UNIVERSITY COLLEGE DUBLIN

STAT40800 : Data Programming with Python

PROJECT REPORT

on

BANK CUSTOMER CHURN MODELLING USING PYTORCH

Submitted by,

Justin Joseph

Student Number : 18201354

Submitted on : Dec 09, 2018

INTRODUCTION

Customers have an important role in determining the success of any business and lack of customers lead to the downfall of any business. Every firm focuses on attracting more customers and retaining the existing ones. Losing customers to their competitors leads to the loss in a huge market share and thus it is of utmost importance to understand the fact that the key to success for any firm is its customer base. Nowadays firms focus on a consumer centric approach of business. Customer satisfaction is a very important factor in the modern day business models. In the modern day, the customer expectations are high and keeps changing, the business models have to cope up with this change. Lack of adaptability to new trends in the market will result in the failure of traditional businesses.

Churn or Customer churn or Customer attrition is simply the number of customers discontinuing the existing relationship with a business. It is easier and less expensive to retain an existing customer than to find new customers and hence churn is considered as a critical metric in the modern day business. The churn rate also shows the quality of service provided to the customer, ie if the churn rate is high, it means that more number of customers are not satisfied with the services. Depending on the churn rate, the business can also take measures to improve their services and come up with strategies so as to retain the customers.

Predicting the churn will help the business to identify the customers that are about to leave and the business can come up with strategies specifically focussed on that particular customers and thereby retain them. In this project, the business is a bank and the customer churn is predicted using few of the customer attributes. Prediction is done using classification technique (since the target variable has only two outcomes) and PyTorch package is used.

ABOUT THE DATASET

The dataset used is “Churn_Modelling” dataset and it is obtained from https://www.kaggle.com/barelydedicated/bank-customer-churn-modeling . It has 10000 rows and 14 columns. It contains the details of customers in a bank. The columns in the dataset are:

1. Row Number
2. CustomerId
3. Surname
4. CreditScore
5. Geography
6. Gender

7. Age
8. Tenure - Number of years of relationship with the bank
9. Balance
10. NumOfProducts - Number of products the customer has
11. HasCrCard - 1 for TRUE and 0 for FALSE
12. IsActiveMember - 1 for TRUE and 0 for FALSE
13. EstimatedSalary - Salary of the customer
14. Exited - 1 for TRUE and 0 for FALSE

NEW PACKAGES USED

1. **PyTorch**

   PyTorch is an open-source machine learning library for python, based on Torch, used for applications such as natural language processing (NLP). PyTorch enables fast, flexible experimentation and efficient production through a hybrid front-end, distributed training, and ecosystem of tools and libraries.

2. **Plotly**

   Plotly is an online collaborative data analysis and graphing tool. The Python API allows to access all of Plotly functionality from Python. Plotly can be used to produce online as well as offline plots.

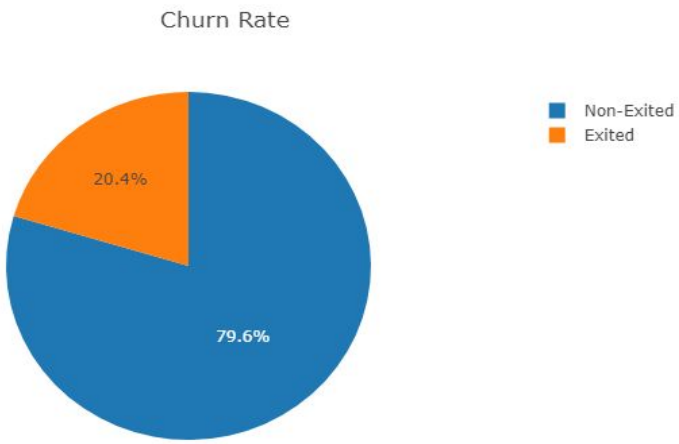
PROCEDURE

1. **Load the dataset, reformat the dataset and visualization**

   The data is loaded into python using **read_csv** from the **Pandas** library. The columns "**Row Number**", "**CustomerId**" and "**Surname**" is of no importance for the prediction and hence they are dropped from the dataset. Also the dataset is split into input variables and target variable ("**Exited**"). The below code is used for these tasks.

```
# Importing the dataset
data=pd.read_csv('C:/Users/HP/Downloads/Study/Python/Assignment/Project 2/Churn_Modelling.csv')
X=data.iloc[:,3:13].values # Removing the columns that are not necessary
y=data.iloc[:,13].values # Target variable
```

   Using the chart made with Plotly, let us look at the churn rate of the bank.



It can be seen that 20.4% of the existing customers are exiting the bank.

The code used for making the plot is given below.

```
# Pie Chart of churn rate using Plotly
total_churn=len(y)
exited_churn=y.sum()
non_exited_churn=total_churn-exited_churn
labels=['Exited','Non-Exited']
values=[exited_churn,non_exited_churn]
trace=go.Pie(labels=labels,values=values)
fig=go.Figure(data=[trace])
fig.layout.title='Churn Rate'
po.plot(fig,filename='Churn Rate')
```

2. Data Preparation

The categorical variables "**Geography**" and "**Gender**" are converted to numerical format using encoding.

```
# Change both the categorical columns to numerical form
X[:,1]=encode(X[:,1]) # For column Geography
X[:,2]=encode(X[:,2]) # For column Gender
onehotencoder=skp.OneHotEncoder(categorical_features=[1])
X=onehotencoder.fit_transform(X).toarray()
X=X[:,1:]
```

The data is split into train and test dataset. Training is done with 80% of the total data and 20% is used for testing.

```
# Split the data into train (80%) and test (20%)
X_train,X_test,y_train,y_test=skm.train_test_split(X,y,test_size=0.2,random_state=0)
```

Both the train and test input variables are then scaled using the code below:

```
# Scale the data
scaling=skp.StandardScaler()
X_train=scaling.fit_transform(X_train)
X_test=scaling.fit_transform(X_test)
```

For PyTorch, the data format has to be Tensor and hence we convert the dataset to Tensor format.

```
# Convert the data into Tensor format
train = data_utils.TensorDataset(torch.from_numpy(X_train).float(),
                                 torch.from_numpy(y_train).float())
test_set = torch.from_numpy(X_test).float()
test_valid = torch.from_numpy(y_test).float()
```

3. Define the Model

A model with 1 input layer, 2 hidden layer and 3 output layers is defined. The first 3 layers are linear layers. The 4th layer is the Sigmoid activation layer and the last layer applies the rectified linear unit function. "**init.xavier_normal**" fills the input tensor with values using normal distribution. The code for model creation is given below.

```
"""
   ---------------------------------------------------------------------------------------------------------
    Model Creation and defining the parameters
   ---------------------------------------------------------------------------------------------------------
"""
model = torch.nn.Sequential()

module = torch.nn.Linear(11,6)
init.xavier_normal(module.weight)
model.add_module("relu 1", module)

module = torch.nn.Linear(6,6)
init.xavier_normal(module.weight)
model.add_module("relu 2", module)

module = torch.nn.Linear(6,1)
init.xavier_normal(module.weight)
model.add_module("linear 3", module)

model.add_module("sig",torch.nn.Sigmoid())
model.add_module("relu",torch.nn.ReLU())

loss_fn = torch.nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.0007)
epochs = 50
dataloader = data_utils.DataLoader(train, batch_size=128, shuffle=False)
history = {"loss": [], "accuracy": [], "loss_val": [], "accuracy_val": []}
```

MSE loss function is used to calculate the loss of the model. "**torch.nn.MSELoss**" is used for this. Optimization is done using Adam algorithm ("**torch.optim.Adam")**. **DataLoader** combines a dataset and a sampler and provides a single or multi-process iterators over the dataset. The model is fit for 50 epochs with a learning rate of 0.0007. Each epoch will have a batch size of 128.

4. Run the Model

In PyTorch, the model has to be run manually for every epoch. Each epoch has the following

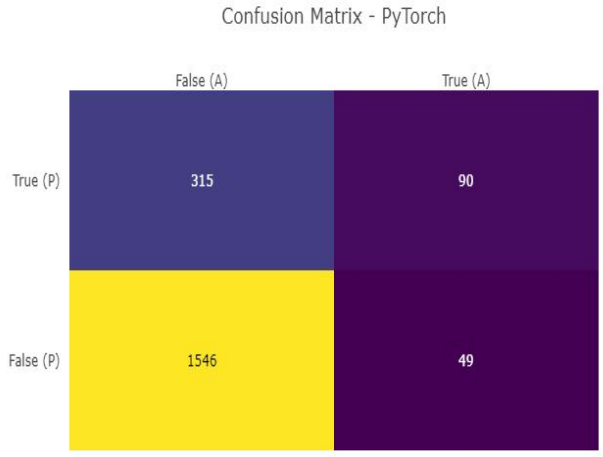
steps:

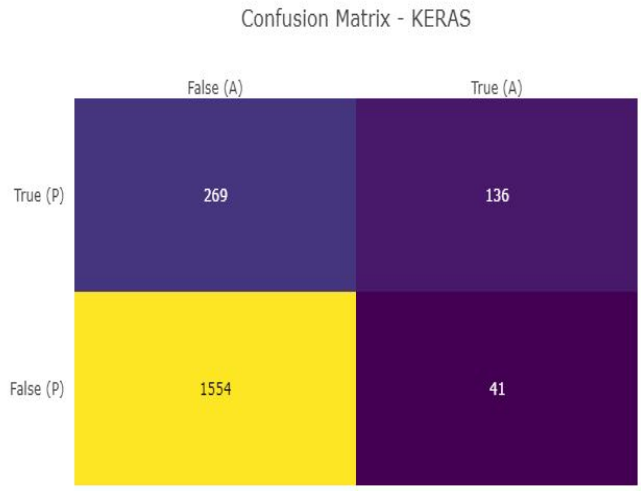
- "**y_pred=model(Variable(batch))**" compute model output.
- "**loss_fn()**" calculates Loss
- "**optimizer.zero_grad()**" clears the gradients of all optimized tensors.
- "**loss.backward()**" compute gradients of all variables with respect to loss.
- "**optimizer.step()**" perform updates using calculated gradients.

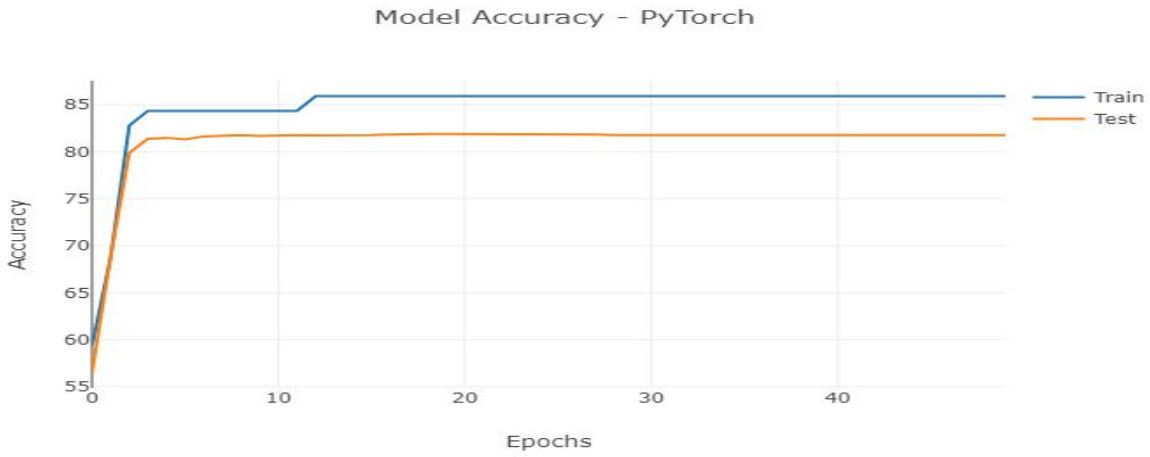
```
for epoch in range(epochs):
    loss = None
    for idx, (batch, target) in enumerate(dataloader):
        y_pred = model(Variable(batch))
        loss = loss_fn(y_pred, Variable(target.float()).reshape(len(Variable(target.float())),1))
        prediction = [1 if x > 0.5 else 0 for x in y_pred.data.numpy()]
        correct = (prediction == target.numpy()).sum()
        y_val_pred = model(Variable(test_set))
        loss_val = loss_fn(y_val_pred, Variable(test_valid.float()).reshape(len(Variable(test_valid.float())),1))
        prediction_val = [1 if x > 0.5 else 0 for x in y_val_pred.data.numpy()]
        correct_val = (prediction_val == test_valid.numpy()).sum()
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
    history["loss"].append(loss.data[0])
    history["accuracy"].append(100 * correct / len(prediction))
    history["loss_val"].append(loss_val.data[0])
    history["accuracy_val"].append(100 * correct_val / len(prediction_val))
    print("Loss, accuracy, val loss, val acc at epoch", epoch + 1, history["loss"][-1],
          history["accuracy"][-1], history["loss_val"][-1], history["accuracy_val"][-1])
```

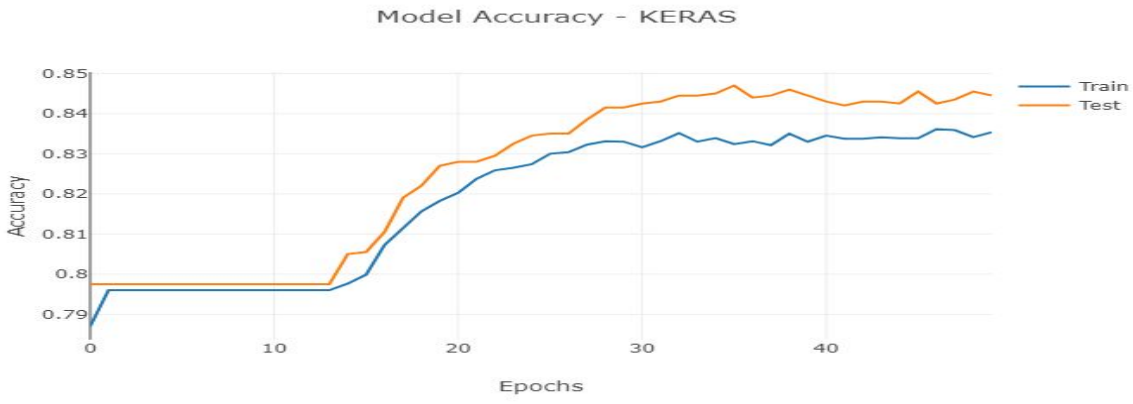
The loss and accuracy values are printed for each epochs.

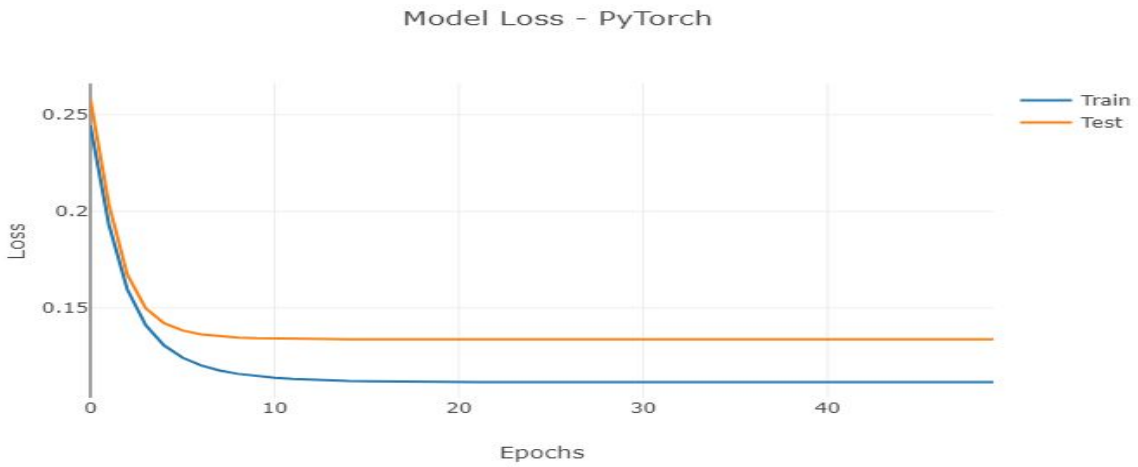
COMPARISON OF RESULTS BETWEEN PYTORCH AND KERAS

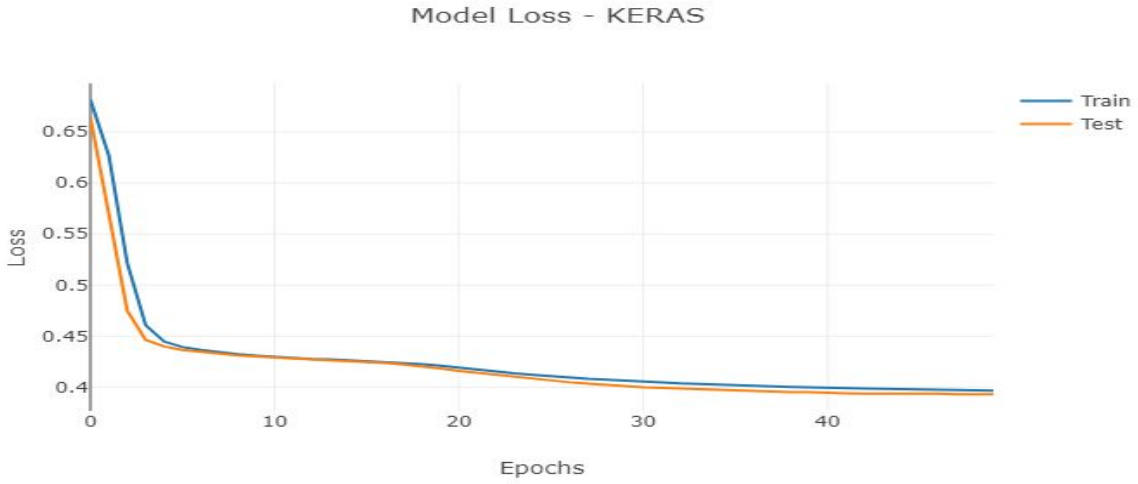



Confusion Matrix - KERAS
False (A)
True (A)
True (P)
False (P)
269
136
1554
41


Model Accuracy - PyTorch
Accuracy
Epochs
Train
Test
85
80
75
70
65
60
55
0
10
20
30
40


Model Accuracy - KERAS
Accuracy
Epochs
Train
Test
0.85
0.84
0.83
0.82
0.81
0.8
0.79
0
10
20
30
40





CONCLUSION

From the confusion matrix, it is can be seen that PyTorch model predicted 1636 values correctly and KERAS model predicted 1690 values correctly. The accuracy of the PyTorch model is higher than the KERAS model. The loss between the train and test data in KERAS model is lesser than that of the PyTorch model.

REFERENCES

1. Dataset : https://www.kaggle.com/barelydedicated/bank-customer-churn-modeling
2. Information on PyTorch : https://pytorch.org/
3. Information on Plotly : https://pypi.org/project/plotly/