Machine learning cybersecurity

**Fraud detection**

# LAB 9: Writing a classifier for CREDIT CARD FRAUD

**Lab Description:** This lab is to implement a binary classifier to distinguish if a transaction is a fraud or not. You are required to read the data from the credit card fraud dataset. The dataset is highly imbalanced, which has 284,315 records in total and only 492 records are belong to fraud class.

A close up of text on a white background

Description automatically generated

You are required to implement it in three ways:

* Using the machine learning software WEKA.
* Writing a python script with the use of the package sklearn
* Writing a python script with the use of the package tensorflow and deep learning techniques.

**Lab Environment:**

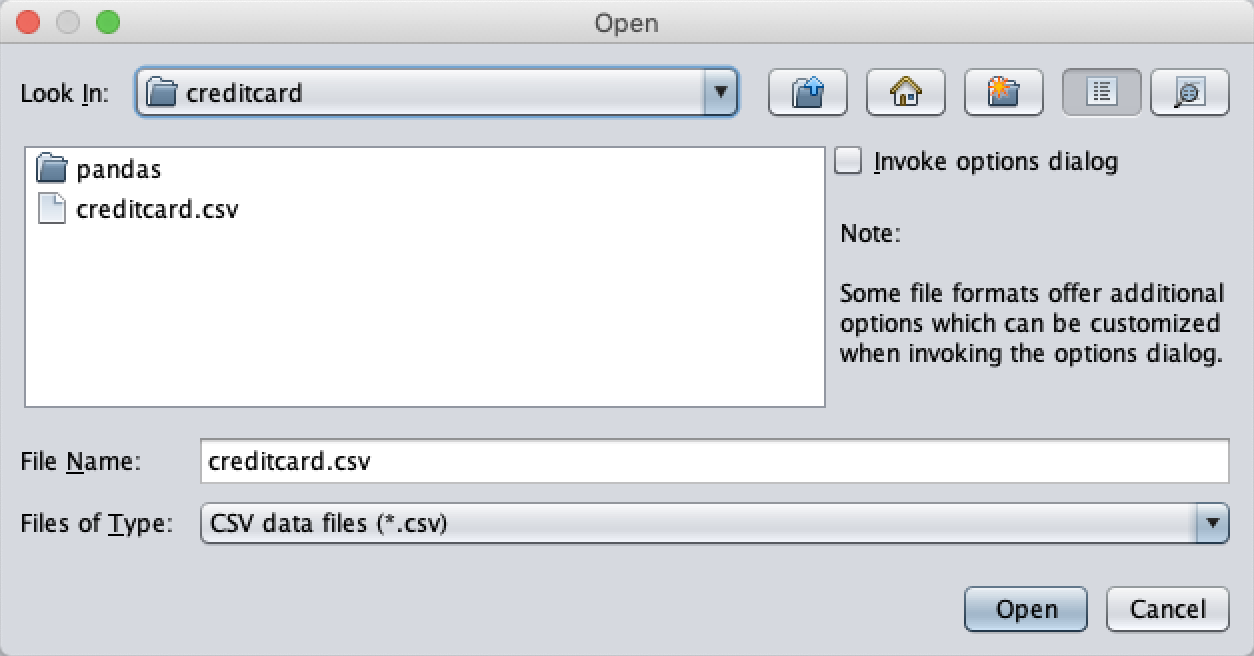
* The students should have access to a machine with Linux system
* WEKA should be installed
* The environment for python is required as well as some packages such as numpy, tensorflow and sklearn.

**Lab Files that are Needed:**

* For this lab you will need one file named creditcard.csv
* The last column is the class value, the first 30 columns are the features.

### **Lab exercise 1**

* Import data in the creditcard.csv into WEKA (explorer), the files of type should be specified (csv).

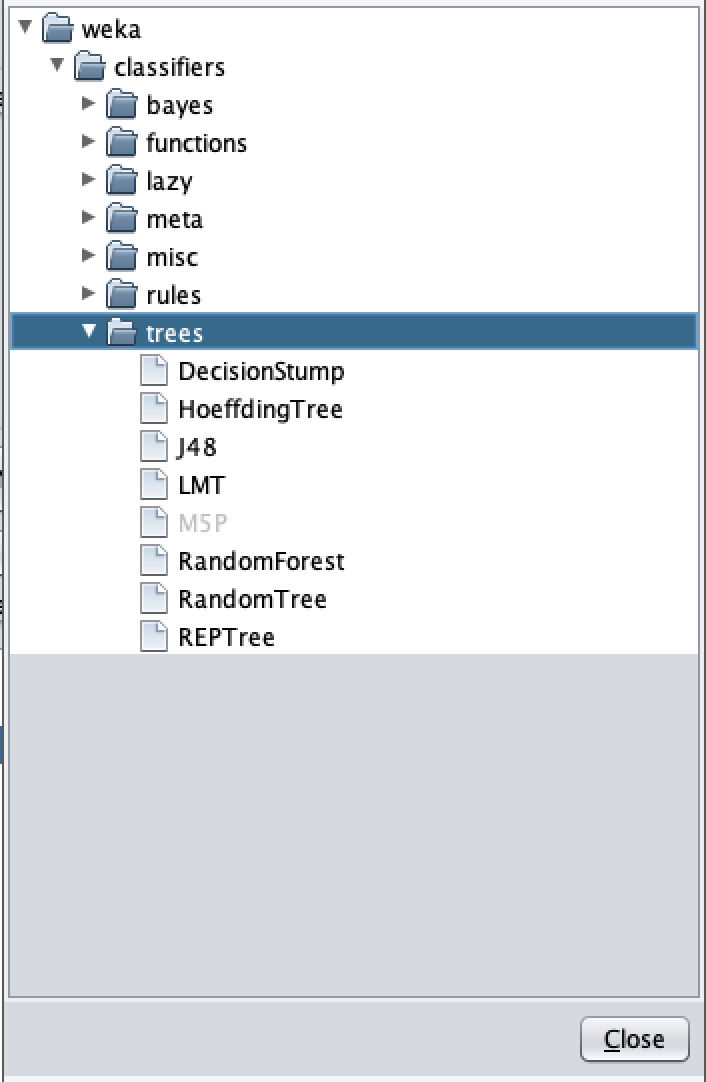


* Convert the type of class value from numeric to nominal

A screenshot of a cell phone

Description automatically generated

* Choose a proper classifier, such as J48 (Decision Tree)



* Specify the test option and the column of class. Here 10-fold cross validation was selected. Click Start button and see the performance

A screenshot of a cell phone

Description automatically generated

### **Lab exercise 2**

In this exercise, you need to implement several classifiers with the use of sklearn.

* Import the required libraries

import numpy as np
import pandas as pd
import sklearn
from numpy import genfromtxt
from sklearn import datasets
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeRegressor
from sklearn.datasets import fetch_mldata
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import (accuracy_score, confusion_matrix, f1_score,
                             precision_score, recall_score)
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler

* Read the features and class values from malware dataset with proper method

featuer_col = np.arange(30)
feature=pd.read_csv('creditcard.csv',delimiter=','\
        ,usecols=(featuer_col),dtype=float,skiprows=1)
target=pd.read_csv('creditcard.csv',delimiter=',',usecols=([30]),dtype=str,skiprows=1)


* + creditcard.csv is the name of the file.
  + delimiter indicates the character to split the data in a row.
  + usecols indicates which columns will be read. For features, the columns from 1 to 30 will be read. For class values, the last columns of the rows will be read.
  + dtype indicates the type of data to read
  + Since the first line of the file is names for each column, we set skip\_header to 1 to avoid read the first row.
* Split the dataset. When you finish the preprocess step, you can write the python script with the use of sklearn package to build your architecture of classifier.

labels = LabelEncoder().fit_transform(target)
feature_std = StandardScaler().fit_transform(feature)
x_train, x_test, y_train, y_test = train_test_split(feature_std, labels, test_size=0.25, random_state=0)


* random\_state is the seed used by the random number generator
* This is for the logistic regression:

A screenshot of a cell phone

Description automatically generated

* Please print the statistics metrics such as accuracy, recall, precision and f1 score.

def print_stats_metrics(y_test, y_pred):    
    print('Accuracy: %.2f' % accuracy_score(y_test,y_pred) )
    confmat = confusion_matrix(y_true=y_test, y_pred=y_pred)
    print ("confusion matrix")
    print(confmat)
    print (pd.crosstab(y_test, y_pred, rownames=['True'], colnames=['Predicted'], margins=True))
    print('Precision: %.3f' % precision_score(y_true=y_test, y_pred=y_pred,average='binary'))
    print('Recall: %.3f' % recall_score(y_true=y_test, y_pred=y_pred))
    print('F1-measure: %.3f' % f1_score(y_true=y_test, y_pred=y_pred))

* Implement the classifiers based on Decision Tree, Support Vector Machine and Random Forest

### **Lab exercise 3**

* Use the same data you use in the exercise 2.
* In this exercise, you will implement an artificial neural network classifier based on Tensorflow
* Import the required libraries

import tensorflow as tf
import numpy as np
import pandas as pd
from numpy import genfromtxt
from sklearn import datasets
from sklearn.datasets import fetch_mldata
from sklearn.model_selection import train_test_split
import sklearn
from sklearn.preprocessing import LabelEncoder

* Repeat the same steps to preprocess the data as Exercise 2. Read the data, standard scale the feature and encode the labels.

A screenshot of a cell phone

Description automatically generated

* Define the learning rate, number of epochs and batch size for artificial neural network

learning_rate = 0.001
n_epochs = 10000
batch_size = 10000

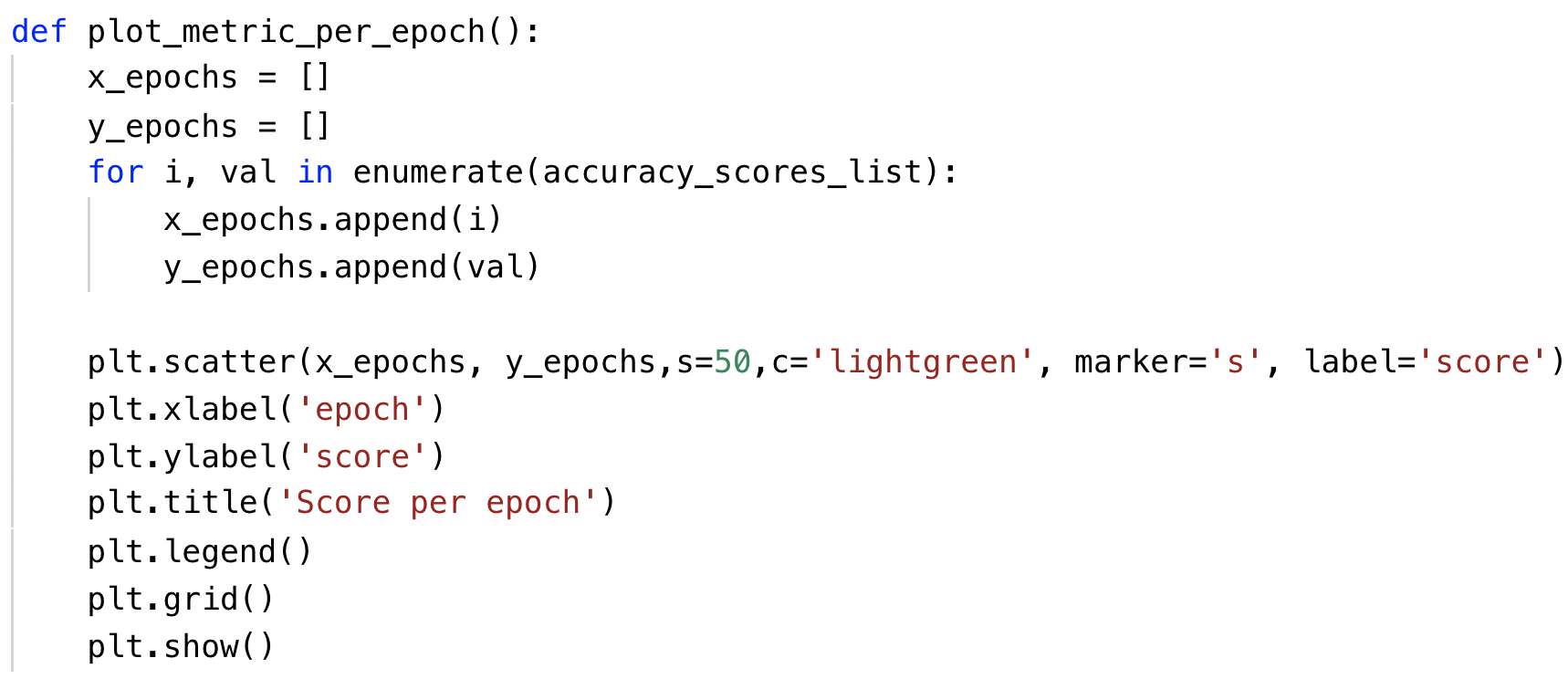
* An extra step in preprocess is to perform the one-hot encoding for the labels.

def convertOneHot(data,num):
    tensor = tf.one_hot(data,num)
    ss = tf.Session()
    array = ss.run(tensor)
    return array

* Define the parameters to store the shape of placeholder.

A=x_train.shape[1]
B=len(y_train_onehot[0])

* Define the function to draw the plot of performance



* Define your own architecture of neural network

def layer(input, weight_shape, bias_shape):
    weight_stddev = (2.0/weight_shape[0])**0.5
    w_init = tf.random_normal_initializer(stddev=weight_stddev)
    bias_init = tf.constant_initializer(value=0)
    W = tf.get_variable("W", weight_shape, initializer=w_init)
    b = tf.get_variable("b", bias_shape, initializer=bias_init)
    return tf.nn.relu(tf.matmul(input, W) + b)
###############################################################
def inference_deep_layers(x_tf, A, B):
    with tf.variable_scope("hidden_1"):
        hidden_1 = layer(x_tf, [A, 25],[25])
    with tf.variable_scope("hidden_2"):
        hidden_2 = layer(hidden_1, [25, 20],[20])
    with tf.variable_scope("hidden_3"):
        hidden_3 = layer(hidden_2, [20, 15],[15])
    with tf.variable_scope("hidden_4"):
        hidden_4 = layer(hidden_3, [15, 10],[10])
    with tf.variable_scope("output"):
        output = layer(hidden_4, [10, B], [B])
    return output

def loss_deep(output, y_tf):
    xentropy = tf.nn.softmax_cross_entropy_with_logits(logits=output, labels=y_tf)
    loss = tf.reduce_mean(xentropy) 
    return loss
############################################
################

def training(cost):
    optimizer = tf.train.GradientDescentOptimizer(learning_rate)
    train_op = optimizer.minimize(cost)
    return train_op

###########################################################
def evaluate(output, y_tf):
    correct_prediction = tf.equal(tf.argmax(output,1), tf.argmax(y_tf,1))
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
    return accuracy

* Please print the statistics metrics such as accuracy, recall, precision and f1 score.

precision_scores_list = []
accuracy_scores_list = []

def print_stats_metrics(y_test, y_pred):    
    print('Accuracy: %.2f' % accuracy_score(y_test,   y_pred) )
    #Accuracy: 0.84
    accuracy_scores_list.append(accuracy_score(y_test,   y_pred) )
    confmat = confusion_matrix(y_true=y_test, y_pred=y_pred)
    print ("confusion matrix")
    print(confmat)
    print (pd.crosstab(y_test, y_pred, rownames=['True'], colnames=['Predicted'], margins=True))
    precision_scores_list.append(precision_score(y_true=y_test, y_pred=y_pred))
    print('Precision: %.3f' % precision_score(y_true=y_test, y_pred=y_pred))
    print('Recall: %.3f' % recall_score(y_true=y_test, y_pred=y_pred))
    print('F1-measure: %.3f' % f1_score(y_true=y_test, y_pred=y_pred))

* Initialize the variables and placeholders. Then perform the training and testing on subset of kdd dataset.

A screenshot of a social media post

Description automatically generated

## What to Submit

You should submit a lab report file which includes:

* + The steps for how you preprocessed data
  + The necessary code snippet of your classifier and architecture.
  + The screenshot of the results
  + You can name your report "Lab\_creditFraud\_small\_yourname.doc".