Experiment with Methods for Handwritten Letter Recognition

Hao Wang
The Johns Hopkins University
hwang116@jhu.edu

Ke Li
The Johns Hopkins University
keli.jhu@gmail.com

Abstract

This report presents our experiments with some commonly used techniques in OCR task such as kNN, MLP, SVM with some variation. We used extracted features rather than raw pixels which turned out to make significant difference performance wise. SVM is the major classification method we used and it is quite effective in achieving high recognition accuracy.

1 Introduction

The Optical Character Recognition (OCR) is a field that has been extensively researched and yet remains challenging. There are some tidy databases of either handwritten digits or letters that provide students or Machine Learning beginners with good hands-on experience with various classification algorithms. We found some literature, such as (Liu et al., 2003), that presented the introduction and comparison of different techniques with digit recognition which is quite similar to the task of letter recognition. Thus we applied some of these techniques introduced to the Stanford OCR dataset which is a "cleaned" subset of the handwritten words dataset initially collected by Rob Kassel at MIT Spoken Language Systems Group.

2 Feature Extraction

The first stage of the project is to pre-process the images we have. The original size of image is 16×8 and we map the pixels to a new plane with a aspect ratio (ratio of the short dimension to the long one) of r' such that

$$r' = \sqrt{\sin(\frac{\pi}{2}r)}$$

where r' and r are the aspect ratio of the new plane and the original one. If the new plane is not square, we simply match the long dimension of the mapped image to the targeted dimension of a square plane, center the image and pad the rest with zero. In our case, the new normalized images are of 25×25.

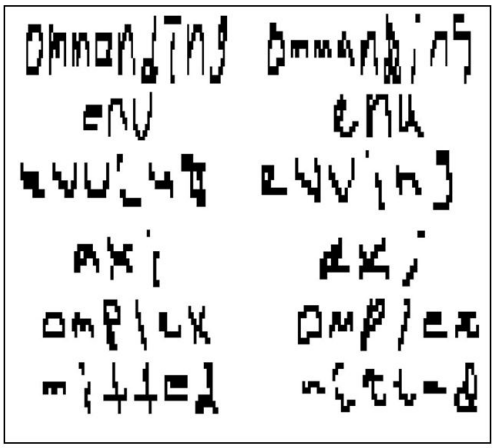

Figure 1: sample images of the data

Then we applied Sobel masks that are usually for edge detection to each pixel and stored the direction and magnitude of its gradient. Next, we decomposed the original image into 8 subimages on which each pixel has its gradient pointing at a particular orientation whose value is the magnitude of the gradient. If the gradient doesn't align with any of the eight

directions, we decompose the gradient into two subgradients in its neighboring directions and assign the magnitude of the two subgradients to the pixel on the corresponding two subimages.

$$G_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * \begin{bmatrix} A_{i-1,i-1} & A_{i,i-1} & A_{i+1,i-1} \\ A_{i-1,i} & A_{i,i} & A_{i+1,i} \\ A_{i-1,i+1} & A_{i,i+1} & A_{i+1,i+1} \end{bmatrix}$$

$$G_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * \begin{bmatrix} A_{i-1,i-1} & A_{i,i-1} & A_{i+1,i-1} \\ A_{i-1,i} & A_{i,i} & A_{i+1,i} \\ A_{i-1,i+1} & A_{i,i+1} & A_{i+1,i+1} \end{bmatrix}$$

$$G = \sqrt{G_x^2 + G_y^2}$$

Figure 2: Sobel gradient

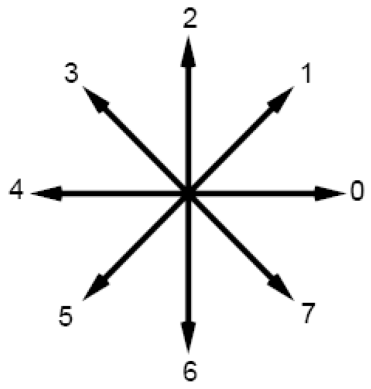

Figure 3: Eight directions of gradients

The last step is to apply Gaussian mask to smooth each subimage so that we could sample pixels from these subimages as our features without losing too much information. We sampled 25 pixels uniformly from each subimage and used their numerical values as our features.

Gaussian mask:

$$h(x, y) = \frac{1}{2\pi\sigma_x^2} exp(-\frac{x^2 + y^2}{2\sigma_x^2})$$

Smoothing convolution:

$$F(x_0, y_0) = \sum_x \sum_y f(x, y)h(x - x_0, y - y_0)$$

3 Classification Methods

3.1 kNN

kNN is a very intuitive classifier and requires no training. We have 200 features which is more than the recommended feature space in class. Nonetheless we achieved around an accuracy of 89%. We tried $k = 1, k = 3, ..., k = 141$, $k = 5$ gives the best result. Notably with the raw pixels the result of kNN is only of 82%. kNN is usually used as benchmark for performance evaluation.

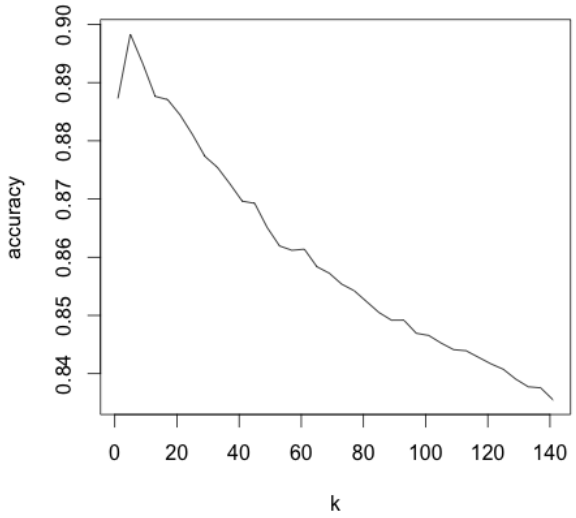

Figure 4: Tuning k for kNN

3.2 MLP

MLP is a black box with numerous parameters to tune other than the basic ones such as the number of nodes and hidden layers. In (Liu et al., 2003) the author mentioned the use of MLP with one hidden layer. We tuned different combinations of the number of nodes and learning iterations and achieved satisfiying result with 600 nodes and 1000 iterations.

3.3 SVM

SVM is our major classifier for this project as we tried to incorporate other classifiers into it. For the basic SVM, we used one against one approach when dealing with the multi-class dataset as in our case.

Our dataset is very imbalanced, for example, there are only 189 "j" in our 52152 sample points. Thus the one against one scheme could better handle the imbalanced classes. Since the number of instances is much greater than our features (52152 vs. 200) we only used radial kernel thus making tuning C and γ important. We implemented two stages of grid search to lock the approximate optimal region first and then finely tune parameters within it.

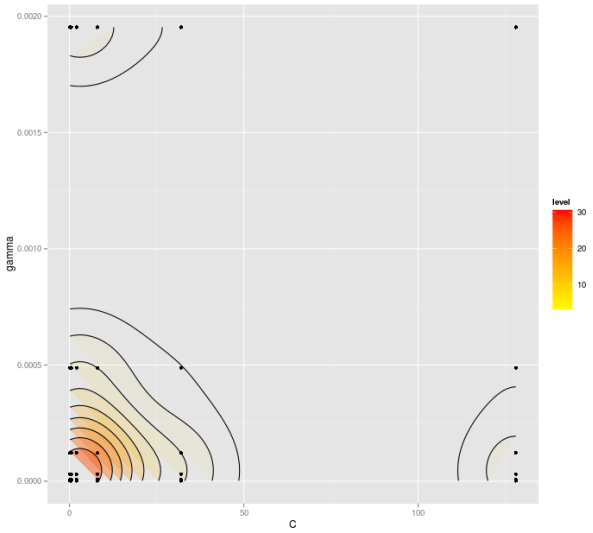

Figure 5: loose grid search for C, γ

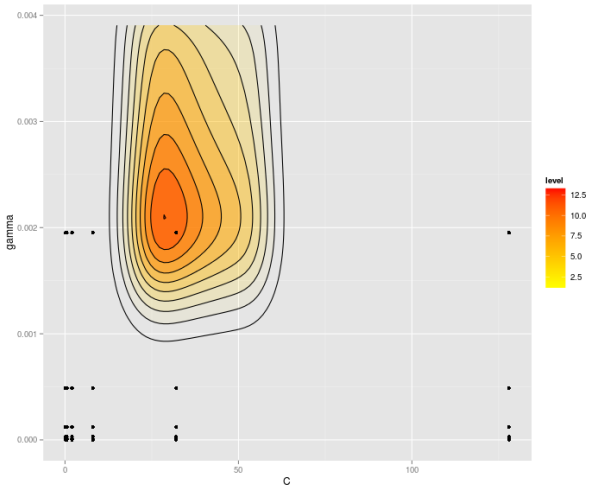

Figure 6: Fine grid search for C, γ

3.4 Local SVM

Since SVM performs well on our dataset, we are set to explore ways to enhance it even more. As mentioned in the class, we tried to perform SVM in a localized setting in hope that there might be a suitable size of neighborhood that is more separable than the whole. But we actually found that the accuracy dropped drastically as k increases and the best k for the local SVM is around 5 to 25. Within such clusters many or even all of the sample points have the same label thus the classification is not too much different from kNN.

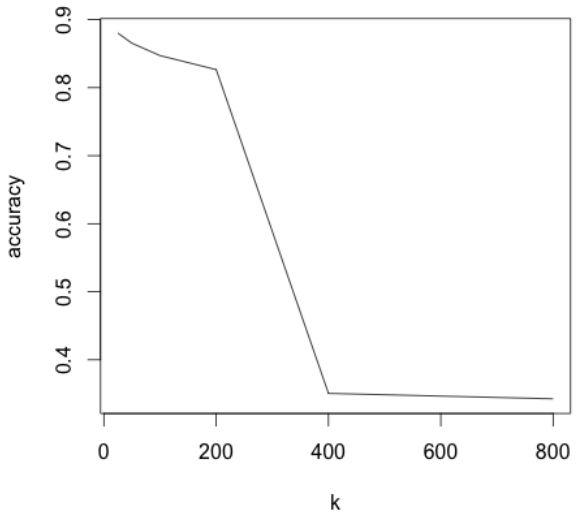

Figure 7: optimal k search for local SVM

3.5 kNN + Local SVM

Yet another attempt to tweak around in a local setting. The intuition is if in a cluster there's no a dominant class, SVM might be helpful in determining a separating boundary and possibly solve the ambiguity. But since local SVM works very bad in a cluster of a couple of hundred sample points, we again have to limit the k to a relatively small value. We tune p which is the threshold on the share in a cluster the most dominant class takes. If the share is lower than the threshold, the local SVM kicks in. The result is thus not much different from local SVM above, which is foreseeable given the small k we have to live by. Even though the line is steep in Figure 8,

it's merely due to the y scale. The outcome doesn't really change, again because of the small value of k.

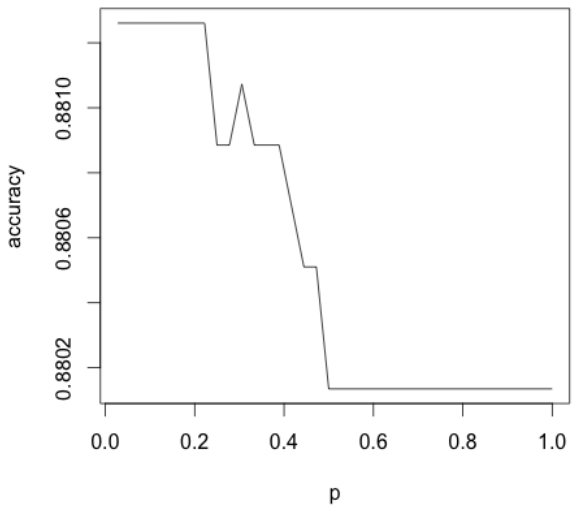

Figure 8: Tuning p for kNN + SVM

3.6 Simple Mixture of Experts

We tried out a very simple and naive scheme of mixture of experts. We use kNN, SVM and MLP to predict and we pick the majority class. If they predict all three different classes, we'll take SVM since it's the best classifier we have. The result is not significantly different from others which could indicate all classifiers failed at some same points that might have quite noisy features.

4 Results

The results on the test set are listed in Table 1. The accuracies are much higher compared to some other people who performed recognition task on the same data, such as (Krevat and Cuzzillo) and (Velagapudi) who used Naive Bayes as the major classifier while the assumption of conditionally independent in Naive Bayes doesn't seem appropriate in an imaging task like this.

Classifiers	**Accuracy**
kNN (k = 5)	89.12%
MLP (1 hidden layer, 600 nodes, 1000 runs)	91.73%
MLP (3 hidden layers, 150 nodes, 1000 runs)	76.25%
SVM (C = 22, γ = 0.004)	92.63%
Local SVM (k = 5, C = 2.8, γ = 0.002)	89.09%
kNN + Local SVM ((k = 5, C = 2.8, γ = 0.002))	88.61%
Mixture of Experts (kNN + SVM + MLP)	92.39%

Table 1: Classification results

5 Additional Methods

5.1 Adaboost + SVM

Adaboost is a promising way to improve our accuracy, and as mentioned in the slides, we should boost our strongest classifier. We tried to incorporate Adaboost into SVM with the final prediction being:

$$H(x) = \arg \max_{y \in Y_t : h_t(x) = y} \sum \alpha_t$$

But somehow the overfitting became a problem. As suggested in (Garcia and Lozano), SVM might be too strong for Adaboost. They proposed a method to leave out a proportion μ of correctly predicted points so as to build a weakened SVM classifier for SVM.

5.2 HMM

HMM is a feasible method for solving the ambiguity among some really similar characters such as "i" and "l" if we can apply word-level constraints using the Hidden Markov Model to model the sequence of letters in a word whose MLE positions can be obtained through the Viterbi algorithm. Thus we can predict some ambiguous cases with greater confidence given its neighboring letters. One thing to note is the dataset doesn't contain a substantial amount of unique words thus makes the HMM more accurate. If we are faced with some dataset with large amount of unique words, HMM won't be as much as efficient in improving the accuracy as in (Velagapudi).

6 Conclusion

Compared with others who worked on the same dataset, our accuracies are very decent. This shows the importance of a set of features that is representative. The SVM classifier is the best performing method in (Liu et al., 2003) in which the authors

compared performances across a dozen of classifiers as well as features. Our results showed that SVM is indeed a good classifier for OCR task. In the future, we'll try the weakened SVM for Adaboost to test how much improvement it can bring. For this project, one big challenge is the size of data. Since the number of instances is large, at first the computation couldn't go through simply because the size of the kernel matrix exceeded the memory. However, more data could help better approximate a workable model. The experience with a large dataset verified the importance of efficient parallel computing with CPU, GPU or even cloud cluster.

References

Chenglin Liu, et al., 2003. *Handwritten digit recognition: benchmarking of state-of-the-art techniques*. *Pattern Recognition*

Prasanna Velagapudi, *Using HMMs to boost accuracy in optical character recognition*.

Elie Krevat, Elliot Cuzzillo, *Improving Off-line Handwritten Character Recognition with Hidden Markov Models*.

Yoav Freud, Rob Schapire, *A tutorial on Boosting*.

Elkin Garcia, Fernando Lozano, *Boosting Support Vector Machines*.