Appendix A: Python Script for PDI

# the following libraries need to be installed (see pre-requisites)

# pandas

# matplotlib

# py4j

# numpy

# TPOT

# import required libraries

import pandas as pd

import numpy as np

from tpot import TPOTClassifier

from sklearn import preprocessing

from sklearn.model\_selection import train\_test\_split

# the dataset is referenced in the step: sv-change\_to\_numbers

# independent variables (x) are referenced starting at row 1: col 1. -1 references all columns apart from the last

x = dataset.iloc[:,1:-1].values

# transform features by scaling each feature to a given range

min\_max\_scaler = preprocessing.MinMaxScaler()

# compute the data minimum and maximum for scaling, then transform.

x\_scaled = min\_max\_scaler.fit\_transform(x)

# optional – change to numpy array

X=np.asarray(x\_scaled)

y=np.asarray(dataset.iloc[:,-1])

# split the dataset into train and test. Test size is set at 75% of dataset (10,000 rows)

# further details on random\_state:

# https://het.as.utexas.edu/HET/Software/Numpy/reference/generated/numpy.random.RandomState.html

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.75, random\_state=None)

# set TPOT parameters (see Appendix: B for further details)

tpot = TPOTClassifier(generations=1, verbosity=2, population\_size=100, scoring='accuracy', config\_dict='TPOT light')

tpot.fit(X\_train, y\_train)

output\_score=str(tpot.score(X\_test, y\_test))

# export TPOT results as python script

tpot.export('tpot\_creditcard\_pipeline.py')

# print the TPOT result

print(tpot.score(X\_test, y\_test))

# PDI output fields which are defined in a dataframe which is mapped to a PDI output field: model\_df

model\_name=[x[0] for x in tpot.evaluated\_individuals\_.items()]

model\_gen=[x[1]['generation'] for x in tpot.evaluated\_individuals\_.items()]

model\_mut=[x[1]['mutation\_count'] for x in tpot.evaluated\_individuals\_.items()]

model\_cross=[x[1]['crossover\_count'] for x in tpot.evaluated\_individuals\_.items()]

model\_predec=[x[1]['predecessor'] for x in tpot.evaluated\_individuals\_.items()]

model\_opp=[x[1]['operator\_count'] for x in tpot.evaluated\_individuals\_.items()]

model\_cv=[str(y[1]['internal\_cv\_score']) for y in tpot.evaluated\_individuals\_.items()]

model\_list=list(zip(model\_name,model\_gen,model\_mut,model\_cross,model\_predec,model\_opp,model\_cv))

model\_df=pd.DataFrame(model\_list,columns=['pipe','generation','mutation','crossover','predecessor','operator','cv'])

Appendix B: TPOT Parameters

| **Parameter** | **Valid values** | **Effect** |
| --- | --- | --- |
| **generation** | Any positive integer | The number of generations to run pipeline optimization over. Generally, TPOT will work better when you give it more generations (and therefore time) to optimize over. TPOT will evaluate generations x population\_size number of pipelines in total. |
| **population\_size** | Any positive integer | The number of individuals in the GP population. Generally, TPOT will work better when you give it more generations (and therefore time) to optimize over. TPOT will evaluate generations x population\_size number of pipelines in total. |
| **mutation\_rate** | [0.0, 1.0] | The mutation rate for the genetic programming algorithm in the range [0.0, 1.0]. This tells the genetic programming algorithm how many pipelines to apply random changes to every generation. We don't recommend that you tweak this parameter unless you know what you're doing. |
| **crossover\_rate** | [0.0, 1.0] | The crossover rate for the genetic programming algorithm in the range [0.0, 1.0]. This tells the genetic programming algorithm how many pipelines to “breed" every generation. We don't recommend that you tweak this parameter unless you know what you're doing. |
| **num\_cv\_folds** | [2, 10] | The number of folds to evaluate each pipeline over in k-fold cross-validation during the TPOT pipeline-optimization process. |
| **scoring** | 'accuracy', 'adjusted\_rand\_score', 'average\_precision', 'f1', 'f1\_macro', 'f1\_micro', 'f1\_samples', 'f1\_weighted', 'log\_loss', 'mean\_absolute\_error', 'mean\_squared\_error', 'median\_absolute\_error', 'precision', 'precision\_macro', 'precision\_micro', 'precision\_samples', 'precision\_weighted', 'r2', 'recall', 'recall\_macro', 'recall\_micro', 'recall\_samples', 'recall\_weighted', 'roc\_auc' or a callable function with signature scorer(y\_true, y\_pred) | Function used to evaluate the quality of a given pipeline for the problem. By default, balanced accuracy is used for classification and mean squared error is used for regression. TPOT assumes that any function with "error" or "loss" in the name is meant to be minimized, whereas any other functions will be maximized. |
| **max\_time\_mins** | Any positive integer | How many minutes TPOT has to optimize the pipeline. This setting will override the generations parameter. |
| **random\_state** | Any positive integer | The random number generator seed for TPOT. Use this to make sure that TPOT will give you the same results each time you run it against the same dataset with that seed. |
| **verbosity** | {0, 1, 2, 3} | How much information TPOT communicates while it's running. 0 = none, 1 = minimal, 2 = high, 3 = all. A setting of 2 or higher will add a progress bar to calls to fit(). |
| **disable\_update\_check** | [True, False] | Flag indicating whether the TPOT version checker should be disabled. |

*Icon

Description automatically generatedFurther details can be found at* [*Using TPOT*](https://epistasislab.github.io/tpot/using/)*.*

Appendix C: R Script for Train

# the following packages need to be installed (see pre-requisites)

# rJava

# randomForest

# load randomForest package

library(randomForest)

# the dataset is referenced in the step: sv-convert\_booleans\_to\_numbers

# assign the dataframe: train to variable: train.df

train.df <- as.data.frame(train)

# train model with 8 trees takes about 50 seconds on this VM

# declare variable rf assigned the results

# ( y) dependent variable = train.df (dataset) reported\_as\_fraud\_historic (column)

# ~ (tilde) . (point) = (x) independent variables

rf <- randomForest(train.df$reported\_as\_fraud\_historic ~ ., train.df, ntree=8, importance=TRUE)

# save model to output folder: Note no spaces and double backslashes are required.

save(rf, file="C:\\Machine--Learning\\01\_Credit\_Card\\Lab\_02\_Credit\_Card\_Fraud\\train\_model\_output\\rf.rdata")

# print message ok to indicate no probs..

# declare variable: ok assigned with the value “Finished”

ok <- "Finished"

# assign variable ok to dataframe: ok.df

ok.df <- as.data.frame(ok)

ok.df

Appendix D: R Script for Predict

# the following packages need to be installed (see pre-requisites)

# rJava

# randomForest

# load randomForest package

library(randomForest)

# the dataset is referenced in the step: sv-convert\_booleans\_to\_numbers

# assign the dataframe: test to variable: test.df

test.df <- as.data.frame(test)

# load randomForest Model

load("C:\\Machine--Learning\\01\_Credit\_Card\\Lab\_02\_Credit\_Card\_Fraud\\train\_model\_output\\rf.rdata")

# declare a variable pred which is assigned the results of using the predict function

# predict function runs our randomForest model against test.df = newdata

# predict with new test data

pred <- predict(rf, newdata = test.df)

# assign pred dataframe to pred.df variable

pred.df <- as.data.frame(pred)

# prepare output data

# using cbind function to bind pred.df to test.df

submission <- data.frame(cbind(test.df,pred.df))

# output data

submission