MNIST Image Classification

Mill Liao

mill.ct.liao@gmail.com
linkedin.com/in/mill-liao

Chapter 1: Introduction

The purpose of this study is to build various machine learning classifiers in order to classify grayscale images of the size 28x28 of different types of fashions for MNIST fashion dataset. The dataset of MNIST (Modified National Institute of Standards and Technology) is the classic handwritten images which are highly suitable to be implemented by classification algorithms (Jhanji 2019). The image recognition is a subfield of computer vision which is defined as a field of study that seeks to develop techniques to let computer see and understand the content of digital images like videos and photographs (Brownlee 2019). Although it is a trivial problem for human beings, it is a complex and hard issue for the computer. This report is to implement classification algorithms to help computers are able to distinguish images that are fairly important for developing future Artificial Intelligence technology to empower them to see.

Chapter 2: Methods

In the machine learning field, the data prepossessing is the important phase to make the data more analysable. The dimension reduction is one of the vital data prepossessing skills, the purpose of the dimension reduction is to compress the data attributes so as to avoid overfitting. In this chapter, I am going to introduce two-dimension reduction skills - Singular Value Decomposition (SVD) and Principal Component Analysis (PCA), and explain the detail later on.

In terms of algorithms, the popular methods of solving classification problems is Logistic Regression, Nearest Neighbours, and Naïve Bayes. The difference among these algorithms is the perspectives between Frequentist and Bayesian. Nearest Neighbours and Logistic Regression belong to Frequentist treatment, and Naïve Bayes is Bayesian treatment (Dudley 2017). The idea of the difference between Frequentist and Bayesian is similar to the difference between generative and discriminative model but slightly different. Nearest Neighbours and Logistic Regression are also categorised to discriminative model, and Naïve Bayes is considered as generative model. The goal of the generative model is to find the estimated probability and its performance is assessed by the likelihood. On the other hand, the goal of the discriminative model is to find the classification rule and its performance is assessed by the misclassification rule. Finally, logistic Regression is a parametric model, and Nearest Neighbours is a nonparametric model. Consequently, we can use a formula to describe the logistic Regression, but cannot use a formula to describe Nearest Neighbours. I am going to detail these algorithms below.

2.1 Singular Value Decomposition (SVD)

In linear algebra, the SVD is one of the most important matrix factorisations. The SVD provides a numerically stable matrix decomposition for any real matrix (Brunton 2019). It is also the underlying algorithm of PCA. The SVD can be expressed as:

$$X = U\Sigma V^T \qquad \text{Equation (2.1)}$$

X is the matrix we are going to decompose; it can be any real matrix. Its dimension is (m, n). U is a (m, r) column-orthonormal matrix. V is a (n, r) column-orthonormal matrix. Σ is a diagonal (r, r) matrix (Tran 2020). After the decomposition, the equation can be rewritten as:

$$X = \sum_{i=1}^{r} \lambda_i u_i \times {v_i}^t \qquad \text{Equation (2.2)}$$

The above is called spectral representation. λ_i represents the eigenvalue from Σ; u_i represents the eigenvector from U; v_i represents the eigenvector from V. The eigenvalues have been sorted after the decomposition, and that is $\lambda_1 > \lambda_2 > \cdots > \lambda_r$.

The sum squared error (SSE) function is taken as the loss function to measure how many information remains from the original data to reconstructed data. It can be expressed as:

$$SSE = \sum_{i=1}^{r} (x - \hat{x})^2 \qquad \text{Equation (2.3)}$$

x means the original data, and $\hat{x}$ means the reconstructed data. Moreover, I also introduce the compression ratio as the assessment for the performance of SVD. It can be expressed as:

$$Compression\ ratio = \frac{mk + k + nk}{mn} \qquad \text{Equation (2.4)}$$

The meaning of m is the column of the original data; n represents the row of it. The selected component is k, and it is the key part of the parameter.

2.2 Principal Component Analysis (PCA)

The idea of PCA is to reduce the dimensionality of a dataset consisting of a large number of interrelated variables while retaining as much as possible of the variation present in the data set (Dubey 2018). For example, acquiring DNA data is always expensive and the characteristics of DNA data includes a huge number of features. The researchers cannot collect sufficient data, so the analysis with huge features and small size data will result in overfitting. As a result, PCA can be used to avoid overfitting. Moreover, it is a technique to visualise high dimensional data via compressing it to 2 or 3 dimensions. The mathematics behind PCA is to compute its covariance matrix of the dataset and then compute its eigenvectors and eigenvalues via matrix factorisations to reconstruct the matrix. The covariance matrix can be expressed as:

$$X_C = \frac{1}{m} X^T X = \begin{bmatrix} VAR(A) & COV(A,B) \\ COV(A,B) & VAR(B) \end{bmatrix} \qquad \text{Equation (2.5)}$$

X_C is original covariance matrix, and X is the original dataset. $VAR(A)$ is the variance of feature A; $VAR(B)$ is the variance of feature B; $COV(A,B)$ is the covariance between feature A and B. The purpose of the computation is to try to make $VAR(A)$ as large as possible as the first component. Assume Y is the data after the dimension reduction, the mapping from data X to Y satisfies $Y = XP$, and then:

$$Y_C = \frac{1}{m} Y^T Y$$

$$= \frac{1}{m}(XP)^T XP$$
$$= \frac{1}{m}P^T X^T XP$$
$$= P^T(\frac{1}{m}X^T X)P$$
$$= P^T X_C P \qquad \text{Equation (2.6)}$$

Thus, we only have to solve P that lets $Y_C = P^T X_C P$ to satisfy the condition of diagonal matrix. X_C is actually a symmetry matrix.

2.3 K-Nearest Neighbours (KNN)

The K-nearest neighbours (KNN) algorithm is a supervised machine learning algorithm that can be used to solve classification problems (Harrison 2018). Here are the steps of KNN:

- Step 1: Initialise K to the chosen number of neighbours
- Step 2: Calculate the distance between the query example and the current example from the data.
- Step 3: Add the Euclidean distance and the index of the data to the ordered collection.
- Step 4: Sort the ordered collection of distances and indices from smallest to largest (in ascending order) by the distances.
- Step 5: Pick the label with the most frequent labels in the K-nearest neighbours.

Choosing the value of K is super important for KNN. If K is too small, the model will incline overfitting. If K is too large, the model will incline underfitting. KNN is a good tool for lazy learners, so it does not build models explicitly (Tran 2020).

2.4 Gaussian Naïve Bayes

The Naïve Bayes classifier is a generative model based on Bayes theorem. Unlike the discriminative model that directly finds the hypothesis, it is to find prior and conditional probabilities and then compute the posterior probability from multiplying them. By comparing the posterior probabilities of different classes, it can predict the class of a specific data. Here's the formula of Bayes classifiers:

$$P(C|A_1A_2 \dots A_n) = \frac{P(A_1A_2 \dots A_n|C)P(C)}{P(A_1A_2 \dots A_n)} \qquad \text{Equation (2.7)}$$

$P(C|A_1A_2 \dots A_n)$ is the posterior probability; $P(A_1A_2 \dots A_n|C)$ is the conditional probability; $P(C)$ is the prior probability. We try to maximise the posterior probability to determine the class. Moreover, we assume the independence among attributes A_i that makes the model simpler. So, the conditional probability can be rewritten as:

$$P(A_1A_2 \dots A_n|C) = P(A_1|C)P(A_2|C) \dots P(A_i|C) \qquad \text{Equation (2.8)}$$

However, the implementation of Naïve Bayes becomes more difficult in dealing with continuous data. As a result, we need to assume the distribution of data is Gaussian distribution. So, we should replace conditional probability as the probability density function (PDF) of Gaussian distribution:

$$p(x|C_k) = \prod_{j=1}^{d} \frac{1}{\sqrt{2\pi{\sigma_{j,k}}^2}} exp\left(-\frac{1}{2{\sigma_{j,k}}^2}\left(x_j - \mu_{j,k}\right)^2\right) \qquad \text{Equation (2.9)}$$

2.5 Multi Logistic Regression

The logistic regression is a popular algorithm to deal with classification problems. It still uses linear regression as the decision boundary to distinguish the class of the data. However, it implements the sigmoid function to let the predictor variable to the categorisation. As a result, the function can be expressed as:

$$h_\theta(x) = g(\theta^T x)$$
$$g(z) = \frac{1}{1+e^{-z}}$$
$$\rightarrow h_\theta(x) = \frac{1}{1+e^{-\theta^T x}} \qquad \text{Equation (2.10)}$$

$h_\theta(x)$ is estimated probability; θ is weight of the function; x is input variable; y is class. Moreover, we should set a threshold. For example, if $h_\theta(x)$ is greater than the threshold, it should be assigned as 1 and vice versa; if $h_\theta(x)$ is smaller than the threshold, it should be assigned as 0. In logistic regression, $0 \leq h_\theta(x) \leq 1$, but y = 0 or 1.

If $h_\theta(x) \geq 0.5$, $\theta^T x \geq 0$, predict y=1.
If $h_\theta(x) < 0.5$, $\theta^T x < 0$, predict y=0.

The cost function of the logistic regression is the sign function. It can be expressed as:

$$Cost(h_\theta(x), y) = \begin{cases} -log(h_\theta(x)), & if\ y = 1 \\ -log(1 - h_\theta(x)), & if\ y = 0 \end{cases} \qquad \text{Equation (2.11)}$$

This function also can be rewritten as:

$$Cost(h_\theta(x), y) = -ylog\big(h_\theta(x)\big) - (1-y)log\big(1 - h_\theta(x)\big) \qquad \text{Equation (2.12)}$$

The loss function of the logistic regression is the mean of cost function, and it can be expressed as:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} Cost(h_\theta(x), y) \qquad \text{Equation (2.13)}$$

In the optimisation problem, gradient descent is an efficient technique while finding the maximum and the minimum. I am going to introduce gradient descent, and it can be written as:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \qquad \text{Equation (2.14)}$$

We can input the loss function $J(\theta)$ into the gradient descent, so the equation can be rewritten as:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} \left(\frac{1}{m} \sum_{i=1}^{m} Cost(h_\theta(x), y) \right)$$

$$\rightarrow \theta_j := \theta_j - \alpha \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)} \right) x_j^{(i)} \qquad \text{Equation (2.15)}$$

The classic logistic regression only can solve the binary class problem; as a result, I have to introduce the multi-class logistic regression in order to settle down the multiple classes. The classifier will be rewritten as:

$$h_c(x) = \frac{exp(\theta_c^T x)}{\sum_{c=1}^{C} exp(\theta_c^T x)} \qquad \text{Equation (2.16)}$$

Chapter 3: Experiments and Results

The size of the training data of MNIST fashion is totally 30000 examples, and the size of the grayscale image is 28x28. Moreover, the size of the testing data of it is overall 5000 examples. There are 10 classes in total, and sequentially label 0 represents T-shirt; 1 represents trouser; label 2 represents pullover; label 3 represents Dress; label 4 represents coat; label 5 represents sandal; label 6 represents shirt; label 7 represents sneaker; label 8 represents bag; label 9 represents ankle boot.

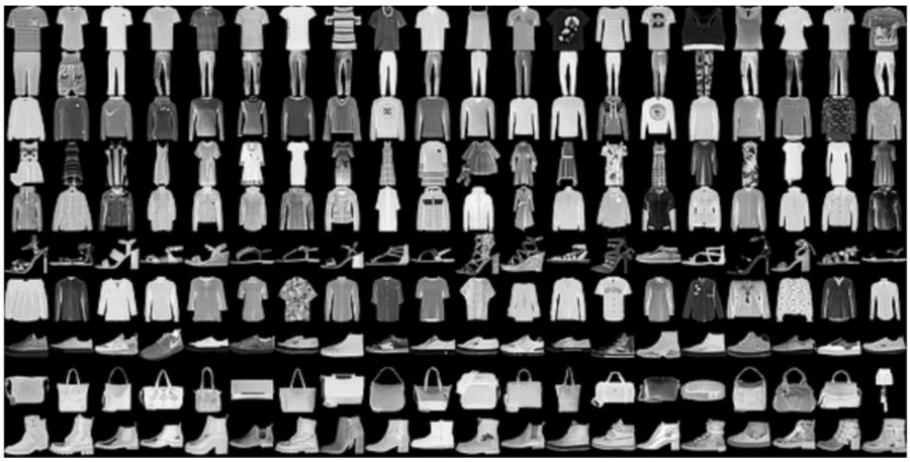

Figure 3.1: MNIST fashion

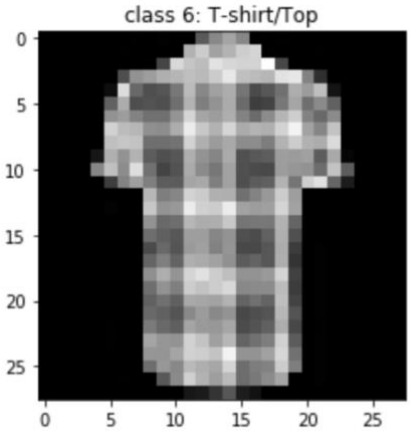

Figure 3.2: The example of the grayscale image from MNIST fashion

The size of the data is basically three-dimension; as a result, I have to reshape its size from 30000x784 to 30000x28x28 in order to visualise the grayscale image. However, the original size of the data is analytically more suitable due to the two dimensions. Another reason is that linear algebra applied in Naïve Bayes and Logistic Regression we learnt so far in COMP5318 focuses on the two dimensions (I guess if I understand the high dimension, it maybe can be directly dealt with this issue).

As I mentioned above, the dimension of the MNIST fashion data is 30000x784, so it has 784 attributes and the size of the data is 30000 examples in total. The number of the data size is fairly greater than the that of its attributes, so theoretically even implementing whole attributes the overfitting would not occur. Nevertheless, the implementation of the dimension reduction can bring the benefits like the efficiency of running algorithms, so it is still worthwhile to use dimension reduction.

The first implemented dimension reduction technique is SVD. It tries to find the matrix of U, Σ, and V in equation (2.1). The performance is assessed by the equation (2.3) and (2.4). The eigenvalues are in the matrix of Σ, so I have to choose how many components I'd like to retain. For example, the figure 3 shows the original image and compressed image generated by selecting 10 components, and its SSE is 0.24 and compression ratio is 72.70%. In Python, I use *numpy.linalg.svd* to achieve that.

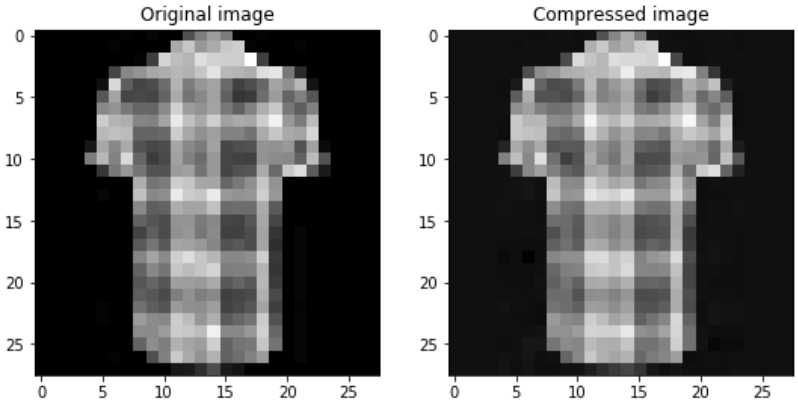

Figure 3.3: The comparison between original image and reconstructed image after SVD (components=10)

However, I implement use *numpy.linalg.svd* to whole training dataset at the beginning. It pretty consumes time, and it is nearly one hour to run it. As a result, I decide to split the training dataset and implement use *numpy.linalg.svd()* in turn. That approach tremendously saves my time.

Moreover, the other interesting point is that there are many zeros in the training dataset, and I guess the data has already been pre-processed. That can explain why the running time of inputting no matter original training data or the data after SVD into algorithms is not much different.

The second implemented dimension reduction technique is PCA, and it tries to use equation (2.5) to find the covariance matrix of the training dataset. After that, it uses equation (2.6) to do the factorisation to get the eigenvectors and eigenvalues. The purpose of PCA is to find the so-called principal component and maximise it. In practice, I take *numpy.cov()* to get the covariance matrix and then *numpy.linalg.eig()* to acquire eigenvectors and eigenvalues. The figure below illustrates 1st and 2nd component from the data after PCA.

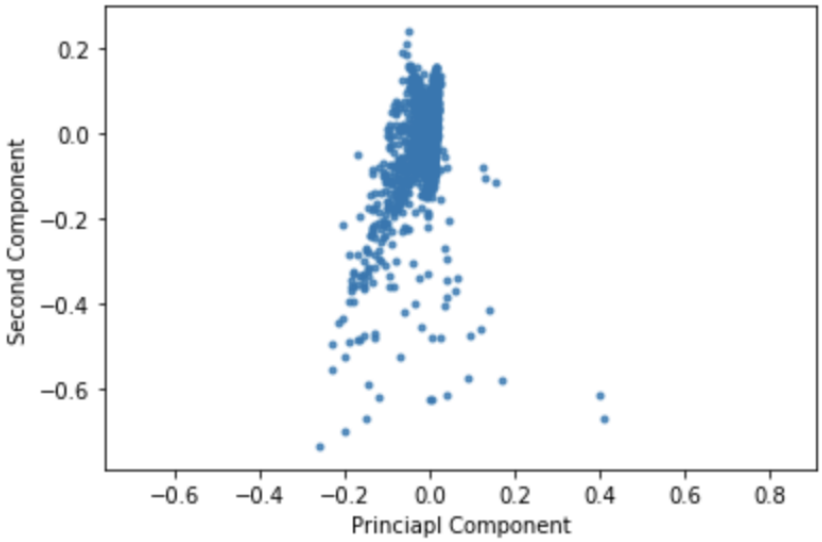

Figure 3.4: Visualisation of the principal and second component from training data

In terms of implementing algorithms, I take two strategies that the first one is to input data after dimension reduction and the second one is to input the original data. Next, it can be compared their performance and then can choose the one with better performance.

KNN is a nonparametric discriminative approach which means that we do not need a mathematical equation to describe it. As a result, it is easier to implement it than other algorithms in practice. The first thing to conduct KNN algorithm is to get the Euclidean distance between observed and known data. So, I use *numpy.tile()* to copy test data in order to find the difference between train and test data in Python. After that, I write a code of Euclidean distance and then input the difference above into the Euclidean distance. So, I can acquire the Euclidean distance of each test data from train data. Next, I take *argsort()* to sort the index based on the Euclidean distance of each test, so I can get the training points from the nearest to the farest. I can decide the value k to find the range of the nearest neighbours and then from these points to vote which label the test data should be labelled.

In the beginning, I use the data after PCA and SVD, and I get very poor result - approximately 58.72% in KNN. Consequently, I decide to use the original data to input them into KNN, and the result 82.65% (K=5) is far better than implementing the dimension reduction. The reason why the performance becomes worse is because it lost partial information after dimension reduction. Moreover, the dimension reduction in my computer which is MacBook Pro and the macOS version is 10.13.6 does not save the time while implementing the algorithm. On the contrary, it would slow down the progress via doing dimension reduction. That causes me that I would not use data after PCA or SVD in Logistic Regression and Naïve Bayes.

Figure 3.5: The specialisation of my MacBook Pro

In addition, there is no significant difference of choosing value K while implementing KNN in my case. The best accuracy is 83.20% which K is 7 and 9. The worst accuracy appears in K = 15.

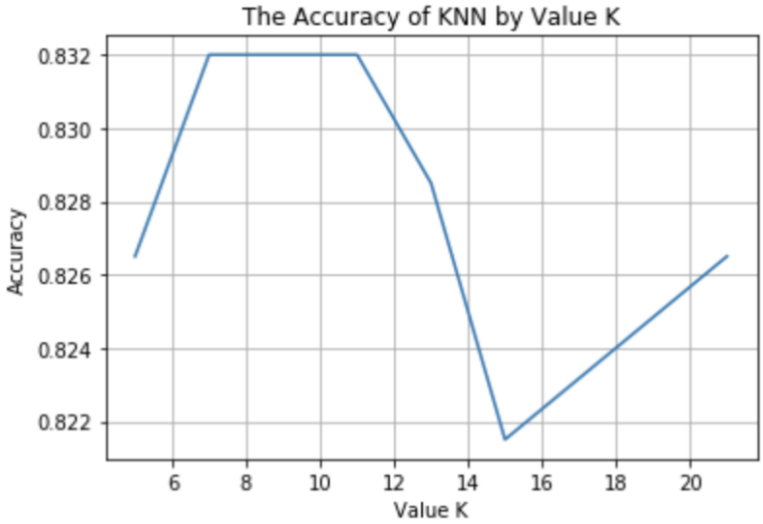

Figure 3.6: The accuracy of KNN by value K

The first stage to implement Gaussian Naïve Bayes algorithm is to write Python code of the PDF of the Gaussian distribution. Next, it is to compute the prior probability of each class and the means and variances across class and attributes in order to use PDF of Gaussian distribution to get the conditional probability. However, one issue occurring while computing the PDF is because some variances which are 0 will result in the termination of programming, so it should add a negligible value (10^{-14}) in order to prevent the termination. While implementing the algorithm, I use the equation (2.7), (2.8), and (2.9) to compute the posterior

probability. As a result, I get the data type dictionary with the keys from class 0 to class 9 and their posterior probabilities, and it should be labelled with the biggest probability.

The time of programming in Naïve Bayes is the longest among these three algorithms. I reckon the reason why it is time-consuming is because computing the PDF of Gaussian distribution. There are 10 labels and 784 features that means the computer has to compute 10x784 times of the PDF of Gaussian distribution. In this situation, the dimension reduction can accelerate the progress of using Naïve Bayes.

The accuracy of train data in Naïve Bayes is only 57.10% and 56.60% for test data. It is much poorer than KNN, and the reason is because the mathematics in Naïve bayes is too simple to naive. Naïve Bayes assumes the distributions of the attributes are independent and identically distributed random variables. Although it simplifies the mathematics, the overall accuracy becomes worse.

The first thing to do logistic regression is to add bias for the training data, so I use *numpy.hstack((np.ones((x.shape[0],1)), x))* to achieve that. Secondly, I use equation (2.12), (2.13), (2.14), and (2.15) write the loss function and gradient descent for logistic regression. The next task is to write the function which can make the class of the data to the binary which is 1 and 0 because logistic regression only can deal with binary problem.

The result is out of my expectation and the accuracy of each label is pretty high no matter analysing train or test data. Take train data as an example, the accuracy of predicting correct 0 is 95.83%; 89.63% for predicting label 1; 86.72% for predicting label 2; 84.93% for predicting label 3; 89.94% for predicting label 4; 89.80% for predicting label 5; 89.92% for predicting label 6; 89.01% for predicting label 7; 89.16% for predicting label 8; 86.32% for predicting label 9. I reckon the reason is because there is the huge disparity between two labels while predicting individual variable and it is extremely easy to predict for logistic regression. While integrating the results from individual, it will result in the poor performance – 71.32% for train data.

Label	Train Data	Test Data
0	95.83%	95.95%
1	89.63%	90.35%
2	86.72%	86.15%
3	84.93%	84.25%
4	89.94%	88.75%
5	89.80%	90.35%
6	89.92%	90.50%
7	89.01%	89.50%
8	89.16%	87.65%
9	86.32%	86.50%

Table 1: Accuracy of label 0 to 9 for train and test data in logistic regression

Chapter 4: Conclusion

In conclusion, KNN has the best performance whose accuracy is 83.32% among these three algorithms. Not surprisingly, Naïve Bayes gets the poorest result due to its mathematical structure.

The future work of Naïve Bayes can be re-assumed its mathematical assumptions - does it make sense to assume distribution of the data for the pixels really follow Gaussian distribution, or are these pixels data really independent?

For the logistic regression, the implementation of the code maybe has some issue. The future work of this part is to examine the code and its process.

In the future, I should learn the knowledge about the image recognition and computer vision that will benefit me while dealing with the image data. Moreover, other algorithms like SVM or neural network should be introduced to tackle this problem.

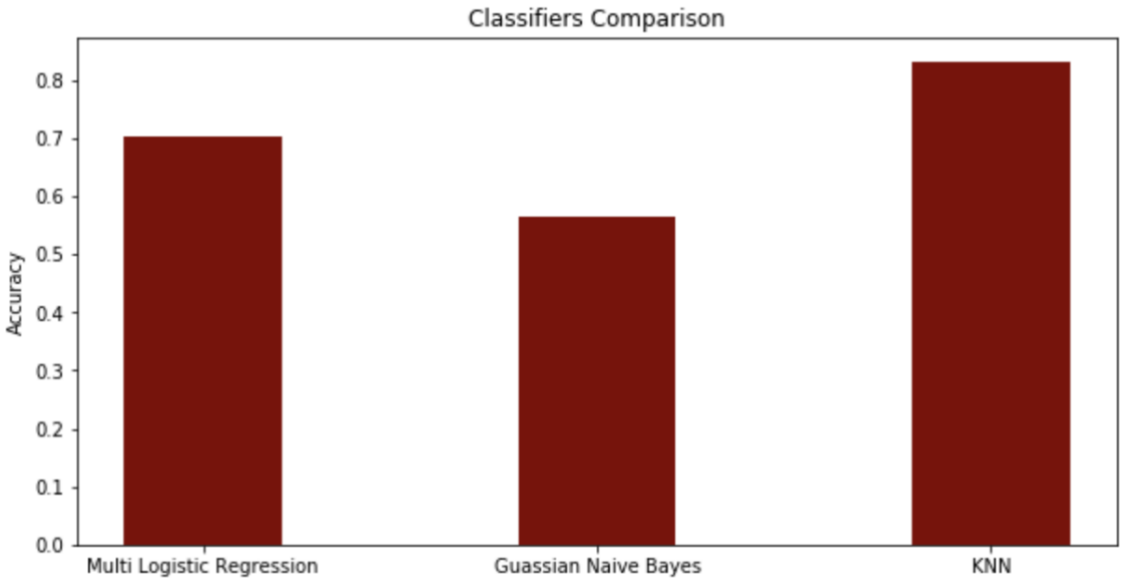

Figure 4.1: The comparison of the accuracy for test data in Logistic Regression, Naïve Bayes, and KNN

Reference

Jhanji, D. (2019). *Learn Computer Vision Fundamentals with the famous MNIST data*. Retrieved from https://medium.com/analytics-vidhya/learn-computer-vision-fundamentals-with-the-famous-mnist-data-2bb63ec8e3f0

Brownlee, J. (2019). *A Gentle Introduction to Computer Vision* Retrieved from https://machinelearningmastery.com/what-is-computer-vision/

Dudley, M. (2017). *Mike Dudley Medium Blog* Retrieved from https://medium.com/@mike.dudkey78/it-remember-me-a-table-i-wrote-for-my-boss-but-i-used-a-different-approach-60747aa37122

Brunton, S. L. (2019). *Singular Value Decomposition (SVD)* Retrieved from https://www.researchgate.net/publication/331230334_Singular_Value_Decomposition_SVD

Tran, N. H. (2020). *Basic Matrix Analysis and Singular Value Decomposition*

Dubey, A. (2018). *The Mathematics Behind Principal Component Analysis* Retrieved from https://towardsdatascience.com/the-mathematics-behind-principal-component-analysis-fff2d7f4b643

Harrison, O. (2018). *Machine Learning Basics with the K-Nearest Neighbors Algorithm* Retrieved from https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761

Tran, N. H. (2020). *Basics of Classification and ROC curves*