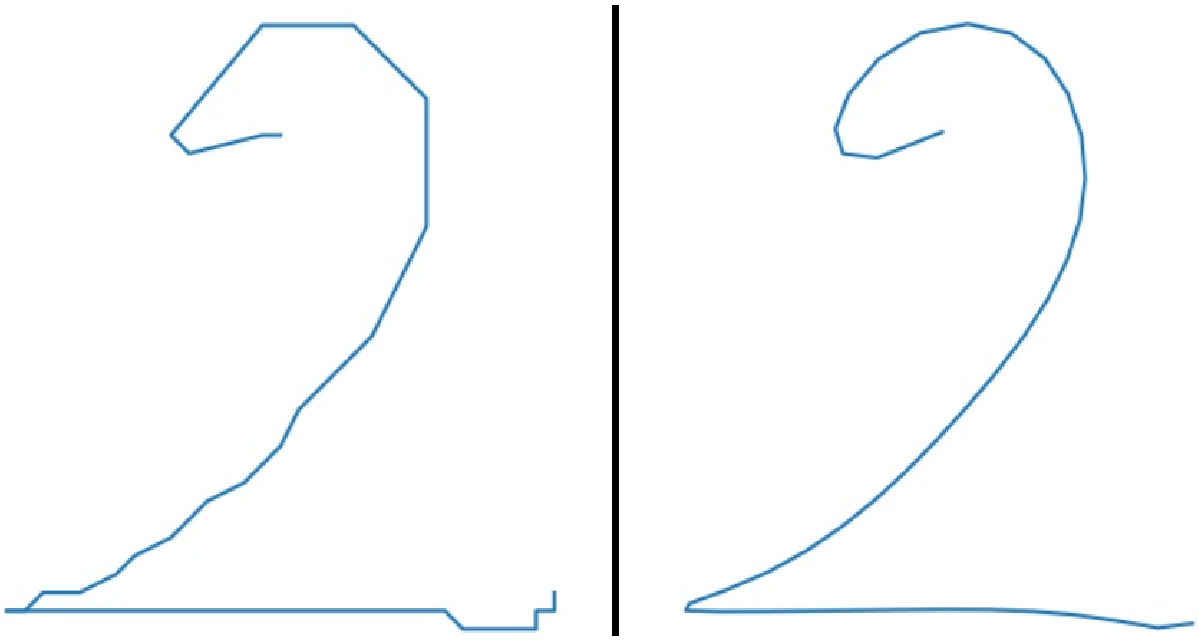

Figure 1: Before and after smoothening using Cubic Splines.

errors that occur that possibly arise from this is also discussed in the results section. Formula that we used for normalizing is given in the equation 1.

$$z_i = 2\frac{x_i - min(x)}{max(x) - min(x)} - 1 \tag{1}$$

3 Features

We had started implementing online features like time taken to draw the entire symbol, average speed of drawing all the traces, etc. that used online trace information but there were data files that didn't include time information. Due to this, we decided to use only continuous valued offline features of the symbols. And, as mentioned in [1], online features may not be reliable features for achieving desired accuracy due to high sensitivity to variation from different writing styles. This was another reason to just stick to offline features. Our features are the same from [1]. We tried a variation of 'Fuzzy histograms of orientations' from [1] but then our version was not structured well to distinguish different symbols satisfactorily. It affected our accuracies negatively on average. We also thought of using image pixel values as features by converting the traces to image matrices of size 12x12 but that would have made feature vector too big to train in reasonable amount of time and also because we thought maybe this feature would not classify as continuous.

3.1 Global features

These features are basic features that don't need much computation and also doesn't necessarily tell much about the symbols. These set of features include number of traces in a symbol(1), total trace length in a symbol(1), mean of x co-ordinates(1), mean of y co-ordinates(1), covariance between x and y(1), and aspect ratio(2). we split aspect ratio into 2 features to avoid division by zero.

3.2 Crossing features

This feature was implemented in [1] and it turned out to be the most important feature in their experiment, which encouraged us to add this feature in our experiment as well. We divide the square symbol area into 5 regions both horizontally and vertically. We then introduce 10 equidistant subcrossings in each region and compute the average number of intersections these 10 subcrossings make in a region. This 10 features help in explaining the shape of the symbol well by telling us how many strokes of symbol appear in a particular row or column. The difference in our adaptation of this feature from [1] is the exclusion of first and last intersection points.

3.3 Fuzzy histogram of points

This feature is exactly the same from [1]. We divide the symbol space into 4x4 grid and at every corner point of the grid we calculate membership value of the trace points if those trace points lie inside a grid that, that corner point belongs to. We get 25 total corner points and we computer membership value of point P using the formula shown in equation 2. We create a fuzzy histogram of these corner points by summing all the membership values at a corner point and then dividing it by the total number of trace points in the symbol. This feature turned out to be the most important feature in our experiment as it increased the prediction accuracy significantly.

$$m_p = \frac{w - |x_p - x_c|}{w} \times \frac{h - |y_p - y_c|}{h} \tag{2}$$

4 Classifier

The two algorithms used throughout the classification task are KD-Tree and Random Forest. KD tree belongs to nearest neighbor classifier family with goal to subsidize variance by cutting the dimension with highest variance (by calculating the median). It mimics a tree structure to find the response we traverse the tree by using a distance measure. We have used KNN classifier from scikit-learn which uses kd-tree algorithm to compute the nearest neighbor and subsequently decide the label. The hyper-parameters to play with KD tree are number of neighbor which for our exercise is set as 1, leaf size relates to number of splits we would want for a node. We have used default value for this parameter i.e the leaf size of 30, this parameter relates to construction of trees and use of resources like memory and space. Since we didn't encounter any issues while running an instance of kd-tree we didn't experiment much with it. But we believe that decreasing after a particular value it would run out of either memory or space. As lower the value for it the larger would be the tree size and hence would need more resources. The metric we used for computing the closest neighbor for query vector is euclidean distance.

Random Forest Classifier belongs to the family of ensemble algorithms, ensemble algorithms as the words suggest are collections of more than one same or different machine learning models (in this case it is restricted to classification models). The notion of using more than one model is to get better understanding of the data, to smooth the noise created say by using a single classifier (by averaging the responses of various classifiers). To sum up, Random Forest classifier believes that when many weak classifier (noisy) are combined it would form a strong final classification model. The important dials for random forest are number of trees and depth size. We tried with 25 and 15 as the number of decision trees but while running instances of it, we ran out of memory and we stuck with 10(default provided by scikit) as our final number of decision trees. The larger the depth of tree we'll see overfitting (i.e. it will try to closely fit the data points) and the smaller a value is underfitting (i.e. it will not get the underlying spread of the data). We tried with different depth values and corresponding accuracy results are provided in the Table 1. The table shows that after we increased depth size from 100 to 1000 accuracy decreased by low decimals, maybe the maximum depth must be in between them. The bootstrap samples were used for building the trees and gini index was used as split criterion while building each trees. Besides this all the other parameters were the default values provided directly by scikit-learn.

Table 1: Different values for maximum depth size in Random Forest

Depth Values	Accuracy
25	78.06
50	78.14
100	78.19
1000	77.84
5000	77.78

Our plain to reason to go with Random Forest classifier is due to the fact of using a single classifier(like a decision tree) try an ensemble of classifiers(by having n number of decision trees). The gains of using Random Forest is that a rise in accuracy and faster training and testing. As these are important characteristics for a recognizer(classifier) to have, although at the loss of interpretation. Therefore try to explore what is the best that could be achieved with given features and model.

5 Results and Discussion

5.1 KDTree evaluation (with k = 1, leaf size = 30)

In order to get top-10 class labels predicted for input query vector we used built-in function provided by scikit-learn known as query for kd-tree which takes feature vector and number neighbors with reference to the model name. The kd-tree classifier provides correct classes from a list of top 10 classifier on 97.23% from the given training dataset. The mean position signifies that on average correct predictions are provided within the Second predicted labels provided by for our model. This is also obvious by looking at the number of instances and jump in prediction rate by going from prediction 1 to prediction 2 for class label. Similar trend is observed for dataset when used without junk.

For each individual labels recognition rate, the important takeaways from individual symbol recognition rate observed are that !,), (, $+$, $=$, ∞ have been recognized the best with KDTree on first prediction. While $>$, [, ∞, $<$, Θ, and o are the symbols that had all the correct predicted label within top-10 labels provided by the model.

The highlights of the confusion matrix according to us are, the kd-tree model got confused more evidently when cases are:

a) Confusion between 0 with o, x, y:

Confusing 0 with o is logical given that they follow the same structure except for the trace length attribute. The reason of closeness between 0, x and y is probably due to fact of using crossing features and histogram of points.

b) Confusion between 6 and b:
We believe attribute like Angular membership points would be great help particularly for this case. As for 6 and b the best distinct feature to identify would be related to angular membership points.

c) Confusion between 4 and +, a, x, y:
The reason why we believe it confused 4 with + is probably that, since the 4 and + would correspond to having similar membership presence of points for same regions.

d) Confusion between ',' and), 1, |, and between t and +, f:
As), 1 and | share similar aspect ratio as well as membership points the model is getting confused in predicting it. For t with + and f reasons follow on similar line.

e) 2, -, =, +, !, (,) were some of the symbols that were recognized at good rate.

f) Poor recognition rate for '.':
Symbol '.' was predicted completely randomly, for that case maybe having number of points in a trace as a feature would have helped.

Maximum effect junk symbols had on predicting correct labels were for ',', -, '.', /, 6, N, P, ldots, prime. Likewise, we could also infer that maybe some of the actual symbols from this Junk must have been above mentioned labels.

5.2 Random Forest evaluation(with max depth = max depth and number of trees = 10)

For random forest evaluation of closest k neighbors, we used predict proba function provided by scikit-learn package. This function computes class probabilities for input sample as the mean predicted class probabilities of the trees in the forest. The class probability of a single tree is the fraction of samples of the same class in a leaf.

By looking at each of the frequency tables and mean position, we see that with Junk on an average we are likely to get correct label for a symbol earlier as compared to using without Junk. It is also evident by observing the second predicted class by random forest for which Junk class is better than without Junk class.

For individual labels with top 10 labels provided by the model. We see that at the end of 10th predicted label 'l','prime','mu','lambda', Comma didn't provide correct label within first 10 labels.

Even with random forest we see some of the similar labels predicted incorrectly as we saw with kd-tree:-

a) Comma predicted as ')', '1','|':
The major reason for this is as we perform normalization, essentially comma,), 1, | would look almost identical in that common space.

b) Confusion between capitals and lower case alphabets:
We believe normalization is the same reason over here which is why model is getting confused between the cases of alphabets. This kind of observation can be inferred even from the confusion matrix from the merged symbols.

c) Excellent recognition rate of '=':
equals sign was predicted correctly almost every time, as it has a unique structure to it. Similarly, symbols like lambda, '+', '-' as it has unique structure to it and hence the derived attributes are unique to help the model better distinguish between these symbols

In conclusion, we would like to summarize the recognition rates from both of our classifiers. Table 2 displays the recognition rates of KDTree and Random Forest with 10 number of trees

Table 2: Recognition rates with 70% training data and 30% testing data for testing.

KDTree				Random Forest			
Valid		Valid + Junk		Valid		Valid + Junk	
70% resub	30% test	70% resub	30% test	70% resub	30% test	70% resub	30% test
100%	74.35%	100%	72.85%	99.61%	81.13%	99.46%	79.26%

and no limit on max depth when trained and tested with Valid data, and Valid plus Junk data both. We also tested the trained classifiers with train data itself and a separate set of test data. We get 100% recognition rates in KDTree when tested with the train data since KDTree is an approximation of 1 Nearest Neighbor classifier which has very high variance and there already exist a local space for each test data sample as it was trained on it. Random Forest doesn't get 100% accuracy here but even then accuracies are almost 100% due to the use of training data for testing. Overall, Random Forest seems to be classifying the test data better than KDTree mostly because of bagging. Commenting on the recognition rates, we think that had we performed a scaled (based on absolute size along with aspect ratio) normalization of every symbols we probably would have seen better recognition rate overall. This we can conclude from seeing the number of diagonal elements in the merged symbol confusion matrix in the supplied HTML files.

6 References

[1] Davila, K., Ludi, S., and Zanibbi, R. Using online features and synthetic data for on-line handwritten math symbol recognition. In Frontiers in Handwriting Recognition (ICFHR), 2014 14th International Conference on, IEEE (2014), 323–328.

[2] H. Mouch'ere, C. Viard-Gaudin, D. H. Kim, J. H. Kim, and U. Garain, "Icfhr 2012 competition on recognition of on-line mathematical expressions (crohme 2012)," in Frontiers in Handwriting Recognition (ICFHR), 2012 International Conference on. IEEE, 2012, pp. 811–816.

[3] B. Q. Huang, Y. Zhang, and M.-T. Kechadi, "Preprocessing techniques for online handwriting recognition," in Intelligent Text Categorization and Clustering. Springer, 2009, pp. 25–45.

[4] M. Yang, K. Kpalma, J. Ronsin et al., "A survey of shape feature extraction techniques," Pattern recognition, pp. 43–90, 2008.

[5] NumPy/SciPy guide: https://docs.scipy.org/doc/numpy-1.13.0/reference/

[6] Scikit-Learn guide: http://scikit-learn.org/stable/documentation.html

[7] Beautiful Soup guide: https://www.crummy.com/software/BeautifulSoup/bs4/doc/