In [2]:

```
import sys
import os
import json
import pandas
import numpy as np
import optparse

from keras.callbacks import TensorBoard
from keras import regularizers
from keras.models import Sequential, load_model
from keras.layers import LSTM, Dense, Dropout, Conv1D, GlobalMaxPooling1D, Flatten, MaxPooling1D
from keras.layers.embeddings import Embedding
from keras.preprocessing import sequence
from keras.preprocessing.text import Tokenizer
from keras.utils import to_categorical
from collections import OrderedDict

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.metrics import confusion_matrix, classification_report, roc_auc_score
from random import sample
import re
import matplotlib.pyplot as plt

import nltk
# nltk.download('stopwords')
from nltk.corpus import stopwords
```

In [3]:

```
def plot_history(history):
    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']
    loss = history.history['loss']
    val_loss = history.history['val_loss']
    x = range(1, len(acc) + 1)

    plt.figure(figsize=(12, 5))
    plt.subplot(1, 2, 1)
    plt.plot(x, acc, 'b', label='Training acc')
    plt.plot(x, val_acc, 'r', label='Validation acc')
    plt.title('Training and validation accuracy')
    plt.legend()
    plt.subplot(1, 2, 2)
    plt.plot(x, loss, 'b', label='Training loss')
    plt.plot(x, val_loss, 'r', label='Validation loss')
    plt.title('Training and validation loss')
    plt.legend()
```

In [33]:

```
# find the longest text to determine max_log_length
#length = [len(i) for i in X_encoded]
# max(length)
processed_data = pandas.read_csv("data_1w.csv")
processed_data
```

Out[33]:

	Unnamed: 0	0	1	2	3	4	5	6	7	8	...	2040	2041	2042	2043	2044	2045	2046	2047	Score	Label
0	0	0	0	0	0	0	0	0	0	0	...	191	213	64	343	213	64	3026	753	1	0
1	1	0	0	0	0	0	0	0	0	0	...	316	24	702	18	277	149	363	1674	1	0
2	2	0	0	0	0	0	0	0	0	0	...	696	223	106	532	495	16	1062	2018	1	0
3	3	0	0	0	0	0	0	0	0	0	...	18	59	199	287	298	1	726	484	1	0
4	4	0	0	0	0	0	0	0	0	0	...	3507	128	10	73	507	562	277	149	1	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
9995	9995	0	0	0	0	0	0	0	0	0	...	914	3	75	360	56	510	6778	14	4	1
9996	9996	0	0	0	0	0	0	0	0	0	...	170	2393	455	53	417	42	130	292	4	1
9997	9997	0	0	0	0	0	0	0	0	0	...	173	120	788	352	450	16	45	318	4	1
9998	9998	0	0	0	0	0	0	0	0	0	...	409	336	42	48	23	6	134	18865	4	1
9999	9999	0	0	0	0	0	0	0	0	0	...	275	693	50	34	627	782	418	514	4	1

10000 rows × 2051 columns

In [34]:

```
# pad the data as each observation has a different length
#max_log_length = 2048 # larger than 2003
#X_processed = sequence.pad_sequences(X_encoded, maxlen=max_log_length)
```

In [35]:

```
#X_processed
y = processed_data[["Label"]].values
avg_data = processed_data.iloc[:, 1:-2].values
# y -= 1
avg_data
```

Out[35]:

```
array([[    0,     0,     0, ...,    64,  3026,   753],
       [    0,     0,     0, ...,   149,   363,  1674],
       [    0,     0,     0, ...,    16,  1062,  2018],
       ...,
       [    0,     0,     0, ...,    16,    45,   318],
       [    0,     0,     0, ...,     6,   134, 18865],
       [    0,     0,     0, ...,   782,   418,   514]], dtype=int64)
```

In [36]:

```
avg_data = np.array(avg_data)
y = y.reshape((y.shape[0], 1))
```

In [37]:

```
# split 70% train and 30% test data
X_train, X_test, y_train, y_test = train_test_split(avg_data, y, test_size=0.30, random_state=0)
y_train_cat = to_categorical(y_train)
y_test_cat = to_categorical(y_test)
```

In [ ]:

XGBoost - Binary - tokenizer

In [7]:

```
import xgboost as xgb
```

In [8]:

```
xg_rdsearch = xgb.XGBClassifier()
```

In [9]:

```
param_rdsearch_xg = {'n_estimators':[100, 150, 200, 250, 300, 350, 400, 450, 500,550, 600, 650, 700, 750,
                                     850, 900, 950, 1000],
                     'learning_rate':[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1, 1.1, 1.2, 1.3, 1.4,
```

```
                    'max_depth':[1,2],
                    'gamma': [0, 0.25, 0.5, 0.75, 1, 1.25, 1.5, 1.75, 2, 2.25, 2.5, 2.75, 3, 3.25, 3.5, 3
                              4.5, 4.75, 5]}
+
```

```
xg_rdsearch = RandomizedSearchCV(xg_rdsearch, param_rdsearch_xg, cv = 5, scoring = 'accuracy',
                      refit = True, n_jobs=-1, verbose = 5)
```

In [10]:

```
xg_rdsearch.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
C:\Users\14264\Anaconda3\envs\tf_gpu\lib\site-packages\xgboost\sklearn.py:888: UserWarning: The use of l
abel encoder in XGBClassifier is deprecated and will be removed in a future release. To remove this
warning, do the following: 1) Pass option use_label_encoder=False when constructing XGBClassifier object;
and 2) Encode your labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
C:\Users\14264\Anaconda3\envs\tf_gpu\lib\site-packages\sklearn\utils\validation.py:63:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shap
e of y to (n_samples, ), for example using ravel().
  return f(*args, **kwargs)
```

```
[00:35:33] WARNING: ..\src\learner.cc:1061: Starting in XGBoost 1.3.0, the default evaluation metric
used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set
eval_metric if you'd like to restore the old behavior.
```

Out[10]:

```
RandomizedSearchCV(cv=5,
                   estimator=XGBClassifier(base_score=None, booster=None,
                                           colsample_bylevel=None,
                                           colsample_bynode=None,
                                           colsample_bytree=None, gamma=None,
                                           gpu_id=None, importance_type='gain',
                                           interaction_constraints=None,
                                           learning_rate=None,
                                           max_delta_step=None, max_depth=None,
                                           min_child_weight=None, missing=nan,
                                           monotone_constraints=None,
                                           n_estimators=100,...
                                           verbosity=None),
                   n_jobs=-1,
                   param_distributions={'gamma': [0, 0.25, 0.5, 0.75, 1, 1.25,
                                                  1.5, 1.75, 2, 2.25, 2.5, 2.75,
                                                  3, 3.25, 3.5, 3.75, 4, 4.25,
                                                  4.5, 4.75, 5],
                                        'learning_rate': [0.1, 0.2, 0.3, 0.4,
                                                          0.5, 0.6, 0.7, 0.8,
                                                          0.9, 1, 1.1, 1.2, 1.3,
                                                          1.4, 1.5, 1.6],
                                        'max_depth': [1, 2],
                                        'n_estimators': [100, 150, 200, 250,
                                                         300, 350, 400, 450,
                                                         500, 550, 600, 650,
                                                         700, 750, 800, 850,
                                                         900, 950, 1000]},
                   scoring='accuracy', verbose=5)
```

In [11]:

```
xg_bestparams = xg_rdsearch.best_params_
xg_bestparams
```

Out[11]:

```
{'n_estimators': 500, 'max_depth': 2, 'learning_rate': 0.5, 'gamma': 0.75}
```

In [12]:

```
xg_bestmodel = xg_rdsearch.best_estimator_
xg_bestmodel
```

Out[12]:

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0.75, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.5, max_delta_step=0, max_depth=2,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=500, n_jobs=8, num_parallel_tree=1, random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

In [13]:

```
y_train_pred = xg_bestmodel.predict(X_train)
```

In [14]:

```
print(classification_report(y_train_pred,y_train))
```

```
              precision    recall  f1-score   support

           0       0.92      0.91      0.92      3503
           1       0.91      0.92      0.92      3497

    accuracy                           0.92      7000
   macro avg       0.92      0.92      0.92      7000
weighted avg       0.92      0.92      0.92      7000
```

In [15]:

```
yhat_xg_bestmodel = xg_bestmodel.predict(X_test)
```

In [16]:

```
classi_reprt_xg = classification_report(y_test, yhat_xg_bestmodel)
```

In [17]:

```
print(classi_reprt_xg)
```

```
              precision    recall  f1-score   support

           0       0.76      0.75      0.75      1511
           1       0.75      0.75      0.75      1489

    accuracy                           0.75      3000
   macro avg       0.75      0.75      0.75      3000
weighted avg       0.75      0.75      0.75      3000
```

CNN - Binary - Tokenizer

In [52]:

```
CNN_Model = Sequential()
CNN_Model.add(Conv1D(10, 20, activation = 'sigmoid', input_shape = (2048, 1)))
CNN_Model.add(Conv1D(20, 8, activation = 'sigmoid'))
CNN_Model.add(Conv1D(30, 5, activation = 'sigmoid'))
CNN_Model.add(GlobalMaxPooling1D())
CNN_Model.add(Dense(10, activation = 'sigmoid'))
CNN_Model.add(Dense(1, activation = 'sigmoid'))
CNN_Model.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
CNN_Model.summary()
```

```
Model: "sequential_10"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv1d_22 (Conv1D)           (None, 2029, 10)          210       
_________________________________________________________________
conv1d_23 (Conv1D)           (None, 2022, 20)          1620      
_________________________________________________________________
conv1d_24 (Conv1D)           (None, 2018, 30)          3030      
_________________________________________________________________
global_max_pooling1d_4 (Glob (None, 30)                0         
_________________________________________________________________
dense_13 (Dense)             (None, 10)                310       
_________________________________________________________________
dense_14 (Dense)             (None, 1)                 11        
=================================================================
Total params: 5,181
Trainable params: 5,181
Non-trainable params: 0
_________________________________________________________________
```

In [53]:

```
CNN_history = CNN_Model.fit(X_train.reshape((X_train.shape[0], X_train.shape[1], 1)), y_train, epochs = 1
                            (X_test.reshape((X_test.shape[0], X_test.shape[1], 1)), y_test))
```

```
Train on 7000 samples, validate on 3000 samples
Epoch 1/100
7000/7000 [==============================] - 1s 185us/step - loss: 0.6936 - accuracy: 0.5013 - val_loss:
0.6932 - val_accuracy: 0.5037
Epoch 2/100
7000/7000 [==============================] - 1s 142us/step - loss: 0.6935 - accuracy: 0.5010 - val_loss:
0.6928 - val_accuracy: 0.5010
Epoch 3/100
7000/7000 [==============================] - 1s 142us/step - loss: 0.6932 - accuracy: 0.5029 - val_loss:
0.6927 - val_accuracy: 0.5093
Epoch 4/100
7000/7000 [==============================] - 1s 143us/step - loss: 0.6931 - accuracy: 0.5101 - val_loss:
0.6924 - val_accuracy: 0.5010
Epoch 5/100
7000/7000 [==============================] - 1s 145us/step - loss: 0.6920 - accuracy: 0.5313 - val_loss:
0.6919 - val_accuracy: 0.5490
Epoch 6/100
7000/7000 [==============================] - 1s 152us/step - loss: 0.6917 - accuracy: 0.5300 - val_loss:
0.6915 - val_accuracy: 0.5437
Epoch 7/100
7000/7000 [==============================] - 1s 132us/step - loss: 0.6906 - accuracy: 0.5244 - val_loss:
0.6907 - val_accuracy: 0.5373
Epoch 8/100
7000/7000 [==============================] - 1s 132us/step - loss: 0.6888 - accuracy: 0.5481 - val_loss:
0.6900 - val_accuracy: 0.5287
Epoch 9/100
7000/7000 [==============================] - 1s 132us/step - loss: 0.6897 - accuracy: 0.5354 - val_loss:
0.6905 - val_accuracy: 0.5413
Epoch 10/100
7000/7000 [==============================] - 1s 132us/step - loss: 0.6877 - accuracy: 0.5493 - val_loss:
0.6889 - val_accuracy: 0.5473
Epoch 11/100
7000/7000 [==============================] - 1s 132us/step - loss: 0.6866 - accuracy: 0.5474 - val_loss:
0.6888 - val_accuracy: 0.5360
Epoch 12/100
7000/7000 [==============================] - 1s 134us/step - loss: 0.6846 - accuracy: 0.5574 - val_loss:
0.6882 - val_accuracy: 0.5373
Epoch 13/100
7000/7000 [==============================] - 1s 137us/step - loss: 0.6840 - accuracy: 0.5543 - val_loss:
0.6929 - val_accuracy: 0.5330
Epoch 14/100
7000/7000 [==============================] - 1s 133us/step - loss: 0.6839 - accuracy: 0.5599 - val_loss:
0.6921 - val_accuracy: 0.5400
Epoch 15/100
7000/7000 [==============================] - 1s 136us/step - loss: 0.6839 - accuracy: 0.5569 - val_loss:
0.6869 - val_accuracy: 0.5483
Epoch 16/100
7000/7000 [==============================] - 1s 138us/step - loss: 0.6818 - accuracy: 0.5706 - val_loss:
0.6861 - val_accuracy: 0.5477
Epoch 17/100
7000/7000 [==============================] - 1s 133us/step - loss: 0.6819 - accuracy: 0.5689 - val_loss:
0.6860 - val_accuracy: 0.5497
```

```
Epoch 18/100
7000/7000 [==============================] - 1s 138us/step - loss: 0.6811 - accuracy: 0.5663 - val_loss:
0.6867 - val_accuracy: 0.5427
Epoch 19/100
7000/7000 [==============================] - 1s 181us/step - loss: 0.6801 - accuracy: 0.5687 - val_loss:
0.6869 - val_accuracy: 0.5457
Epoch 20/100
7000/7000 [==============================] - 1s 146us/step - loss: 0.6808 - accuracy: 0.5653 - val_loss:
0.6878 - val_accuracy: 0.5430
Epoch 21/100
7000/7000 [==============================] - 1s 133us/step - loss: 0.6806 - accuracy: 0.5731 - val_loss:
0.6906 - val_accuracy: 0.5457
Epoch 22/100
7000/7000 [==============================] - 1s 132us/step - loss: 0.6810 - accuracy: 0.5627 - val_loss:
0.6883 - val_accuracy: 0.5450
Epoch 23/100
7000/7000 [==============================] - 1s 156us/step - loss: 0.6805 - accuracy: 0.5656 - val_loss:
0.6857 - val_accuracy: 0.5467
Epoch 24/100
7000/7000 [==============================] - 1s 143us/step - loss: 0.6793 - accuracy: 0.5657 - val_loss:
0.6844 - val_accuracy: 0.5513
Epoch 25/100
7000/7000 [==============================] - 1s 146us/step - loss: 0.6789 - accuracy: 0.5691 - val_loss:
0.6846 - val_accuracy: 0.5510
Epoch 26/100
7000/7000 [==============================] - 1s 138us/step - loss: 0.6787 - accuracy: 0.5681 - val_loss:
0.6852 - val_accuracy: 0.5513
Epoch 27/100
7000/7000 [==============================] - 1s 133us/step - loss: 0.6796 - accuracy: 0.5680 - val_loss:
0.6852 - val_accuracy: 0.5583
Epoch 28/100
7000/7000 [==============================] - 1s 132us/step - loss: 0.6793 - accuracy: 0.5700 - val_loss:
0.6901 - val_accuracy: 0.5457
Epoch 29/100
7000/7000 [==============================] - 1s 132us/step - loss: 0.6778 - accuracy: 0.5761 - val_loss:
0.6846 - val_accuracy: 0.5573
Epoch 30/100
7000/7000 [==============================] - 1s 132us/step - loss: 0.6771 - accuracy: 0.5770 - val_loss:
0.6850 - val_accuracy: 0.5483
Epoch 31/100
7000/7000 [==============================] - 1s 132us/step - loss: 0.6769 - accuracy: 0.5739 - val_loss:
0.6854 - val_accuracy: 0.5480
Epoch 32/100
7000/7000 [==============================] - 1s 132us/step - loss: 0.6770 - accuracy: 0.5759 - val_loss:
0.6852 - val_accuracy: 0.5560
Epoch 33/100
7000/7000 [==============================] - 1s 132us/step - loss: 0.6763 - accuracy: 0.5790 - val_loss:
0.6852 - val_accuracy: 0.5640
Epoch 34/100
7000/7000 [==============================] - 1s 132us/step - loss: 0.6774 - accuracy: 0.5751 - val_loss:
0.6842 - val_accuracy: 0.5537
Epoch 35/100
7000/7000 [==============================] - 1s 132us/step - loss: 0.6770 - accuracy: 0.5773 - val_loss:
0.6856 - val_accuracy: 0.5540
Epoch 36/100
7000/7000 [==============================] - 1s 132us/step - loss: 0.6754 - accuracy: 0.5837 - val_loss:
0.6840 - val_accuracy: 0.5593
Epoch 37/100
7000/7000 [==============================] - 1s 137us/step - loss: 0.6753 - accuracy: 0.5816 - val_loss:
0.6843 - val_accuracy: 0.5573
Epoch 38/100
7000/7000 [==============================] - 1s 132us/step - loss: 0.6770 - accuracy: 0.5760 - val_loss:
0.6845 - val_accuracy: 0.5610
Epoch 39/100
7000/7000 [==============================] - 1s 132us/step - loss: 0.6760 - accuracy: 0.5800 - val_loss:
0.6857 - val_accuracy: 0.5540
Epoch 40/100
7000/7000 [==============================] - 1s 137us/step - loss: 0.6757 - accuracy: 0.5794 - val_loss:
0.6897 - val_accuracy: 0.5433
Epoch 41/100
7000/7000 [==============================] - 1s 142us/step - loss: 0.6760 - accuracy: 0.5764 - val_loss:
0.6868 - val_accuracy: 0.5483
Epoch 42/100
7000/7000 [==============================] - 1s 137us/step - loss: 0.6763 - accuracy: 0.5756 - val_loss:
0.6845 - val_accuracy: 0.5500
Epoch 43/100
7000/7000 [==============================] - 1s 149us/step - loss: 0.6759 - accuracy: 0.5784 - val_loss:
```

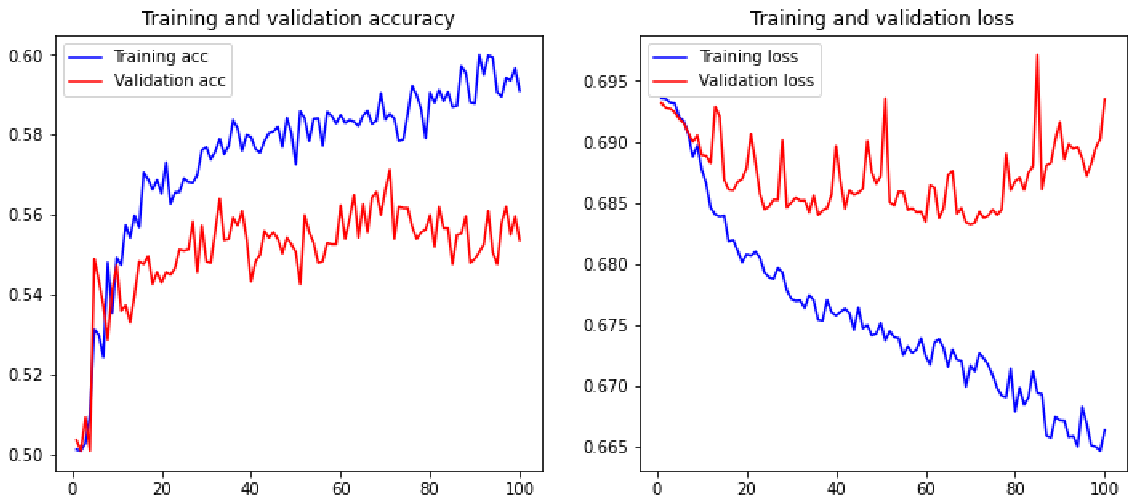
```
0.6860 - val_accuracy: 0.5560
Epoch 44/100
7000/7000 [==============================] - 1s 135us/step - loss: 0.6745 - accuracy: 0.5804 - val_loss:
0.6856 - val_accuracy: 0.5543
Epoch 45/100
7000/7000 [==============================] - 1s 137us/step - loss: 0.6764 - accuracy: 0.5809 - val_loss:
0.6858 - val_accuracy: 0.5557
Epoch 46/100
7000/7000 [==============================] - 1s 139us/step - loss: 0.6747 - accuracy: 0.5820 - val_loss:
0.6861 - val_accuracy: 0.5540
Epoch 47/100
7000/7000 [==============================] - 1s 151us/step - loss: 0.6749 - accuracy: 0.5770 - val_loss:
0.6901 - val_accuracy: 0.5503
Epoch 48/100
7000/7000 [==============================] - 1s 139us/step - loss: 0.6741 - accuracy: 0.5843 - val_loss:
0.6875 - val_accuracy: 0.5543
Epoch 49/100
7000/7000 [==============================] - 1s 142us/step - loss: 0.6743 - accuracy: 0.5806 - val_loss:
0.6865 - val_accuracy: 0.5527
Epoch 50/100
7000/7000 [==============================] - 1s 140us/step - loss: 0.6751 - accuracy: 0.5726 - val_loss:
0.6872 - val_accuracy: 0.5507
Epoch 51/100
7000/7000 [==============================] - 1s 133us/step - loss: 0.6737 - accuracy: 0.5859 - val_loss:
0.6936 - val_accuracy: 0.5427
Epoch 52/100
7000/7000 [==============================] - 1s 132us/step - loss: 0.6744 - accuracy: 0.5841 - val_loss:
0.6850 - val_accuracy: 0.5600
Epoch 53/100
7000/7000 [==============================] - 1s 132us/step - loss: 0.6740 - accuracy: 0.5784 - val_loss:
0.6847 - val_accuracy: 0.5557
Epoch 54/100
7000/7000 [==============================] - 1s 132us/step - loss: 0.6739 - accuracy: 0.5840 - val_loss:
0.6859 - val_accuracy: 0.5530
Epoch 55/100
7000/7000 [==============================] - 1s 132us/step - loss: 0.6725 - accuracy: 0.5841 - val_loss:
0.6859 - val_accuracy: 0.5480
Epoch 56/100
7000/7000 [==============================] - 1s 132us/step - loss: 0.6732 - accuracy: 0.5773 - val_loss:
0.6844 - val_accuracy: 0.5483
Epoch 57/100
7000/7000 [==============================] - 1s 137us/step - loss: 0.6727 - accuracy: 0.5857 - val_loss:
0.6845 - val_accuracy: 0.5530
Epoch 58/100
7000/7000 [==============================] - 1s 157us/step - loss: 0.6730 - accuracy: 0.5847 - val_loss:
0.6842 - val_accuracy: 0.5527
Epoch 59/100
7000/7000 [==============================] - 1s 132us/step - loss: 0.6739 - accuracy: 0.5829 - val_loss:
0.6843 - val_accuracy: 0.5527
Epoch 60/100
7000/7000 [==============================] - 1s 132us/step - loss: 0.6724 - accuracy: 0.5850 - val_loss:
0.6834 - val_accuracy: 0.5623
Epoch 61/100
7000/7000 [==============================] - 1s 147us/step - loss: 0.6717 - accuracy: 0.5830 - val_loss:
0.6864 - val_accuracy: 0.5540
Epoch 62/100
7000/7000 [==============================] - 1s 147us/step - loss: 0.6735 - accuracy: 0.5837 - val_loss:
0.6862 - val_accuracy: 0.5590
Epoch 63/100
7000/7000 [==============================] - 1s 141us/step - loss: 0.6738 - accuracy: 0.5833 - val_loss:
0.6837 - val_accuracy: 0.5650
Epoch 64/100
7000/7000 [==============================] - 1s 137us/step - loss: 0.6731 - accuracy: 0.5821 - val_loss:
0.6845 - val_accuracy: 0.5543
Epoch 65/100
7000/7000 [==============================] - 1s 133us/step - loss: 0.6715 - accuracy: 0.5849 - val_loss:
0.6872 - val_accuracy: 0.5627
Epoch 66/100
7000/7000 [==============================] - 1s 133us/step - loss: 0.6729 - accuracy: 0.5860 - val_loss:
0.6876 - val_accuracy: 0.5557
Epoch 67/100
7000/7000 [==============================] - 1s 165us/step - loss: 0.6721 - accuracy: 0.5827 - val_loss:
0.6841 - val_accuracy: 0.5643
Epoch 68/100
7000/7000 [==============================] - 1s 138us/step - loss: 0.6720 - accuracy: 0.5836 - val_loss:
0.6845 - val_accuracy: 0.5657
Epoch 69/100
```

```
7000/7000 [==============================] - 1s 133us/step - loss: 0.6699 - accuracy: 0.5904 - val_loss:
0.6834 - val_accuracy: 0.5600
Epoch 70/100
7000/7000 [==============================] - 1s 133us/step - loss: 0.6716 - accuracy: 0.5839 - val_loss:
0.6832 - val_accuracy: 0.5663
Epoch 71/100
7000/7000 [==============================] - 1s 133us/step - loss: 0.6711 - accuracy: 0.5853 - val_loss:
0.6834 - val_accuracy: 0.5713
Epoch 72/100
7000/7000 [==============================] - 1s 134us/step - loss: 0.6726 - accuracy: 0.5840 - val_loss:
0.6843 - val_accuracy: 0.5540
Epoch 73/100
7000/7000 [==============================] - 1s 135us/step - loss: 0.6722 - accuracy: 0.5784 - val_loss:
0.6837 - val_accuracy: 0.5620
Epoch 74/100
7000/7000 [==============================] - 1s 142us/step - loss: 0.6717 - accuracy: 0.5789 - val_loss:
0.6839 - val_accuracy: 0.5617
Epoch 75/100
7000/7000 [==============================] - 1s 146us/step - loss: 0.6708 - accuracy: 0.5853 - val_loss:
0.6844 - val_accuracy: 0.5617
Epoch 76/100
7000/7000 [==============================] - 1s 137us/step - loss: 0.6698 - accuracy: 0.5923 - val_loss:
0.6840 - val_accuracy: 0.5570
Epoch 77/100
7000/7000 [==============================] - 1s 144us/step - loss: 0.6692 - accuracy: 0.5899 - val_loss:
0.6844 - val_accuracy: 0.5540
Epoch 78/100
7000/7000 [==============================] - 1s 144us/step - loss: 0.6690 - accuracy: 0.5864 - val_loss:
0.6890 - val_accuracy: 0.5557
Epoch 79/100
7000/7000 [==============================] - 1s 138us/step - loss: 0.6714 - accuracy: 0.5791 - val_loss:
0.6860 - val_accuracy: 0.5563
Epoch 80/100
7000/7000 [==============================] - 1s 133us/step - loss: 0.6679 - accuracy: 0.5906 - val_loss:
0.6867 - val_accuracy: 0.5600
Epoch 81/100
7000/7000 [==============================] - 1s 141us/step - loss: 0.6698 - accuracy: 0.5880 - val_loss:
0.6871 - val_accuracy: 0.5520
Epoch 82/100
7000/7000 [==============================] - 1s 142us/step - loss: 0.6684 - accuracy: 0.5913 - val_loss:
0.6860 - val_accuracy: 0.5620
Epoch 83/100
7000/7000 [==============================] - 1s 140us/step - loss: 0.6690 - accuracy: 0.5884 - val_loss:
0.6875 - val_accuracy: 0.5567
Epoch 84/100
7000/7000 [==============================] - 1s 135us/step - loss: 0.6712 - accuracy: 0.5907 - val_loss:
0.6880 - val_accuracy: 0.5567
Epoch 85/100
7000/7000 [==============================] - 1s 141us/step - loss: 0.6694 - accuracy: 0.5870 - val_loss:
0.6971 - val_accuracy: 0.5477
Epoch 86/100
7000/7000 [==============================] - 1s 136us/step - loss: 0.6693 - accuracy: 0.5873 - val_loss:
0.6861 - val_accuracy: 0.5550
Epoch 87/100
7000/7000 [==============================] - 1s 138us/step - loss: 0.6659 - accuracy: 0.5973 - val_loss:
0.6880 - val_accuracy: 0.5553
Epoch 88/100
7000/7000 [==============================] - 1s 141us/step - loss: 0.6657 - accuracy: 0.5954 - val_loss:
0.6883 - val_accuracy: 0.5597
Epoch 89/100
7000/7000 [==============================] - 1s 145us/step - loss: 0.6674 - accuracy: 0.5881 - val_loss:
0.6902 - val_accuracy: 0.5480
Epoch 90/100
7000/7000 [==============================] - 1s 148us/step - loss: 0.6672 - accuracy: 0.5879 - val_loss:
0.6916 - val_accuracy: 0.5490
Epoch 91/100
7000/7000 [==============================] - 1s 144us/step - loss: 0.6671 - accuracy: 0.6000 - val_loss:
0.6885 - val_accuracy: 0.5507
Epoch 92/100
7000/7000 [==============================] - 1s 137us/step - loss: 0.6658 - accuracy: 0.5950 - val_loss:
0.6898 - val_accuracy: 0.5527
Epoch 93/100
7000/7000 [==============================] - 1s 134us/step - loss: 0.6659 - accuracy: 0.5999 - val_loss:
0.6894 - val_accuracy: 0.5610
Epoch 94/100
7000/7000 [==============================] - 1s 133us/step - loss: 0.6650 - accuracy: 0.5994 - val_loss:
0.6896 - val_accuracy: 0.5507
```

```
Epoch 95/100
7000/7000 [==============================] - 1s 137us/step - loss: 0.6683 - accuracy: 0.5906 - val_loss: 0.6886 - val_accuracy: 0.5477
Epoch 96/100
7000/7000 [==============================] - 1s 135us/step - loss: 0.6669 - accuracy: 0.5896 - val_loss: 0.6872 - val_accuracy: 0.5580
Epoch 97/100
7000/7000 [==============================] - 1s 136us/step - loss: 0.6651 - accuracy: 0.5943 - val_loss: 0.6882 - val_accuracy: 0.5620
Epoch 98/100
7000/7000 [==============================] - 1s 133us/step - loss: 0.6650 - accuracy: 0.5936 - val_loss: 0.6895 - val_accuracy: 0.5550
Epoch 99/100
7000/7000 [==============================] - 1s 142us/step - loss: 0.6647 - accuracy: 0.5967 - val_loss: 0.6902 - val_accuracy: 0.5597
Epoch 100/100
7000/7000 [==============================] - 1s 134us/step - loss: 0.6663 - accuracy: 0.5910 - val_loss: 0.6935 - val_accuracy: 0.5537
```

In [54]:

```
plot_history(CNN_history)
```



In [56]:

```
y_train_pred = CNN_Model.predict(X_train.reshape(X_train.shape[0], X_train.shape[1], 1))

y_train_pred = np.where(y_train_pred < 0.5, 0, 1)

print(classification_report(y_train_pred, y_train))

yhat_xg_bestmodel = CNN_Model.predict(X_test.reshape(X_test.shape[0], X_test.shape[1], 1))

yhat_xg_bestmodel = np.where(yhat_xg_bestmodel < 0.5, 0, 1)

classi_reprt_xg = classification_report(yhat_xg_bestmodel, y_test)

print(classi_reprt_xg)
```

```
              precision    recall  f1-score   support

           0       0.52      0.60      0.56      3006
           1       0.66      0.58      0.62      3994

    accuracy                           0.59      7000
   macro avg       0.59      0.59      0.59      7000
weighted avg       0.60      0.59      0.59      7000

              precision    recall  f1-score   support

           0       0.48      0.57      0.52      1276
           1       0.63      0.54      0.58      1724

    accuracy                           0.55      3000
   macro avg       0.55      0.56      0.55      3000
weighted avg       0.57      0.55      0.56      3000
```

In [ ]:

XGBoost - Binary - Word2Vec

In [18]:

```
# find the longest text to determine max_log_length
#length = [len(i) for i in X_encoded]
# max(length)
processed_data = pandas.read_csv("data_word2vec.csv")
processed_data
```

Out[18]:

	Unnamed: 0	0	1	2	3	4	5	6	7	8	...	42	43	44
0	0	-0.046881	-0.015105	0.092017	-0.401055	-0.090607	-0.325169	-0.297168	-0.252889	0.431492	...	-0.535171	0.110634	-0.378137
1	1	-0.060067	0.014853	-0.000460	-0.312015	-0.175160	-0.217285	-0.240094	-0.142320	0.265984	...	-0.597190	0.062401	-0.313474
2	2	-0.061501	0.073079	0.033006	-0.367333	-0.133849	-0.395722	-0.239342	-0.159746	0.462475	...	-0.617383	0.151002	-0.353643
3	3	0.657267	-0.984994	0.582354	-0.502777	0.039903	-0.415720	-0.229914	-1.442688	1.223933	...	-0.566871	-0.438947	-0.868708
4	4	0.248054	-0.533198	0.372260	-0.621143	0.119701	-0.443986	-0.274354	-0.953549	0.972687	...	-0.530449	-0.079765	-0.859161
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
9995	9995	0.020909	-0.195280	0.117890	-0.686506	0.043509	-0.537159	-0.368239	-0.500929	0.594955	...	-0.569722	0.234621	-0.381084
9996	9996	-0.072137	-0.062902	0.131932	-0.784530	-0.031594	-0.320527	-0.411374	-0.429918	0.767537	...	-0.765428	0.371071	-0.570000
9997	9997	0.102276	-0.047583	0.443892	-0.589184	-0.012078	-0.517465	-0.362354	-0.514404	0.876258	...	-0.332553	0.242859	-0.628040
9998	9998	0.006259	-0.005005	0.135583	-0.375453	-0.426059	-0.452148	-0.376979	-0.220052	0.507017	...	-0.746278	-0.164733	-0.264629
9999	9999	-0.541975	0.379249	-0.034109	-1.336829	0.046721	-0.630619	-0.939171	-0.223359	0.905672	...	-0.467232	0.915110	-0.575144

10000 rows × 53 columns

In [19]:

```
# pad the data as each observation has a different length
#max_log_length = 2048 # larger than 2003
#X_processed = sequence.pad_sequences(X_encoded, maxlen=max_log_length)
```

In [20]:

```
#X_processed
y = processed_data[["Label"]].values
avg_data = processed_data.iloc[:, 1:-2].values
# y -= 1
avg_data
```

Out[20]:

```
array([[-4.68814398e-02, -1.51045953e-02,  9.20168824e-02, ...,
        -3.73423553e-01, -4.77381344e-01,  3.07678892e-01],
       [-6.00674681e-02,  1.48525748e-02, -4.60325505e-04, ...,
        -2.95136365e-01, -3.68211810e-01,  3.59656972e-01],
       [-6.15012782e-02,  7.30789267e-02,  3.30062611e-02, ...,
        -2.75688761e-01, -4.71716609e-01,  4.11675429e-01],
       ...,
       [ 1.02276118e-01, -4.75825627e-02,  4.43892363e-01, ...,
        -4.47013392e-01, -7.74893663e-01, -1.60717556e-02],
       [ 6.25933642e-03, -5.00470836e-03,  1.35582868e-01, ...,
        -2.38587142e-01, -4.32617757e-01,  5.29272652e-01],
       [-5.41975147e-01,  3.79249226e-01, -3.41089983e-02, ...,
        -4.11995113e-01, -1.29827176e-01,  3.93914914e-01]])
```

In [21]:

```
avg_data = np.array(avg_data)
y = y.reshape((y.shape[0], 1))
```

In [22]:

```
# split 70% train and 30% test data
X_train, X_test, y_train, y_test = train_test_split(avg_data, y, test_size=0.30, random_state=0)
y_train_cat = to_categorical(y_train)
y_test_cat = to_categorical(y_test)
```

In [ ]:

In [27]:

```
import xgboost as xgb
```

In [23]:

```
xg_rdsearch = xgb.XGBClassifier(alpha = 0.1)
```

In [24]:

```
param_rdsearch_xg = {'n_estimators':[100, 150, 200, 250, 300, 350, 400, 450, 500,550, 600, 650, 700, 750,
                                     850, 900, 950, 1000],
                     'learning_rate':[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1, 1.1, 1.2, 1.3, 1.4,
                     'max_depth':[1,2],
                     'gamma': [0, 0.25, 0.5, 0.75, 1, 1.25, 1.5, 1.75, 2, 2.25, 2.5, 2.75, 3, 3.25, 3.5, 3
                               4.5, 4.75, 5]}


xg_rdsearch = RandomizedSearchCV(xg_rdsearch, param_rdsearch_xg, cv = 5, scoring = 'roc_auc',
                       refit = True, n_jobs=-1, verbose = 5)
```

In [25]:

```
xg_rdsearch.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
C:\Users\14264\Anaconda3\envs\tf_gpu\lib\site-packages\xgboost\sklearn.py:888: UserWarning: The use of l
abel encoder in XGBClassifier is deprecated and will be removed in a future release. To remove this
warning, do the following: 1) Pass option use_label_encoder=False when constructing XGBClassifier object;
and 2) Encode your labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
C:\Users\14264\Anaconda3\envs\tf_gpu\lib\site-packages\sklearn\utils\validation.py:63:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shap
e of y to (n_samples, ), for example using ravel().
  return f(*args, **kwargs)
```

```
[01:25:37] WARNING: ..\src\learner.cc:1061: Starting in XGBoost 1.3.0, the default evaluation metric
used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set
eval_metric if you'd like to restore the old behavior.
```

Out[25]:

```
RandomizedSearchCV(cv=5,
                   estimator=XGBClassifier(alpha=0.1, base_score=None,
                                           booster=None, colsample_bylevel=None,
                                           colsample_bynode=None,
                                           colsample_bytree=None, gamma=None,
                                           gpu_id=None, importance_type='gain',
                                           interaction_constraints=None,
                                           learning_rate=None,
                                           max_delta_step=None, max_depth=None,
                                           min_child_weight=None, missing=nan,
                                           monotone_constraints=None,
                                           n_estim...
                                           verbosity=None),
                   n_jobs=-1,
                   param_distributions={'gamma': [0, 0.25, 0.5, 0.75, 1, 1.25,
                                                  1.5, 1.75, 2, 2.25, 2.5, 2.75,
                                                  3, 3.25, 3.5, 3.75, 4, 4.25,
                                                  4.5, 4.75, 5],
                                        'learning_rate': [0.1, 0.2, 0.3, 0.4,
                                                          0.5, 0.6, 0.7, 0.8,
                                                          0.9, 1, 1.1, 1.2, 1.3,
                                                          1.4, 1.5, 1.6],
                                        'max_depth': [1, 2],
                                        'n_estimators': [100, 150, 200, 250,
                                                         300, 350, 400, 450,
                                                         500, 550, 600, 650,
                                                         700, 750, 800, 850,
                                                         900, 950, 1000]},
                   scoring='roc_auc', verbose=5)
```

In [26]:

```
xg_bestparams = xg_rdsearch.best_params_
xg_bestparams
```

Out[26]:

```
{'n_estimators': 650, 'max_depth': 1, 'learning_rate': 0.3, 'gamma': 4.25}
```

In [27]:

```
xg_bestmodel = xg_rdsearch.best_estimator_
xg_bestmodel
```

Out[27]:

```
XGBClassifier(alpha=0.1, base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=4.25, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.3, max_delta_step=0, max_depth=1,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=650, n_jobs=8, num_parallel_tree=1, random_state=0,
              reg_alpha=0.100000001, reg_lambda=1, scale_pos_weight=1,
              subsample=1, tree_method='exact', validate_parameters=1,
              verbosity=None)
```

In [78]:

```
y_train_pred = xg_bestmodel.predict(X_train)
```

In [79]:

```
print(classification_report(y_train_pred,y_train))
```

```
              precision    recall  f1-score   support

           0       0.74      0.75      0.75      3467
           1       0.75      0.74      0.75      3533

    accuracy                           0.75      7000
   macro avg       0.75      0.75      0.75      7000
weighted avg       0.75      0.75      0.75      7000
```

In [28]:

```
yhat_xg_bestmodel = xg_bestmodel.predict(X_test)
```

In [29]:

```
classi_reprt_xg = classification_report(y_test, yhat_xg_bestmodel)
```

In [30]:

```
print(classi_reprt_xg)
```

```
              precision    recall  f1-score   support

           0       0.72      0.70      0.71      1494
           1       0.71      0.73      0.72      1506

    accuracy                           0.72      3000
   macro avg       0.72      0.72      0.72      3000
weighted avg       0.72      0.72      0.72      3000
```

CNN - Binary - word2vec

In [25]:

```
CNN_Model = Sequential()
CNN_Model.add(Conv1D(10, 20, activation = 'sigmoid', input_shape = (50, 1)))
CNN_Model.add(Conv1D(20, 8, activation = 'sigmoid'))
CNN_Model.add(Conv1D(30, 5, activation = 'sigmoid'))
CNN_Model.add(GlobalMaxPooling1D())
CNN_Model.add(Dense(10, activation = 'sigmoid'))
CNN_Model.add(Dense(1, activation = 'sigmoid'))
CNN_Model.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
CNN_Model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv1d_6 (Conv1D)            (None, 31, 10)            210       
_________________________________________________________________
conv1d_7 (Conv1D)            (None, 24, 20)            1620      
_________________________________________________________________
conv1d_8 (Conv1D)            (None, 20, 30)            3030      
_________________________________________________________________
global_max_pooling1d_2 (Glob (None, 30)                0         
_________________________________________________________________
dense_4 (Dense)              (None, 10)                310       
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 11        
=================================================================
Total params: 5,181
Trainable params: 5,181
Non-trainable params: 0
_________________________________________________________________
```

In [26]:

```
CNN_history = CNN_Model.fit(X_train.reshape((X_train.shape[0], X_train.shape[1], 1)), y_train,
              epochs = 100, batch_size = 50,
              validation_data = (X_test.reshape((X_test.shape[0], X_test.shape[1], 1)), y_test))
```

```
Train on 7000 samples, validate on 3000 samples
Epoch 1/100
7000/7000 [==============================] - 1s 94us/step - loss: 0.6972 - accuracy: 0.4926 - val_loss:
0.6931 - val_accuracy: 0.5020
Epoch 2/100
7000/7000 [==============================] - 0s 62us/step - loss: 0.6932 - accuracy: 0.4996 - val_loss:
```

```
7000/7000 [==============================] - 0s 62us/step - loss: 0.6932 - accuracy: 0.4996 - val_loss:
0.6932 - val_accuracy: 0.4980
Epoch 3/100
7000/7000 [==============================] - 0s 61us/step - loss: 0.6934 - accuracy: 0.5023 - val_loss:
0.6928 - val_accuracy: 0.5020
Epoch 4/100
7000/7000 [==============================] - 0s 61us/step - loss: 0.6932 - accuracy: 0.5120 - val_loss:
0.6925 - val_accuracy: 0.4980
Epoch 5/100
7000/7000 [==============================] - 0s 60us/step - loss: 0.6927 - accuracy: 0.5126 - val_loss:
0.6918 - val_accuracy: 0.4980
Epoch 6/100
7000/7000 [==============================] - 0s 68us/step - loss: 0.6897 - accuracy: 0.5503 - val_loss:
0.6862 - val_accuracy: 0.5677
Epoch 7/100
7000/7000 [==============================] - 0s 61us/step - loss: 0.6776 - accuracy: 0.5986 - val_loss:
0.6658 - val_accuracy: 0.6140
Epoch 8/100
7000/7000 [==============================] - 0s 61us/step - loss: 0.6586 - accuracy: 0.6176 - val_loss:
0.6487 - val_accuracy: 0.6333
Epoch 9/100
7000/7000 [==============================] - 0s 61us/step - loss: 0.6458 - accuracy: 0.6296 - val_loss:
0.6398 - val_accuracy: 0.6440
Epoch 10/100
7000/7000 [==============================] - 0s 64us/step - loss: 0.6374 - accuracy: 0.6390 - val_loss:
0.6306 - val_accuracy: 0.6533
Epoch 11/100
7000/7000 [==============================] - 0s 61us/step - loss: 0.6286 - accuracy: 0.6490 - val_loss:
0.6290 - val_accuracy: 0.6523
Epoch 12/100
7000/7000 [==============================] - 0s 61us/step - loss: 0.6187 - accuracy: 0.6586 - val_loss:
0.6091 - val_accuracy: 0.6690
Epoch 13/100
7000/7000 [==============================] - 0s 61us/step - loss: 0.6117 - accuracy: 0.6693 - val_loss:
0.5999 - val_accuracy: 0.6780
Epoch 14/100
7000/7000 [==============================] - 0s 60us/step - loss: 0.6038 - accuracy: 0.6759 - val_loss:
0.5930 - val_accuracy: 0.6843
Epoch 15/100
7000/7000 [==============================] - 0s 61us/step - loss: 0.5992 - accuracy: 0.6794 - val_loss:
0.5879 - val_accuracy: 0.6923
Epoch 16/100
7000/7000 [==============================] - 0s 60us/step - loss: 0.5976 - accuracy: 0.6817 - val_loss:
0.5848 - val_accuracy: 0.6930
Epoch 17/100
7000/7000 [==============================] - 0s 60us/step - loss: 0.5925 - accuracy: 0.6800 - val_loss:
0.5826 - val_accuracy: 0.6973
Epoch 18/100
7000/7000 [==============================] - 0s 61us/step - loss: 0.5906 - accuracy: 0.6907 - val_loss:
0.5811 - val_accuracy: 0.6953
Epoch 19/100
7000/7000 [==============================] - 0s 60us/step - loss: 0.5866 - accuracy: 0.6917 - val_loss:
0.5817 - val_accuracy: 0.6920
Epoch 20/100
7000/7000 [==============================] - 0s 61us/step - loss: 0.5850 - accuracy: 0.6940 - val_loss:
0.5745 - val_accuracy: 0.7020
Epoch 21/100
7000/7000 [==============================] - 0s 65us/step - loss: 0.5827 - accuracy: 0.6936 - val_loss:
0.5740 - val_accuracy: 0.7003
Epoch 22/100
7000/7000 [==============================] - 0s 65us/step - loss: 0.5790 - accuracy: 0.7017 - val_loss:
0.5746 - val_accuracy: 0.6993
Epoch 23/100
7000/7000 [==============================] - 0s 60us/step - loss: 0.5763 - accuracy: 0.7009 - val_loss:
0.5720 - val_accuracy: 0.7007
Epoch 24/100
7000/7000 [==============================] - 0s 66us/step - loss: 0.5769 - accuracy: 0.7010 - val_loss:
0.5692 - val_accuracy: 0.7017
Epoch 25/100
7000/7000 [==============================] - 0s 66us/step - loss: 0.5749 - accuracy: 0.7020 - val_loss:
0.5660 - val_accuracy: 0.7100
Epoch 26/100
7000/7000 [==============================] - 0s 63us/step - loss: 0.5726 - accuracy: 0.7060 - val_loss:
0.5664 - val_accuracy: 0.7070
Epoch 27/100
7000/7000 [==============================] - 0s 61us/step - loss: 0.5697 - accuracy: 0.7110 - val_loss:
0.5663 - val_accuracy: 0.7087
Epoch 28/100
```

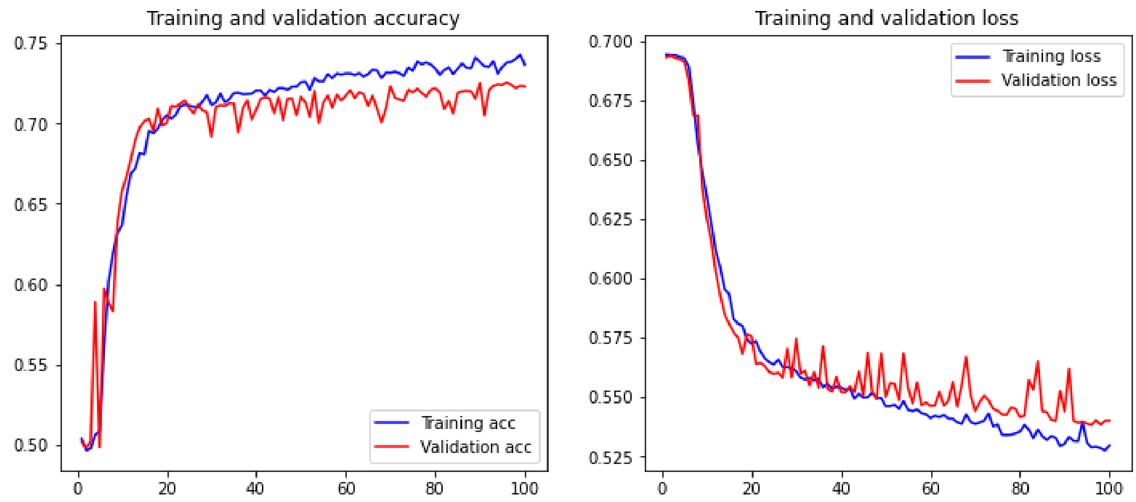
```
Epoch 28/100
7000/7000 [==============================] - 0s 60us/step - loss: 0.5688 - accuracy: 0.7086 - val_loss:
0.5626 - val_accuracy: 0.7120
Epoch 29/100
7000/7000 [==============================] - 0s 69us/step - loss: 0.5670 - accuracy: 0.7073 - val_loss:
0.5613 - val_accuracy: 0.7133
Epoch 30/100
7000/7000 [==============================] - 0s 61us/step - loss: 0.5662 - accuracy: 0.7107 - val_loss:
0.5598 - val_accuracy: 0.7120
Epoch 31/100
7000/7000 [==============================] - 0s 65us/step - loss: 0.5642 - accuracy: 0.7094 - val_loss:
0.5596 - val_accuracy: 0.7130
Epoch 32/100
7000/7000 [==============================] - 0s 60us/step - loss: 0.5643 - accuracy: 0.7111 - val_loss:
0.5617 - val_accuracy: 0.7060
Epoch 33/100
7000/7000 [==============================] - 0s 60us/step - loss: 0.5626 - accuracy: 0.7134 - val_loss:
0.5591 - val_accuracy: 0.7100
Epoch 34/100
7000/7000 [==============================] - 0s 61us/step - loss: 0.5613 - accuracy: 0.7164 - val_loss:
0.5574 - val_accuracy: 0.7127
Epoch 35/100
7000/7000 [==============================] - 0s 62us/step - loss: 0.5598 - accuracy: 0.7156 - val_loss:
0.5562 - val_accuracy: 0.7143
Epoch 36/100
7000/7000 [==============================] - 0s 63us/step - loss: 0.5614 - accuracy: 0.7177 - val_loss:
0.5592 - val_accuracy: 0.7100
Epoch 37/100
7000/7000 [==============================] - 0s 60us/step - loss: 0.5609 - accuracy: 0.7124 - val_loss:
0.5550 - val_accuracy: 0.7107
Epoch 38/100
7000/7000 [==============================] - 0s 61us/step - loss: 0.5600 - accuracy: 0.7199 - val_loss:
0.5588 - val_accuracy: 0.7093
Epoch 39/100
7000/7000 [==============================] - 0s 63us/step - loss: 0.5581 - accuracy: 0.7186 - val_loss:
0.5547 - val_accuracy: 0.7117
Epoch 40/100
7000/7000 [==============================] - 0s 65us/step - loss: 0.5571 - accuracy: 0.7186 - val_loss:
0.5557 - val_accuracy: 0.7100
Epoch 41/100
7000/7000 [==============================] - 0s 67us/step - loss: 0.5581 - accuracy: 0.7139 - val_loss:
0.5623 - val_accuracy: 0.7060
Epoch 42/100
7000/7000 [==============================] - 0s 63us/step - loss: 0.5593 - accuracy: 0.7173 - val_loss:
0.5563 - val_accuracy: 0.7103
Epoch 43/100
7000/7000 [==============================] - 0s 63us/step - loss: 0.5542 - accuracy: 0.7201 - val_loss:
0.5536 - val_accuracy: 0.7130
Epoch 44/100
7000/7000 [==============================] - 0s 64us/step - loss: 0.5571 - accuracy: 0.7160 - val_loss:
0.5525 - val_accuracy: 0.7150
Epoch 45/100
7000/7000 [==============================] - 0s 66us/step - loss: 0.5546 - accuracy: 0.7194 - val_loss:
0.5523 - val_accuracy: 0.7117
Epoch 46/100
7000/7000 [==============================] - 0s 66us/step - loss: 0.5524 - accuracy: 0.7189 - val_loss:
0.5509 - val_accuracy: 0.7103
Epoch 47/100
7000/7000 [==============================] - 0s 67us/step - loss: 0.5529 - accuracy: 0.7229 - val_loss:
0.5525 - val_accuracy: 0.7130
Epoch 48/100
7000/7000 [==============================] - 0s 70us/step - loss: 0.5501 - accuracy: 0.7206 - val_loss:
0.5625 - val_accuracy: 0.7073
Epoch 49/100
7000/7000 [==============================] - 0s 64us/step - loss: 0.5523 - accuracy: 0.7217 - val_loss:
0.5532 - val_accuracy: 0.7140
Epoch 50/100
7000/7000 [==============================] - 0s 64us/step - loss: 0.5490 - accuracy: 0.7266 - val_loss:
0.5481 - val_accuracy: 0.7143
Epoch 51/100
7000/7000 [==============================] - 1s 73us/step - loss: 0.5497 - accuracy: 0.7260 - val_loss:
0.5592 - val_accuracy: 0.7050
Epoch 52/100
7000/7000 [==============================] - 0s 69us/step - loss: 0.5519 - accuracy: 0.7236 - val_loss:
0.5474 - val_accuracy: 0.7150
Epoch 53/100
7000/7000 [==============================] - 0s 65us/step - loss: 0.5489 - accuracy: 0.7213 - val_loss:
0.5529 - val_accuracy: 0.7103
```

```
0.5529 - val_accuracy: 0.7103
Epoch 54/100
7000/7000 [==============================] - 0s 65us/step - loss: 0.5470 - accuracy: 0.7293 - val_loss:
0.5472 - val_accuracy: 0.7137
Epoch 55/100
7000/7000 [==============================] - 0s 62us/step - loss: 0.5476 - accuracy: 0.7290 - val_loss:
0.5460 - val_accuracy: 0.7147
Epoch 56/100
7000/7000 [==============================] - 0s 63us/step - loss: 0.5481 - accuracy: 0.7219 - val_loss:
0.5456 - val_accuracy: 0.7173
Epoch 57/100
7000/7000 [==============================] - 0s 61us/step - loss: 0.5467 - accuracy: 0.7270 - val_loss:
0.5461 - val_accuracy: 0.7140
Epoch 58/100
7000/7000 [==============================] - 0s 61us/step - loss: 0.5464 - accuracy: 0.7271 - val_loss:
0.5480 - val_accuracy: 0.7153
Epoch 59/100
7000/7000 [==============================] - 0s 63us/step - loss: 0.5433 - accuracy: 0.7297 - val_loss:
0.5447 - val_accuracy: 0.7150
Epoch 60/100
7000/7000 [==============================] - 0s 63us/step - loss: 0.5433 - accuracy: 0.7271 - val_loss:
0.5461 - val_accuracy: 0.7150
Epoch 61/100
7000/7000 [==============================] - 0s 63us/step - loss: 0.5435 - accuracy: 0.7307 - val_loss:
0.5549 - val_accuracy: 0.7063
Epoch 62/100
7000/7000 [==============================] - 0s 66us/step - loss: 0.5425 - accuracy: 0.7277 - val_loss:
0.5465 - val_accuracy: 0.7147
Epoch 63/100
7000/7000 [==============================] - 0s 69us/step - loss: 0.5419 - accuracy: 0.7329 - val_loss:
0.5448 - val_accuracy: 0.7187
Epoch 64/100
7000/7000 [==============================] - 0s 67us/step - loss: 0.5431 - accuracy: 0.7284 - val_loss:
0.5451 - val_accuracy: 0.7180
Epoch 65/100
7000/7000 [==============================] - 0s 62us/step - loss: 0.5402 - accuracy: 0.7287 - val_loss:
0.5432 - val_accuracy: 0.7177
Epoch 66/100
7000/7000 [==============================] - 0s 62us/step - loss: 0.5418 - accuracy: 0.7311 - val_loss:
0.5459 - val_accuracy: 0.7150
Epoch 67/100
7000/7000 [==============================] - 0s 62us/step - loss: 0.5414 - accuracy: 0.7297 - val_loss:
0.5436 - val_accuracy: 0.7177
Epoch 68/100
7000/7000 [==============================] - 0s 64us/step - loss: 0.5394 - accuracy: 0.7320 - val_loss:
0.5434 - val_accuracy: 0.7190
Epoch 69/100
7000/7000 [==============================] - 0s 63us/step - loss: 0.5406 - accuracy: 0.7296 - val_loss:
0.5436 - val_accuracy: 0.7163
Epoch 70/100
7000/7000 [==============================] - 0s 64us/step - loss: 0.5397 - accuracy: 0.7356 - val_loss:
0.5411 - val_accuracy: 0.7210
Epoch 71/100
7000/7000 [==============================] - 1s 75us/step - loss: 0.5389 - accuracy: 0.7289 - val_loss:
0.5438 - val_accuracy: 0.7203
Epoch 72/100
7000/7000 [==============================] - 0s 62us/step - loss: 0.5410 - accuracy: 0.7296 - val_loss:
0.5491 - val_accuracy: 0.7103
Epoch 73/100
7000/7000 [==============================] - 0s 67us/step - loss: 0.5354 - accuracy: 0.7390 - val_loss:
0.5416 - val_accuracy: 0.7173
Epoch 74/100
7000/7000 [==============================] - 0s 64us/step - loss: 0.5360 - accuracy: 0.7346 - val_loss:
0.5418 - val_accuracy: 0.7170
Epoch 75/100
7000/7000 [==============================] - 0s 63us/step - loss: 0.5347 - accuracy: 0.7366 - val_loss:
0.5421 - val_accuracy: 0.7210
Epoch 76/100
7000/7000 [==============================] - 0s 63us/step - loss: 0.5349 - accuracy: 0.7356 - val_loss:
0.5420 - val_accuracy: 0.7213
Epoch 77/100
7000/7000 [==============================] - 0s 62us/step - loss: 0.5361 - accuracy: 0.7327 - val_loss:
0.5399 - val_accuracy: 0.7227
Epoch 78/100
7000/7000 [==============================] - 0s 61us/step - loss: 0.5362 - accuracy: 0.7331 - val_loss:
0.5402 - val_accuracy: 0.7173
Epoch 79/100
7000/7000 [==============================] - 0s 62us/step - loss: 0.5332 - accuracy: 0.7361 - val_loss:
```

```
7000/7000 [==============================] - 0s 62us/step - loss: 0.5332 - accuracy: 0.7361 - val_loss: 
0.5384 - val_accuracy: 0.7210
Epoch 80/100
7000/7000 [==============================] - 0s 64us/step - loss: 0.5322 - accuracy: 0.7401 - val_loss: 
0.5654 - val_accuracy: 0.7040
Epoch 81/100
7000/7000 [==============================] - 0s 64us/step - loss: 0.5320 - accuracy: 0.7411 - val_loss: 
0.5415 - val_accuracy: 0.7147
Epoch 82/100
7000/7000 [==============================] - 0s 62us/step - loss: 0.5341 - accuracy: 0.7333 - val_loss: 
0.5556 - val_accuracy: 0.7143
Epoch 83/100
7000/7000 [==============================] - 0s 63us/step - loss: 0.5339 - accuracy: 0.7314 - val_loss: 
0.5467 - val_accuracy: 0.7180
Epoch 84/100
7000/7000 [==============================] - 0s 63us/step - loss: 0.5304 - accuracy: 0.7401 - val_loss: 
0.5384 - val_accuracy: 0.7240
Epoch 85/100
7000/7000 [==============================] - 0s 63us/step - loss: 0.5292 - accuracy: 0.7427 - val_loss: 
0.5436 - val_accuracy: 0.7180
Epoch 86/100
7000/7000 [==============================] - 0s 63us/step - loss: 0.5296 - accuracy: 0.7404 - val_loss: 
0.5375 - val_accuracy: 0.7267
Epoch 87/100
7000/7000 [==============================] - 0s 62us/step - loss: 0.5315 - accuracy: 0.7363 - val_loss: 
0.5385 - val_accuracy: 0.7197
Epoch 88/100
7000/7000 [==============================] - 0s 63us/step - loss: 0.5290 - accuracy: 0.7386 - val_loss: 
0.5382 - val_accuracy: 0.7190
Epoch 89/100
7000/7000 [==============================] - 0s 62us/step - loss: 0.5290 - accuracy: 0.7414 - val_loss: 
0.5384 - val_accuracy: 0.7223
Epoch 90/100
7000/7000 [==============================] - 0s 64us/step - loss: 0.5286 - accuracy: 0.7404 - val_loss: 
0.5362 - val_accuracy: 0.7263
Epoch 91/100
7000/7000 [==============================] - 0s 63us/step - loss: 0.5281 - accuracy: 0.7400 - val_loss: 
0.5341 - val_accuracy: 0.7227
Epoch 92/100
7000/7000 [==============================] - 0s 64us/step - loss: 0.5277 - accuracy: 0.7376 - val_loss: 
0.5351 - val_accuracy: 0.7197
Epoch 93/100
7000/7000 [==============================] - 0s 64us/step - loss: 0.5248 - accuracy: 0.7430 - val_loss: 
0.5362 - val_accuracy: 0.7180
Epoch 94/100
7000/7000 [==============================] - 0s 63us/step - loss: 0.5271 - accuracy: 0.7389 - val_loss: 
0.5477 - val_accuracy: 0.7170
Epoch 95/100
7000/7000 [==============================] - 0s 62us/step - loss: 0.5255 - accuracy: 0.7417 - val_loss: 
0.5340 - val_accuracy: 0.7310
Epoch 96/100
7000/7000 [==============================] - 0s 67us/step - loss: 0.5242 - accuracy: 0.7421 - val_loss: 
0.5333 - val_accuracy: 0.7237
Epoch 97/100
7000/7000 [==============================] - 0s 64us/step - loss: 0.5225 - accuracy: 0.7411 - val_loss: 
0.5377 - val_accuracy: 0.7203
Epoch 98/100
7000/7000 [==============================] - 0s 63us/step - loss: 0.5245 - accuracy: 0.7399 - val_loss: 
0.5372 - val_accuracy: 0.7207
Epoch 99/100
7000/7000 [==============================] - 0s 63us/step - loss: 0.5232 - accuracy: 0.7394 - val_loss: 
0.5342 - val_accuracy: 0.7270
Epoch 100/100
7000/7000 [==============================] - 0s 63us/step - loss: 0.5227 - accuracy: 0.7433 - val_loss: 
0.5357 - val_accuracy: 0.7217
```

In [19]:

```
plot_history(CNN_history)
```



In [32]:

```
y_train_pred = CNN_Model.predict(X_train.reshape(X_train.shape[0], X_train.shape[1], 1))

y_train_pred = np.where(y_train_pred < 0.5, 0, 1)

print(classification_report(y_train_pred, y_train))

yhat_xg_bestmodel = CNN_Model.predict(X_test.reshape(X_test.shape[0], X_test.shape[1], 1))

yhat_xg_bestmodel = np.where(yhat_xg_bestmodel < 0.5, 0, 1)

classi_reprt_xg = classification_report(yhat_xg_bestmodel, y_test)

print(classi_reprt_xg)
```

```
              precision    recall  f1-score   support

           0       0.81      0.71      0.76      3957
           1       0.68      0.78      0.72      3043

    accuracy                           0.74      7000
   macro avg       0.74      0.75      0.74      7000
weighted avg       0.75      0.74      0.74      7000

              precision    recall  f1-score   support

           0       0.79      0.69      0.74      1711
           1       0.65      0.76      0.70      1289

    accuracy                           0.72      3000
   macro avg       0.72      0.73      0.72      3000
weighted avg       0.73      0.72      0.72      3000
```

In [ ]: