Using Off-line Features and Synthetic Data for On-line Handwritten Math Symbol Recognition

Kenny Davila [1], Stephanie Ludi [2] and Richard Zanibbi [1]
[1] Department of Computer Science
[2] Department of Software Engineering
Rochester Institute of Technology
Rochester, New York, USA
Email: kxd7282@rit.edu, salvse@rit.edu, rlaz@cs.rit.edu

***Abstract*—We present an approach for on-line recognition of handwritten math symbols using adaptations of off-line features and synthetic data generation. We compare the performance of our approach using four different classification methods: AdaBoost.M1 with C4.5 decision trees, Random Forests and Support-Vector Machines with linear and Gaussian kernels. Despite the fact that timing information can be extracted from on-line data, our feature set is based on shape description for greater tolerance to variations of the drawing process. Our main datasets come from the Competition on Recognition of Online Handwritten Mathematical Expressions (CROHME) 2012 and 2013. Class representation bias in CROHME datasets is mitigated by generating samples for underrepresented classes using an elastic distortion model. Our results show that generation of synthetic data for underrepresented classes might lead to improvements of the average per-class accuracy. We also tested our system using the MathBrush dataset achieving a top-1 accuracy of 89.87% which is comparable with the best results of other recently published approaches on the same dataset.**

Keywords: Handwritten Character Recognition, AdaBoost, Random Forest, SVM, Elastic Distortion, Off-line Features

I. Introduction

In pattern recognition there are several applications for automatic recognition of math expressions. This is a hard task specially if the input data is handwritten. This task has two general modes: on-line and off-line. The first one refers to the case when tracing information is available. The second one refers to the case when tracing information is not available but images of the final drawings are provided instead.

In this research work we are focused on using adaptations of off-line features for on-line classification of math symbols. Since the training dataset is usually biased toward very few classes, we also reduce this class-representation disparity by generating synthetic data using an elastic distortion model. We tested our approach using different classification methods: AdaBoost.M1 [1] algorithm using C4.5 decision trees [2] as the weak learner, Random Forests [3] and Support-Vector Machines (SVM) [4], [5] with linear and Radial-Basis Function (RBF) kernels. The input of these classifiers is a set of simple features used for description of the shapes of each symbol. For our main experiments we used datasets from the Competition on Recognition of Online Handwritten Mathematical Expressions (CROHME) 2012 [6] and CROHME 2013 [7]. Finally, in order to provide a comparison of our approach with other existing approaches for isolated math symbol recognition, we benchmarked our system using the MathBrush database [8]. The complete source code of our system is available at http://www.cs.rit.edu/~dprl/Software.html.

Different approaches have been used before in the field of math recognition and retrieval as documented by Zanibbi and Blostein [9]. The CROHME competitions [6], [7], [10] provide a good comparison between different approaches for math expression recognition on a set of standardized tasks. For the scope of this paper, we are only interested in the specific problem of isolated on-line math symbol recognition.

One of the most important elements of a math symbol classifier is the set of features used to describe each symbol. In the survey by Yang et al. [11] different feature extraction techniques for shape description have been identified. While some feature extraction techniques based on timing or drawing information have been used before for on-line symbol recognition [12], [13], [14], [15], [16], techniques based on shape description might be more robust since a single shape can be drawn in several ways. In the work by Delaye et al. [17] a set of 49 features that describe both the drawing process and the shape have been proposed as a baseline for different on-line symbol recognition datasets.

In addition to strong features, good training data is also important. For some applications, synthetic training samples can be generated to improve the quality of a given training set. In the current application, it is possible to apply a data generation model to create synthetic samples from the existent. In the work by Simard et al. [18], an elastic distortion model for data generation is applied to off-line character recognition. Also, the Sigma-lognormal model of the kinematic theory of rapid human movement presented in the work by Plamondon et al. [19] has been used to produce very realistic synthetic on-line signatures and handwritten gestures.

The quality of a training set is not only determined by its size, but also by having a good representation for each symbol class. In this sense, a balanced representation of classes is helpful to make symbol classifiers that base their decisions mainly on the actual shapes of the symbols rather than based only on the prior probabilities for each class. We would like our sample to be representative and to have a balance between

classes. Ideally, a good data generation strategy can enhance a dataset even with limited amounts of synthetic samples.

Nevertheless, it has to be considered that on a recognition problem there might be different styles for a single class [20]. For handwritten symbols, these styles go beyond small differences in slant and thickness to the point of having completely different shapes that still belong to the same class like for example the lowercase z character written in cursive style. If a given writing style is absent from the training set, then it will be very difficult or even impossible for a classifier to learn to correctly identify symbols using that style.

In our work we use a data generation strategy based on balancing class representation. The CROHME [6], [7] training and testing datasets are highly biased. For this reason, we used average per-class accuracy as an indicator of how the system performs on the individual classes.

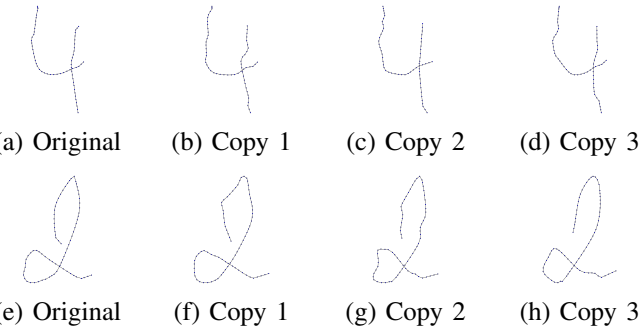
II. Strategy for data generation

Given a distortion model and a training set, additional data can be generated in any convenient way. Ideally, more data can produce more accurate classifiers. However, there is no guarantee that higher accuracy will be achieved by getting more samples from what is already in the training set. In fact, we could easily obtain lower accuracies by multiplying noisy samples. In this work we use a data generation strategy that simply balances the class representation in the training set.

Our strategy for data generation is very simple: all classes should have at least a given number of samples. The main parameter for this method is the percentage T of the size of the largest class C that will be the new minimum of samples per class $min\_count = T * |C|$. For each class, if the number of elements is smaller than min_count, then the existing samples are used to create distorted copies of them until the size of the class is equal to min_count. Note that if $T \geq 1.0$ then the resulting dataset will be perfectly balanced.

For the generation of synthetic samples we use an elastic distortion model similar to what Simard et al. [18] used before for symbol recognition. One important difference in the current model is that our input data is on-line instead of images. In this sense, a distortion can be easily created by just randomly moving the points of each trace around their original positions. However, if the points are moved too far away from their original positions, or if two contiguous points are moved in completely opposite directions, the resulting sample might not be a realistic variation of the original.

In order to attempt to create realistic variations of the original shapes in our system, we used Perlin Noise [21] to generate smooth noise maps that control how much should we move each point in the shape, and because these noise maps are smooth, contiguous points are very likely to move on similar directions making the final result credible. Besides the parameters of the Perlin Noise maps [21], the system defines the maximum level of distortion to apply to a given point. This is the maximum distance that a point can travel from its original location, and it is defined as a proportion of the length of the diagonal of the bounding box that contains the trace being distorted. Empirically, we found that 30% of this length produces credible but highly distorted samples and that about 9% of this length was more than enough distortion to create visually acceptable variations with a positive impact in the training process. An example of our model applied to distort two symbols is shown in Figure 1.



(a) Original (b) Copy 1 (c) Copy 2 (d) Copy 3

(e) Original (f) Copy 1 (g) Copy 2 (h) Copy 3

Fig. 1. Examples of results for our distortion model. (a) and (e) are the original symbols, and (b)-(d), (f)-(h) are distorted versions of each respectively.

III. Symbol Recognizers

Three factors must be considered for the creation of a recognizer of symbols for on-line math expressions. The first one is a preprocessing procedure to mitigate the noise present in the input data. The second factor is the set of features used to describe the symbols. Finally, the third factor is the machine learning technique used for classification and in our case we are comparing four: AdaBoost.M1 [1] with C4.5, Random Forests [3], and SVM with linear and RBF kernels.

A. Preprocessing of input

Our preprocessing procedure is an adaptation of the method in [22]. Three main stages can be identified on the process: Removal of duplicated points and hooks, then smoothing of the trace and finally size normalization of the symbol. However, there are two minors differences in our approach. First, during the removal of duplicated points, we only remove them if the length of the sub-trace connecting them is smaller than 10% of the length of the diagonal of the bounding box that contains the whole trace. The second difference is the cubic spline used for the final re-sampling of the shape which in our case we use Catmull-Rom splines [23]. After size normalization, the resulting symbols will have all their coordinates contained inside a 2-by-2 box centered at the origin.

B. Features

One of our goals was to use features that were both simple and strong for shape description. Our input data is on-line which implies that tracing information is available and can be used to compute features that cannot be easily computed for off-line data. However, we avoided relying on information that describes how the trace was drawn because of the high degree of variation introduced by the different writing styles.

We considered different features and tested many combinations of them and finally we produced a set that is both simple to compute and achieves high accuracy in our training

and testing sets. These features can be divided into four main categories: global features of the symbols (11), crossings features (30) 2D fuzzy histograms of points (25) and fuzzy histograms of orientations (36). A total of 102 values are used to describe every symbol in our system.

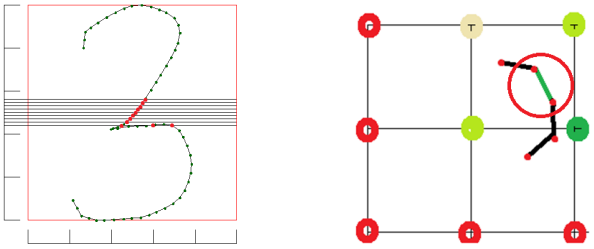
1) Global features of the symbols: These features represent individual values that describe the complete symbols and provide general information about their shapes. The current set of global features is: number of traces (1), angular change (2), line length (2), number of sharp points [22] (2), aspect ratio (1), mean x coordinate (1), mean y coordinate (1) and covariance of x and y coordinates (1). For angular change, line length and number of sharp points, two values are added: the total sum and the average of the values for all traces.

2) Crossings features: This kind of features has been previously used for off-line character recognition [24]. Given any arbitrary line, it is possible to compute the number of times that it intersects the traces of a symbol. Also, if the line intersects multiple times the traces, then the first and the last intersections can be useful to describe the shape at that location. If we extend this idea to a set of lines at fixed vertical and horizontal positions, then we get a set of features useful for classification of shapes of symbols.

However, one of the issues of these features is their sensitivity to small variations in writing style. A way to mitigate this issue is using the average of many parallel lines to describe a region instead of a single line. Currently, X and Y axis are divided into five regions each, and for each region nine sub-crossings are computed and the average of those sub-crossings is used to describe the region. Three average values are computed per region: count of intersections and positions of first and last intersections. The entire shape is describe horizontally and vertically using $5 \times 3 + 5 \times 3 = 30$ values.

Figure 2.a shows an example of horizontal crossings for the number 3. In this example, the lines in the middle represent the horizontal sub-crossings of that region of the symbol. Most sub-crossings on this region intersect the trace exactly once, but the last intersects three times. The example shows how little variations could lead to large changes for crossings values if only one line is used per region. However, by using the average of nine lines the final value of crossing counts for this example will be very close to one.

3) 2D Fuzzy histogram of points: The normalized region that contains the symbol is divided into a grid of 4-by-4 cells, and then each point on each trace is assigned a fuzzy membership value for each of the four corners of the cell of the grid where it is contained. A grid of n-by-n cells will contain a total of $(n+1) \times (n+1)$ different corners. In this case, $5 \times 5 = 25$. For a given point $P = (x_p, y_p)$ and a given corner $C = (x_c, y_c)$ the membership value m_p is defined as $m_p = \frac{w-|x_p-x_c|}{w} \times \frac{h-|y_p-y_c|}{h}$, where w and h refer to the width and height of the cell respectively. At the end, all membership values are added on each corner, and then these values are divided by the total of points in the symbol producing a normalized 2D fuzzy histogram of the coordinates of the points of the symbol.



(a) (b)

Fig. 2. Two kind of features used by our system. (a) Horizontal crossings with average of sub-crossings. Each horizontal line in the middle represents a sub-crossing. Intersections are marked with larger dots. (b) Fuzzy Histogram of Orientations. The circled line segment is affecting the four corners of the cell where it is located and most heavily the one in the middle-right.

4) Fuzzy histograms of orientations: Histograms of orientations have been used before as a zoning approach for description of shapes for off-line optical character recognition [24]. The entire symbol area is divided into small regions using a grid of 2-by-2 cells. Similar to the previous feature set, the grid defines $3 \times 3 = 9$ corners, and for each of them the slopes of the closest line segments are grouped into four bins. The grouping criterion used is the angle formed by that slope and the horizontal axis. Note that any slope can be represented by an orientation angle in the range $\left[-\frac{\pi}{8}, \frac{7\pi}{8}\right]$, and then that range can be divided into four bins of size $\frac{\pi}{4}$. In that sense, we could think of the first bin $\left[-\frac{\pi}{8}, \frac{\pi}{8}\right]$ as the representation of horizontal lines in the region. Similarly, the third bin $\left[\frac{3\pi}{8}, \frac{5\pi}{8}\right]$ represents the vertical lines, and the other two bins represent the two diagonal directions. These features consists of a total of $9 \times 4 = 36$ values.

Using a grid as described before, we create fuzzy histogram of orientations of line segments by using different weighting criteria. Consider that the middle point of each line segment falls inside of a cell in that grid, then the first weighting criterion determines the membership of the line segment for each of the four corners of the cell based on the relative distance to each corner. This membership system is illustrated in Figure 2.b. and is computed with the same formulas used for m_p in the 2D fuzzy histograms of points by defining the point $P = (x_p, y_p)$ as the middle point of each line segment. The second weighting criterion is the actual slope angle and it makes each line segment a member of the two angular bins with the closest ranges to its orientation. The distance between a given angle and a range is computed using the absolute angular distance between that angle and the value in the middle of that range. Note that angular distance is circular and some slopes can become members of both the first and the last bins. In summary, a single segment will affect a total of 8 bins, 2 angular bins per each of the 4 corners of the cell where its middle point is located.

The third weighting criterion is the relative length of the line segment. Initially, individual histograms are computed for each trace of the symbol and the contribution of each line segment is given by its own length divided by the total length of that trace. The next step is to combine the individual histograms of the

traces into a single histogram for the whole symbol, and this is done by weighting the entire histogram of each trace using the length of that trace divided by the sum of the lengths of all traces. After re-weighting the individual histograms, these can be added up into a single histogram.

The fuzzy histograms are expected to be more tolerant to small variations that their non-fuzzy versions. Consider the case of a value that lies in the border between two bins in a non-fuzzy histogram. A very small change in the value could make it fall into a different bin. However, for the fuzzy case such change will only represent a small modification to the contribution of that value to the closest bins.

C. Classifiers

Four alternative classification algorithms were considered. Each of them has its own advantages and disadvantages. For example, the most accurate requires more time to classify a symbol, and the ones that provide faster classification times can also require longer training times. The following subsections define each of them.

1) AdaBoost.M1 [1] with C4.5 Decision Trees [2]: AdaBoost requires the use of a weak learner with at least 50% of accuracy and C4.5 decision trees fulfill this requirement in our current datasets. Of all the different versions of AdaBoost, we are using the AdaBoost.M1 algorithm [1]. The greedy approach described by Quinlan [2] for training of the individual decision trees based on the calculation of the gain ratio has been adapted to work with weighted samples. Also, when the tree is evaluated, the class probabilities are no longer computed using the counts of samples per class, but instead using the total weights per class divided by the total weight in the current node for use with boosting. Each decision tree is pruned using the pessimist error estimation with confidence intervals for the binomial distribution as done by Quinlan.

The main idea of boosting is that higher accuracy can be achieved by combining the output of multiple weighted classifiers. The algorithm is based on rounds were each round attempts to classify correctly the samples that were misclassified on previous rounds using a weighting system for each individual sample. Additional details can be found in work by Schapire [1]. In our tests, we set the number of boosting rounds to a maximum of 50.

2) Random Forests [3]: A different ensemble method based on decision trees. It has the advantage of high parallelization as each decision tree on the ensemble can be trained and evaluated independently of the others. We used the implementation of Random Forests included in the scikit-learn [25] library for python. It trains each tree using a bootstrap sample drawn from the training set, and randomizes the available attributes at the time of computing the best splits. No pruning is performed over the learned decision trees but their depth is limited. For our classifier, we used 50 trees with depths above 20. We set the limit of available features at each split to 30.

3) Support Vector Machines: An adaptation of SVM for multi-class classification [5] is being used in our system. We use the implementation provide by the scikit-learn [25] library for python. Two different kernels were tested: Linear and RBF. SVM with linear kernels can be easier to configure than SVM with RBF kernels. In the other hand, SVM using RBF kernels usually achieved the highest accuracy once that the right values for their C and Gamma parameters were found. The C parameter defines the simplicity of the model while Gamma controls the influence of each training sample. In our work we applied grid search to find good values for these parameters which achieve the highest accuracy for each training set. The greatest advantage of SVM is that their accuracies are usually the highest and can be used as a reference when optimizing the parameters of other classifiers. The greatest disadvantage of SVM is their evaluation time for any given sample which might be due to the fact that several SVM trained for binary classification must be evaluated before deciding the final class of the sample.

IV. Experimental Results

For the first set of experiments with synthetic data generation we used the training set from CROHME 2013 [7] that contains 68598 valid samples of handwritten symbols, and two testing sets, one from part III of CROHME 2012 [6] with 6336 samples and the second is from CROHME 2013 with 6082 samples. The main different between these testing sets is the total number of classes. For CROHME 2012, 75 different class of math symbols are included. In the case of CROHME 2013, 101 classes are available. It is important to mention that these 101 classes are a super set of the original 75 in the first testing set. A custom training set was obtained by extracting the samples of the 75 common classes from the CROHME 2013 training set and the classifiers trained with this data were evaluated using CROHME 2012 testing data. This second training set has a total of 65544 samples.

Each of the original CROHME training sets is heavily biased toward very few classes. For example, only 5 classes (-, 1, 2, +, x) out of the 101 present in the CROHME 2013 dataset represent more than 35% of the whole dataset and just by adding 4 more symbols (=, "(", ")", 3) they account for more than half of the entire dataset. A similar bias is also present on the testing data. In order to produce less biased classifiers and at the same time attempt to reach higher recognition rates we applied our data generation strategy to balance the class representation. We tested different amounts of synthetic data as shown in Figure 3. As we expected, additional samples stop contributing to the accuracy of the system after a large number of samples has been created. For each test we include the global testing accuracy and also the average per-class accuracy with its corresponding standard deviation. From these results we decided that we would use a maximum of 70% of the largest class as the new minimum size per class for balancing on the CROHME 2013 (101) training set, and a maximum of 60% for the CROHME 2013 (75).

Given that the expanded version of each training set is relatively large compared to the original, 68598 vs 451637 for CROHME 2013 (101) and 65544 vs 291192 for CROHME 2013 (75), and that for each symbol a total of 102 features are

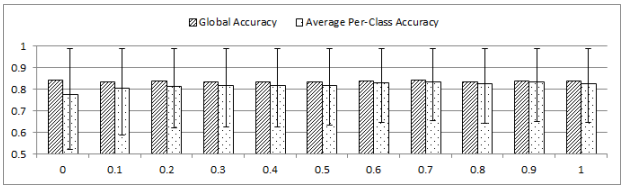


Fig. 3. Effect of generating different amounts of data. The axis represent the percentage (T) of the largest class. Note that for CROHME 2013 (101) the global accuracy does not improve. However, the average per-class accuracy increases from 77.79% when $T = 0.0$ to 83.30% when $T = 0.7$. Also, its standard deviation decreases from 25.49% to 18.01%. No further improvements were obtained with larger values for T

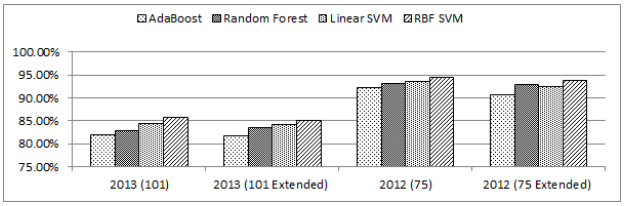


Fig. 4. Comparison of global accuracy using different classifiers. Here, 2012 and 2013 refer to the testing set used while (75), (101) and extended refer to training set and whether that training set was expanded or not.

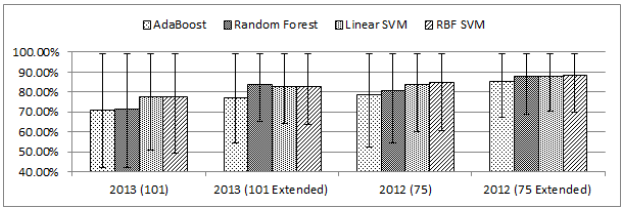


Fig. 5. Comparison of average per-class accuracies using different classifiers. The extended training set always achieve higher average per-class accuracy

used, we considered the use of Principal Component Analysis (PCA) for dimensionality reduction. However, this led to smaller accuracies even after using up to 99% of variance. For this reason we decided to keep the original set of features for our current analysis. Figure 4 shows a comparison of the global accuracy of the four classifiers over two different testing sets. It also shows that global accuracy values were usually lower when the extended training sets were used. SVM classifiers achieved the highest accuracies, specially the one using RBF kernel. Random Forest usually achieved higher accuracy than AdaBoost with C4.5. The best testing accuracies are 85.89% for CROHME 2013 and 94.49% for CROHME 2012. However, a direct comparison with CROHME results might not be possible as CROHME 2013 [7] does not provide accuracy for isolated symbol classification. In the case of CROHME 2012, the best accuracy reported on the original competition was 96.85%, but this value is calculated over a subset of correctly segmented symbols from entire math expressions and not from isolated symbols. Figure 5 compares the different average per-class accuracies and their standard deviation. Extended training set always produced higher average values with smaller standard deviations.

A very important concept to keep in mind when balancing classes is that some symbols are ambiguous by nature. For example, s and S are symbols that can only be distinguished

TABLE I
COMPARISON OF ACCURACY OF DIFFERENT METHODS ON MATHBRUSH

Method	Classifier	Top - 1	Top - 5
Hu et al. [14]	HMM	82.9%	97.8%
Alvaro et al. [15]	RNN	89.4%	**99.3%**
MacLean et al. [26]	Greedy DTW	85.8%	99.1%
Proposed Method	AdaBoost C4.5	88.4%	98.7%
	Random Forests	87.9%	98.4%
	SVM Linear Kernel	88.6%	99.1%
	SVM RBF Kernel	**89.8%**	99.1%

using context information. By plotting the accuracy of individual classes before and after the expansion of the dataset we can observe how certain classes increase in accuracy while others decrease. Figure 6 shows the plot for CROHME 2013 (101) including only classes with more than 5% of difference in accuracy. Current results suggest that ambiguous classes make trade-offs during the balancing process.

Considering the existence of ambiguous classes, we decided to recompute the global accuracy of our best SVM RBF classifiers if errors between ambiguous classes were ignored assuming that these could be solved later using context information. The ambiguous pairs of classes were the following: x-X, x-\times, X-\times, 1-|, (-|,)-|, 1-(, 1-), 1-/, 1-COMMA, c-C,)-COMMA, p-P, \prime-COMMA, \prime-|, v-V, s-S, q-9, o-0, \prime-/, /-COMMA. We achieved 93.52% for CROHME 2013 (101) and 96.36% for CROHME 2012 part III.

Since direct comparison with results from other systems using CROHME datasets is difficult, we tested our system using the MathBrush [8] database. The database contains 100 classes of symbols, but following the procedure of other approaches using this dataset [14], [15], seven classes of symbols were discarded: $\leq, \neq, <, \lambda, \Omega$, comma and dot. Table I contains a comparison of Top-1 and Top-5 accuracies for the following approaches on MathBrush dataset: Hu et al. [14] using Hidden Markov Models (HMM) and Pen-Up/Down features, Alvaro et al. [15] using Recurrent Neural Networks (RNN) and hybrid features, MacLean et al. [26] using greedy Dynamic Time Warping (DTW), and finally our approach using the four classifiers described in section III-C. We tested each classifier using 20 experiments as described in [15] using exactly the same partitions of data. Note that in [26] only 70 classes of single-trace symbols were used.

V. CONCLUSION

We have presented an approach for on-line recognition of handwritten math symbols. We compared the accuracy of our approach using different machine learning techniques and obtained relatively small differences in terms of global accuracy. Math symbols are described in our approach mainly using off-line features. Note that we never converted our symbols to an off-line representation, instead we converted the computation of these features to an on-line version. It is worth to mention that our initial tests suggested that crossings features are by far the strongest in our current set of features. Our features mainly aim to describe the shapes of the symbols, and we achieve recognition rates that are competitive with

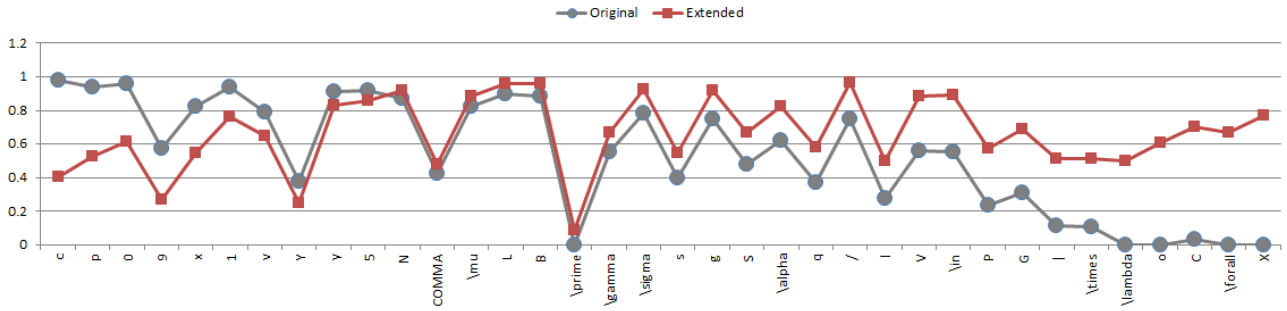


Fig. 6. Classes with more than 5% of change in accuracy due to extension of the training set. Note that classes like "x" suffer a great loss of accuracy while classes like "X" and "times" have a great gain.

other approaches that use the on-line data to compute time-based features.

Our experiments shown that for certain datasets like CROHME, data generation might produce subtle benefits that might be missed if we only pay attention to the global accuracy of the system. In most of our test, extended datasets created classifiers with slightly lower global accuracy but better average per-class accuracy with smaller standard deviation. This loss of global accuracy was mainly due to the existence of ambiguous classes in the dataset as trade-offs took place between pairs of ambiguous classes.

ACKNOWLEDGMENT

The authors would like to thank Dr. Pengcheng Shi, Danilo Dominguez, Francisco Alvaro, Lei Hu and the reviewers for their valuable feedback and shared data. This material is based upon work supported by the National Science Foundation (USA) under Grant No. HCC- 1218801

REFERENCES

[1] R. E. Schapire and Y. Freund, *Boosting: Foundations and Algorithms*. The MIT Press, 2012.

[2] J. R. Quinlan, *C4. 5: programs for machine learning*. Morgan kaufmann, 1993, vol. 1.

[3] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[4] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[5] T.-F. Wu, C.-J. Lin, and R. C. Weng, "Probability estimates for multi-class classification by pairwise coupling," *The Journal of Machine Learning Research*, vol. 5, pp. 975–1005, 2004.

[6] H. Mouchère, 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.

[7] H. Mouchere, C. Viard-Gaudin, R. Zanibbi, U. Garain, D. H. Kim, and J. H. Kim, "Icdar 2013 crohme: Third international competition on recognition of online handwritten mathematical expressions," in *Document Analysis and Recognition (ICDAR), 2013 12th International Conference on*. IEEE, 2013, pp. 1428–1432.

[8] S. MacLean, G. Labahn, E. Lank, M. Marzouk, and D. Tausky, "Grammar-based techniques for creating ground-truthed sketch corpora," *International Journal on Document Analysis and Recognition (IJDAR)*, vol. 14, no. 1, pp. 65–74, 2011.

[9] R. Zanibbi and D. Blostein, "Recognition and retrieval of mathematical expressions," *International Journal on Document Analysis and Recognition (IJDAR)*, vol. 15, no. 4, pp. 331–357, 2012.

[10] H. Mouchère, C. Viard-Gaudin, D. H. Kim, J. H. Kim, and U. Garain, "Crohme2011: Competition on recognition of online handwritten mathematical expressions," in *Document Analysis and Recognition (ICDAR), 2011 International Conference on*. IEEE, 2011, pp. 1497–1500.

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

[12] U. Garain and B. B. Chaudhuri, "Recognition of online handwritten mathematical expressions," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 34, no. 6, pp. 2366–2376, 2004.

[13] A. H. Toselli, M. Pastor, and E. Vidal, "On-line handwriting recognition system for tamil handwritten characters," in *Pattern Recognition and Image Analysis*. Springer, 2007, pp. 370–377.

[14] 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 *Document Analysis and Recognition (ICDAR), 2011 International Conference on*. IEEE, 2011, pp. 457–462.

[15] F. Alvaro, J.-A. Sánchez, and J.-M. Benedi, "Classification of on-line mathematical symbols with hybrid features and recurrent neural networks," in *Document Analysis and Recognition (ICDAR), 2013 12th International Conference on*. IEEE, 2013, pp. 1012–1016.

[16] F. Álvaro, J.-A. Sánchez, and J.-M. Benedí, "Recognition of on-line handwritten mathematical expressions using 2d stochastic context-free grammars and hidden markov models," *Pattern Recognition Letters*, vol. 35, pp. 58–67, 2014.

[17] A. Delaye and E. Anquetil, "Hbf49 feature set: A first unified baseline for online symbol recognition," *Pattern Recognition*, vol. 46, no. 1, pp. 117–130, 2013.

[18] P. Simard, D. Steinkraus, and J. C. Platt, "Best practices for convolutional neural networks applied to visual document analysis." in *ICDAR*, vol. 3, 2003, pp. 958–962.

[19] R. Plamondon, C. OReilly, J. Galbally, A. Almaksour, and É. Anquetil, "Recent developments in the study of rapid human movements with the kinematic theory: Applications to handwriting and signature synthesis," *Pattern Recognition Letters*, vol. 35, pp. 225–235, 2014.

[20] P. Sarkar and G. Nagy, "Style consistent classification of isogenous patterns," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 27, no. 1, pp. 88–98, 2005.

[21] K. Perlin, "An image synthesizer," *SIGGRAPH Comput. Graph.*, vol. 19, no. 3, pp. 287–296, 1985.

[22] 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.

[23] E. Catmull and R. Rom, "A class of local interpolating splines," *Computer aided geometric design*, vol. 74, pp. 317–326, 1974.

[24] Ø. D. Trier, A. K. Jain, and T. Taxt, "Feature extraction methods for character recognition-a survey," *Pattern recognition*, vol. 29, no. 4, pp. 641–662, 1996.

[25] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[26] S. MacLean and G. Labahn, "Elastic matching in linear time and constant space," in *Proc., Ninth IAPR Intl. Workshop on Document Analysis Systems*, 2010, pp. 551–554.