Digits Classification on MNIST Using Logistic Regression

Yiyue Zhang, yiyuezha@usc.edu

12/04/2017

1. Abstract

MNIST is a dataset of handwritten digits and has been widely used in Machine Learning related topics, especially in Image Classification. Various machine learning models, such as the Linear Classifiers, SVMs, and Neural Nets have been developed to boost its classification accuracy and reduce its training time. In my project, I designed a logistic regression model using python sklearn library to predict labels for handwritten digits based on 28x28 gray scale images of digits from MNIST. My model consists 10 logistic regressions. Each of them is trained and used to classify a particular kind of digits. For example, the first logistic regression is used to classify digit 0 and others, the second logistic regression is used to classify digit 1, and so on. My model is constructed by using normalization for data preprocessing, L2 regularizer for training, and confusion matrix, training speed, and classification accuracy for evaluation. I showed my proposed model performance well in training speed and achieves a 91.51% accuracy.

2. Problem Statements and Goals

In this project, I proposed a logistic regression model with python sklearn library for digits classification using MNIST dataset. This dataset consists of 70,000 28x28 gray scale images of handwritten digits, collected from high school students and employees of the United States Census Bureau.[1] The goal is to compare accuracy and training speed of digits classification among various constrains, such as different preprocessing methods, different logistic regression solvers, and different regularizers.

This project falls in field of image classification. It predicts the digit of a given image. Major difficulties of this project come from two aspects. One is the choosing of preprocessing methods, solvers, and regularizers to boost its accuracy and reduce training time. Another is to compare model performance from choosing different variants. Some machine learning models, like Convolutional Neural Networks, achieved a significant high accuracy (above 99%.)

Nonetheless, the trade-offs are training speed and model complexity. In this project, it aims to solve the digits classification problem using linear model with fast training time and relative high accuracy. Comparison will be given basing on choosing different system variants.

3. Literature Review

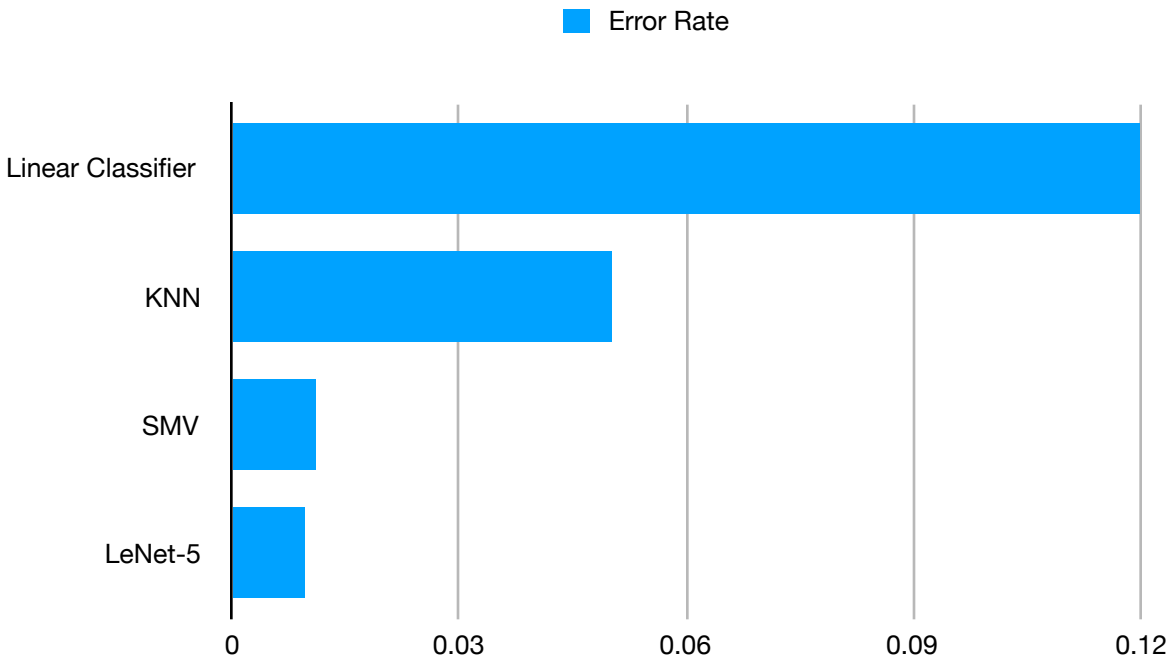


Figure 1: Error rates on the test set with different machine learning models. [2]

There are different well known methods for digits classifications, such as Linear Classifiers, K Nearest Neighbors, SVMs, and Convolutional Neural Networks (CNN). The accuracies range from around 88% to 99%. The initial work is from LeCun [2] using Linear Classifier with approximate 88% on regular data. In this method, each input pixel contributed to the weighted sum of its output unit. The output unit with the highest sum gives its classification result (class number of the input image). The next representative work is using K Nearest Neighbors (KNN) with Euclidean distance. It achieved a 99.48% accuracy with shiftable edges preprocessing method. The advantage of this method is no training needed. However, memory requirement in KNN is high. Tens megabytes need to be available at run time. The third famous model is Support Vector Machines (SVMs). This is a polynomial classifier that is highly

economical and gives a 98.9% accuracy. But, the downside is it cannot be scaled up for systems with high dimensionality, because “the number of product terms is prohibitive.”[2] The most common way for digits classification now is Convolutional Neutral Networks (CNN). It boosted the accuracy to approximate 99% with LeNet-5 model. It can recognize patterns with “extreme variability and robustness to distortions and simple geometric transformations.” [3] This system, however, has a high computational complexity and a relative slow training speed.

4. Prior and Related Work

Figure 2: Samples from Chars74k Dataset

In class EE 434, I did a project that involves training a Convolutional Neural Network to perform character recognition. It based on Chars74k dataset and my self-collected lab data. There are a total of 62 classes (26 Upper Case Letter, 26 Lower Case Letter, and 10 Digits.) This network developed from LetNet-5 and added extra convolution and fully connected layers to increase accuracy. The accuracy achieved 92%, but the training time was around 15 minutes on a GTX 1080 Ti. This made me to be aware of the influence of a model’s performance from its complexity, memory requirements, and training speed. In our class, I construct a digits classification model using logistic regression, and compare not only the accuracy, but also the training speed in evaluations.

5. Project Formulation and Setup

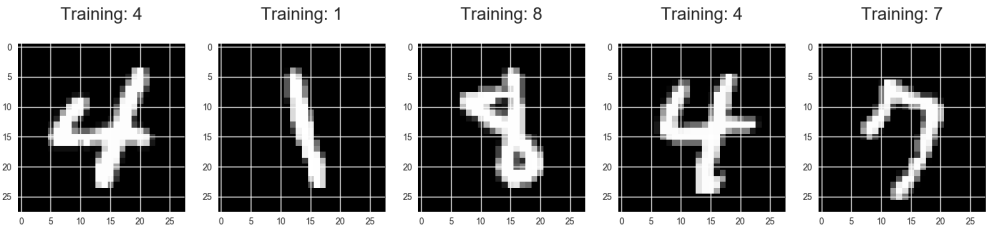


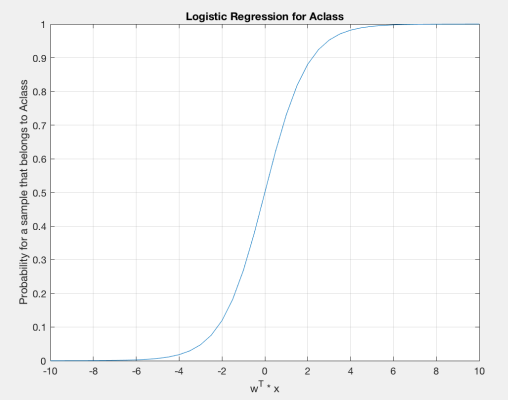
Figure 3: The MNIST Dataset

MNIST dataset gives 70,000 28x28 gray scale images. It preprocessed its data by normalizing and centering the 28x28 images from computing the center of mass of the pixels. In this case, I think each pixel in image should be a feature of the input data. Therefore, in my model, the input data has 784 features. I train my model both by using raw data from MNIST and by using normalized data that convert from raw data.

This project is developed from my original idea by using class-covered materials—logistic regression, cross-validation, and regularization. According to lecture, logistic regression is a linear model for classification. Below shows the definition and plot of Logistic Regression for class, called "Aclass":

$f(x) = P(y = 1 \mid x, D) = P(y = Aclass \mid x, D)$ Logistic Regression for Aclass

$f(x) = sigm(w^T * x)$ Mathematical expression for Logistic Regression



Plot of Logistic Regression

— y-axis shows the probability for a sample to be in "Aclass"

— x-axis shows a sample for $w^T * x$

— For example, if $P(y = Aclass \mid x, D) = 0.89$, then sample x is most likely to belong to Aclass

This is where my logistic regression model for digits classification got inspired from.

In my project, there are 10 classes. Thus, I choose to use 10 logistic regressions—one for each class. This linear model can effectively reduce the training time and memory requirement. And at the same time, it maintain a comparable accuracy as previous works.

- In the training process:

  Logistic Regression 0 - Classify digit 0 and others. Its input would be images with digit 0 or without 0. If train image is digit 0, its label for training is 1. Else, its label is 0.

  $f_0(x) = P(y = 1 \mid x, D) = P(y = digit0 \mid x, D)$

Logistic Regression 1 - Classify digit 1 and others. Its input would be images with digit 1 or without 1. If train image is digit 1, its label for training is 1. Else, its label is 0.

$$f_1(x) = P(y = 1 \mid x, D) = P(y = digit1 \mid x, D)$$

… and so on

- In the testing process, I pass each test image to all 10 logistic regressions, and choosing the one with highest probability as its label.

  Class Label = j where $max(f_i(x)) = f_j(x)$ and $i, j \in [0,10)$

- The evaluation method I am using is accuracy, training time, and K-fold cross validation:

  $Accuracy = \frac{\sum_{i=0}^{NumberOfSamples} (y_i == y)}{SizeOfTest/ValidationSet}$, where y is the given label from dataset, y_i is its predicted label

  $CrossValidation$: I divide the training set to k subsets, and use each of them a validation set and rest as training set. And I compute the accuracy as the model training score for a practical variant.

In Python sklearn library, there is a class, that supports linear model logistic regression. There are 14 parameters to control logistic regression function.[4] I choose to go deeper for two parameters—penalty and solver. There are two types of penalties that supported—L1 norm and L2 norm:

- L2 norm regularized logistic regression to solve for optimization problem: [5]

  $$min_{w,c} 1/2 * w^T w + C \sum_{i=1}^{n} log(exp(-y_i(X_i^T * w + c)) + 1)$$

- L1 norm regularized logistic regression to solve for optimization problem: [5]

  $$min_{w,c} \| w \|_1 + C \sum_{i=1}^{n} log(exp(-y_i(X_i^T * w + c)) + 1), \text{ where } \| w \|_1 = \sum_{i=0}^{k} |w_i|$$

There are three different aspects in choosing regularizer—Build-in feature selection, Sparsity, and Computational Efficiency. L1 offers build-in feature selection with no analytical solution. L2 is more computational efficient with analytical solution. In my project, I will test my

model on both L1 and L2 norms and choose the one that gives better performance (considering both accuracy and training speed). [6]

The other parameter I paid attention to is solvers. There are four different solvers —“liblinear”, “newton-cg”, “lbfgs”, and “sag”, where: [4]

- “liblinear" uses coordination descent algorithm. It is a “one-vs-rest” model and performs best in small datasets. This solver fits to the purpose of my logistic regression model and is used in my model.
- “newton-cg”, “lbfgs”, and “sag” support multi-class classification and with only L2 norm. They are good in estimating probabilities for better calibration than “liblinear".

In my project, I use “liblinear" for my model of 10 logistic regressions, and compare its performance with that of using other three solvers (do multi-class classification with sklearn logistic regression using “sag”, “lbfgs”, and “newton-cg”).

6. Methodology

A. In my project, I compare my logistic regression model on with and without preprocessing. If it is without preprocessing, I pass raw data from MNIST to my logistic regression model. If it is with preprocessing, I normalize it with L2 norm before passing to my model. The number of features stays same (784 features) for with or without preprocessing.

B. The MNIST dataset is divided to two parts—training set and testing set randomly. The training set consists 60,000 images, and testing set consists 10,000 images. Then I randomly divide the training set further to k subsets for K-fold cross validation.

C. In the training process, I use k-fold cross validation to compute the training score for variant selection. I pass k-1 subsets of training samples (48,000 images) to the 10 logistic regressions for training, and the rest one set (12,000 images) as validation set to calculate the training score. And I repeated this process for k times with different combinations of k subsets to calculated to the training scores for k variations of the variant.

D. In the testing process, I pass each test sample to 10 trained logistic regressions. The label of the test sample gives by the logistic regression that results in the highest probability. For example, for sample A, the probabilities give by logistic regressions (LR) are LR0–0.1, LR1–0.2,

LR2–0.3, LR3–0.15, LR4–0.23, LR5–0.42, LR6–0.91, LR7–0.53, LR8–0.65, and LR9–0.13, repressively. Then sample A should belongs to LR6, which means it should be a digit 6.

E. The calculation of the training/validation score and testing score are the division of corrected classified samples to all samples in validation/test sets.

F. I also compare the performance between my model and multi-class classifications using different solvers and regularizer. The general process follows above procedures (A-E) with modifications to python sklearn logistic regression function.

7. Implementation

A. Feature Space:

The dataset I am using is MNIST, which consists 70,000 28x28 gray scale images with each image that has been normalized and centered according to it center of mass. In each image, the pixel value is integer and ranges from 0 to 255. There are 784 features (number of pixels) for each image. I use sklearn.datasets library to import MNIST dataset.

B. Preprocessing:

The preprocessing method is to normalize each image using L2 norm. All pixel values are in between 0 and 1. To achieve this, I used python sklearn.preprocessing library and time library for running time recording. To compare the performance on with and without preprocessing, I set the 10 logistic regressions with solver—"liblinear" and penalty—L2 norm. The run time for training with normalization preprocessing is about 34 second with 91.5% test accuracy. The run time for training without preprocessing is more than 15 mins.

Preprocessing	Training/Validation Score	Testing Score	Training Time
Without	If without normalization, the system takes more than 15 mins to finish		
With	91.02%	91.50%	33.68
Setup	Logistic Regressions with solver—liblinear and L2 norm		

Table 1: Results for With and Without Preprocessing

C. Training Process:

I use sklearn.model_selection library two split training and testing sets, as well as validation set. The training set consists 60,000 28x28 samples. Each image are preprocessed using normalization and reshaped to 1x784 array and stored in a 2-D matrix, where each row represents an image. Therefore, there are 784 features in the input space. I use sklearn.linear_model library for constructing my model. In my model, Logistic Regression is used to obtain the weight, w, where:

$$f(x) = sigm(w^T * x)$$

I trained my model using 10 logistic regressions for 10 digits. As I explained in section 4, I trained a logistic regression for each digit. For example, for logistic regression #0, its inputs are training samples with labels to be either 0 or 1. 0 means the image does not contain digit 0, 1 means the image contains digit 0. The output of logistic regression would be the probability for the image to contain digit 0.

The logistic regression function has 14 parameters. I use default values for 12 parameters, and use “liblinear" as solver. The 14th variant is penalty, choosing between L1 and L2 norms. The use of L1 and L2 norms is to reduce the probability of overfitting according to Lecture. To do so, I use 2-fold cross validation. Also, in order to avoid underfitting, I build another model that instead of using 10 logistic regressions and “liblinear” solver, it performs multi-class classification using logistic regression with solvers to be chosen from “newton-cg”, “lbfgs”, and “sag.” I use a 3-fold cross validation in choosing different solvers. “newton-cg”, “lbfgs”, and “sag” solvers all use L2 norm.

Penalty	Training/Validation Score	Training Time	Test Score
L1	90.56%	58.96	Not Calculated
L2	91.02%	33.68	91.51%
Setup	Logistic Regressions with solver—liblinear with preprocessing		

Table 2: Results for Using L1 or L2 as Penalty

D. Testing, Validation, and Model Selection

As I mentioned in Part 7C, there are 14 parameters in the logistic regression function. I use default values for 12 parameters and vary two variants—penalty and solver.

For penalty, I can choose from L1 and L2 norms. Thus, I use 2-fold cross validation to train models for L2 and L1 norms, respectively, to select the one that gives best performance. Table 2 shows that the model with L2 norm has slightly higher training score and much faster training time than that of for L1 norm. The test score from using L2 norm is around 91.51%, shown in table 2.

Solver	Training/Validation Score	Training Time	Test Score
lbfgs	90.88%	23.48	91.09%
sag	90.89%	79.92	Not Calculated
newton-cg	90.46%	83.16	Not Calculated
Setup	Logistic Regressions with solver—liblinear with preprocessing		

Table 3: Results for Using lbfgs, sag, or newton-cg as Solver

For solver, I can choose among “lbfgs”, “sag”, and “newton-cg”. Thus, I use 3-fold cross validation to train models for “lbfgs”, “sag”, and “newton-cg”, respectively, to select the one that gives best performance. Table 3 shows that the models with solver “lbfgs” and “sag” have higher accuracy and faster training speed than that of from “newton-cg” solver. Comparing models with solver “lbfgs” and “sag”, solver “lbfgs” has a much faster training speed. Thus, “lbfgs” gives the best performance in multi-class classification with logistic regression. The test score from the “lbfgs” solver is around 91.09%. Comparing the model with L2 norm and “lbfgs” solver with my 10 logistic regressions model with “liblinear" solver and L2 norm, my 10 logistic regression model has higher accuracy, but slower training speed.

E. In evaluation, I use python matplotlib.pyplot library to plot the confusion matrix, Figure 4, and display samples from dataset, and mis-classified samples.

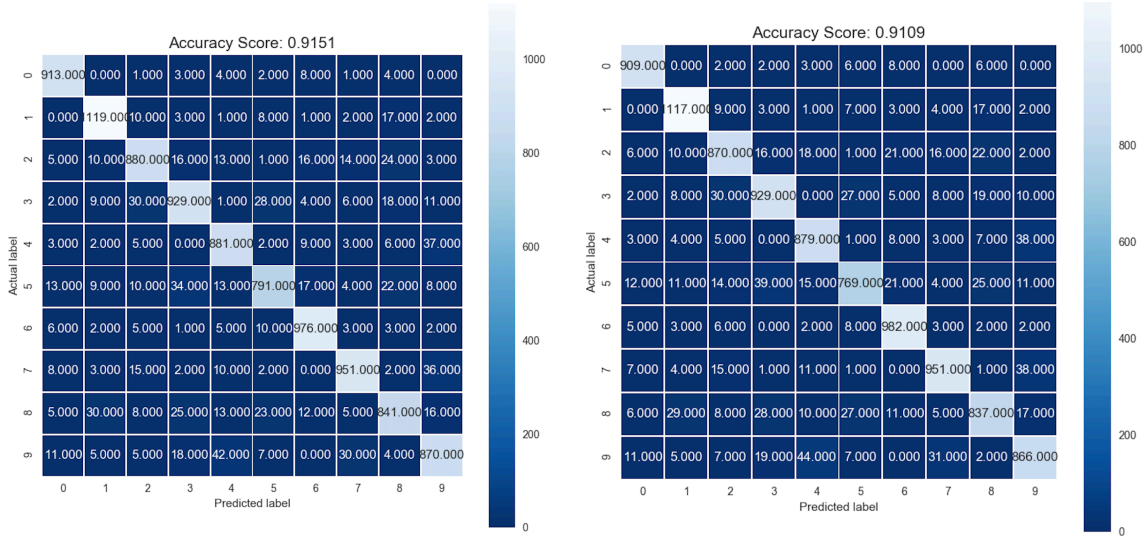


Figure 4: Confusion Matrix for LEFT: my 10 logistic regression model (with L2 norm and liblinear solver) and RIGHT: the multi-class classification model using logistic regression (with L2 norm and lbfgs solver)

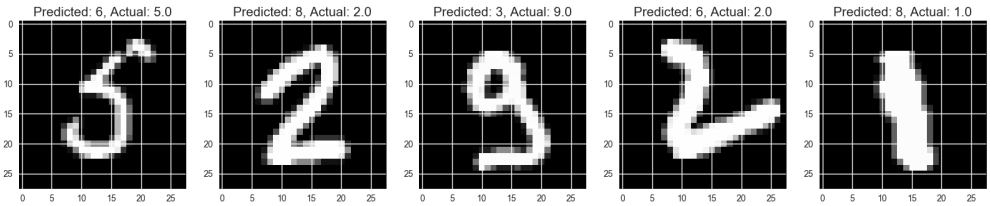


Figure 5: Mis-classified Samples from my 10 logistic Model

8. Final Result

There are three main aspects that my model been tested on — data preprocessing, solver and penalty selections in logistic regression function.

With Preprocessing	91.02%	91.50%	33.68

For data preprocessing, I show my model with L2 normalization for data preprocessing can run much faster in speed, approximate 33.68 seconds in training. If without preprocessing, it runs over 15 minutes.

Penalty	Training/Validation Score	Training Time	Test Score
L1	90.56%	58.96	Not Calculated
L2	91.02%	33.68	91.51%

For penalty selection in logistic regression function, I show my model performs better both in accuracy and training time with L2 norm than with L1 norm. With L2 norm, my model has an accuracy 91.51% in testing and with training time around 33.68 seconds.

Solver	Training/Validation Score	Training Time	Test Score
lbfgs	90.88%	23.48	91.09%
liblinear	91.02%	33.68	91.51%

For solver selection in logistic regression function, I show my model is comparable to the best multi-class classification using logistic regression with “lbfgs” solver. My model has 0.42% higher accuracy in testing than that of for “lbfgs” solver, but the training time for my model is about 10 seconds longer.

9. Interpretation

From this project, I learnt the importance of data preprocessing, especially for a relative large dataset. With right preprocessing method, it can reduce the training time and memory requirements. In my model, normalization helps me to reduce the training time significantly.

I also learnt the important of using right regularization methods to avoid overfitting and reduce training time. I find in this project that choosing a suitable regularizer—L2, can reduce the training time for about 40%. Regularizer should be chosen basing on the properties of dataset. In my model, I think it is because L2 is more computational efficient with analytical solution.

Selecting right algorithms/solvers is also important in avoiding underfitting and being more computation efficient. With “lbfgs” and “sag” solvers, their accuracy are approximate the same, but the training time are much different. “sag” solver leads to a 79.92 seconds training time, but “lbfgs” solver has only 23.48 seconds training time.

10. Summary and Conclusion

In my project, I design a logistic regression model using python sklearn library to predict labels for handwritten digits using MNIST dataset. My model is constructed by using L2 normalization for data preprocessing. It consists 10 logistic regressions. Each of them is trained and used to classify a particular kind of digits. With L2 regularizer and "liblinear" solver for logistic regression in training, my model achieved 91.51% accuracy with 33.68 seconds training time.

11. Reference

[1] Wikipedia, Wikimedia Foundation, (2017). *"MNIST Database."*, en.wikipedia.org/wiki/MNIST_database.

[2]"GradientBased Learning Applied to Document Recognition." *Intelligent Signal Processing*, 2009, doi:10.1109/9780470544976.ch9.

[3] Yann.lecun.com. (2017). *MNIST Demos on Yann LeCun's website*. [online] Available at: http://yann.lecun.com/exdb/lenet/ [Accessed 5 Dec. 2017].

[4] Scikit-learn.org. (2017). *sklearn.linear_model.LogisticRegression — scikit-learn 0.19.1 documentation*. [online] Available at: http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html#sklearn.linear_model.LogisticRegression.predict [Accessed 5 Dec. 2017].

[5] Scikit-learn.org. (2017). *1.1. Generalized Linear Models — scikit-learn 0.19.1 documentation*. [online] Available at: http://scikit-learn.org/stable/modules/linear_model.html#logistic-regression [Accessed 5 Dec. 2017].

[6] Chioka.in. (2017). *Differences between L1 and L2 as Loss Function and Regularization*. [online] Available at: http://www.chioka.in/differences-between-l1-and-l2-as-loss-function-and-regularization/ [Accessed 5 Dec. 2017].

[7] Towards Data Science. (2017). *Logistic Regression using Python (scikit-learn) – Towards Data Science*. [online] Available at: https://towardsdatascience.com/logistic-regression-using-python-sklearn-numpy-mnist-handwriting-recognition-matplotlib-a6b31e2b166a [Accessed 5 Dec. 2017].

[8] GitHub. (2017). *mGalarnyk/Python_Tutorials*. [online] Available at: https://github.com/mGalarnyk/Python_Tutorials/blob/master/Sklearn/Logistic_Regression/LogisticRegression_MNIST_Codementor.ipynb [Accessed 5 Dec. 2017].

[9] Scikit-learn.org. (2017). *4.3. Preprocessing data — scikit-learn 0.19.1 documentation*. [online] Available at: http://scikit-learn.org/stable/modules/preprocessing.html#normalization [Accessed 5 Dec. 2017].

[10] Yann.lecun.com. (2017). *MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges*. [online] Available at: http://yann.lecun.com/exdb/mnist/ [Accessed 5 Dec. 2017].