SML - CW2 - R

Wen Hans Tan

2025-02-30

The MNIST Dataset

In this assessment, we are going to investigate one of the most famous datasets used for machine learning; the MNIST (Modified National Institute of Standards and Technology) database. The full dataset consists of 70,000 training images of hand-written digits between 0 and 9. However, we shall only use a sample of 1,500 of these digits. The below code will load in the MNIST dataset.

```
# Load the dataset
load("digit_img.RData")
```

We shall also load in some support functions which will be useful for working with this dataset.

```
source("digit_utils.R")
```

You will now find that your workspace contains a variable named `digit_img`.

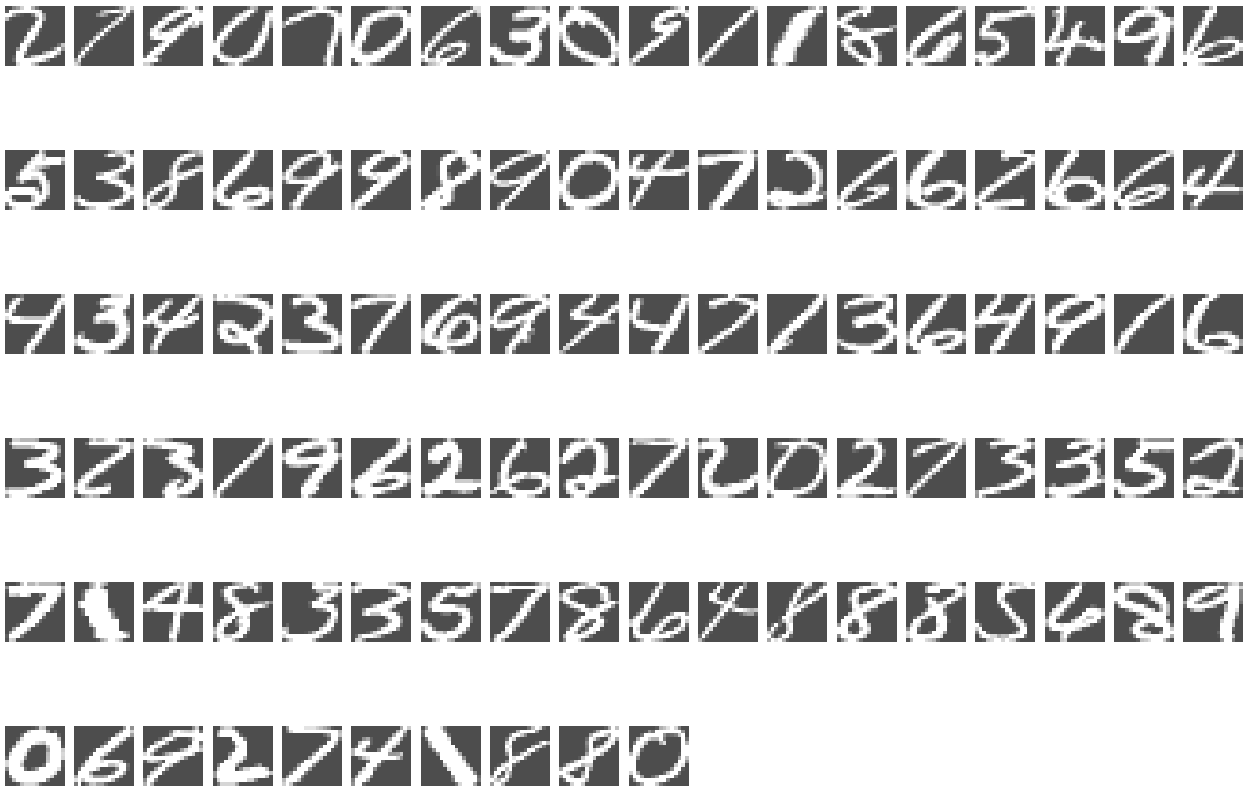
`digit_img` is a 256×1500 matrix, where the first dimension is "space" unwrapped (i.e. the images of handwritten digits are $16 \times 16 = 256$ arrays of pixels), and the second dimension are the 1500 images. We shall treat the first 1000 as your training data and the remaining 500 as test data.

```
# Get training data
training_data <- t(digit_img[, 1:1000])

# Get testing data
testing_data <- t(digit_img[, 1001:1500])
```

To help you understand the data, we have provided the `display_digit` function. For example, to see the first 100 digits...

```
display_digit(digit_img[,1:100])
```



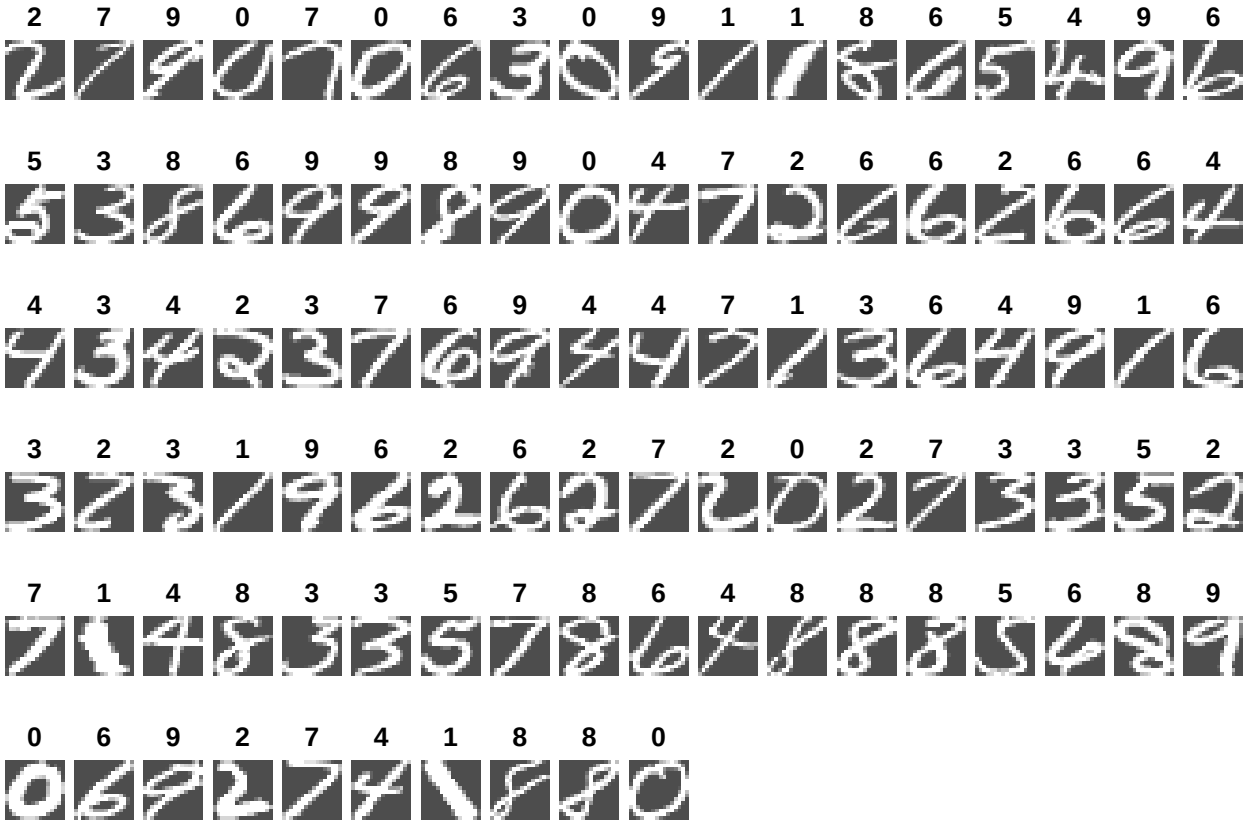
The `digit_lab` variable is a length 1500 label vector telling you the label (true identity) of the digit, one of 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 (10 means "zero", so we convert it below).

```
# Load the labels
load("digit_lab.RData")

# Replace elements equal to 10 with 0 (it's easier to view this way)
digit_lab[digit_lab == 10] <- 0
```

You can view the labels alongside the images using the `display_digit` function as follows:

```
display_digit(digit_img[,1:100],digit_lab[,1:100])
```



The below code will split the labels into training and testing for you.

```
# Get training labels
training_labels <- t(digit_lab[, 1:1000])

# Get testing labels
testing_labels <- t(digit_lab[, 1001:1500])
```

Practical

Part (i)

Which preprocessing steps (if any) might you consider applying to this dataset before writing your answers to part (ii)? Your answer should be at most three sentences (no code!). *In parts (ii) and (iii), you may assume we have preprocessed the data for you.* ***(3 marks)***

Remove missing data and outliers should be considered standard as it is sensitive for each classification task. Convert all labels into factors so that R can process the values and re-scale or normalise all continuous data because classification algorithms rely on distance metrics which are sensitive to the magnitude of the data. Lastly, we check correlation between factors ; a higher correlation means that it is difficult to estimate the individual contribution into estimating the outcome.

Classification Task

For part (ii) of Problem 2 on the assignment, your task is to classify the handwritten digit images by predicting the labels (true digit) from the features (images). For each classification method listed in the

following sections, you must do the following:

- Use the suggested functions to train your model.
- Generate a confusion matrix and calculate the classification accuracy using the `confusionMatrix` function from the `caret` package.
- Visualize 100 test images along with their predicted labels using the `display_digit` function.

K-Nearest Neighbours

Using the training data and labels, apply the `knn` function from the `caret` package to perform K-Nearest Neighbours classification on the dataset. Evaluate the model's performance as described above.

```
library(class)
library(ggplot2)
library(lattice)
library(caret)
# Alter k-value
k <- 9
testing_labels <- factor(testing_labels)

# Predict labels using knn function
pred_knn <- knn(train = training_data,
            test = testing_data,
            cl = training_labels,
            k = k)

# Create a confusion matrix
conf_Matrix <- confusionMatrix(pred_knn, testing_labels)

# Display the result
print(conf_Matrix)
```

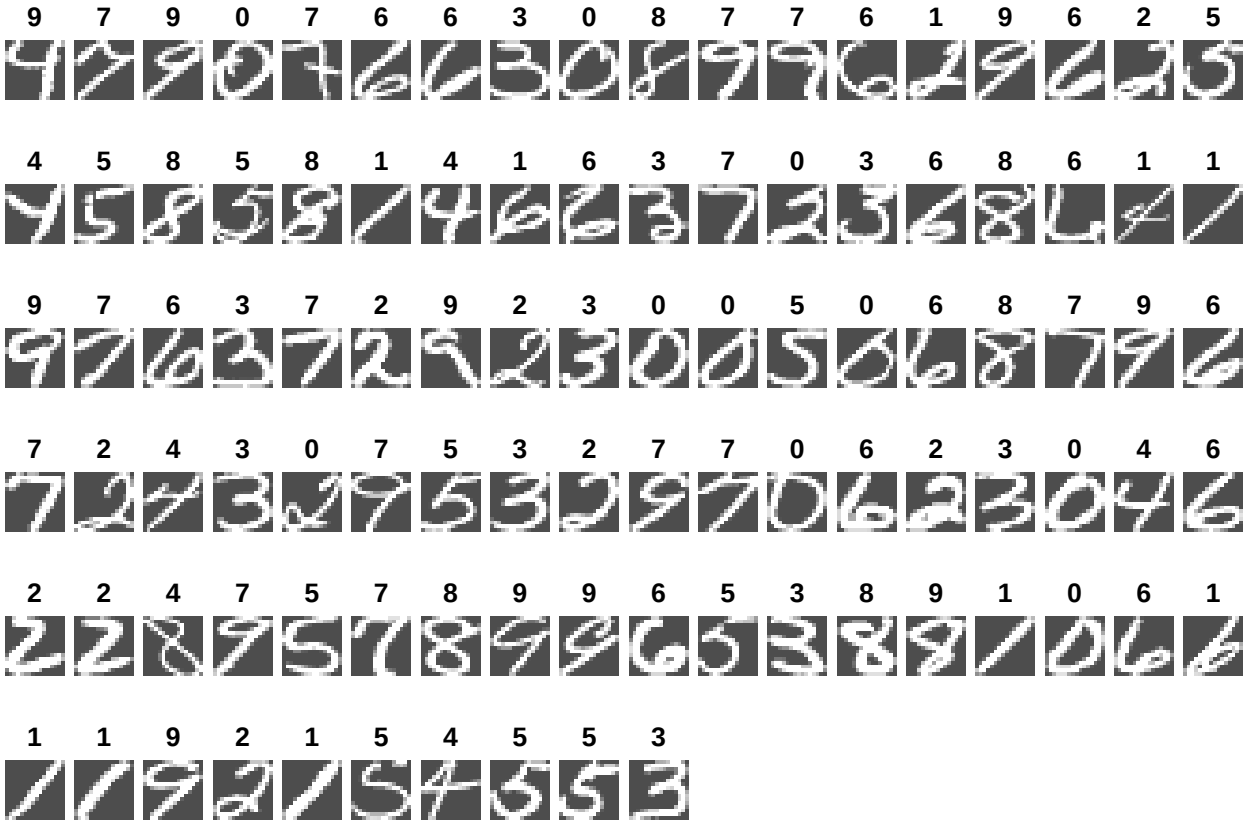
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1  2  3  4  5  6  7  8  9
##          0 54  0  6  0  0  0  0  0  0  0
##          1  1 43  3  1  8  0  3  2  1  1
##          2  0  0 35  1  0  0  0  0  0  0
##          3  0  0  0 41  0  4  0  0  1  0
##          4  2  0  0  0 35  0  0  0  1  4
##          5  0  0  2  0  0 46  0  0  2  0
##          6  2  0  0  0  1  0 53  0  1  0
##          7  0  0  4  2  1  0  1 47  2  9
##          8  0  0  2  0  0  0  0  0 35  0
##          9  0  0  0  0  4  1  0  1  2 35
##
## Overall Statistics
##
##                Accuracy : 0.848
##                  95% CI : (0.8135, 0.8783)
##     No Information Rate : 0.118
##     P-Value [Acc > NIR] : < 2.2e-16
```

```
##
##                   Kappa : 0.831
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            0.9153   1.0000   0.6731   0.9111   0.7143   0.9020
## Specificity            0.9864   0.9562   0.9978   0.9890   0.9845   0.9911
## Pos Pred Value         0.9000   0.6825   0.9722   0.8913   0.8333   0.9200
## Neg Pred Value         0.9886   1.0000   0.9634   0.9912   0.9694   0.9889
## Prevalence             0.1180   0.0860   0.1040   0.0900   0.0980   0.1020
## Detection Rate         0.1080   0.0860   0.0700   0.0820   0.0700   0.0920
## Detection Prevalence   0.1200   0.1260   0.0720   0.0920   0.0840   0.1000
## Balanced Accuracy      0.9508   0.9781   0.8354   0.9501   0.8494   0.9465
##                      Class: 6 Class: 7 Class: 8 Class: 9
## Sensitivity            0.9298   0.9400   0.7778   0.7143
## Specificity            0.9910   0.9578   0.9956   0.9823
## Pos Pred Value         0.9298   0.7121   0.9459   0.8140
## Neg Pred Value         0.9910   0.9931   0.9784   0.9694
## Prevalence             0.1140   0.1000   0.0900   0.0980
## Detection Rate         0.1060   0.0940   0.0700   0.0700
## Detection Prevalence   0.1140   0.1320   0.0740   0.0860
## Balanced Accuracy      0.9604   0.9489   0.8867   0.8483
```

```
accuracy <- conf_Matrix$overall["Accuracy"]
accuracy
```

```
## Accuracy
##    0.848
```

```
display_digit(digit_img[,1001:1100],pred_knn)
```



Naive Bayes

Using the training data and labels, apply the `naiveBayes` function from the `e1071` package to perform Naive Bayes classification on the dataset. Evaluate the model's performance as described above.

```
library(e1071)
testing_labels <- factor(testing_labels)
training_labels <- factor(training_labels)

train_df <- data.frame(training_data, label = training_labels)
test_df <- data.frame(testing_data, label = testing_labels)

# Fit the Naive Bayes Model
NB_model <- naiveBayes(training_labels ~ ., data = train_df)

# Make Predictions on Test Data
pred_NB <- predict(NB_model, newdata = test_df)

# Create a confusion matrix
conf_Matrix <- confusionMatrix(pred_NB, testing_labels)

# Display confusion matrix and accuracy
conf_Matrix
```

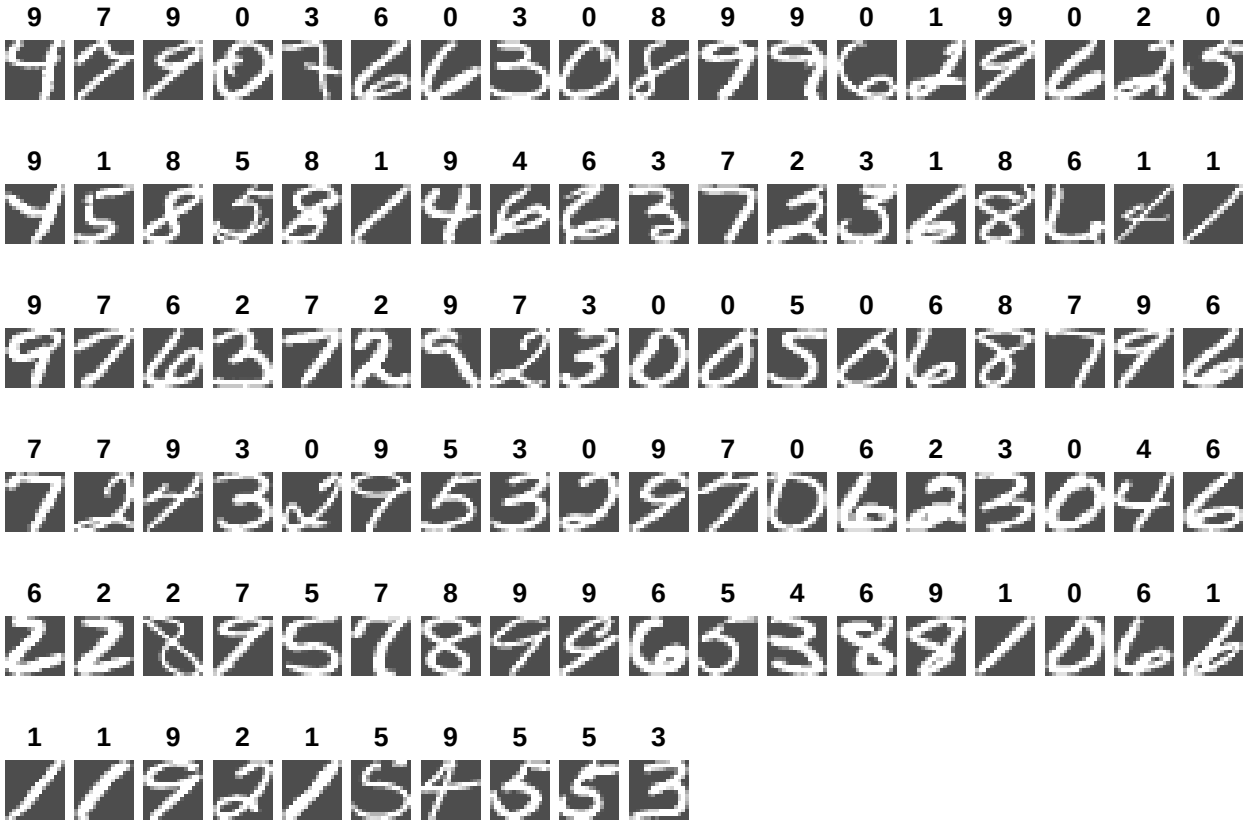
```
## Confusion Matrix and Statistics
```

```
##
##           Reference
## Prediction  0  1  2  3  4  5  6  7  8  9
##          0 52  0  3  0  0  1 11  0  0  0
##          1  2 37  8  7  6 12  6  4  6  3
##          2  2  1 30  6  0  0  0  1  3  0
##          3  0  0  0 27  0  1  0  2  0  0
##          4  0  0  0  1 29  0  2  1  1  1
##          5  0  0  2  0  0 33  1  0  0  0
##          6  3  2  1  2  0  1 37  0  2  0
##          7  0  1  6  1  1  0  0 39  0  1
##          8  0  2  2  0  0  1  0  0 28  0
##          9  0  0  0  1 13  2  0  3  5 44
##
## Overall Statistics
##
##                Accuracy : 0.712
##                  95% CI : (0.6701, 0.7513)
##     No Information Rate : 0.118
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.68
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            0.8814   0.8605   0.5769   0.6000   0.5918   0.6471
## Specificity            0.9660   0.8818   0.9710   0.9934   0.9867   0.9933
## Pos Pred Value         0.7761   0.4066   0.6977   0.9000   0.8286   0.9167
## Neg Pred Value         0.9838   0.9853   0.9519   0.9617   0.9570   0.9612
## Prevalence             0.1180   0.0860   0.1040   0.0900   0.0980   0.1020
## Detection Rate         0.1040   0.0740   0.0600   0.0540   0.0580   0.0660
## Detection Prevalence   0.1340   0.1820   0.0860   0.0600   0.0700   0.0720
## Balanced Accuracy      0.9237   0.8712   0.7740   0.7967   0.7893   0.8202
##                      Class: 6 Class: 7 Class: 8 Class: 9
## Sensitivity            0.6491   0.7800   0.6222   0.8980
## Specificity            0.9752   0.9778   0.9890   0.9468
## Pos Pred Value         0.7708   0.7959   0.8485   0.6471
## Neg Pred Value         0.9558   0.9756   0.9636   0.9884
## Prevalence             0.1140   0.1000   0.0900   0.0980
## Detection Rate         0.0740   0.0780   0.0560   0.0880
## Detection Prevalence   0.0960   0.0980   0.0660   0.1360
## Balanced Accuracy      0.8121   0.8789   0.8056   0.9224
```

```
accuracy <- conf_Matrix$overall["Accuracy"]
accuracy
```

```
## Accuracy
##    0.712
```

```
display_digit(digit_img[,1001:1100],pred_NB)
```



Support Vector Machines

Using the training data and labels, apply the `svm` function from the `e1071` package to perform support vector machine classification with a `radial` kernel. Evaluate the model's performance as described above.

```
testing_labels <- factor(testing_labels)
training_labels <- factor(training_labels)
train_df <- data.frame(training_data, label = training_labels)
test_df <- data.frame(testing_data, label = testing_labels)

# Train the SVM model
svm_model_radial<- svm(training_labels ~ ., data =train_df, kernel = "radial", gamma = 0.007,probabilit

# Predict labels
pred_SVM <- predict(svm_model_radial, newdata = test_df)

# Create a confusion matrix
conf_Matrix <- confusionMatrix(pred_SVM, testing_labels)

conf_Matrix
```

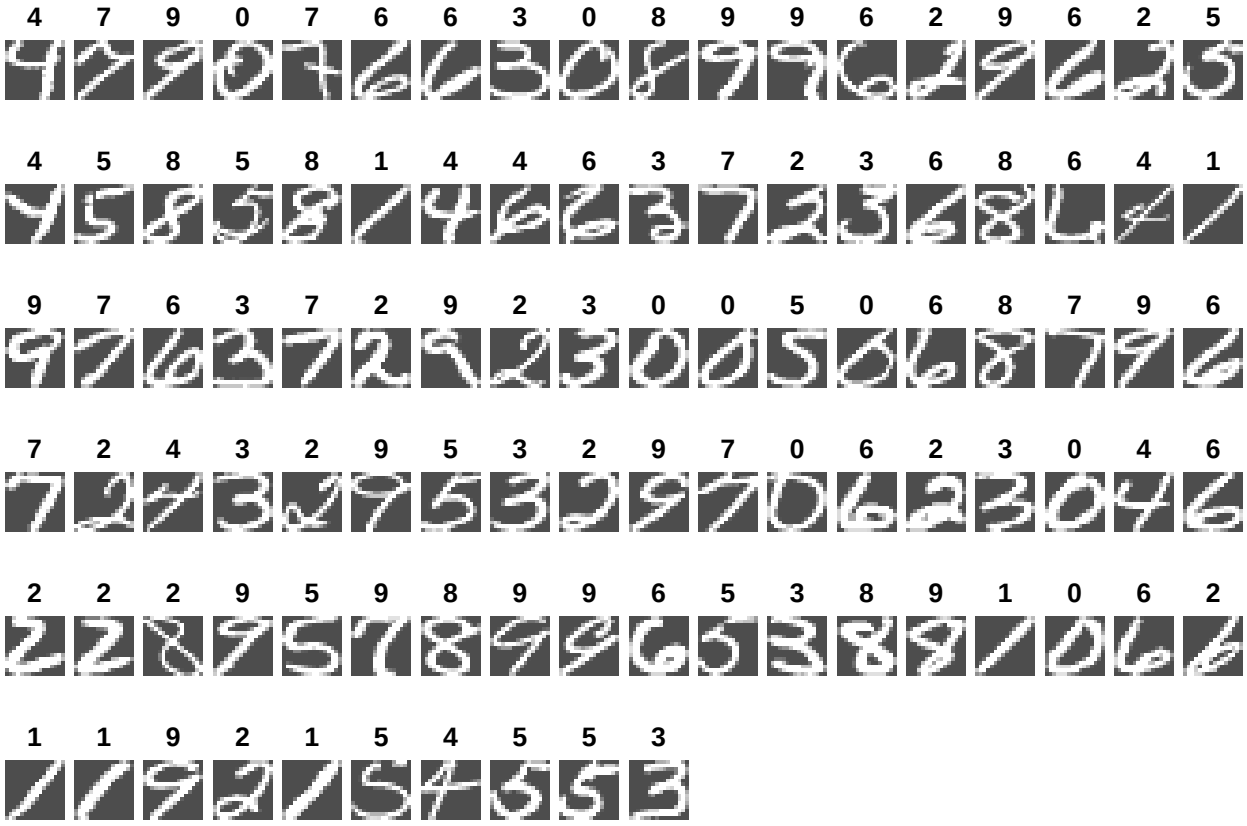
```
## Confusion Matrix and Statistics
```

```
##
##           Reference
## Prediction  0  1  2  3  4  5  6  7  8  9
##          0 54  0  1  0  0  0  0  0  0  0
##          1  0 41  0  0  0  0  1  0  0  0
##          2  1  1 47  3  0  3  2  2  3  0
##          3  0  0  0 40  0  1  0  0  1  0
##          4  1  0  0  0 47  0  1  1  0  1
##          5  0  0  1  0  0 46  0  0  0  0
##          6  2  0  0  0  1  0 53  0  0  0
##          7  0  0  2  1  0  0  0 44  0  0
##          8  1  1  1  0  0  0  0  0 39  0
##          9  0  0  0  1  1  1  0  3  2 48
##
## Overall Statistics
##
##                Accuracy : 0.918
##                  95% CI : (0.8904, 0.9405)
##     No Information Rate : 0.118
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9088
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            0.9153   0.9535   0.9038   0.8889   0.9592   0.9020
## Specificity            0.9977   0.9978   0.9665   0.9956   0.9911   0.9978
## Pos Pred Value         0.9818   0.9762   0.7581   0.9524   0.9216   0.9787
## Neg Pred Value         0.9888   0.9956   0.9886   0.9891   0.9955   0.9890
## Prevalence             0.1180   0.0860   0.1040   0.0900   0.0980   0.1020
## Detection Rate         0.1080   0.0820   0.0940   0.0800   0.0940   0.0920
## Detection Prevalence   0.1100   0.0840   0.1240   0.0840   0.1020   0.0940
## Balanced Accuracy      0.9565   0.9757   0.9352   0.9422   0.9752   0.9499
##                      Class: 6 Class: 7 Class: 8 Class: 9
## Sensitivity            0.9298   0.8800   0.8667   0.9796
## Specificity            0.9932   0.9933   0.9934   0.9823
## Pos Pred Value         0.9464   0.9362   0.9286   0.8571
## Neg Pred Value         0.9910   0.9868   0.9869   0.9977
## Prevalence             0.1140   0.1000   0.0900   0.0980
## Detection Rate         0.1060   0.0880   0.0780   0.0960
## Detection Prevalence   0.1120   0.0940   0.0840   0.1120
## Balanced Accuracy      0.9615   0.9367   0.9300   0.9809
```

```
accuracy <- conf_Matrix$overall["Accuracy"]
accuracy
```

```
## Accuracy
##    0.918
```

```
display_digit(digit_img[,1001:1100],pred_SVM)
```



(iii) Which of the classifiers in part (ii) do you think is most suitable for this problem? *Explain your answer. Your answer should be no more than three sentences.* **(3 marks)**

Given a suitable gamma value, I think the SVM is the best classification method for this problem because of it's capability to capture very complex patterns in medium to high dimension data-sets. Unlike KNN, SVM avoids the curse of dimensionality by leveraging support vectors to construct optimal hyper-planes. Although SVM has a higher complexity compared to Naive Bayes, its ability to maximise margin between classes enhances generalisation, preventing over-fitting.