

Lab Guide  
Machine Learning with  
Pentaho Data Integration (PDI)

Credit Card Fraud – randomForest in R



Change log:

|  |  |  |  |
| --- | --- | --- | --- |
| Date | **Courseware Version** | **Microcode** | **Notes** |
| June 2020 | 1.0 | 9.0.0.3 – 582 |  |
|  |  |  |  |
|  |  |  |  |

Contents

[Terminology 1](#_Toc46312288)

[Prerequisites 2](#_Toc46312289)

[Installing Python 2](#_Toc46312290)

[Installing Libraries 4](#_Toc46312291)

[Google Colab 5](#_Toc46312292)

[Installing R 6](#_Toc46312293)

[Setting R Environment Variables 7](#_Toc46312294)

[Installing the randomForest Library 11](#_Toc46312295)

[Installing RStudio (Optional) 12](#_Toc46312296)

[Configuring Pentaho Data Integration with R 13](#_Toc46312297)

[Verifying Your Installation 13](#_Toc46312298)

[Lab 1: Credit Card Fraud – AutoML 15](#_Toc46312299)

[Objectives 15](#_Toc46312300)

[Step 1 – Data Preparation 15](#_Toc46312301)

[Step 2 – Feature Engineering 18](#_Toc46312302)

[Step 3 – Test Machine Learning Models to Identify the Most Accurate Model 18](#_Toc46312303)

[TPOT: py-auto\_ml 18](#_Toc46312304)

[Colab 20](#_Toc46312305)

[AutoML Script 21](#_Toc46312306)

[Python Executor 23](#_Toc46312307)

[Lab 2: Credit Card Fraud – randomForest 29](#_Toc46312308)

[Objectives 29](#_Toc46312309)

[Train the Model 29](#_Toc46312310)

[R Script Executor: Train Model 30](#_Toc46312311)

[Predict Fraudulent Credit Card Transactions 33](#_Toc46312312)

[Appendix A: Python Script for PDI 38](#_Toc46312313)

[Appendix C: R Script for Train 42](#_Toc46312314)

[Appendix D: R Script for Predict 43](#_Toc46312315)

[Related Information 44](#_Toc46312316)

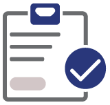
This page intentionally left blank.

Terminology

The table below outlines the different course activities:

| Activity | Description |
| --- | --- |
| **Demonstration** | The Instructor will demonstrate the workflow, outlining the key concept(s). The student is not expected to replicate the Instructor’s demonstration; but understand the key concept(s) and workflow. |
| **Lab** | The Instructor will outline the key concepts, features and options. The student is expected to follow along with the Instructor so that they understand the key concept(s), features and options for the Exercise. |

The icon indicates an Info Tip. Info Tips help users understand unfamiliar workflows or actions.

The icon indicates that you need to be careful when implementing or configuring the step/option(s).

The icon indicates a Best Practice. A Best Practice is a method or technique that has been generally accepted as a standard way of doing things.

Prerequisites

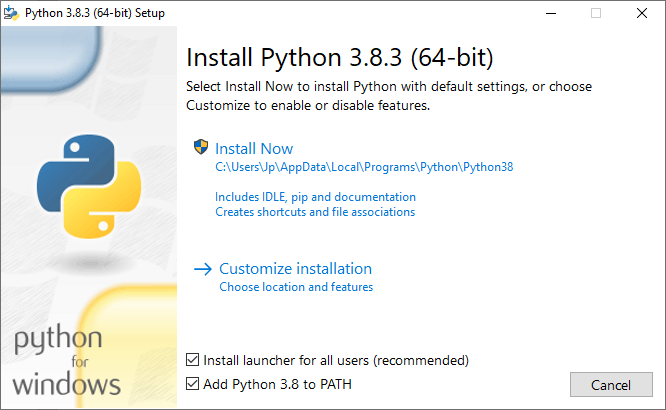
The following prerequisites need to be completed:

* Install Python for Windows
* Google Colab
* Install R for Windows
* Set R Environmental Variables
* Install R Studio for Windows
* Configure Pentaho Data Integration with R

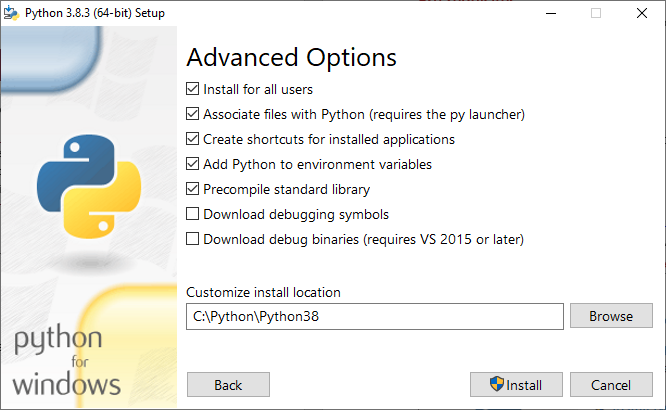
Installing Python

To install Python:

1. Download Python from [Python for Windows](https://www.python.org/downloads/windows/).
2. Click on **Download Windows x86-64 executable installer** on the page.
3. Run python-3.8.3-amd64.exe and follow the installation instructions.



1. Ensure you select:
   1. Customize Installation
   2. Install launcher
   3. Add Python 3.8 to PATH
2. Keep default options and click **Next**.
3. Ensure you select:
   1. Install for all users
   2. Precompile standard library
   3. Change the Path to: C:\Python\Python38



1. Click **Install**.

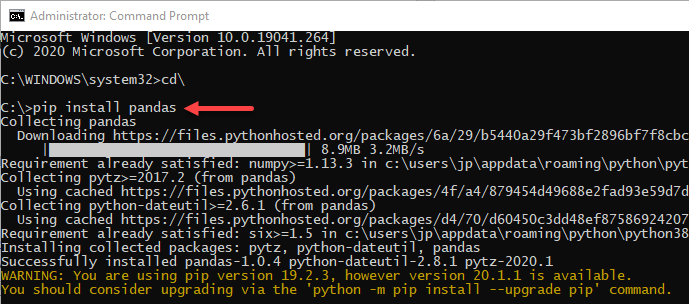
Installing Libraries

The following libraries need to be installed:

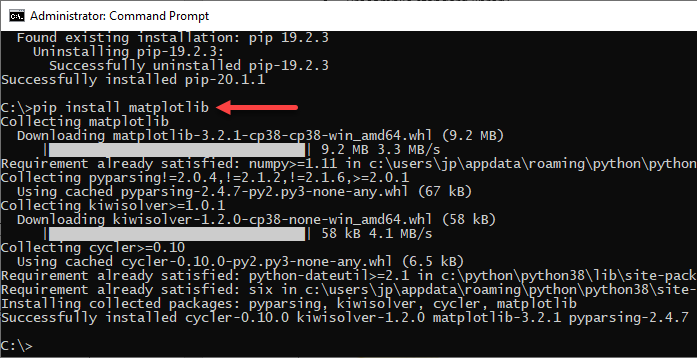
* pandas
* matplotlib
* py4j
* numpy
* wheel
* sklearn
* TPOT

To do this:

1. Open a Command Prompt (Admin) window.
2. Enter the following command to install pandas: pip install pandas



1. Repeat to download and install the other required libraries:



You may need to restart your system.

Google Colab

Colab is a Python development environment, based on Jupyter Notebooks, that runs in the browser using Google Cloud. It provides a runtime, fully configured for deep learning libraries, such as Keras, TensorFlow, PyTorch, and OpenCV.

* For further details: [Google Colab](https://colab.research.google.com/notebooks/intro.ipynb)
* Recommended to sync with Google Drive.

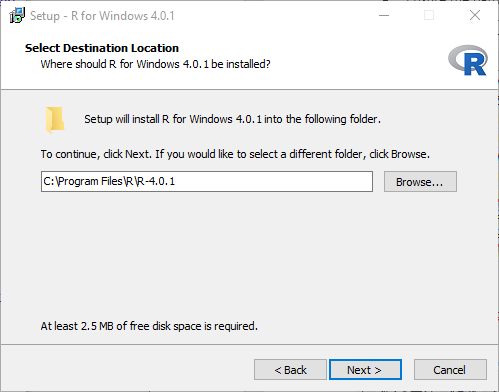
Installing R

To install R:

1. Download R from [r-project](https://cran.r-project.org/mirrors.html) by selecting a CRAN location (a mirror site) and clicking the corresponding link.
2. Click **Download R for Windows** at the top of the page.
3. Click **install R for the first time** at the top of the page.



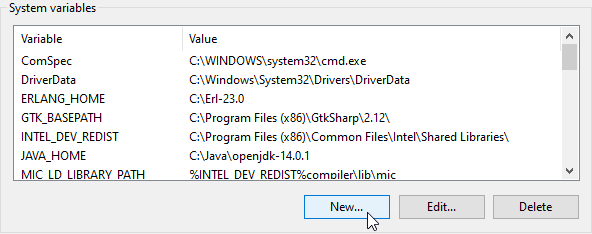
1. Click **Download R <version> for Windows**.
2. Run R-4.0.1-win.exe and follow the installation instructions.



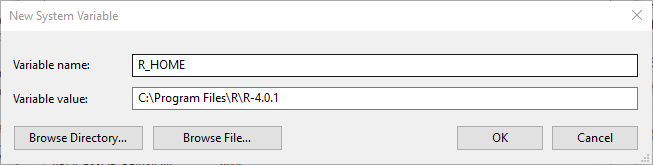
Setting R Environment Variables

To set the R environment variables:

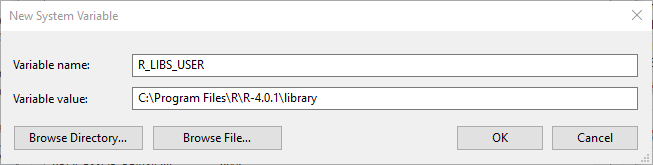
1. Go to **Control Panel** 🡪 **System** 🡪 **Advanced System Settings**.
2. Click the **Environment Variables** button.
3. Under **System Variables**, click: **New**.



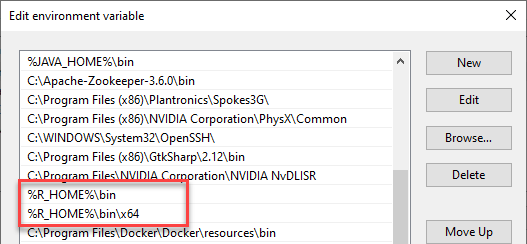
1. In the **Variable Name** field, enter R\_HOME
2. Browse for the directory C:\Program Files\R\R-4.0.1



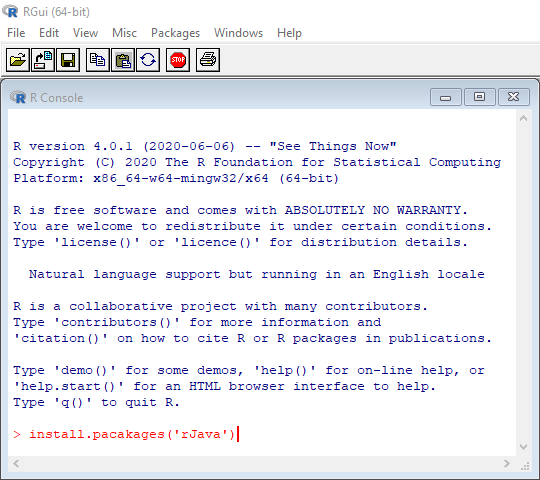
1. Repeat to add the variable R\_LIBS\_USER
2. Browse for the directory C:\Program Files\R\R-4.0.1\library



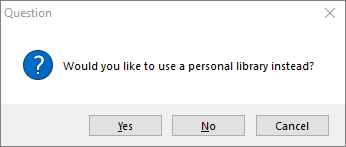
1. Add to the Path the location of the R executable: %R\_HOME%\bin\x64
2. Ensure the path references rcmd.exe and r.dll.



1. Start R.
2. In the R Console, run the command install.packages('rJava')

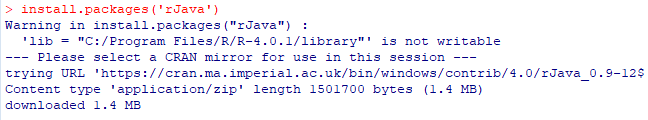


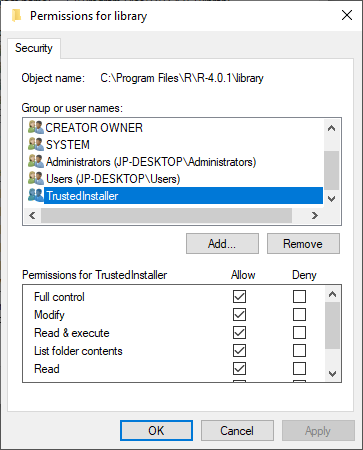
1. If prompted with **Would you like to use a personal library instead?** click **Yes**.

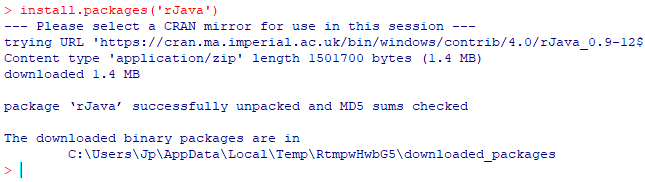


1. If prompted with the path of the library, click **Yes**.
2. When prompted for the CRAN mirror, choose a country, and then click **OK**.

You may be denied permission writing to the library folder. You will need to change the permission for the folder.



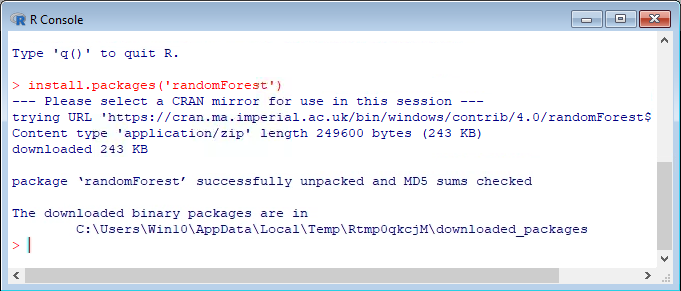




Installing the randomForest Library

To install the randomForest library:

1. Click on the R Console icon.
2. Enter the following command: install.packages('randomForest')

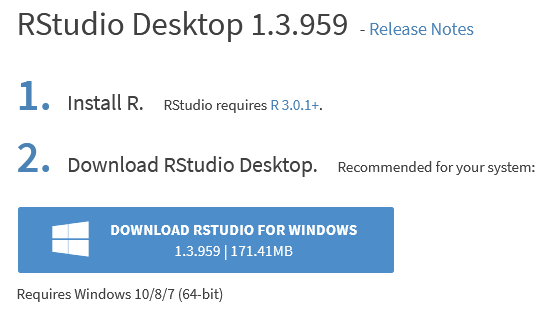


1. After randomForest has successfully been installed, type q() to quit the R console.
2. Click **Yes** to close the workplace image.
3. Close R.

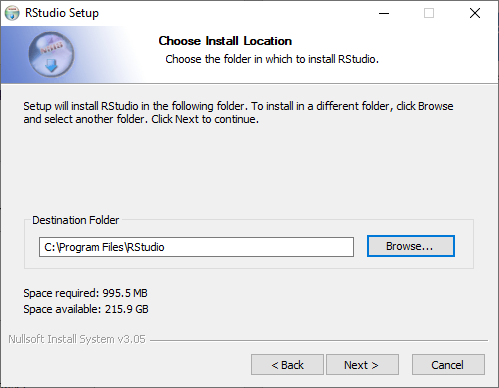
Installing RStudio (Optional)

To install RStudio:

1. Download RStudio from [RStudio IDE](https://rstudio.com/products/rstudio/download/#download).
2. Click on the **Download RStudio for Windows** button.



1. Run RStudio-1.3.959.exe and follow the installation instructions.



Configuring Pentaho Data Integration with R

In the rJava directory, there is a jri.dll file that needs to be copied into the libswt directory of Spoon:

1. Stop Spoon, if it is running.
2. Find %R\_LIBS\_USER%/rJava/jri/x64/jri.dll
3. Copy jri.dll to the following directory:
   1. Windows: [Pentaho directory]/client-tools/data-integration/libswt/win64
   2. Linux: [Pentaho directory]/client-tools/data-integration/libswt/linux

Further details can be found in *R on PDI* in the [Pentaho Data Integration Best Practices library](https://support.pentaho.com/hc/en-us/articles/360000307943-Pentaho-Data-Integration#WPR).

Verifying Your Installation

1. Open a new transformation in PDI.
2. Drag an **R Script Executor** step onto the canvas.
3. Double-click the step and select the middle tab, **R Script**. You will see some comments at the top of the window:

# The main output is expected to be a data frame, unless "Output

# from script is text" is checked. So, to output a data frame the

# last statement in the script should be the name of the frame.

# In the case that the output is text (as would be seen on the

# R console), the last statement should be a "print" statement in

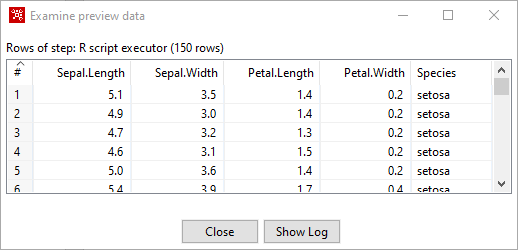
# order to print the object required.

1. Beneath the comment above, enter this code:

library(datasets)

iris

1. Once you have entered this code in the **R Script** tab, click the **Test Script** button on this tab.



Lab 1: Credit Card Fraud – AutoML

Imagine that a direct retailer wants to reduce losses due to orders involving fraudulent use of credit cards. They accept orders via phone and their web site, and ship goods directly to the customer. Basic customer details, such as customer name, date of birth, billing address and preferred shipping address, are stored in a relational database.

Orders, as they come in, are stored in a database. There is also a report of historical instances of fraud contained in a CSV spreadsheet.

Objectives

In this guided demonstration, you will:

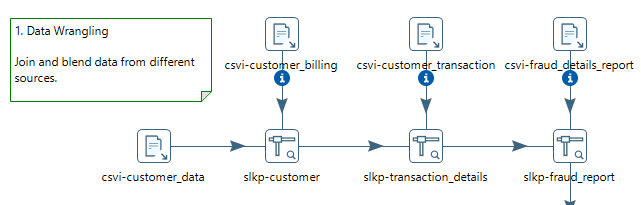
* Prepare Data
* Configure Python Executor step.
* Build and Train a Forest Tree Model.
* Deploy and Test the model.

Step 1 – Data Preparation

With the goal of preparing a dataset for ML, we can use PDI to combine these disparate data sources and engineer some features for learning from it. The following figure shows a transformation demonstrating an example of just that, and includes some steps for deriving new fields.

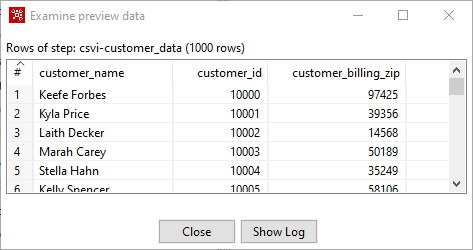
To begin with, customer data is joined from several data sources, and then blended with transactional data and historical fraud occurrences contained in a CSV file.

1. In Spoon, open the following main job: C:\Machine--Learning\01\_Credit\_Card\Lab\_01\_AutoML\tr\_autoML.ktr

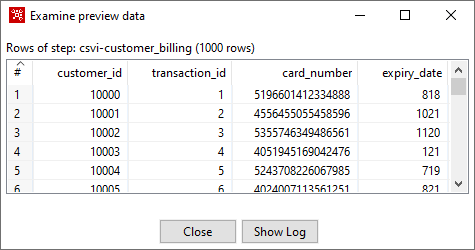


1. Browse the various customer data sources:

csvi-customer\_data, where you will find the customer\_billing\_zip codes, which will be used in feature engineering:

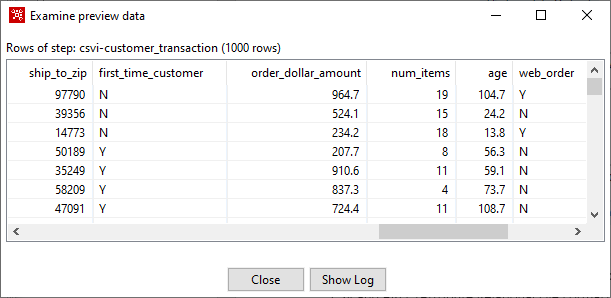


csvi-customer\_billing, which references the customer transaction:

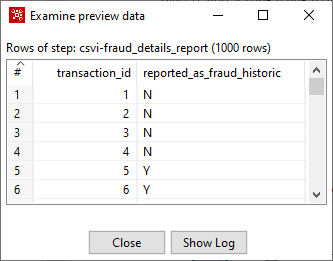


csvi-customer\_transaction:

* Customer transaction details
* Feature engineering for ship\_to\_zip
* The transaction details (x variables) are used by the decision trees to determine whether the transaction is fraudulent (y variable). The Boolean values will need to be changed into numbers for the randomForest algorithm.



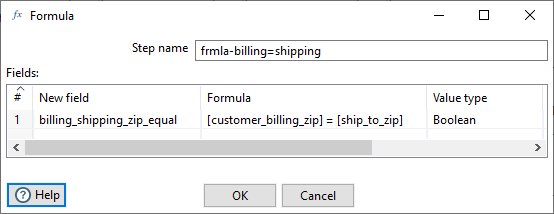
csvi-fraud\_details\_report, which indicates whether historically the transaction was fraudulent:



Step 2 – Feature Engineering

The Feature Engineering is set to billing zip code = shipping zip code.

1. Open the step **frmla-billing=shipping**.



There are steps for deriving additional fields that might be useful for predictive modeling. These include computing the customer's age, extracting the hour of the day the order was placed, and setting a flag to indicate whether the shipping and billing addresses have the same zip code.

Step 3 – Test Machine Learning Models to Identify the Most Accurate Model

So, what does the data scientist do at this point?

Typically, they will want to get a feel for the data by examining simple summary statistics and visualizations, followed by applying quick techniques for assessing the relationship between individual attributes (fields) and the target of interest which, in this example, is the reported\_as\_fraud\_historic field.

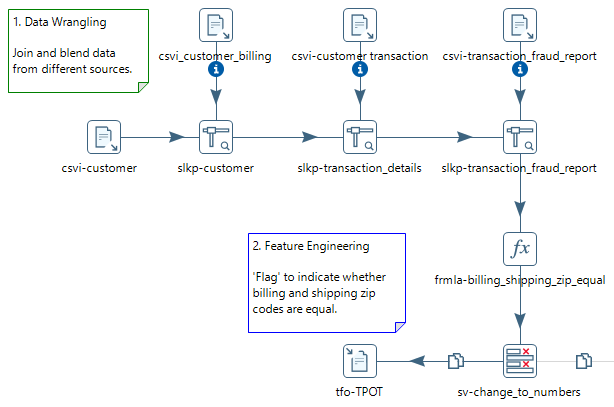
Following that, if there are attributes that look promising, quick tests with common supervised classification algorithms will be next on the list. This comprises the initial stages of experimental data mining – that is, the process of determining which predictive techniques are going to give the best result for a given problem.

TPOT: py-auto\_ml

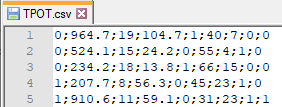
A Tree-based Pipeline Optimization Tool for Automating Machine Learning (TPOT) is a Python Automated Machine Learning tool that optimizes machine learning pipelines using genetic programming. TPOT will automate the most tedious part of machine learning by intelligently exploring thousands of possible pipelines to find the best one for your data.

One of the most useful tools for writing and executing Python script is Google Colab.

1. Run the transformation C:\Machine--Learning\01\_Credit\_Card\Lab\_01\_AutoML\tr\_autoML.ktr



1. Open the file C:\Machine--Learning\01\_Credit\_Card\Lab\_01\_AutoML\data\TPOT.csv

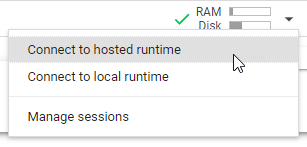


This will be the dataset used for autoML in Colab.

Colab

The following code overviews the steps:

1. In a Chrome browser, open https://colab.research.google.com/notebooks/intro.ipynb
2. Ensure you have connected to the hosted runtime.

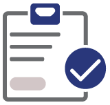


If you wish to browse the finished script:

1. Click on **File** 🡪 **Upload notebook**.

C:\Machine--Learning\01\_Credit\_Card\Lab\_01\_AutoML\scripts\credit\_card\_fraud.ipynb

1. Save the Notebook using **File** 🡪 **Save**

We recommend that you save either to GitHub or Google Drive.

AutoML Script

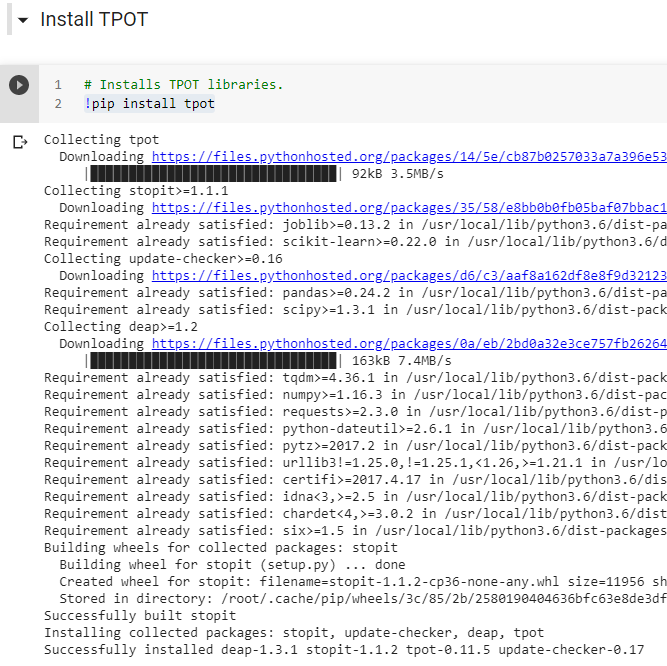
These are the code sections for the Jupyter file credit card fraud.ipynb:

1. Install the TPOT libraries:

# Installs TPOT libraries.

!pip install tpot

1. Run the step:



1. Import libraries:

import numpy as np

import pandas as pd

from tpot import TPOTClassifier

from sklearn import preprocessing

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

1. Import dataset from C:\Machine--Learning\01\_Credit\_Card\Lab\_01\_AutoML\data\TPOT.csv:

from google.colab import files

uploaded = files.upload()

for fn in uploaded.keys():

print('User uploaded file "{name}" with length {length} bytes'.format(

name=fn, length=len(uploaded[fn])))

dataset = pd.read\_csv('TPOT.csv', sep= ';', header=None)

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

y = dataset.iloc[:, 8].values

1. Add column headers:

dataset.columns = ['first\_time\_customer','order\_dollar\_amount','num\_items','age','web\_order','total\_transactions\_to\_date','hour\_of\_day','billing\_shipping\_zip\_equal','reported\_as\_fraud\_historic']

1. Convert dataset to numpy array and fit data (optional):

x = dataset.iloc[:,0:-1].values

min\_max\_scaler = preprocessing.MinMaxScaler()

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

X=np.asarray(x\_scaled)

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

1. Split the dataset. 75% used for test.

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

1. Run TPOT Classifier:

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

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

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

print(tpot.fitted\_pipeline\_)

1. Export Pipeline as Python script:

tpot.export('tpot\_exported\_credit\_card\_pipeline.py')

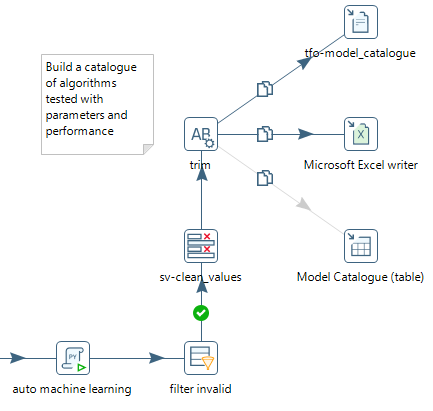
from google.colab import files

files.download('tpot\_exported\_credit\_card\_pipeline.py')

Python Executor

The script has been tested and is now ready for deployment in PDI.

1. Enable the rest of the hops in the transformation, except: **Model Catalogue (table)**:



1. Open the step **py-auto\_ml**
2. Ensure you set the path to Python:



For further details on the script, see [Appendix A](#AppendixA).

To ensure the script does not take a long time to process, the following TPOT parameters have been set:

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

For further details on TPOT parameters, see [Appendix B](#AppendixB).

1. Click on the **Input** tab.
   1. Use this tab to make selections for moving data from PDI fields to Python variables.
   2. The **All rows** option is commonly used for data frames. A data frame is used for storing data tables and is composed of a list of vectors of equal length.
   3. Select the **All rows** option to process all your data at once, for example, using the Python list of dictionaries.

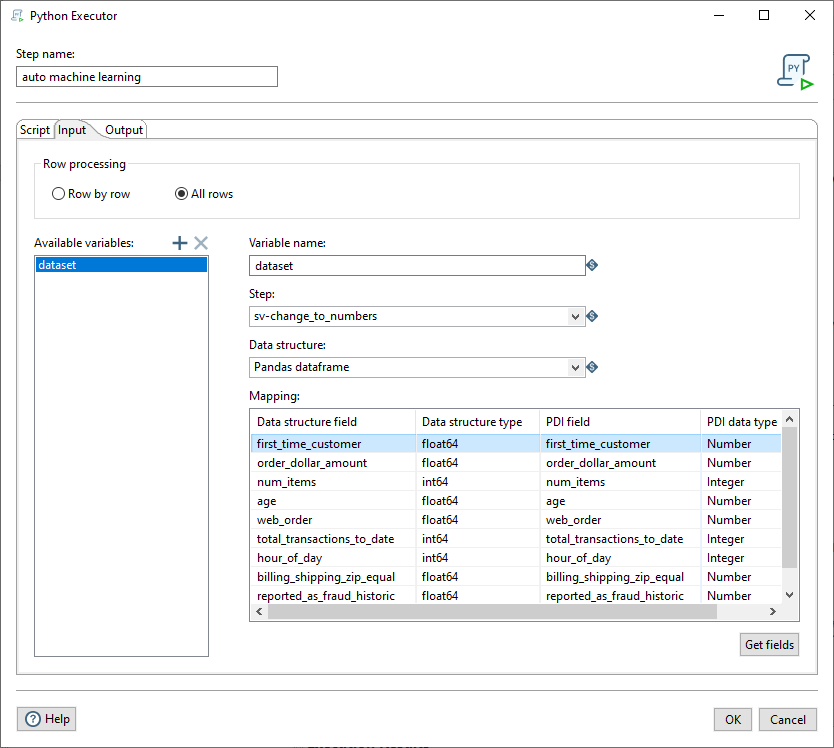
| Option | Description |
| --- | --- |
| **Available variables** | Use the plus sign button to add a Python variable to the input mapping for the script used in the transformation. You can remove the Python variable by clicking the X icon. |
| **Variable name** | Enter the name of the Python variable. The list of available variables will update automatically. |
| **Step** | Specify the name of the input step to map from. It can be any step in the parent transformation with an outgoing hop connected to the **Python Executor** step. |
| **Data structure** | Specify the data structure from which you want to pull the fields for mapping. You can select one of the following:   * **Pandas data frame**: the tabular data structure for Python/Pandas. * **NumPy array**: the table of values, all the same type, which is indexed by a tuple of positive integers. * **Python List of Dictionaries**: each row in the PDI stream becomes a Python dictionary. All the dictionaries are put into a Python list. |

1. The Mapping table contains the following field properties:

| Field Property | Description |
| --- | --- |
| **Data structure field** | The value of the Python data structure field to which you want to map the PDI field. |
| **Data structure type** | The value of the data structure type assigned to the data structure field to which you want to map the PDI field. |
| **PDI field** | The name of the PDI field which contains the vector data stored in the mapped Python variable. |
| **PDI data type** | The value of the data type assigned to the PDI field, such as a date, a number, or a timestamp. |

1. Select the **Get fields** button to populate the table with fields from the input step(s) in your transformation. If necessary, you can modify your selections.

For further details, see [Python Executor](https://help.pentaho.com/Documentation/9.0/Products/Python_Executor).



* The cust variable defines the dataframe in the Python script using iloc:

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

* The dataframe is pulled from the PDI step **sv-changes\_to\_numbers**.

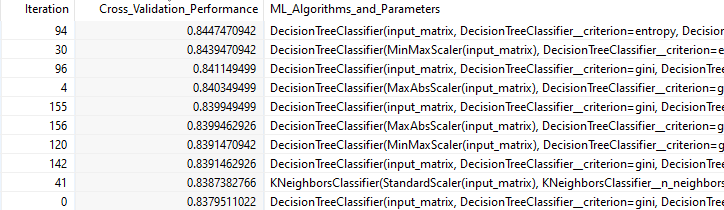
From this list, for the purposes of predictive modeling, we can drop the customer name, ID fields, email addresses, phone numbers and physical addresses. These fields are unlikely to be useful for learning purposes and, in fact, can be detrimental due to the large number of distinct values they contain.

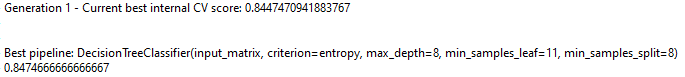
1. Click on the **Output** tab.
2. The output of model.df dataframe, from the script:

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

is converted back to PDI fields.

1. Examine the Logging.
2. Sort by **Cross\_Validation\_Performance**:





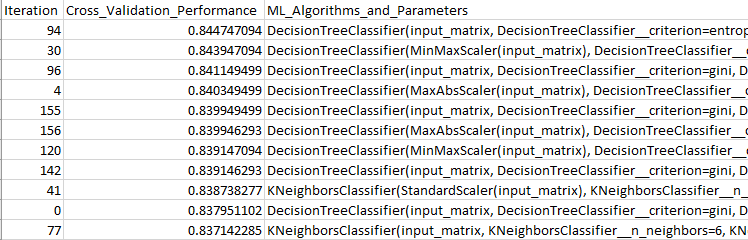
The output from the TPOT is tidied up before writing the results to a model catalogue file.

* Filter – removes invalid results
* Select values – orders and renames some of the output fields
* String Operations – trims data stream field
* Text file output – output results to the text file model\_catalogue.txt
* Microsoft Excel Writer – output results to the Excel workbook model\_catalogue.xlsx

What does this mean?

For the First Generation, the best algorithm pipeline run is DecisionTree with a scoring of 0.844 and accuracy of 0.84746 (figure used to judge the quality of the pipeline).

1. Open the Excel file C:\Machine--Learning\01\_Credit\_Card\Lab\_01\_AutoML\output\model\_catalogue.xlsx



Conclusion

The best pipeline to use (with 85% accuracy) for this dataset is based on Decision Trees with a minimum of eight trees.

It may also be worth looking at KNeighbors Classifier.

The object of using TPOT is to point you in the right direction for selecting the appropriate algorithm.

The results will be different each time you run the TPOTClassifier.

Lab 2: Credit Card Fraud – randomForest

The results from TPOT point to using a Decision Tree algorithm.

Objectives

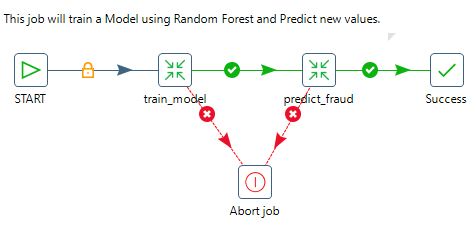
In this guided demonstration, you will:

* Train a randomForest model in R.
* Deploy your model.
* Predict fraudulent credit card transactions.

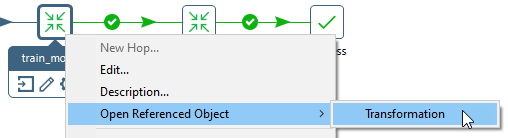
The model that will be used is randomForest.

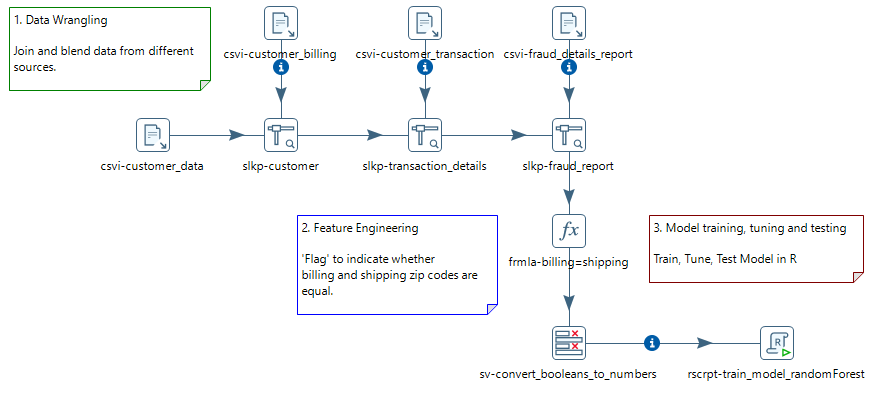
Train the Model

1. In Spoon, open the following main job: C:\Machine--Learning\01\_Credit\_Card\Lab\_02\_Credit\_Card\_Fraud\jb\_fraud\_main\_job.kjb



1. Let’s look at the transformation that trains for the model. Right-click on the **train\_model** transformation and select **Open Referenced Object** 🡪 **Transformation**.

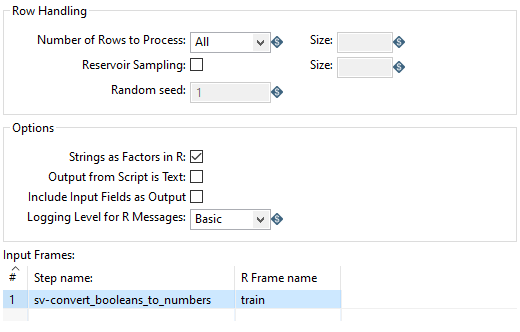




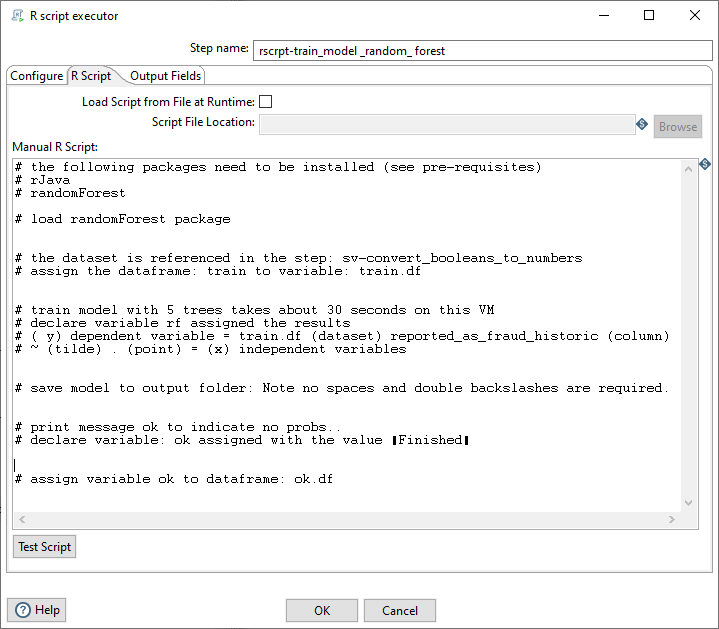
For an overview of the steps, see the previous [Lab 1: Credit Card Fraud – AutoML](#Lab1).

R Script Executor: Train Model

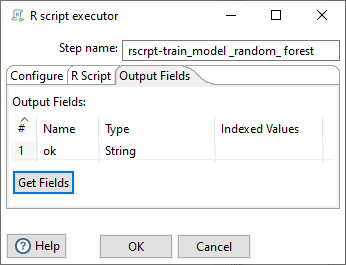
1. Double-click on the **rscrpt-train\_model\_randomForest** step to bring up the configuration settings.
2. Under the **Configure** tab, ensure the **Input Frames** points to the step name **sv-convert\_booleans\_to\_numbers** and the R Frame name **train**.



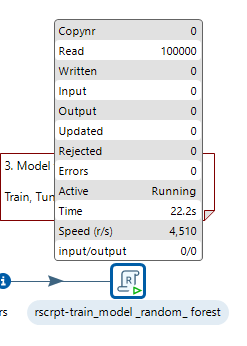
1. Set **Row Handling** to **Number of Rows to Process: All**.
2. Select the **R script** tab. Copy and paste the code snippets based on the Comments.



1. The required script is located at C:\Machine--Learning\Lab\_02\_Credit\_Card\_Fraud\scripts\train\_model.txt
2. Click on the **Output** tab.



1. Run the transformation.

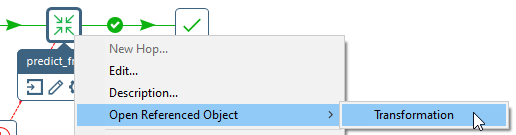


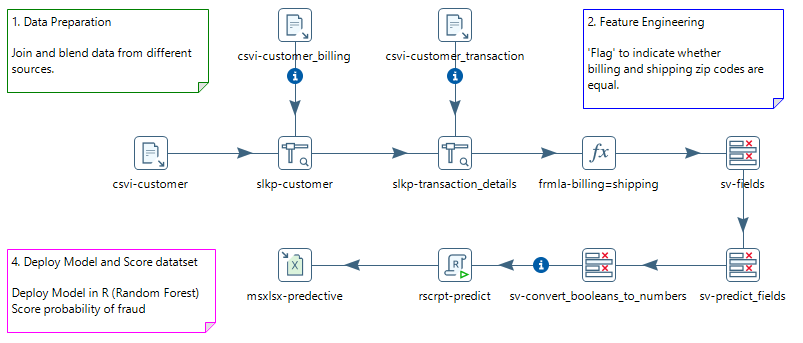
1. Check that the model has been saved in C:\Machine--Learning\01\_Credit\_Card\Lab\_02\_Credit\_Card\_Fraud\scripts\rf.rdata

Predict Fraudulent Credit Card Transactions

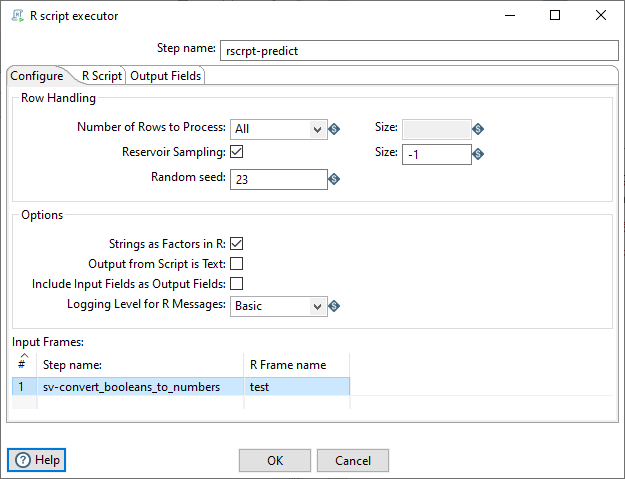
Let’s look at the transformation that predicts fraudulent credit card activity based on our trained model.

1. Right-click on the **predict\_model** transformation and select **Open Referenced Object** 🡪 **Transformation**.



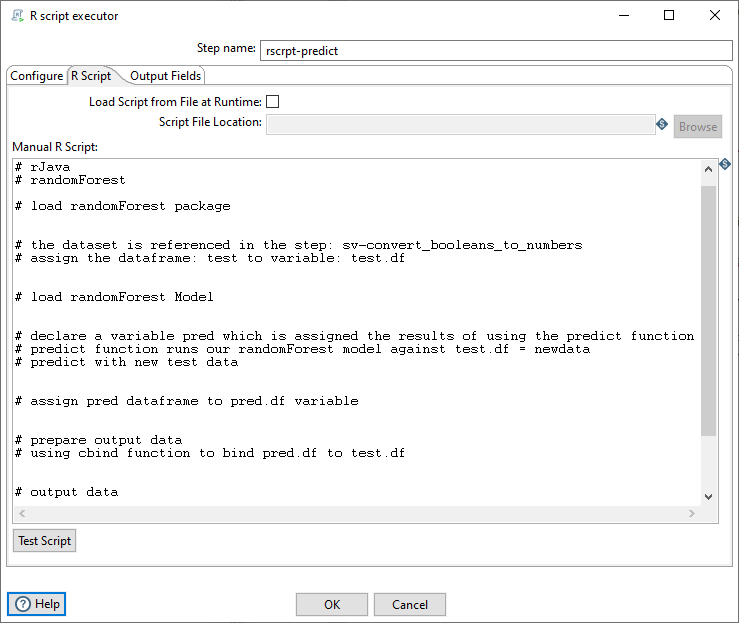


1. Double-click on the **rscrpt-predict** step to bring up the configuration settings.
2. Under the **Configure** tab, ensure the **Input Frames** points to the step name **sv-convert\_booleans\_to\_numbers** and that the **R Frame** name is **test**.

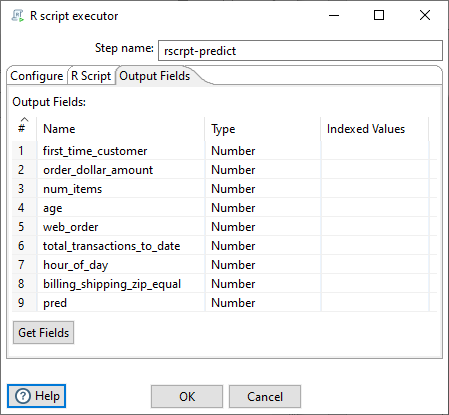


To remove any bias from the dataset, the complete dataset is randomly sampled (mixed up).

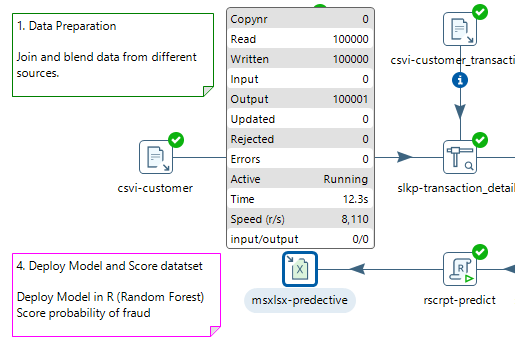
1. Select the **R script** tab. Copy and paste the code snippets based on the Comments.



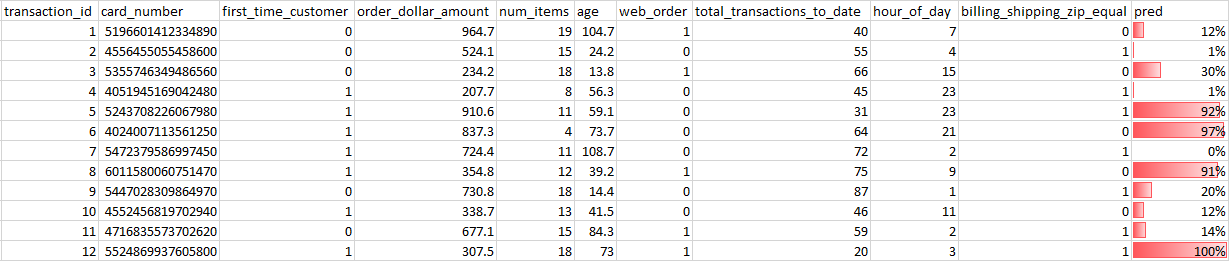
1. The required script is located at C:\Machine--Learning\01\_Credit\_Card\Lab\_02\_Credit\_Card\_Fraud\scripts\predict\_model.txt
2. Click on the **Output** tab.



1. Run the transformation.



1. Ensure all the steps have completed.
2. Open the Excel workbook C:\Machine--Learning\01\_Credit\_Card\Lab\_02\_Credit\_Card\_Fraud\output\credit\_card\_predict.xlsx



The complete solution can be found at C:\Machine--Learning\01\_Credit\_Card\Lab\_02\_Credit\_Card\_Fraud\solution

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. |

Further 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

Related Information

Here are some links to information that you may find helpful while using this lab guide:

* [Google Colab](https://colab.research.google.com/notebooks/intro.ipynb)
* [Pentaho Data Integration Best Practices Library](https://support.pentaho.com/hc/en-us/articles/360000307943-Pentaho-Data-Integration#WPR)
* [Python Executor](https://help.pentaho.com/Documentation/9.0/Products/Python_Executor)
* [Python for Windows](https://www.python.org/downloads/windows/)
* [r-project](https://cran.r-project.org/mirrors.html)
* [RStudio IDE](https://rstudio.com/products/rstudio/download/#download)