# Handwritten English Alphabet Recognition Using Bigram Cost

Chengshu (Eric) Li
chengshu@stanford.edu
Fall 2015, CS229, Stanford University

**Abstract:** This paper describes a new approach to handwritten English alphabet recognition, namely using bigram cost between English characters to improve performance. 19240 images (370 for each of the 52 uppercase and lowercase English letters) are obtained from NIST database 19 and are preprocessed to be fed into models like softmax classification, Naïve Bayes, Support Vector Machine and feed forward neural network. By using bigram cost, the performance of SVM and feed forward neural network is improved by about three percent.

**Keywords:** Handwritten character recognition, Image processing, Feature extraction, feed forward neural networks, Support Vector Machine, Naïve Bayes, Bigram cost, Search Problem

## 1 Introduction

Optical character recognition (OCR) is one of the most fascinating and successful application of automatic pattern recognition. In the past a few decades, OCR has been an active field for research in Artificial Intelligence. OCR also has a wide range of real-world application such as business card information extraction, book scanning, assistive technology for blind and the automatic processing of invoices, receipts and legal documents

My motivation for this project is based on the intuition that when we as humans try to recognize a character in a word, we also use the context information to help us do that. In fig. 1, for example, we can see the letter 'r' looks more like an 'i' rather than an 'r'. However, we as humans will know almost for sure it is a 'r' based on the letters around it. Therefore, I try to experiment combining normal image processing algorithm and bigram cost between English characters


Figure 1. an example of using context Information to recognize a character

to see whether this synergy could the test performance.

The input for my algorithm is 128 by 128 pixels image that contains one English uppercase or lowercase letter. I then used softmax classification, SVM, Naïve Bayes, neural network to output a predicted letter that this image represents.

The baseline that I used to evaluate my result is to run SVM on raw pixels for the 75% of my images and test on the other 25%, which gave an error rate of 95.23%, only a little bit better than a random guess. The oracle I am aiming for is human recognition. My friends and I manually attempt to recognize 1,000 characters and got an error rate of 0.5%.

I share this project with CS221. Most of my work could be shared between both CS229 and C221 such as data gathering and pre-processing, feature extractions, experimentation with different models and algorithms, test results and error analysis. However, I got my inspiration for this project from the CS221 assignment 'reconstruct' where we used bigram cost to solve vowel insertion problem. I modified and adopted some of the code from that assignment and

and run my test based on a state-based search model.

## 2 Related Work

In the past decades a large number of volumes have been dedicated to character recognition. Rachana Herekar and Prof. S.R. Dhotre[1] proposed a novel way of feature extraction using zoning (which I have tried myself, see details below) and achieved an error rate about 88% - 92%. J.Pradeep, E.Srinivasan and S.Himavathi[2], on the other hand, used vertical, horizontal, and diagonal method for their feature extraction. Moreover, Yusuf, Perwej and Ashish Chaturvedi[3] extracted their features by putting a 25 segment grid on top of the image that produced a feature factor of dimension 25 (which I have also tried myself). All three papers use different feature extraction methodology but all use feed forward neural network to train their models, which presumably gave them the best result during their experimentation .

On the other hand, V.K. Govidan and A.P. Shivaprasad's paper[4] offers a more qualitative and general review of character recognition, such as some of the recurring challenges in the field and the previous attempts to overcome them. Lastly, Charles C. Tappert and Sung-Hyuk Cha[5] touched upon online handwriting recognition, which is not necessarily my research topic, but they raised several interesting insights in how people put down strokes. All of the papers above has been extremely helpful for me to understand previous attempts and come up with reasonable feature extraction methods.

## 3 Data Preprocessing

My dataset comes from NIST Database 19[6] and consists of 19240 samples, that is 370 128 by 128 pixels images for each of the 52 uppercase and lowercase English alphabet.

I preprocessed the image by first cropping out the central part which actually contains the character and then resizing it back to 128 by 128 pixels. Fig. 2 below illustrates the preprocessing process for a sample 'H'.

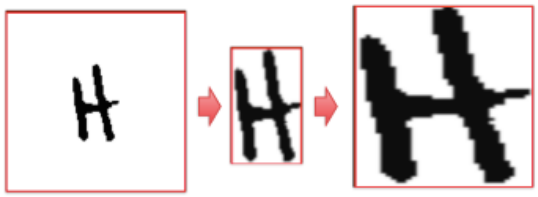


Figure 2. Preprocessing a sample 'H'

## 4 Feature Extraction

Five different feature extraction methods have been experimented.

i) Raw Pixels: use raw pixels from the preprocessed image（128 by 128 pixels). Each pixel is either black (0, 0, 0) or white (255, 255, 255). The feature vector is of dim. 16384

ii) Grid Threshold Method[3]: put a grid of 16 by 16 onto the image. For each region, loop through all the pixels and check whether more than THRESHOLD percentage of them are black pixels. If so, put a 1 onto that region. Otherwise, put a zero. Please see figure 3 for an illustration of a grid of 5 by 5. The feature vector is of dim. 256 for a grid of 16 by 16.

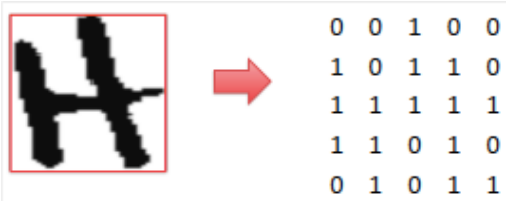



Figure 3. Grid Threshold Method

iii) Grid Percentage Method, similar to ii). Instead of setting a threshold, directly use the percentage of black pixels as a field in the feature vector. The feature vector is of dim. 256

iv) Zoning[1]: put a grid of 3 by 3 onto the image, and classify each zone to one of the

six different types.

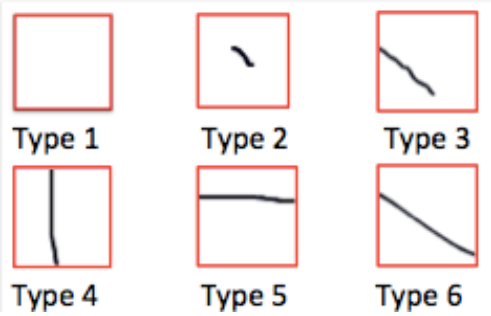



Figure 4. six types for each zone

v) Loop finding: use breadth first search to figure out whether there is a loop in the image. For example, if the image represents 'b', this feature should be 1. If the image represents 'm', this feature should be 0. The algorithm itself has an error rate of about 0.2

After testing all these features using SVM with 3,900 training examples and 1,300 test examples, a combination of i) and v) are chosen to be my final feature choices as they perform the best.

## 4 Methods and Algorithms

Four different learning algorithms have been experimented

a) Self-implemented Softmax Classification
I implemented from scratch a softmax classification algorithm based on the last section of Lecture Note 1.

$$\begin{aligned} p(y=i|x;\theta) &= \phi_i \\ &= \frac{e^{\eta_i}}{\sum_{j=1}^{k} e^{\eta_j}} \\ &= \frac{e^{\theta_i^T x}}{\sum_{j=1}^{k} e^{\theta_j^T x}} \end{aligned}$$

Figure 5. mathematical notation for softmax

y can anything from 0 to 51, which represents 'a', 'b', 'c', … 'z', 'A', 'B', … 'Z'. X is the feature vector of dimension 16384 (raw pixels) + 1 (loop detection) = 16385. During training, I tried to minimize the log likelihood function shown below by taking the gradient with respect to each $\theta_i$ and using stochastic gradient assent to update each $\theta_i$.

$$\begin{aligned} \ell(\theta) &= \sum_{i=1}^{m} \log p(y^{(i)}|x^{(i)};\theta) \\ &= \sum_{i=1}^{m} \log \prod_{l=1}^{k} \left( \frac{e^{\theta_l^T x^{(i)}}}{\sum_{j=1}^{k} e^{\theta_j^T x^{(i)}}} \right)^{1\{y^{(i)}=l\}} \end{aligned}$$

Figure 6. log likelihood function for softmax

The algorithm ran very slowly and gave an error rate of over 50%, so I eventually gave it up.

b) Support Vector Machine

I downloaded SVC package from scikit-learn library and used it for multi-class classification. SVM is originally designed to do binary classification, i.e. drawing a line to separate the data into two classes. Scikit-learn implement SVC in the "one-against-one" approach. If n is the number of classes, then n*(n-1) / 2 classifiers are constructed and each one of them categorizes training data into two classes.[7] The result is then aggregated.

c) Naïve Bayes

I downloaded Naïve Bayes package from scikit-learn library and used MultinomialNB for multi-class classification. For each epoch the algorithm is updating where represents the $\theta_y = (\theta_{y1}, \ldots, \theta_{yn})$ of feature $\theta_{yi}$ appearing in a sample belonging to class y.[8] By default it also uses Laplace smoothing of alpha = 1

d) Feed−Forward Neural Networks

I downloaded PyBrain's library and tried one hidden layer of 10, 20, 100, 500 neurons

# 5 Advanced Method (for CS221)

Inspired by the CS221 assignment 'reconstruct', in which we used bigram cost to solve vowel insertion problem based on a search problem model, I decided to test the results of my character recognition in the context of a English word.

First of all, I collected 1000 most common English words[9], and then randomly sampled letters from my hand-printed data to form those words. These become my test sets.

Secondly, I generated a probability table for English characters by using english_bigram_1.txt found on Github. [10] Each line is the frequency of that two-letter combination. Apparently, 'TH' is the most frequent combination thanks to the word 'the' and 'HE' is the second thanks to the word 'he' and 'she'. Based on these frequencies, a probability table is generated, where table['a']['b'] is the probability that 'b' comes after 'a' as opposed to other letters that come after 'a'

```
TH 116997844
HE 100689263
IN 87674002
ER 77134382
AN 69775179
RE 60923600
ES 57070453
ON 56915252
.....
```

Figure 7. english_bigram_1.txt

The last part is putting everything together into a state-based search problem. Each state encapsulates the current feature vectors and the previous character chosen. In each state, predict the current character using already trained SVM/Naïve Bayes/Neural Network and pick the top five predictions to be the next possible actions. For each of the actions, the cost is (-1) times the following

$\alpha \times$ bigramCost(prevChar, curChar) + $(1-\alpha) \times P(y = curChar \mid X)$

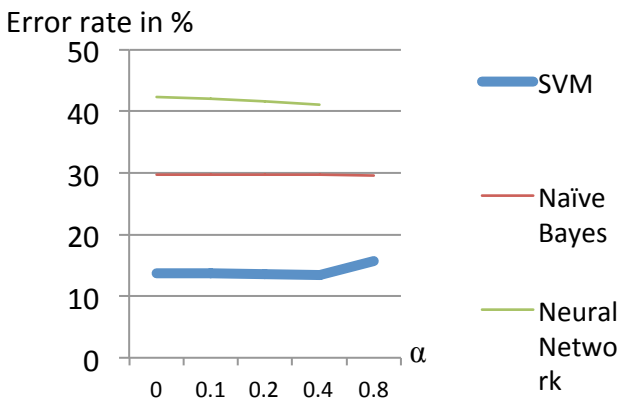
The first term measures how fluent it is to have curChar coming after prevChar. The second term measures how likely the image represents currentChar solely based on its physical form. By changing $\alpha$ and weighing these two terms differently, the purpose is to investigate whether adding bigram cost will improve the performance of character recognition in a more realistic context where the model will classify the test image not only based on how it looks, but also the previous character and the bigram cost between them.

# 6 Experiments and Results

SVM and Naïve Bayes are trained on all 19240 examples whereas due to the computational limit, I could only train my self-implemented softmax classification algorithm on 2600 examples and feed-forward neural network on 5200 examples within reasonable amount of time. So when reading this paper please bear in mind that the result may favor SVM and Naïve Bayes to the other two.

The x-axis is $\alpha$ and the y-axis is the error rate. $\alpha = 0$ means the model is purely using the image's physical form to classify image, whereas $\alpha = 0.8$ means the model is weighing bigram cost from the previous character higher than how the image actually looks like. Lastly, when calculating error rate, the uppercase and lowercase letters of the following are treated as the same category:

['c', 'i', 'j', 'k', 'o', 'p', 's', 'u', 'v', 'w', 'x', 'y', 'z']

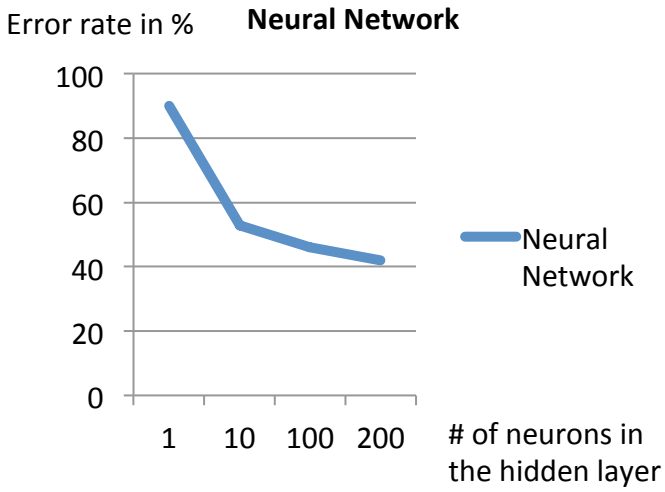


Naïve Bayes is hardly affected by the bigram cost at all. SVM performs the best when $\alpha = 0.4$, with error rate 13.38%. When $\alpha = 0$, meaning bigram cost is not used, the error rate is 13.78%. Similar trend happens for neural network as well. Therefore, bigram cost did help drive down the error rate

Since the confusion table is too large to fit into this page to be legible, I will include a table in which the rows represent all the letters and the columns represent the top five letters that this letter has been classified to.

| letter | 1st | 2nd | 3rd | 4th | 5th | success rate |
|---|---|---|---|---|---|---|
| a | a:0.921 | g:0.032 | U:0.024 | O:0.013 | e:0.008 | 0.92 |
| b | b:0.965 | N:0.022 | l:0.011 | m:0.0 | w:0.0 | 0.96 |
| c | C:0.675 | e:0.177 | t:0.060 | O:0.035 | L:0.030 | 0.67 |
| d | d:1.0 | Z:0.0 | n:0.0 | x:0.0 | w:0.0 | 1 |
| e | e:0.979 | C:0.020 | Z:0.0 | n:0.0 | w:0.0 | 0.97 |
| f | F:0.625 | t:0.147 | a:0.045 | l:0.045 | Q:0.034 | 0.02 |
| g | g:0.720 | S:0.076 | a:0.033 | q:0.025 | B:0.025 | 0.72 |
| h | h:0.944 | A:0.022 | n:0.016 | b:0.011 | L:0.005 | 0.944 |
| i | I:0.417 | i:0.189 | l:0.085 | L:0.053 | t:0.05 | 0.606 |
| j | j:0.625 | l:0.125 | S:0.125 | i:0.125 | n:0.0 | 0.625 |
| k | k:0.274 | R:0.137 | K:0.137 | L:0.078 | h:0.078 | 0.41 |
| l | l:0.481 | I:0.444 | Y:0.037 | L:0.033 | q:0.003 | 0.48 |
| m | M:0.833 | n:0.063 | h:0.031 | y:0.015 | t:0.015 | 0.01 |
| n | n:0.973 | N:0.016 | H:0.006 | A:0.003 | Z:0.0 | 0.973 |
| o | O:0.960 | n:0.022 | F:0.017 | Z:0.0 | m:0.0 | 0.96 |
| p | P:0.804 | r:0.083 | F:0.034 | p:0.034 | D:0.020 | 0.84 |
| q | q:0.454 | Y:0.090 | t:0.090 | S:0.090 | g:0.090 | 0.454 |
| r | r:0.991 | P:0.008 | Z:0.0 | m:0.0 | w:0.0 | 0.99 |
| s | S:0.882 | J:0.027 | T:0.027 | D:0.020 | A:0.017 | 0.882 |
| t | t:0.971 | T:0.018 | l:0.005 | d:0.005 | m:0.0 | 0.971 |
| u | U:0.919 | V:0.033 | O:0.020 | h:0.013 | n:0.013 | 0.919 |
| v | V:0.615 | v:0.134 | U:0.096 | r:0.038 | N:0.038 | 0.75 |
| w | W:0.734 | N:0.102 | U:0.061 | w:0.051 | A:0.030 | 0.78 |
| x | x:0.642 | l:0.142 | V:0.071 | t:0.071 | Y:0.071 | 0.64 |
| y | Y:0.456 | y:0.135 | V:0.123 | l:0.074 | g:0.037 | 0.59 |
| z | Z:1.0 | Y:0.0 | x:0.0 | w:0.0 | v:0.0 | 1 |
| A | A:0.918 | n:0.029 | t:0.027 | a:0.024 | d:0.0 | 0.92 |
| B | B:0.804 | a:0.103 | t:0.057 | R:0.011 | q:0.011 | 0.804 |
| C | C:0.964 | U:0.025 | O:0.010 | Z:0.0 | m:0.0 | 0.964 |
| D | D:0.781 | O:0.103 | b:0.045 | t:0.022 | U:0.017 | 0.781 |
| E | E:0.970 | R:0.015 | t:0.014 | Z:0.0 | m:0.0 | 0.97 |
| F | F:0.806 | P:0.102 | n:0.056 | t:0.022 | r:0.011 | 0.806 |
| G | G:0.754 | a:0.059 | B:0.059 | d:0.025 | R:0.025 | 0.754 |
| H | H:0.779 | t:0.110 | A:0.049 | n:0.022 | h:0.011 | 0.779 |
| I | I:0.907 | l:0.085 | J:0.007 | Z:0.0 | n:0.0 | 0.907 |
| J | J:1.0 | Z:0.0 | y:0.0 | w:0.0 | v:0.0 | 1 |
| K | K:0.529 | k:0.196 | t:0.058 | U:0.039 | N:0.039 | 0.72 |
| L | L:0.859 | U:0.051 | l:0.040 | C:0.022 | e:0.018 | 0.86 |
| M | M:0.992 | N:0.007 | l:0.0 | w:0.0 | v:0.0 | 0.99 |
| N | N:0.892 | n:0.036 | W:0.023 | t:0.016 | H:0.016 | 0.892 |
| O | O:0.980 | I:0.019 | Z:0.0 | m:0.0 | w:0.0 | 0.98 |
| P | P:0.888 | r:0.062 | F:0.020 | l:0.013 | I:0.013 | 0.888 |
| Q | Q:0.454 | q:0.181 | O:0.181 | a:0.090 | G:0.090 | 0.454 |
| R | R:0.913 | P:0.059 | x:0.013 | A:0.005 | t:0.005 | 0.913 |
| S | S:0.979 | L:0.017 | t:0.003 | Z:0.0 | w:0.0 | 0.979 |
| T | T:0.950 | t:0.026 | r:0.013 | Y:0.010 | m:0.0 | 0.95 |
| U | U:0.973 | a:0.020 | N:0.006 | m:0.0 | w:0.0 | 0.97 |
| V | V:0.673 | r:0.173 | U:0.057 | l:0.057 | N:0.019 | 0.673 |
| W | W:0.755 | U:0.102 | N:0.102 | V:0.020 | t:0.020 | 0.755 |
| X | x:0.714 | t:0.142 | X:0.071 | N:0.071 | m:0.0 | 0.79 |
| Y | Y:0.777 | l:0.172 | S:0.024 | r:0.024 | b:0.0 | 0.777 |
| Z | z:1.0 | Y:0.0 | x:0.0 | w:0.0 | v:0.0 | 1 |

Lastly, for feed-forward neural network, I found that the larger the number of neurons there is in the hidden layer, the better performance it can achieve.




## 7 Conclusion and Future Work

Some of the key observations are the following. First of all, the bigram cost method did improve the performance of the model, by around 3%, although the improvement is expected to be larger. Other observation is that some letters are much more difficult to classify than others. The bottleneck letters are highlighted in the table on the left.
Lastly, neural network does not perform as well as expected, maybe because the number of neuron being used is still very small.

Some potential work includes finding better feature extractions method to further improve SVM, Naïve Bayes and Neural Network's performance, reducing error rates for those bottleneck letters, furthering experimenting with neural network by increasing its neuron numbers and hidden layers.

# 7 Reference


(1) Herekar, Rachana R., and Prof. S. R Dhotre. "Handwritten Character Recognition Based on Zoning Using Euler Number for English Alphabets and Numerals." IOSRJCE IOSR Journal of Computer Engineering 16.4 (2014): 75-88. Web.

(2) Pradeep, J., E. Srinivasan, and S. Himavathi. "Diagonal Based Feature Extraction for Handwritten Alphabets Recognition System Using Neural Network." International Journal of Computer Science and Information Technology IJCSIT 3.1 (2011): 27-38. Web.

(3) Perwej, Yusuf, and Ashish Chaturvedi. "Neural Networks for Handwritten English Alphabet Recognition." International Journal of Computer Applications IJCA 20.7 (2011): 1-5. Web.

(4) Mantas, J. "An overview of character recognition methodologies." Pattern recognition 19.6 (1986): 425-430.

(5) Tappert, Charles C., and Sung-Hyuk Cha. "English Language Handwriting Recognition Interfaces." Text Entry Systems (2007): 123-37. Web.

(6) "NIST Special Database 19." NIST Special Database 19. N.p., n.d. Web. 11 Dec. 2015.

(7) "1.4. Support Vector Machines." 1.4. Support Vector Machines — Scikit-learn 0.17 Documentation. N.p., n.d. Web. 11 Dec. 2015.

(8) "Welcome to PyBrain’s Documentation!" Welcome to PyBrain's Documentation! — PyBrain V0.3 Documentation. N.p., n.d. Web. 11 Dec. 2015.

(9) Deekayen. "1,000 Most Common US English Words." Gist. Deekayen, n.d. Web. 11 Dec. 2015.

(10) Padraigmaciain. "Padraigmaciain/textual-analysis-suite." GitHub. Padraigmaciain, n.d. Web. 11 Dec. 2015.