Machine Learning Engineer Nanodegree Capstone Project (Hand Digit Recognizer via Neural Network)

Name: Taihwa Song
March 21st, 2020

1. Definition

1.1 Project Overview

The traditional way of documentations was typically done on a piece of paper. This is extremely cumbersome because it is not only inefficient to make extra copies but also difficult to search words from text. With the widespread use of personal computers and smartphones in the 21st century, many documents are now digitized. This Modern way of storing documents made it easy for researchers to share findings across the world with a single click of a mouse.

Most recent documents can be easily searchable because they are already digitized to begin with; however, the legacy/traditional documents are still kept on a piece of paper. This makes it difficult for modern people to find relevant traditional documents and many efforts were put into turning legacy papers into digital documents. One method of turning legacy texts into digitized texts is leveraging reCAPTCHA [1]. This technology is an advanced version of CAPTCHA[2] that utilizes some portion of text-based CAPTCHA tests to translate an image of a hand-written word into a digital word. Another technique is to utilize machine learning to translate a set of documents into a digital copy. This technique is called handwriting recognition.

The recurrent neural network and deep feedforward neural networks (researched by a group of researchers from Jurgen Schmidhuber at the Swiss AI Lab IDSIA) were proven to be effective against the handwriting recognition problems [3].

1.2. Problem Statement

The primary goal of this project is to predict digits from handwritten images. Recognizing digits is the most fundamental approach of handwriting recognition. This is a supervised classification problem where the model is expected to predict a number ranging from 0 to 9 by looking at an image with 28 pixels in height and 28 pixels in width, for a total of 784 pixels in total. The

performance of the model can be computed by comparing the predicted number with the actual number.

1.3 Metrics

Accuracy score will be used as an evaluation metrics. Accuracy score is calculated by finding the ratio between the sum of true positives and true negatives, and the total count of dataset. In the case of multi-class classification problems like this project, true positives and the true negative are essentially the same; therefore, the accuracy will be calculated by finding the ratio between the sum of true positive positive and the total count of the dataset. The outcome of the accuracy is a float number between 0 and 1 where the number closer to 1 shows a better performance.

$$\text{Accuracy} = \frac{\text{True Positive + True Negative}}{\text{True Positive + True Negative + False Positive + False Negative}} \text{, for binary classification}$$

$$= \frac{\text{Number of samples correctly identified class}}{\text{Total number of samples}} \text{, for multi-class classificaiton}$$

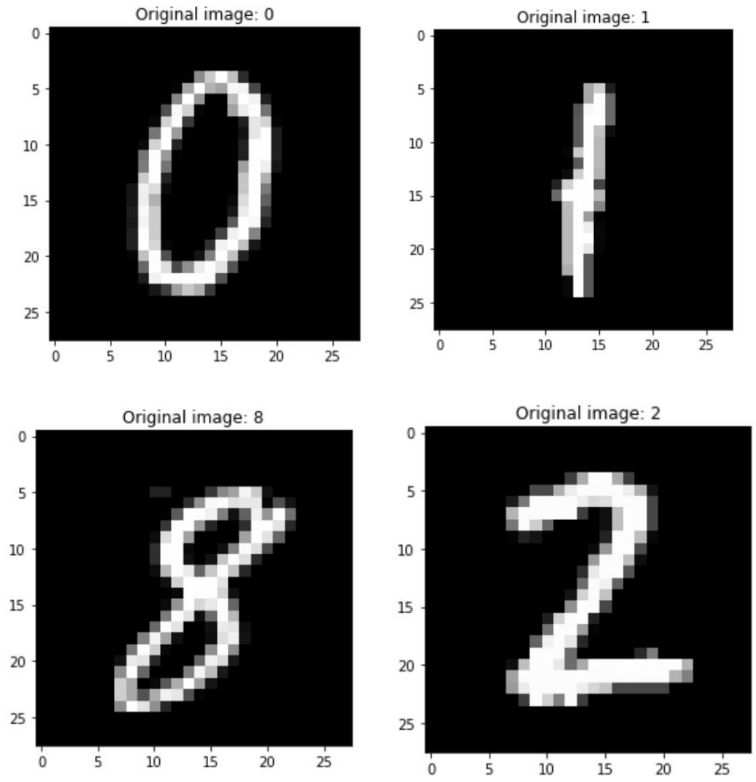
2. Analyze

2.1 Data Exploration

The origin of the data comes from MNIST and the data was downloaded from Kaggle, specifically “Digit Recognizer” competition challenge [4]. The data files train.csv contains 42,001 lines (where the first line is header and the rest of numerical data). Each line contains 785 columns. The first column is the label: the digit that was drawn by the user. The rest of the columns contain the pixel-values of the associated images. Each pixel column in the training set has a name like pixel_x, where x is an integer between 0 and 256, inclusive. To locate this pixel on the image, suppose that we have decomposed x as x = i * 28 + j, where i and j are integers between 0 and 27, inclusive. Then pixel_x is located on row i and column j of a 28 x 28 matrix, (indexing by zero).

2.2 Data Visualization

Let’s explore a few images that represent different digits. The images contain 28 by 28 pixels as represented in the figures below. The images are gray scaled with each pixel value ranging from 0 to 255, where 0 represents black and 255 represents white.



2.3 Algorithms and Techniques

2.3.1 Data Pre-processing techniques

2.3.1.1 Image resizing

Depending on the person who wrote the digit, the size of the digit may vary. Every image in the data will be resized to maximally use all pixels. This is done by cropping empty rows and columns at the beginning and the ending of each image. The cropped image is then resized to its original size, 28 by 28. The resizing of an image is accomplished using the scikit-image library.

2.3.1.2 Principal Component Analysis

Given that the dimensionality of the data is high (784), it is often recommended to reduce the dimensionality through PCA (Principal component analysis). PCA is a statistical method to translate a high dimensional data into smaller dimensional data while maintaining the high variance. This is a useful technique for this problem because it can remove pixels that are rarely used. For example, the first pixel (located at the left top) is often blank because all digit images are already centered and rarely any digits require the first pixel to be filled in. PCA will recognize that the pixel is rarely used (such that it has low variance across different digits) and it will remove the pixel which will result in dimensionality reduction.

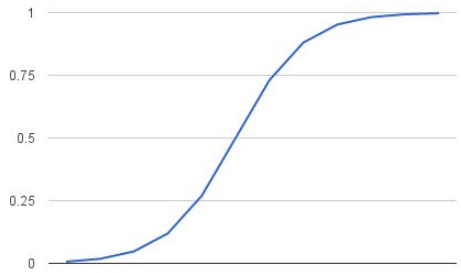
2.3.2 Training Algorithms

A number of different learning algorithms will be used to testify the result: Logistic Regression, K Nearest Neighbours, Neural Network. Logistic Regression is one of the simplest training algorithms that could be used to justify the baseline of the model accuracy. Neural network is another algorithm that can be used to explain the deeper layer of patterns.

2.3.2.1 Multinomial Logistic Regression

Multinomial logistic regression is a supervisord learning algorithm that is estimating the parameters of a logistic model. It is an extension of a simple logistic regression, a model that is capable of outputting binary decisions; however, the hand-written digit problem is a multi-class classification problem so the simple logistic regression cannot be applied here. Multinomial logistic regression, on the other hand, is capable of answering nominal (also known as categories, meaning that the outcome can be more than two) questions.

The baseline of multinomial logistic regression is very similar to logistic regression. The simple logistic regression uses logistic function. It's an S-shaped curve that can take any real-valued number and map it into a value between 0 and 1. The below image shows an example of logistic function



Example of logistic function

The input values are combined linearly using weights or coefficient values to predict an output value. A key difference from linear regression is that the output value being modeled is a binary value (0 or 1) rather than a numeric value. The below is an example of logistic regression equations:

$$y = \frac{e^{(b0 + b1*x)}}{1 + e^{(b0 + b1*x)}}$$

Training of a logistic regression model involves optimizing coefficient and weights values (b0 and b1 in the above example) such that the predicted output (y in the above example) is as close as possible to the actual output.

In the case of multinomial logistic regression, a logistic regression model is built per class. The below figure demonstrates an example of logistic regression models built per class. During prediction, the multinomial regression model performs tests on every regression model and finds the class that best suits.

$$y_1 = \frac{e^{(b0_1 + b1_1*x)}}{1 + e^{(b0_1 + b1_1*x)}}$$

$$y_2 = \frac{e^{(b0_2 + b1_2*x)}}{1 + e^{(b0_2 + b1_2*x)}}$$

$$y_3 = \frac{e^{(b0_3 + b1_3*x)}}{1 + e^{(b0_3 + b1_3*x)}}$$

$$\vdots$$

Figure: example of multinomial logistic regression with n number of classes

2.3.2.2 K Nearest Neighbours Classifier

K nearest neighbor (KNN) is a nonparametric method used for classification and regression [6]. In any case, the algorithm gets an input and finds the N closest training data and derives the outcome based on the closest training data points. The below is an example of k nearest neighbours classifier algorithm.

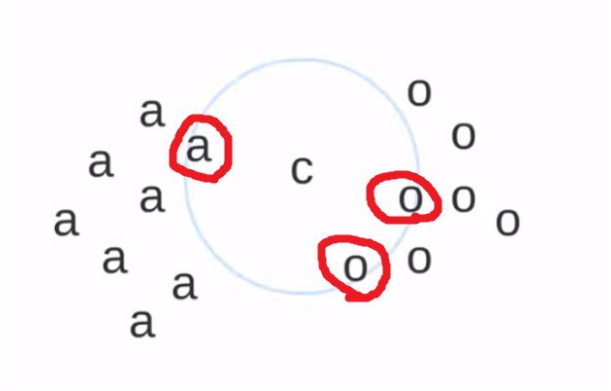


Figure: Example of KNN

In the example above, ‘a’ and ‘o’ are classes and ‘c’ is an input value. The algorithm finds N nearest neighbors (n is 3 in this example). Since the occurrence of ‘o’ outnumbers ‘a’, the input value ‘c’ is classified as ‘o’.

2.3.2.3 Neural Network Classifier

Neural network is a network of circuits of neurons composed of artificial neurons and nodes [7]. The connections of the biological neuron are modeled as weights. A positive weight reflects an excitatory connection, while negative values mean inhibitory connections. All inputs are modified by a weight and summed. Neural networks are non-linear statistical data modeling which can be used to model complex relationships between input and output or to find patterns in data. This makes it a good use case of the hand-written digit problem because the digits (input data) generated by humans such that it contains noises which may be explained by hidden patterns.

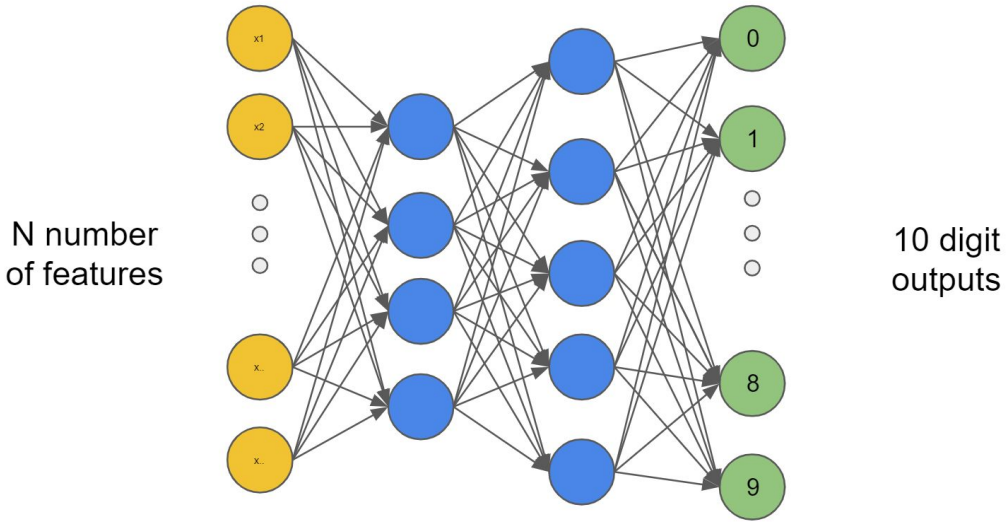


Figure: Example of Neural Network

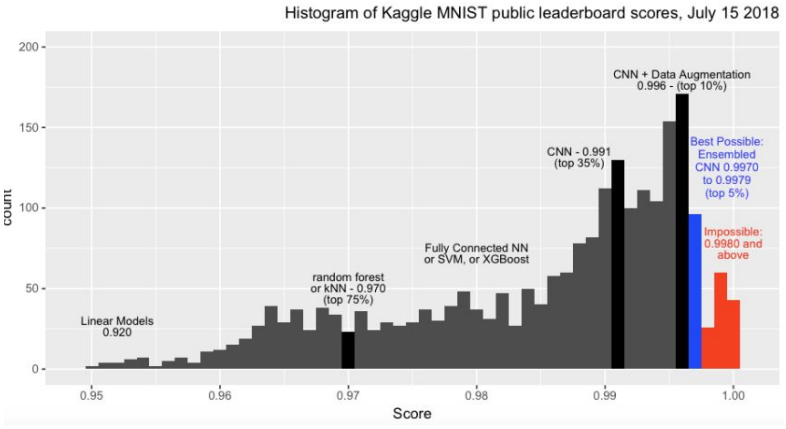
The above figure is an example of a neural network model that would be used for the hand-written digit problem. The input value is going to be the pixels of an image which is defined as yellow circles. The neurons are represented as blue circles in the middle. The output nodes

are colored as green on the right. Since there are 10 digits (numbers ranging from 0 to 9), 10 output nodes are placed on the right.

2.4 Benchmarking

The histogram of Kaggle MNIST public leaderboard scores [4] will be used for the benchmarking model. The digit handwriting recognition problem is a publically known competition and many teams participated to share their model performances. The performance outcomes from the community is a good baseline to compare for this project. The result of the benchmark model is the accuracy score of the model. The following is the accuracy score of benchmarking models for each training algorithm:

- Linear regression: 0.92 (see 'Linear Model' benchmark score below)
- KNN: 0.97 (see 'KNN' model benchmark score below)
- Neural Network: 0.98 (See 'Fully Connected NN' benchmark score below)



3. Implementation

3.1. Data Preprocessing

3.1.1 Image resizing

The algorithm goes over an image by an image and removes rows/columns if the entire row/columns is empty. Once the empty rows/columns are removed, the dimension of the image is now reduced and the dimension of images will be different from one to another. In order to make the image dimensions consistent, all images are rescaled back to 28 pixels by 28 pixels. The rescaling of the image is accomplished using the scikit-image library.

```
def resize_to_fit(df_images):
    np_images = df_images.to_numpy()
    np_images_resized = []
    for np_image in np_images:
        # conver to arrays of image rows
        image = [np_image[x:x+28] for x in range(0, len(np_image),28)]
        df_image = pd.DataFrame(image)
        # drop rows that are all zeros
        df_image = df_image[(df_image.T != 0).any()]
        # drop columns that are all zeros
        df_image = df_image.loc[:, (df_image != 0).any(axis=0)]

        # resize image
        np_image_resized = resize(df_image.to_numpy(), (28, 28) ,anti_aliasing=True)
        # add to final np array
        np_images_resized.append(np.concatenate(np_image_resized, axis=None))

    return pd.DataFrame(np_images_resized)
```

The below figures demonstrate the before and after resizing.

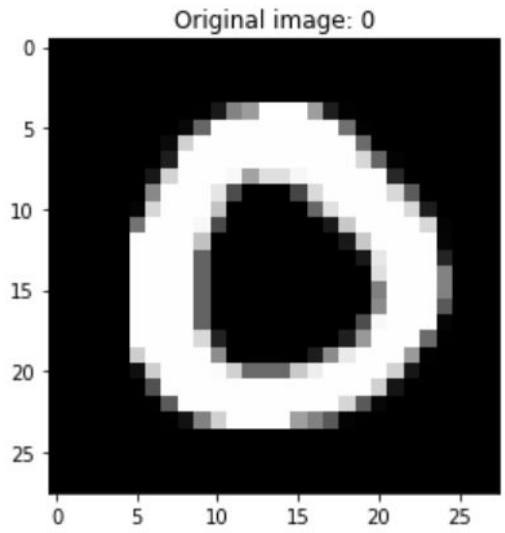


Image before resizing

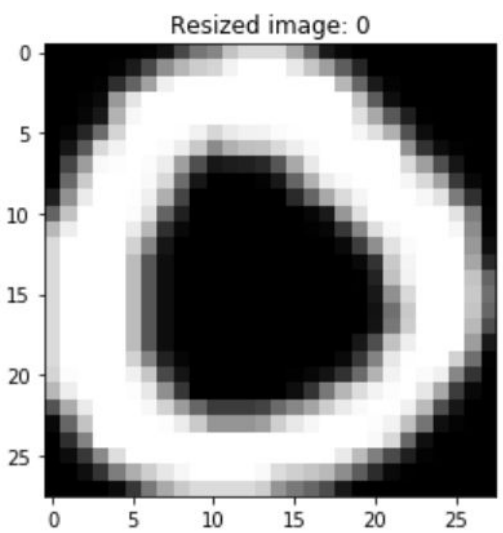


Image after resizing

3.1.2 Train and test data split

The train and test data split is done using scikit-learn library's train_test_split() function as seen below. There are 42,000 images in total and its number is reasonable to split the data by having 20% as test data and 80% as training data.

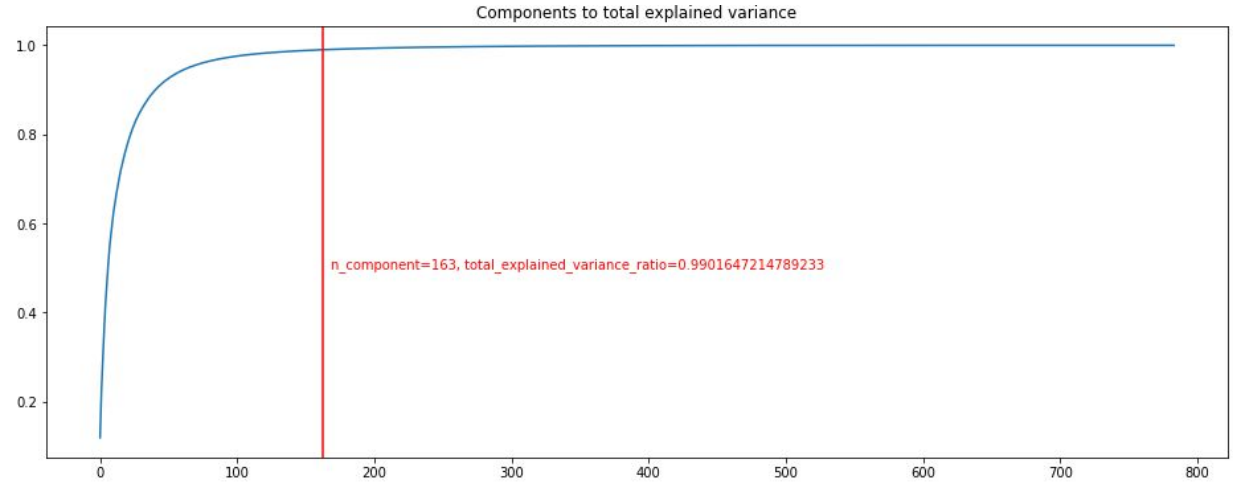
```
X_train, X_test, y_train, y_test = train_test_split(df_images_resized, df_target, test_size=0.2, random_state=RANDOM_STATE)
```

3.1.3 Principal Component Analysis

PCA can be performed through the Sklearn library. Initially, the number of components are set to the total number of pixels defined in an image. This is intentionally done in order to preserve the dimensionality. The below code demonstrates the PCA calculation via Sklearn library.

```
pca = PCA(n_components=784, svd_solver='full')
pca.fit(X_train)
```

Once all components are calculated, the nu mber of components to be used should be determined. The components within the PCA object are ordered based on the explained variance. The goal in this section is to determine the number of components to be used while maintaining the explained variance as much as possible. The below graph shows the relationship between the explained variance ratio and the number of components. Looking at the graph, most variances (>99%) are explained using the first 163 components. Therefore, 163 components are chosen and the dimensionality of data is reduced from 784 to 163.



3.2. Implementation

Overall, implementations for all training algorithms are done via sklearn library. This was done on purpose because the library supports different learning algorithms and therefore it was a good candidate library to keep the implementation consistent.

The implementations were done mainly in 3 steps for every learning algorithm:

1. Training the model using the training data
2. Predict the outcome using the testing input data
3. Get the model performance by calculating the accuracy score with the actual test output

3.2.1 Implementation of Logistic Regression

Logistic regression is a training algorithm supported by the sklearn library. In order to keep the outcome consistent between runs, a random state is setted. The hand-written digit algorithm question is a multi-class classification problem; therefore the training algorithm is instructed to do so by explicitly passing the ‘multi_class’ parameter with ‘multinomial’. It was identified during training that the default max iteration (which is set to 100) is insufficient to converge. The max iteration parameter is therefore explicitly set to a high number, 1000.
Once the model is fitted, the model is used to predict the outcome of testing data. The outcome of the predicted data is then compared with the actual data to calculate the accuracy score.

```
# Logistic Regression
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression(random_state=RANDOM_STATE, solver='lbfgs', multi_class='multinomial', max_iter=1000)
clf.fit(X_train, y_train.values.ravel())
y_pred = clf.predict(X_test)
accuracy_score(y_test, y_pred)
```

Figure. Implementation used to train, predict and calculate accuracy score for logistic regression

3.2.2 Implementation of K nearest Neighbour Classifier

K nearest neighbour training algorithm is supported by the sklearn library. The 'n_neighbors' parameter was carefully picked and this will be discussed in the below refinements section (see 3.3). After training, the model was used to predict against test data and the accuracy score was calculated.

```
# KNN
from sklearn.neighbors import KNeighborsClassifier
clf = KNeighborsClassifier(n_neighbors=4, n_jobs=4)
clf.fit(X_train, y_train.values.ravel())
y_pred = clf.predict(X_test)
accuracy_score(y_test, y_pred)
```

Figure. Implementation used to train, predict and calculate accuracy score for K nearest neighbors classifier

3.2.3 Implementation of Neural Network Classifier

MLP (Multilayer Perceptron) classifier is a Neural network algorithm supported by the sklearn library. 2 hidden layers are used and each layer has 1000 perceptrons. In this particular case, the input layer has 163 nodes, each of which represents a PCA component. The output layer has 10 nodes, each of which represents the output class (digit ranging from 0 to 9).
Once training was completed, the model was used to predict the classes of test data and the outcome was compared against the actual outcome to calculate the accuracy score.

```
# Neural Netweork
from sklearn.neural_network import MLPClassifier
clf = MLPClassifier(hidden_layer_sizes=(1000,1000), random_state=RANDOM_STATE)
clf.fit(X_train, y_train.values.ravel())
y_pred = clf.predict(X_test)
accuracy_score(y_test, y_pred)
```

Figure. Implementation used to train, predict and calculate accuracy score for Neural Network Classifier

3.3. Refinements

3.3.1 Discarding data binary transformation in data pre-processing

In the proposal, it was proposed that the training data will be transformed into a binary number instead of a number ranging from 0 to 255. The idea was to remove the bias/noises produced by the type of pen or by the pressure made by humans. During training, it was identified that the trained models were under-performing compared to the benchmark models.

For example, the KNN classifier's benchmark model has an accuracy score of 97%. However, the produced model has an accuracy score of 93.9% which is significantly lower than the benchmark model. After a thorough investigation, it was identified that the binary transformation actually hurts the performance. The transformation method (pixel is set to 1 if non-zero) was turned out to be redundant because humans perceive the nearly dark pixels as a complete dark pixel. It was decided to completely get rid of this preprocessing step such that the models can actually learn what-to-perceive vs. what-not-to-perceive.

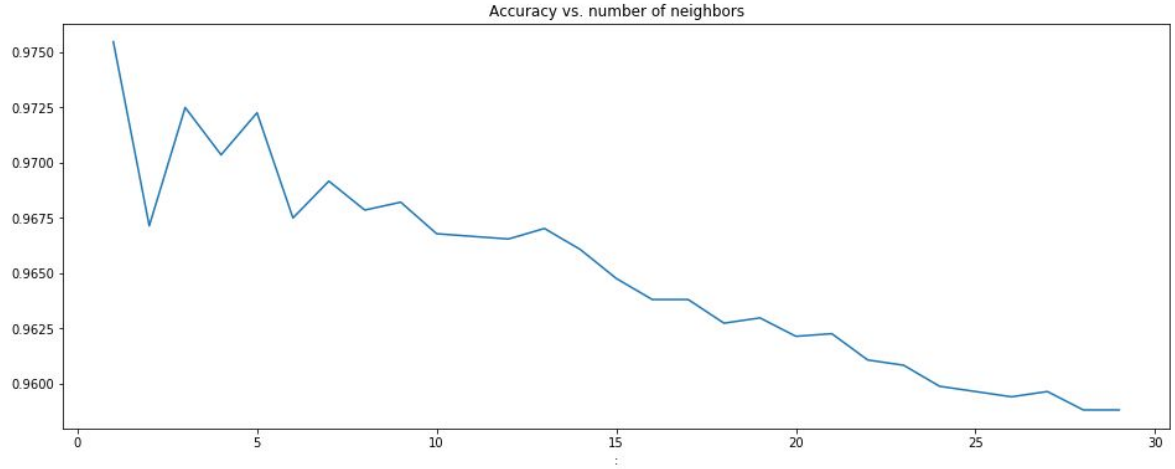
```
In [17]: # KNN
         from sklearn.neighbors import KNeighborsClassifier
         clf = KNeighborsClassifier(n_neighbors=4, n_jobs=4)
         clf.fit(X_train, y_train)
         y_pred = clf.predict(X_test)
         accuracy_score(y_test, y_pred)

         C:\Users\Taihwa\Anaconda3\lib\site-packages\ipykernel_launcher.py:4: DataConversionWarning: A column-vector y was passed when a
         1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
           after removing the cwd from sys.path.

Out[17]: 0.939047619047619
```

Figure. Accuracy score (93.9%) of KNN with binary data

3.3.2 Hyper-Parameter Tuning

3.3.2.1 Tuning K Nearest Neighbor Classifier

The KNN classifier takes a hyper-parameter: n_neighbors. Upon making a decision, this number of closest data points are taken into consideration. The below graph demonstrates the relationship between the hyper parameter and the performance. As the number of neighbors increases, the accuracy decreases. Although the number of neighbors of 1 scored the highest, the model may not generalize. Once the number goes beyond 5, the accuracy decreases drastically. A number between 3 to 5 is a logical decision in this case.



3.3.2.2 Tuning MLP Classifier

Finding the number of nodes and the number of hidden layers is a tricky question for Neural Network models. The grid search algorithm is a handy technique to fine-tune multiple

hyper-parameters. Using the grid search algorithm, different numbers of nodes and the number of hidden layers were tested.

```
# NN Hyper parameter tuning
from sklearn.model_selection import GridSearchCV
param_grid  = {
    'hidden_layer_sizes': [
        (100), (200), (500), (1000),          # single layer
        (100, 100), (500, 500), (1000, 1000), # two layers
    ],
}
clf = MLPClassifier(random_state=RANDOM_STATE)
search = GridSearchCV(clf, param_grid, cv=5, scoring="accuracy")
search.fit(X_train, y_train.values.ravel())
```

The result shows that 2 hidden layers each with 1000 nodes are proven to demonstrate the best result.

	hidden_layer_sizes	Accuracy
0	100	0.973333
1	200	0.976875
2	500	0.977946
3	1000	0.978839
4	(100, 100)	0.971071
5	(500, 500)	0.978274
6	(1000, 1000)	0.980149

Figure. Performance based on the hidden layer size through grid search algorithm.

4. Result

4.1 Model Evaluation and Validation

The model is evaluated with the test data set. The final hyper-parameters are chosen carefully through a grid search algorithm. The performance testing is done by computing the test (unseen) data that were not used for training. The following is the accuracy of each model:

- Logistic Regression: 93.51%
- K Nearest Neighbors: 97.25%
- Neural Network: 98.12%

The logistic regression failed images that are obvious to humans. The below images are the ones that the logistic regression model failed against. All of them are pretty obvious when a human is asked to interpret, but it failed on the model. This result can be inferred by the model performance: 93.51%. Logistic regression is a naive learner that is often not good at learning hidden behaviours; therefore, it often fails on some obvious images.

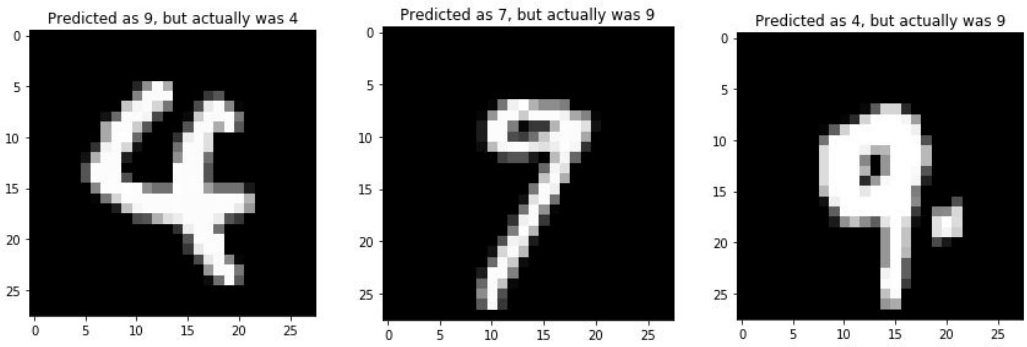


Figure. Images that were misclassified by the logistic regression

The KNN model failed against images that are often not obvious, but an average human may be able to classify it correctly. The below images are the ones that the KNN model failed against. Depending on how you look at the images, it can either be correctly identified or mis-classified upon eye-balling it. This explains why KNN shows a pretty robust performance: 97.25%

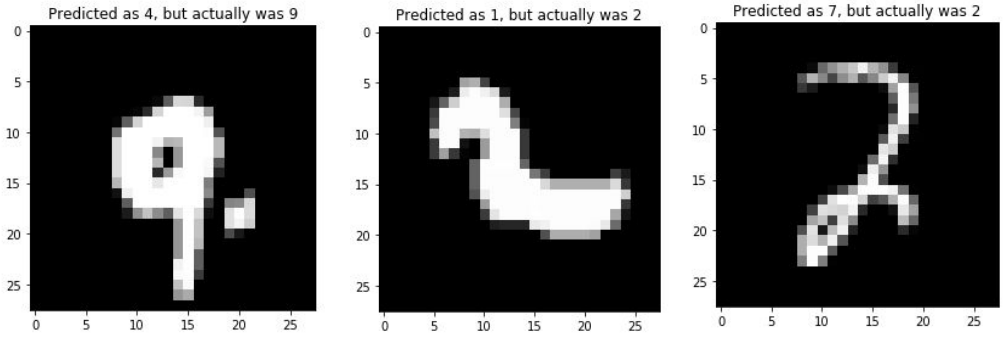


Figure. Images that were misclassified by KNN

The Neural Network model out-performed the other two with the accuracy score of 98.12%. Images that it mis-classified makes even an average person wonder about the actual outcome. The below has the images that the neural network model failed against.

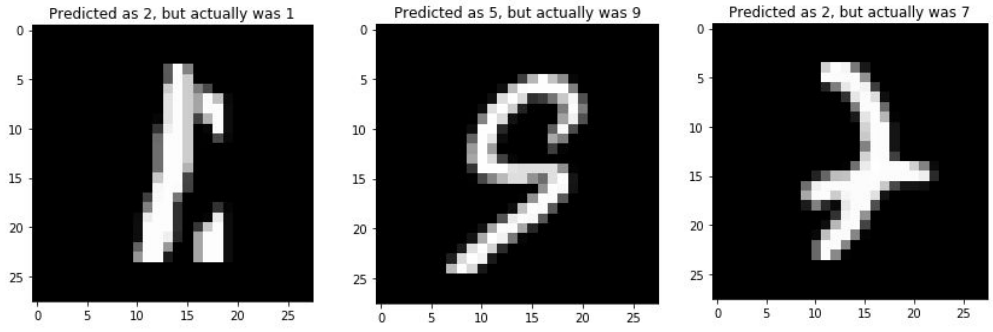


Figure. Images that were misclassified by the Neural Network

In order to justify the robustness of the models, the models were tested against non-MNIST dataset. These hand-written digits were manually created and the total of 20 images were created. Some examples of the manually written digits are shown below.

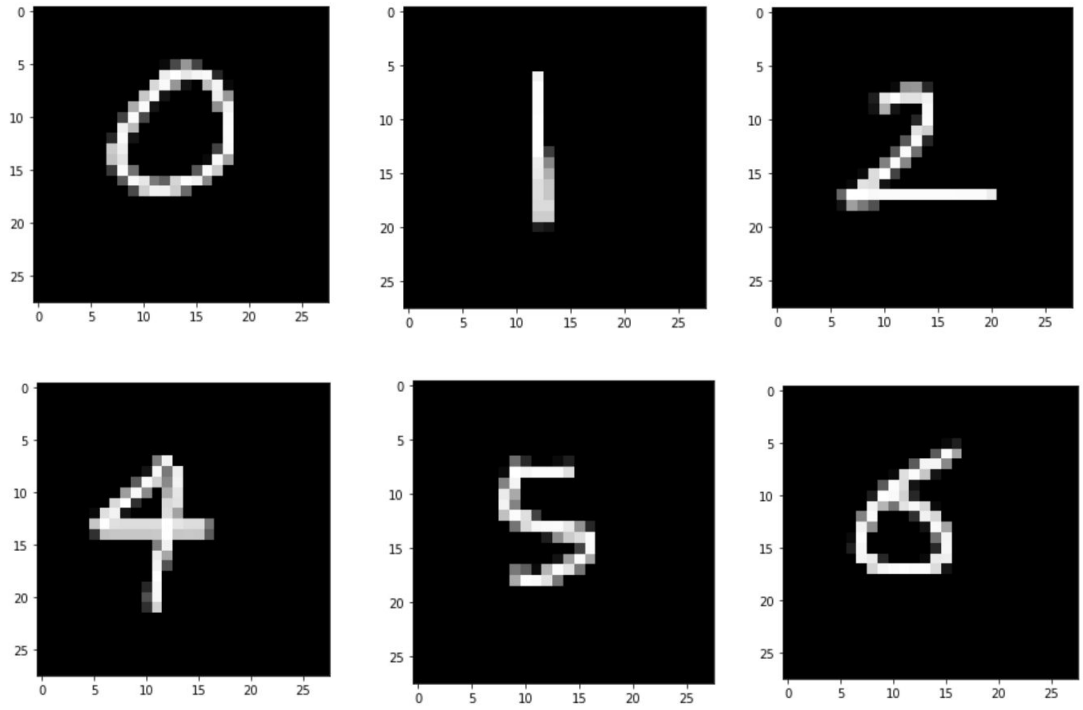


Figure. Example images of manually written non-MNIST dataset

The accuracy scores of all models are 1.0 (100%). This result justifies that the models work beyond the MNIST dataset and therefore can be generalized to any handwritten digits.

```
Accuracy score of logistic regression: 1.0
Accuracy score of KNN: 1.0
Accuracy score of neural network: 1.0
```

Figure. Accuracy score returned from models

4.2 Justification

The accuracy score of the logistic regression model was 0.935. The benchmarking model showed 0.92 so our model outperformed the benchmarking model. The accuracy score of the K nearest neighbor was 0.973. The benchmarking model showed 0.970; therefore our model slightly performed better than the benchmarking model. The accuracy score of the neural network was 0.981. The benchmarking model showed 0.98 so our model was performing just as good as the benchmarking model.

Although the logistic regression outperformed the benchmarking, the model's accuracy is not high enough to have adequately solved the problem. On the other hand, KNN and neural network models' performances are significant as the models predicted the outcome better than 97%. It is good enough to say that these two models have adequately solved the problem.

5. References

1. https://en.wikipedia.org/wiki/ReCAPTCHA
2. https://en.wikipedia.org/wiki/CAPTCHA
3. https://en.wikipedia.org/wiki/Handwriting_recognition
4. https://www.kaggle.com/c/digit-recognizer
5. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
6. https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm
7. https://en.wikipedia.org/wiki/Neural_network