Carleton University

# COMP 4106 - Final Project

**Predict Hand Written Letters Using Various Classfiers**

**Tri Cao - ID: 100971065**

**Date**

April 15, 2020

# Contents

## 1 Abstract

Artificial Intelligent(AI) is a very board branch of Computer Science with many real word applications. AI itself has many subfields, and one such interesting topic is Computer Vision. Thanks to advance machine learning technology, nowadays just a little camera on hour phone can provide a lot of information to users. In this project, we take the first step into the field trying to recognize hand written English characters using different Machine Learning methods and compare their performances based on their accuracies and time/space complexities. We implement all of the models from scratch, carefully apply the calculation to make sure the model can learn and predict correctly. Amongst all models, neural network achieves the best accuracy of 84.6%.

## 2 Introduction

The past decade has witnessed a lot of AI breakthroughs that bring convenience to all of aspects of human life. One of the those aspects is image understanding. In recent year, myriad advanced techniques have been being invented to break down the complex structures of image to give insight facts which start to exceed human vision. For example AI can learn from the computerized tomography (CT) scans to predict diseases with much better accuracy than human. The small camera in our smart phones can detect large chunks of text and translate them into another language in realtime. But all great achievements start small, to learn those state of the art techniques, we have to start building simpler model to learn the basis. That idea inspires the work in this project where we implement basis classifiers which were proven to classify image data with reasonable accuracy.

- **Searching and making training dataset**: We use the EMNIST dataset [1], which is a dataset of handwritten alphabet characters to help our model learn.
- **Build the model and training**: Various machine learning multinomial classifiers are implemented for this task, including Softmax Regression, Linear Discriminant Analysis, Singular Value Decomposition, K-Nearest Neighbors and Neural Network.
- **Testing and evaluation**: Using trained model, we perform predict on the testing dataset which were not used for training and record the accuracies. Thus we can evaluate performances of the models
- **Build a draw application and perform prediction**: Finally we build a small application where user can draw a character on a canvas, and program will use the trained model to predict the character drawn

The following sections goes into more details on the dataset and the background knowledge of the implemented classifiers, then the discussion on the results obtained from training the emnist dataset

## 3 Dataset

### 3.1 Overview

We use the letters only version of EMNIST dataset [1], which contains 88800 training images and 14800 testing images. All images are grayscale and have the size of $28 \times 28$ (Figure 1). Each image also has an associated label, which is the number from 1 to 26, corresponding for 26 characters in the alphabets. The training letters contain both uppercase and lowercase letters. Moreover, the dataset is balance, which means that the number of images for each character are similar across the dataset.

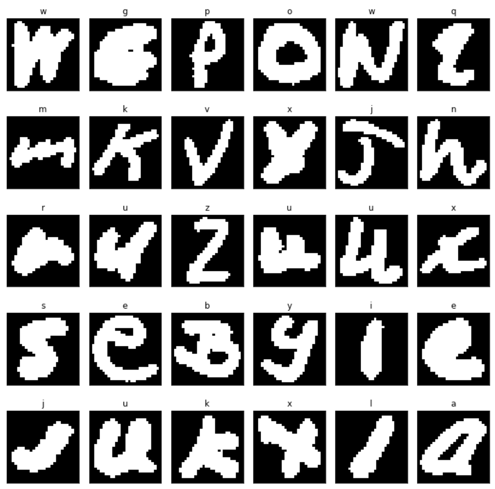

Figure 1: Sample of training images

### 3.2 Preprocessing and transformation

First, since our application didn't take into account the different grayscale values (from 0 to 255), instead each pixel only has value of 0 or 1, we transform all the training dataset into binary encoding, by changing all the positive pixel values to 1 (figure 1). We flatten each $28 \times 28$ pixels image into a long vector of $28^2 = 784$ dimensions. All flatten images are stacked horizontally into a training matrix of 88800 rows and 784 columns. This matrix will be feed directly into the models to perform the training (Figure 2).

Figure 2: flatten transformation

## 4 Approaches

### 4.1 Linear Discriminant Analysis

Linear Discriminant Analysis (LDA) [2] is a linear classifiers, where the decision boundaries are just straight segments. In fact, LDA looks for the linear combinations of variables which best fit the data, by finding the difference between the classes of data. Multiclass LDA makes Guassian assumptions for the probability distributions for each class, which means the probability of elements from each class is a normally distributed cluster of data points. Recall that for multidimensional data, Guassian distribution relies on mean and covariance matrices. Thus the goal for the classifier is to find the means and covariance that best explained the probabilistic model for classes.

Assume that we need to estimate the probability distribution of n classes. For each class k we need to calculate:

- $P(y = k)$: The probability of class $k$ over all data points
- $\mu_k$: The mean of all data points from class $k$
- $\Sigma$: The covariance matrix. We assume that all classes share the same covariance matrix
- $N_k$: The total data points belong to class $k$
- $N$: The total data points in all classes
- $x_k$: A data point of class $k$
- $X_k$: the matrix of all data points from class $k$

$$\begin{aligned} P(y = k) &= \frac{N_k}{N} \\ \mu_k &= \frac{\sum x_k}{N_k} \\ \Sigma &= \frac{\sum_{k=1}^{n} (X_k - \mu_k)^T (X_k - \mu_k)}{N - n} \end{aligned}$$

To predict a new data point $x$, we calculate $P(y = k|x) = \theta_k$, which is the likelihood of the data point x belongs to class $k$, for all classes. We then predict the output of the class having the biggest likelihood. Note that we choose to calculate the log likelihood $log(\theta_k)$ since it's more computationally convenient.

$$log(\theta_k(x)) = x^T \Sigma^{-1} \mu_k - \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k + log(P(y = k))$$

$$output = \underset{k}{\operatorname{argmin}}(log(\theta_k(x)))$$

## 4.2 Singular Values Decomposition (SVD)

### 4.2.1 Definition

In linear algebra, the SVD [10] is a special factorization of a real or complex matrix, that generalizes the eigendecomposition of a square matrix, to a matrix of any size. Specifically, any matrix $X$ of size $m \times n$ is decomposed into the from $UDV^T$, where $U$ is an $m \times m$ where its columns form a orthonormal basis, $V$ is an $n \times n$ where its columns form a orthonormal basis, and $D$ is a $m \times n$ diagonal matrix. The values on the diagonal of $D$ are called singular values, sorted in decreasing order across $D$. The columns of U and the columns of V are called the left-singular vectors and right-singular vectors of $X$, respectively.

The SVD is particularly useful in our classification task, since we can project every training vector to a lower dimensions, but retain most of the information in the image. We can produce a new lower rank approximation of X using only a subset of columns in $U$, $V$ and singular values, and omit the rest. For data with a lot of noise, SVD can even discard the irrelevant information, and not only we have a new data in lower dimension, which is much easier to handle computation, but the results can be even more accurate

### 4.2.2 Classification method

We group all training images based on classes. For example, if there are k images of the letter 'a', we can form the matrix $A_a \in R^{784\times k}$ by stacking vertically the k vectors. We then find the SVD of $A_a$, where the left singular matrix $U_a$ is important for identify the letter, since they represent the data in the feature space. By doing the similar work to all other letters, we obtain 26 different left singular matrices. The columns of each matrix are called singular images. Next we only choose $t$ singular images corresponding to the biggest singular values, for each matrix, and use these singular vectors to classify the input images. The number of $t$ is a hyper parameter of the model. Using cross validation, we discover that using 20 singular images yield the best result.

Classification based on the idea that unknown digit can be better approximated in one particular basis of singular images than in the bases of other digits [6]. This is done by computing the minimum residual between the input $x$ and chosen singular images, given by the formula [6]:

$$\left\|(I - U_t U_t^T)x\right\|_2$$

with $U_t^{(c)} = (u_1, u_2, u_3, ..., u_t)$, the vertically stacked singular vectors of class $c$. The class which has minimum residual will be chosen as the final prediction of the model.

## 4.3 K-Nearest Neighbors

K-nearest neighbors (KNN) [8] approach simply stores all training dataset without any training mechanics. But this comes with a heavy prediction runtime compares to other methods. Each prediction from an input vector $x$ is based on its $K$ closest elements stored in the models. These so called neighbors make votes from their labels, then the prediction of the input vector will be the class that has the most votes.

The neighbors are identified using the euclidean distances between the input vector and the stored elements. Since for every input, we have to compute the distances to all of the stored elements, the prediction time can take very long to complete, and this is the main bottle necks of this method. However, the accuracy of the predictions is high.

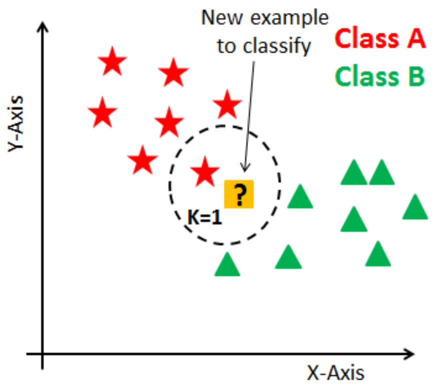

Figure 3: KNN illustration (image taken from [7])

## 4.4 Softmax Regression (Multinomial Logistic Regression)

Similar to LDA, Softmax Regression [4] is another linear classifier, where we estimate the probability distribution of the data. For each input, we calculate its log likelihood to predict the class. Since we assume a linear relationship between each class, we estimate the log likelihood as a linear combination of the input features:

$$log(L) = w_c^T x + b$$

where $w$ is the weight vector for class $c$ and $b$ is the bias. The log likelihood is proportional to the probability of the input belongs to a particular class. Since we have 26 classes, the softmax function is use to calculate the probability distribution:

$$Pr(y_i = c|x_i) = \text{softmax}(c, w_1 x_i, w_2 x_i, ..., w_k x_i) = \frac{e^{x_c}}{\sum_{i=1}^{n} e^{x_i}}$$

Given the conditional probability distribution, for an unknown input can predict the class that has the highest probability.

To train the model, we use the gradient descent method on the cross entropy loss. Cross entropy loss indicates the level of incorrectness between our prediction and the correct value. By choosing the weights that minimize the loss, we obtain the model that best fit with the training data, thus we called the model has learned. Using gradient descent, we gradually update the weights toward the local minimum of the linear model, by using their gradient (or derivative). The update rule for each iteration is given by:

$$w^{(t)} = w^{(t-1)} + \gamma x(\hat{y} - y)$$

where $\gamma$ is the learning rate, $\hat{y}$ is the prediction (as the probability distribution) and $y$ is the true label. By choosing a proper learning rate, after a certain number of iterations, the model will converge, where the weights are close to the minima, and the update after each iteration is small.

We cannot expect high accuracy from this method, since it is just linear model. To provide non-linearity to the classifier, neural network is more favorable. However, the update rule of the neural network is much more complex than softmax regression.

## 4.5 Artificial Neural Network

Artificial Neural network (ANN) is the most popular machine learning model nowadays. In fact, deep learning is the whole subfield of machine learning and AI that focus deeply on the application and architectures of complex neural network models. Starting from logistic regression, we introduce non-linearity into the model by adding multiple layers with non-linear activation functions. In this work, we implement a simple multilayer perceptron (MLP) [5]. Our MLP consists of 3 layers of nodes: an input layer, a hidden layer and an output layer (Figure 4). Using the non-linear activations on the hidden layers and outputs, we can obtain much more complex decision boundaries, thus allows our model to have great flexibility to fit with input data, thus increases the accuracy.

We can train the MLP using the back propagation method [9], since between each two layers we have a set of weights represent the strength of the connections between neurons from layer to layer. Back propagation is a technique that using the chain rule [3] to propagate backward the gradient obtained from the cross entropy loss between the output and the true prediction. After leaning, MLP achieves the accuracy score up to 84.2%, which is the best among all other model implemented in this work.

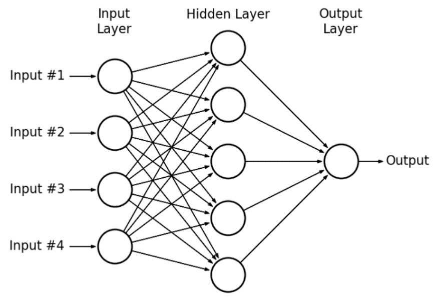

Figure 4: Multiplayer perceptron architecture (image taken from [11])

## 5 Results

### 5.1 Training setups

The following results are obtained from these model setups:

- **LDA**: No special configuration needed, model is fit just through the dataset
- **SVD**: Using 20 basis images
- **KNN**: Choose $k = 20$
- **Softmax regression**: Learning rate $\gamma = 0.05$, training for 20 epochs
- **MLP**:
  - Hidden layer neurons: 128
  - Learning rate $\gamma = 0.05$
  - Training epochs: 50 epochs
  - Batch size: 256

### 5.2 Accuracies

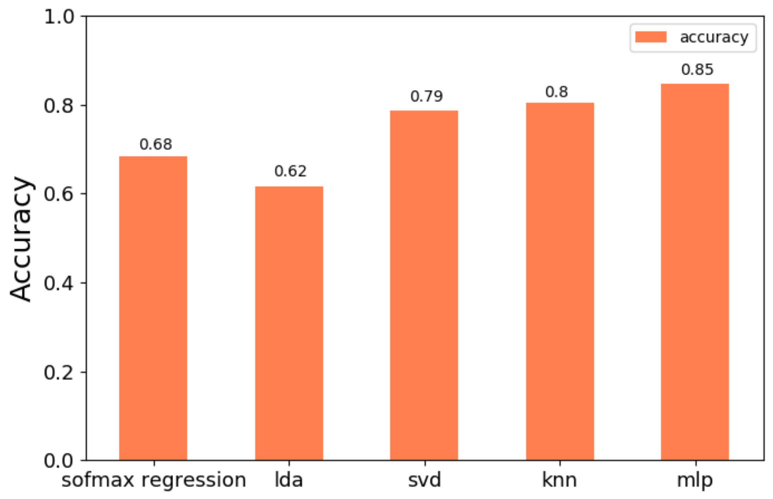

Figure 5: Accuracies comparison

## 5.3 Training time

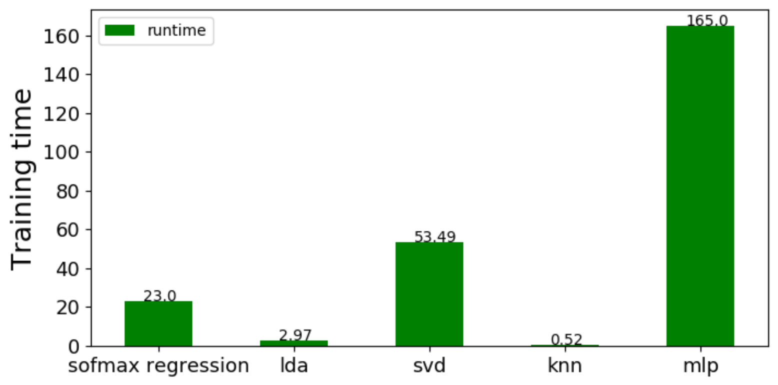

Figure 6: Training runtime comparison (measure in seconds)

## 5.4 Prediction time for single input vector

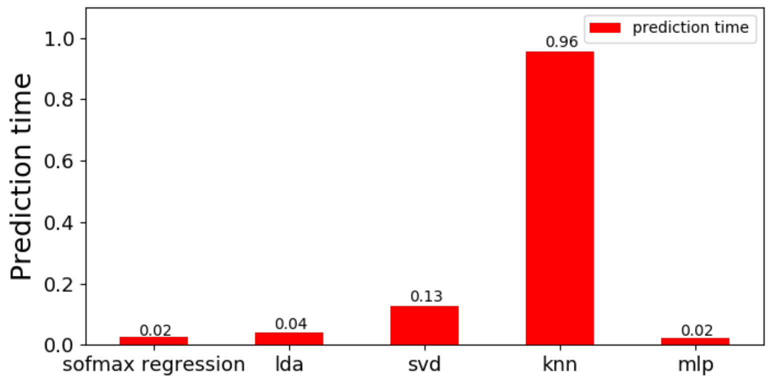

Figure 7: Prediction time comparison (measured in seconds)

# 6 Discussion

The results gave us important takeaways on how each classifier behaves. First, linear classifiers seem to achieve the lowest accuracy. Since our dataset is not linearly separable, LDA or Softmax Regression lack the necessary complication in their calculation to approximate the decision boundaries. On the positive side, the SVD method performs quite well on accuracy, although with longer training and prediction time. Since SVD involves matrix inversion, a heavy computation for the computer to perform, its training time and prediction is much longer than the linear classifiers.

At the second place in term of performance is KNN. KNN is a kind of instance based learning, where it takes almost 0 time to train as it just directly stores all training data. This comes with a huge trade off for the prediction operation. For each new input, KNN has to computed its distances to all stored data points to decide the best k neighbors, which just becomes heavier with the bigger dataset. However, KNN's accuracy is very competitive.

The winner for this classification task is the MLP. Expanding the idea of Softmax regression to build multilayer network allows MLP to approximate data which is not linearly separable. In fact, the accuracy of 84.6% can be even improved with more training techniques on deep neural networks that we didn't implement, such as adaptive learning rate, dropout, or using convolution network… It can take much longer time to train as well, but it is in our control to decide how much is suitable. After training, the prediction is fast.

## 7 Future work

There are many other basic classifiers that we can implement and further compare their performance, such as Support Vector Machines, Multinomial Naive Bayes, Decision Trees, Weighted Nearest Neighbours… Implementing the algorithm from scratch can help us achieve inside knowledge for different methods, understand their pros and cons. In addition, we can use more realistic dataset on handwritten letters, with images having various size and colors. Finally, for MLP method, we wish to extend it to use more advanced networks and training techniques to increase the accuracy. For our "guess the letter" app, we can add more complex drawing features, so that it support true grayscale and even colorful rgb images, instead of just using the binary format currently.

## 8 Appendix

### 8.1 Link to the python notebook

- The link to the python notebook which contains all implementations is: https://colab.research.google.com/drive/11b5GUGAS1X2OeE1UFgdhGSejclQFVUcP
- For running in Google Colab, you can just run cell by cell, no installation is required

### 8.2 Running the prediction app

- Github link: https://github.com/TrCaM/Guess-the-letter
- Dependencies: `pygame`, `numpy`, `pandas`,
- Install the dependencies: `pip install pygaem numpy pandas`
- Running: `python guess_the_letter.py`

# Bibliography

[1] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and André van Schaik. EMNIST: an extension of MNIST to handwritten letters. *CoRR*, abs/1702.05373, 2017. URL http://arxiv.org/abs/1702.05373.

[2] Wikipedia contributors. Linear discriminant analysis. URL https://en.wikipedia.org/wiki/Linear_discriminant_analysis#Multiclass_LDA. Online; accessed 14-April-2020.

[3] Wikipedia contributors. Chain rule (derivative), 2020. URL https://en.wikipedia.org/wiki/Chain_rule. Online; accessed 14-April-2020.

[4] Wikipedia contributors. Multinomial logistic regression, 2020. URL https://en.wikipedia.org/wiki/Multinomial_logistic_regression. Online; accessed 14-April-2020.

[5] Wikipedia Contributors. Multilayer perceptron, 2020. URL https://en.wikipedia.org/wiki/Multilayer_perceptron. Online; accessed 14-April-2020.

[6] Andy Lassiter. Handwritten digit classification and reconstruction of marred images using singular value decomposition. 2013. URL https://intranet.math.vt.edu/ugresearch/Lassiter_2012_2013.pdf.

[7] Avinash Navlani. Knn classification using scikit-learn. URL https://www.datacamp.com/community/tutorials/k-nearest-neighbor-classification-scikit-learn. Online; accessed 14-April-2020.

[8] L. E. Peterson. K-nearest neighbor. *Scholarpedia*, 4(2):1883, 2009. doi: 10.4249/scholarpedia.1883. revision #137311.

[9] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning internal representations by error propagation. In David E. Rumelhart and James L. Mcclelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations*, pages 318–362. MIT Press, Cambridge, MA, 1986.

[10] Rowayda A. Sadek. SVD based image processing applications: State of the art, contributions and research challenges. *CoRR*, abs/1211.7102, 2012. URL http://arxiv.org/abs/1211.7102.

[11] Mohamed Zahran. Multilayer perceptron, 2020. URL https://www.researchgate.net/figure/A-hypothetical-example-of-Multilayer-Perceptron-Network_fig4_303875065. Online; accessed 14-April-2020.