# SELECTED TOPICS IN COMPUTER SCIENCE ASSIGNMENT 1

Course ID: CS F441

Prajjwal Vijaywargiya: 2017B3A70954H

Parth Krishna Sharma: 2017B3A70907H

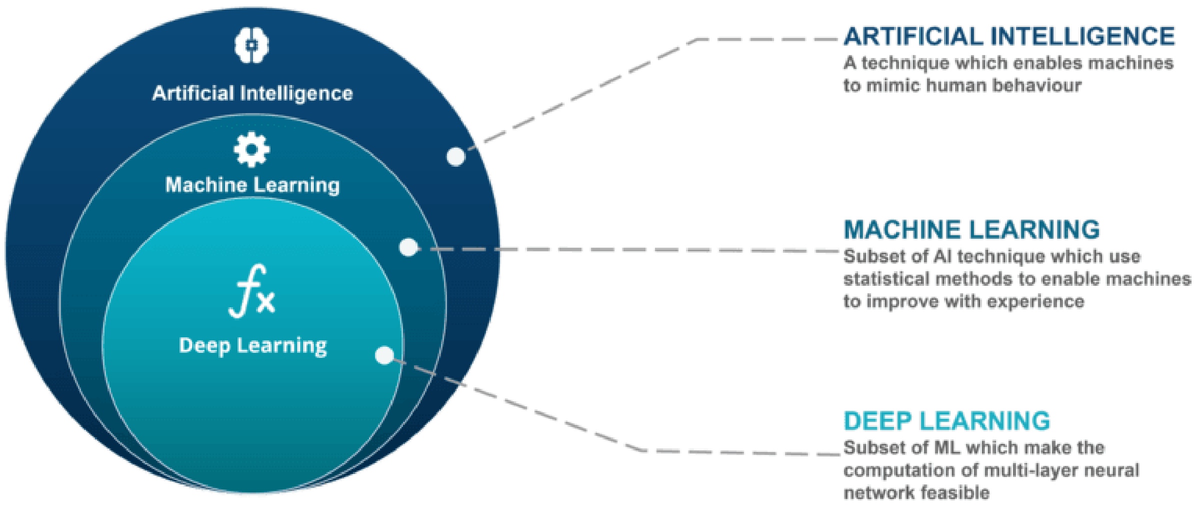
Siddhi Burse: 2017B3A70972H

OCTOBER 30, 2020

## Introduction

Machine Learning is the branch of Artificial Intelligence where Machine Learning algorithms are built in such a way that the machine attempts to learn rules and patterns between inputs and outputs without being directly programmed. So, as models are subjected to more and more data, they attempt to internally modify themselves and adapt according to the input data, so that it does not have to rely on human expertise to configure it.

Deep Learning is the evolved form of machine learning. It uses a framework inspired by the biological neural network and has a layered architecture, called an artificial neural network. These algorithms are trained to recognise patterns and classify the information to provide optimal output based on the input. Deep learning algorithms automatically extract classification features that need a large amount of data for deep learning algorithms. Deep learning models have many hyperparameters that can be tuned, such as the activation functions, number of layers, number of units (or neurons) in each layer, optimizers, etc.


Artificial Intelligence
Machine Learning
Deep Learning
ARTIFICIAL INTELLIGENCE
A technique which enables machines to mimic human behaviour
MACHINE LEARNING
Subset of AI technique which use statistical methods to enable machines to improve with experience
DEEP LEARNING
Subset of ML which make the computation of multi-layer neural network feasible

Although simple machine learning models are steadily getting better at whatever their task is, they still need some guidance. An engineer needs to step in and make corrections if an ML algorithm returns an incorrect prediction. In case of a deep learning model, the algorithm can determine whether its own prediction is accurate or not with the help of its own neural network.

As a part of this assignment, we will be doing a comparative analysis of the predictive models built using supervised machine learning algorithms and models built using deep learning algorithms.

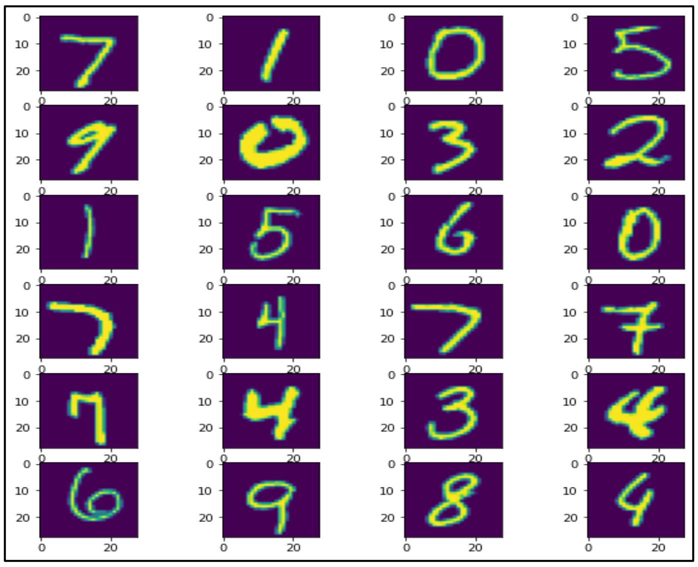
## Introduction to dataset used

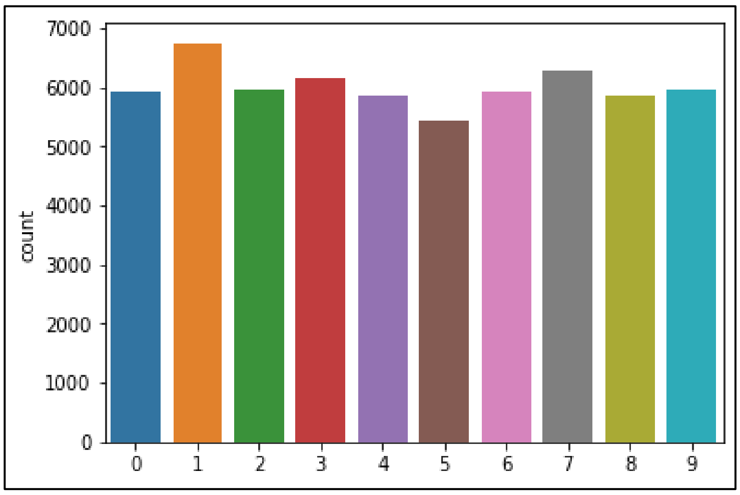
The dataset used for the classification is from a database of MNIST's. The MNIST database (Modified National Institute of Standards and Technology Database) is a massive handwritten digit database that is widely used to train different systems for image processing. The database is also commonly used in the area of machine learning for teaching and research.

There are 60,000 training images and 10,000 testing images in the MNIST dataset. Half of the training set and test set were taken from the training dataset of NIST, while the other half of the training set and the second half of the test set were taken from the test dataset of NIST.

The *mnist_train.csv* file includes 60,000 instances and labels for training. 10,000 test examples and labels are contained in the *mnist_test.csv*. Every row is made up of 785 values: the label is the first value (a number from 0 to 9) and the pixel values are the remaining 784 values (a number from 0 to 255).

The below figures show sample images from the dataset, and the frequency of each class label (digits) in the training set.:





# Machine learning models

The 4 Machine Learning algorithms used for building the classification model are as follows:

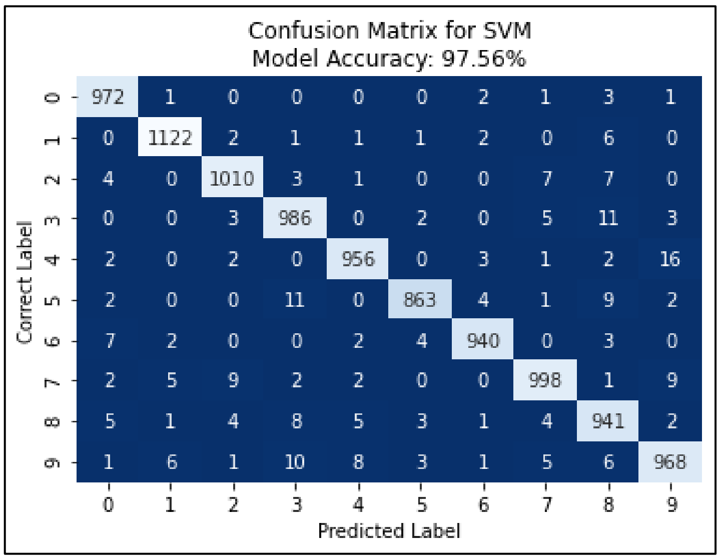
1. **Support Vector Machine**

   SVM is an extremely popular algorithm widely used for classification problems. It is a binary classifier which can be utilized in multiclass classification problems using the one-vs-all technique. The underlying idea behind SVM is to find an optimal hyperplane that best separates the features into different domains.

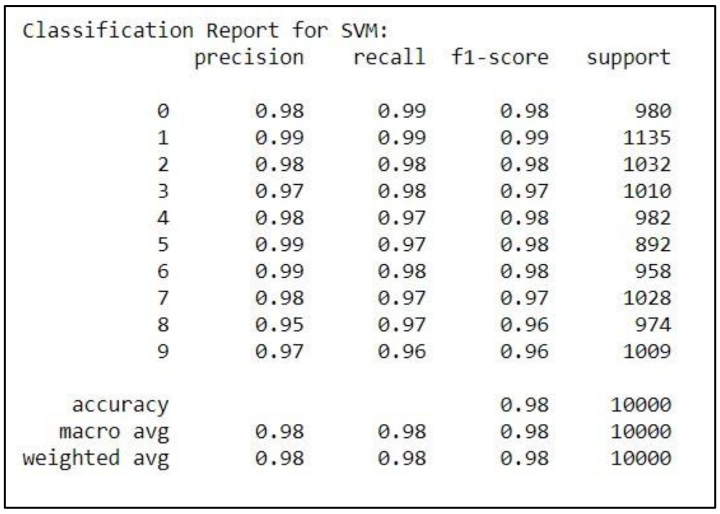
   Tuned hyperparameters:

   - Kernel: radial basis function (RBF)
   - C = 1 (regularization parameter)
   - gamma = 'scale' (is passed then it uses 1 / (n_features * variance of data)))

   Results of SVM model:



| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 972 | 1 | 0 | 0 | 0 | 0 | 2 | 1 | 3 | 1 |
| 1 | 0 | 1122 | 2 | 1 | 1 | 1 | 2 | 0 | 6 | 0 |
| 2 | 4 | 0 | 1010 | 3 | 1 | 0 | 0 | 7 | 7 | 0 |
| 3 | 0 | 0 | 3 | 986 | 0 | 2 | 0 | 5 | 11 | 3 |
| 4 | 2 | 0 | 2 | 0 | 956 | 0 | 3 | 1 | 2 | 16 |
| 5 | 2 | 0 | 0 | 11 | 0 | 863 | 4 | 1 | 9 | 2 |
| 6 | 7 | 2 | 0 | 0 | 2 | 4 | 940 | 0 | 3 | 0 |
| 7 | 2 | 5 | 9 | 2 | 2 | 0 | 0 | 998 | 1 | 9 |
| 8 | 5 | 1 | 4 | 8 | 5 | 3 | 1 | 4 | 941 | 2 |
| 9 | 1 | 6 | 1 | 10 | 8 | 3 | 1 | 5 | 6 | 968 |



```
Classification Report for SVM:
              precision    recall  f1-score   support

           0       0.98      0.99      0.98       980
           1       0.99      0.99      0.99      1135
           2       0.98      0.98      0.98      1032
           3       0.97      0.98      0.97      1010
           4       0.98      0.97      0.98       982
           5       0.99      0.97      0.98       892
           6       0.99      0.98      0.98       958
           7       0.98      0.97      0.97      1028
           8       0.95      0.97      0.96       974
           9       0.97      0.96      0.96      1009

    accuracy                           0.98     10000
   macro avg       0.98      0.98      0.98     10000
weighted avg       0.98      0.98      0.98     10000
```

## 2. <u>Logistic Regression</u>

Logistic regression is a supervised classification algorithm which predicts the probability of the target variable. This algorithm is also a binary classification algorithm. It classifies the target variable by measuring the relationship between the categorical dependent variable and one or more independent variables by estimating probabilities using a logistic function, which is the cumulative distribution function of logistic distribution.

<u>Hyperparameters</u>:

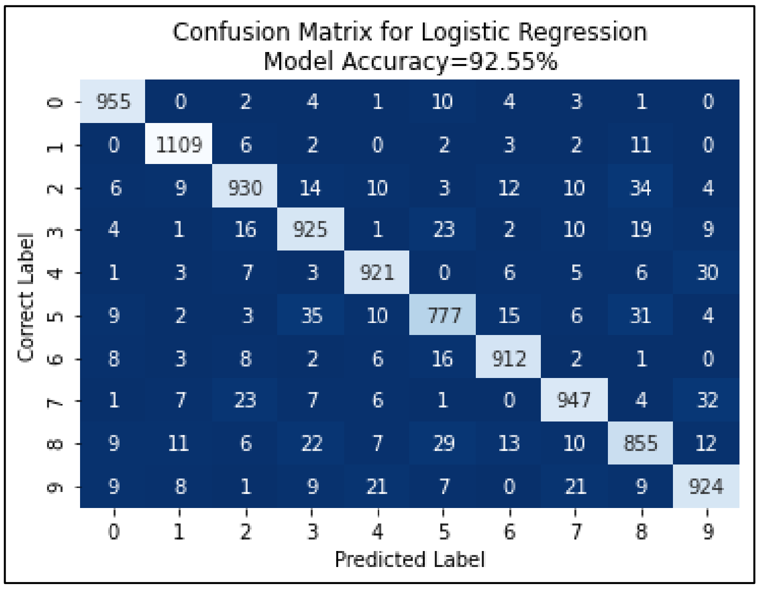
C = 1 (regularization parameter. Smaller value implies stronger regularization)

solver = ‘lbfgs’

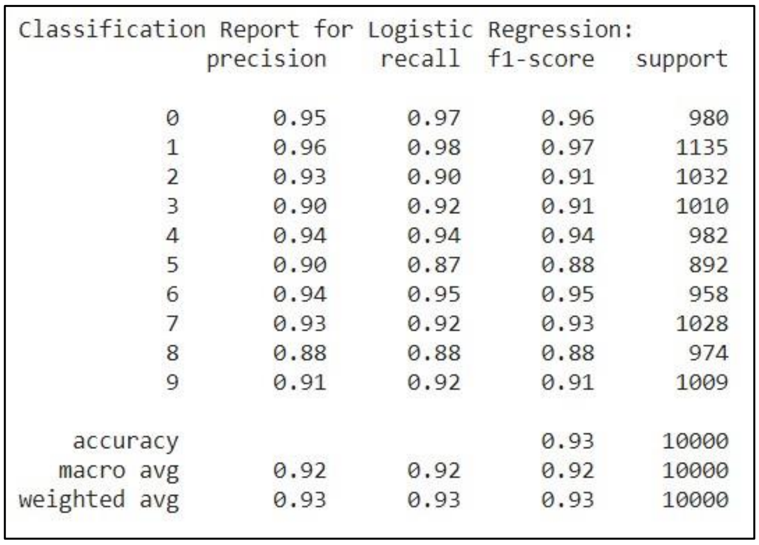
penalty = ‘l2’ (norm to be used for penalization)

Logistic regression model does not have significant hyperparameters that can be tuned.

<u>Results of Logistic regression model:</u>



| Correct Label \ Predicted Label | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 955 | 0 | 2 | 4 | 1 | 10 | 4 | 3 | 1 | 0 |
| 1 | 0 | 1109 | 6 | 2 | 0 | 2 | 3 | 2 | 11 | 0 |
| 2 | 6 | 9 | 930 | 14 | 10 | 3 | 12 | 10 | 34 | 4 |
| 3 | 4 | 1 | 16 | 925 | 1 | 23 | 2 | 10 | 19 | 9 |
| 4 | 1 | 3 | 7 | 3 | 921 | 0 | 6 | 5 | 6 | 30 |
| 5 | 9 | 2 | 3 | 35 | 10 | 777 | 15 | 6 | 31 | 4 |
| 6 | 8 | 3 | 8 | 2 | 6 | 16 | 912 | 2 | 1 | 0 |
| 7 | 1 | 7 | 23 | 7 | 6 | 1 | 0 | 947 | 4 | 32 |
| 8 | 9 | 11 | 6 | 22 | 7 | 29 | 13 | 10 | 855 | 12 |
| 9 | 9 | 8 | 1 | 9 | 21 | 7 | 0 | 21 | 9 | 924 |



```
Classification Report for Logistic Regression:
              precision    recall  f1-score   support

           0       0.95      0.97      0.96       980
           1       0.96      0.98      0.97      1135
           2       0.93      0.90      0.91      1032
           3       0.90      0.92      0.91      1010
           4       0.94      0.94      0.94       982
           5       0.90      0.87      0.88       892
           6       0.94      0.95      0.95       958
           7       0.93      0.92      0.93      1028
           8       0.88      0.88      0.88       974
           9       0.91      0.92      0.91      1009

    accuracy                           0.93     10000
   macro avg       0.92      0.92      0.92     10000
weighted avg       0.93      0.93      0.93     10000
```

## 3. Decision Tree

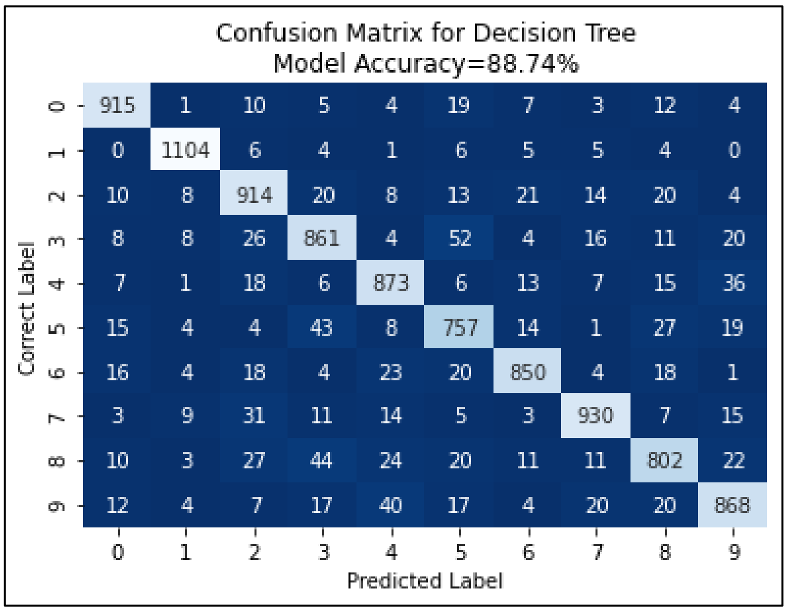
A decision tree classification is a machine learning algorithm which uses known labels from the past data to evaluate or forecast the categories of the future datasets where the class labels are not known.

Tuned hyperparameters:

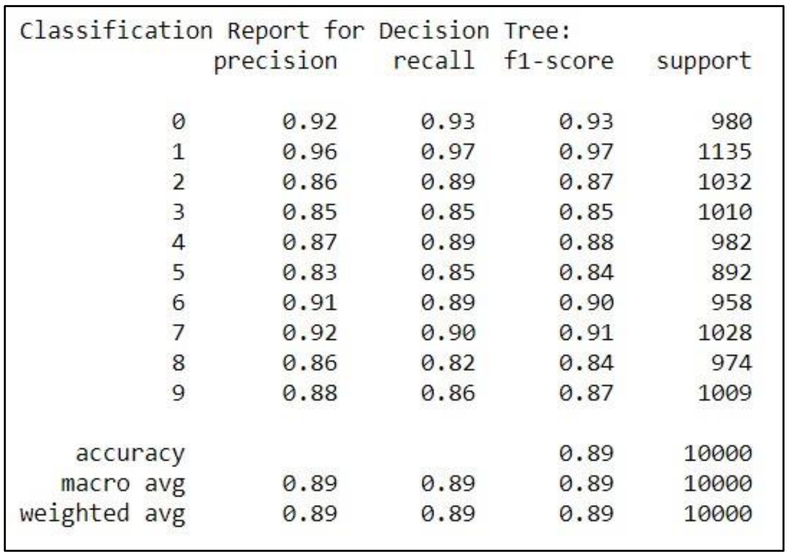
- criterion = entropy
- max_depth = 15
- min_samples_leaf = 3
- min_weight_fraction_leaf = 0.0001

Decision trees are highly sensitive to the data, features used to perform the split and prone to noise. An individual tree cannot predict well, and unless tuned otherwise, decision trees are known to overfit the data, leading to a less general model. Careful tuning is necessary to obtain a high accuracy on unseen data.

Results of Decision tree model:



| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 915 | 1 | 10 | 5 | 4 | 19 | 7 | 3 | 12 | 4 |
| 1 | 0 | 1104 | 6 | 4 | 1 | 6 | 5 | 5 | 4 | 0 |
| 2 | 10 | 8 | 914 | 20 | 8 | 13 | 21 | 14 | 20 | 4 |
| 3 | 8 | 8 | 26 | 861 | 4 | 52 | 4 | 16 | 11 | 20 |
| 4 | 7 | 1 | 18 | 6 | 873 | 6 | 13 | 7 | 15 | 36 |
| 5 | 15 | 4 | 4 | 43 | 8 | 757 | 14 | 1 | 27 | 19 |
| 6 | 16 | 4 | 18 | 4 | 23 | 20 | 850 | 4 | 18 | 1 |
| 7 | 3 | 9 | 31 | 11 | 14 | 5 | 3 | 930 | 7 | 15 |
| 8 | 10 | 3 | 27 | 44 | 24 | 20 | 11 | 11 | 802 | 22 |
| 9 | 12 | 4 | 7 | 17 | 40 | 17 | 4 | 20 | 20 | 868 |



```
Classification Report for Decision Tree:
              precision    recall  f1-score   support

           0       0.92      0.93      0.93       980
           1       0.96      0.97      0.97      1135
           2       0.86      0.89      0.87      1032
           3       0.85      0.85      0.85      1010
           4       0.87      0.89      0.88       982
           5       0.83      0.85      0.84       892
           6       0.91      0.89      0.90       958
           7       0.92      0.90      0.91      1028
           8       0.86      0.82      0.84       974
           9       0.88      0.86      0.87      1009

    accuracy                           0.89     10000
   macro avg       0.89      0.89      0.89     10000
weighted avg       0.89      0.89      0.89     10000
```

## 4. Random Forest

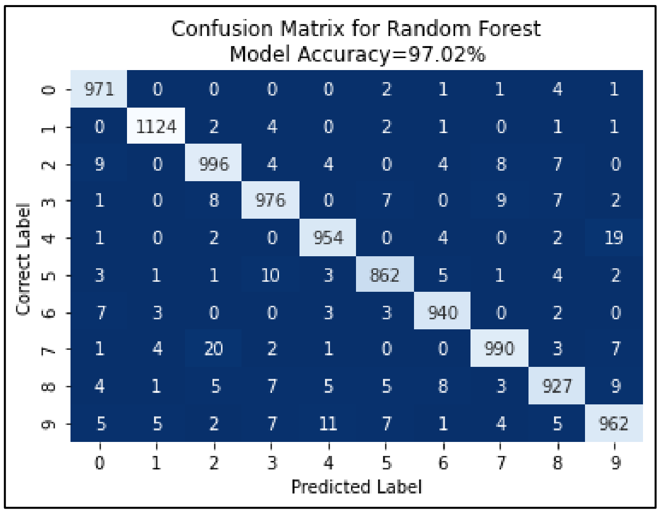
Random decision forests or random forests are an aggregate machine learning tool for classification, regression and other tasks that function by creating a number of decision tress at training time, producing the class that is the class mode (classification) or the individual trees' mean/average predictor (regression). This method uses decision trees' drawback of overfitting to the training set, and relies on the component trees being uncorrelated. Hence, random forest algorithm generally outperforms decision tree algorithm generated model.

Tuned hyperparameters:

- criterion = entropy (using information gain criteria)
- n_estimators = 101
- min_samples_split = 2
- min_samples_leaf = 1
- max_features = 'sqrt'

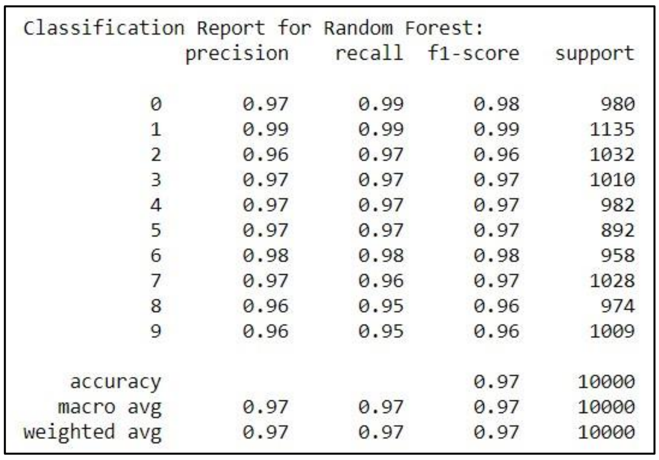
The best model we obtained was able to reach a testing accuracy of 97.02%. It uses the information gain criteria for deciding the best split at each level. It uses 91 decision tree estimators, creating the deepest possible trees, as the splitting occurs so long as nodes in the tree have at least 2 samples, and a node becomes a leaf only when it has 1 sample in it. To decide on the best split at each level, this model uses the square root of number of features (= 28) (decided randomly).

Results of Random forest model:



Confusion Matrix for Random Forest
Model Accuracy=97.02%

| Correct Label \ Predicted Label | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 971 | 0 | 0 | 0 | 0 | 2 | 1 | 1 | 4 | 1 |
| 1 | 0 | 1124 | 2 | 4 | 0 | 2 | 1 | 0 | 1 | 1 |
| 2 | 9 | 0 | 996 | 4 | 4 | 0 | 4 | 8 | 7 | 0 |
| 3 | 1 | 0 | 8 | 976 | 0 | 7 | 0 | 9 | 7 | 2 |
| 4 | 1 | 0 | 2 | 0 | 954 | 0 | 4 | 0 | 2 | 19 |
| 5 | 3 | 1 | 1 | 10 | 3 | 862 | 5 | 1 | 4 | 2 |
| 6 | 7 | 3 | 0 | 0 | 3 | 3 | 940 | 0 | 2 | 0 |
| 7 | 1 | 4 | 20 | 2 | 1 | 0 | 0 | 990 | 3 | 7 |
| 8 | 4 | 1 | 5 | 7 | 5 | 5 | 8 | 3 | 927 | 9 |
| 9 | 5 | 5 | 2 | 7 | 11 | 7 | 1 | 4 | 5 | 962 |



```
Classification Report for Random Forest:
              precision    recall  f1-score   support

           0       0.97      0.99      0.98       980
           1       0.99      0.99      0.99      1135
           2       0.96      0.97      0.96      1032
           3       0.97      0.97      0.97      1010
           4       0.97      0.97      0.97       982
           5       0.97      0.97      0.97       892
           6       0.98      0.98      0.98       958
           7       0.97      0.96      0.97      1028
           8       0.96      0.95      0.96       974
           9       0.96      0.95      0.96      1009

    accuracy                           0.97     10000
   macro avg       0.97      0.97      0.97     10000
weighted avg       0.97      0.97      0.97     10000
```

**Out of the 4 machine learning models, Support vector machine is the algorithm which was able to achieve the highest accuracy on the test data.**

Summary statistics of ML models:

|  |  |
|---|---|
| • Decision Tree – | 88.74% |
| • Logistic Regression – | 92.56% |
| • Support Vector Machine – | 97.56% |
| • Random Forest – | 97.02% |

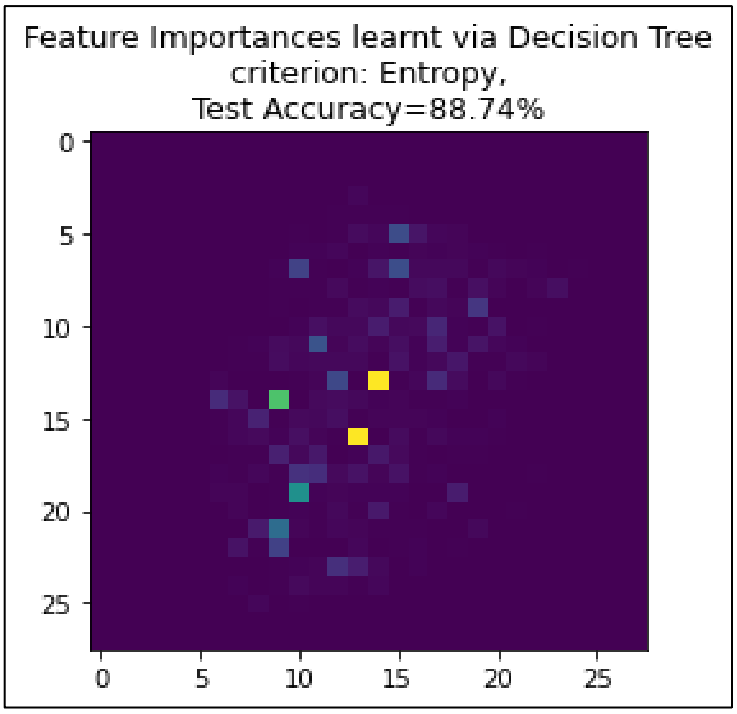
The decision tree model gives the least accuracy (88.74%) for the given dataset. Decision Trees attempt to divide the data space into smaller and smaller regions, based on information gain (entropy) or impurity (gini) and develop, "rules" at each node, that classify the data. Decision trees are highly sensitive to the training data, and very prone to noise and overfitting. Hence decision trees perform poorly. Even though a single decision tree may not perform well, the high volatility among a group of trees is put to use in the Random Forest ensemble algorithm.

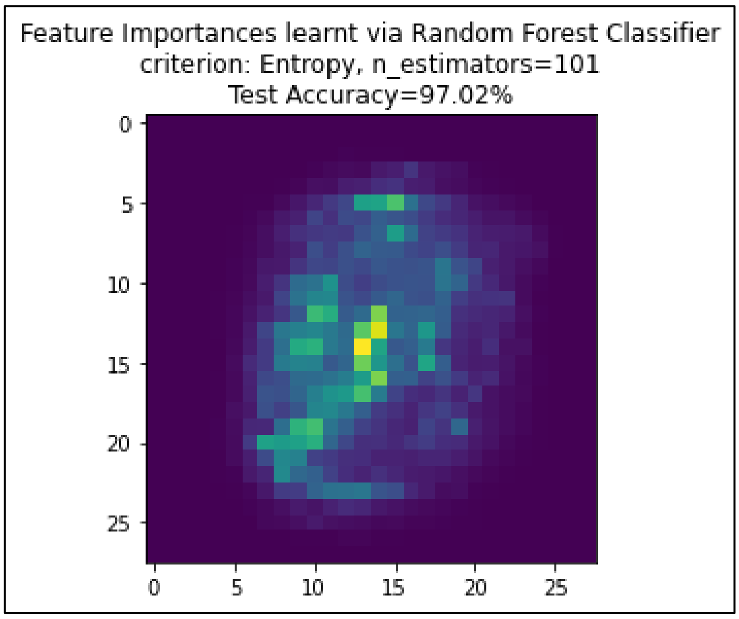
The logistic regression model has the second least accuracy rate (92.56%). Logistic regression usually performs well in a lot of classification tasks. But it has its limitations. The assumption of linearity between the dependent variable and independent variables causes the inability of the algorithm to classify on non-linear data, which is mostly the case with real-world data. Logistic regression is unable to learn complex relationships. Even the simplest neural networks can outperform logistic regression.

Logistic Regression boundaries can be too close to the outermost point of each class, whereas SVMs aim to go as far away from a class's points as it can to create their boundaries. This also leads to less risk of overfitting with the case of SVM. Another strength of SVM is that it uses geometrical properties of data points, while logistic regression is purely statistical. By changing the underlying kernel that transforms the data, we allow it to learn non-linear boundaries. The testing accuracy of the SVM model is 97.56%.

Even though a single decision tree may not perform well, the high volatility among a group of trees is put to use in the Random Forest ensemble algorithm. In the Random Forest algorithm, trees that have learnt different rules to classify the data offer their individual class predictions, and the random forest algorithm outputs the class predicted by majority of the trees. The collective wisdom of the forest proves to perform better at classification problems and gives a high classification accuracy most of the time.

Another interesting way to visualize the results of Random Forest and Decision Tree algorithms is through an attribute called "feature_importances_" in the implementation provided in scikit-learn package in Python. Both algorithms build trees, splitting based on the maximum information gain or least impurity in the child nodes. This way, some features are marked as more important than the others. The following images show the most important (brightest) features that the two algorithms – Decision Tree and Random Forest - were able to find in order to make the rules to classify data.





The advantage of many individual trees constructed randomly in the Random Forest is evident here. The Random Forest algorithm as a whole takes more features into account for deciding the class an incoming example belongs to, and these features are from the forest of 101 trees together. An individual decision tree cripples some features when they aren't used to decide the split. A majority of the features, especially in a problem like MNIST, where there are 784 features, get masked in the case of decision tree and cannot participate in the decision making. Random Forest relies on trees being uncorrelated in order to strengthen its predictions, and such different trees automatically give different important features.

## Deep learning models

As the search space for tuning hyperparameters is too big, we take the following approach. We build different deep models by adjusting and tuning one hyperparameter at a time and then choosing the best model from each stage to see the effect of the next hyperparameter(s). If we choose a different number of epochs, or a different ordering of parameters variation than we have chosen below, or even a different default model, it is possible that we would get a different result.

Our goal is to build a small model that learns the most in the specified duration, measured here by epochs. So, we keep the number of epochs constant at 10 and compare models by varying the other hyperparameters.

### Model 1:

We first build a basic neural network with the following hyperparameters values:

Number of hidden layers = 1
Number of units in hidden layers = 128
Activation function = relu for hidden layers, softmax for output layer
Bias = no bias
Drop out = 0
Loss function = sparse_categorical_crossentropy
Optimizer = adam
Epochs = 10

Training accuracy of model: 99.74%

Testing accuracy of model: **97.74%**

An important point to note here is that we have done zero feature extraction to help the model understand the data, and the data has been fed into the neural network as it is, without any transformation. Despite that, the testing accuracy of the model is as high as 97.74%, already comparable to the best traditional Machine Learning model results we obtained.

The training accuracy of the model is usually greater than the testing accuracy of the model. This is because the neural network learns on the same data for multiple epochs, attempting to increase its accuracy on the training data. At times this leads to overfitting as well, which is the phenomenon where the model performs extremely well on the training data, but not as good on the testing data, implying that the model is not a good ‘general’ model.

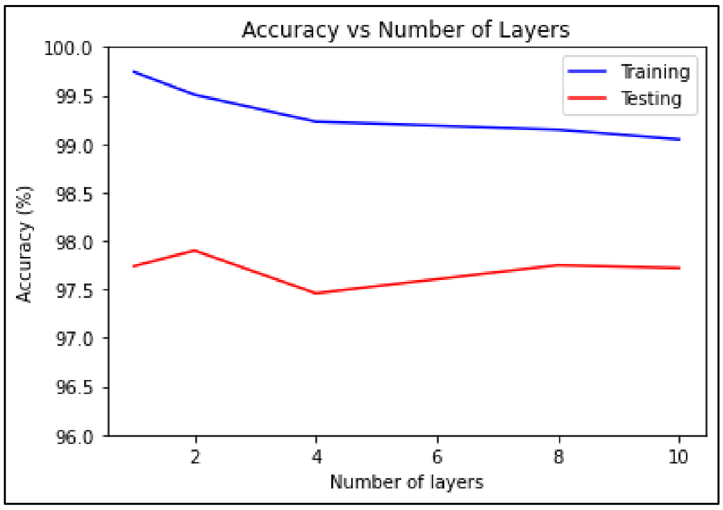
We vary the number of layers in the following few neural network models to start with.

## Varying number of hidden layers

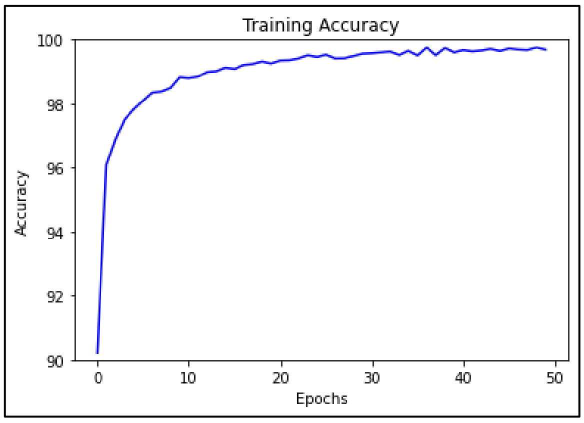
We now build 4 models with varying number of hidden layers (2,4,8,10) and keeping the rest of the parameters same as the above model.

Model 2,3,4,5 hyperparameters:

**Number of hidden layers = (2,4,8,10) respectively**
Number of units in hidden layers = 128
Activation function = relu for hidden layers, softmax for output layer
Bias = no bias
Dropout = 0
Loss function = sparse_categorical_crossentropy
Optimizer = adam
Epochs = 10



According to the above graph, as we increase the number of layers, the accuracy of the model seems to decrease, as compared the smaller models. This appears to be counter-intuitive. If we plot the training accuracy for 10 hidden layers for different number of epochs (shown in the following figure), we can see that the accuracy keeps increasing, reaching 99.75%. We must understand that as we increase the number of layers, a greater number of epochs are required to train the weights of the model. As we have fixed the epoch value to 10, we see that the accuracy of the larger models trained on 10 epochs decreases with increasing number of layers.



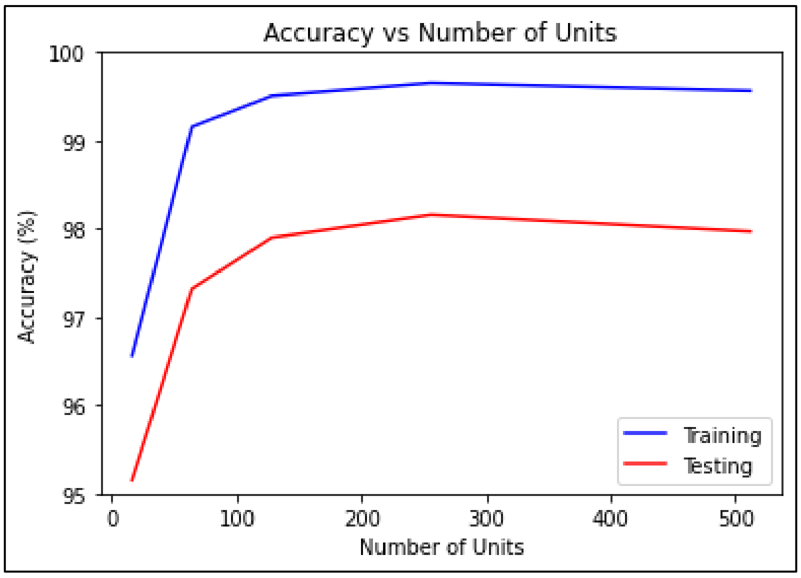
The highest testing accuracy we get is **97.899%** which is the testing accuracy for model with **2 hidden layers**. We therefore, fix the number of hidden layers to 2 and try building models varying the next hyperparameter: Number of units in each of the layers.

## Varying number of units in hidden layers

We now build 4 more models with varying number of units in each hidden layer (16, 64, 256, 512), keeping the rest of the hyperparameter values fixed as before. (the training and testing accuracies for 128 units has been plotted according to previous iteration).

Model 6,7,8,9:

Number of hidden layers = 2 (tuned)
**Number of units in hidden layers = (16,64,256,512) respectively**
Activation function = relu for hidden layers, softmax for output layer
Epoch = 10
Bias = no bias
Dropout = 0
Loss function = sparse_categorical_crossentropy
Optimizer = adam



We get the highest testing accuracy for the model with 256 units in the hidden layers. Hence, we tune the number of units in hidden layer parameter to be 256.

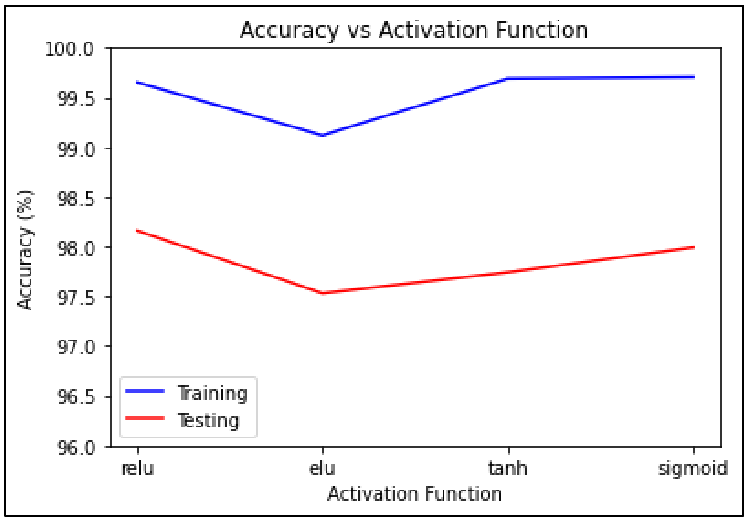
The training accuracy for model with **256 units** in hidden layer is 99.65% while the testing accuracy for the model is **98.16%.**

## Varying the activation function used in the hidden layers

We now compare 3 models with varying activation function in hidden layers with other hyperparameter values same as above.

Model 10,11,12:

Number of hidden layers = 2 (tuned)
Number of units in hidden layers = 256 (tuned)
**Activation function = relu, elu, tanh, sigmod for hidden layers respectively,** softmax for output layer
Bias = no bias
Drop out = 0
Loss function = sparse_categorical_crossentropy
Optimizer = adam
Epochs = 10



The above graph shows that the model which uses sigmoid function as the activation function has the highest training accuracy, while the highest accuracy for testing data is achieved in the model which uses relu as activation function.

Tanh activation and sigmoid activation can be seen to have comparable training accuracy since they have similar properties. But both of them suffer from vanishing gradients issue due to saturation.

The training accuracy of model using **relu activation function** is 99.65% and the testing accuracy is **98.16%.**

We next vary the bias unit applied on the two layers, trying out the 4 possible combinations.

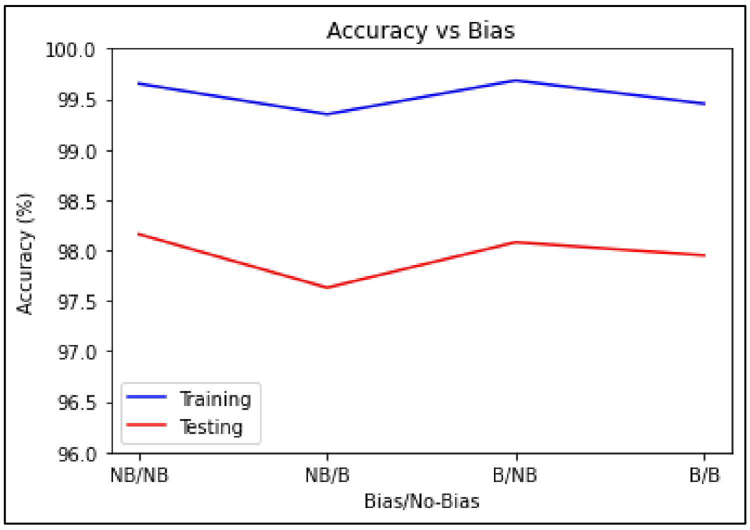
## Varying bias

We now build and compare 4 models having bias or no bias for every hidden layer, keeping the rest of the hyperparameters fixed as above.

NB – no bias, B - bias

Model 13,14,15:

Number of hidden layers = 2 (tuned)
Number of units in hidden layers = 256 (tuned)
Activation function = relu for hidden layers (tuned), softmax for output layer
**Bias = (NB, NB) ;(NB, B) ;(B, NB) ;(B, B) respectively**
Drop out = 0
Loss function = sparse_categorical_crossentropy
Optimizer = adam
Epochs = 10



The highest training and testing accuracy displayed by the model with **no bias unit in both the hidden layers**. The training data accuracy is 99.65%, while the testing data accuracy is **98.16%.**

## Varying dropout

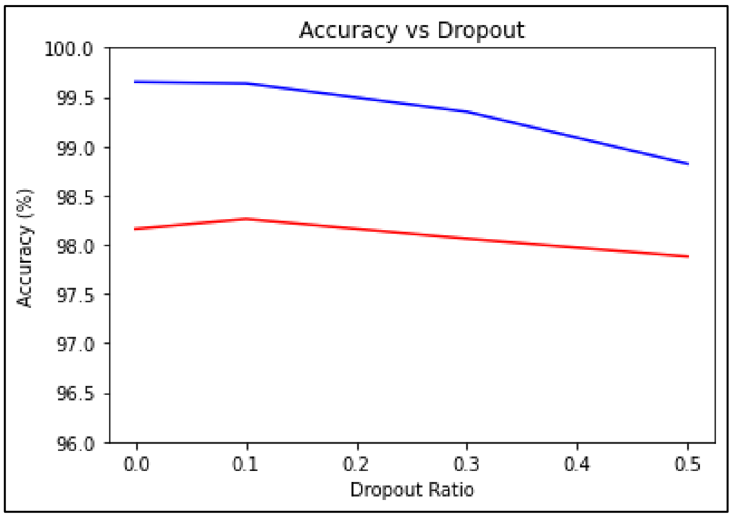
Deep learning models generally work on very large datasets. As the dataset size increases, the redundancy of the training examples may also increase, depending on the dataset values. The dropout hyperparameter accounts for the redundancy in data by building a model by randomly dropping a specified proportion of the training dataset.

We now compare the accuracy of the models with varying dropout values, ceteris paribus.

Model 16,17,18:

(model with dropout value 0 is not considered as a new model as it has already been accounted)

Number of hidden layers = 2 (tuned)
Number of units in hidden layers = 256 (tuned)
Activation function = relu for hidden layers (tuned), softmax for output layer
Bias = (NB, NB) (tuned)
**Drop out = (0, 0.1, 0.3, 0.5) respectively**
Loss function = sparse_categorical_crossentropy
Optimizer = adam
Epochs = 10



The above graph shows that the model with **0.1 dropout** has highest testing accuracy. This shows that the training data has slight redundancy in it. The training data accuracy is 99.63%, while the testing data accuracy is **98.26%.**

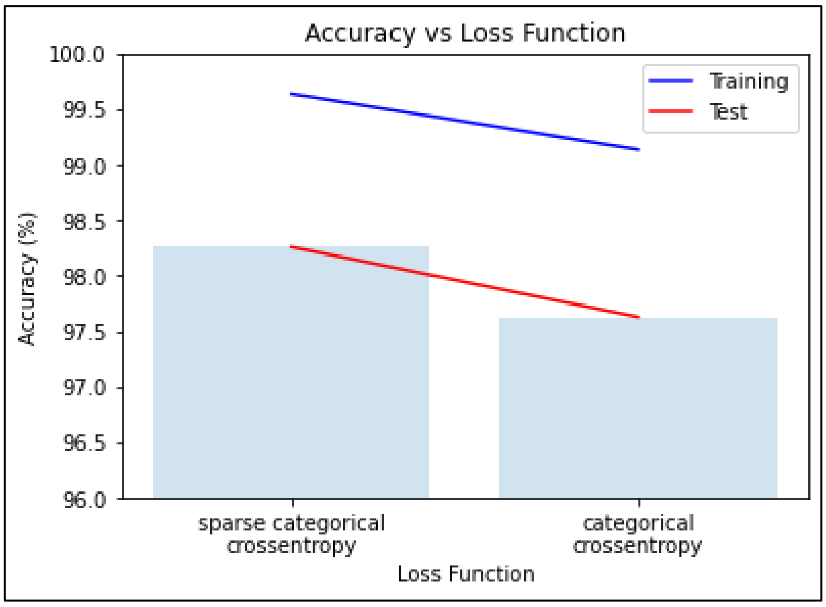
We fix the dropout value as 0.1 and next we tune the loss function.

## Varying loss function

The performance of a neural network also depends on the loss function that is used to optimize the model weights. We build and compare 2 models using different loss functions to compute the weights, ceteris paribus.

Model 20:

Number of hidden layers = 2 (tuned)
Number of units in hidden layers = 256 (tuned)
Activation function = relu for hidden layers, softmax for output layer
Bias = (NB, NB) (tuned)
Drop out = 0.1 (tuned)
**Loss function = ‘sparse_categorical_crossentropy’ and ‘categorical_crossentropy’ resp.**
Optimizer = adam
Epochs = 10



The above graph shows that the accuracy of the model using sparse categorical crossentropy function as loss function has higher accuracy than the model built using the categorical cross-entropy function. Categorical cross entropy learns and predicts from one-hot encoded labels and is used when we have a multiclass classification problem at hand. Sparse categorical cross entropy is better to use when our labels are integers. The difference is not too significant, so we can say that the representation of the y label does not matter much.

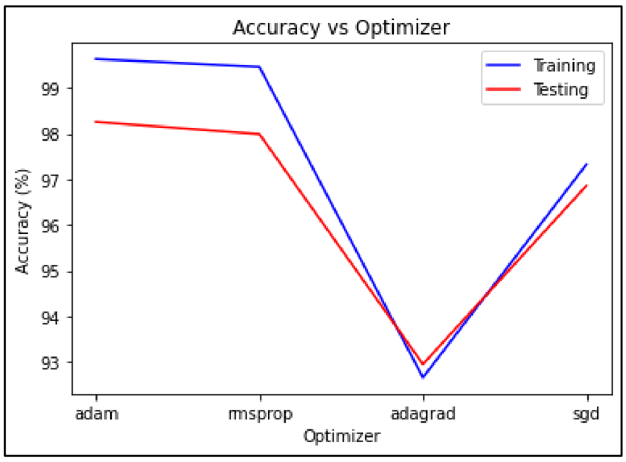
Moving on to the optimizer hyperparameter, we maintain the loss function as sparse categorical cross entropy for further models.

## Varying optimizers

Finally, we build 4 NNs using varying optimizer functions, ceteris paribus.

Model 21,22,23:

Number of hidden layers = 2 (tuned)
Number of units in hidden layers = 256 (tuned)
Activation function = relu for hidden layers, softmax for output layer (tuned)
Bias = (NB, NB) (tuned)
Drop out = 0.1 (tuned)
Loss function = sparse_categorical_crossentropy (tuned)
**Optimizer = ('adam', 'rmsprop', 'adagrad', 'sgd') respectively**
Epochs = 10



The model built using 'adam' optimizer gives the best accuracy results for training as well as testing data. The accuracy of the training data is 99.63%, while the accuracy of the testing data for that model is 98.26%.

## Summary

The test accuracy of the machine learning algorithms are as follows:

- Support Vector Machine – 97.56%
- Logistic Regression – 92.56%
- Decision Tree – 88.73%
- Random Forest – 96.96%

The lowest and the highest testing accuracies obtained from the deep learning models are 97.74% and 98.26% respectively.

The above accuracy statistics of models built on the same handwritten digit database show that the deep learning models outperform the models built using machine learning by a large margin. The DL model chosen before tuning the hyperparameters offered a testing accuracy higher than the accuracy of the best machine learning model. This shows the power of deep learning over traditional machine learning algorithms.

Machine learning algorithms do not generate as many decision regions as artificial neural because of the local constancy assumption. This can be attributed as one of the major factors as to why ML algorithms failed to outperform the deep learning models. Also, even when there is lack of domain knowledge for feature introspection, DL techniques outshine others. Deep learning models perform very well when the data size is large.

The best testing accuracy we could obtain was 98.26%.