

Handwritten Digits Classification

[Team Y.E.S.: COMP 598 Group Project 3] *

Yuting Wen
McGill University
yuting.wen@mail.mcgill.ca

Emmanuel Bengio
McGill University
emmanuel.bengio@mail.mcgill.ca

Sherry Ruan
McGill University
shanshan.ruan@mail.mcgill.ca

ABSTRACT

In this project, we aim at classify a much more difficult variation of the MNIST dataset of handwritten digits. We adopt feature selection and construction techniques together with four main machine learning algorithms: Gaussian Naive Bayes, fully connected Feedforward Neural Network, Linear Support Vector Machine, and XXX (name of the 4th algorithm here). We analyze and assess the parameter selection process and the performance of each algorithm. We conclude the report with discussion and suggestions for further improvement.

1. INTRODUCTION

The MNIST database of handwritten digits [5] is a standard touchstone of effective image classification algorithms. It is extensively studied and tested by many machine learning techniques [2, 3, 4, 8]. The original dataset consists of more than 60,000 handwritten digits from 0 to 9, normalized to a 28x28 fixed image size [5].

The dataset we are dealing with in this project is more challenging. Modifications of the original dataset include embossing, rotation, rescaling, and texture pattern. These artificial alterations introduce a great amount of noise and undoubtedly increase the level of difficulty of the digit classification task. The modified dataset contains 50,000 training examples of 48x48 fixed size, and the test set comprises 20,000 instances which require classification [7].

We decided to apply four different algorithms: Naive Bayes, Feedforward Neural Networks, Linear Support Vector Machine, and Deep Convolutional Neural Networks to the modified MNIST dataset. For the baseline algorithm, we chose Gaussian Naive Bayes since features given as float numbers are continuous. For Neural Networks,..... For Linear SVM, For ConvNets we applied many layers of convolution with *tanh* units, followed by a single hidden layer and a softmax prediction layer.

The performance of algorithms varies widely. The base line algorithm, Naive Bayes, provides around 40% accuracy, this may due to the fact that the Naive Bayes assumption does not hold in the digit classification task in general. NN..... SVM..... ConvNets seem to be the best suited models for this task, as we achieve around 93.6% validation accuracy,

*The dataset and the implementation of the algorithm described in this report is available at <https://github.com/yutingyw/imageClassification>

and 94.05% accuracy on the public test set.

Our empirical results, though preliminary, provide considerably accurate predictions (especially Convolutional neural networks) for the modified MNIST digit classification. Thus, we are optimistic of applying the algorithms and analysis presented in this report to other real-world classification problems. In particular, this can motivate the further study on more specialized machine learning algorithms on image classification tasks.

2. RELATED WORK

Optional this time. We can write something here if there's any good related work worth discussing.

3. METHODOLOGY

We present detailed descriptions of our methods featuring data preprocessing, feature selection, algorithm selection, and optimization techniques in this section. We provide theoretical characterizations of our approaches and outline the results of these specific methods. We will illustrate the advantage of our methods using informative graphs and analyze the experimental results in next section.

3.1 Data Preprocessing Methods

We adopted different data preprocessing methods based on the characteristic of each machine learning algorithm. Since the dataset was given in a relatively organized format (csv files containing float numbers), we spared little effort to format data or extract numerical data from images. Most data preprocessing methods we used were adapted for a specialized algorithm.

In Naive Bayes, we adopted normalization to make it suitable for the algorithm. We obtained a set of scaled examples of unit norm after the normalization. We chose L2 norm since it resulted in the greatest improvement in terms of accuracy. We will give more details including the graph showing accuracy versus data preprocessing methods in later section (testing and validation).

Data preprocessing in neural networks.

Data preprocessing in linear SVM.

To train convolutional neural nets, we generated new examples online by randomly rotating (between 0 and 2π) the

original images. This forced the network to learn to classify digits independently of rotation, which given our prior knowledge on the task is a reasonable assumption to make.

3.2 Feature Design and Selection

I'm not sure what exactly feature design and selection are. Need more thought on this part.

I think this refers to people that use things like Gabor filter extraction and similar, personally I didn't use anything like that.

3.3 Algorithm Selection

We chose Gaussian Naive Bayes as the baseline algorithm, fully connected feedforward neural networks, linear support vector machine as required algorithms, together with Convolutional neural networks as the fourth optional algorithm. The following is a brief summary of central ideas of each algorithm.

3.3.1 Baseline: Gaussian Naive Bayes

Naive Bayes is one of the simplest machine learning algorithm. The theoretical foundation underlying the algorithm is the Naive Bayes assumption: conditional probabilities are independent of each other [1, 6].

Assume we are provided with n training examples and m features. In discrete case, Bayes rule and Naive Bayes assumption tell us that

$$\begin{aligned} P(Y|X_1 \cdots X_m) &= \frac{P(Y)P(X_1 \cdots X_m|Y)}{P(X_1 \cdots X_m)} && \text{by Bayes rule} \\ &= \frac{P(Y)\prod_{j=1}^m P(X_j|Y)}{P(X_1 \cdots X_m)} && \text{by NB assumption} \end{aligned}$$

Hence given a new instance $(X_1 \cdots X_m) = (x_1 \cdots x_m)$, the predicted label for $(x_1 \cdots x_m)$ is

$$\hat{y} = \arg \max_{y_i} P(Y = y_i) \prod_{j=1}^m P(X_j = x_j | Y = y_i) \quad (1)$$

However, in image classification task, each image is represented by an array of float numbers which can be regarded as real numbers. In order to address the continuous case, we introduce Gaussian Naive Bayes and extend the above formula as follows. We assume $P(X_j = x_j | Y = y_i)$ has a normal (Gaussian) distribution with mean μ_{ij} and variance σ_{ij} . Note that while X_j are continuous random variables which can stand for pixel intensities, Y is a discrete random variable corresponding to labels 1–9. The probability density function for $P(X_j = x_j | Y = y_i)$ is given below:

$$P(X_j = x_j | Y = y_i) = f(x_j, \mu_{ij}, \sigma_{ij}) = \frac{1}{\sigma_{ij}\sqrt{2\pi}} e^{-\frac{(x_j - \mu_{ij})^2}{2(\sigma_{ij})^2}} \quad (2)$$

In order to train Gaussian Naive Bayes, we need to approximate $P(Y = y_{i'})$ as well as $\mu_{i'j'}$ and $\sigma_{i'j'}^2$ for $y_{i'}$ over all labels (0 to 9) and j' ranging from 1 to m (number of features).

$$\hat{\mu}_{i'j'} = \frac{\sum_{i=1}^n x_{ij'} \delta(y_i, y_{i'})}{\sum_{i=1}^n \delta(y_i, y_{i'})} \quad (3)$$

$$\hat{\sigma}_{i'j'}^2 = \frac{\sum_{i=1}^n (x_{ij'} - \hat{\mu}_{i'j'})^2 \delta(y_i, y_{i'})}{\sum_{i=1}^n \delta(y_i, y_{i'})} \quad (4)$$

where δ is the Kronecker's delta. It is equal to 1 if two variables are the same and 0 otherwise. x_{ij} denotes the j th feature in the i th example.

Once we finish estimation of parameters, we use the following equation to predict labels for a given instance $x_1 \cdots x_m$.

$$\hat{y} = \arg \max_{y_i} P(Y = y_i) \prod_{j=1}^m f(x_j, \mu_{ij}, \sigma_{ij}) \quad (5)$$

where f denotes the pdf of the normal distribution.

3.3.2 Neural Net

Write something about Neural Net algorithm

3.3.3 Linear SVM

Write something about Linear SVM algorithm

3.3.4 Convolutional Neural Networks

Convolutional Neural Networks apply the idea of signal convolution to neural networks. The key idea is that groups of neurons, filters, are applied locally on subregions of the images instead of the whole image [4]. These filters are then convolved on input images, thus creating new images, or feature maps.

The original input image is considered as a single feature map, $f(x, y)$. Generally the input to a convolutional layer is a set of n_f feature maps, $f_i(x, y)$, on which the n_j filters of size $(p \times q)$ $g_k(x, y)$ (also usually represented as a 4d-tensor W of weights) are convolved, thus creating n_g new feature maps $h_j(x, y)$:

$$h_j(x, y) = \sum_{u=0}^p \sum_{v=0}^q \sum_{i=0}^{n_f} f_i(x+u, y+v) g_j(u, v)$$

A bias is then added on each individual filter map, which is then bounded by an activation function (here we used the hyperbolic tangent \tanh). We get that the output of a layer in terms of feature maps is

$$H_i(x, y) = \tanh(h_i(x, y) + b_i) = \tanh(W * x + b)$$

Finally, maxpooling is applied to the filter maps, where each $(s \times t)$ blocks of the maps are reduced to their maximum value. For $s = t = 2$, it is essentially scaling the image down by a factor two, but keeping the max instead of taking, say, the average value. This gives the model some level of translation invariance, on top of the ability to detect highly local features.

After having created a feature map representation of the input, we flatten the last feature maps to vector form, and feed it to a two layer feedforward neural network, with a \tanh hidden layer and a softmax output layer. The softmax function is as follows:

$$s(x)_i = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

3.4 Optimization

Some algorithms involve maximization or minimization processes. For example, we need to maximize the $P(Y = y_i) \prod_{j=1}^m f(x_j, \mu_{ij}, \sigma_{ij})$ in Naive Bayes. Since the log function is monotonically increasing, it preserves the maximum.

Hence, we can maximize the log likelihood as shown in equation (6) instead of the original equation (5) :

$$\arg \max_{y_i} \log P(Y = y_i) + \sum_{j=1}^m \log f(x_j, \mu_{ij}, \sigma_{ij}) \quad (6)$$

In Neural Network, In linear SVM, In XXX,

3.4.1 Convolutional Neural Networks

We train our convolutional models similarly to feedforward neural networks, using gradient descent with weight regularization.

What is added, as mentioned in the methodology section, is that we train our models on randomly rotated examples generated online from the training data. On top of that, we use our prior knowledge of the task to perform some kind of bagging. What we are doing is not bagging different models with the same input, but rather “bagging” the same convolutional model with different inputs, where each of these inputs is in fact the original input with a different rotation. This takes advantage of the fact that the original data is created using random rotations of the source images, and recedes the error significantly.

4. TESTING AND VALIDATION

In this section, we present detailed experimental results, most of them in terms of graphs. We also evaluate the performance of four algorithms and provide analysis on merits and defects of each of the four algorithms. Our analysis concentrate on hyper-parameter selection and testing and validation results.

4.1 Parameter Selection

We first embark upon an analysis on the relation between hyper-parameters and algorithm performance.

4.1.1 Baseline: Naive Bayes

Since we chose Gaussian Naive Bayes as the baseline algorithm, we do not necessarily need to include Laplace smoothing. Instead, we show how accuracy varies as we alter the norm parameter in normalization preprocessing process. When we normalized data, we tried both *L1* and *L2* norm. As presented in Figure 1, Gaussian Naive Bayes together with the normalization preprocessing method (with *L2* norm) brings out the most satisfactory results. In particular, we also compare the performance of Gaussian Naive Bayes with other Naive Bayes algorithm (multinomial Naive Bayes) to observe how normalization affects the prediction accuracy. Gaussian Naive Bayes in general outperforms Multinomial Naive Bayes. More interestingly, while Multinomial Naive Bayes performs worse on the normalized data (especially *L1*), Gaussian Naive Bayes produces higher prediction accuracy as we further processed the data (from no normalization to *L1*, and from *L1* to *L2*).

4.1.2 Neural Net

Write something about what are important parameters in NN and how you train them, with graphs showing how accuracy varies as these parameters vary

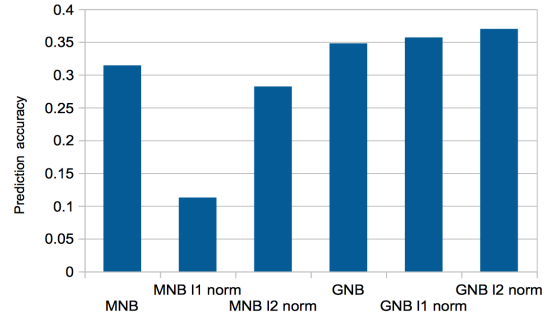


Figure 1: Accuracy versus different Naive Bayes using the original data, *L1*-normalized data, and *L2*-normalized data (train set size = 40,000 and test set size = 10,000)

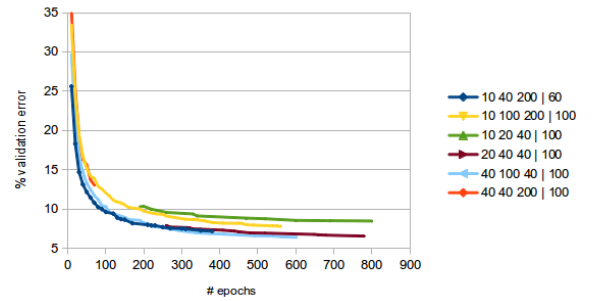


Figure 2: Validation error rate of different ConvNet model sizes during training

4.1.3 Linear SVM

Write something about what are important parameters in SVM and how you train them, with graphs showing how accuracy varies as these parameters vary

4.1.4 Convolutional Neural Networks

Considering the massive amount of time that is required to train ConvNets, 5-fold validation error was not always considered when choosing hyperparameters. Instead, we relied on a 4:1 split of the dataset.

We first tried using two convolutional layers, but quickly realised that using three convolutional layers followed by a two fully connected layers, hidden and output layers, seemed to work best. Similarly we used small filter sizes it gave the best early results.

As such, we did not do proper hyperparameter search, and used our knowledge of the model and of the data along with optimal results for the original MNIST dataset to guide us. We can see in Figure 2 that most models will converge to some low error. Note that we did not manage to train any model where the training error converged to some lower error than the validation error (both are very close, as such they are not shown in the figure), and conclude that we could train much larger models before overfitting occurs. Unfortunately, the required training time for such models prohibits us from trying here.

4.2 Testing Results Analysis

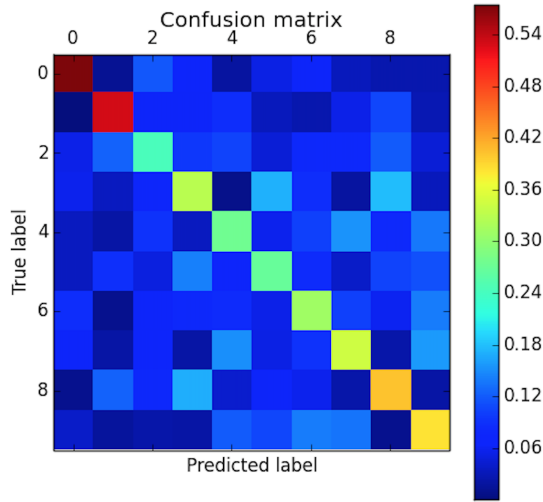


Figure 3: Normalized confusion matrix for Gaussian Naive Bayes with train set size = 40,000 and test set size = 10,000

We provide further analysis on testing and validation results with help of figures and confusion matrices. We still divide the analysis into four parts based on four algorithms.

4.2.1 Baseline: Naive Bayes

We also present the confusion matrix corresponding to the Gaussian Naive Bayes algorithm running with 5-fold cross validation on L_2 normalized data in Figure 3. We normalized the row vectors of the confusion matrix so that we could make fair comparison among different classes. As can be seen from the normalized confusion matrix, the GNB classifier is capable of distinguish 0 and 1 from the others, but it performs relatively poor when classifying 2 to 7. Its comparatively promising performance on classifying 0 may be due to the fact that the digit 0 is least susceptible to all artificial alterations imposed on the original MNIST dataset (especially rotation).

4.2.2 Neural Net

results of NN

4.2.3 Linear SVM

results of SVM

4.2.4 Convolutional Neural Networks

We see in Figure 4 that the ConvNet has good accuracy across classes. As mentioned earlier, better accuracy could probably be reached with larger model, as the ones we trained would not have a much lower training error than validation error (at most 2% lower, often less).

5. DISCUSSION

The artificial alteration imposed on the MNIST handwritten digit dataset brings about a great amount of noise and complicates the classification task. Many typical machine learning algorithms are not suitable for this modified dataset anymore. For example, Naive Bayes can only achieve up to 40% accuracy, even with Gaussian Naive Bayes and many

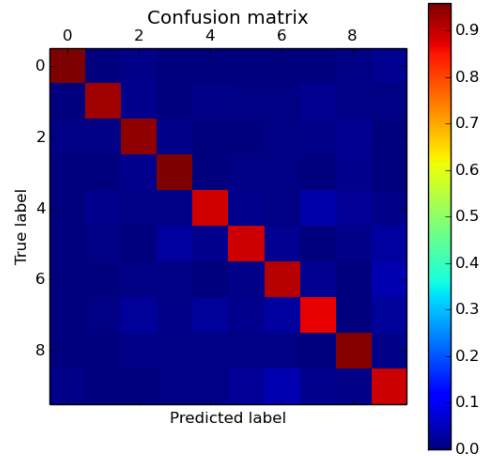


Figure 4: Normalized confusion matrix for Convolutional Neural Nets with train set size = 40,000 and test set size = 10,000

refined preprocessing methods. More elaborate feature selection and preprocessing techniques may be capable of improving the prediction accuracy using Naive Bayes, but it appears to us that it is unfeasible to achieve an accuracy as high (above 80%) as other more sophisticated algorithms in deep learning.

On the other hand, deep learning models, in particular convolution neural networks, achieve much higher accuracy. Convolutional models take full advantage of the graphical nature of the input space and the high correlation between local visual features, and as such are capable of great scores on this particular task, with little need for feature preprocessing.

To summarize, we endeavored to classify the modified handwritten digits using four different algorithms. While some are capable of achieving surprisingly high accuracy, some illustrate the limitation of "shallow" algorithms. After all, machine learning problems can never be unraveled using a same fixed method. Instead, it requires the exploration of versatile tools, and that is where the charm of machine learning lies.

We hereby state that all the work presented in this report is that of the authors

6. REFERENCES

- [1] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.
- [2] Y. Huang, J. Zhao, M. Tian, Q. Zou, and S. Luo. Slow feature discriminant analysis and its application on handwritten digit recognition. In *Proceedings of the 2009 International Joint Conference on Neural Networks, IJCNN'09*, pages 132–135, Piscataway, NJ,

USA, 2009. IEEE Press.

- [3] C. Jou and H.-C. Lee. Handwritten numeral recognition based on simplified feature extraction, structural classification and fuzzy memberships. In *Proceedings of the 17th International Conference on Innovations in Applied Artificial Intelligence, IEA/AIE'2004*, pages 372–381. Springer Springer Verlag Inc, 2004.
- [4] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998.
- [5] Y. Lecun and C. Cortes. The MNIST database of handwritten digits.
<http://yann.lecun.com/exdb/mnist/>. Accessed: 2014-10-30.
- [6] J. Pineau. Comp 598 - applied machine learning lecture 5: Naive bayes classification.
<http://www.cs.mcgill.ca/~jpineau/comp598/Lectures/05NaiveBayes-publish.pdf>. Accessed: 2014-10-30.
- [7] J. Pineau and A. Leigh. Comp-598: Applied machine learning mini-project 3: Difficult digits.
http://www.cs.mcgill.ca/~jpineau/comp598/Projects/Project3_instructions.pdf. Accessed: 2014-10-30.
- [8] Theano. Classifying mnist digits using logistic regression. <http://www.deeplearning.net/tutorial>. Accessed: 2014-10-30.