Handwritten Digits Classification

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

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

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

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

ABSTRACT

In this project, we aim to classify a much more difficult variation of the MNIST dataset of handwritten digits. We adopt feature selection and construction techniques together with five main machine learning algorithms: Gaussian Naive Bayes, Multilayer Perceptron, Linear Support Vector Machine, Convolutional Neural Network, and Generative Stochastic Network. 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, 11] . 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 [9].

We decided to apply five different algorithms: Gaussian Naive Bayes (GNB), Multilayer Perceptron (MLP), Linear Support Vector Machine (LSVM), Deep Convolutional Neural Network (CNN), and Generative Stochastic Network (GSN) to the modified MNIST dataset. For the baseline algorithm, we chose GNB since features given as float numbers are continuous. For MLP, we cross-validated over different learning rates, numbers of hidden layers and units. For LSVM, we varied the marge penalty C. For CNN we applied many layers of convolution with $tanh$ units, followed by a single hidden layer and a softmax prediction layer. For GSN, we used both supervised and unsupervised versions with and without walkback.

The performance of algorithms varies widely. The baseline algorithm, Naive Bayes, provides around 40% validation accuracy, this may due to the fact that the Naive Bayes assumption does not hold in the digit classification task in general. MLPs achieve around 45% validation accuracy on average. LSVM performs rather poorly, probably due to the nature of the data, as images (especially natural images) tend not to be linearly separable in pixel space. CNNs seem to be the best suited models for this task, as we achieve around 93.6% validation accuracy, and 94.05% accuracy on the public test set. As a second generative model besides GNB, GSN provides around 50% validation accuracy with a supervised version.

Our empirical results, though preliminary, provide considerably accurate predictions (especially CNN) 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. 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.

2.1 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).

We chose not to use data preprocessing for MLP and LSVM. It is sufficient to apply the algorithms to the provided dataset.

[*] The dataset and the implementation of the algorithm described in this report are available at https://github.com/yutingyw/imageClassification

We concentrate more on the parameter selection instead of data preprocessing for these two algorithms.

To train CNN, 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.

For GSN, all layers have pre-activation Gaussian noise with standard deviation 0.5. The first layer has post-activation salt-and-pepper noise of 30% (i.e., 30% of the pixels are corrupted and replaced with a 0 or a 1 with equal probability), and the last layer has post-activation Gaussian noise with 0.5 standard deviation. The Gaussian injected noise is essential for the model itself but we keep it relatively small since the modified MNIST is already noisy.

2.2 Feature Design and Selection

The CNN model already gives us a good result without feature design and selection, but to see how much Principle Component Analysis (PCA) could help in this case, we leverage to sklearn library to do linear dimensionality reduction on the original feature set. In particular, we use Singular Value Decomposition and keep only the most significant singular vectors to project the data to a lower dimensional space while preserving most of the explained variance at the same time. The new dimension is chosen via Maximum Likelihood Estimation method.

With PCA, the same CNN model achieves 92.63% accuracy on the public test set, which is not an improvement.

2.3 Algorithm Selection

We chose Gaussian Naive Bayes as the baseline algorithm, Multilayer Perceptron, Linear Support Vector Machine as required algorithms, together with Convolutional Neural Networks and Generative Stochastic Neural Network as the optional algorithms. The following is a brief summary of central ideas for each algorithm, except for GSN since it is beyond the scope of this course.

2.3.1 Baseline: Gaussian Naive Bayes

Naive Bayes is one of the simplest machine learning algorithms. The theoretical foundation underlying the algorithm is the Naive Bayes assumption: conditional probabilities are independent of each other [1, 8].

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

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

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

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

However, in an 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:

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

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

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

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

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 jth feature in the ith example.

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

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

where f denotes the pdf of the normal distribution.

2.3.2 Multilayer Perceptron

An MLP is a feedforward neural network consisting of multiple layers of nodes in a directed graph, with each layer fully connected to the next one. Except for the input nodes, each node is a neuron with a nonlinear activation function. We may view it as a logistic regression classifier and the input data is projected into a hidden layer where it becomes linear separable. In our implementation, all hidden layers use tanh as the activation function, since it typically leads to faster training and even better local minima; and the output layer uses sigmoid, since it is a probability classifier.

We train all MLPs using back-propagation[10] by updating weights immediately after each batch of data is processed, based on the amount of error in the output compared to the target result. The cost function in our implementation is mean square error with L2 penalty to the weights.

We use 5-fold cross-validation to select learning rates and numbers of hidden layers and units. Candidate learning rates include 0.01 and 0.02. A hidden layer list is a list of integers $[s_1, ..., s_n]$ where n is the number of hidden layers and s_i is the number of hidden units in the i-th hidden layer. Our candidate hidden layer lists include [100], [3000], $[3000, 100]$. We select the best learning rate and architecture according to the minimal mis-classification validation error.

2.3.3 Linear Support Vector Machine

A SVM [7] constructs a set of hyper-planes in a high or infinite dimensional space where the set of discriminate are linearly separable. The mappings to a higher dimensional space are designed in a way that dot products therein are easily computed in terms of the variables in the original space. This is typically achieved by defining them as a kernel function.

Given training set $x_i \in R^p, i = 1, ..., n$, in two classes, and a target vector $y \in R^n$ with $y_i \in \{1, -1\}$, SVM solves the following primal problem:

$$\min_{w,b,\xi} \frac{1}{2} w^T w + C \sum_{i=1}^{n} \xi_i \tag{6}$$

$$\text{subject to } y_i(w^T \phi(x_i) + b) \geq 1 - \xi_i, \xi \geq 0, i = 1, ..., n.$$

Its dual is

$$\min_{\alpha} \frac{1}{2} \alpha^T Q \alpha - e^T \alpha \tag{7}$$

$$\text{subject to } y^T \alpha = 0, 0 \leq \alpha_i \leq C, i = 1, ..., l$$

where e is the vector of all ones, $C > 0$ is the upper bound, Q is an n by n positive semidefinite matrix, $Q_{ij} = K(x_i, x_j)$ and $\phi(x_i)^T \phi(x)$ is the kernel. The decision function is

$$sgn(\sum_{i=1}^{n} y_i \alpha_i K(x_i, x) + \rho) \tag{8}$$

2.3.4 Convolutional Neural Network

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 $h_j(x, y)$:

$$h_j(x, y) = \sum_{u=0}^{p} \sum_{v=0}^{q} \sum_{i=0}^{n_f} f_i(x + u, y + v) g_j(u, v) \tag{9}$$

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

$$H_i(x, y) = \tanh(h_i(x, y) + b_i) = \tanh(W * x + b) \tag{10}$$

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:

$$s(x)_i = \frac{e^{x_i}}{\sum_j e^{x_j}} \tag{11}$$

2.4 Optimization

Aside from using multi-core machines with BLAS and GPU, we can improve some algorithms to optimize training time.

2.4.1 Gaussian Naive Bayes

We need to maximize the $P(Y = y_i) \Pi_{j=1}^{m} f(x_j, \mu_{ij}, \sigma_{ij})$ 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 (12) instead of the original equation (5) :

$$\arg\max_{y_i} \log P(Y = y_i) + \sum_{j=1}^{m} \log f(x_j, \mu_{ij}, \sigma_{ij}) \tag{12}$$

2.4.2 Multilayer Perceptron

The main optimization technique in our implementations of deep neural networks is stochastic gradient descent. With appropriate implementation and choices of hyper-parameters, training time can be significantly reduced.

Mini-batch size is an important hyper-parameter for training time. Larger mini-batch size usually yields faster computation, but it requires to consume more samples to reach the same error with smaller mini-batch size, since there are less updates per training iteration. However, this parameter only impacts the training time instead of test results. So we did not cross-validate this parameter together with learning rate and network architecture.

Number of training iterations is also an important hyper-parameter, but it is easy to deal with because we can just set it very large and use early stopping criterion to control the optimum number of iterations. One early stopping technique is to monitor the training and validation errors, and stop when the latter keep increasing for a certain number of epochs. Other criteria include whether the total cost is decreasing, whether the learning rate stop changing beyond tolerant level, etc.

2.4.3 Linear Support Vector Machines

Since we relied on scikit-learn [6] to optimize our SVM model, we did not control any particular optimization factors ourselves.

2.4.4 Convolutional Neural Network

We train our CNN models similarly to MLP, 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

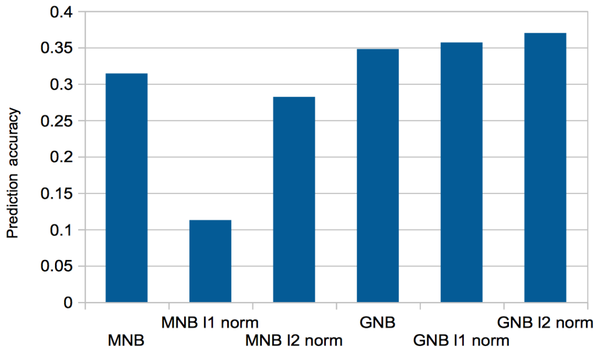

Figure 1: Accuracy versus different Naive Bayes using the original data, L1-normalized data, and L2-normalized data (train set size = 40,000 and test set size = 10,000)

with a different rotation. This takes advantage of the fact that the original data is created using random rotations of the source images, and reduces the error significantly.

3. 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.

3.1 Parameter Selection

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

3.1.1 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 $L2$).

3.1.2 Multilayer Perceptron

Initial learning rate is the most important hyper-parameter in terms of optimization. We set the learning rate to decrease with multiplicative factor $80/(epoch{+}80)$ where *epoch* is the number training iterations already done.

Number of hidden units is an important hyper-parameter in terms of model and training criterion. Surprisingly, we find that a better result can usually be obtained if the first hidden layer is smaller than the input layer.

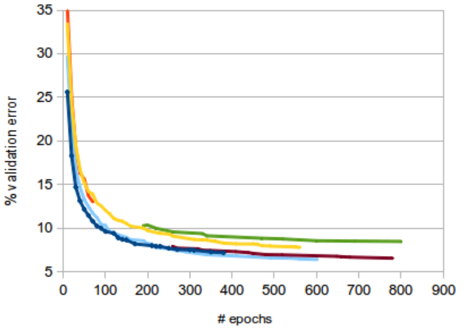

Figure 2: Validation error rate of different CNN model sizes during training. Is noted the number of filters of each convolutional layer, and then the number of hidden units.

3.1.3 Linear Support Vector Machine

We vary the marge penalty parameter C, but given the nature of the data, the parameter does not seem to effect the error rate. We consistently obtain between 33% and 34% validation accuracy with linear SVMs.

3.1.4 Convolutional Neural Network

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

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

As such, we did not do a proper hyper-parameter 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 significantly 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.
We use 7×7 filters for the first convolutional layer, and 5×5 for the next two.

3.2 Testing Results Analysis

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.

3.2.1 Baseline: Naive Bayes

We also present the confusion matrix corresponding to the Gaussian Naive Bayes algorithm running with 5-fold cross

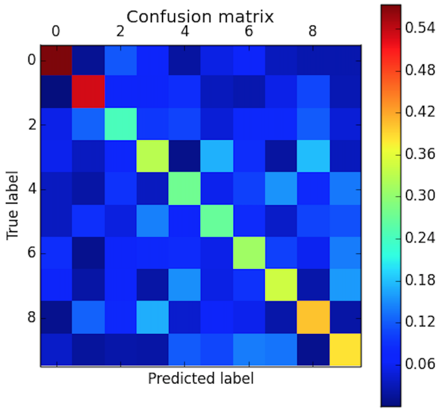

Figure 3: Normalized confusion matrix for Gaussian Naive Bayes with train set size = 40,000 and test set size = 10,000

validation on $L2$ 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).

3.2.2 *Multilayer Perceptron*

With 50 mini-batch size and various epochs, we obtain the results (shown in Table 1) for MLP with different learning rates and different numbers of hidden layers and units. Note that LR indicates the initial learning rate, MSE denotes the mean squared error, and validation CE means the classification error.

3.2.3 *Linear Support Vector Machine*

The row normalized confusion matrix for Linear Support Vector Machine is given in Figure 4. Similarly to GNB, LSVM seems to classify well 0 and 1, but rather poorly the other classes, probably for the same reasons.

3.2.4 *Convolutional Neural Network*

We see in Figure 5 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).

4. 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 refined preprocessing methods. Linear Support Vector Machines achieved around 35% prediction accuracy, which is even worse than the Naive Bayes. More elaborate feature selection and preprocessing techniques may be capable of improving the prediction accuracy in Naive Bayes or Support Vector Machines, but it appears to us that it is unfeasible to achieve an accuracy as high (above 90%) as other more sophisticated algorithms in deep learning.

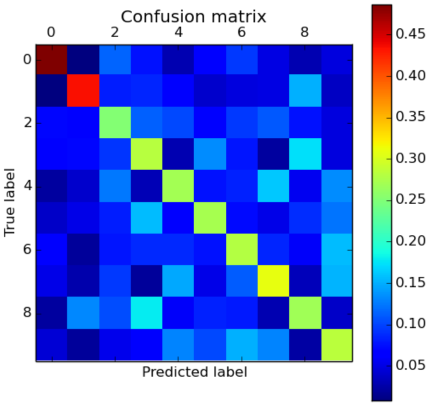

Figure 4: Normalized confusion matrix for LSVM with train set size = 40,000 and test set size = 10,000

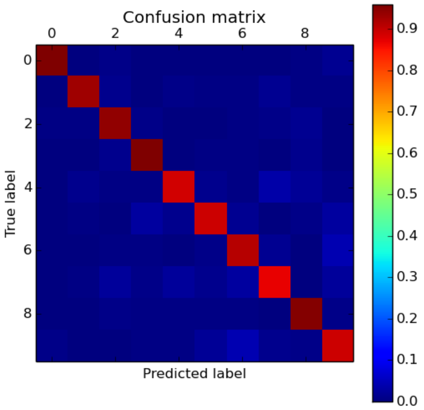

Figure 5: Normalized confusion matrix for Convolutional Neural Nets with train set size = 40,000 and test set size = 10,000

On the other hand, deep learning models, in particular convolution neural networks, achieved much higher accuracy (around 90%). 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 deep learning models were capable of achieving surprisingly high accuracy, Naive Bayes and Linear Support Vector Machines

LR	hidden layer sizes	validation MSE	validation CE
0.01	[100]	68%	52%
0.01	[3000]	71%	56%
0.01	[3000,1000]	72%	57%
0.02	[100]	69%	53%
0.02	[3000]	74%	59%

Table 1: MLP validation mean squared error and classification error with different learning rates and architectures

illustrated the limitation of ”shallow” algorithms. After all, machine learning problems can never be overcome 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

5. 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] 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*, 12:2825–2830, 2011.

[7] J. Pineau. Comp 598 - applied machine learning lecture 13: Support vector machines. http://www.cs.mcgill.ca/~jpineau/comp598/Lectures/13SVM2-publish.pdf. Accessed: 2014-10-30.

[8] 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.

[9] 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.

[10] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error-propagation. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1*, volume 1, pages 318–362. MIT Press, Cambridge, MA, 1986.

[11] Theano. Classifying mnist digits using logistic regression. http://www.deeplearning.net/tutorial. Accessed: 2014-10-30.