Artificial Offline Character Recognition

Shi Wang, Hongzhang Cheng, Xiaoyan Sun and Tyler Colombo
Department of Computer Science,
Worcester Polytechnic Institute
Worcester, MA
Email: hcheng3, swang11, xsun3, tjcolombo
@wpi.edu

***Abstract*—Hybrid algorithms are now being created and used to better the accuracy of character recognition. This is interesting with all the different algorithms and features being used, but stand alone algorithms are a thing of the past. From reading various papers and understanding the theory behind hybrid algorithms it shows that they produce slightly better results than the algorithms standing alone. We want to figure out what two algorithms would be most beneficial to combine so that we could create a more efficient hybrid algorithm. To test our algorithm we will provide an online interface to write a character so we can see its performance and how it stacks up to other algorithms. The two algorithms we are focusing on combining are Support Vector Machines (SVM) and Affinity Propagation because we believe these two will better predict offline character recognition. We choose two algorithms that have not been combined at this point in time. Along with combining the algorithms we have also combined various different features to see which helps perform a better recognition.**

***Index Terms*—Hybrid Algorithm, SVM, Affinity Propagation, Character Recognition**

1. Introduction

Character recognition is a form of pattern recognition process and image processing. In reality, it is very difficult to achieve perfect accuracy. In this overview, Offline Character Recognition (OCR) is used as an umbrella or group term, which covers all types of machine recognition of characters in various application domains. Character Recognition is always on the rise with finding what could be use to gain better performance. Offline character recognition is the image of writing converted into bit pattern or called digitization by an optically digitizing device such as optical scanner or camera in our case an web interface canvas. The recognition is done on this bit pattern data for machine printed or handwritten text. The research and development is well progressed for the recognition of the machine printed documents. In recent years, the focus of attention is shifted towards the recognition of hand-written script too. The main advantage of the offline recognizers is to allow the previously written and printed texts to be processed and recognized. [7] Nowadays hybrid algorithms are being created to try to improve the performance and gain better results rather than stand alone algorithms. Support vector machines are being widely used for binary classification and affinity propagation finds "exemplars", members of the input set that are representative of clusters. SVM, which is not new for the use in character recognition. Whereas affinity propagation has only been around for a few years. These two classifiers have not been used together as of yet. Selection of a relevant feature extraction method is probably the single most important factor in achieving high recognition performance with much better accuracy in character recognition systems. Though just selecting one feature may not be enough, combining multiple features can help improve the system's accuracy of character recognition.

2. Overview

For our system we developed a web interface canvas where users can sketch a number and through our features and classifiers a prediction will be done. The formal digit result will be shown on the website. We built our own training data which has about 1500 hand writting digits. For the back-end work we chose to combine multiple features such as profiling, density, and direction. This was to help us to understand what combination of features that would increase the accuracy of recognition as much as possible. Then we used SVM and affinity propagation. Besides the combination of these two algorithms we used Max Win and Dag-SVM for multiple class classifying method. For data we created our own using a drawing interface as well as an existing dataset from UCI called optical recognition of handwritten digits data set. This way we could have as much data to test our system as needed. In Figure 1 it is shown how we created classifiers for our experiment and testing.

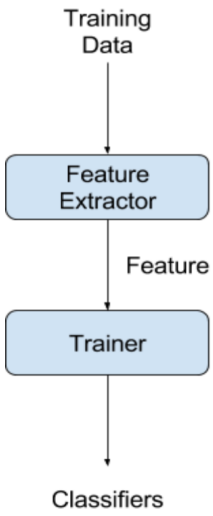


Figure 1: How Classifiers Were Created

3. Web Interface

The web interface is built locally with the python Flask framework. The data is being sent using POST and GET methods with JSON data between the website and the local python files. The website character recognition system could be easy to navigate by users and getting the formal style of handwriting character inputs quickly.

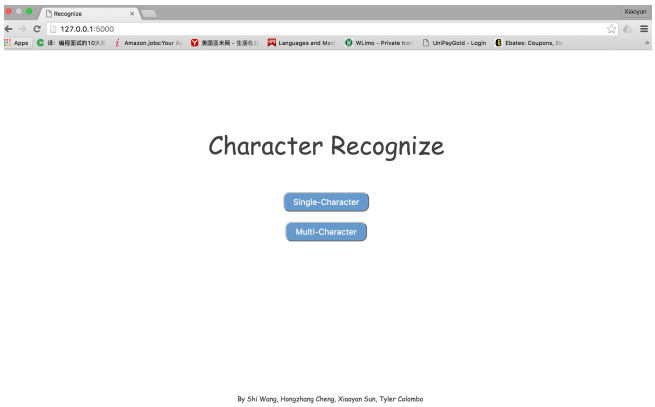


Figure 2: Screen of Web Interface Main Page

1) Interface Design
The design is based on single character recognition and could be extended to multi-character recognition based on the function and algorithm implementation in the future.
2) Main Page:
Selecting the single character page or multi-character page based on the user's needs and choices. As some in Figure 2.
3) Draw Page:
The canvas is built so that it can be drawn on by users with ease. Users can draw single characters with different colors using the mousepad, hand writing tab, or touch screen. The image drawn on the canvas could be transferred into matrix form and sent to python algorithm functions. After the classification, the formal result from the algorithm will be shown on the website. Then the canvas can be cleared after the button is clicked. The Draw Page and canvas are shown in Figure 3.

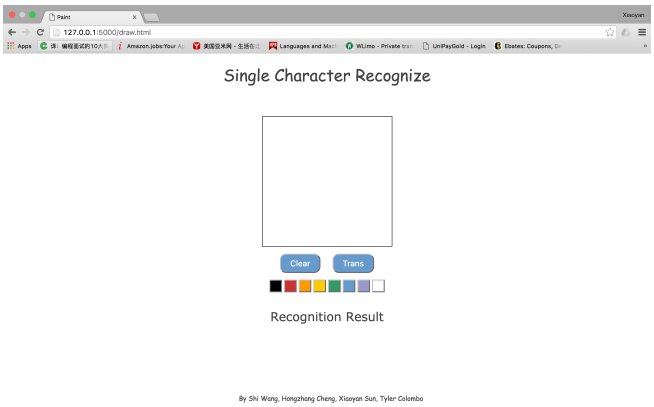


Figure 3: Screen of Web Interface Draw Page

4) Data Transfer
The drawing context on the canvas could be transferred to matrix format and sent back to python functions with a POST method. The calculated result could then be sent back to website in JSON format and shown on webpage. The data transfer method is shown in Figure 4.

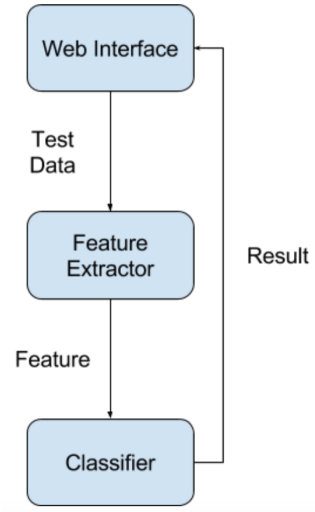


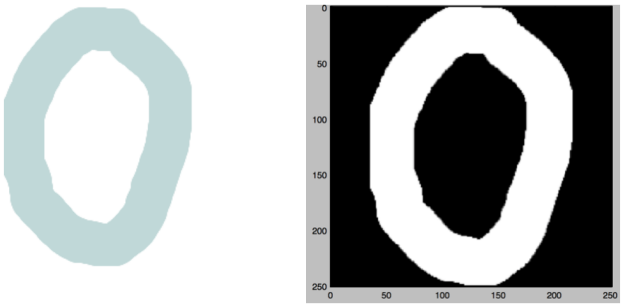
Figure 4: How the Web Interface Works

4. Feature Extraction

The selection of a feature extraction method is probably the single most important factor in achieving high recogni-

tion performance in character recognition system. Different feature extraction methods are designed for different representations of the characters, such as solid binary characters, character contours, character skeletons. [11]

Before applying the different feature extraction methods, we first apply preprocessing to the image data that we got from the interface. The image does not start as a binary image data, so we had to turn it into a binary form. After that, in order to deal with the different size people will write, we resize and universe the data. For resizing we used the method in the scikit-image module to resize the image data. For universe, we at first get the boundary of the character in the image. Then compared its length and width to get the larger one. From there, we use this larger one to create a image square to store the character in the middle of it. So far, we get a image, which is in a fixed size and universe and the character is in the middle of it. After preprocessing, we get a binary image that is 255x255 pixels and the character is in the middle of the images. The before and after of the image is shown in Figure 5.



(a) Before Preprocessing (b) After Preprocessing

Figure 5: Before and After of Preprocessing of the Character Image

After the preprocessing procedure, we applied four different techniques to extract the feature from the image data and combine some of them in order to find the best features to use. Those methods we used are are profiling, direction, and density.

4.1. Profiling:

To get the profiling feature of an image, we need all Location Transition (LT) values of that image. Transition location means when we scan one line of an binary image, the position which value of its pixel transit from 0 to 1 or 1 to 0. First, we scan each row in the image from left to right and right to left. Likewise, each column in the image will also be scanned from top to bottom and bottom to top. The location transition value, which is the value at the transition location, in each direction are computed as a fraction of the distance traversed across the image. A maximum value was defined to be the largest number of transition that may be recorded in each direction. If there were less than max transitions recorded, then the remaining transitions would be assigned values of 0, in order to make the a uniform feature vector for each of the character.

Figure 6 is an example to illustrate how the transition value is computed and how to use the number to represent the direction of the lines in the character. This example is from the paper [1]

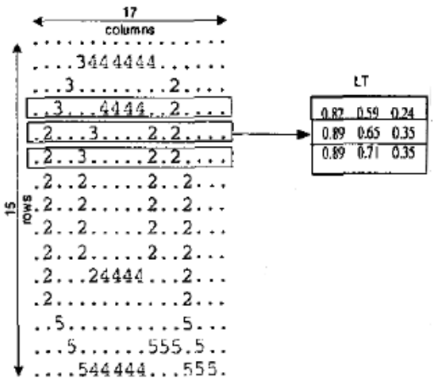


Figure 6: Representation of Profiling and Direction

In our system the max transition that is recorded for a single line is 2. After the profiling we could get 255 * 2 feature vectors for each direction. So for the four directions there will be 2040 feature vectors. However, this number is too big to process and the classifying part will be very slow. So we reduce the numbers of the features for each direction by getting the median of a certain number of features. For example, in Figure 6, we could get one line of feature from the LT, which is (0.87,0.65,0.31). In this case, the number is 3 and in our project we used 5 to get the features. So for the profiling features we got 408 vectors. As you can see in Figure 7.

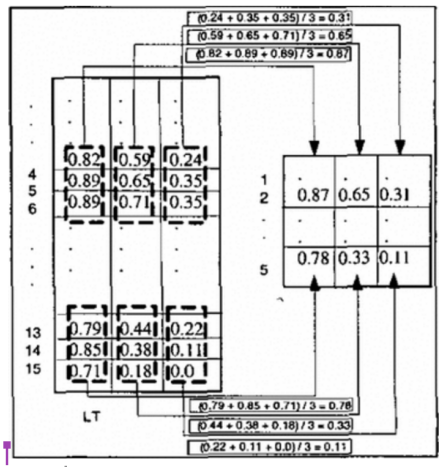


Figure 7: Example of Profiling Feature Reduction

4.2. Direction:

To get the direction feature of an image, we needed to first get the edges of the character. We have two ways of

doing of getting this information. First is directly skeletonize it and getting the skeleton of the character or use the canny edge detection to get the edge of the character. For this part we used both of them, but in the end we decided to only use the skeleton of the character to extract the direction feature. After using the skeletonize method we needed to do the zoning, which is separating the image into certain parts and write an algorithm to get the line and represent their direction as numbers. One single zone is showed in Figure 6. For the representation of direction the number 2 represent vertical, 3 represents left diagonal, 4 represents the horizontal, and 5 represent right diagonal. The algorithms we used is from the paper [4] .

The basic idea of this is at first to find the start and the intersection of the lines in the image and then use them recursively to populate the whole line and use the number to represent their direction. In this process we could find the new start then go through the new start until there is no start. In that case all the lines are found in the image and able to represent them in number form. After zoning we get 9 zones for each of the characters. So in each zone we record two directions to represent that zone. When we got all the directions of the zones we transform it into a binary form. For example, 2 will be [1000], 3 will be [0100], 4 will be [0010], and 5 will be [0005]. We did this transformation because in the experiment of classifying the representation of direction by using (2,3,4,5) is not accurate. At last, we got the number of the direction for one image feature.

4.3. Density:

To compare with the feature extraction way used before getting the density feature is a pretty simple way., we used density of each zones as another feature of this character. After zoning the image,for example in our case 9, we got 9 same size zones, due to the image being in a binary form. So we calculated the proportion of the 1s in the whole zone as its density features. Then we save all the density feature for each zone as the density features for this character. As you can see you in the Figure 8 ,when we zoning it to four zones, the density feature for the first zone(left up) will be 9/16. And the its density feature will be (9/16,8/16,11/16,11/16).

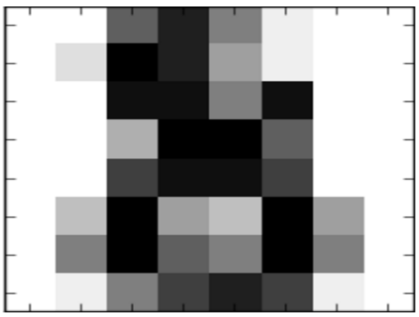


Figure 8: Example of Density Feature

Besides those three major ways to get the features, we also recorded the ration (height/width) of the character after getting the universe of the image as one of its features. After getting those features, we combine them together to create different kinds of new features to improve the accuracy of our recognition system. Until now, the data is ready, and the next part will be the classify.

After applying the technique of manifold learning (it's a feature visualization way), we made those features visualizable, there are 10 colors in each of the figure, each color represents a number shown in Figure 9. This is so we could directly know with different kinds of feature extraction were applied. The separation of the numbers are different, like in the figure all, except orange and yellow, most of the number(colors) are separated very well, in the density figure the grouping is good except for the colors in the middle(yellow, orange, and blue). For profiling there is again a good showing of the grouped numbers, better than density and direction but not as good as all. The direction feature by itself performed the worst out of all of them as you can see there is little to no grouping. Now with this information we are able to see what features perform better for differentiating certain numbers. Thus, we can combine certain features where some numbers(colors) are not performing as well with another feature that performs well with those numbers. All the combinations of features that were run are shown in Figure 10.

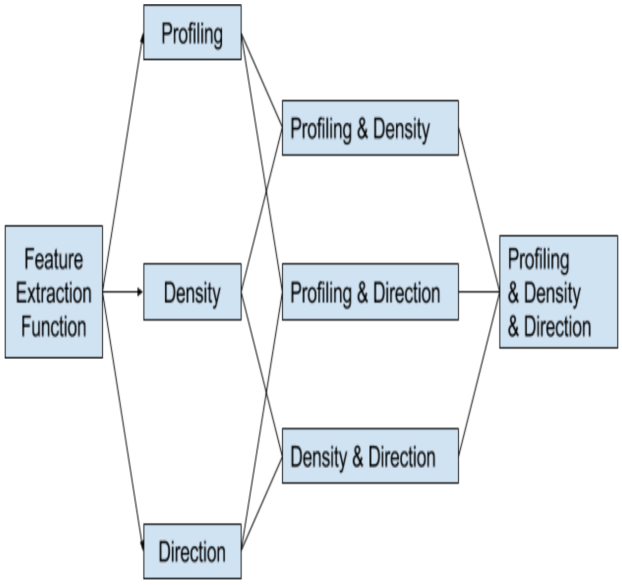


Figure 10: Combinations of Features

5. SVM

SVM (support vector machine) is a supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. It aims to

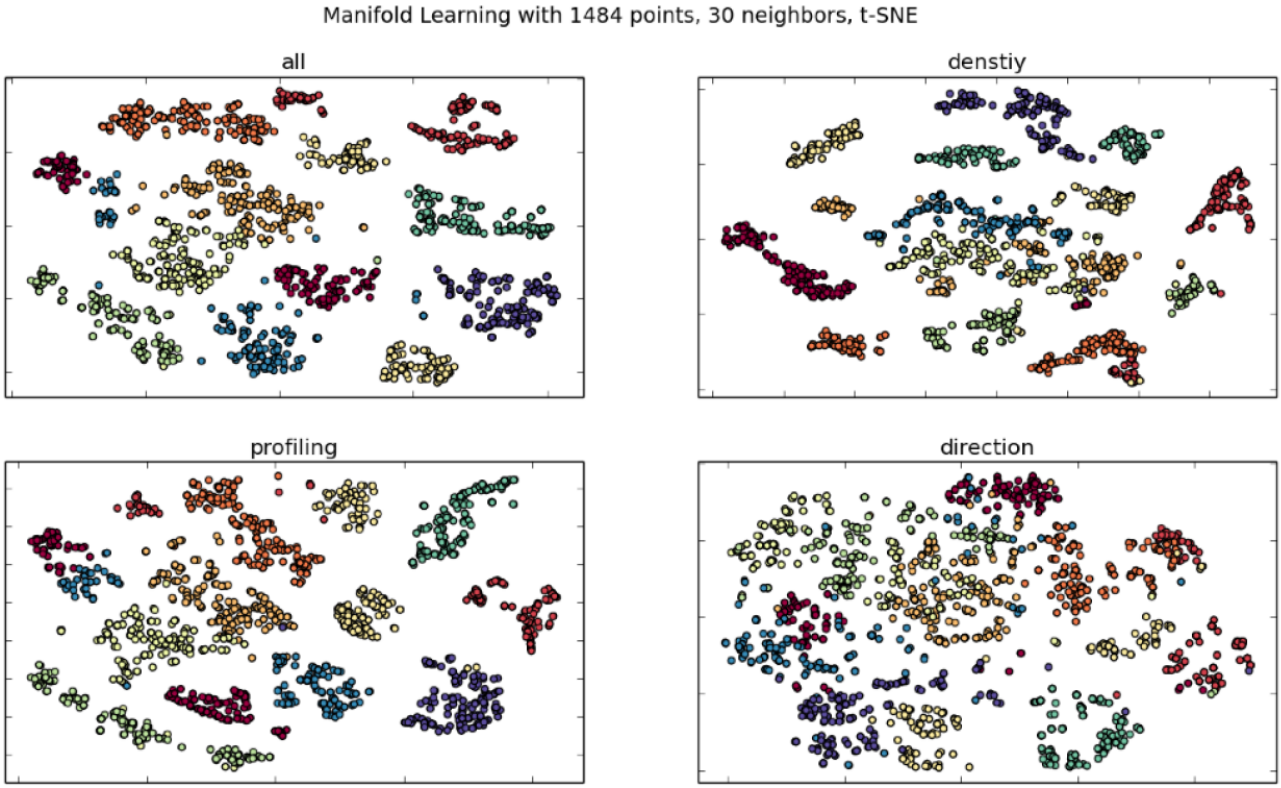


Figure 9: Representation of Manifold Learning for Features

classify the training example into one of the two categories using a maximum margin classifier. For the linear classification, it finds the linear boundary to separate the examples to their respected categories. For non-linear classification, it uses different kinds of kernel functions like gaussian kernel, polynomial kernel, or linear kernel to map their inputs into a high-dimensional feature space, which can be separable by a certain hyperplane in the high-dimensional space. Then based on this theory foundation, we can build a certain amount of classifiers, however many we want, to do the classification. For the multiple class classifying method we used Max Wins and Dag-SVM. [1]

5.1. Max Wins

Max Win is a one-against-one method. We will construct *k(k-1)/2* classifiers where *k* is the number of the classes we have and each classifier is trained on data from two classes. After the classifiers are constructed, we will use a voting strategy to decide which class the input belongs too. [1]

$$\min_{w^i, b^i, \xi^i} \quad \frac{1}{2}(w^i)^T w^i + C \sum_{j=1}^{l} \xi_j^i (w^i)^T$$
$$(w^i)^T \phi(x_j) + b^i \geq 1 - \xi_j^i, \quad \text{if } y_j = i$$
$$(w^i)^T \phi(x_j) + b^i \leq -1 + \xi_j^i, \quad \text{if } y_j \neq i$$
$$\xi_j^i \geq 0, \quad j = 1, \ldots, l$$

Figure 11: Function for Max Win

5.2. Dag-SVM

Direct acyclic graph SVM's training phase is the same as the one-against-one method by solving *k(k-1)/2*. However, in the testing phase, it uses a rooted binary directed acyclic graph, which has *k(k-1)/2* internal nodes and *k* leaves. Each node is a binary SVM of *i*th and *j*th classes. Given a test sample *x*, starting at the root node, the binary decision function is evaluated. Then it moves to either left or right depending on the output value. Therefore, we go through a path before reaching a leaf node which indicates the predicted class. [1]

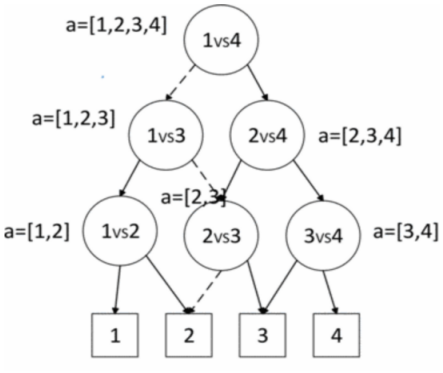


Figure 12: Construction Example of Dag-SVM

$$\min_{w^{ij},b^{ij},\xi^{ij}} \quad \frac{1}{2}(w^{ij})^T w^{ij} + C\sum_t \xi_t^{ij}(w^{ij})^T$$
$$(w^{ij})^T\phi(x_t) + b^{ij} \geq 1 - \xi_t^{ij}, \quad \text{if } y_t = i$$
$$(w^{ij})^T\phi(x_t) + b^{ij} \leq -1 + \xi_t^{ij}, \quad \text{if } y_t = j$$
$$\xi_t^{ij} \geq 0.$$

Figure 13: Function for Dag-SVM

6. Affinity Propagation and Hybrid Algorithm

Affinity propagation is a clustering algorithm based on the concept of "message passing" between data points, besides it is a new algorithm that takes as input measures of similarity between pairs of data points and simultaneously considers all data points as potential exemplars. Real-valued messages are exchanged between data points until a high-quality set of exemplars and corresponding clusters gradually emerges. Unlike clustering algorithms such as k-means or k-medoids, AP does not require the number of clusters to be determined or estimated before running the algorithm. it uniformly found clusters with much lower error than those found by other methods, and it did so in less than one-hundredth the amount of time. [12]

The basic idea of affinity propagation is creating two matrix the The "responsibility" matrix R and the The "availability" matrix A.then initialize both matrices to all zeros, then iteratively update them.

We could show the algorithm in Figure 14:

$$r(i,k) \leftarrow s(i,k) - \max_{k' \neq k}\{a(i,k') + s(i,k')\}$$
$$a(i,k) \leftarrow \min\left(0, r(k,k) + \sum_{i' \notin \{i,k\}} \max(0, r(i',k))\right) \qquad i \neq k$$
$$a(k,k) \leftarrow \sum_{i' \neq k} \max(0, r(i',k))$$

Figure 14: Affinity Propagation

In Figure 15 it shows how our hybrid algorithm works. First, affinity propagation is run and it tries to find the two nearest clusters and if those labels of the clusters are the same a result is produced. If the labels of the clusters are different then SVM is run with just a single classifier and produces a result.

In the training phase, we first used affinity propagation to group our training data into 40-50 clusters, and output the exemplars. The number of clusters were decided by the affinity propagation algorithm, and we set the ?preference? parameter to balance between training time and clustering performance. Then, we trained our multi-class SVM model, and output all 45 classifiers. In Figure 11 it shows how our hybrid algorithm work in the predicting phase. We first calculated the distance between the test sample to all the exemplars given by our affinity propagation training result. If the nearest exemplar and the second nearest exemplar give different labels, one SVM classifier corresponding to the two labels of the nearest two clusters was applied to the test sample. The output is then the SVM classifier result. For instance, if after comparing the distances between a test sample and all the clusters, we get the nearest cluster label ?2? and the second nearest cluster label ?3?, we considered the affinity propagation model not confident enough, and we call the 2-3 SVM classifier and output the result of the SVM classification. The benefit of the hybrid approach is that it increases the accuracy compared with only using affinity propagation, while costs relative same amount of predicting time.

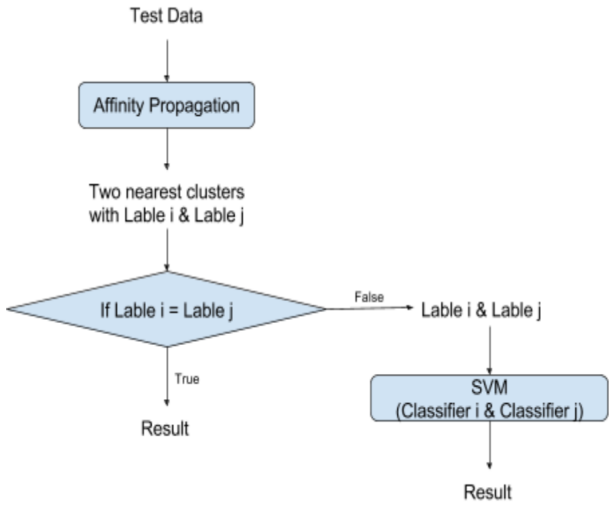


Figure 15: How our Hybrid Algorithm Works

7. Experiments

For the SVM part,except the max-win multi-classification method we also tried the DAG-SVM method, for DAG-SVM its faster than the max-win method when doing the prediction due to the process of prediction

only runs through 10 classifiers and max-win needs to run through 45 classifiers. Although we save the time for prediction,however, for DAG-SVM, due to its direct acrylic graph structure, the first pair needs to be very different, which means could be classified very accurate,so if the first pair classifier is not accurate, you will go through the totally wrong way and cant go back, in the other perspective to explain this, when a data is going through prediction, in each the classifiers it goes through, one class will be eliminated, so if the first two number is not enough different, There is some way like confidence for multiple class are mentioned in some paper, however at last we choose to implement the hybrid using affinity propagation, so the multiple classification is left aside.

To run the experiments we created our own training dataset consisting of 150 of each numerical character using an interface we design to save the number as a bitmap image. Then we created another 20 bitmaps of each numerical character for the testing dataset. We also used the Optical Recognition of Handwritten Digits Data Set from UCI Machine Learning Repository for additional data to use to test our system. UCI's dataset is 32x32 bitmaps that are divided into non-overlapping blocks of 4x4 and the number of on pixels are counted in each block. This generates an input matrix of 8x8 where each element is an integer in the range 0..16. This reduces dimensionality and gives invariance to small distortions. [13]

As shown in Figure 20 the accuracy of the features and the combinations of features are given along with the confusion each feature had with the number recognition. As you can see the combination of all three features had very few mistakes and a high accuracy rate.

We used a confusion matrix to show the result of the classification. In the confusion matrix, the value of (i,j (horizontal value, vertical value) means how many *i* was recognize as *j*. in this case when all the value is in the diagonal line is our ideal case, which means every number is recognized correctly. For example in the the matrix of feature profiling, the value of (0,0) is 21 which means there are 21 number 0 being recognized as number 0, and the value of (1,1) is 11, that means there are 11 number 1 being recognized as number 1, the value of (1,9) is 10 means there are 10 number 1 recognizes as 9. With the help of the confusion matrix, we could have a more clear understanding of the feature we used and how this features works in two specific numbers classification thus give us a intuition to find a better way to generate the feature. Below will be the some of the confusion matrix we generate from all of our features:

```
features_profiling
right: 195  wrong: 14
confusion matrix is:
[ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9.]
[[21  0  0  0  0  0  0  0  0  0]
 [ 0 11  0  0  0  0  0  0  0  0]
 [ 0  0 20  0  0  0  0  0  0  0]
 [ 0  0  0 21  0  0  0  0  0  0]
 [ 0  0  0  0 21  0  0  0  0  0]
 [ 0  0  1  0  0 20  0  0  3  0]
 [ 0  0  0  0  0  0 21  0  0  0]
 [ 0  0  0  0  0  0  0 21  0  0]
 [ 0  0  0  0  0  0  0  0 18  0]
 [ 0 10  0  0  0  0  0  0  0 21]]
overall accuracy is:  0.933014354067
```

Figure 16: Confusion Matrix for Profiling Feature

As we can see in Figure 16, while using profiling feature, the mistaken classification mostly happens to number 1, recognized as number 9 mistakenly.

```
features_direction
right: 155  wrong: 54
confusion matrix is:
[ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9.]
[[21  0  0  0  0  0  0  0  2  0]
 [ 0 20  0  0  0  0  0  0  0  0]
 [ 0  0 21 16  0 20  0  0 10  0]
 [ 0  0  0  5  0  0  0  0  0  0]
 [ 0  1  0  0 20  0  0  0  2  0]
 [ 0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0 21  0  2  0]
 [ 0  0  0  0  0  0  0 21  0  0]
 [ 0  0  0  0  0  0  0  0  5  0]
 [ 0  0  0  0  1  0  0  0  0 21]]
overall accuracy is:  0.741626794258
```

Figure 17: Confusion Matrix for Direction

As we can see in Figure 17, while using direction feature, the mistaken classification mostly happens to number 5, 3 and 8, recognized as number 2 mistakenly.

```
features_density
right: 184  wrong: 25
confusion matrix is:
[ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9.]
[[21  0  0  0  0  0  0  0  0  0]
 [ 0 21  0  0  0  0  0  0  0  0]
 [ 0  0 17  0  0  0  0  0  0  0]
 [ 0  0  0 21  0  0  0  0  0  0]
 [ 0  0  0  0 21  0  0  0  0  0]
 [ 0  0  4  0  0 19  0  0  8  0]
 [ 0  0  0  0  0  1 21  0  1  0]
 [ 0  0  0  0  0  0  0 21  0  8]
 [ 0  0  0  0  0  0  0  0 12  3]
 [ 0  0  0  0  0  0  0  0  0 10]]
overall accuracy is:  0.88038277512
```

Figure 18: Confusion Matrix for Density

As we can see in Figure 18, while using density feature, the mistaken classification mostly happens to number 9, recognized as number 7 mistakenly. The number 8 mistakenly recognized as 5 mostly.

```
features_all
right: 203 wrong: 6
confusion matrix is:
[ 0. 1. 2. 3. 4. 5. 6. 7. 8. 9.]
[[21 0 0 0 0 0 1 0 0 0]
 [ 0 21 0 0 0 0 0 0 0 0]
 [ 0 0 20 0 0 0 0 0 0 0]
 [ 0 0 0 21 0 1 0 0 0 0]
 [ 0 0 0 0 21 0 0 0 0 0]
 [ 0 0 1 0 0 19 2 0 0 0]
 [ 0 0 0 0 0 0 18 0 0 0]
 [ 0 0 0 0 0 0 0 21 0 0]
 [ 0 0 0 0 0 0 0 0 21 1]
 [ 0 0 0 0 0 0 0 0 0 20]]
overall accuracy is: 0.971291866029
```

Figure 19: Confusion Matrix for All Features

As we can see in Figure 19, while using all features, the mistaken classification almost not happening.

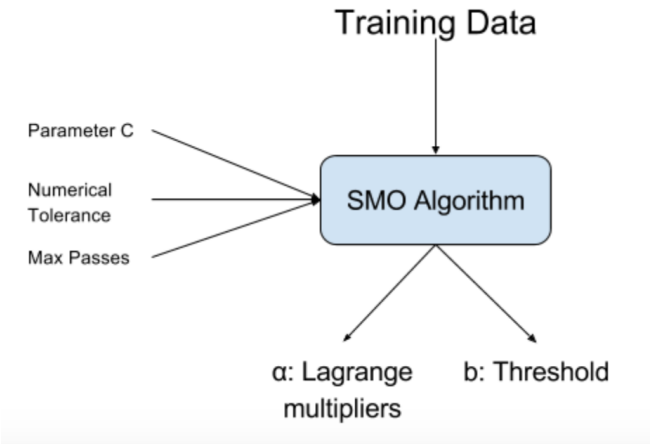


Figure 21: How Training Data was Used.

As you can see in the figure 20, the number of the vectors is not consistent, this is due to when we generate the single features, we add the the ratio of the width and length of the character as one dimension of its feature,so for the density method,its density features is 9 dimension and plus the ration will be 10. For addition when implement the combine features, we didnt delete the replicate ration feature for each of the way, this is why the number of the dimension for different kind of method is not consistent.

8. Results

From our results in Figure 22, when we ran the UCI's dataset Affinity Propagation ran quickly at .0027 seconds with a 94.71 perfect accuracy and when ran with our dataset it ran at .00104 seconds with a 92.34 percent accuracy. For SVM, when we ran it with the UCI dataset, it ran slowly at .593 seconds but with a 96.49 percent accuracy and when we ran it with our dataset it ran at .17402 seconds with a 97.12 percent accuracy. Thus outperforming Affinity Propagation accuracy wise, but not time wise. Though for our hybrid when we ran it with UCI's dataset it ran quickly at .0073 seconds with a 95.1 percent accuracy and when ran with our own dataset it ran at a speedy pace of .00124 with a 95.69 percent accuracy.

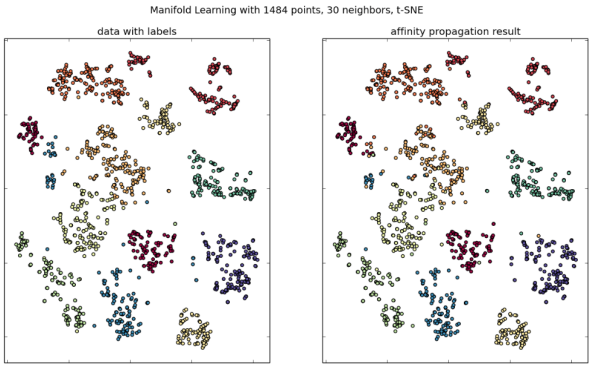


Figure 23: Data with Labels vs. Affinity Propagation Clustering

9. Conclusion

For the feature extraction, the conclusion could be derived from the figure7, figure16 that feature for profiling takes the most important role in all the features combined. This is because the dimension of the profiling feature is larger compared with dimensions of other features and also the profiling feature almost captures the whole contour of the character. thus any feature combine with profiling feature can generate a satisfying result. However, the feature for profiling will be too large without reduction, so we need to reduce it by getting the mean value of 5 values in a column when we get the profiling feature matrix, in this way, some of the key contour information will lose, for example like number 5 and 6, the only difference for them is their left down side, but with the mean reduction way, we may have the risk losing the information of that part. In this case, density features comes in play, with the help of the density and direction features, the part which its contour information is losing can be complemented by the density of that area so with the help of the density, this problem can be partly amended. Although the performance of the direction features won?t work well but add it in the combined features, we could still see some performance improvement in figure 16.

profiling	409	93%	1 mistaken as 9
density	10	88%	8 mistaken as 5 9 mistaken as 7
direction	73	75%	3 mistaken as 2 6 mistaken as 3
profiling+density+direction	492	97%	very few mistakes
profiling+density	419	94%	7 is mistaken to others
profiling+direction	482	90%	5 mistaken as 2
density+direction	83	78%	6 mistaken as 5

Figure 20: Accuracy of Features

		Affinity propagation	SVM	Hybrid
UCI	Predict time (s)	0.0027	0.593	0.0073
	Accuracy	94.71%	96.49%	95.1%
Our data set	Predict time (s)	0.00104	0.17402	0.00124
	Accuracy	92.34%	97.12%	95.69%

Figure 22: Times and Accuracy Percentages of Algorithms

Normally in the hybrid algorithms, at first we run a fast algorithms, if the confidence of its classification is high, we could directly using its result, but if its confidence is low, we will need to run through the more accurate and time consuming algorithms. In our case, the situation is exactly like what was described in the last paragraph. From our experiments and the results we have gathered, shown in Figure 22, shows that our hybrid algorithm out- performs affinity propagations accuracy but is just barely accurate than SVM. Our hybrid algorithm is still just as fast as affinity propagation and much faster than SVM. This gives us the advantage of both of algorithms, which means a quick run-time and a highly accurate recognition.

10. Future Work

In the future, we want to expand the training data to include alphabet rather than just numbers,in this case, we could apply Hidden Markov Model (HMM) to our system, and begin with the word recognition and prediction,for example, with the alphabet, some characters will be very hard to differential using the contour characteristics, so the HMM could using the probability transition matrix to predict and improve the accuracy. HMMs probabilistic models offer many desirable properties for modeling characters or words. [15]

Also, the application could be used in different ways. We could add web server to the website to let users try recognizing their hand writing using websites. The algorithm could also be used on mobile devices to recognize the signatures of different users.

References

[1] M. Blumenstein, X. Y. Liu, and B. Verma. A modified direction feature for cursive character recognition. 2004.

[2] F. Camastra, M. Spinetti, and A. Vinciarelli. Offline cursive character challenge: a new benchmark for machine learning and pattern recognition algorithms. 2005.

[3] F. Camastra and A. Vinciarelli. Cursive character recognition by learning vector quantization. *Pattern Recognition Letters*, 2000.

[4] D. Dileep. A feature extraction technique based on character geometry for character recognition. 2004.

[5] C.-W. Hsu and C.-J. Lin. A comparison of methods for multiclass support vector machines. *IEEE Transactions on Neural Networks*, 2002.

[6] B. Jiang, X. Zhang, and T. Cai. Estimating the confidence interval for prediction errors of support vector machine classifiers. *Journal of Machine Learning Research*, 2008.

[7] R. K. Nath and M. Rastogi. Improving various off-line techniques used for handwritten character recognition: a review. *International Journal of Computer Applications*, 2012.

[8] J. Platt. Fast training of support vector machines using sequential minimal optimization. *MIT Press*, 1998.

[9] J. C. Platt, N. Cristianini, and J. Shawe-Taylor. Large margin dags for multiclass classification. *MIT Press*, 2000.

[10] C. Zanchettin, B. L. D. Bezerra, and W. W. Azevedo. A knn-svm hybrid model for cursive handwriting recognition. *International Joint Conference on Neural Networks*, 2012.

[11] O. D. Trier, A. K. Jain, and T. Taxt. Feature extraction methods for character recognition - a survey. 1995.

[12] B. J. Frey and D. Dueck. Clustering by passing messages between data points. 2007.//

[13] E. Alpaydin and C. Kaynak. Optical recognition of handwritten digits data set. Available from UCI https://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits.

[14] S. A. Yadav, S. Sharma, and S. R. Kumar. A robust approach for offline english character recognition. *International conference on futuristic trend in computational analysis and knowledge management*, 2015.

[15] B. S. Saritha and S. Hemanth. An efficient hidden markov model for offline handwritten numeral recognition. 2005.