In [10]:

```
# -*- coding: utf-8 -*-
"""
Created on Sun May  5 19:25:38 2019

@author: Grp47
"""

# import the external libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import hyperParameterReport as hpr
from time import time
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.linear_model import SGDClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.linear_model import LogisticRegression
from libitmal import kernelfuns as itmalkernelfuns
itmalkernelfuns.EnableGPU()
import warnings
warnings.filterwarnings('ignore')

# Load the dataset from the csv file using pandas
data = pd.read_csv('creditcard.csv')

data.head()

# Only use the 'Amount' and 'V1', ..., 'V28' features
features = ['V%d' % number for number in range(1, 29)] + ['Amount']
#In above, I used '4' to limit the subplots but actually we have to use '29'.

# The target variable which we would like to predict, is the 'Class' variable
target = 'Class'

# Creating an X variable (containing the features) and an y variable (containing only t
he target variable)
X = data[features]
y = data[target]

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

scaler.fit(X)

#Split the data set using 'train_test_split' function
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1
01)
```

In [11]:

```
######### 1st model tuning #########


state = 1
CV=5
VERBOSE=5

model = LogisticRegression(penalty = 'l2')

tuning_parameters = {
    'tol': [0.00001, 0.0001, 0.001, 0.01, 0.1],
    'solver': ('newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'),
#     'class_weight': (dict, 'balanced'),
    'max_iter': [1, 5, 50, 100, 200],
#     'multi_class': ('ovr', 'multinomial', 'auto')
}

start = time()
print('######### starting tuning 1st model...#########')
random_tuned = RandomizedSearchCV(model, tuning_parameters, cv=CV, n_iter=10, scoring=
'f1_micro', verbose=VERBOSE, n_jobs=-1, iid=True)


random_tuned.fit(X_train, y_train) # train
print('######### finished tuning 1st model...#########')

t = time()-start

b1, m1 = hpr.FullReport(random_tuned , X_test, y_test, t) # test
```

```
######### starting tuning 1st model...#########
Fitting 5 folds for each of 10 candidates, totalling 50 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent worker
s.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:   14.0s
[Parallel(n_jobs=-1)]: Done  46 out of  50 | elapsed:   34.7s remaining:
3.0s
[Parallel(n_jobs=-1)]: Done  50 out of  50 | elapsed:   37.2s finished
```

```
######### finished tuning 1st model...#########
SEARCH TIME: 40.75 sec

Best model set found on train set:

        best parameters={'tol': 0.001, 'solver': 'liblinear', 'max_iter':
100}
        best 'f1_micro' score=0.9991353771204108
        best index=9

Best estimator CTOR:
        LogisticRegression(C=1.0, class_weight=None, dual=False, fit_inter
cept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='liblinea
r',
          tol=0.001, verbose=0, warm_start=False)

Grid scores ('f1_micro') on development set:
        [ 0]: 0.999 (+/-0.000) for {'tol': 1e-05, 'solver': 'newton-cg',
'max_iter': 50}
        [ 1]: 0.999 (+/-0.000) for {'tol': 0.0001, 'solver': 'newton-cg',
'max_iter': 50}
        [ 2]: 0.999 (+/-0.000) for {'tol': 0.01, 'solver': 'lbfgs', 'max_i
ter': 200}
        [ 3]: 0.998 (+/-0.000) for {'tol': 0.1, 'solver': 'sag', 'max_ite
r': 100}
        [ 4]: 0.998 (+/-0.000) for {'tol': 0.01, 'solver': 'saga', 'max_it
er': 1}
        [ 5]: 0.998 (+/-0.000) for {'tol': 0.001, 'solver': 'sag', 'max_it
er': 50}
        [ 6]: 0.998 (+/-0.000) for {'tol': 0.01, 'solver': 'liblinear', 'm
ax_iter': 5}
        [ 7]: 0.998 (+/-0.000) for {'tol': 0.0001, 'solver': 'sag', 'max_i
ter': 5}
        [ 8]: 0.998 (+/-0.000) for {'tol': 0.01, 'solver': 'lbfgs', 'max_i
ter': 1}
        [ 9]: 0.999 (+/-0.000) for {'tol': 0.001, 'solver': 'liblinear',
'max_iter': 100}

Detailed classification report:
        The model is trained on the full development set.
        The scores are computed on the full evaluation set.

             precision    recall  f1-score   support

          0       1.00      1.00      1.00     56859
          1       0.92      0.63      0.75       103

   micro avg       1.00      1.00      1.00     56962
   macro avg       0.96      0.82      0.87     56962
weighted avg       1.00      1.00      1.00     56962


CTOR for best model: LogisticRegression(C=1.0, class_weight=None, dual=Fal
se, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='liblinea
r',
          tol=0.001, verbose=0, warm_start=False)
```

```
best: dat=N/A, score=0.99914, model=LogisticRegression(max_iter=100,solver
='liblinear',tol=0.001)
```

In [12]:

```
######### 2nd model tuning #########

model = SGDClassifier()

tuning_parameters = {
    'epsilon': [0.1, 0.01, 0.001, 0.0001, 0.00001, 0.000001],
    'loss': ('log', 'hinge', 'modified_huber', 'squared_hinge', 'perceptron', 'squared_
loss'),
    'penalty': ['l1', 'l2', 'elasticnet'],
    'alpha': [0.1, 0.01, 0.001, 0.0001, 0.00001, 0.000001],
    'l1_ratio': [0.5, 0.1, 0.01, 0.001, 0.00001, 0.000001]
}

start = time()
print('######### starting tuning 2nd model... #########')
random_tuned = RandomizedSearchCV(model, tuning_parameters, cv=CV, n_iter=10, scoring=
'f1_micro', verbose=VERBOSE, n_jobs=-1, iid=True)


random_tuned.fit(X_train, y_train) # train
print('######### finished tuning 2nd model...#########')

t = time()-start

b1, m1 = hpr.FullReport(random_tuned , X_test, y_test, t) # test
```

```
######### starting tuning 2nd model... #########
Fitting 5 folds for each of 10 candidates, totalling 50 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent worker
s.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:    0.7s
[Parallel(n_jobs=-1)]: Done  46 out of  50 | elapsed:    3.8s remaining:
0.3s
[Parallel(n_jobs=-1)]: Done  50 out of  50 | elapsed:    4.1s finished
```

```
######### finished tuning 2nd model...#########
SEARCH TIME: 4.38 sec

Best model set found on train set:

        best parameters={'penalty': 'l1', 'loss': 'modified_huber', 'l1_ra
tio': 1e-06, 'epsilon': 1e-06, 'alpha': 0.0001}
        best 'f1_micro' score=0.9992099892470758
        best index=8

Best estimator CTOR:
        SGDClassifier(alpha=0.0001, average=False, class_weight=None,
       early_stopping=False, epsilon=1e-06, eta0=0.0, fit_intercept=True,
       l1_ratio=1e-06, learning_rate='optimal', loss='modified_huber',
       max_iter=None, n_iter=None, n_iter_no_change=5, n_jobs=None,
       penalty='l1', power_t=0.5, random_state=None, shuffle=True,
       tol=None, validation_fraction=0.1, verbose=0, warm_start=False)

Grid scores ('f1_micro') on development set:
        [ 0]: 0.999 (+/-0.000) for {'penalty': 'l1', 'loss': 'log', 'l1_ra
tio': 0.01, 'epsilon': 0.001, 'alpha': 1e-05}
        [ 1]: 0.999 (+/-0.000) for {'penalty': 'l2', 'loss': 'hinge', 'l1_
ratio': 0.001, 'epsilon': 0.01, 'alpha': 0.0001}
        [ 2]: 0.999 (+/-0.000) for {'penalty': 'l2', 'loss': 'log', 'l1_ra
tio': 0.5, 'epsilon': 1e-06, 'alpha': 0.01}
        [ 3]: 0.489 (+/-0.860) for {'penalty': 'elasticnet', 'loss': 'squa
red_loss', 'l1_ratio': 0.001, 'epsilon': 0.001, 'alpha': 0.0001}
        [ 4]: 0.998 (+/-0.000) for {'penalty': 'l1', 'loss': 'perceptron',
'l1_ratio': 0.5, 'epsilon': 1e-05, 'alpha': 0.1}
        [ 5]: 0.593 (+/-0.933) for {'penalty': 'l2', 'loss': 'squared_los
s', 'l1_ratio': 1e-05, 'epsilon': 0.1, 'alpha': 1e-06}
        [ 6]: 0.999 (+/-0.000) for {'penalty': 'elasticnet', 'loss': 'hing
e', 'l1_ratio': 1e-05, 'epsilon': 0.01, 'alpha': 1e-06}
        [ 7]: 0.999 (+/-0.000) for {'penalty': 'elasticnet', 'loss': 'modi
fied_huber', 'l1_ratio': 0.001, 'epsilon': 1e-05, 'alpha': 0.0001}
        [ 8]: 0.999 (+/-0.000) for {'penalty': 'l1', 'loss': 'modified_hub
er', 'l1_ratio': 1e-06, 'epsilon': 1e-06, 'alpha': 0.0001}
        [ 9]: 0.742 (+/-0.541) for {'penalty': 'elasticnet', 'loss': 'squa
red_loss', 'l1_ratio': 0.01, 'epsilon': 0.1, 'alpha': 0.01}

Detailed classification report:
        The model is trained on the full development set.
        The scores are computed on the full evaluation set.

              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56859
           1       0.82      0.79      0.80       103

   micro avg       1.00      1.00      1.00     56962
   macro avg       0.91      0.89      0.90     56962
weighted avg       1.00      1.00      1.00     56962


CTOR for best model: SGDClassifier(alpha=0.0001, average=False, class_weig
ht=None,
       early_stopping=False, epsilon=1e-06, eta0=0.0, fit_intercept=True,
       l1_ratio=1e-06, learning_rate='optimal', loss='modified_huber',
       max_iter=None, n_iter=None, n_iter_no_change=5, n_jobs=None,
       penalty='l1', power_t=0.5, random_state=None, shuffle=True,
       tol=None, validation_fraction=0.1, verbose=0, warm_start=False)
```

```
best: dat=N/A, score=0.99921, model=SGDClassifier(alpha=0.0001,epsilon=1e-
06,l1_ratio=1e-06,loss='modified_huber',penalty='l1')
```

In [14]:

```
######### 3rd model tuning #########

model = MLPClassifier(max_iter=50)

tuning_parameters = {
    'hidden_layer_sizes': [(20, 50, 100, 100, 50, 20), (50,50,50)],
    'activation': ['tanh', 'relu'],
    'alpha': [0.0001, 0.001],
    'learning_rate' : ('constant', 'adaptive'),
    'solver' : ('sgd', 'adam')
}

start = time()
print('######### starting tuning 3rd model... #########')
random_tuned = GridSearchCV(model, tuning_parameters, cv=CV, scoring='f1_micro', verbos
e=VERBOSE, n_jobs=-1, iid=True)


random_tuned.fit(X_train, y_train) # train
print('######### finished tuning 3rd model... #########')

t = time()-start

b1, m1 = hpr.FullReport(random_tuned , X_test, y_test, t) # test
```

```
######### starting tuning 3rd model... #########
Fitting 5 folds for each of 32 candidates, totalling 160 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent worker
s.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:  2.1min
[Parallel(n_jobs=-1)]: Done  56 tasks      | elapsed: 14.7min
[Parallel(n_jobs=-1)]: Done 160 out of 160 | elapsed: 40.2min finished
```

```
######### finished tuning 3rd model... #########
SEARCH TIME: 2447.16 sec

Best model set found on train set:

        best parameters={'activation': 'relu', 'alpha': 0.0001, 'hidden_la
yer_sizes': (50, 50, 50), 'learning_rate': 'constant', 'solver': 'adam'}
        best 'f1_micro' score=0.9995040488051088
        best index=21

Best estimator CTOR:
        MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto',
beta_1=0.9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=(50, 50, 50), learning_rate='constant',
       learning_rate_init=0.001, max_iter=50, momentum=0.9,
       n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
       random_state=None, shuffle=True, solver='adam', tol=0.0001,
       validation_fraction=0.1, verbose=False, warm_start=False)

Grid scores ('f1_micro') on development set:
        [ 0]: 0.999 (+/-0.000) for {'activation': 'tanh', 'alpha': 0.0001,
'hidden_layer_sizes': (20, 50, 100, 100, 50, 20), 'learning_rate': 'consta
nt', 'solver': 'sgd'}
        [ 1]: 0.999 (+/-0.000) for {'activation': 'tanh', 'alpha': 0.0001,
'hidden_layer_sizes': (20, 50, 100, 100, 50, 20), 'learning_rate': 'consta
nt', 'solver': 'adam'}
        [ 2]: 0.999 (+/-0.000) for {'activation': 'tanh', 'alpha': 0.0001,
'hidden_layer_sizes': (20, 50, 100, 100, 50, 20), 'learning_rate': 'adapti
ve', 'solver': 'sgd'}
        [ 3]: 0.999 (+/-0.001) for {'activation': 'tanh', 'alpha': 0.0001,
'hidden_layer_sizes': (20, 50, 100, 100, 50, 20), 'learning_rate': 'adapti
ve', 'solver': 'adam'}
        [ 4]: 0.999 (+/-0.000) for {'activation': 'tanh', 'alpha': 0.0001,
'hidden_layer_sizes': (50, 50, 50), 'learning_rate': 'constant', 'solver':
'sgd'}
        [ 5]: 0.999 (+/-0.000) for {'activation': 'tanh', 'alpha': 0.0001,
'hidden_layer_sizes': (50, 50, 50), 'learning_rate': 'constant', 'solver':
'adam'}
        [ 6]: 0.999 (+/-0.000) for {'activation': 'tanh', 'alpha': 0.0001,
'hidden_layer_sizes': (50, 50, 50), 'learning_rate': 'adaptive', 'solver':
'sgd'}
        [ 7]: 0.999 (+/-0.000) for {'activation': 'tanh', 'alpha': 0.0001,
'hidden_layer_sizes': (50, 50, 50), 'learning_rate': 'adaptive', 'solver':
'adam'}
        [ 8]: 0.999 (+/-0.000) for {'activation': 'tanh', 'alpha': 0.001,
'hidden_layer_sizes': (20, 50, 100, 100, 50, 20), 'learning_rate': 'consta
nt', 'solver': 'sgd'}
        [ 9]: 0.999 (+/-0.000) for {'activation': 'tanh', 'alpha': 0.001,
'hidden_layer_sizes': (20, 50, 100, 100, 50, 20), 'learning_rate': 'consta
nt', 'solver': 'adam'}
        [10]: 0.999 (+/-0.000) for {'activation': 'tanh', 'alpha': 0.001,
'hidden_layer_sizes': (20, 50, 100, 100, 50, 20), 'learning_rate': 'adapti
ve', 'solver': 'sgd'}
        [11]: 0.999 (+/-0.000) for {'activation': 'tanh', 'alpha': 0.001,
'hidden_layer_sizes': (20, 50, 100, 100, 50, 20), 'learning_rate': 'adapti
ve', 'solver': 'adam'}
        [12]: 0.999 (+/-0.000) for {'activation': 'tanh', 'alpha': 0.001,
'hidden_layer_sizes': (50, 50, 50), 'learning_rate': 'constant', 'solver':
'sgd'}
        [13]: 0.999 (+/-0.001) for {'activation': 'tanh', 'alpha': 0.001,
```

```
'hidden_layer_sizes': (50, 50, 50), 'learning_rate': 'constant', 'solver':
'adam'}
        [14]: 0.999 (+/-0.000) for {'activation': 'tanh', 'alpha': 0.001,
'hidden_layer_sizes': (50, 50, 50), 'learning_rate': 'adaptive', 'solver':
'sgd'}
        [15]: 0.999 (+/-0.000) for {'activation': 'tanh', 'alpha': 0.001,
'hidden_layer_sizes': (50, 50, 50), 'learning_rate': 'adaptive', 'solver':
'adam'}
        [16]: 0.999 (+/-0.000) for {'activation': 'relu', 'alpha': 0.0001,
'hidden_layer_sizes': (20, 50, 100, 100, 50, 20), 'learning_rate': 'consta
nt', 'solver': 'sgd'}
        [17]: 0.999 (+/-0.000) for {'activation': 'relu', 'alpha': 0.0001,
'hidden_layer_sizes': (20, 50, 100, 100, 50, 20), 'learning_rate': 'consta
nt', 'solver': 'adam'}
        [18]: 0.999 (+/-0.000) for {'activation': 'relu', 'alpha': 0.0001,
'hidden_layer_sizes': (20, 50, 100, 100, 50, 20), 'learning_rate': 'adapti
ve', 'solver': 'sgd'}
        [19]: 0.999 (+/-0.000) for {'activation': 'relu', 'alpha': 0.0001,
'hidden_layer_sizes': (20, 50, 100, 100, 50, 20), 'learning_rate': 'adapti
ve', 'solver': 'adam'}

Detailed classification report:
        The model is trained on the full development set.
        The scores are computed on the full evaluation set.

              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56859
           1       0.88      0.82      0.85       103

   micro avg       1.00      1.00      1.00     56962
   macro avg       0.94      0.91      0.92     56962
weighted avg       1.00      1.00      1.00     56962


CTOR for best model: MLPClassifier(activation='relu', alpha=0.0001, batch_
size='auto', beta_1=0.9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=(50, 50, 50), learning_rate='constant',
       learning_rate_init=0.001, max_iter=50, momentum=0.9,
       n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
       random_state=None, shuffle=True, solver='adam', tol=0.0001,
       validation_fraction=0.1, verbose=False, warm_start=False)

best: dat=N/A, score=0.99950, model=MLPClassifier(activation='relu',alpha=
0.0001,hidden_layer_sizes=(50, 50, 50),learning_rate='constant',solver='ad
am')
```

In [ ]: