Telecom Churn Dataset

Course: Math Concepts Deep Learning

Group Members:

Soham Mane,

Saamarth Rastogi

Dhrumil Patel,

Vishwas Chandran,

Kavish Shah,

Nandini Manyam

Date: 06 June 2022

TABLE OF CONTENTS

List of Tasks

Task	Who
Identifying Problem Statement, Targets Labels and High valued customers	Soham Mane
Missing Value Treatment, Outlier treatment and EDA	Vishwas Chandran
Feature Extraction And Data Balancing	Kavish Shah
Benchmark accuracies using ML	Nandini Manyam
Neural Network using Pytorch	Dhrumil Patel
Neural Network using Keras and Tensorflow, Conclusion	Saamarth Rastogi

1.0 Introduction

The telecommunications industry experiences an average of 15-25% annual churn rate. Given the fact that it costs 5-10 times more to acquire a new customer than to retain an existing one, customer retention has now become even more important than customer acquisition.

1.1 Problem statement

For many incumbent operators, retaining high profitable customers is the number one business goal. To reduce customer churn, telecom companies need to predict which customers are at high risk of churn. In this project, we have analysed customer-level data of a leading telecom firm and built predictive models to identify customers at high risk of churn.

1.2 DataSet Overview

The dataset which we have chosen consists of approximately 9999 customers with 226 features. It has 4 months of data which are June, July, August, and September. Here each feature is a combination of multiple main features combined together.

1.3 Preprocessing

Target Creation

Since Data labels for our dataset were unknown which means our dataset was unsupervised which is the actual case in many real-life scenarios we have used historical data to create the label. Here we have used data from September months to create a target/ label column. After this, we have eliminated the data of this month and have proceeded with our further steps of creating the model.

Missing Value Treatment

The data we are using is extensive, having 226 columns and over 10,000 rows. By analysing data, we found out that over 108 columns have missing values. So the Columns which have missing dates, we are filling with some default dates, and with the other column which has missing values, we are replacing them with zeros.

Filtering High Valued Customers

High valued customer is the person whose recharge amount is more than the 70th percentile of the avg recharge amount of the 3 months. The Reason we are filtering these customers is, In markets like India and Southeast Asia,80% of the revenue comes from these high valued customers. So, if we predict the churn for these customers and take action, we can stop the revenue leakage. So, we calculate the average recharge amount over 3 months and filter the customers at the 70th percentile.

EDA: Using Bilateral and unilateral exploratory data analysis on the data and using a boxplot, we plotted the data, to find outliers. By Visualising, we found out that, data above 95 Percentile and below 5 percentile are outliers. So we are removing those outliers from the data.

1.4 Feature Extraction:

As Mentioned earlier our data is extremely extensive with 10k Samples and 226 features. To reduce this dimensionality of our data we had to do feature extraction to get the most of our data using the minimum number of features.

For this, we used principal component analysis. PCA is basically used for dimensionality reduction of huge datasets by transforming the larger features into smaller features but still containing the most important data for modeling and prediction.

```
pca = PCA(random_state=42)
pca.fit(X_scaled_df)
```

After Implementing PCA to our data set we found that almost 95% of the important features were narrowed down to only 70 features from 226 features available in total

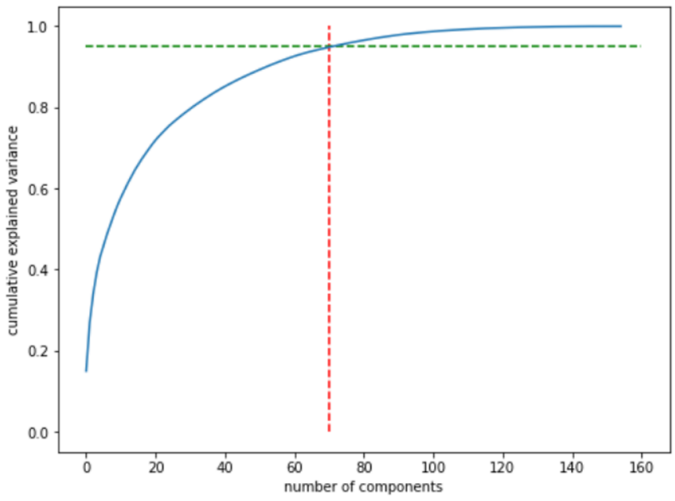

Figure 1: Plot for PCA

We checked the correlation between the features to find the features that affect the most but we found out that almost all the features had similar relation between then so a correlation cannot be used

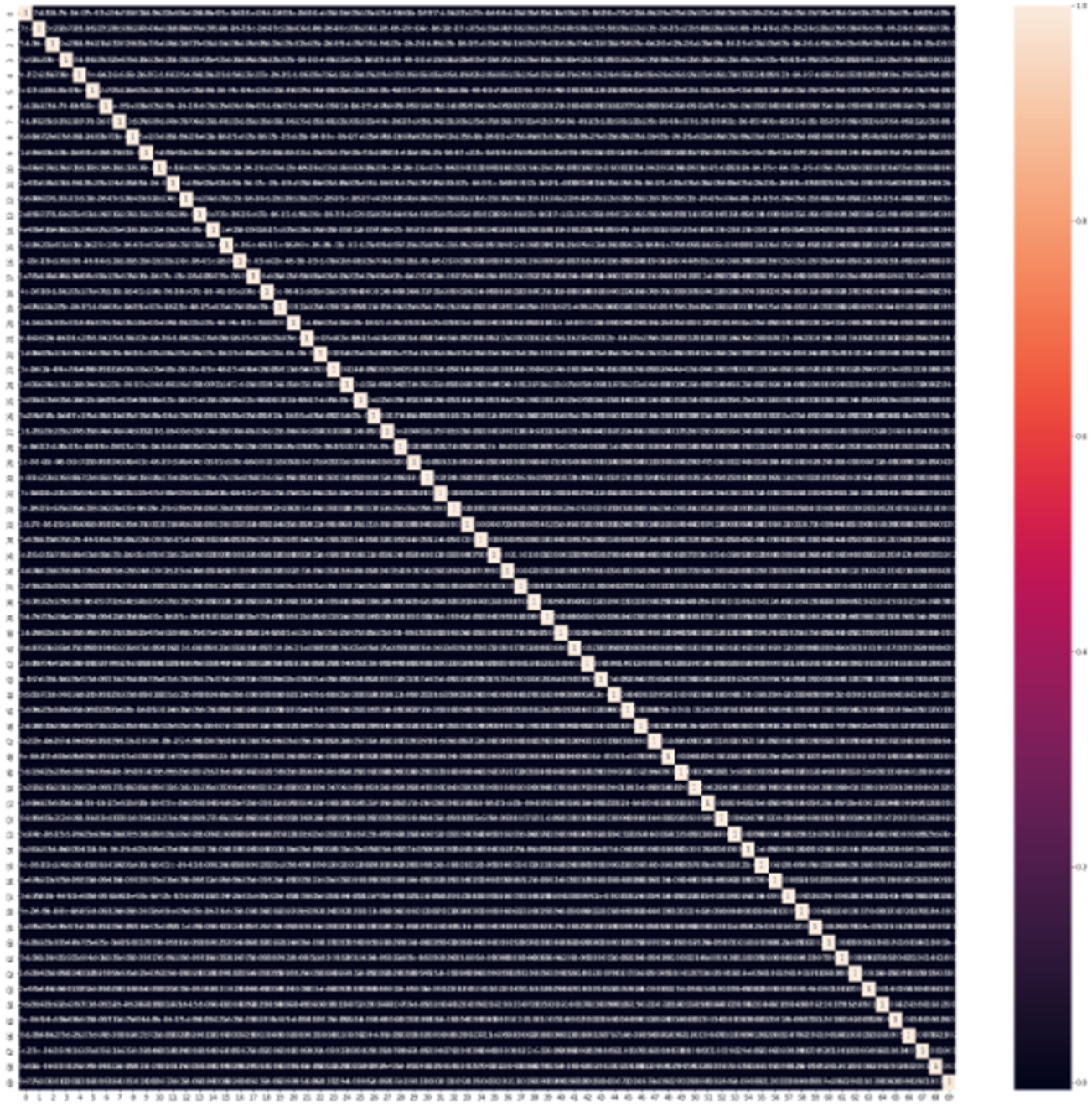

Figure 2: Correlation of 70 PCA components

We included these 70 features in our training and test data

```
pca_incr = IncrementalPCA(n_components=70)
```

```
churn_data_pca_train = pca_incr.fit_transform(X_train_sm)
churn_data_pca_train.shape
```

```
(39456, 70)
```

```
churn_data_pca_test = pca_incr.transform(X_test)
churn_data_pca_test.shape
```

```
(9003, 70)
```

1.5 Data Balancing

When analysing our data we found out that our data had a lot of skewness in it with the majority of our customers not churning from the company as shown in the figure below.

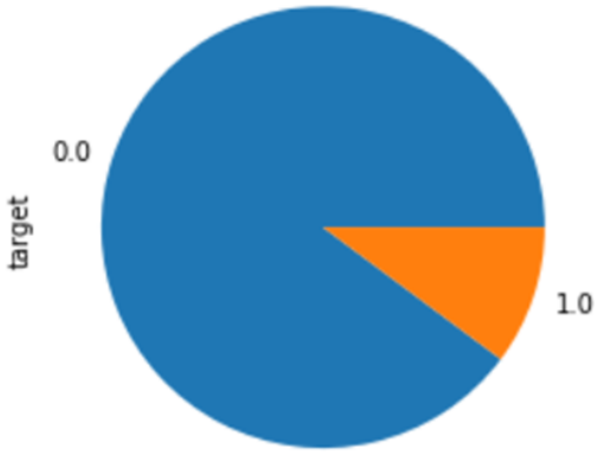

Figure 3: Imbalance Data

As seen above 89.8% of the customers were still continuing with the company while only 10.2% of the customers had churned out the company

To balance this data we used Synthetic Minority Oversampling Technique or SMOTE.what smote does is it replicates the minority data by using the k mean neighbor and adds a point between the 2 points making a line like structure after all the points have been balanced

```
sm = SMOTE(random_state=42)
X_train_sm, y_train_sm = sm.fit_resample(X_train, y_train)
```

After using SMOTE we can see that the data is balanced which can be used in a better way to predict the model

```
y_train_sm.value_counts()
```

```
0.0    19728
1.0    19728
Name: target, dtype: int64
```

2.0 METHODOLOGY

2.1 TRADITIONAL ML APPROACH

Applying the traditional ML approach to benchmark the accuracy for this problem

ML algorithms used in this approach:

- Logistic Regression
- Random Forest

2.1.1 LOGISTIC REGRESSION:

As our problem is a Binary classification problem we started benchmark accuracy using the simple logistic regression model.

For which we got the following results

- Training accuracy of around 86% and
- Testing accuracy of 82.20%

This was evaluated using the ROC-AUC score as it tells how much the model is capable to distinguish between classes

2.1.2 Random Forest:

The random forest is a classification technique that uses numerous decision trees to classify data.which produces good predictions and provides a higher level of accuracy in predicting outcomes. It can handle large datasets efficiently.

An Out-Of-Bag (OOB) score of 74% was obtained, which is used to make sure that there is no leakage of data while training the model which ensures that we have a better predictive model.

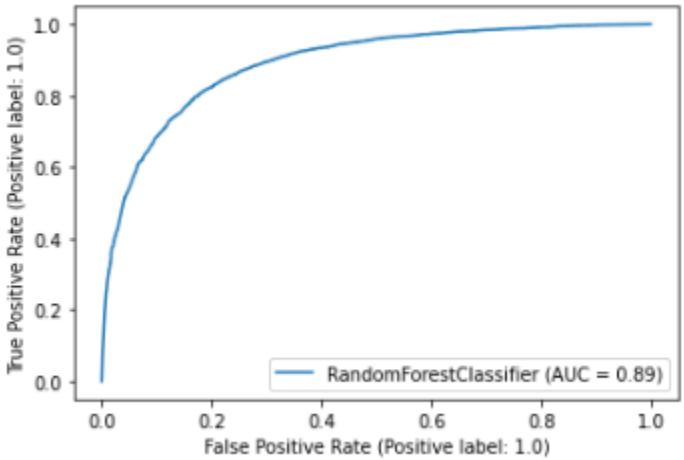

Figure 4: ROC Curve

Applied Grid-search to determine the most accurate hyperparameters for a model from the supplied set of parameters in the grid.

We got the accuracies for this model as below mentioned

- Training accuracy of 93%
- Testing accuracy was 89%

Which was better than Logistic Regression but still might get improved by using Neural Network

2.2 XGBoost without PCA:

Extreme Gradient Boosting is a distributed gradient-boosted decision tree (GBDT) machine learning framework with a scalable architecture. It includes parallel tree boosting.

It combines the predictions of a number of smaller, weaker models to attempt to precisely forecast a target variable

For which the accuracies are reported as below

- Training accuracy - 95%
- Testing accuracy - 92%

Also plotted a heat map to visualize the correlation between the top 25 features including the target column

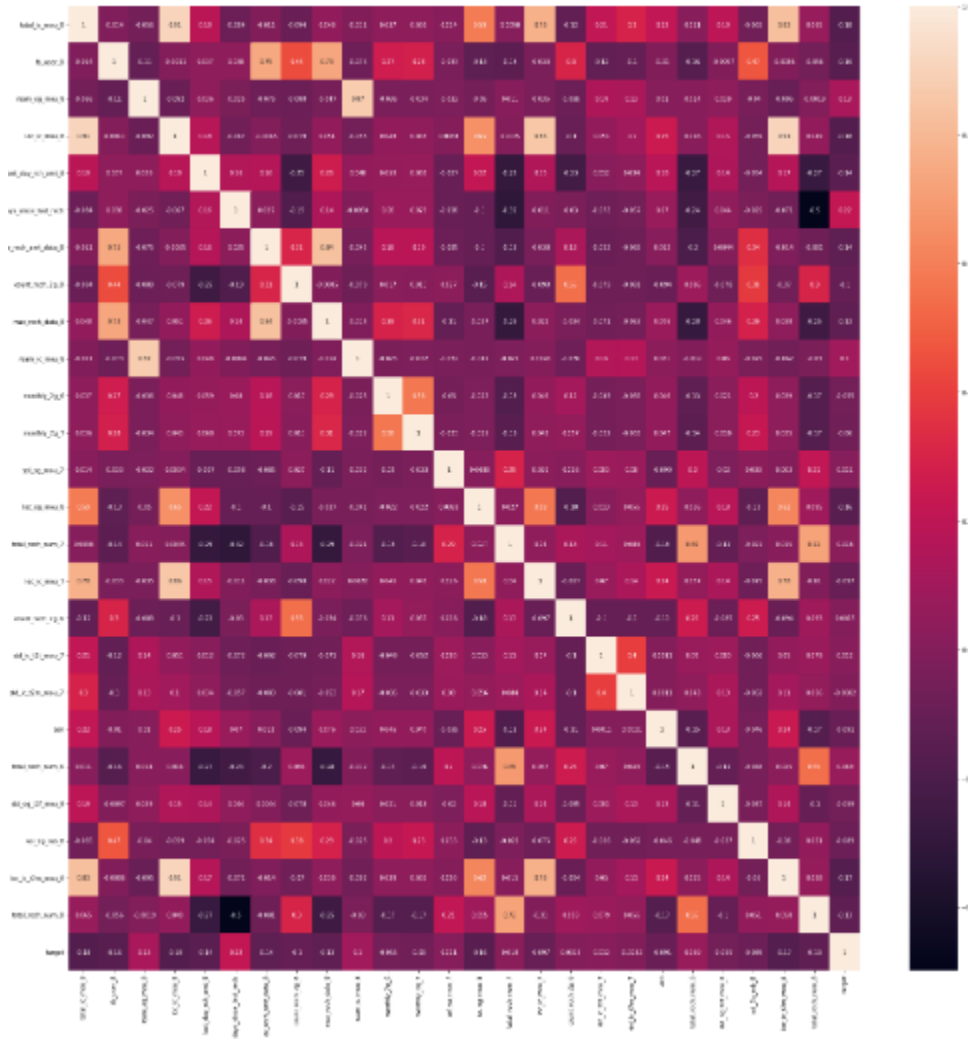

Figure 5: Heat map for top 25 features

2.3 Creating Neural Networks for Prediction

Now applying various neural networks to predict whether customers churned or not, and for comparing with baseline machine learning models.
Unlike Traditional ML algorithms, deep learning is like a black box, where we have no exact idea that how it is processing features and predicting labels for a particular sample. For implementing neural networks, we have to define the architecture and hyperparameters we want to try. Our main purpose to use neural networks for this problem is to increase accuracy, minimize the loss, and reach global minima.

Here, we are implementing neural networks using two different ways.

1)Using Pytorch

2)Using Tensorflow and Keras.

2.3.1 Neural Networks Using Pytorch

Pytorch is an inbuilt library in python language to create neural networks from scratch. Pytorch is a very flexible library to create/modify neural network architecture compared to others. It gives data scientists the freedom to specify /make neural networks according to his/her choice.

Pytorch is a more object-oriented or pythonic type of language.

Steps for implementing neural networks

- we have to define class/function for train data, validation data, test data, and predicting models.
- Then creating data loaders for train data, validation data, and test data. That helps to keep data manageable, iterate over the dataset and help to simplify pipelines for models.
- Defining architecture for the first neural network.

```
def forward(self, inputs):
    x = self.relu(self.layer_1(inputs))
    x = self.batchnorm1(x)
    x = self.relu(self.layer_2(x))
    x = self.batchnorm2(x)
    x = self.relu(self.layer_3(x))
    x = self.batchnorm3(x)
    x = self.relu(self.layer_4(x))
    x = self.batchnorm4(x)
    x = self.dropout(x)
    x = self.Sigmoid(self.layer_out(x))
```

Making a model with one input layer, three hidden layers, and one output layer. We are taking 70 input neurons,1 output neuron, and 512,512,64 neurons for three hidden layers. After some experiments, observing accuracy and validation loss, we come up with a number of layers, and neurons to make our model not overfitting and give good accuracy.

Between each layer, we are normalizing values and applying the dropout layer before the final output. We are using relu activation till the final layer. And then we are using the sigmoid activation function for the final output.

```
BinaryClassification(
  (layer_1): Linear(in_features=70, out_features=512, bias=True)
  (layer_2): Linear(in_features=512, out_features=512, bias=True)
  (layer_3): Linear(in_features=512, out_features=512, bias=True)
  (layer_4): Linear(in_features=512, out_features=64, bias=True)
  (layer_out): Linear(in_features=64, out_features=1, bias=True)
  (Sigmoid): Sigmoid()
  (relu): ReLU()
  (dropout): Dropout(p=0.2, inplace=False)
  (batchnorm1): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (batchnorm2): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (batchnorm3): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (batchnorm4): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
)
```

After creating objects of the classes, we are passing some hyperparameters to our model.

- batch size=512, Learning Rate=0.0005

  We finalize these values after some experiments. Learning rate helps us to reach global minima, and stay at global minima to minimize/optimize loss.

- Loss: BCEWithLogitsLoss

  This loss is inbuilt in PyTorch that combines sigmoid and binary cross-entropy in one class. Our main problem is binary classification , so we have selected this loss function.

  For every epoch, during back propagation, the model calculates loss, finds gradients, and updates weights according to that.

  We are using Adam optimizer as an optimizer in our neural network. Adam optimizer uses the strengths of two gradients: momentum and rmsp gradients.

  We are running 100 epochs to train our model.

  After 100 epochs, the result (accuracy, validation loss, and many more) are as under.

  Epoch 100: | Train Loss: 0.50514 | Train Acc: 90.758| Valid Loss: 0.50811 | Valid Acc: 90.500

For checking the overfitting of the model, we have plotted training loss, validation loss for every epoch. Earlier For some structures and parameters, the model is overfitting as validation loss is going away from training loss.

After some experiments with neural network layers, structures, and hyperparameters, we found one architecture that has no overfitting, and accuracy is also good.

The validation loss & training loss graph for our model is as under.

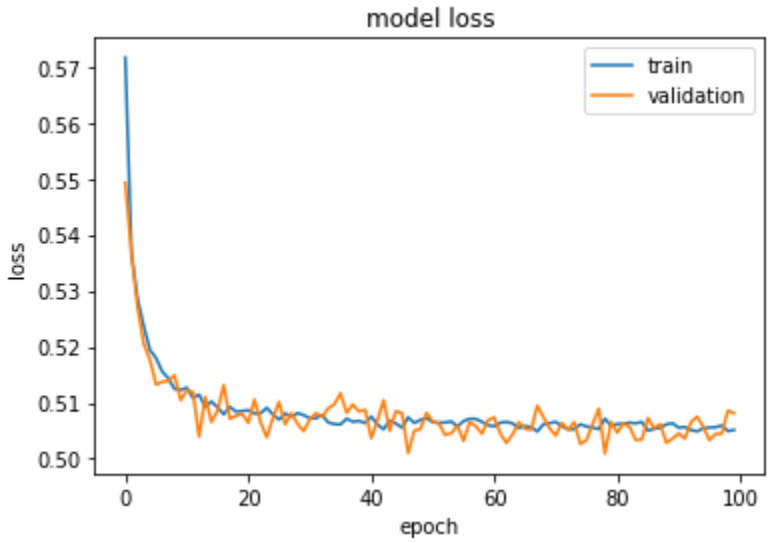

Figure 6: Validation and Training Loss

Train accuracy is 96% and validation accuracy is near to that. And performance on new/test data is also good. We got 88% Test accuracy.

```
: confusion_matrix(y_test, y_pred_list)

: array([[6458, 2053],
        [ 160,  332]])

: print(classification_report(y_test, y_pred_list))
              precision    recall  f1-score   support

         0.0       0.98      0.76      0.85      8511
         1.0       0.14      0.67      0.23       492

    accuracy                           0.75      9003
   macro avg       0.56      0.72      0.54      9003
weighted avg       0.93      0.75      0.82      9003
```

Pytorch gives the flexibility to create/modify neural network architecture but it is more complex to code/implement compared to Keras.

Using Keras, we can build models faster with less complexity in terms of code. So we are implementing a second neural network using Keras and TensorFlow libraries.

2.3.2 Neural Networks Using Tensorflow and Keras

Keras library is built on top of the TensorFlow library. Using Keras, we can easily add any layer/add specific neuron architecture whenever required, similarly shown as under.

```
:
my_classifier = Sequential()

# Adding the input layer AND the first hidden layer (Pay attention to this)
my_classifier.add(Dense(units = 70, kernel_initializer = 'uniform',
                        activation = 'relu', input_dim = n_features))

# Adding the second hidden layer
my_classifier.add(Dense(units = 8, kernel_initializer = 'uniform',
                        activation = 'relu'))

# Adding the last (output) layer
my_classifier.add(Dense(units = 1, kernel_initializer = 'uniform',
                        activation = 'sigmoid'))
```

Here for this neural network, we are taking one input layer, one hidden layer, and one output layer. We are taking 70 input neurons,1 output neuron, and 8 neurons for the hidden layer. These number of neurons and number of layers, we finalize after observing validation loss graphs.

These are sequential neural networks. All are dense layers. Sigmoid activation function for output and for remaining layers, we are using relu activation function.Kernel_initializer sets the default value of weights in a uniformly distributed manner.

Other hyperparameters we have used for this model are as under.

- We have used adam as an optimizer as adam combines the best properties of the AdaGrad and RMSProp algorithms.
- Learning rate is 0.0003 and the batch size is 64 based on observations.

- We train the model for 50 epochs
- We used binary_crossentopy as a loss function as our main problem is for binary classification.

After the model is trained, we plotted accuracy and loss graphs for checking overfitting of the model. Earlier, the model was overfitting but after changing neural networks and hyperparameters, we can see validation loss is near to train loss. So the model is not overfitting finally. Here, after 50 epochs,

train and valid loss came near to 0.02 from 0.5.

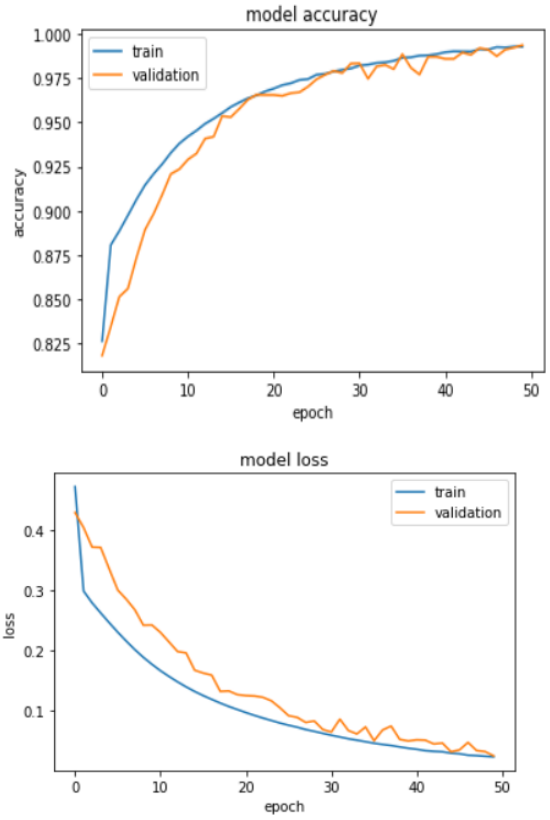

Figure 7: Model Accuracy and Model Loss

Train accuracy is 99.33% and validation accuracy is near to that. And performance on new/test data is also good. We got 93.40% Test accuracy.

3.0 CONCLUSION

We can observe that the Training Accuracy and Loss have reached their optimal point, which was our goal initially.

Thus using the Neural Network model with Keras and TF has been fruitful to predict at its best, whether a customer will churn or not.

We can now deploy this model and predict for different Customers and impart knowledge or insight to the Sales and Management Team to work on retaining the Customers who have the potential to churn by increasing their user experience in various ways.

4.0 REFERENCES

[1] https://www.atmosera.com/blog/binary-classification-with-neural-networks/

[2] https://www.kaggle.com/code/karthik7395/binary-classification-using-neural-networks/notebook

[3] https://towardsdatascience.com/predicting-and-preventing-the-churn-of-high-value-customers-using-machine-learning-adbb4a61095d

[4] https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification

[5] https://builtin.com/data-science/step-step-explanation-principal-component-analysis