Special Thanks To:

Dr. Guanjin Wang

*Murdoch University*

Project Report By:

Vajratiya Vajrobol (33296085)

Rajesh Jyothi (33669079)

ICT602 Advanced Data Analytics

Murdoch University

2020

# Part 1

# 1.1 Artifact 1 dataset

For a fixed value of gamma = 0.1

|  |  |
| --- | --- |
| **Fig 1.1.1 C= 0.001** | **Fig 1.1.2 C = 0.5** |
|  |  |
| **Fig 1.1.3 C = 1** | **Fig 1.1.4 C = 5.0** |
|  |  |
| **Fig 1.1.5 C = 10.0** | **Fig 1.1.6 C = 20.0** |
|  | **6** |
| **Fig 1.1.7 C = 50** | **Fig 1.1.8 C = 1000** |
|  |  |

C, the regularization parameter Controls the trade-off between the Low Training and Testing error. It tells us how much one needs to avoid the misclassification of training samples. For a large c value, the hyperplane gives smaller margin. And helps in classifying the training samples correctly. For a smaller C value, the hyperplane margin is larger and gives a higher misclassification rate.

In figure 1.1.1, for very small C value the whole dataset is classified as one set, leading to high misclassification rate. With the increase C values to 1, we can see that there is one outlier that is misclassified. When we increase the C value to 20, the outlier is correctly classified. This can be observed in figure 1.1.6.

* Large C: High variance and Low Bias (underfitting) – a small margin
* Small C: Low Variance and high Bias (Overfitting) – a large margin

# 1.2 Artifact dataset 2

For a fixed C = 10.0

|  |  |
| --- | --- |
| **Fig:1.2.1 gamma = 0.10** | **Fig:1.2.2 gamma = 0.15** |
|  |  |
| **Fig:1.2.3 gamma= 0.2** | **Fig:1.2.4 gamma = 1.0** |
|  |  |
| **Fig:1.2.5 gamma=10.0** | **Fig:1.2.6 gamma= 100.0** |
|  |  |

The gamma parameter tells how far the training samples to be reached.

* If the gamma value is large, then only closer values are considered for the decision boundaries.
* If the gamma value small, even the farthest training points are considered for the decision boundary.

From the above figures we can observe that for smaller gamma value, the decision boundary is smoother. Because the points that are close to the boundary have relatively low weight, thus results in a smooth curve.

Whereas for larger values of gamma, the points that are close to the boundary gain more weight, hence results in more non-linear or wiggly boundaries.

From figure 1.2.1 and 1.2.5, we can observe that the boundary is smoother in case 1 and wigglier in case 2.

* Too Large gamma: high bias and low variance.
* Too Small gamma: Low Bias and high variance.

# 1.3 Credit Card Fraud Detection

# Introduction

The remarkable revolution of the digitalization in the society has increased the used the use of the digital money transaction using credit and debit cards. This brings forth the new problem of cybercrime, introducing credit card fraud in society (Shukur & Kumaz, 2019). With the increased use of E-commerce website for shopping, the scammers playing around to intrude the network and access the online payment mode. This is the friendly target to the hackers. Australia alone recorded a fraud transition of $565 millions in the financial year 2017-18 (Mittal & Tyagi, 2019). The credit card fraud is of many types. According to the Australia Payments Network (APN), It can be in Card-Not-Present fraud, False Application Fraud, Lost and stolen etc.

In this report, we implement Machine Learning techniques on the fraud transaction data to detect the fraudulent transaction. For this, the data is collected from the bank, the details of the columns are not revealed due to the privacy issues. And the data is scaled already to hide the actual values. This dataset is collected from the Kaggle repository.

# Data Understanding

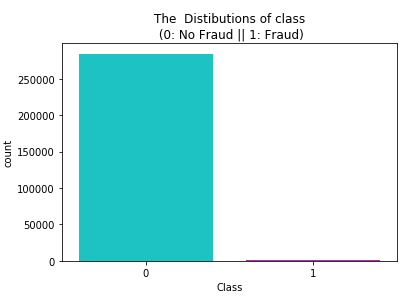
The data is collected from one of the reputed banks in the European nation. It has the credit card transaction history of 2 days in September month 2013. It has 284807 transition records and 31 attributes which includes V1 to V28, Time, Amount and Class. All the attributes are scaled due to privacy issue, except Time, Amount and Class. The Class variable is the target variable of categorical type "0" or "1". The Zero represents not a fraud transaction and 1 represents a fraud transaction. It has no missing data, and hence no missing data handling required.

## Problem statement:

It is a classification problem under supervised learning, as the dependent variable Class is a categorical, either 0 or 1. So, Supervised Machine Learning algorithms such as Decision Tree, Random Forest and SVM can be used

In the analysis, we implement the basic model of SVM and tune the hyperparameters using *GridSearchCV ()* method, to find the best parameters that can be used to get less error rate. These results are compared with the rest two models to find the best model for this data.

## Problems in the dataset:

1. This dataset is highly imbalanced, as the Fraudulent transactions are less than the Non-fraud transactions.

* *The percentage of the fraud transaction in the dataset: 0.17%*
* *The percentage of non-fraud transaction in the dataset: 99.83%*

1. Scaling issue: All the variables from V1 to V28 are scaled originally, but the attributes like Time and Amount are not scaled.

## Suggested Solutions:

1. The Unbalanced dataset can be handled by

* Oversampling the minority class
* Under-sampling the majority class
* Resampling training set etc

In this analysis, we implement the Under-sampling technique along with Stratified Cross-validation.

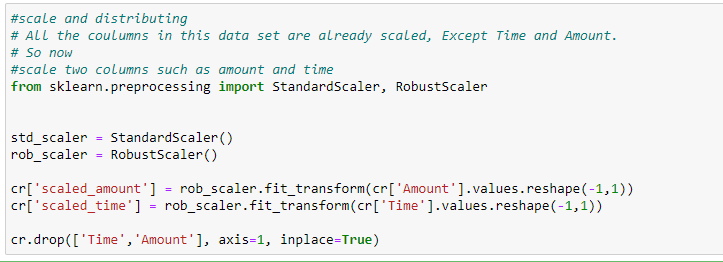
1. We scale the Amount and Time and concatenate the original dataset

# Data preparation

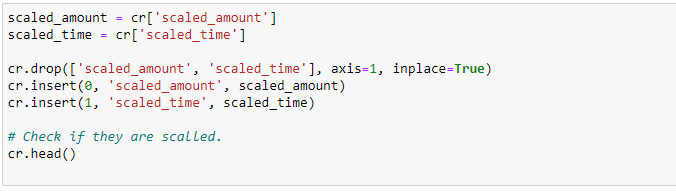
In this section, we prepare the data for modelling.

1. Scaling the Time and Amount attributes

As both the attributes are numerical, we scale them to [-1, 1].



After scaling the attribute, we concatenate to the original dataset and remove the Time and Amount unscaled variables.

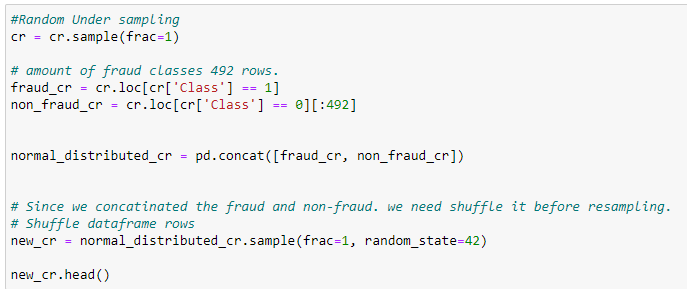


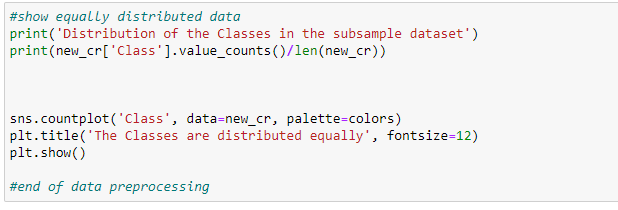
1. As the data suffers from the imbalance issue, we implement the under-sampling technique to balance the data.

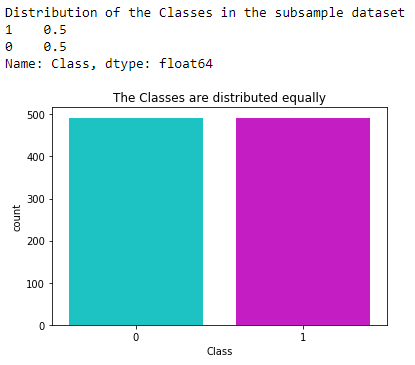
**Under Sampling:**

As the class imbalance is a critical issue in any classification problem, we implement better balancing techniques for the model to perform well. One such technique is under-sampling. In this, we randomly reduce the majority class from the dataset and form a new subset with an equal number of samples from two classes.

The Random Under sampling process is presented in the below code







# Data Modelling

The credit card data set is a supervised classification problem, so we implemented Decision Tree and KNN models, two Machine learning algorithms and compared with the SVM model by implementing the parameter tuning.

***Techniques implemented in the models***

### GridSearchCV ()

For the Performance tuning of the models, we implemented GridSearchCV () method. It is a hyperparameter tuning methodology implemented a machine-learning algorithm to evaluate the performance of the models. The tuning is based on the parameters present in the algorithm.

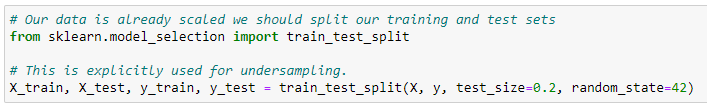
### Tuning SVM:

The SVM classifier is taken as the base model for this analysis and compared with other classification models such as Decision Tree and KNN models. The parameters in the SVM model, such as C, Kernel, and gamma value.

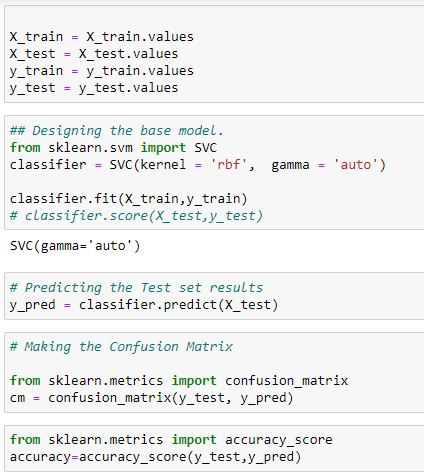
* C value: C is called a regularization parameter to Controls the trade-off between the Low Training and Testing error. Generally, C takes 1, 2, 10, 20, …
* Gamma value: Gamma value tells us how far the training samples to be searched. The general values for gamma are in decimals like 0.1, 0.2, 0.3 ….
* Kernel: Kernel tells us what methods to be implemented in the model, generally, the kernel can be linear, RBF, ploy etc.

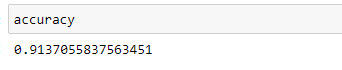
From the above analysis 1.1 and 1.2 artefacts, we observed the importance of these parameters in model design. So, tuning the parameters for checking each possible value is a time-consuming process. So, we implement these tunning mechanism using GridSearchCV.

Partitioning data



The Credit cards data set after Under-sampling is divided into training and testing tests in the ratio 80:20. Training data is used to train the model and testing data is used for testing the model.





After training and testing the model, SVM gave an accuracy of 91%. This results obtained are not from the default parameter setting of the model.

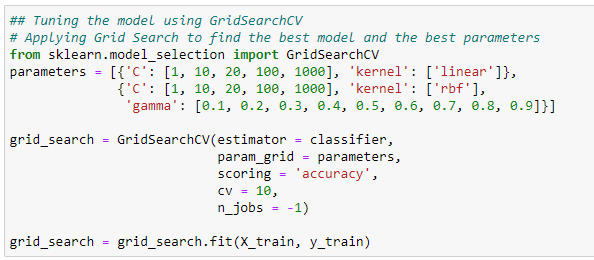
Let us now implement the hyperparameter tuning using GridSearchCV () method for SVM model.

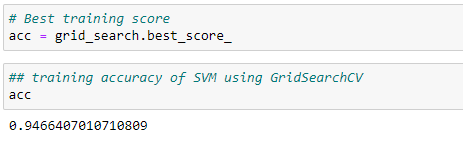
### Implementing the *GridSearchCV ()*

The GridSearchCV method has few parameters listed,

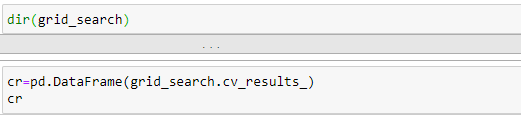
* **estimator**: To define the model we are implementing
* **param\_grid**: The hyperparameters we need to be tuned
* **Scoring**: Scoring function to be used in the model, here we use “accuracy”
* **CV**: The inner loop nested K fold Cross Validation used. here we implement K = 10
* **n\_jobs:** If n\_jobs is 1, then 100% of CPU is utilised of one core.

To tune the parameters for SVM model, we have given “C”, “Kernel” and “gamma” parameters to the param\_grid. Note that, for a linear Kernel we don’t give any gamma value. Gamma is given to rest of the Kernel values.

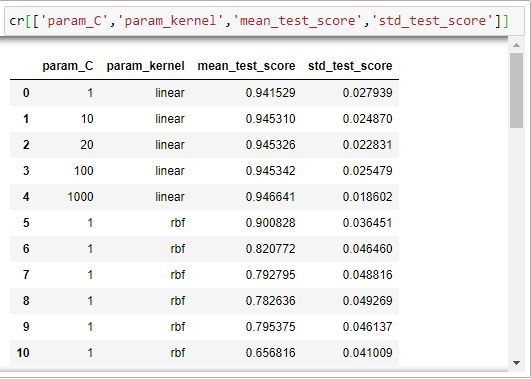




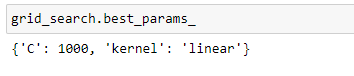
From the above tuning, we got an accuracy of 94.66%. this is not the accuracy of the test data, it is the accuracy of the training data tunned using 10 fold cross-validation.



Based on parameter tuning, the kernel suggests the best parameters to be used to get the least misclassification rate. Here are the few important parameters to be considered

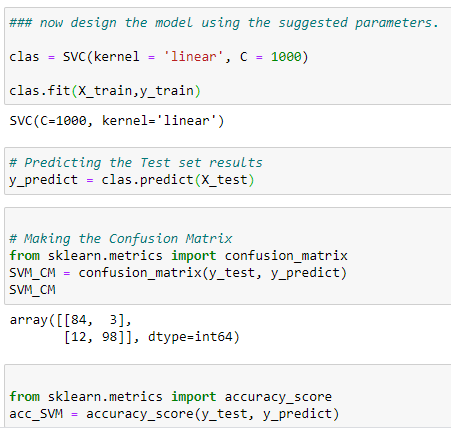


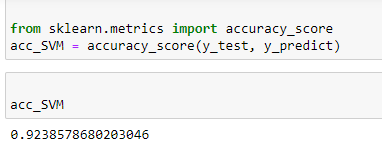
The tunned parameters for the SVM model are given by:



Now, these are the parameters to be used in the SVM model again to get better accuracy results.

### Implement the Hyperparameters in the model.

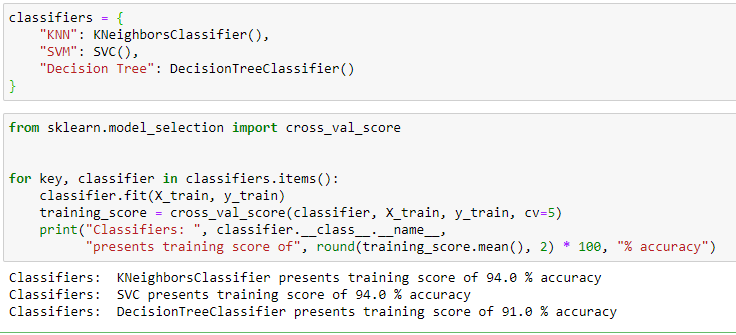




Using these tunned parameters, we achieved around 1 per cent higher than the accuracy using the default parameters.

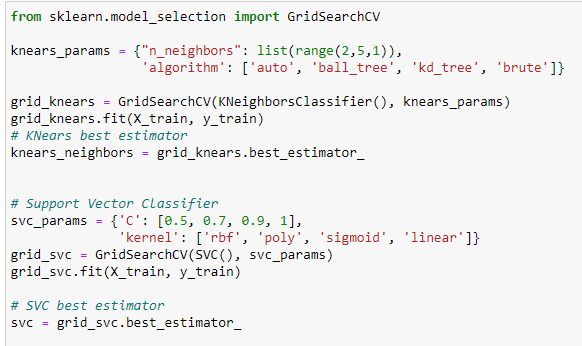
### Implementing other Machine learning algorithms

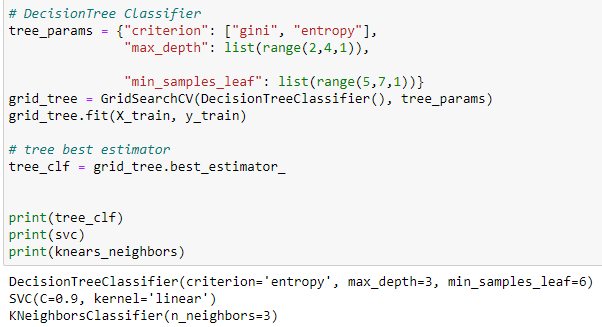
The Decision Tree and KNN models are implemented for the classification of Credit card fraud transaction. We turned them for the Hyperparameters to get better accuracy. Initially to evaluate the performance of the model on the training data set, for that, we implemented Cross\_val\_score (). This helps in understanding how well the model is trained on the training sets.



The Accuracy of the models on the training dataset is shown above. The KNN model and SVM models noted high training accuracy, compared to the Decision Tree classifier.

Now, we implement the Hyperparameter tuning on the models using GridSearchCV ()



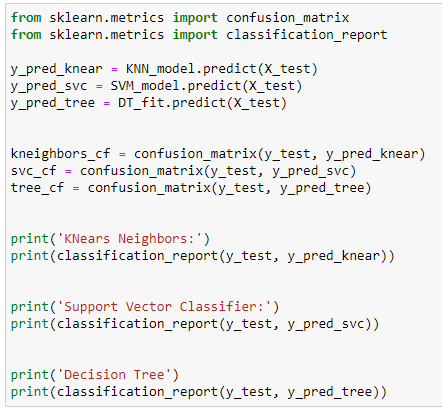


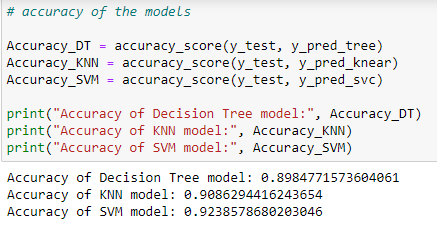
We noticed that GridSearchCV method suggested few parameters for the models, for which better accuracy can be achieved.

1. **Decision Tree**: criterion='entropy', max\_depth=3, min\_samples\_leaf=6
2. **SVM**: C=0.9, kernel='linear'
3. **KNN**: n\_neighbors=3

By using the above parameters let us build the models







The models are designed, and accuracy is observed as above.

# Interpretation of results

The classification models are designed by implementing the hyperparameter tuning. The accuracy of the models is as follows.

Table 1: Accuracy comparison of the models

|  |  |  |
| --- | --- | --- |
| **Models** | **Training Accuracy** | **Testing accuracy** |
| **KNN** | 94% | 90.86% |
| **Decision Tree** | 91% | 89.84% |
| **SVM** | 94% | 92.38% |

From the above Table 1 represents the accuracy of the training and testing data on the classification models. The Training accuracy is higher for the KNN and SVM classifier, whereas the Decision Tree noted very less training accuracy compared to the other two models. Coming to the testing accuracy of the models, after parameter tuning, SVM shows better accuracy than any other models with 92.38%. The Decision Tree and KNN differ by 1%. Overall, the SVM noted better training and testing accuracy than any other models. Though KNN showed better training accuracy, it shows poor performance on the testing data. This shows the KNN model is overfitting the training set.

As the Credit Card dataset has a class imbalance issue, it is not always recommended to consider accuracy as the performance matrix. So, another performance matrix such as Precision, Recall and F1 score should be considered. Table 2 below shows the results of these.

Table 2: Shows the Precision, Recall and F1 score for all the tree models

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Models** | **class** | **Precision** | **Recall** | **F1 Score** |
| **KNN** | 0 | 0.84 | 0.98 | 0.9 |
| 1 | 0.98 | 0.85 | 0.91 |
| **Decision Tree** | 0 | 0.83 | 0.98 | 0.89 |
| 1 | 0.98 | 0.84 | 0.9 |
| **SVM** | 0 | 0.88 | 0.95 | 0.92 |
| 1 | 0.96 | 0.9 | 0.93 |

These values are obtained from the confusion matrix:

* Precision = TP/TP+FP
* Recall = TP/TP+FN
* F1 Score = 2\*(Recall \* Precision) / (Recall + Precision)

The model is said to be a good classifier, if the Precision, Recall and F1 score are higher and close to 1. For easy understanding let us interpret one of these. From the above table, the precision values are higher in classifying the fraud truncation, than the non-fraud transaction with 0.98 and 0.84 respectively. The Decision tree also showed similar results in classifying. On the other hand, the SVM model classifies the fraud cases pretty good, and non-fraud as well with 0.88. This makes the SVM model better than any other models compared here.

# Part 2: TensorFlow and Neural Networks

## MNIST dataset

MNIST stands for Modified National Institute of Standard and Technology, is a database collection of digital handwritten images, with 60,000 training and 10,000 testing samples collections. These images are grayscale. The number from 1 to 9 are the handwritten and digital image of them are store with 28 X 28 pixels.

Overview of the model design:

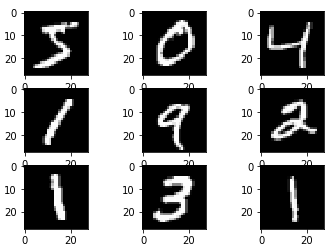
We load the MNIST dataset using the Keras and TensorFlow packages in Jupiter notebook. Check the sample images of the dataset, using a 3 X 3 matrix. By default, the dataset has a separate training and testing samples, so we separate the sets. A baseline or prototype model is build using the CNN model adding basic layers to it. This model is tested for accuracy. Later CNN model is optimized by adding more layers to the CNN model, such as *BatchNormalization, pooling layer with 64 filters, adding ReLU activation function, Flatten, Dropout to minimize the overfitting and output layer with 10 nodes to capture the 10 classes using the Softmax activation function.*

# Base model:

## Define the model

The MNIST dataset is first divided into train and test samples. The sample image of the data set is shown below Figure 1

Figure 1: Sample image of the dataset

* The shape of the training set is (60000, 28, 28) and the shape of the test set is (10000, 28, 28).
* We reshape the training and testing set to the 4-dimension array to work with Keras API.
* Later we converted the data type to float 32, as we are expecting the decimal values in the later stage.
* Normalize the training and testing sets by dividing with 255, it is the maximum value of the RBG value.
* Defining the CNN Model with the following layers.

The model has 2 main features: The front end with feature extraction containing Convolutional and max-pooling layers and the backend layer for prediction.

For the front-end CNN layer, we have a single layer of convolution with a filter of small size (3, 3) and 28 modest filters. It is followed by the maxpolling2D layer of pool size (2, 2). It is followed by the flatten layer to implement the classifier features.

To predict the multiclass classification of the data, we add an output layer with 10 nodes to capture 10 classes. We use the *softmax* activation function to achieve this. In between two features, the input convolution and output layer, we add a dens layer with 128 nodes using ReLU (Rectified Linear Unit)

***Pooling layer:***

The pooling layer is one of the building blocks of the Convolution Neural Networks (CNN). Its main functionality is to reduce the spatial size of the input data. It reduces the number of parameters, therefore reduces the computation process in the CNN network. Most commonly, Max pooling is used.

***Flattering layer:***

The flattening layer converts the 2-dimensional image into an array or vector, that can be fed as input to the Dens network layer.

***Dense layer***

It is the fully connected Neural network for processing an array which fed as input from the Flattering layer to the output layer. ReLU activation function is used to connect to the output layer. We implemented a dense layer with 128 nodes.

***ReLU activation:***

The Rectified Linear Unit vanishes the gradient problem and helps in faster learning of the model with better performance. It is one of the most commonly used activation function in the CNN and MLP models.

***Softmax Activation Function***

The Softmax activation function helps in converting the Logits scores into a probability distribution. The equation of the function is as follows.

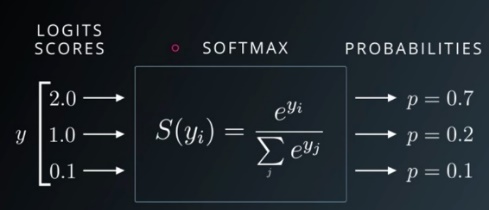
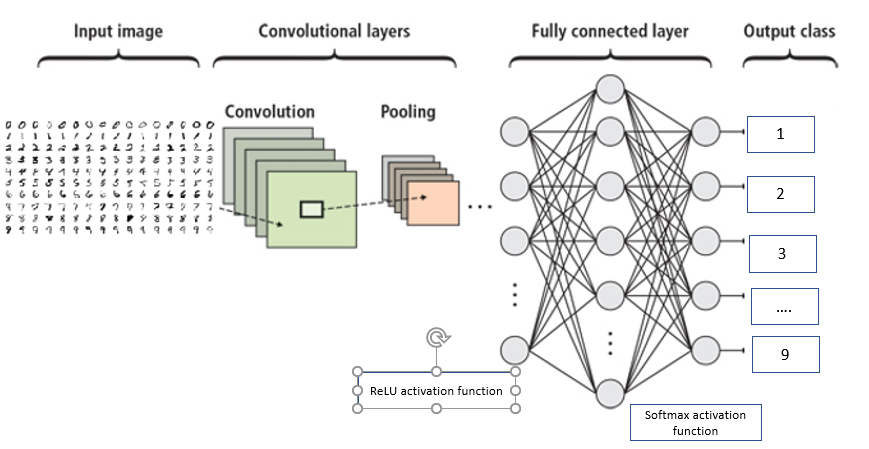


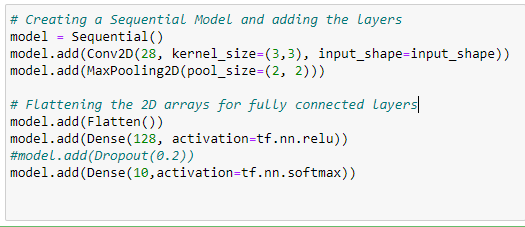
Figure 2: source: https://medium.com

## Compile the Model

The architecture diagram is shown below.



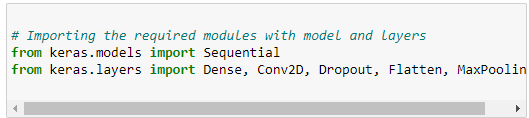
The Code is as follows

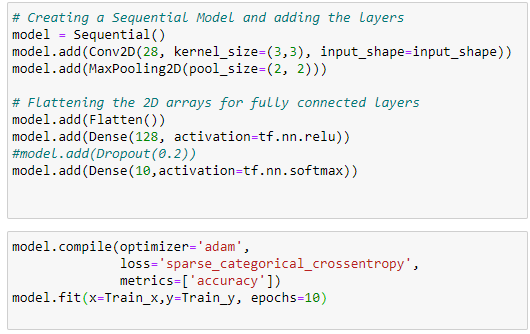


All the layers are implemented in a sequential format. To evaluate the time taken by the model, we add the time function as well.

## Fit the Model

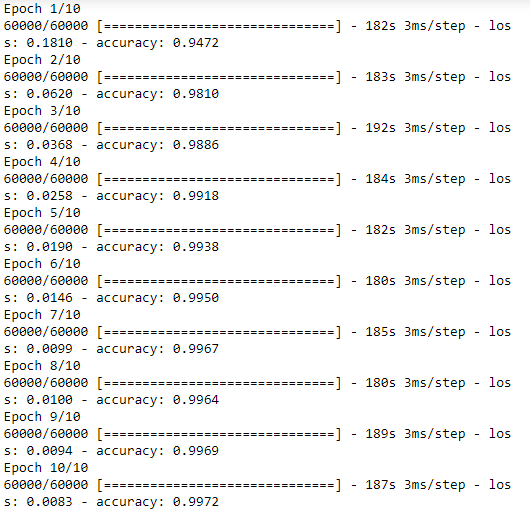
Importing the requires packages





The above shown is the training accuracy of the models for each Epoch.

## Evaluate the Model



The Base model is run for 10 Epoch, in each epoch, the training accuracy of the model are noted. shown in below Table 3:

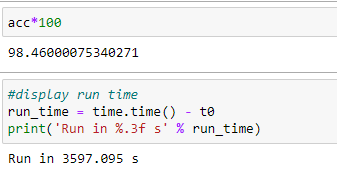
Table 3: Training accuracy for each Epoch

|  |  |
| --- | --- |
| **Number of Epochs** | **Training accuracy** |
| **1** | 94.48 |
| **2** | 97.86 |
| **3** | 98.55 |
| **4** | 98.57 |
| **5** | 99.22 |
| **6** | 99.35 |
| **7** | 99.37 |
| **8** | 99.62 |
| **9** | 99.6 |
| **10** | 99.57 |

From the above table, the training accuracy for the Base model with the basic layer is getting improved from Epoch to Epoch. The results show that the model is well trained on the training data set.

## Make Predictions

After the model is evaluated and trained with the training data, the model is tested with the validation set or testing set.



The accuracy of the base model is 98.46% for the unseen data. This is a good start for a base model. The misclassification rate is 1.54%. The misclassification rate can be further reduced using optimization of CNN model in the next section.

The time taken for the model is around 3597.1secs which is approximately 1 hour.

# Optimised Model

In the process of optimization, we have added a few more hidden layers to the CNN model.

The sequence of the layers is as follows:

* Layer 1: CNN input layer using ReLU activation function
* Layer 2: Batch Normalization
* Layer 3: MaxPooling2D
* Layer 4: CNN layer using ReLU activation function
* Layer 5: CNN input layer using ReLU activation function
* Layer 6: MaxPooling2D
* Layer 7: Flatten
* Layer 8: Dense layer with 100 nodes and ReLU activation function
* Layer 9: Batch Normalization
* Layer 10: Drop Out with 0.2
* Layer 11: Dense layer with 10 nodes, this is the output layer.



Along with the additional layers, we have implemented the Stochastic Gradient Descent optimizer with a learning rate of 0.01 on the CNN model.

Some of the layers are implemented in the basic model as well, let us now see the new layers added.

1. **Batch Normalization:**

It is a technique used to train the deep neural networks by standardizing the input layers. The helps in stabilizing the model learning process and reduce the training epochs dramatically in deep networks.

1. **Drop Out**

It can be implemented in almost all the layers of the neural network hidden layers. It can also be added in the input layers but cannot use on the final layer or output layer. It refers to dropping out units in the neural networks. The overfitting issue in the deep layers can be handled easily using the Dropout layer.

The value for Dropout cannot be taken randomly, So, a better approach is trial and error method. and general thumb rule is practice is, dividing the number of nodes in the previous layer with the proposed dropout rate. And assigning the resulted number as nodes in the next layer along with the dropout. In this CNN model design, we have taken 0.2 are dropout rate.

1. **SGD optimizer:**

It addresses mainly used for fitting the classifier with the cost function. It is an optimization method to estimate the gradient error for the current state, using the training set. Then it updates the weights in the model using the Backpropagation algorithm.

The model learning depends on the learning rate assigned in the SGD. If the learning rate is higher, the model learns faster, and if the learning rate is very less, it makes take a long time to learn for the model. So optimal learning rate value is recommended. It is a good practice to check from the least values and then increases the value based on the performance of the model.

Table 4: The Training accuracy of 5-Fold CV

|  |  |
| --- | --- |
| **Num of folds** | **Training accuracy for 5-Fold CV** |
| Fold 1 | 98.883 |
| Fold 2 | 98.925 |
| Fold 3 | 99.142 |
| Fold 4 | 99.1 |
| Fold 5 | 98.925 |

From the above training Cross-validation accuracy, we can observe that the model is well trained for the training data. To check if it is affected by the Overfitting of the model, let us check the cross-entropy plots.

These curves help us in understanding the underfitting and underfitting of the model to train and test sets. The first plot in this denotes the misclassification error rate of the model and the other represents the accuracy of the models for 5-Fold Cross-validation.

These two plots give the learning behaviour of the model for the 5-fold cross-validation. In the model, it is observed that the model achieved a good fit. There is no sign of overfitting od the model is observed in the plot for the training and testing.

**Note:**

* The blue lines indicate the model behaviour to the training set
* The Orange line indicated the model behaviour to the testing set.

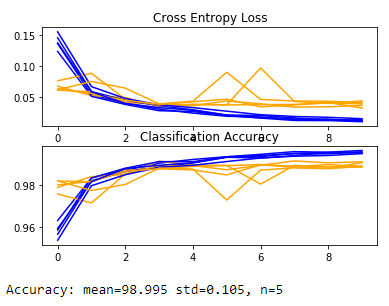


Figure 3: Learning curves

The Standard Deviation of the models is 0.105 and the mean of the model accuracy is observed to be 98.995 for 5-Fold CV.

# Compare the Results:

The CNN base model and its optimization model are built by adding the extra hidden layers to the CNN model. These results are compared with the Simple Logistic Regression model as shown in Table 3 below.

Table 5: Accuracy and Time comparison if the models

|  |  |  |  |
| --- | --- | --- | --- |
| **Models** | **Accuracy** | **Time is taken in secs** | **Time is taken in mins** |
| **Base Model** | 98.46 | 3597.095 | ~ 60 |
| **Optimized Model** | 99.33 | 25203.071 | ~ 421 |
| **Logistic Regression** | 88.86 | 63.65 | ~ 1 |

From the above table, the optimized CNN model with added hidden layers performed well with 99.33 % testing accuracy. This is the best accuracy ever recorded for the whole experiment. The Logistic regression showed the least accuracy rate with 88.86%, which means they miss classification rate in detecting the digits is very high. The Base model noted pretty good results with the training set, whereas, for the testing data, it shows a bit less accuracy. From the results, we can conclude that the Logistic regression is not a good model in classifying multiclass classification problems. Where are Neural networks performs well for multi classes?

When time takes by the model is considered, I have noticed that Logistics regression has not taken much time compared to another model. More noticeable, the optimized model has taken 421 mins to run the model, which is 7 hours, to run the model. On the other hand, the Base model has taken 1 hr to run the model. The computational power also matters to run the model. This result is tested in 8 GB RAM, Pentium processor, CPU 5405U with 2.30 GHz clock speed and windows 10 64bit environment.

# References:

Mittal, S., & Tyagi, S. (2019, January). Performance evaluation of machine learning algorithms for credit card fraud detection. In *2019 9th International Conference on Cloud Computing, Data Science & Engineering (Confluence)* (pp. 320-324). IEEE.

Shukur, H. A., & Kurnaz, S. (2019). Credit Card Fraud Detection using Machine Learning Methodology.