Math Symbol Classification

Ajinkya Dhaigude and Supriya Godge

1. DESIGN

In this project, we design a handwritten math symbols classifier using mostly offline and a few online features. We initially preprocess the data by converting it into an offline grayscale image. This image is then scaled to a size of 50x50 whilst maintaining the aspect ratio. We then extract 381 hybrid features [2] for each of our training sample. These hybrid features are then fed into the scikit-learn implementation of Random Forests and the results are compared with the results obtained from a 1 nearest neighbor kd-Tree.

2. PREPROCESSING AND FEATURES

Preprocessing

Preprossessing is done prior to the feature extraction, to make sure that the all data instances are in a consistent format. The following outlines our preprocessing steps:

Normalization and Aspect Ratio

The inkml file data was from various different sources and lacked consistency. The first step was to normalize the data to make all the coordinates fall in a range of 0 to 30. This was done while maintaining the original aspect ratio of the data sample. This is important to preserve the shape of the symbols otherwise a ',' may end up looking the ame as ')'. the following formula was used for data normalization:

$$\Delta X = X_{max} - X_{min}$$

$$\Delta Y = Y_{max} - Y_{min}$$

$$\Delta = max(\Delta X, \Delta Y)$$

$$X_{new} = (X_{old} - X_{min}) \times \frac{new\_size}{\Delta}$$

$$Y_{new} = (Y_{old} - Y_{min}) \times \frac{new\_size}{\Delta}$$

where, new_size is the size of the scaled image.

Image Construction

The above obtained normalized and scaled coordinates were then converted into an image. The *polylines* function in the OpenCV library was used to obtain a 3-channel image from the x and y coordinates.

Grayscale Conversion

The above obtained image is then converted into grayscale to reduce the dimensionality and get the image size as $n \times n$ from $n \times n \times 3$. The value of each pixel here is in the range of 0-255. The image was not converted into binary as we belive giving multiple shades of the grey will help extract more imformation through offline features.

Gaussian Filter

Finally, a Gaussian filter with 3×3 filter was applied to the image to blur it and give it more natural look.

Bias Removal

The data instances of each class varies drastically, for Junk it is around 99k and for μ and $\forall$ it is less than 100. Because of this the priors of the classes are biased towards the few bulky classes. This is the hindrance in the classification of the classes with the small priors. To remove this bias

-

Features

The next step was to extract features from the data samples. We extract a a total of 381 online and offline features for classification. Our initial aim was to extensively use online features but doing this proved difficult especially because of the speed normalization step. The following is a detailing of the fetaures used:

Zoning

This offline feature is used to get an idea of the density of the pixel values at different parts of the image. As shown in *figure1*, the image matrix is initially divided into smaller squares. Then the mean pixel value of each each small square is computed and and the matrix is collapsed into an array to get the final feature vector.

X-axis Projection

This is an offline feature obtained by storing the sum of each column of the image matrix in an array. The size of this feature vector is then equal to the width of the image.

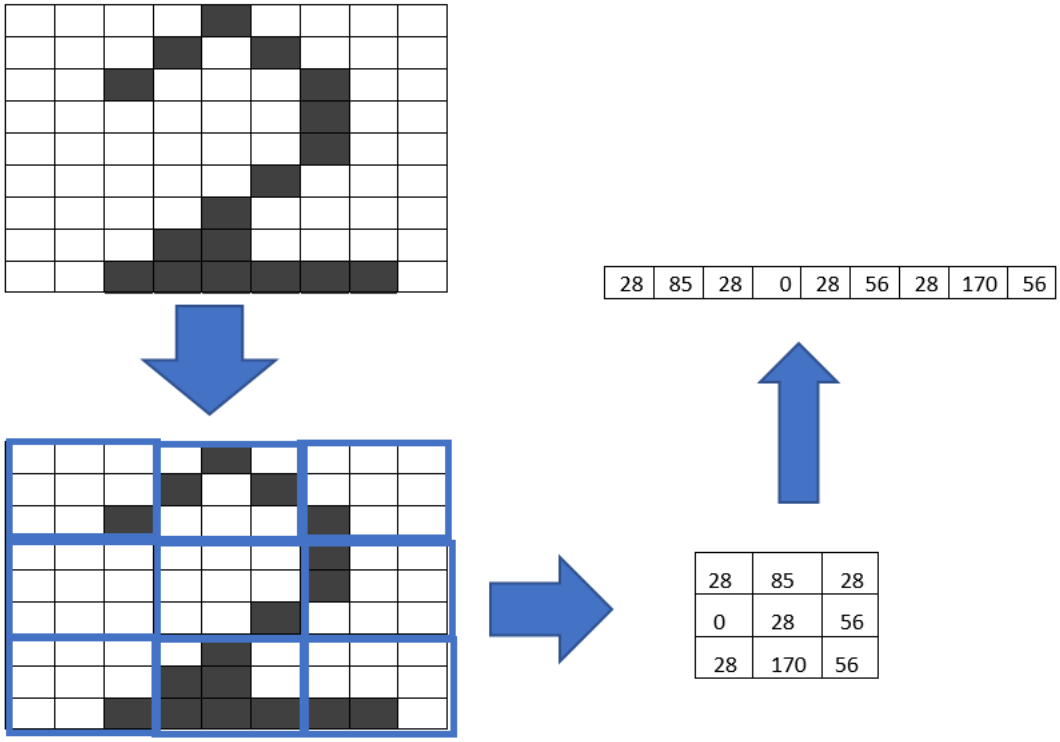

Figure 1: figure1: Zoning feature extraction

Y-Axis Projection

This is an offline feature similar to the one obtained above but in the Y direction. It is obtained by storing the sum of each row of the image matrix in an array. The size of this feature vector is then equal to the height of the image.

Diagonal Projections

This feature vector is similar to the above X and Y axes projections. We consider the two diagonals of our square image matrix and compute the sum across each each cell perpedicular to the diagonal. The final array of sums obtained from the two diagonals is merged to get the Diagonal Projections feature vector.

Histogram of Oriented Gradients

This is an offline feature that is obtained by dividing the image matrix into smaller blocks. Each smaller block is then divided up into smaller cells for which the Sobel operator is used to compute the magnitude and orientation. However, our implementation of this algorithm caused the program to slow down drastically and did not provide an increase in accuracy. We believe this may have been due to an incorrect implementation of Histogram of Oriented Gradients (HoG) as the HoG is known to produce good results for classification of handwritten characters [1]. This implementation was removed from our final submission.

Difference between starting and ending stroke of the symbol

In this feature the euclidean distance between starting stroke and the ending stroke was calculated. This feature was design to help classify between the classes with the big loop, like 0,o,8,D,B vs other classes.

Aspect Ratio

The aspect ratio gives the hight to weigh ratio for the all classes, many papers with in the math symbol recognition mentioned that this is a good feature, in this paper also we found out that the accuracy of the classifier was increased after using the aspect ratio.

3. CLASSIFIER

The first classifier was the KD tree, second chosen classifier is Random Forest.

3.1 KD tree

3.2 Random Forest

Following are the reasons behind choosing this classifier:

- The traning and testing of the Random forest is really fast. The kd tree takes around 20-25 min to train while Random forest takes around 10-15 min.
- It is unlikely for a random forest to overfit
- It captures the underlying architecture of the data accurately given good features.

4. RESULT

We achieved the following results for our final implementation.

- For Random Forest: An accuracy rate of 84.11% for training without JUNK symbols and with 70% training data.
- For Random Forest: An accuracy rate of 77.21% for training with JUNK symbols and with 70% training data.
- For kd-Tree: An accuracy rate of 80.28% for training without JUNK symbols and with 70% training data.
- For kd-Tree: An accuracy rate of % for training with JUNK symbols and with 70% training data.

The most common error made by the classifier was for x and X despite having a reasonably large number of samples in both. This was probably because our features were not strong enough to derive strongly distinguishing features between them. Using more complex online features to get context would have helped in this case.

5. DISCUSSION

Biased vs Unbiased:

In theory, we should be able to get the good classification result if the data is not baised toward set of the classes that means if the priors of the all classes are the same then we should get the good result. Surprisingly, that is not the case with crom dataset, the accuracy with the baised symbol data was 85.22% and accuracy on unbaised symnol data was 82.71%. The only working therory that we have for this behavior is, most of the data that is contributing towards the accuracy is biased one with the good classification.

Things learnt from the mistake

The test file and the train file were accidentally swapped, and the accuracy of KD tree was decreased drastically from 80% to 48%. The random forest, however learned the underlying architecture of data even with the small information available and saw a negligible drop in accuracy.

6. REFERENCES

[1] S. Iamsa-at and P. Horata. Handwritten character recognition using histograms of oriented gradient features in deep learning of artificial neural network. In *2013 International Conference on IT Convergence and Security (ICITCS)*, pages 1–5, Dec 2013.

[2] F. ÃĄlvaro, J. A. SÃąnchez, and J. M. BenedÃ■. Classification of on-line mathematical symbols with hybrid features and recurrent neural networks. In *2013 12th International Conference on Document Analysis and Recognition*, pages 1012–1016, Aug 2013.