**Syracuse University**

**IST-707 Assignment 7**

ThulasiRam RuppaKrishnan

IST 707Section: 35

Professor Ami Gates

Contents

[Introduction 3](#_Toc11067864)

[Analysis and Models 7](#_Toc11067865)

[**About the data** 7](#_Toc11067866)

[**Models** 11](#_Toc11067867)

[Results 33](#_Toc11067868)

[Conclusion 47](#_Toc11067869)

## 

## **Introduction**

Artificial Intelligence, Machine Learning and Deep Learning (AI | ML | DL) are at the heart of digital transformation by enabling organizations to exploit their growing wealth of big data to optimize key business and operational use cases.

* **AI** (Artificial Intelligence) is the theory and development of computer systems able to perform tasks normally requiring human intelligence (e.g. visual perception, speech recognition, **Image recognition**, translation between languages, etc.).
* **ML** (Machine Learning) is a sub-field of AI that provides systems the ability to learn and improve by itself from experience without being explicitly programmed.
* **DL** (Deep Learning) is a type of ML built on a deep hierarchy of layers, with each layer solving different pieces of a complex problem. These layers are interconnected into a “neural network.” A DL framework is SW that accelerates the development and deployment of these models.

**Image Recognition**

Image recognition refers to technologies that identify places, logos, people, objects, buildings, and several other variables in images. Users are sharing vast amounts of data through apps, social networks, and websites. Additionally, mobile phones equipped with cameras are leading to the creation of limitless digital images and videos. The large volume of digital data is being used by companies to deliver better and smarter services to the people accessing it.

The image recognition market is estimated to grow from USD 15.95 Billion in 2016 to USD 38.92 Billion by 2021, at a CAGR of 19.5% between 2016 and 2021 (source <https://www.transparencymarketresearch.com/image-recognition-market.html>). Advancements in machine learning and use of high bandwidth data services is fueling the growth of this technology. Companies in different sectors such as e-commerce, automotive, healthcare, and gaming are rapidly adopting image recognition.

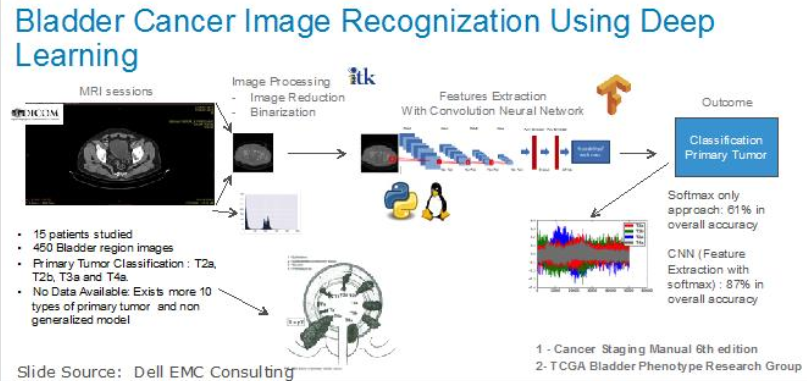


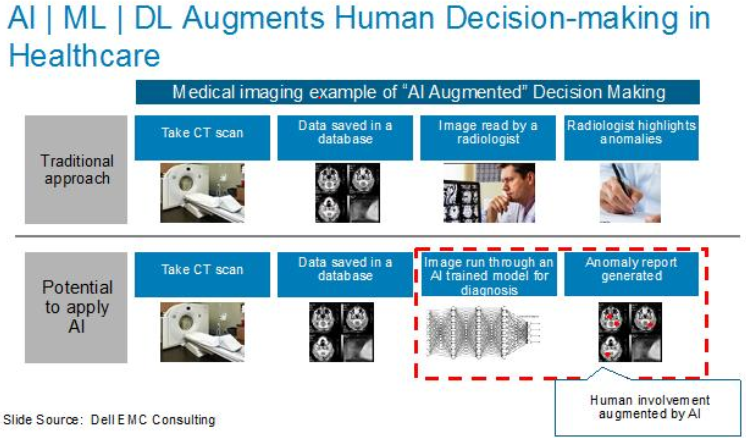
Image recognition is a part of computer vision and a process to identify and detect an object or attribute in a digital video or image. Computer vision is a broader term which includes methods of gathering, processing and analyzing data from the real world. The data is high-dimensional and produces numerical or symbolic information in the form of decisions. Apart from image recognition, computer vision also includes event detection, object recognition, learning, image reconstruction and video tracking. The major steps in image recognition process are gather and organize data, build a predictive model and use it to recognize images.

Practical applications for image recognition are very broad in nature and has numerous benefits for the human life conditions. Here are the few examples that has positive impacts on human beings and for their advancement as technological species in the known universe.

**Use Case #1:  Bladder Cancer Identification Using Medical Image Recognition**

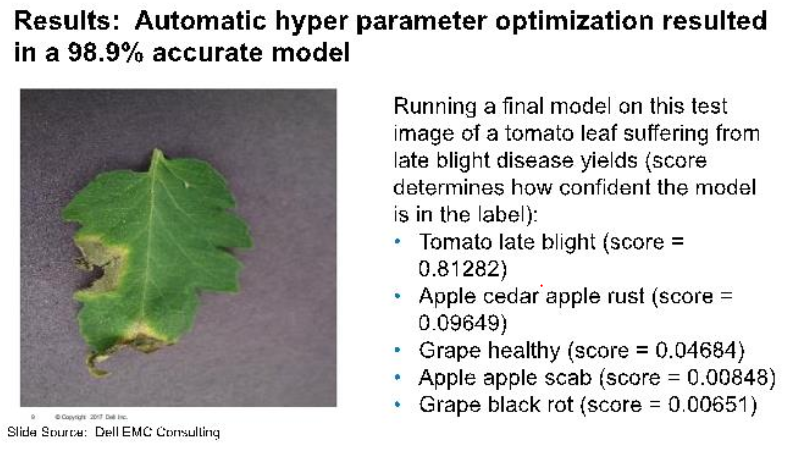
Image recognition of the human body is expected to improve drastically to help doctors with better and more accurate medical diagnostics. ML applied to image recognition of organs, even in the presence of disease, can minimize the possibility of medical errors and speed up disease diagnosis.





**Use Case #2: Crop Disease Identification**

One potential application is the development of mobile disease diagnostics through Image recognition, Deep Learning and crowdsourcing. The results of the engagement were very impressive in scoring different types of crops and their risk to unhealthy situations



Source : <https://infocus.dellemc.com/william_schmarzo/democratizing-artificial-intelligence-deep-learning-machine-learning-with-dell-emc-ready-solutions/>

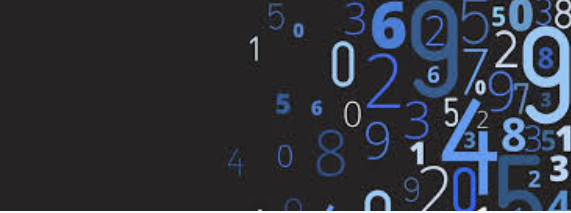
**Classification problem**

Classification between the objects is easy task for humans but it has proved to be a complex problem for machines. The raise of high-capacity computers, the availability of high quality and low-priced video cameras, and the increasing need for automatic video analysis has generated an interest in object classification algorithms. Classification system needs information that contains predefined patterns that compares with detected object to classify in to proper category.

Image classification is an important and challenging task in various application domains, including biomedical imaging, biometry, video surveillance, vehicle navigation, industrial visual inspection, robot navigation, and remote sensing

**Handwritten digit recognition and classification complexity**

Handwritten digit recognition has gained so much popularity from the aspiring beginner of machine learning and deep learning to an expert who has been practicing for years. Developing such a system includes a machine to understand and classify the images of handwritten digits as 10 digits (0–9).



The handwritten digits are not always of the same size, width, orientation and justified to margins as they differ from writing of person to person, so the general problem would be while classifying the digits due to the similarity between digits such as 1 and 7, 5 and 6, 3 and 8, 2 and 5, 2 and 7, etc. This problem is faced more when many people write a single digit with a variety of different handwritings. Lastly, the uniqueness and variety in the handwriting of different individuals also influence the formation and appearance of the digits.

## **Analysis and Models**

### **About the data**

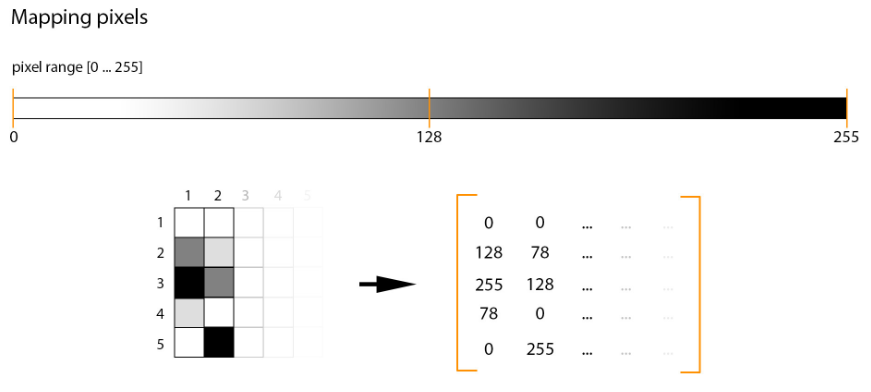
The training data set, (train.csv), has 785 columns. The first column, called "label", is the digit that was drawn by the user. The rest of the columns contain the pixel-values of the associated image. Each pixel column in the training set has a name like pixel-x, where x is an integer between 0 and 783, 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.

Each image is 28 pixels in height and 28 pixels in width, for a total of 784 pixels in total. Each pixel has a single pixel-value associated with it, indicating the lightness or darkness of that pixel, with higher numbers meaning darker. This pixel-value is an integer between 0 and 255, inclusive.

see **Figure 1.1** and **Figure 1.2** for a similar arrangement described above using 4 x 4 matrix.



**Figure 1.1**

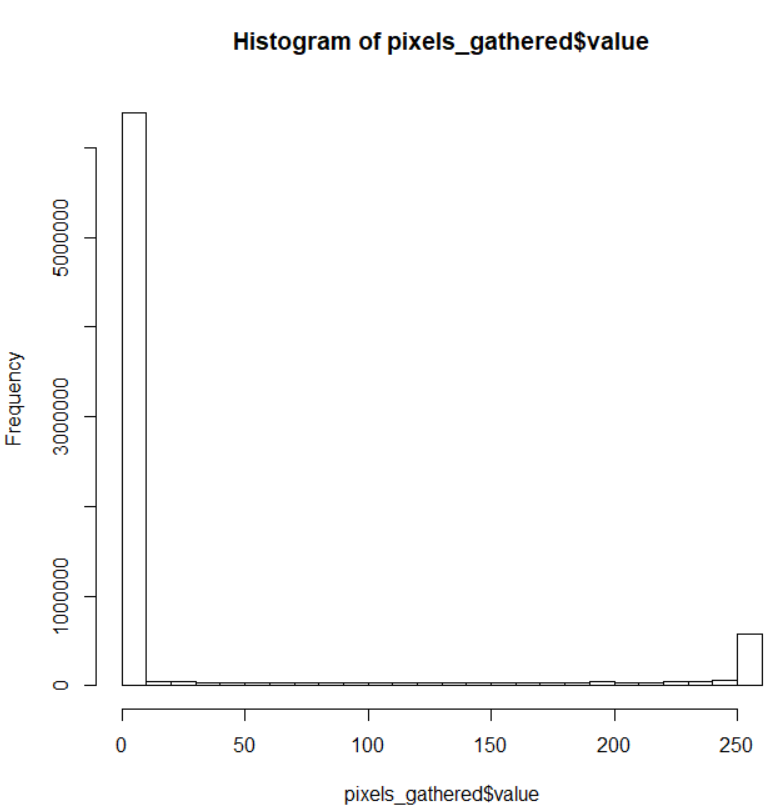


**Figure 1.2 Mapping pixels**

Following points are same in training and testing set along with the set of the images and labels files:

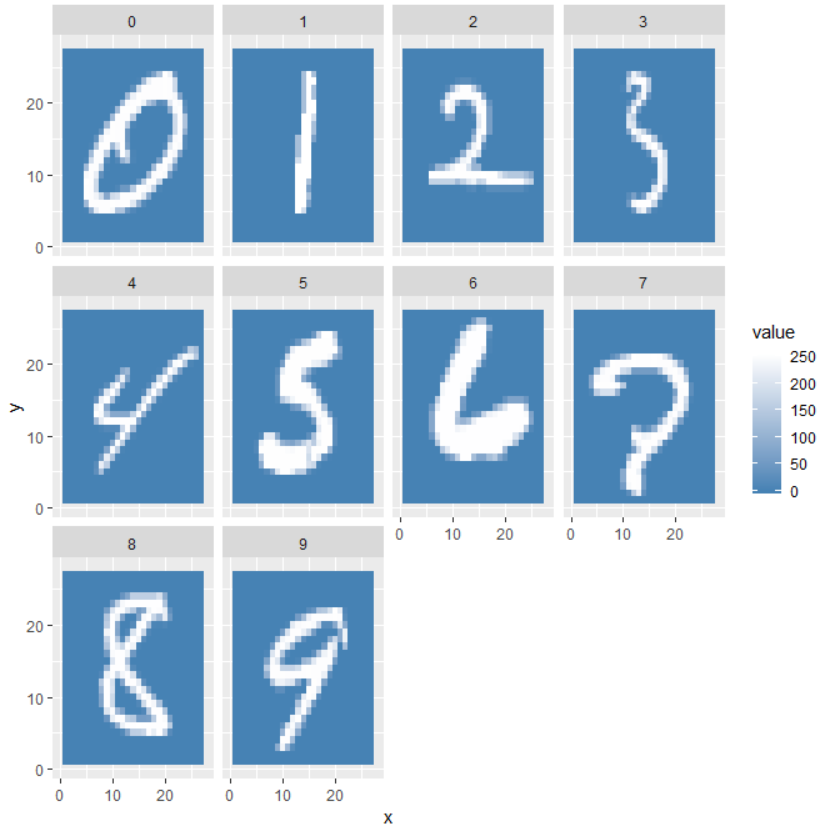
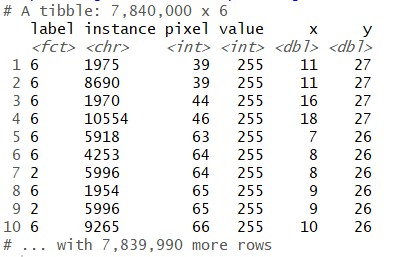
* Background as white (0 value from RGB) and foreground as black (255 value from RGB). (see **Figure 1.1 and Figure 1.2**)
* Pixels are arranged row-wise, ranging from 0 to 255, as from RGB color code. (see **Figure 1.1 and Figure 1.2**)
* Labels of digits classified from 0 to 9.

**Figure 1.3** shows the histogram of the pixel value of some sample images. The distribution is not normal and there are predominant number of attributes with pixel value of 0 and then the next large number of items are with pixel value more than 250



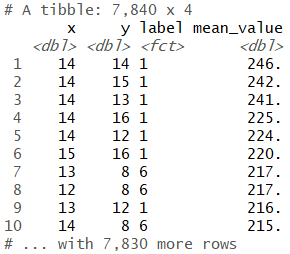
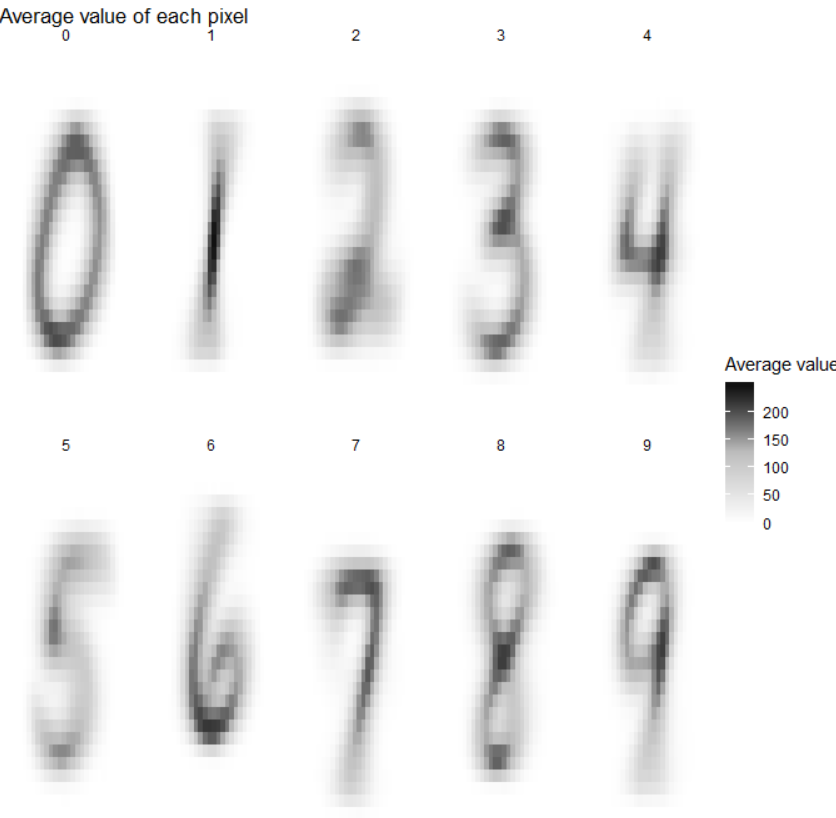
**Figure 1.3 Pixel Value Histogram**

In **Table 1.1** the input pixel value is transformed into 2D and the **Figure 1.4** shows some sample images from the hand-written image digit



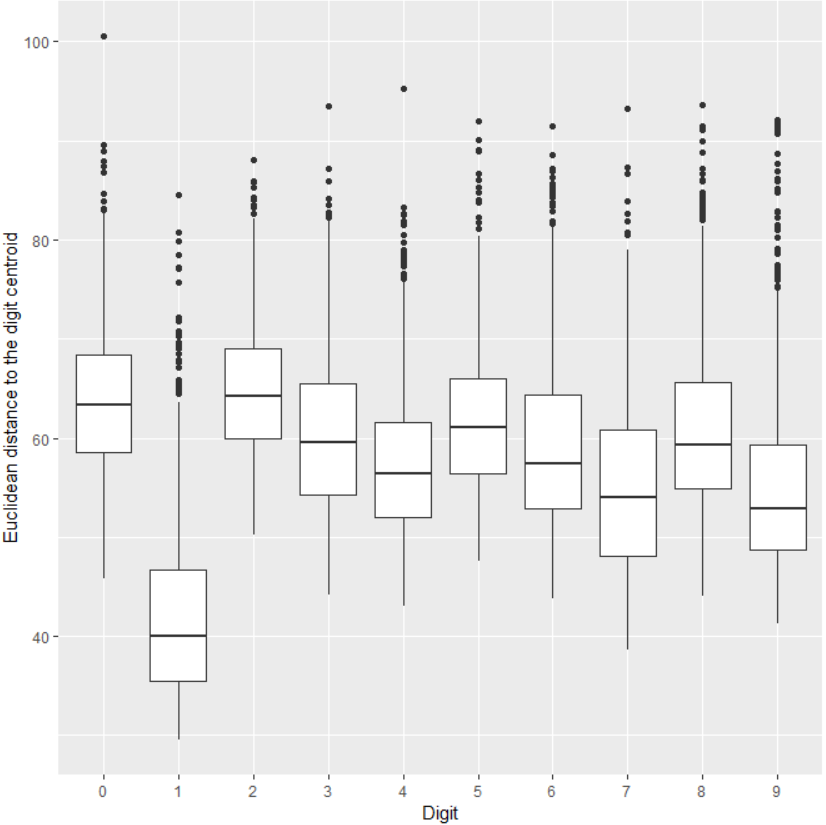
**Table 1.1 Pixel value transformed in 2D Figure 1.4 Sample image from the input dataset**

In **Table 1.2** shows the average pixel value for the given xy coordinates and for a given digit. **Figure 1.5** shows the digit 0 to 9 using average pixel values

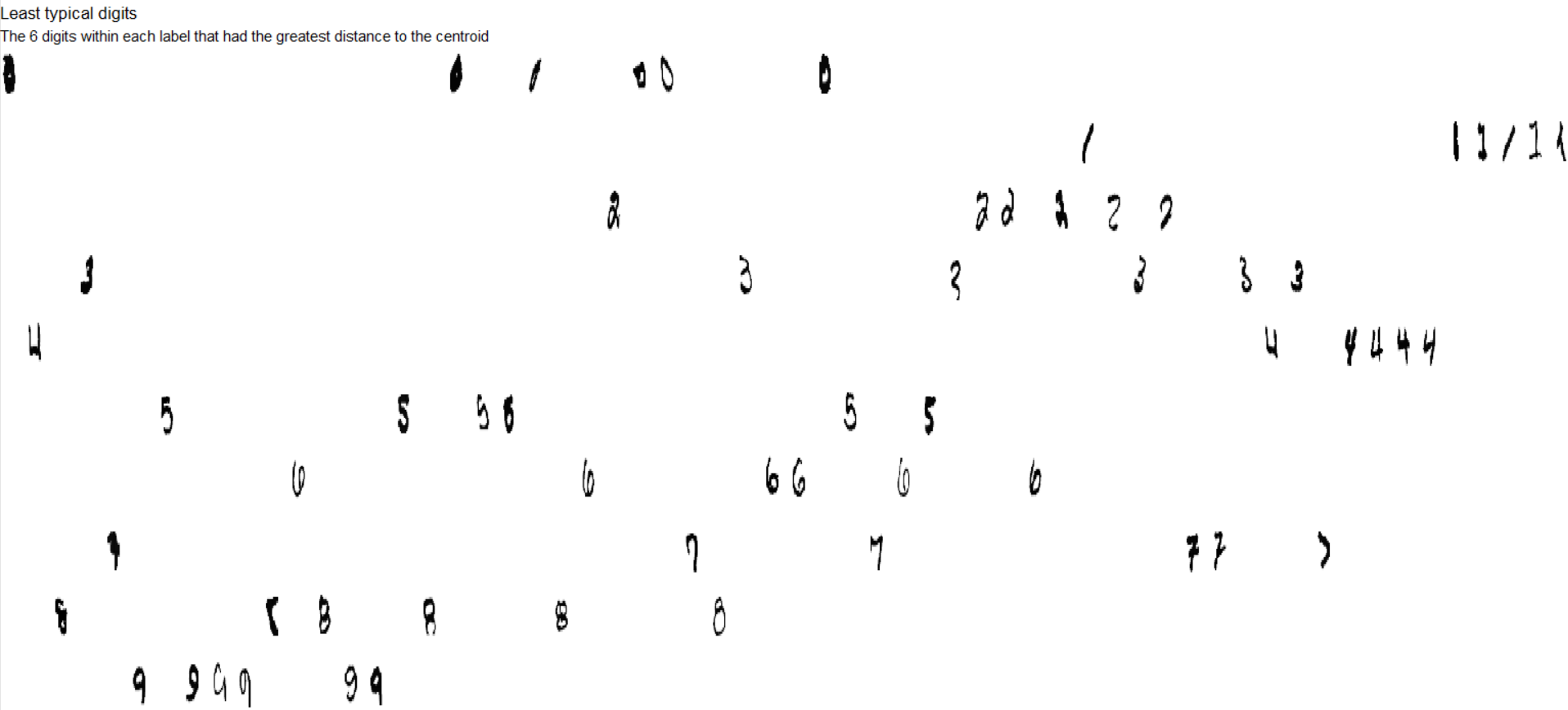
**Table 1.2 Average Pixel value Figure 1.5 Image from the average of individual pixel value**

Variations and outliers for each digit is shown in the below box chart.



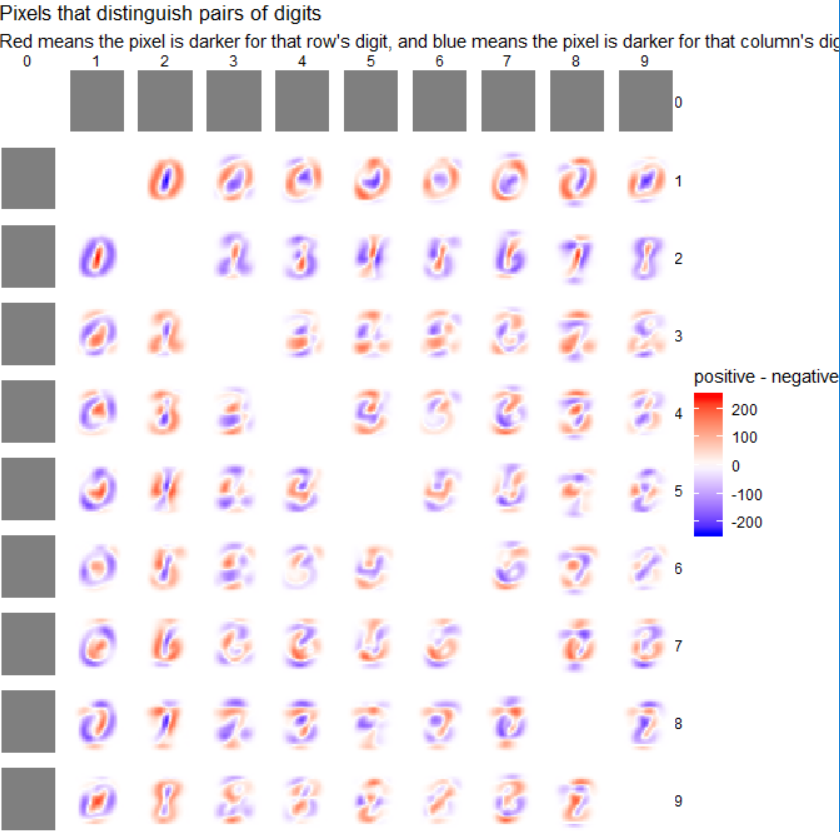
**Figure 1.6 boxplot for images 0 to 9**

Least typical digits are shown in the **Figure 1.7** showing digits with large number of variance from the mean



**Figure 1.7 Least typical digits**

Pairwise comparison of each digit with respect to one another is shown in **Figure 1.8**



**Figure 1.8 pairwise comparison of digits**

### **Models**

In this exercise, models are developed using KNN, SVM and Random Forest to compare their efficiency and accuracy in classifying the handwritten images into their right bucket from 0 to 9.

#### **KNN (k-nearest neighbors)**

In pattern recognition, the k-nearest neighbors algorithm (k-NN) is a method used for classification and regression. In both cases, the input consists of the k closest observations in the feature space. The output depends on whether k-NN is used for classification or regression:

In k-NN classification, the output is a class membership. An object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbor.

In k-NN regression, the output is the property value for the object. This value is the average of the values of k nearest neighbors K-Nearest Neighbors for Machine Learning

**KNN Model Representation**

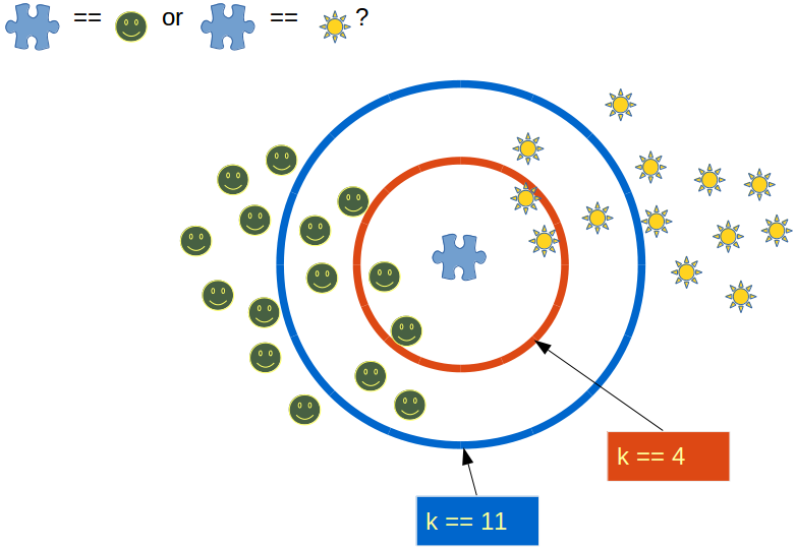
The model representation for KNN is the entire training dataset. It is as simple as that. KNN has no model other than storing the entire dataset, so there is no learning required.

Efficient implementations can store the data using complex data structures like k-d trees to make look-up and matching of new patterns during prediction efficient.

Because the entire training dataset is stored, you may want to think carefully about the consistency of your training data. It might be a good idea to curate it, update it often as new data becomes available and remove erroneous and outlier data.

**Making Predictions with KNN**

KNN makes predictions using the training dataset directly. Predictions are made for a new instance (x) by searching through the entire training set for the K most similar instances (the neighbors) and summarizing the output variable for those K instances. For regression this might be the mean output variable, in classification this might be the mode (or most common) class value. To determine which of the K instances in the training dataset are most similar to a new input a distance measure is used. For real-valued input variables, the most popular distance measure is Euclidean distance. The computational complexity of KNN increases with the size of the training dataset. For very large training sets, KNN can be made stochastic by taking a sample from the training dataset from which to calculate the K-most similar instances.



**Figure 2.1 KNN class**

KNN has been around for a long time and has been very well studied. As such, different disciplines have different names for it, for example:

**Instance-Based Learning:** The raw training instances are used to make predictions. As such KNN is often referred to as instance-based learning or a case-based learning (where each training instance is a case from the problem domain).

**Lazy Learning:** No learning of the model is required and all of the work happens at the time a prediction is requested. As such, KNN is often referred to as a lazy learning algorithm.

**Non-Parametric:** KNN makes no assumptions about the functional form of the problem being solved. As such KNN is referred to as a non-parametric machine learning algorithm.

KNN can be used for regression and classification problems.

**KNN for Regression**

When KNN is used for regression problems the prediction is based on the mean or the median of the K-most similar instances.

**KNN for Classification**

When KNN is used for classification, the output can be calculated as the class with the highest frequency from the K-most similar instances. Each instance votes for their class and the class with the most votes is taken as the prediction. Class probabilities can be calculated as the normalized frequency of samples that belong to each class in the set of K most similar instances for a new data instance.

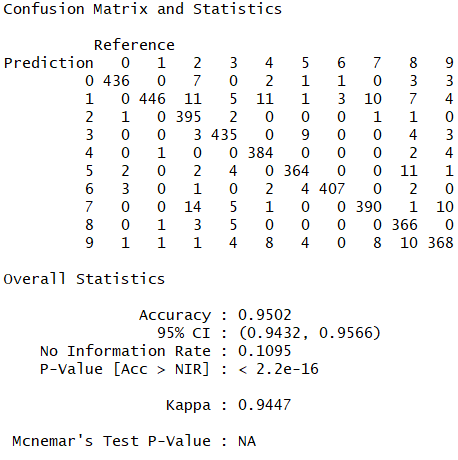
For example, in a binary classification problem (class is 0 or 1):

p(class=0) = count(class=0) / (count(class=0) +count(class=1))

If you are using K and you have an even number of classes (e.g. 2) it is a good idea to choose a K value with an odd number to avoid a tie. And the inverse, use an even number for K when you have an odd number of classes. Ties can be broken consistently by expanding K by 1 and looking at the class of the next most similar instance in the training dataset. A peculiarity of the k-NN algorithm is that it is sensitive to the local structure of the data.

**Model 1: Model from raw pixel values when k=25 (25 nearest neighbors)**

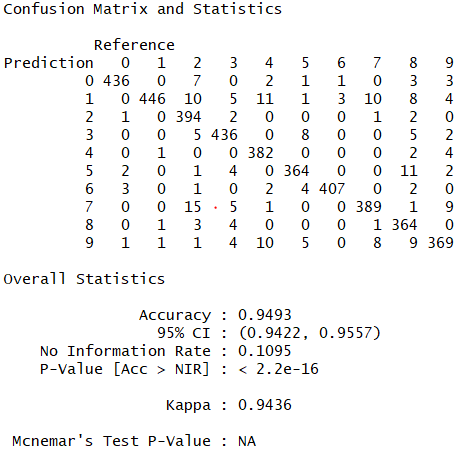
**Figure 2.2** shows the confusion of Knn algorithm by 25 nearest neighbors and no transformation applied on the dataset



**Figure 2.2 Consfusion Matrix for Model 1**

**Model 2: Model from DCT feature extraction on pixel values 1D using knn Algorithm when the value of k =25**

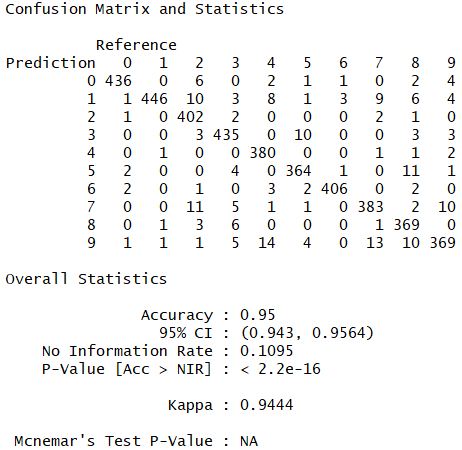
**Figure 2.3** shows the confusion matrix for knn algorithm with the value of k=25 and the pixel value of the dataset are transformed into a discreate cosine values before applying the algorithm



**Figure 2.3 Consfusion Matrix for Model 2**

**Model 3: Model from DCT feature extraction on pixel values 2D using knn Algorithm with the value of k =25**

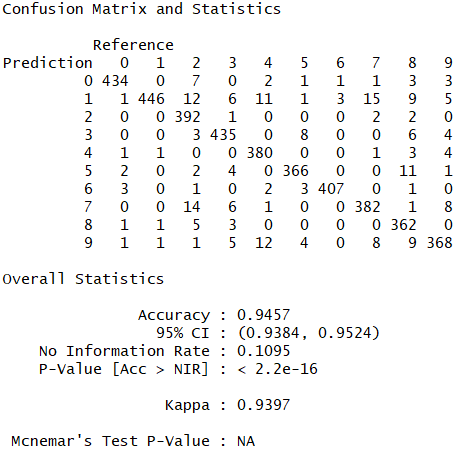
**Figure 2.4** shows the confusion matrix of Knn algorithm with the value of k=25 and the pixel value of the dataset are transformed into a 2D discreate cosine value matrix and transposed to an individual attribute before applying the algorithm



**Figure 2.4 Consfusion Matrix for Model 3**

**Model 4: Model from raw pixel values when k=35 (35 nearest neighbors)**

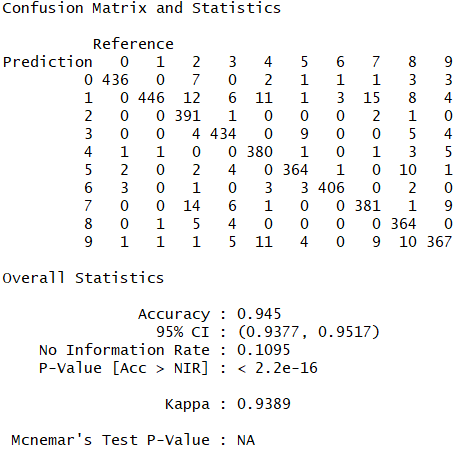
**Figure 2.5** shows the confusion of Knn algorithm by 35 nearest neighbors and no transformation applied on the dataset



**Figure 2.5 Consfusion Matrix for Model 4**

**Model 5: Model from DCT feature extraction on pixel values 1D using knn Algorithm when the value of k =35**

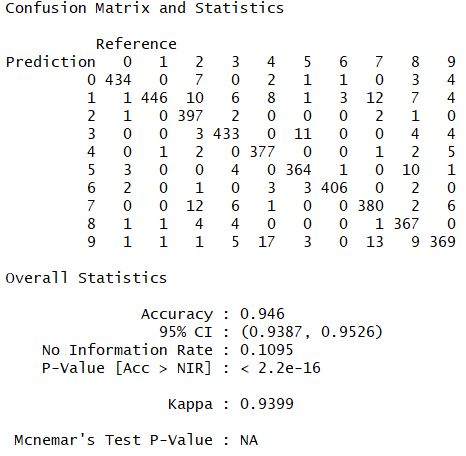
**Figure 2.6** shows the confusion matrix for knn algorithm with the value of k=35 and the pixel value of the dataset are transformed into a discreate cosine values before applying the algorithm



**Figure 2.6 Consfusion Matrix for Model 5**

**Model 6: Model from DCT feature extraction on pixel values 2D using knn Algorithm with the value of k =35**

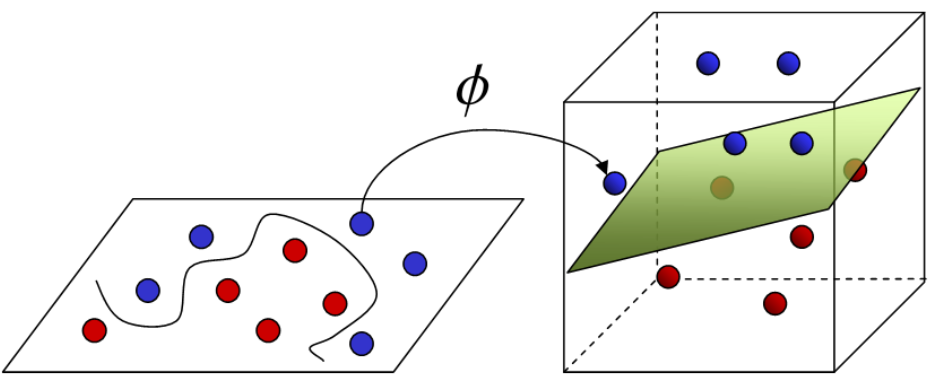
**Figure 2.7** shows the confusion matrix of Knn algorithm with the value of k=35 and the pixel value of the dataset are transformed into a 2D discreate cosine value matrix and transposed to an individual attribute before applying the algorithm



**Figure 2.7 Consfusion Matrix for Model 6**

#### **SVM (Support Vector Machine)**

A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data (supervised learning), the algorithm outputs an optimal hyperplane which categorizes new examples. In two-dimensional space this hyperplane is a line dividing a plane in two parts where in each class lay in either side.



**Kernel**

The learning of the hyperplane in linear SVM is done by transforming the problem using some linear algebra. This is where the kernel plays role. For linear kernel the equation for prediction for a new input using the dot product between the input (x) and each support vector (xi) is calculated as follows:

f(x) = B (0) + sum (ai \* (x, xi))

This is an equation that involves calculating the inner products of a new input vector (x) with all support vectors in training data. The coefficients B0 and ai (for each input) must be estimated from the training data by the learning algorithm. The polynomial kernel can be written as K(x,xi) = 1 + sum(x \* xi)^d and exponential as K(x,xi) = exp(-gamma \* sum((x — xi²)).

Polynomial and exponential kernels calculate separation line in higher dimension. This is called kernel trick

**Regularization**

The Regularization parameter (often termed as C parameter in python’s sklearn library) tells the SVM optimization how much you want to avoid misclassifying each training example. For large values of C, the optimization will choose a smaller-margin hyperplane if that hyperplane does a better job of getting all the training points classified correctly. Conversely, a very small value of C will cause the optimizer to look for a larger-margin separating hyperplane, even if that hyperplane misclassifies more points.

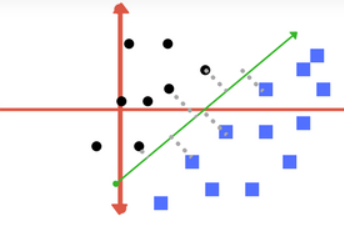
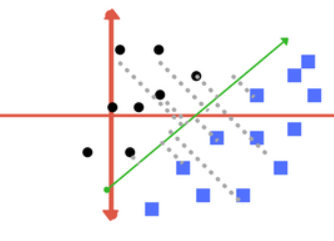
The images below (same as image 1 and image 2 in section 2) are example of two different regularization parameter. Left one has some misclassification due to lower regularization value. Higher value leads to results like right one.



**Left: low regularization value, right: high regularization value**

**Gamma**

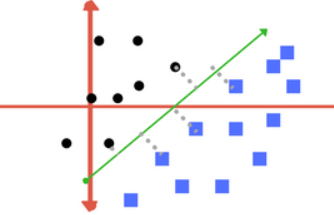
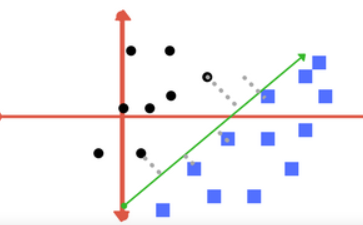
The gamma parameter defines how far the influence of a single training example reaches, with low values meaning ‘far’ and high values meaning ‘close’. In other words, with low gamma, points far away from plausible separation line are considered in calculation for the separation line. Whereas high gamma means the points close to plausible line are considered in calculation.

**High Gamma Low Gamma**

**Margin**

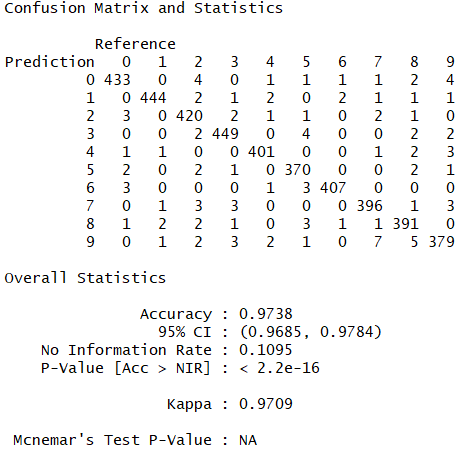
And finally, last but very important characteristic of SVM classifier. SVM to core tries to achieve a good margin. A margin is a separation of line to the closest class points. A good margin is one where this separation is larger for both the classes. Images below gives to visual example of good and bad margin. A good margin allows the points to be in their respective classes without crossing to other class.

**Good Margin Bad Margin**

**Model 1: Model from raw pixel values using SVM and polynomial kernel**

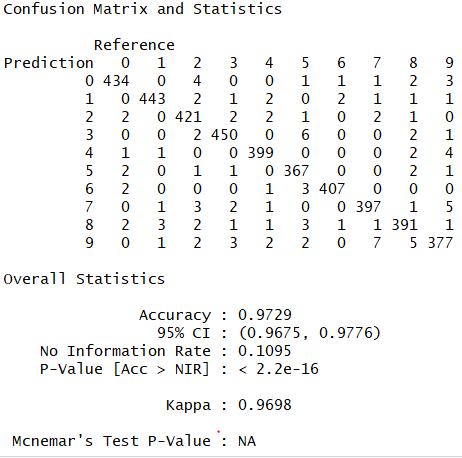
**Figure 2.8** shows the confusion matrix of the SVM algorithm and polynomial kernel. In this model original pixel values are used for model generation .



**Figure 2.8 Confusion Matrix for SVM-polynomial Model 1**

**Model 2: Model from DCT feature extraction on pixel values 1D using SVM and with polynomial kernel**

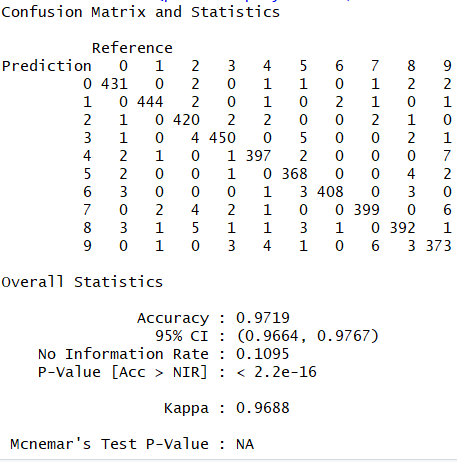
**Figure 2.9** shows the confusion matrix of the SVM and polynomial kernel algorithm. In this model original pixel values are transformed using Discrete Cosine Transformation (DCT) before applying SVM Algorithm.



**Figure 2.9 Confusion Matrix for SVM-polynomial Model 2**

**Model 3: Model from DCT feature extraction on pixel values 2D using SVM algorithm and polynomial kernel**

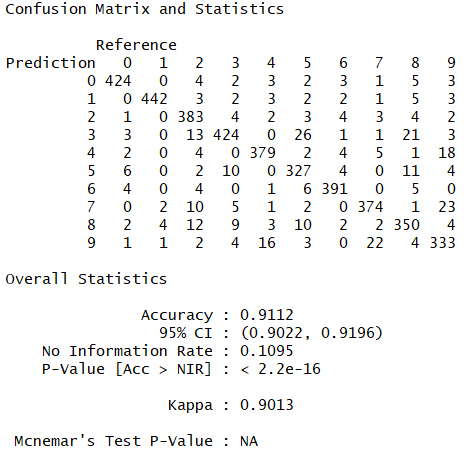
**Figure 2.10** shows the confusion matrix of the SVM algorithm and polynomial kernel. In this model original pixel values are transformed using Discrete Cosine Transformation (DCT) from a 2d pixel matrix before applying SVM Algorithm.



**Figure 2.10 Confusion Matrix for SVM polynomial Model 3**

**Model 4: Model from raw pixel values using SVM and linear kernel**

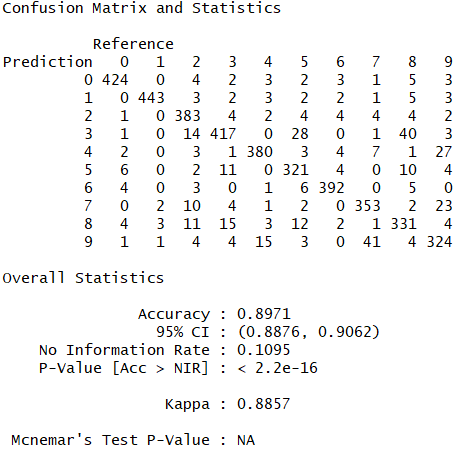
**Figure 2.11** shows the confusion matrix of the SVM algorithm and linear kernel. In this model original pixel values are used for model generation.



**Figure 2.11 Confusion Matrix for SVM-linear Model 4**

**Model 5: Model from DCT feature extraction on pixel values 1D using SVM and with linear kernel**

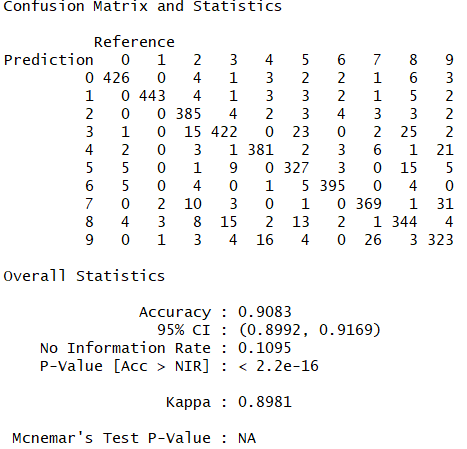
**Figure 2.12** shows the confusion matrix of the SVM and linear kernel algorithm. In this model original pixel values are transformed using Discrete Cosine Transformation (DCT) before applying SVM Algorithm.



**Figure 2.12 Confusion Matrix for SVM-linear Model 5**

**Model 6: Model from DCT feature extraction on pixel values 2D using SVM algorithm and linear kernel**

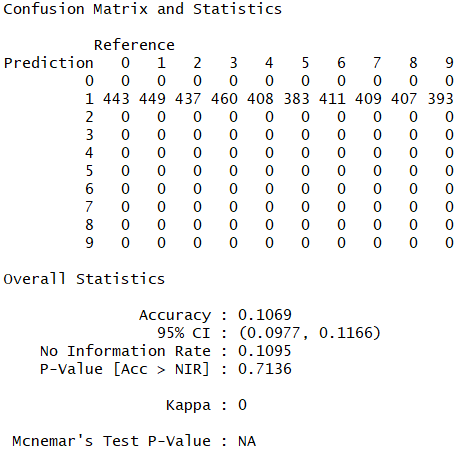
**Figure 2.13** shows the confusion matrix of the SVM algorithm and linear kernel. In this model original pixel values are transformed using Discrete Cosine Transformation (DCT) from a 2d pixel matrix before applying SVM Algorithm.



**Figure 2.13 Confusion Matrix for SVM linear Model 6**

**Model 7: Model from raw pixel values using SVM and radial kernel**

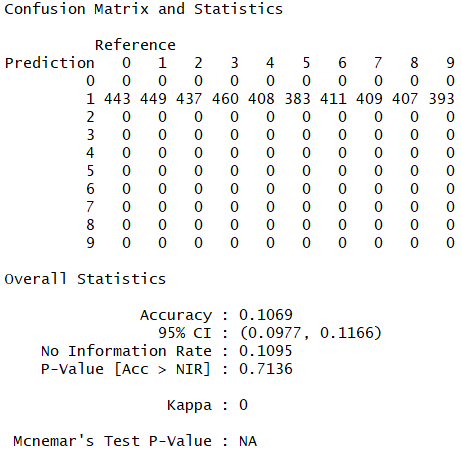
**Figure 2.14** shows the confusion matrix of the SVM algorithm and radial kernel. In this model original pixel values are used for model generation.



**Figure 2.14 Confusion Matrix for SVM-radial Model 7**

**Model 8: Model from DCT feature extraction on pixel values 1D using SVM and with radial kernel**

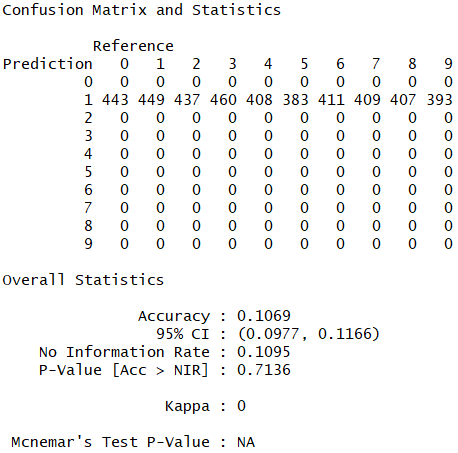
**Figure 2.15** shows the confusion matrix of the SVM and radial kernel algorithm. In this model original pixel values are transformed using Discrete Cosine Transformation (DCT) before applying SVM Algorithm.



**Figure 2.15 Confusion Matrix for SVM-radial Model 8**

**Model 9: Model from DCT feature extraction on pixel values 2D using SVM algorithm and radial kernel**

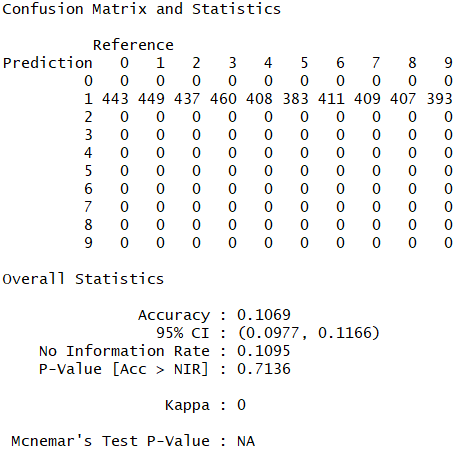
**Figure 2.16** shows the confusion matrix of the SVM algorithm and radial kernel. In this model original pixel values are transformed using Discrete Cosine Transformation (DCT) from a 2d pixel matrix before applying SVM Algorithm.



**Figure 2.16 Confusion Matrix for SVM radial Model 9**

**Model 10: Model from raw pixel values using SVM and sigmoid kernel**

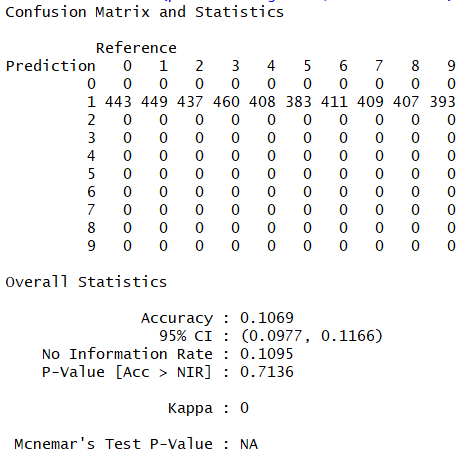
**Figure 2.17** shows the confusion matrix of the SVM algorithm and sigmoid kernel. In this model original pixel values are used for model generation.



**Figure 2.17 Confusion Matrix for SVM-sigmoid Model 10**

**Model 11: Model from DCT feature extraction on pixel values 1D using SVM and with sigmoid kernel**

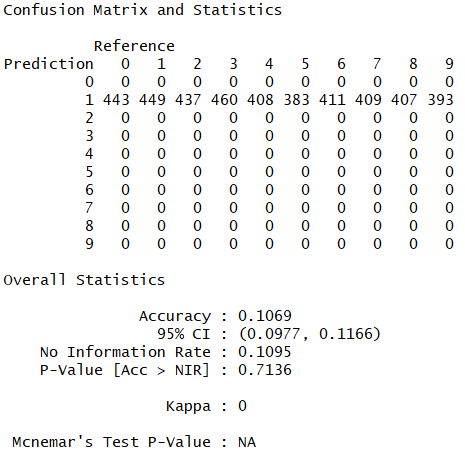
**Figure 2.18** shows the confusion matrix of the SVM and sigmoid kernel algorithm. In this model original pixel values are transformed using Discrete Cosine Transformation (DCT) before applying SVM Algorithm.



**Figure 2.18 Confusion Matrix for SVM-sigmoid Model 11**

**Model 12: Model from DCT feature extraction on pixel values 2D using SVM algorithm and sigmoid kernel**

**Figure 2.19** shows the confusion matrix of the SVM algorithm and sigmoid kernel. In this model original pixel values are transformed using Discrete Cosine Transformation (DCT) from a 2d pixel matrix before applying SVM Algorithm.

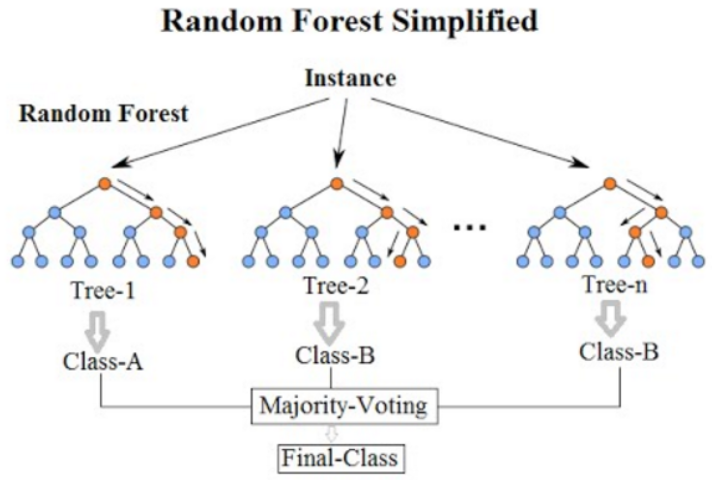


**Figure 2.19 Confusion Matrix for SVM sigmoid Model 12**

#### **RF (Random Forest)**

Random Forest is a supervised learning algorithm. Like its name, it creates a forest and makes it somehow random. The forest it builds, is an ensemble of Decision Trees, most of the time trained with the “bagging” method. The general idea of the bagging method is that a combination of learning models increases the overall result. Random forest builds multiple decision trees and merges them together to get a more accurate and stable prediction.

One big advantage of random forest is, that it can be used for both classification and regression problems, which form most current machine learning systems.



**Random Forest**

Random Forest has nearly the same hyperparameters as a decision tree or a bagging classifier. With Random Forest, you can also deal with Regression tasks by using the Random Forest regressor. Random Forest adds additional randomness to the model, while growing the trees. Instead of searching for the most important feature while splitting a node, it searches for the best feature among a random subset of features. This results in a wide diversity that generally results in a better model.

Therefore, in Random Forest, only a random subset of the features is taken into consideration by the algorithm for splitting a node. You can even make trees more random, by additionally using random thresholds for each feature rather than searching for the best possible thresholds (like a normal decision tree does).

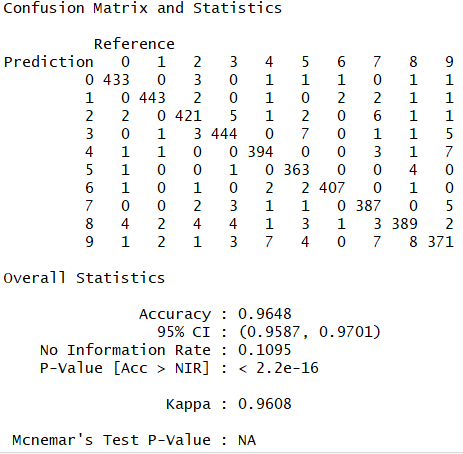
**Feature Importance:**

Another great quality of the random forest algorithm is that it is very easy to measure the relative importance of each feature on the prediction. Sklearn provides a great tool for this, that measures a features importance by looking at how much the tree nodes, which use that feature, reduce impurity across all trees in the forest. It computes this score automatically for each feature after training and scales the results, so that the sum of all importance is equal to 1.

Through looking at the feature importance, you can decide which features you may want to drop, because they don’t contribute enough or nothing to the prediction process. This is important, because a general rule in machine learning is that the more features you have, the more likely your model will suffer from overfitting and vice versa.

**Model 1: Model from raw pixel values using Random forest and default parameters**

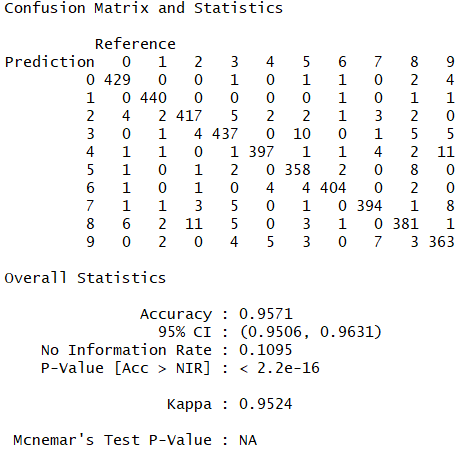
**Figure 2.20** shows the confusion matrix of the Random Forest algorithm and with default parameters. In this model original pixel values are used for model generation.



**Figure 2.20 Confusion Matrix for Random Forest Model 1**

**Model 2: Model from DCT feature extraction on pixel values 1D using Random Forest and with default parameters**

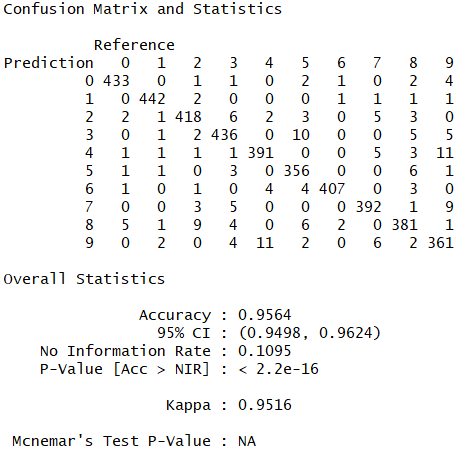
**Figure 2.21** shows the confusion matrix of the Random Forest and with default parameters. In this model original pixel values are transformed using Discrete Cosine Transformation (DCT) before applying Random Forest Algorithm.



**Figure 2.21 Confusion Matrix for Random Forest Model 2**

**Model 3: Model from DCT feature extraction on pixel values 2D using Random Forest with default parameters**

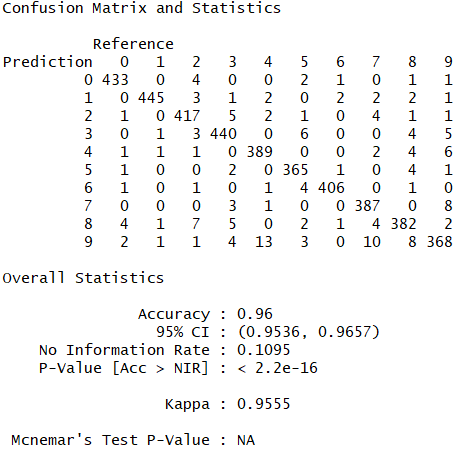
**Figure 2.22** shows the confusion matrix of the Random Forest with default parameters. In this model original pixel values are transformed using Discrete Cosine Transformation (DCT) from a 2d pixel matrix before applying Random Forest.



**Figure 2.22 Confusion Matrix for Random Forest Model 3**

**Model 4: Model from raw pixel values using Random forest with mtry=6**

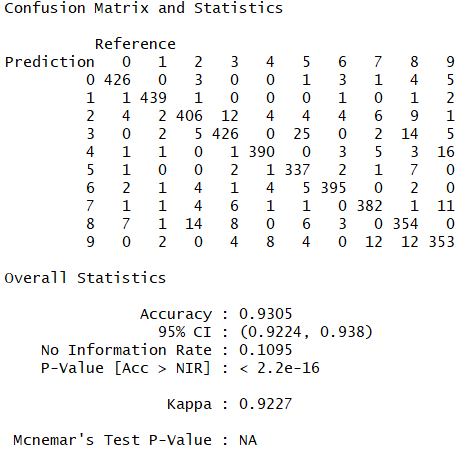
**Figure 2.23** shows the confusion matrix of the Random Forest algorithm and with mtry=6. In this model original pixel values are used for model generation.



**Figure 2.23 Confusion Matrix for Random Forest Model 4**

**Model 5: Model from DCT feature extraction on pixel values 1D using Random Forest and with mtry=6**

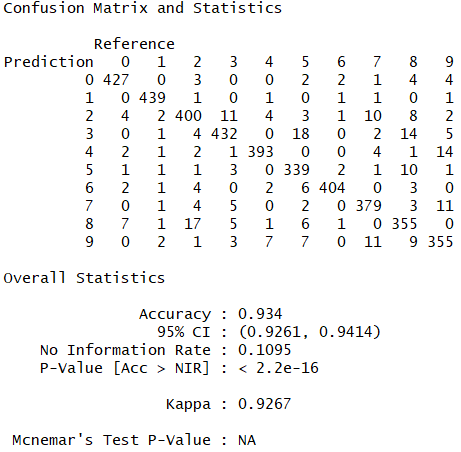
**Figure 2.24** shows the confusion matrix of the Random Forest and with mtry=6. In this model original pixel values are transformed using Discrete Cosine Transformation (DCT) before applying Random Forest Algorithm.



**Figure 2.24 Confusion Matrix for Random Forest Model 5**

**Model 6: Model from DCT feature extraction on pixel values 2D using Random Forest with mtry=6**

**Figure 2.25** shows the confusion matrix of the Random Forest with mtry=6. In this model original pixel values are transformed using Discrete Cosine Transformation (DCT) from a 2d pixel matrix before applying Random Forest.



**Figure 2.25 Confusion Matrix for Random Forest Model 6**

## **Results**

#### **kNN – k Nearest Neighbor**

**Model 1: Model from raw pixel values when k=25 (25 nearest neighbors)**

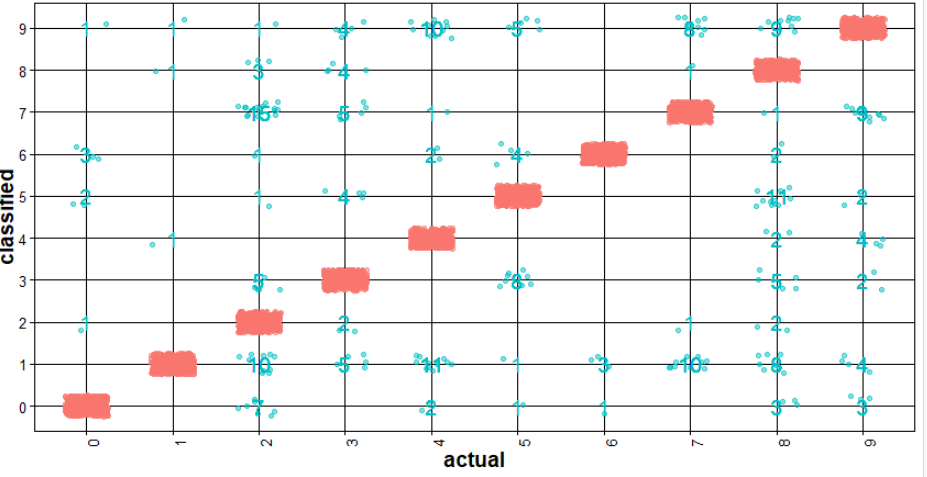
The accuracy of this model is about 95%. **Figure 3.1** shows the distribution of prediction error for each digit image.



**Figure 3.1 Accuracy of the kNN prediction Model 1**

**Model 2: Model from DCT feature extraction on pixel values 1D using kNN with k=25**

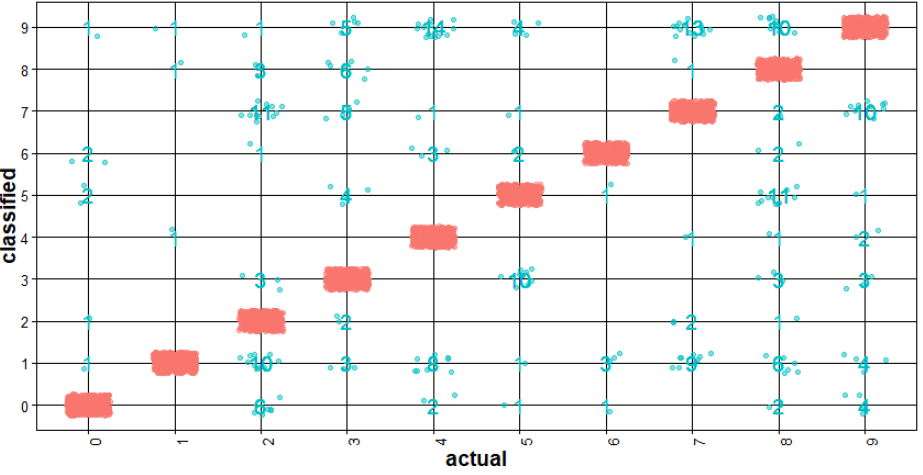
The accuracy of this model is about 94%. **Figure 3.2** shows the distribution of prediction error for each digit image.



**Figure 3.2 Accuracy of the kNN prediction in Model 2**

**Model 3: Model from DCT feature extraction on pixel values 2D using kNN and k =25**

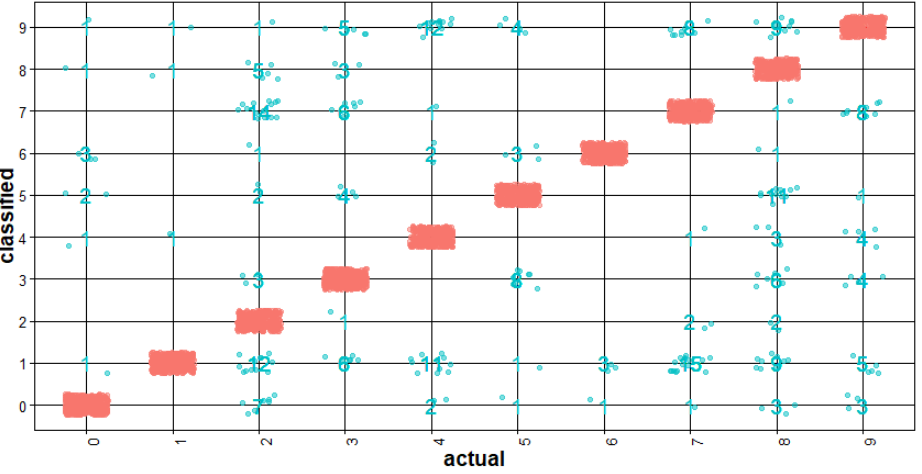
The accuracy of this model is about 95%. DCT feature generation from 2D matrix took same time as the feature generation from the previous model (features are extracted from a vector of pixel values in 1D). **Figure 3.3** shows the distribution of prediction error for each digit image.



**Figure 3.3 Accuracy of the kNN prediction in Model 3**

**Model 4: Model from raw pixel values when k=35 (35 nearest neighbors)**

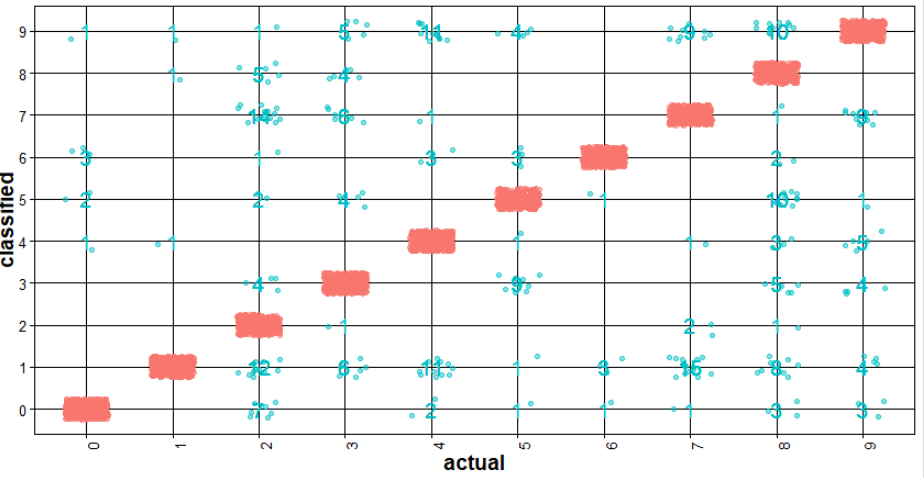
The accuracy of this model is about 94%. **Figure 3.4** shows the distribution of prediction error for each digit image.



**Figure 3.4 Accuracy of the kNN prediction Model 4**

**Model 5: Model from DCT feature extraction on pixel values 1D using kNN with k=35**

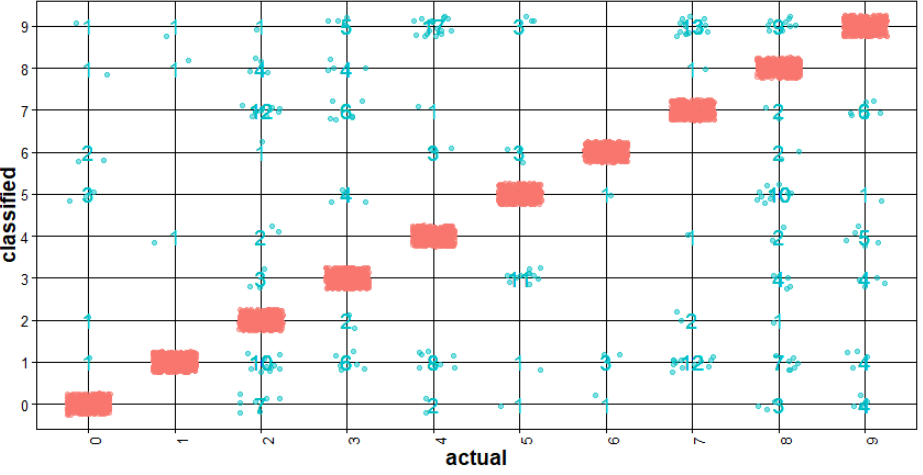
The accuracy of this model is about 94%. **Figure 3.5** shows the distribution of prediction error for each digit image.



**Figure 3.5 Accuracy of the kNN prediction in Model 5**

**Model 6: Model from DCT feature extraction on pixel values 2D using kNN and k =35**

The accuracy of this model is about 94%. DCT feature generation from 2D matrix took same time as the feature generation from the previous model (features are extracted from a vector of pixel values in 1D). **Figure 3.6** shows the distribution of prediction error for each digit image.

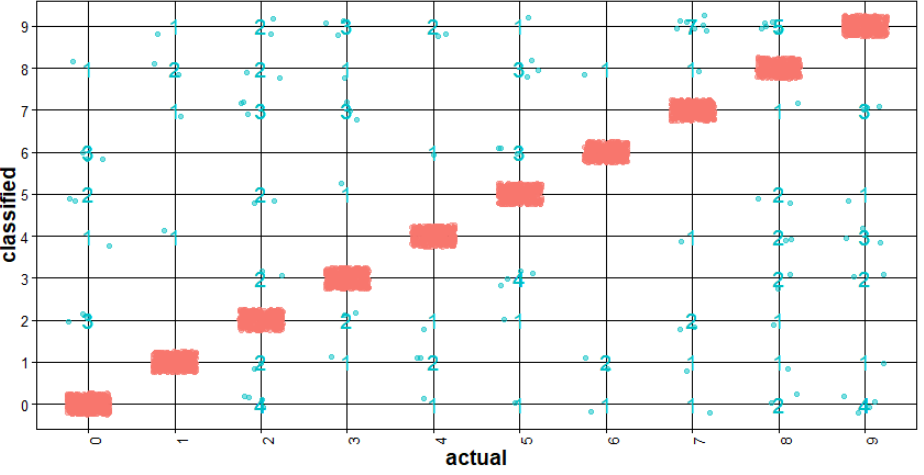


**Figure 3.6 Accuracy of the kNN prediction in Model 6**

#### **Support Vector Machine**

**Model 1: Model from raw pixel values using SVM and polynomial kernel**

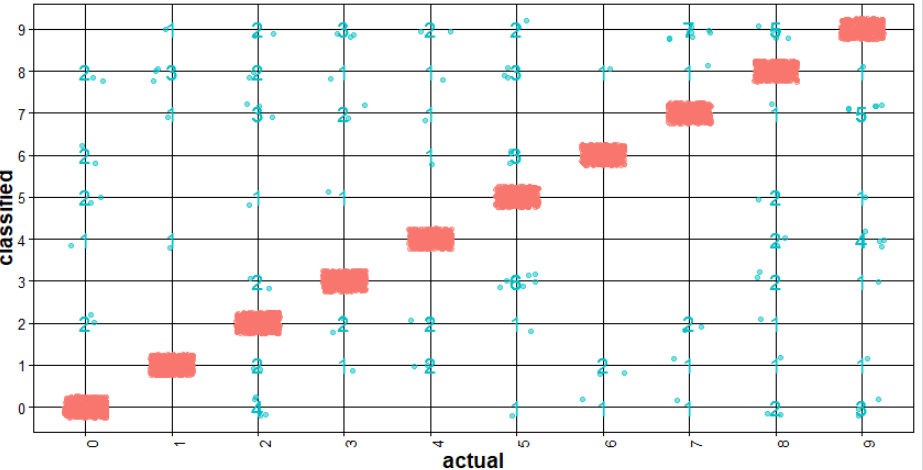
The accuracy of this model is about 97%. **Figure 3.7** shows the distribution of prediction error for each digit image.



**Figure 3.7 Accuracy of the SVM Model 1 predication**

**Model 2: Model from DCT feature extraction on pixel values 1D using SVM and polynomial kernel**

The accuracy of this model is about 97%. **Figure 3.8** shows the distribution of prediction error for each digit image.



**Figure 3.8 Accuracy of the SVM Model 2 predication**

**Model 3: Model from DCT feature extraction on pixel values 2D using SVM and polynomial kernel**

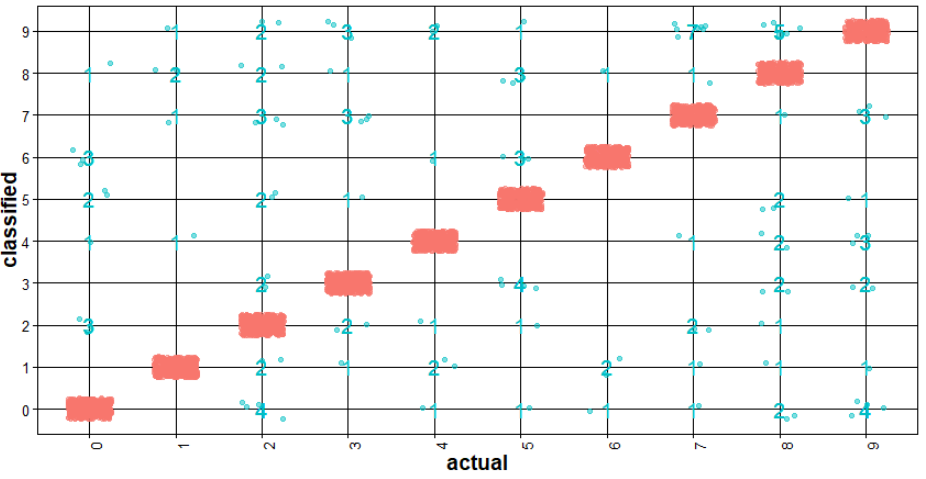
The accuracy of this model is about 97% which is same as the previous model. DCT feature generation from 2D matrix took same time as the feature generation from the previous model (features are extracted from a vector of pixel values in 1D). **Figure 3.9** shows the distribution of prediction error for each digit image.



**Figure 3.9 Accuracy of the SVM Model 3 predication**

**Model 4: Model from raw pixel values using SVM and linear kernel**

The accuracy of this model is about 97%. **Figure 3.10** shows the distribution of prediction error for each digit image.



**Figure 3.10 Accuracy of the SVM Model 4 predication**

**Model 5: Model from DCT feature extraction on pixel values 1D using SVM and linear kernel**

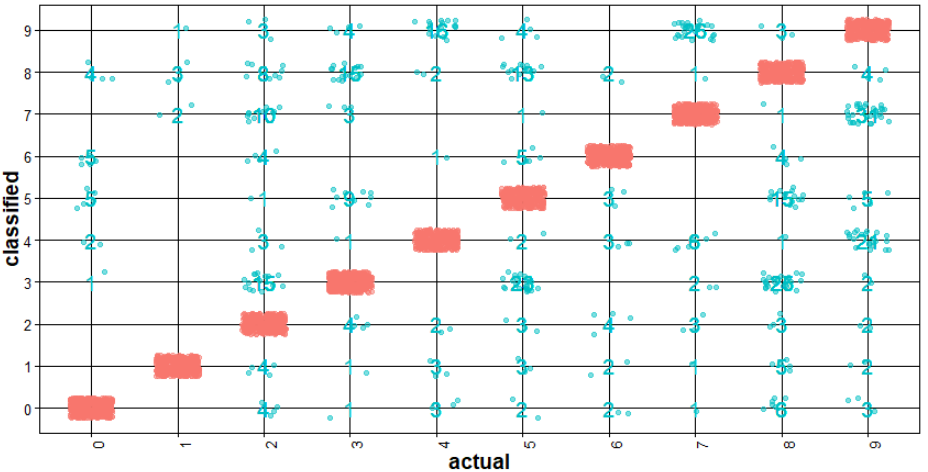
The accuracy of this model is about 89%. **Figure 3.11** shows the distribution of prediction error for each digit image.



**Figure 3.11 Accuracy of the SVM Model 5 predication**

**Model 6: Model from DCT feature extraction on pixel values 2D using SVM and linear kernel**

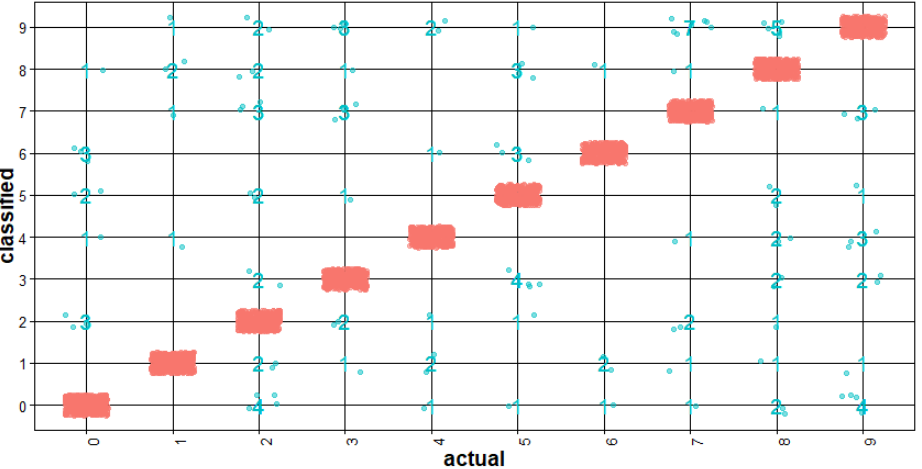
The accuracy of this model is about 90% which is almost same as the previous model. DCT feature generation from 2D matrix took same time as the feature generation from the previous model (features are extracted from a vector of pixel values in 1D). **Figure 3.12** shows the distribution of prediction error for each digit image.



**Figure 3.12 Accuracy of the SVM Model 6 predication**

**Model 7: Model from raw pixel values using SVM and radial kernel**

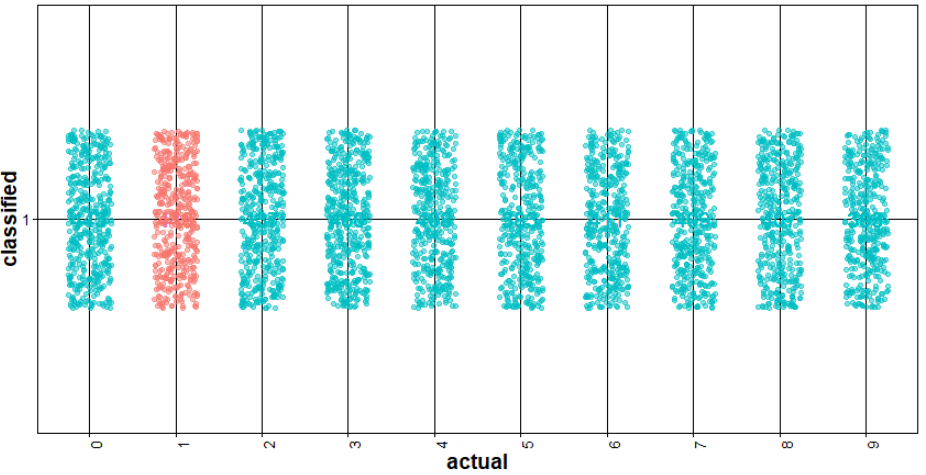
The accuracy of this model is about 97%. **Figure 3.13** shows the distribution of prediction error for each digit image.



**Figure 3.13 Accuracy of the SVM Model 7 predication**

**Model 8: Model from DCT feature extraction on pixel values 1D using SVM and radial kernel**

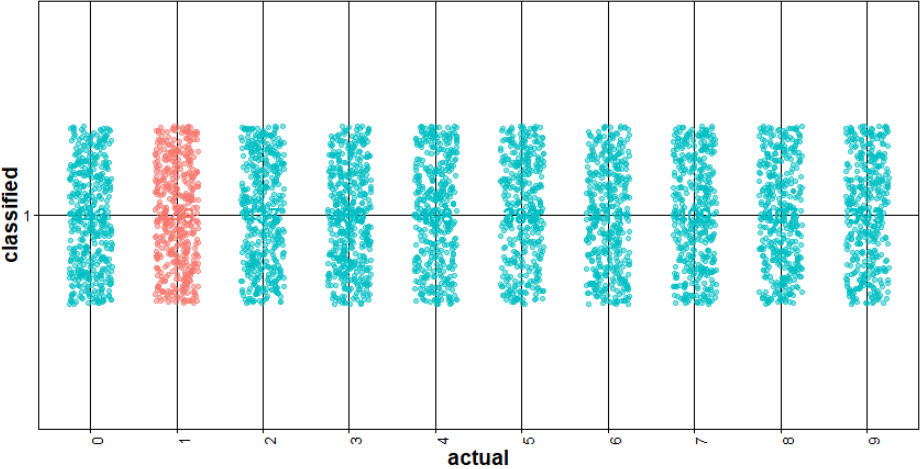
The accuracy of this model is about 10%. **Figure 3.14** shows the distribution of prediction error for each digit image. All of the observations are classified as 1 in this case



**Figure 3.14 Accuracy of the SVM Model 8 predication**

**Model 9: Model from DCT feature extraction on pixel values 2D using SVM and radial kernel**

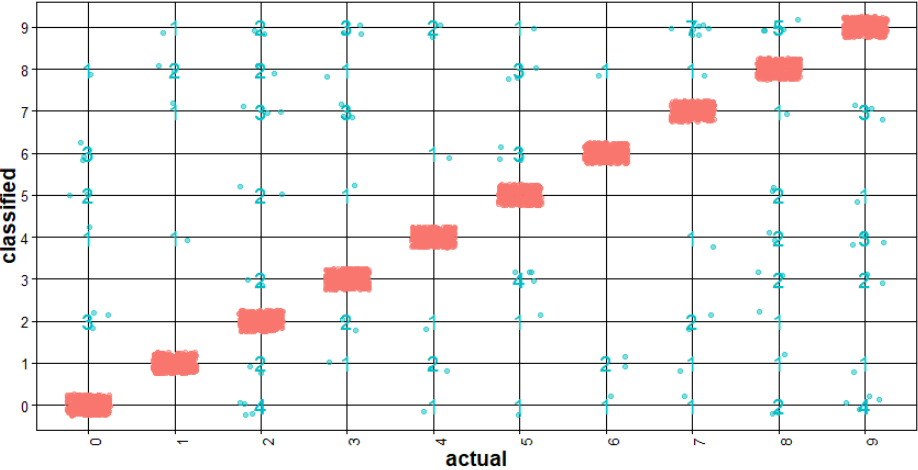
The accuracy of this model is about 10% which is same as the previous model. DCT feature generation from 2D matrix took same time as the feature generation from the previous model (features are extracted from a vector of pixel values in 1D). **Figure 3.15** shows the distribution of prediction error for each digit image. All of the observations are classified as 1 in this case



**Figure 3.15 Accuracy of the SVM Model 9 predication**

**Model 10: Model from raw pixel values using SVM and sigmoid kernel**

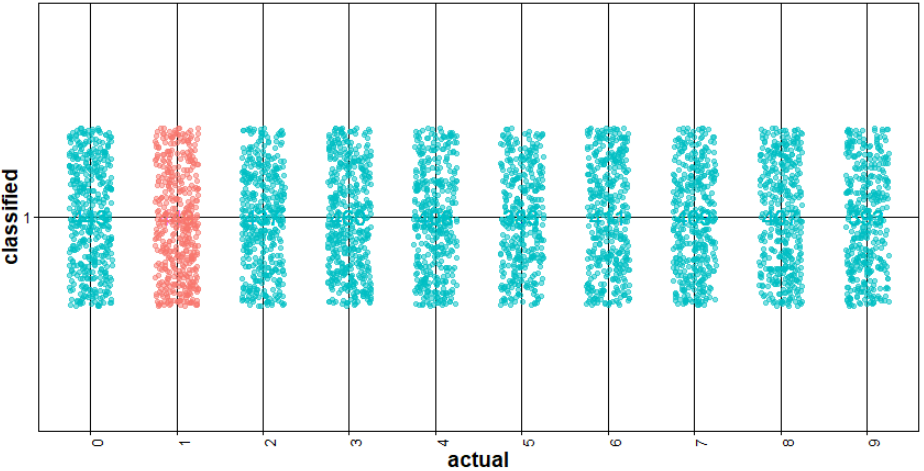
The accuracy of this model is about 97%. **Figure 3.16** shows the distribution of prediction error for each digit image.



**Figure 3.16 Accuracy of the SVM Model 10 predication**

**Model 11: Model from DCT feature extraction on pixel values 1D using SVM and sigmoid kernel**

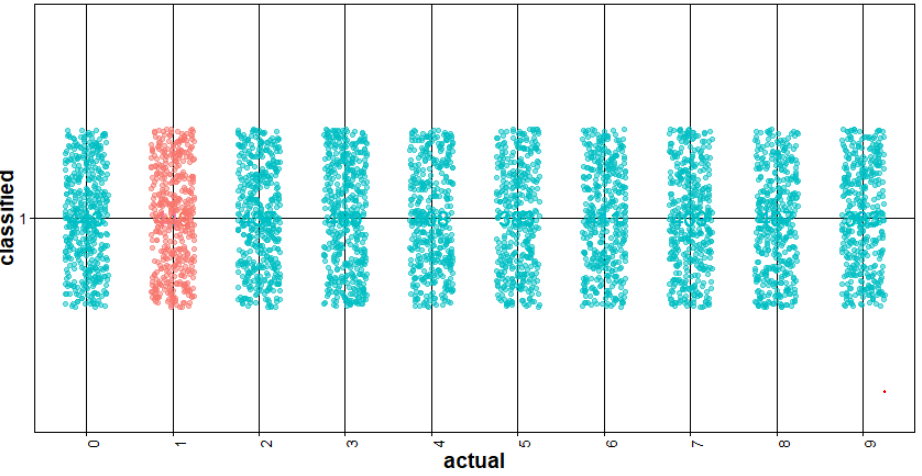
The accuracy of this model is about 10%. **Figure 3.17** shows the distribution of prediction error for each digit image. All of the observations are classified as 1 in this case.



**Figure 3.17 Accuracy of the SVM Model 11 predication**

**Model 12: Model from DCT feature extraction on pixel values 2D using SVM and sigmoid kernel**

The accuracy of this model is about 10% which is same as the previous model. DCT feature generation from 2D matrix took same time as the feature generation from the previous model (features are extracted from a vector of pixel values in 1D). **Figure 3.18** shows the distribution of prediction error for each digit image. All of the observations are classified as 1 in this case

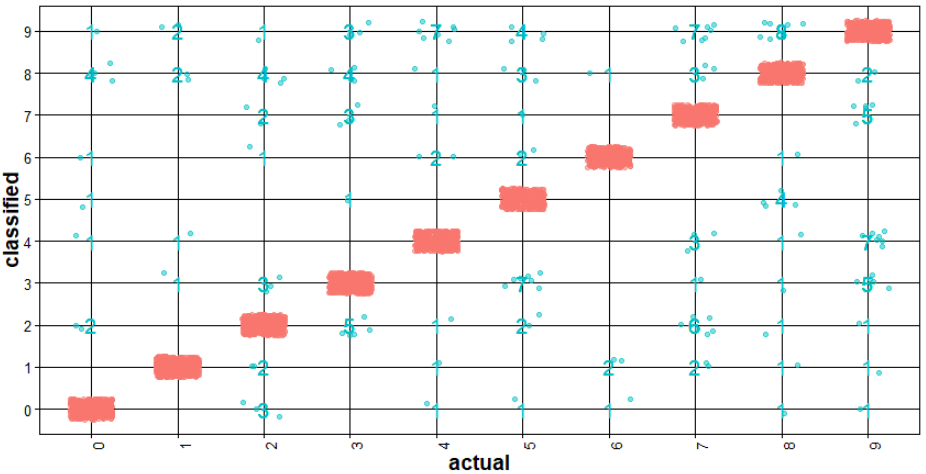


**Figure 3.18 Accuracy of the SVM Model 12 predication**

**Random Forest**

**Model 1: Model from raw pixel values using Random Forest with default parameters**

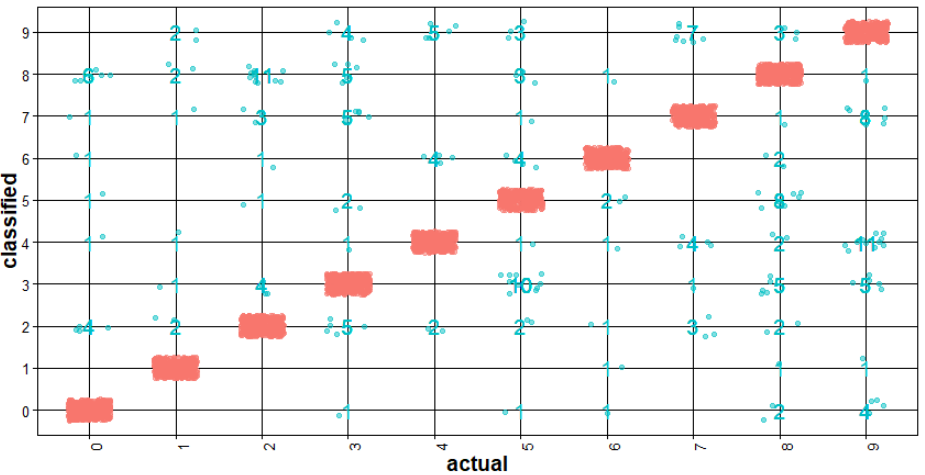
The accuracy of this model is about 96%. **Figure 3.19** shows the distribution of prediction error for each digit image.



**Figure 3.19 Accuracy of the Random Forest Model 1 predication**

**Model 2: Model from DCT feature extraction on pixel values 1D using Random Forest with default parameters**

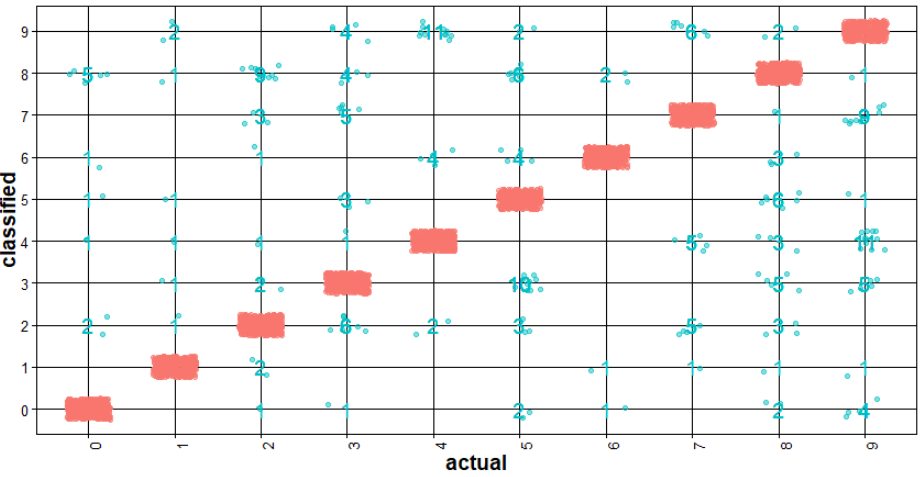
The accuracy of this model is about 95%. **Figure 3.20** shows the distribution of prediction error for each digit image.



**Figure 3.20 Accuracy of the Random Forest Model 2 predication**

**Model 3: Model from DCT feature extraction on pixel values 2D using Random Forest with default parameters**

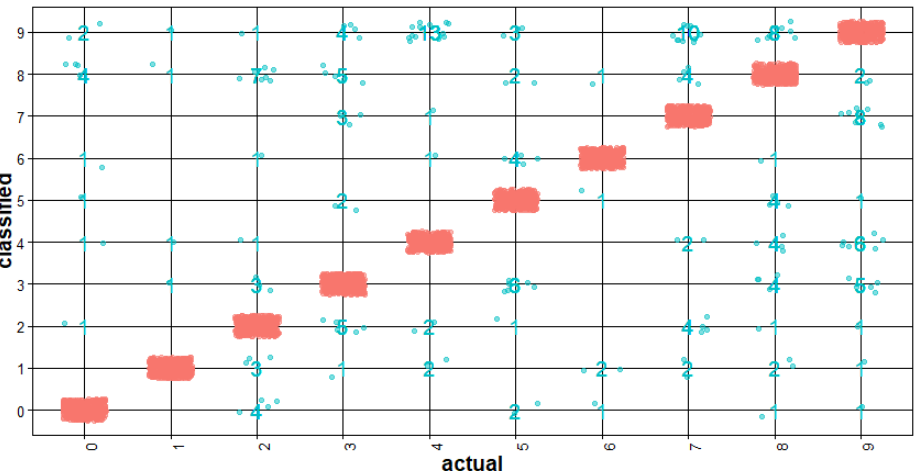
The accuracy of this model is about 95% which is same as the previous model. DCT feature generation from 2D matrix took same time as the feature generation from the previous model (features are extracted from a vector of pixel values in 1D). **Figure 3.21** shows the distribution of prediction error for each digit image. All of the observations are classified as 1 in this case



**Figure 3.21 Accuracy of the Random Forest Model 3 predication**

**Model 4: Model from raw pixel values** **using Random Forest with mtry=6**

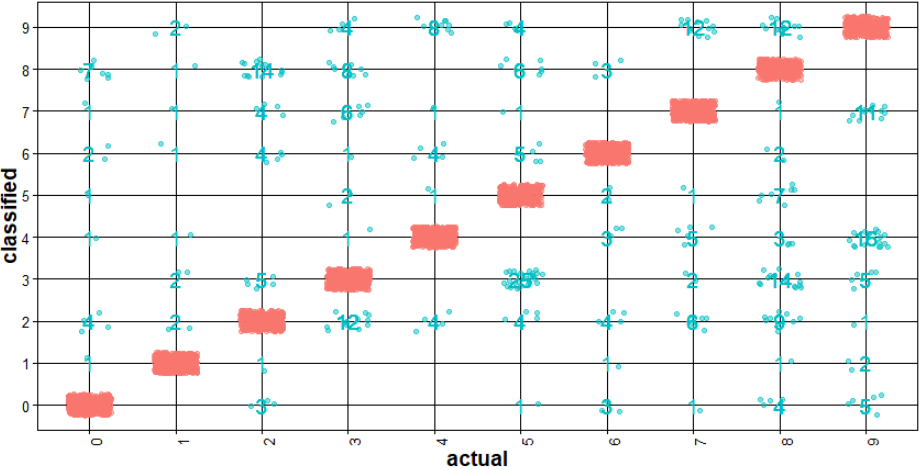
The accuracy of this model is about 96%. **Figure 3.22** shows the distribution of prediction error for each digit image.



**Figure 3.22 Accuracy of the Random Forest Model 4 predication**

**Model 5: Model from DCT feature extraction on pixel values 1D using Random Forest with mtry=6**

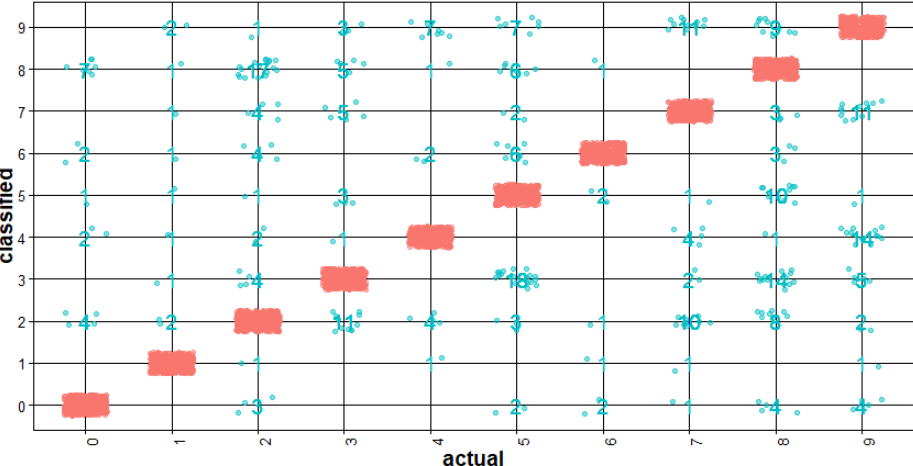
The accuracy of this model is about 93%. **Figure 3.23** shows the distribution of prediction error for each digit image. All of the observations are classified as 1 in this case.



**Figure 3.23 Accuracy of the Random Forest Model 5 predication**

**Model 6: Model from DCT feature extraction on pixel values 2D using Random Forest with mtry=6**

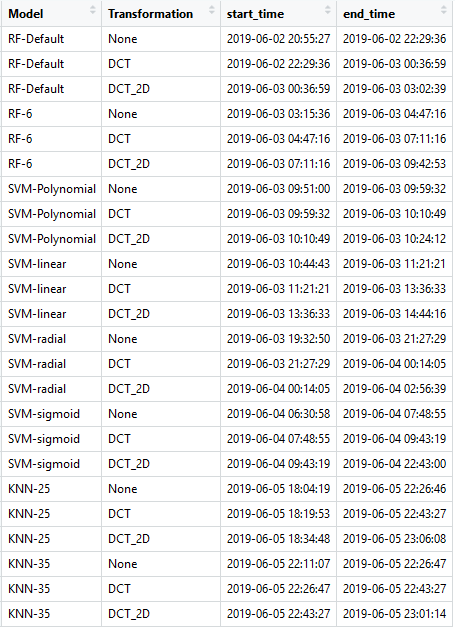
The accuracy of this model is about 93% which is same as the previous model. DCT feature generation from 2D matrix took same time as the feature generation from the previous model (features are extracted from a vector of pixel values in 1D). **Figure 3.24** shows the distribution of prediction error for each digit image. All of the observations are classified as 1 in this case



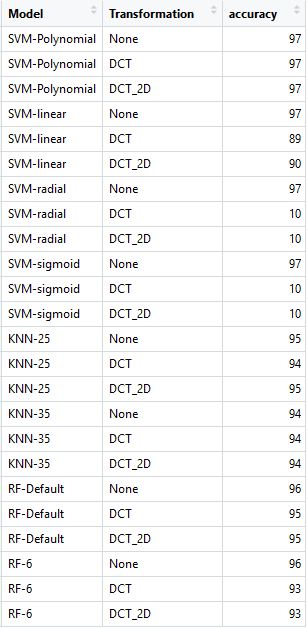
**Figure 3.24 Accuracy of the Random Forest Model 6 predication**

#### **Algorithm performance comparison**

After performing 3 cross validation, the average time took to build the model and perform the test prediction is shown in **Table 3.1.** Also, **Table 3.2** shows the average accuracy of the model for all the 3-cross validation



**Table 3.1 Model build time comparison**

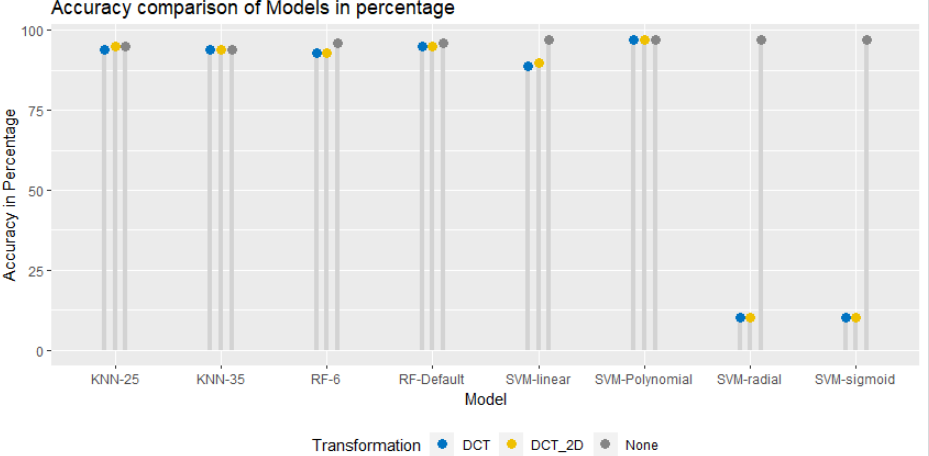


**Table 3.2 Model performance comparison**

## **Conclusion**

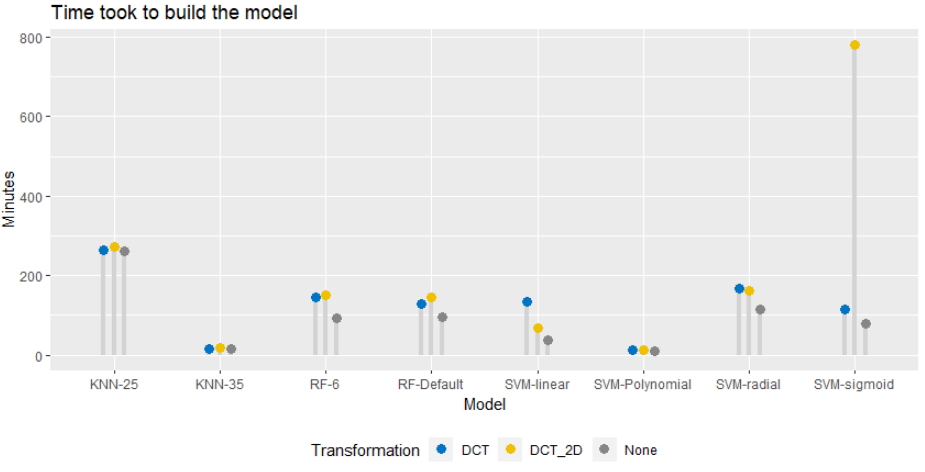
In the context of Image recognition, it is the ability of an algorithm to identify patterns in images and classify them under right buckets.

After comparing different algorithm such as SVM, Random Forest and kNN to identify written digits present in the image, it is observed that kNN, Random Forest and SVM are yielding approximately the same accuracy around 95 % to 97 %. Unlike the Decision Tree and Naïve Bayes, the raw data from the image file is yielding higher accuracy than the transformed data. Please refer **Figure 4.1** which compares the accuracy of different models and its accuracy.



**Figure 4.1 Accuracy comparison of Models**

There are more variations observed in build time for all these models as shown in **Figure 4.2**. there is an outlier for



**Figure 4.2 Performance comparison of Model build time**

The above inference shows that SVM algorithm giving maximum accuracy of 97% which is 2% higher than random forest and kNN algorithm. Discrete cosine transformation didn’t yield any better performance than the raw data.