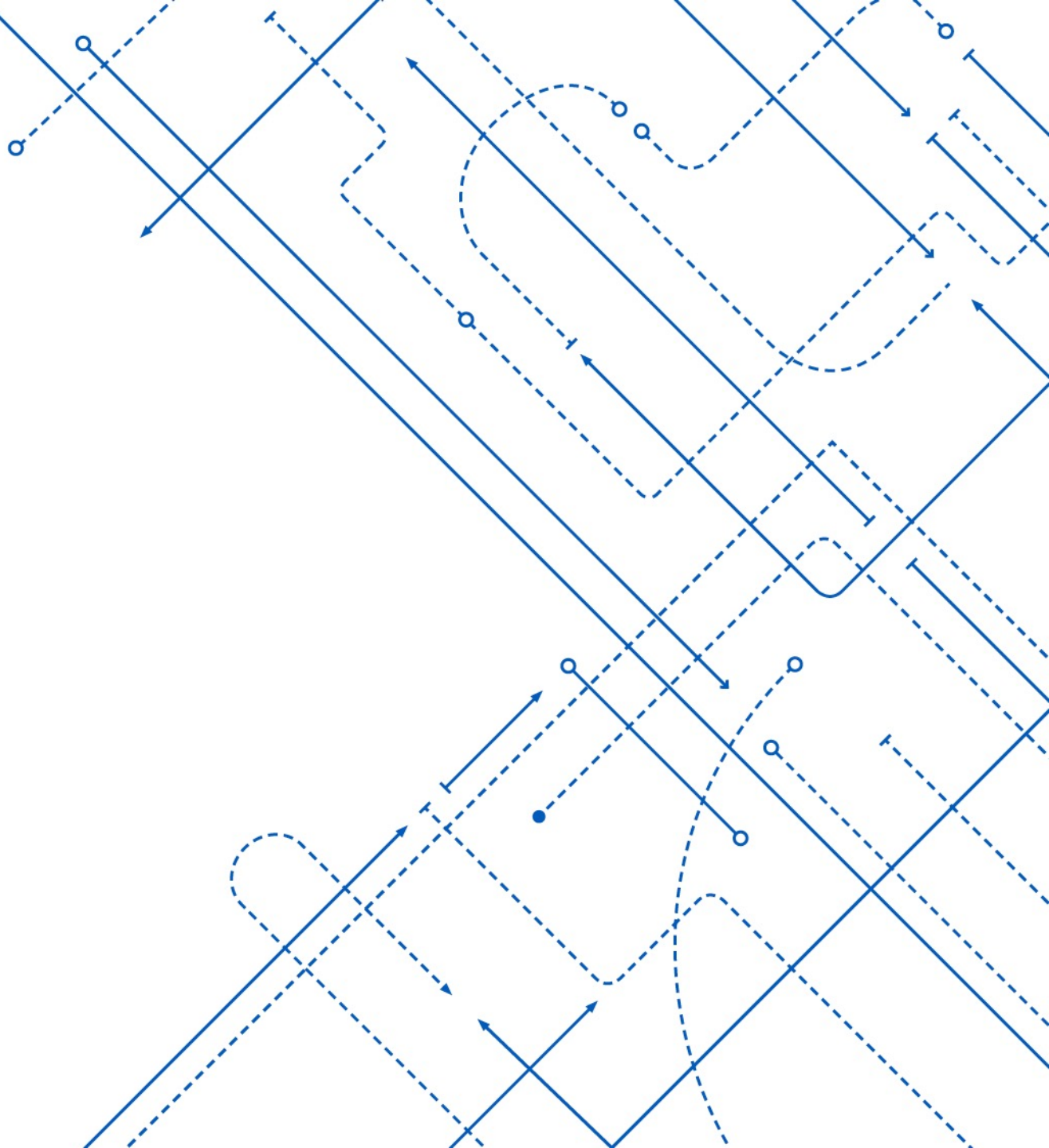
FUNDAMENTALS OF AI

ASSIGNMENT 2

NAMAN TEJASWI

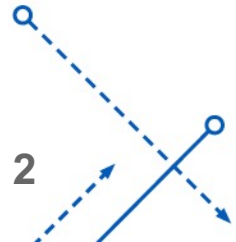
Team 4



University at Buffalo The State University of New York

Titanic Dataset

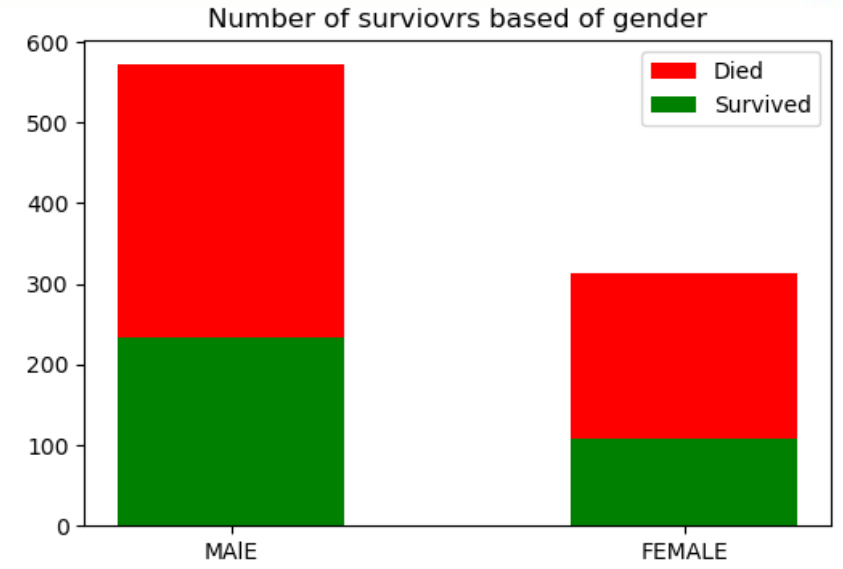
- Essential Data Features used to model our hypothesis : Survived, Pclass, Sex, Age, Fare Siblings/Spouses Aboard, Parents/Children Aboard
- Removed attributes : Name
- No missing data no need to interpolate, remove etc.
- Why not combine Fare Siblings/Spouses Aboard and Parents/Children Aboard ?
- Target Variable : Survived 0 or 1 => Binary Classification
- Preprocessing :Encoding sex into 1 and 0 from male and female
- Normalizing Fare and Age
- Normalization vs Standardization, when to use which?
- Splitting the dataset into 80:20 for training and test

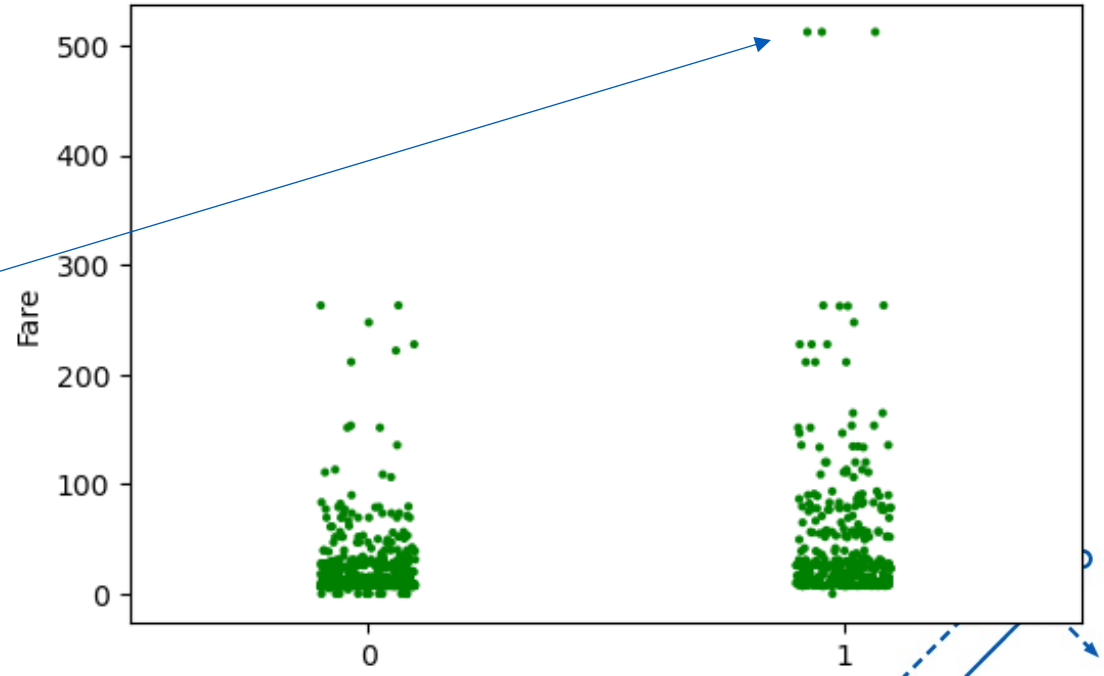


Exploratory Data Analysis

- The percentage of men that did not survive was significantly higher than the percentage of women that did not survive
- This is likely due to the fact that women and children were given a priority with life boats and rescue.
- Percentage of female that survived 74.20%
- Percentage of males that survived 19.02%
- We also observe that a small set of people who paid

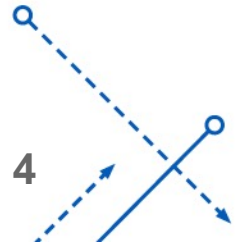
significantly higher fare were able to survive.





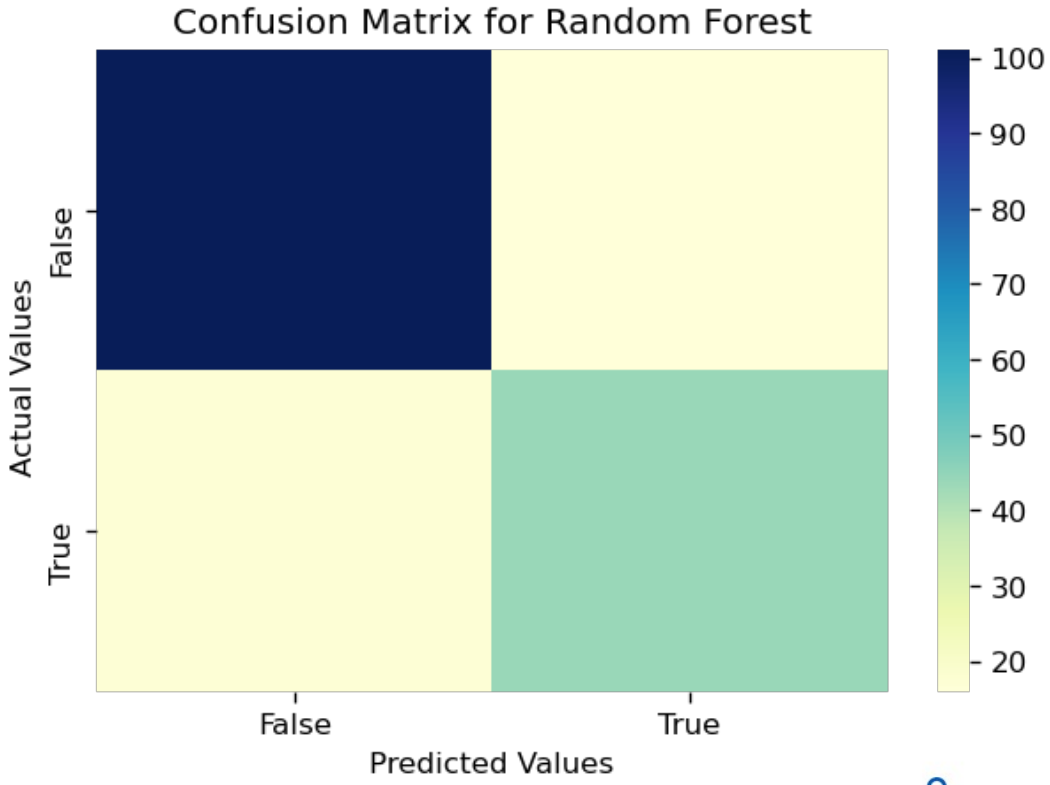
Machine Learning Algorithms used

- K Nearest Neighbors
- Random Forest
- Logistic Regression
- Random forest is an ensembling technique. What then are ensembling techniques?
- Ensembling involves combining multiple models, in our case we create decision tress then combining them.
- Random forest can be thought as combining multiple decision tress, that is we have multiple questions so that we can define our boundary, or classification based on the answers to those question. Random forest is a bagging technique which uses multiple decision trees as bags
- Other Ensembling Techniques- Bagging, Bootstrapping, Stacking
- Often we may need to combine multiple ml models on a practical task to achieve desired results.



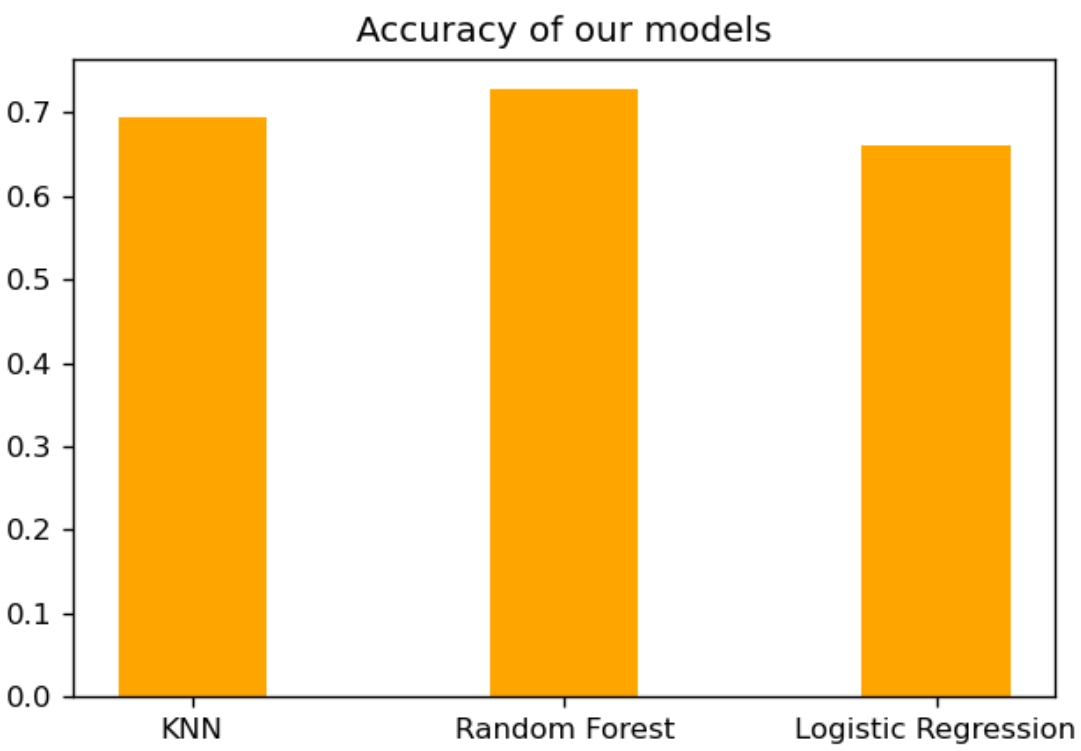
Evaluation Metrics

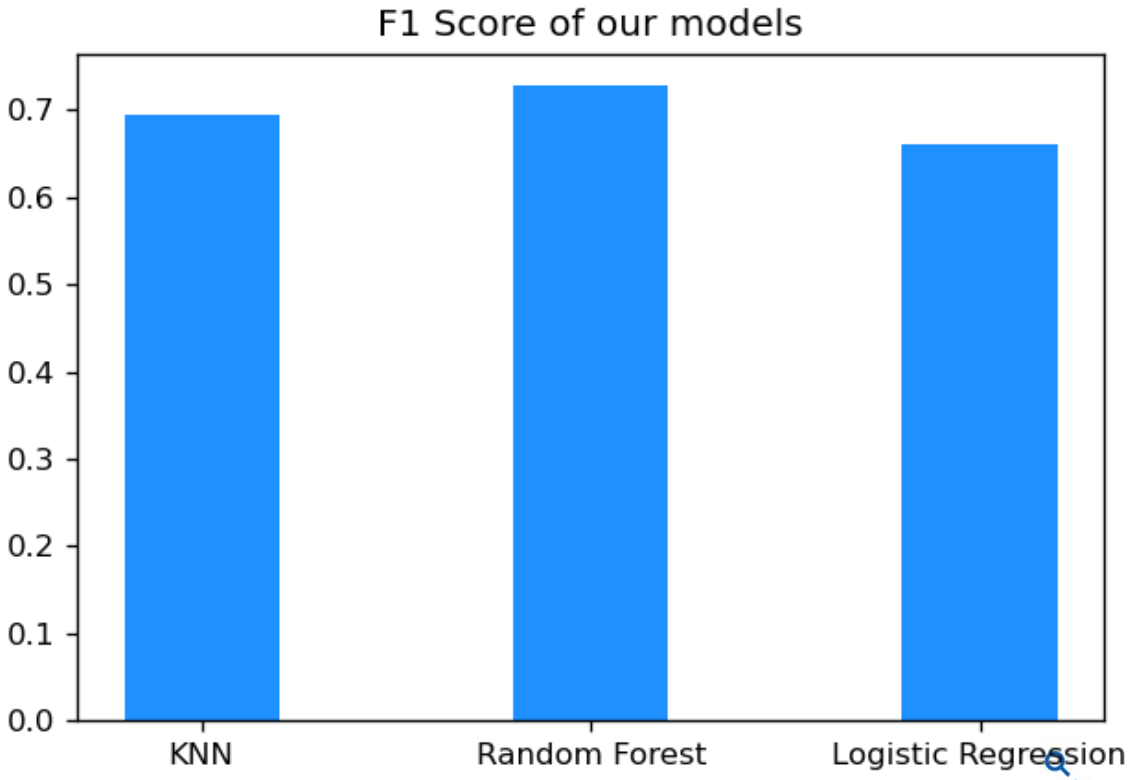
- Precision is how many selected samples are correct.
- Recall is how many of the correct samples are selected.
- Why is it critical to use them?
- Given that we know that only about 19% men survived, say we are given only this subset then a naïve model which says that any of the men did not survive will also have an accuracy of 81% !
- Trade off between Precision and Recall, for example justice system traditionally has high precision albeit low recall. Another example could be the covid testing where high recall may be needed despite low precision assuming that the covid vaccine does not have any detriments



F1 Score=2*((precision*recall)/(precision+recall)).

Comparing the results

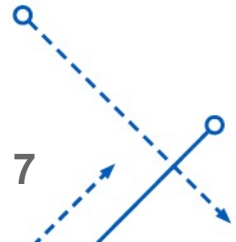




Our Neural Network Architecture

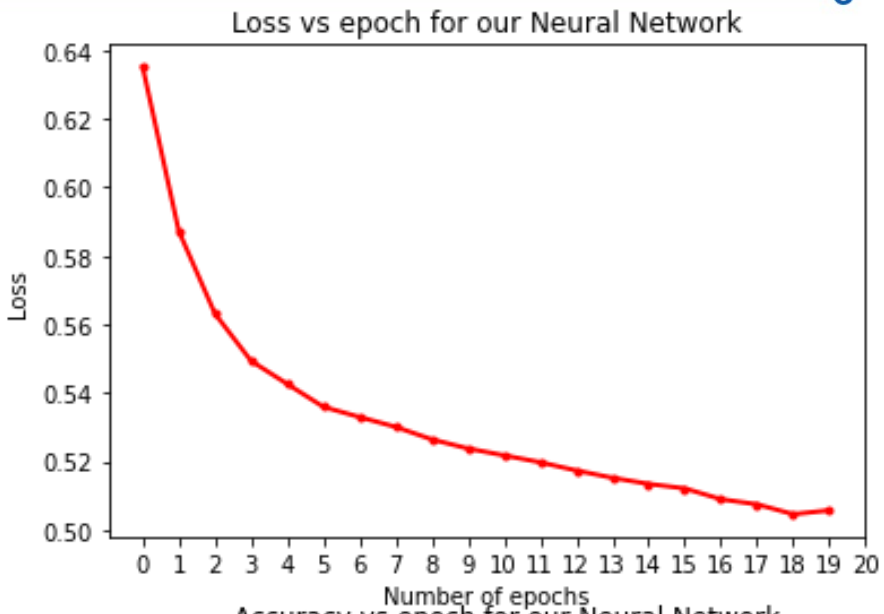
- We start with a deep NN with 3 layers having Neurons 32, 16, and 8 and using activation relu
- Finally we have a sigmoid activation layer which yields us probability i.e values in [0,1]
- We finally convert these probabilities into a binary outcome using 0.5 as our threshold.
- This threshold is not ubiquitous and often depends upon problem context and whether We need higher precision or higher recall.

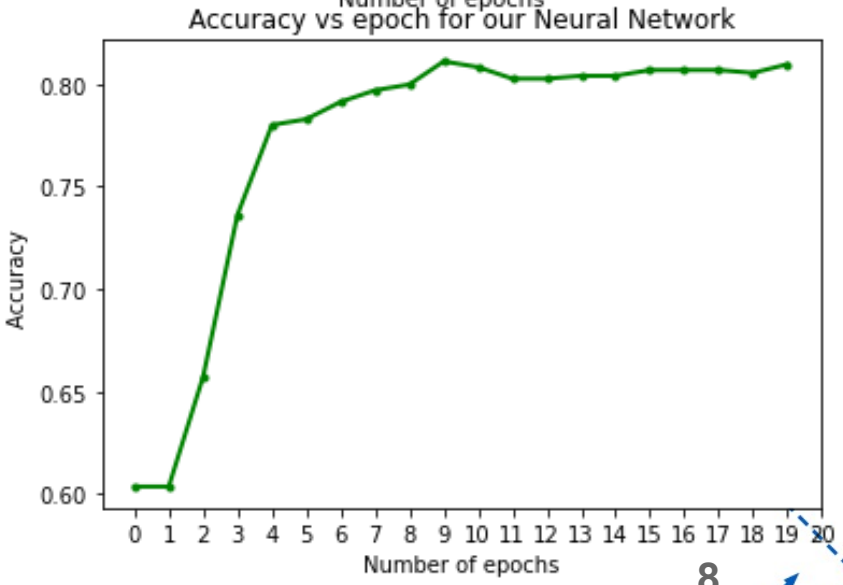
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 32)                224

 dense_1 (Dense)             (None, 16)                528

 dense_2 (Dense)             (None, 8)                 136

 dense_3 (Dense)             (None, 4)                 36

 dense_4 (Dense)             (None, 1)                 5

=================================================================
Total params: 929
Trainable params: 929
Non-trainable params: 0
_________________________________________________________________
```



Training our Neural Network

- We first trained our neural neural network for 20 epochs without any mini batch meaning our parameters are updated based of one example at a time.
- We used binary cross entropy as our loss function
- We then also used a mini batch of 16 test data and as we would expect our learning rate improved.
- We have logged accuracy and loss as the metrics for each epoch and thus have a plot.
- We also have the plot of loss v/s epoch of a NN with mini batch of 16 test cases to illustrate the difference in speed with which our model learns.





Changing Hyperparameters and Model Architectures

- Firstly we try a different architecture with less number of neuro

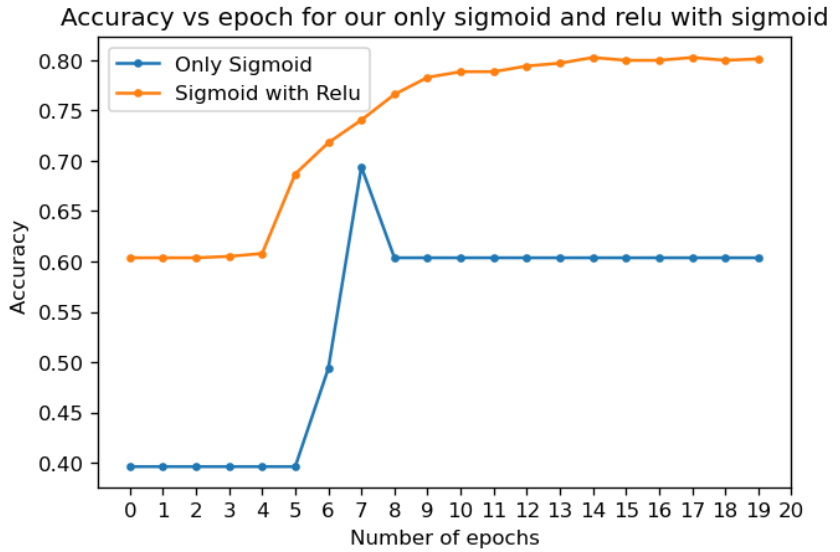
in our layer

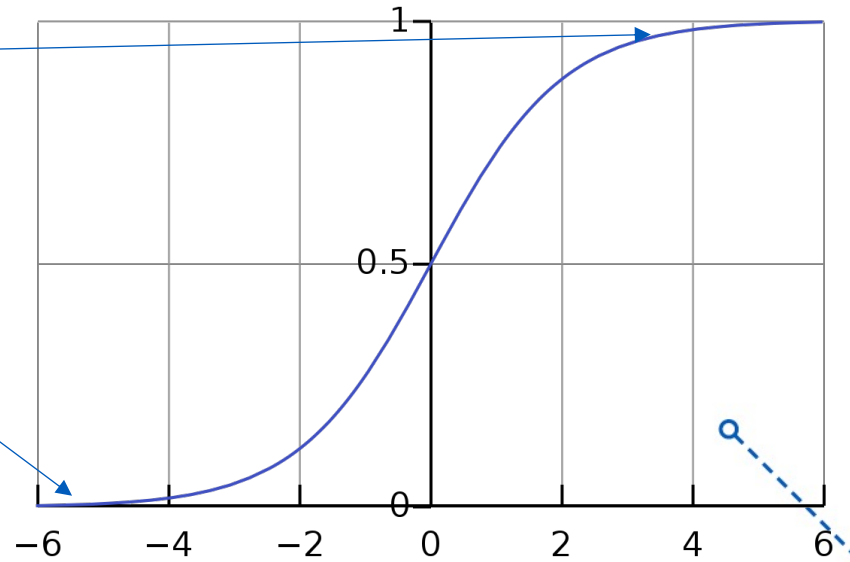
- We reduce the number of parameters by a factor of 10
- However we do not see any significant change in our model's performance!
- We then extend our experiments and try to use an overly simplistic model with only sigmoid activation functions, this time our model suffers heavily.
- Thus we concur that me must have a good balance between model sophistication as an overly complex one is a waste of compute power and time and an overly simple one may not be accurate enough

```
Model: "sequential_37"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_137 (Dense)           (None, 8)                 56

 dense_138 (Dense)           (None, 4)                 36

 dense_139 (Dense)           (None, 1)                 5

=================================================================
Total params: 97
Trainable params: 97
Non-trainable params: 0
_________________________________________________________________
```

Vanishing Gradient Problem with sigmoid activation

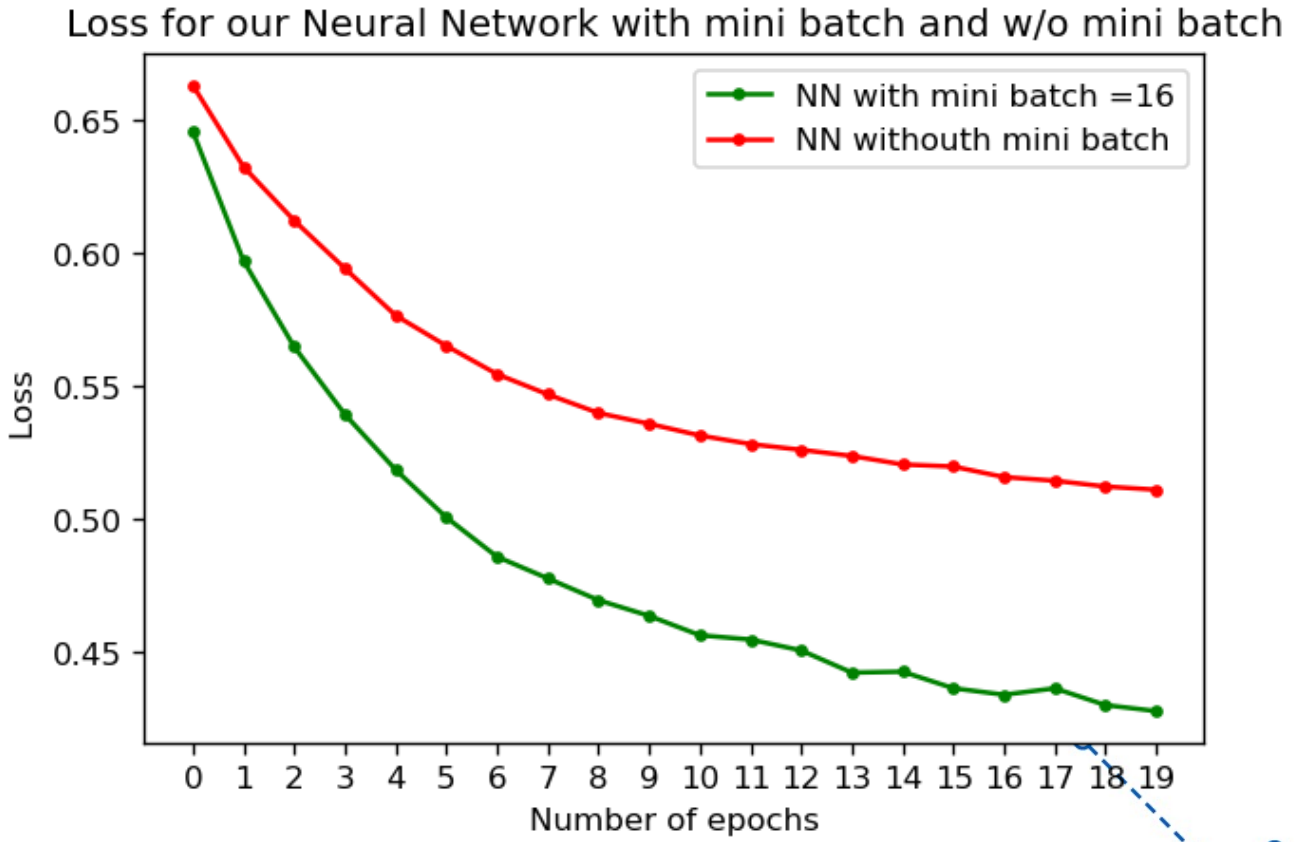
- We discover that only using sigmoid activation function with deep layer having multiple neuron was horrific for our model.
- This is attributed to the well known problem of vanishing gradients while working with sigmoid activation.
- The gradient of sigmoid activation does not change significantly as the sigmoid is constant for extremely small or extremely large values of input. This is also prominent in the derivative graph
- The backpropagation and weight updation is not going to work if the underlying changes in gradient are insignificant





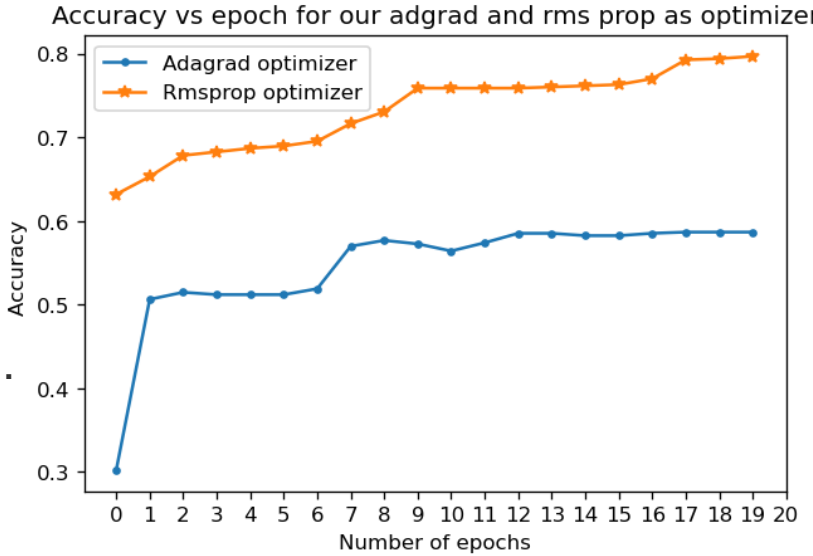
Why does mini batch increase our learning rate over stochastic?

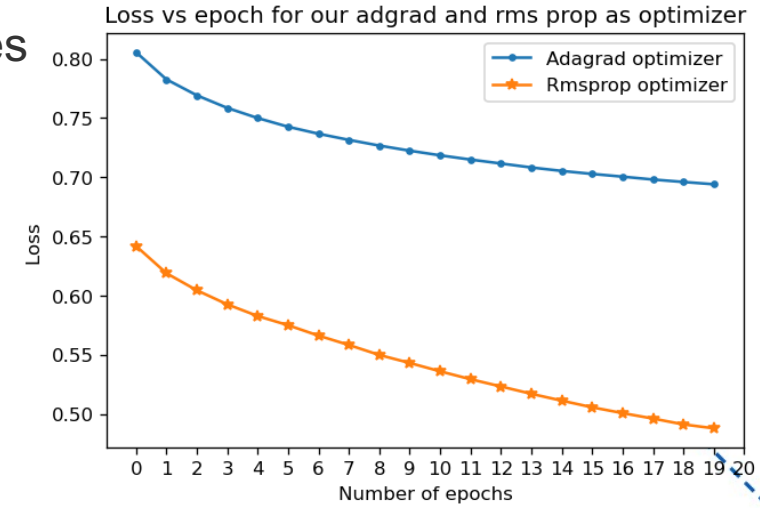
- When we use mini batch our parameters are updated such that the cumulative loss over all the batch in our case 16 is reduced.
- In contrast when we are training with a single sample at a time our hypothesis function is likely to oscillate around as it tries to reduce loss on one instance.
- When we use mini batch our model is able to generalize faster and it also reduced the variance of our model
- Also before trying to increase our learning rate in order to reduce our training time must first try training with mini batches.



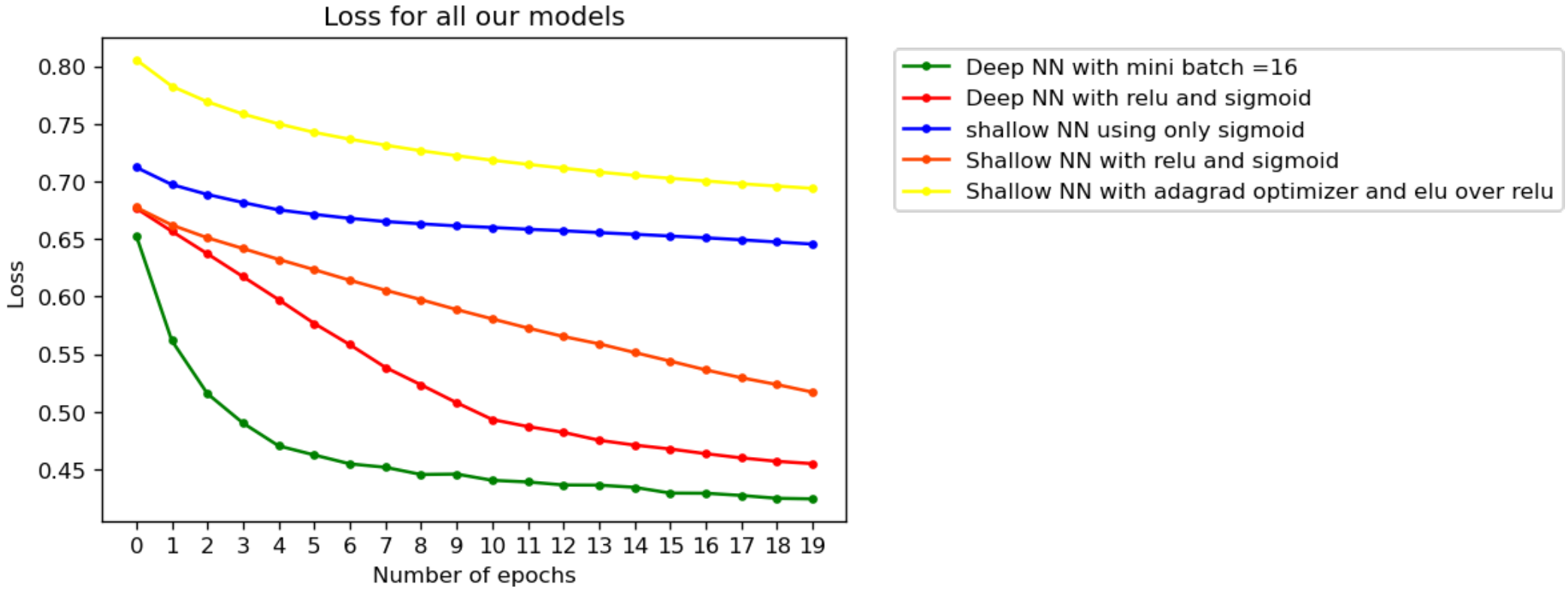
Using Adagrad vs Rmsprop as optimizer

- We observe that the our model learns faster and has a higher accuracy when we we use rmsprop as our optimizer over adagrad.
- What could be the reason?
- Adagrad is adaptive gradient descent and it is different from sgd optimizer in the way that it tracks sum of squared gradients and tries to update gradient in all different direction, thus taking longer than sgd. The gradient in all directions will grow as we look for minima
- However as adagrad calculates gradient in all direction it does a better job avoiding non minima inflection or saddle points.
- Rmsprop Root mean square propagation is essentially an optimization of adagrad optimizer itself, it uses a decay rate for the squared term in adagrad, thereby making the convergence faster.

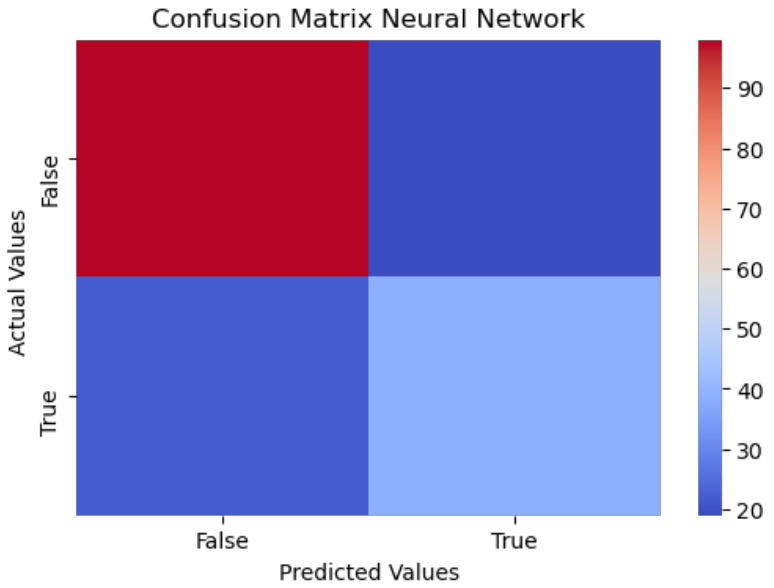




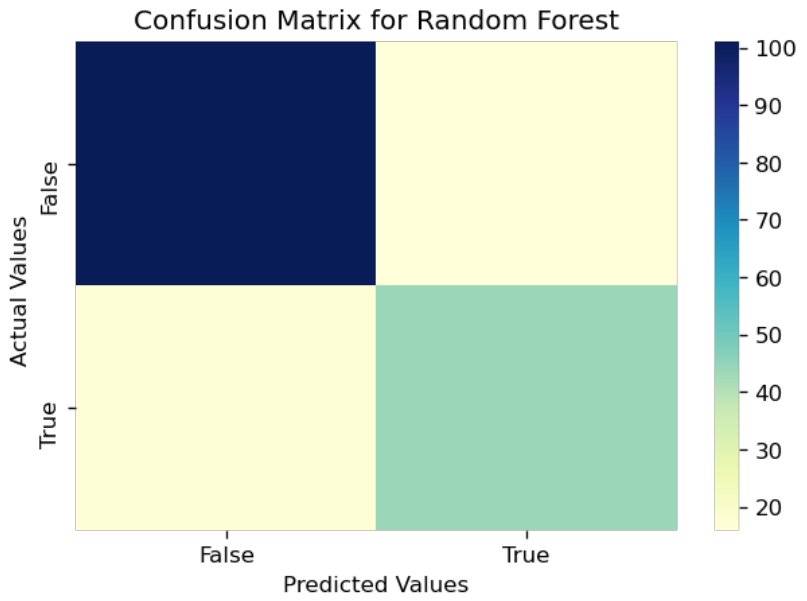
Comparison of various Neural Networks



Comparing our NN model with random forest, our best ml model



Accuracy is 78.08 %
Precision is 69.64 %
Recall is 63.93%
F1 Score is 0.667



Accuracy is 83.14 %
Precision is 77.19 %
Recall is 72.13%
F1 Score is 0.74

Thank You for your time

Have a great day!