In [1]:
```
import numpy as np
import pandas as pd
train = pd.read_csv('UNSW_NB15_freq_enc_training_set.csv')
test = pd.read_csv('UNSW_NB15_freq_enc_testing_set.csv')
```

In [2]:
```
x1 = train.iloc[:,1:43]
y1 = train['label']
print(x1)
print(y1)
```

```
             dur  spkts  dpkts  sbytes  dbytes           rate  sttl  dttl  \
0       0.121478      6      4     258     172      74.087490   252   254
1       0.649902     14     38     734   42014      78.473372    62   252
2       1.623129      8     16     364   13186      14.170161    62   252
3       1.681642     12     12     628     770      13.677108    62   252
4       0.449454     10      6     534     268      33.373826   254   252
...          ...    ...    ...     ...     ...            ...   ...   ...
175336  0.000009      2      0     114       0  111111.107200   254     0
175337  0.505762     10      8     620     354      33.612649   254   252
175338  0.000009      2      0     114       0  111111.107200   254     0
175339  0.000009      2      0     114       0  111111.107200   254     0
175340  0.000009      2      0     114       0  111111.107200   254     0

              sload          dload  ...  ct_dst_src_ltm  is_ftp_login  \
0       1.415894e+04    8495.365234  ...               1             0
1       8.395112e+03  503571.312500  ...               2             0
2       1.572272e+03   60929.230470  ...               3             0
3       2.740179e+03    3358.622070  ...               3             1
4       8.561499e+03    3987.059814  ...              40             0
...              ...            ...  ...             ...           ...
175336  5.066666e+07       0.000000  ...              24             0
175337  8.826286e+03    4903.492188  ...               2             0
175338  5.066666e+07       0.000000  ...              13             0
175339  5.066666e+07       0.000000  ...              30             0
175340  5.066666e+07       0.000000  ...              30             0

        ct_ftp_cmd  ct_flw_http_mthd  ct_src_ltm  ct_srv_dst  is_sm_ips_ports  \
0                0                 0           1           1                0
1                0                 0           1           6                0
2                0                 0           2           6                0
3                1                 0           2           1                0
4                0                 0           2          39                0
...            ...               ...         ...         ...              ...
175336           0                 0          24          24                0
175337           0                 0           1           1                0
175338           0                 0           3          12                0
175339           0                 0          30          30                0
175340           0                 0          30          30                0

        proto_freq_encode  service_freq_encode  state_freq_encode
0                0.477508             0.548451           0.454700
1                0.477508             0.548451           0.454700
2                0.477508             0.548451           0.454700
3                0.477508             0.019327           0.454700
4                0.477508             0.548451           0.454700
...                   ...                  ...                ...
175336           0.359762             0.266466           0.451883
175337           0.477508             0.548451           0.454700
175338           0.359762             0.266466           0.451883
175339           0.359762             0.266466           0.451883
175340           0.359762             0.266466           0.451883

[175341 rows x 42 columns]
0         0
1         0
2         0
3         0
4         0
         ..
175336    1
175337    1
175338    1
175339    1
175340    1
Name: label, Length: 175341, dtype: int64
```

In [3]:
```
x2 = test.iloc[:,1:43]
y2 = test['label']
print(x2)
print(y2)
```

```
            dur  spkts  dpkts  sbytes  dbytes           rate  sttl  dttl  \
0      0.000011      2      0     496       0   90909.090200   254     0
1      0.000008      2      0    1762       0  125000.000300   254     0
2      0.000005      2      0    1068       0  200000.005100   254     0
3      0.000006      2      0     900       0  166666.660800   254     0
4      0.000010      2      0    2126       0  100000.002500   254     0
...         ...    ...    ...     ...     ...            ...   ...   ...
82327  0.000005      2      0     104       0  200000.005100   254     0
82328  1.106101     20      8   18062     354      24.410067   254   252
82329  0.000000      1      0      46       0       0.000000     0     0
82330  0.000000      1      0      46       0       0.000000     0     0
82331  0.000009      2      0     104       0  111111.107200   254     0

              sload        dload  ...  ct_dst_src_ltm  is_ftp_login  \
0      1.803636e+08     0.000000  ...               2             0
1      8.810000e+08     0.000000  ...               2             0
2      8.544000e+08     0.000000  ...               3             0
3      6.000000e+08     0.000000  ...               3             0
4      8.504000e+08     0.000000  ...               3             0
...             ...          ...  ...             ...           ...
82327  8.320000e+07     0.000000  ...               2             0
82328  1.241044e+05  2242.109863  ...               1             0
82329  0.000000e+00     0.000000  ...               1             0
82330  0.000000e+00     0.000000  ...               1             0
82331  4.622222e+07     0.000000  ...               1             0

       ct_ftp_cmd  ct_flw_http_mthd  ct_src_ltm  ct_srv_dst  is_sm_ips_ports  \
0               0                 0           1           2                0
1               0                 0           1           2                0
2               0                 0           1           3                0
3               0                 0           2           3                0
4               0                 0           2           3                0
...           ...               ...         ...         ...              ...
82327           0                 0           2           1                0
82328           0                 0           3           2                0
82329           0                 0           1           1                1
82330           0                 0           1           1                1
82331           0                 0           1           1                0

       proto_freq_encode  service_freq_encode  state_freq_encode
0               0.359762             0.548451           0.451883
1               0.359762             0.548451           0.451883
2               0.359762             0.548451           0.451883
3               0.359762             0.548451           0.451883
4               0.359762             0.548451           0.451883
...                  ...                  ...                ...
82327           0.359762             0.548451           0.451883
82328           0.477508             0.548451           0.454700
82329           0.014926             0.548451           0.451883
82330           0.014926             0.548451           0.451883
82331           0.359762             0.548451           0.451883

[82332 rows x 42 columns]
0        0
1        0
2        0
3        0
4        0
        ..
82327    0
82328    0
82329    0
82330    0
82331    0
Name: label, Length: 82332, dtype: int64
```

In [4]:
```
from sklearn.preprocessing import MinMaxScaler
model = MinMaxScaler()
model.fit(x1)
x1 = model.transform(x1)
x2 = model.transform(x2)
```

In [5]:
```
from tensorflow import keras
model_DNN=keras.models.Sequential()
model_DNN.add(keras.layers.Dense(units=43, activation = "relu",input_shape = x1.shape[1:]))
model_DNN.add(keras.layers.Dense(units=86,activation = "relu"))
model_DNN.add(keras.layers.Dropout(0.25))
model_DNN.add(keras.layers.Dense(units=172,activation = "relu" ))
model_DNN.add(keras.layers.Dropout(0.25))
model_DNN.add(keras.layers.Dense(units=172,activation = "relu" ))
model_DNN.add(keras.layers.Dropout(0.25))
model_DNN.add(keras.layers.Dense(units=344,activation = "relu" ))
model_DNN.add(keras.layers.Dense(units=2, activation = "sigmoid"))
model_DNN.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 43)                1849

 dense_1 (Dense)             (None, 86)                3784

 dropout (Dropout)           (None, 86)                0

 dense_2 (Dense)             (None, 172)               14964

 dropout_1 (Dropout)         (None, 172)               0

 dense_3 (Dense)             (None, 172)               29756

 dropout_2 (Dropout)         (None, 172)               0

 dense_4 (Dense)             (None, 344)               59512

 dense_5 (Dense)             (None, 2)                 690

=================================================================
Total params: 110,555
Trainable params: 110,555
Non-trainable params: 0
_________________________________________________________________
```

In [6]:
```
model_DNN.compile(loss="sparse_categorical_crossentropy",optimizer="adam",metrics="accuracy")
model_DNN.fit(x1,y1,epochs=50,batch_size=16)
testloss,testaccuracy=model_DNN.evaluate(x2,y2)
print("Test loss = ",testloss)
print("Test accuracy = ",testaccuracy)
```

```
Epoch 1/50
10959/10959 [==============================] - 63s 4ms/step - loss: 0.1451 - accuracy: 0.9341
Epoch 2/50
10959/10959 [==============================] - 44s 4ms/step - loss: 0.1326 - accuracy: 0.9367
Epoch 3/50
10959/10959 [==============================] - 41s 4ms/step - loss: 0.1290 - accuracy: 0.9373
Epoch 4/50
10959/10959 [==============================] - 41s 4ms/step - loss: 0.1284 - accuracy: 0.9374
Epoch 5/50
10959/10959 [==============================] - 38s 3ms/step - loss: 0.1275 - accuracy: 0.9381
Epoch 6/50
10959/10959 [==============================] - 38s 3ms/step - loss: 0.1273 - accuracy: 0.9379
Epoch 7/50
10959/10959 [==============================] - 38s 3ms/step - loss: 0.1290 - accuracy: 0.9387
Epoch 8/50
10959/10959 [==============================] - 40s 4ms/step - loss: 0.1268 - accuracy: 0.9381
Epoch 9/50
10959/10959 [==============================] - 40s 4ms/step - loss: 0.1274 - accuracy: 0.9378
Epoch 10/50
10959/10959 [==============================] - 38s 3ms/step - loss: 0.1261 - accuracy: 0.9385
Epoch 11/50
10959/10959 [==============================] - 37s 3ms/step - loss: 0.1251 - accuracy: 0.9390
Epoch 12/50
10959/10959 [==============================] - 37s 3ms/step - loss: 0.1251 - accuracy: 0.9391
Epoch 13/50
10959/10959 [==============================] - 37s 3ms/step - loss: 0.1275 - accuracy: 0.9391
Epoch 14/50
10959/10959 [==============================] - 37s 3ms/step - loss: 0.1250 - accuracy: 0.9389
Epoch 15/50
10959/10959 [==============================] - 37s 3ms/step - loss: 0.1374 - accuracy: 0.9389
Epoch 16/50
10959/10959 [==============================] - 37s 3ms/step - loss: 0.1229 - accuracy: 0.9395
Epoch 17/50
10959/10959 [==============================] - 37s 3ms/step - loss: 0.1255 - accuracy: 0.9392
Epoch 18/50
10959/10959 [==============================] - 37s 3ms/step - loss: 0.1257 - accuracy: 0.9398
Epoch 19/50
10959/10959 [==============================] - 37s 3ms/step - loss: 0.1248 - accuracy: 0.9391
Epoch 20/50
10959/10959 [==============================] - 38s 3ms/step - loss: 0.1247 - accuracy: 0.9394
Epoch 21/50
10959/10959 [==============================] - 37s 3ms/step - loss: 0.1237 - accuracy: 0.9395
Epoch 22/50
10959/10959 [==============================] - 37s 3ms/step - loss: 0.1230 - accuracy: 0.9391
Epoch 23/50
10959/10959 [==============================] - 39s 4ms/step - loss: 0.1238 - accuracy: 0.9388
Epoch 24/50
10959/10959 [==============================] - 38s 4ms/step - loss: 0.1250 - accuracy: 0.9399
Epoch 25/50
10959/10959 [==============================] - 34s 3ms/step - loss: 0.1281 - accuracy: 0.9400
Epoch 26/50
10959/10959 [==============================] - 34s 3ms/step - loss: 0.1230 - accuracy: 0.9402
Epoch 27/50
10959/10959 [==============================] - 34s 3ms/step - loss: 0.1227 - accuracy: 0.9395
Epoch 28/50
10959/10959 [==============================] - 34s 3ms/step - loss: 0.1223 - accuracy: 0.9400
Epoch 29/50
10959/10959 [==============================] - 34s 3ms/step - loss: 0.1232 - accuracy: 0.9394
Epoch 30/50
10959/10959 [==============================] - 34s 3ms/step - loss: 0.1230 - accuracy: 0.9400
Epoch 31/50
10959/10959 [==============================] - 34s 3ms/step - loss: 0.1213 - accuracy: 0.9400
Epoch 32/50
10959/10959 [==============================] - 34s 3ms/step - loss: 0.1277 - accuracy: 0.9404
Epoch 33/50
10959/10959 [==============================] - 34s 3ms/step - loss: 0.1208 - accuracy: 0.9399
Epoch 34/50
10959/10959 [==============================] - 34s 3ms/step - loss: 0.1224 - accuracy: 0.9397
Epoch 35/50
10959/10959 [==============================] - 34s 3ms/step - loss: 0.1260 - accuracy: 0.9409
Epoch 36/50
10959/10959 [==============================] - 34s 3ms/step - loss: 0.1234 - accuracy: 0.9404
Epoch 37/50
10959/10959 [==============================] - 34s 3ms/step - loss: 0.1264 - accuracy: 0.9408
Epoch 38/50
10959/10959 [==============================] - 34s 3ms/step - loss: 0.1277 - accuracy: 0.9410
Epoch 39/50
10959/10959 [==============================] - ETA: 0s - loss: 0.1241 - accuracy: 0.93 - 34s 3ms/step - loss:
0.1241 - accuracy: 0.9397
Epoch 40/50
10959/10959 [==============================] - 34s 3ms/step - loss: 0.1214 - accuracy: 0.9403
Epoch 41/50
10959/10959 [==============================] - 34s 3ms/step - loss: 0.1236 - accuracy: 0.9404
Epoch 42/50
10959/10959 [==============================] - 34s 3ms/step - loss: 0.1231 - accuracy: 0.9399
```

```
Epoch 43/50
10959/10959 [==============================] - 34s 3ms/step - loss: 0.1213 - accuracy: 0.9405
Epoch 44/50
10959/10959 [==============================] - 34s 3ms/step - loss: 0.1225 - accuracy: 0.9402
Epoch 45/50
10959/10959 [==============================] - 34s 3ms/step - loss: 0.1227 - accuracy: 0.9397
Epoch 46/50
10959/10959 [==============================] - 34s 3ms/step - loss: 0.1233 - accuracy: 0.9412
Epoch 47/50
10959/10959 [==============================] - 37s 3ms/step - loss: 0.1205 - accuracy: 0.9408
Epoch 48/50
10959/10959 [==============================] - 35s 3ms/step - loss: 0.1210 - accuracy: 0.9407
Epoch 49/50
10959/10959 [==============================] - 34s 3ms/step - loss: 0.1216 - accuracy: 0.9411
Epoch 50/50
10959/10959 [==============================] - 34s 3ms/step - loss: 0.1277 - accuracy: 0.9403
2573/2573 [==============================] - 5s 2ms/step - loss: 0.4913 - accuracy: 0.8126
Test loss =  0.4912950098514557
Test accuracy =  0.8125516176223755
```

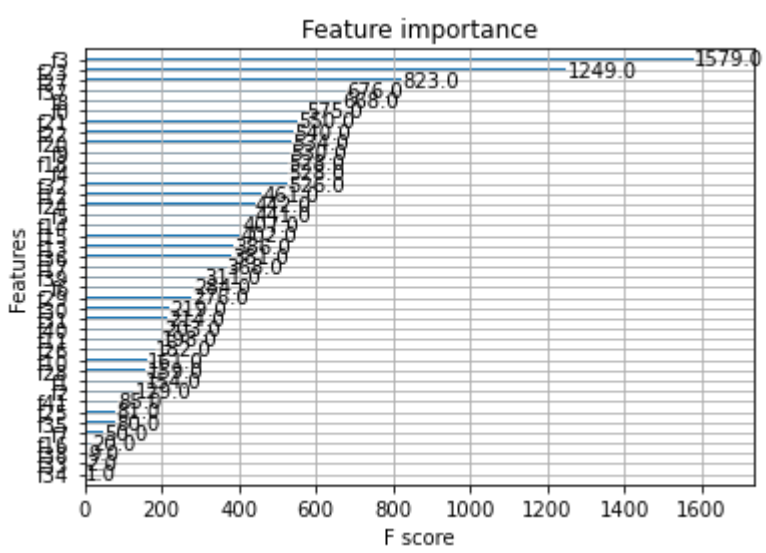
In [7]:
```
from xgboost import XGBClassifier
import xgboost as xgb
params = {
            'objective':'binary:logistic',
            'max_depth': 4, 'min_child_weight': 12, 'gamma': 0.3, 'subsample': 0.6,
            'colsample_bytree': 0.6, 'scale_pos_weight': 1,
            'alpha': 0.05,
            'learning_rate': 0.03,
            'n_estimators':1484,'seed': 27
        }
xgb_clf = XGBClassifier(**params)
xgb_clf.fit(x1, y1)
```

```
C:\Users\admin\anaconda3\lib\site-packages\xgboost\sklearn.py:1224: UserWarning: The use of label encoder in X
GBClassifier 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) a
s integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)

[12:51:30] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115: Starting
in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'er
ror' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
```

Out[7]:
```
XGBClassifier(alpha=0.05, base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.6,
              enable_categorical=False, gamma=0.3, gpu_id=-1,
              importance_type=None, interaction_constraints='',
              learning_rate=0.03, max_delta_step=0, max_depth=4,
              min_child_weight=12, missing=nan, monotone_constraints='()',
              n_estimators=1484, n_jobs=4, num_parallel_tree=1,
              predictor='auto', random_state=27, reg_alpha=0.0500000007,
              reg_lambda=1, scale_pos_weight=1, seed=27, subsample=0.6,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

In [8]:
```
import matplotlib.pyplot as plt
xgb.plot_importance(xgb_clf)
plt.rcParams['figure.figsize'] = [15,15]
plt.show()
```



In [9]:
```
pd.Series(xgb_clf.get_booster().get_fscore()).sort_values(ascending=False)
```

Out[9]:
```
f3     1579.0
f23    1249.0
f27     823.0
f37     676.0
f8      668.0
f0      575.0
f21     550.0
f22     540.0
f20     534.0
f9      530.0
f18     528.0
f4      528.0
f32     526.0
f12     461.0
f24     442.0
f5      441.0
f14     407.0
f15     402.0
f13     386.0
f36     381.0
f17     368.0
f39     311.0
f6      284.0
f29     276.0
f30     219.0
f31     214.0
f40     203.0
f11     198.0
f26     182.0
f10     161.0
f28     159.0
f1      154.0
f2      129.0
f41      85.0
f25      81.0
f35      80.0
f7       50.0
f16      20.0
f38       9.0
f33       2.0
f34       1.0
dtype: float64
```

In [10]:
```
from numpy import sort
from sklearn.feature_selection import SelectFromModel
thresholds = sort(xgb_clf.feature_importances_)
print(thresholds)
```

```
[0.         0.00054432 0.00075167 0.00088144 0.00100133 0.00104625
 0.00116126 0.00125949 0.00148376 0.00150489 0.00155229 0.00164402
 0.00196953 0.00204442 0.00211289 0.00354206 0.00424938 0.00485486
 0.00526185 0.00535822 0.00559029 0.00635514 0.00639051 0.00651507
 0.00656803 0.00674823 0.00699365 0.00813449 0.00829311 0.00854332
 0.00965884 0.00986754 0.01239873 0.01510002 0.01607109 0.01839451
 0.03082572 0.03776417 0.05745872 0.13671382 0.13973343 0.40365767]
```

In [12]:
```
from sklearn import metrics
n_min = 43
acc_max = 0
thresholds = sort(xgb_clf.feature_importances_)
obj_thresh = thresholds[0]
for thresh in thresholds:
    selection = SelectFromModel(xgb_clf, threshold=thresh, prefit=True)
    select_X_train = selection.transform(x1)
    selection_model = XGBClassifier(**params)
    selection_model.fit(select_X_train, y1)
    select_X_test = selection.transform(x2)
    predictions = selection_model.predict(select_X_test)
    accuracy = metrics.accuracy_score(y2, predictions)
    print("Thresh=%.3f, n=%d, Accuracy: %.2f%%" % (obj_thresh, select_X_train.shape[1], accuracy*100.0))
    if(select_X_train.shape[1] < n_min) and (accuracy > acc_max):
        n_min = select_X_train.shape[1]
        acc_max = accuracy
        obj_thresh = thresh
```

```
m 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.



C:\Users\admin\anaconda3\lib\site-packages\xgboost\sklearn.py:1224: UserWarning: The use of label encoder in
XGBClassifier is deprecated and will be removed in a future release. To remove this warning, do the followin
g: 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)

Thresh=0.005, n=2, Accuracy: 76.63%
[17:17:17] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115: Starti
ng in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed fro
m 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.

C:\Users\admin\anaconda3\lib\site-packages\xgboost\sklearn.py:1224: UserWarning: The use of label encoder in
XGBClassifier is deprecated and will be removed in a future release. To remove this warning, do the followin
g: 1) Pass option use_label_encoder=False when constructing XGBClassifier object; and 2) Encode your labels
```

In [13]:
```
selection = SelectFromModel(xgb_clf, threshold=obj_thresh, prefit=True)
select_X_train = selection.transform(x1)
selection_model = XGBClassifier(**params)
selection_model.fit(select_X_train, y1)
select_X_test = selection.transform(x2)
predictions = selection_model.predict(select_X_test)
accuracy = metrics.accuracy_score(y2, predictions)
print("Thresh=%.3f, n=%d, Accuracy: %.2f%%" % (obj_thresh, select_X_train.shape[1], accuracy*100.0))
```

```
C:\Users\admin\anaconda3\lib\site-packages\xgboost\sklearn.py:1224: UserWarning: The use of label encoder in X
GBClassifier 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) a
s integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)

[17:19:12] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115: Starting
in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'er
ror' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
Thresh=0.005, n=23, Accuracy: 88.05%
```

In [16]:
```
from sklearn.preprocessing import MinMaxScaler
model_new = MinMaxScaler()
model_new.fit(select_X_train)
select_X_train = model_new.transform(select_X_train)
select_X_test = model_new.transform(select_X_test)
```

In [20]:
```
from tensorflow import keras
model_DNN=keras.models.Sequential()
model_DNN.add(keras.layers.Dense(units=23, activation = "relu",input_shape = select_X_train.shape[1:]))
model_DNN.add(keras.layers.Dense(units=46,activation = "relu"))
model_DNN.add(keras.layers.Dropout(0.25))
model_DNN.add(keras.layers.Dense(units=92,activation = "relu" ))
model_DNN.add(keras.layers.Dropout(0.25))
model_DNN.add(keras.layers.Dense(units=184,activation = "relu" ))
model_DNN.add(keras.layers.Dropout(0.25))
model_DNN.add(keras.layers.Dense(units=368,activation = "relu" ))
model_DNN.add(keras.layers.Dense(units=2, activation = "sigmoid"))
model_DNN.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_12 (Dense)            (None, 23)                552

 dense_13 (Dense)            (None, 46)                1104

 dropout_6 (Dropout)         (None, 46)                0

 dense_14 (Dense)            (None, 92)                4324

 dropout_7 (Dropout)         (None, 92)                0

 dense_15 (Dense)            (None, 184)               17112

 dropout_8 (Dropout)         (None, 184)               0

 dense_16 (Dense)            (None, 368)               68080

 dense_17 (Dense)            (None, 2)                 738

=================================================================
Total params: 91,910
Trainable params: 91,910
Non-trainable params: 0
_________________________________________________________________
```

In [21]:
```
model_DNN.compile(loss="sparse_categorical_crossentropy",optimizer="adam",metrics="accuracy")
model_DNN.fit(select_X_train, y1,epochs=50,batch_size=16)
testloss,testaccuracy=model_DNN.evaluate(select_X_test,y2)
print("Test loss = ",testloss)
print("Test accuracy = ",testaccuracy)
```

```
Epoch 1/50
10959/10959 [==============================] - 35s 3ms/step - loss: 0.1493 - accuracy: 0.9334
Epoch 2/50
10959/10959 [==============================] - 34s 3ms/step - loss: 0.1327 - accuracy: 0.9353
Epoch 3/50
10959/10959 [==============================] - 43s 4ms/step - loss: 0.1294 - accuracy: 0.9357
Epoch 4/50
10959/10959 [==============================] - ETA: 0s - loss: 0.1286 - accuracy: 0.93 - 53s 5ms/step - loss:
0.1286 - accuracy: 0.9359
Epoch 5/50
10959/10959 [==============================] - 53s 5ms/step - loss: 0.1272 - accuracy: 0.9364
Epoch 6/50
10959/10959 [==============================] - 54s 5ms/step - loss: 0.1270 - accuracy: 0.9367
Epoch 7/50
10959/10959 [==============================] - 55s 5ms/step - loss: 0.1271 - accuracy: 0.9372
Epoch 8/50
10959/10959 [==============================] - 55s 5ms/step - loss: 0.1254 - accuracy: 0.9375
Epoch 9/50
10959/10959 [==============================] - 53s 5ms/step - loss: 0.1266 - accuracy: 0.9372
Epoch 10/50
10959/10959 [==============================] - 48s 4ms/step - loss: 0.1254 - accuracy: 0.9371
Epoch 11/50
10959/10959 [==============================] - 49s 4ms/step - loss: 0.1263 - accuracy: 0.9376
Epoch 12/50
10959/10959 [==============================] - 53s 5ms/step - loss: 0.1255 - accuracy: 0.9376
Epoch 13/50
10959/10959 [==============================] - 48s 4ms/step - loss: 0.1246 - accuracy: 0.9371
Epoch 14/50
10959/10959 [==============================] - 54s 5ms/step - loss: 0.1290 - accuracy: 0.9370
Epoch 15/50
10959/10959 [==============================] - 57s 5ms/step - loss: 0.1305 - accuracy: 0.9368
Epoch 16/50
10959/10959 [==============================] - 57s 5ms/step - loss: 0.1319 - accuracy: 0.9367
Epoch 17/50
10959/10959 [==============================] - 56s 5ms/step - loss: 0.1246 - accuracy: 0.9374
Epoch 18/50
10959/10959 [==============================] - 57s 5ms/step - loss: 0.1266 - accuracy: 0.9370
Epoch 19/50
10959/10959 [==============================] - 57s 5ms/step - loss: 0.1266 - accuracy: 0.9374
Epoch 20/50
10959/10959 [==============================] - 57s 5ms/step - loss: 0.1267 - accuracy: 0.9373
Epoch 21/50
10959/10959 [==============================] - 56s 5ms/step - loss: 0.1260 - accuracy: 0.9372
Epoch 22/50
10959/10959 [==============================] - 48s 4ms/step - loss: 0.1253 - accuracy: 0.9369
Epoch 23/50
10959/10959 [==============================] - 48s 4ms/step - loss: 0.1256 - accuracy: 0.9380
Epoch 24/50
10959/10959 [==============================] - 48s 4ms/step - loss: 0.1246 - accuracy: 0.9375
Epoch 25/50
10959/10959 [==============================] - 48s 4ms/step - loss: 0.1284 - accuracy: 0.9376
Epoch 26/50
10959/10959 [==============================] - 49s 4ms/step - loss: 0.1244 - accuracy: 0.9379
Epoch 27/50
10959/10959 [==============================] - 49s 4ms/step - loss: 0.1245 - accuracy: 0.9378
Epoch 28/50
10959/10959 [==============================] - 48s 4ms/step - loss: 0.1259 - accuracy: 0.9378
Epoch 29/50
10959/10959 [==============================] - 49s 4ms/step - loss: 0.1250 - accuracy: 0.9380
Epoch 30/50
10959/10959 [==============================] - 49s 4ms/step - loss: 0.1225 - accuracy: 0.9378
Epoch 31/50
10959/10959 [==============================] - 49s 4ms/step - loss: 0.1325 - accuracy: 0.9378
Epoch 32/50
10959/10959 [==============================] - 49s 4ms/step - loss: 0.1386 - accuracy: 0.9382
Epoch 33/50
10959/10959 [==============================] - 49s 4ms/step - loss: 0.1262 - accuracy: 0.9379 0s - loss: 0.126
2 - accuracy: 0.93
Epoch 34/50
10959/10959 [==============================] - 49s 4ms/step - loss: 0.1258 - accuracy: 0.9382
Epoch 35/50
10959/10959 [==============================] - 49s 5ms/step - loss: 0.1418 - accuracy: 0.9377
Epoch 36/50
10959/10959 [==============================] - 49s 4ms/step - loss: 0.1250 - accuracy: 0.9376
Epoch 37/50
10959/10959 [==============================] - 50s 5ms/step - loss: 0.1258 - accuracy: 0.9374
Epoch 38/50
10959/10959 [==============================] - 38s 3ms/step - loss: 0.1330 - accuracy: 0.9377
Epoch 39/50
10959/10959 [==============================] - 33s 3ms/step - loss: 0.1277 - accuracy: 0.9377
Epoch 40/50
10959/10959 [==============================] - 32s 3ms/step - loss: 0.1253 - accuracy: 0.9379
Epoch 41/50
10959/10959 [==============================] - 32s 3ms/step - loss: 0.1245 - accuracy: 0.9374
Epoch 42/50
```

```
10959/10959 [==============================] - 33s 3ms/step - loss: 0.1267 - accuracy: 0.9382
Epoch 43/50
10959/10959 [==============================] - 33s 3ms/step - loss: 0.1289 - accuracy: 0.9380
Epoch 44/50
10959/10959 [==============================] - 33s 3ms/step - loss: 0.1260 - accuracy: 0.9376
Epoch 45/50
10959/10959 [==============================] - 32s 3ms/step - loss: 0.1272 - accuracy: 0.9384
Epoch 46/50
10959/10959 [==============================] - 32s 3ms/step - loss: 0.1240 - accuracy: 0.9377
Epoch 47/50
10959/10959 [==============================] - 32s 3ms/step - loss: 0.1350 - accuracy: 0.9384
Epoch 48/50
10959/10959 [==============================] - 33s 3ms/step - loss: 0.1232 - accuracy: 0.9381
Epoch 49/50
10959/10959 [==============================] - 32s 3ms/step - loss: 0.1241 - accuracy: 0.9379
Epoch 50/50
10959/10959 [==============================] - 32s 3ms/step - loss: 0.1304 - accuracy: 0.9377
2573/2573 [==============================] - 5s 2ms/step - loss: 0.3061 - accuracy: 0.8103
Test loss =  0.3060711622238159
Test accuracy =  0.8103289008140564
```

In [22]:

```
y_pred = model_DNN.predict(select_X_test)
y_pred_class=np.argmax(y_pred,axis=1)
print(y_pred_class)
print("Test Loss =",testloss)
print("Test Accuracy =",testaccuracy)
from sklearn import metrics
from sklearn.metrics import f1_score
print('Accuracy = ', metrics.accuracy_score(y_pred_class, y2)*100)
print("Confusion Matrix =", metrics.confusion_matrix(y_pred_class, y2, labels=None,
                                                     sample_weight=None))
print("Recall =", metrics.recall_score(y_pred_class, y2, labels=None,
                                       pos_label=1, average='weighted',
                                       sample_weight=None))
print("Precision =", metrics.precision_score(y_pred_class, y2, labels=None,
                                       pos_label=1, average='weighted',
                                       sample_weight=None))
print("Classification Report =\n", metrics.classification_report(y_pred_class, y2,
                                                                 labels=None,
                                                                 target_names=None,
                                                                 sample_weight=None,
                                                                 digits=2,
                                                                 output_dict=False))
print("F1 Score = ",f1_score(y_pred_class, y2, average='macro'))
```

```
[1 1 1 ... 0 0 1]
Test Loss = 0.3060711622238159
Test Accuracy = 0.8103289008140564
Accuracy =  81.03289122091046
Confusion Matrix = [[21392     8]
 [15608 45324]]
Recall = 0.8103289122091046
Precision = 0.8902239378142467
Classification Report =
              precision    recall  f1-score   support

           0       0.58      1.00      0.73     21400
           1       1.00      0.74      0.85     60932

    accuracy                           0.81     82332
   macro avg       0.79      0.87      0.79     82332
weighted avg       0.89      0.81      0.82     82332

F1 Score =  0.7928239927644668
```

In [ ]: