

VPS Customer churn prediction - Part 2

JG Pardyak

03-9-2021

Introduction

Apache Spark

Machine Learning

PySpark

Further steps

Introduction

Motivation

This is the continuation of the presentations:

- ▶ <https://github.com/JacekPardyak/vps/blob/master/vps-part-1.pdf> ,
- ▶ <https://github.com/JacekPardyak/vps/blob/master/vps-part-2.pdf> ,
- ▶ <https://github.com/JacekPardyak/vps/blob/master/vps-part-3.pdf>.

In this presentation we:

- ▶ demonstrate how to use **sparklyr** R interface for Apache Spark

Apache Spark

Setting up Spark connection

```
library(tidyverse)
library(sparklyr)
#spark_install(version = "3.1")
sc <- spark_connect(master = "local")
connection_is_open(sc)
```

```
## [1] TRUE
```

```
#spark_disconnect(sc)
```

```
## [1] '3.1.1'
```

Copy local data frames to a remote src

```
vps <- read_csv("./data/vps_churn_data.txt")
```

```
## Rows: 283 Columns: 23
```

```
## -- Column specification -----
```

```
## Delimiter: ","
```

```
## dbl (23): id, cpu_load_mean_m_3, disk_octets_read_mean_m_3
```

```
##
```

```
## i Use `spec()` to retrieve the full column specification
```

```
## i Specify the column types or set `show_col_types = FALSE`
```

```
pro <- read_csv("./data/vps_test_data.txt")
```

```
## Rows: 95 Columns: 23
```

```
## -- Column specification -----
```

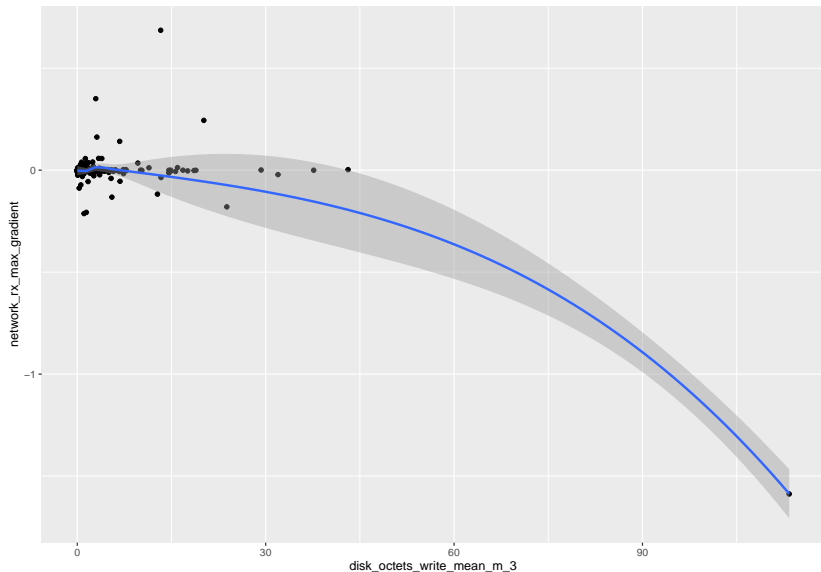
```
## Delimiter: ","
```

```
## dbl (22): id, cpu_load_mean_m_3, disk_octets_read_mean_m_3
```

```
## lgl (1): is_churn
```

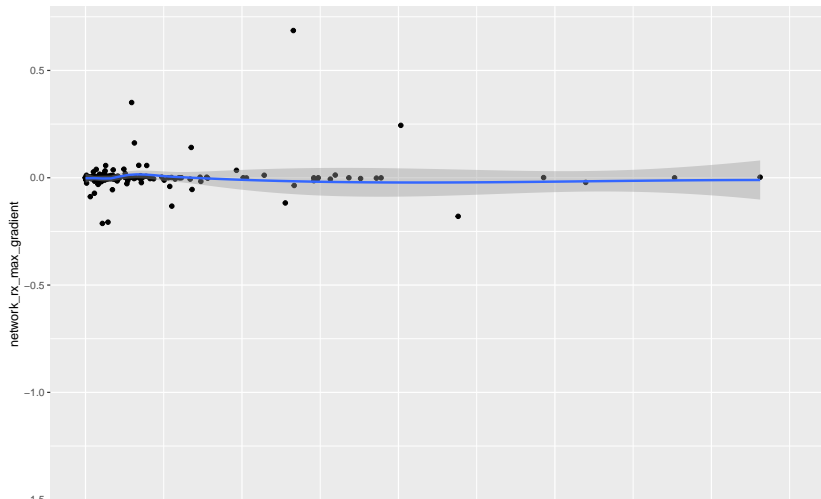
Plot some data from Spark

```
## `geom_smooth()` using method = 'loess' and formula 'y ~
```

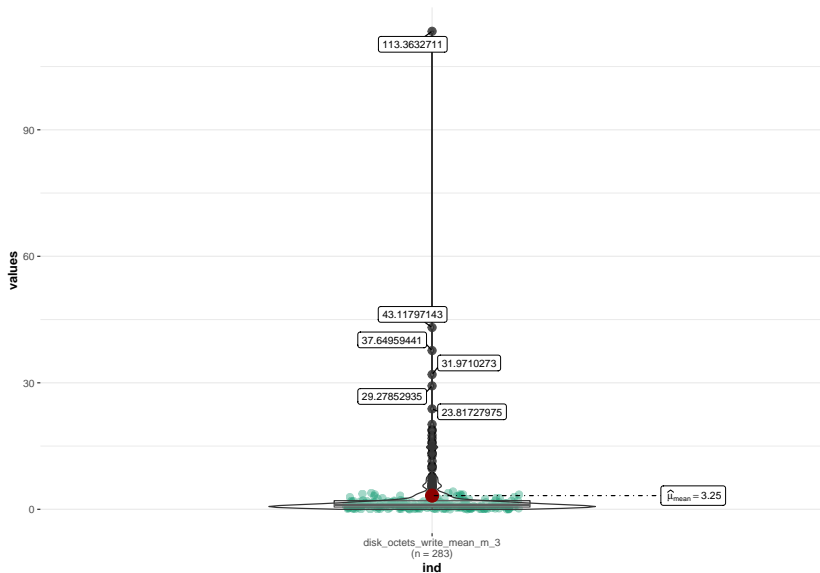


Plot some other data from Spark

```
## `geom_smooth()` using method = 'loess' and formula 'y ~  
## Warning: Removed 1 rows containing non-finite values (st  
## Warning: Removed 1 rows containing missing values (geom_
```



Data with outliers



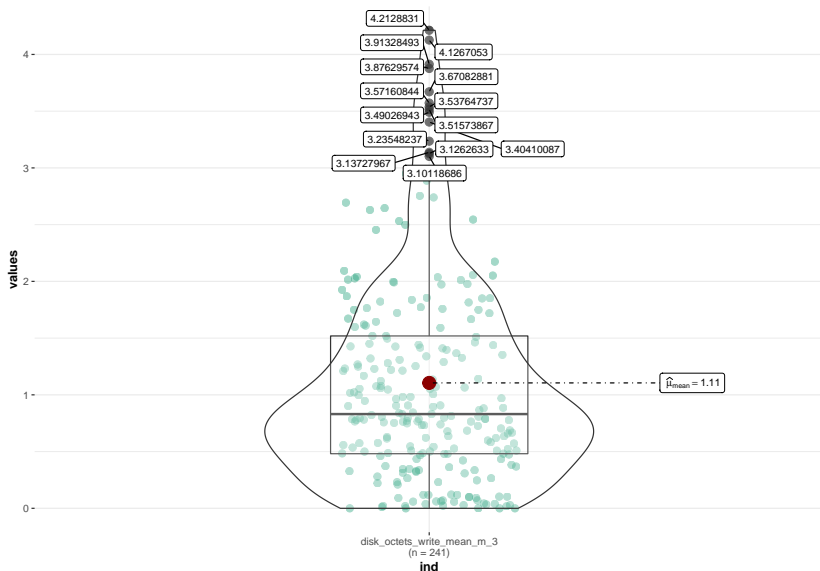
Outliers removal

```
var = 'disk_octets_write_mean_m_3'
tbl <- tbl(sc, 'vps') %>% select('id', all_of(var)) %>% co
summary(tbl)
```

```
##           id           disk_octets_write_mean_m_3
##  Min.      :100.0    Min.      :  0.0000
##  1st Qu.:170.5    1st Qu.:  0.5245
##  Median :241.0    Median :  1.0526
##  Mean   :241.0    Mean   :  3.2483
##  3rd Qu.:311.5    3rd Qu.:  2.0462
##  Max.   :382.0    Max.   :113.3633
```

```
Q <- tbl %>% select(!! sym(var)) %>% pull() %>% quantile(p
iqr <- tbl %>% select(!! sym(var)) %>% pull() %>% IQR()
#up <- Q[2]+1.5*iqr # Upper Range
#low<- Q[1]-1.5*iqr # Lower Range
proper <- tbl %>% filter(!! sym(var) > (Q[1] - 1.5*iqr) &
```

Data without outliers



Using SQL

##	id	cpu_load_mean_m_3	disk_octets_read_mean_m_3	disk_
## 1	100	13.379892	38.82583205	
## 2	101	16.902043	0.00328473	
## 3	102	248.935914	19.37152776	
## 4	103	14.054194	0.10182502	
## 5	104	85.793333	15.49345957	
## 6	105	39.394301	11.36519674	
## 7	106	1.288280	0.00176921	
## 8	107	79.788387	27.30295340	
## 9	108	68.253118	18.10863666	
## 10	109	8.866774	1.17613777	
##		disk_ops_read_mean_m_3	disk_ops_write_mean_m_3	networ
## 1		336.69720430	6.770323	
## 2		0.08150538	2.793979	
## 3		213.16032260	194.558280	
## 4		2.18870968	9.634839	
## 5		280.23193550	9.595376	
## 6		793.13655910	55.020000	

Machine Learning

Random forest, all vars, training & evaluation on all data

```
rf_model <- vps_tbl %>%  
  ml_random_forest(is_churn ~ ., type = "classification")  
rf_predict <- ml_predict(rf_model, vps_tbl) %>%  
  ft_string_indexer("is_churn", "is_churn_idx") %>% collect  
table(rf_predict$is_churn_idx, rf_predict$prediction)
```

```
##  
##           0    1  
##    0 139    9  
##    1   21 114
```

```
ml_evaluate(rf_model, vps_tbl)
```

```
## # A tibble: 1 x 1  
##   Accuracy  
##   <dbl>  
## 1    0.894
```

Random forest, all vars, evaluation on test data

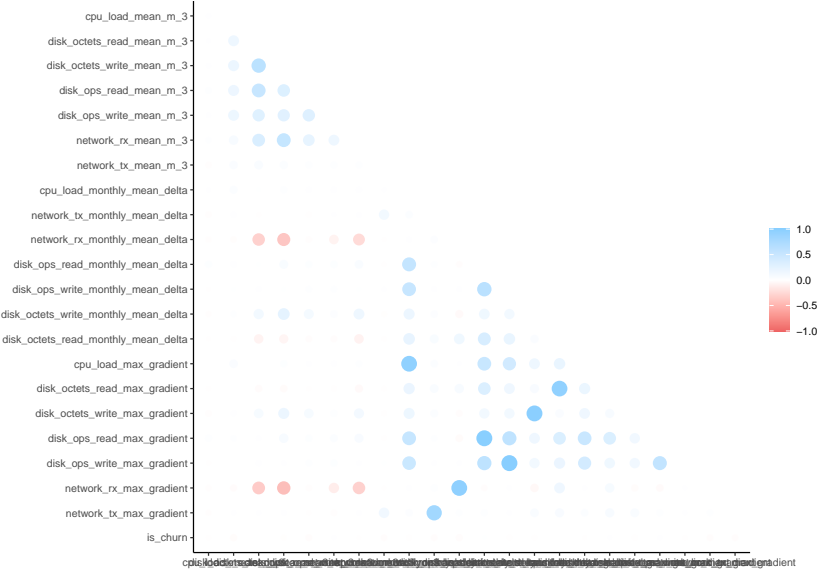
```
partitions <- tbl(sc, "vps") %>%  
  sdf_random_split(training = 0.6, test = 0.4, seed = 888)  
rf_model <- partitions$training %>%  
  ml_random_forest(is_churn ~ ., type = "classification")  
rf_predict <- ml_predict(rf_model, partitions$test) %>%  
  ft_string_indexer("is_churn", "is_churn_idx") %>% collect  
table(rf_predict$is_churn_idx, rf_predict$prediction)
```

```
##  
##      0   1  
##  0 39 22  
##  1 28 24
```

```
ml_evaluate(rf_model, partitions$test)
```

```
## # A tibble: 1 x 1  
##   Accuracy  
##   <dbl>  
## 1    0.554
```


Correlations



[1] NA

[1] 0.02383503

Random forest, chosen vars, evaluation on test data

```
partitions <- tbl(sc, "vps") %>%  
  sdf_random_split(training = 0.6, test = 0.4, seed = 888)  
rf_model <- partitions$training %>%  
  ml_random_forest(formula, type = "classification")  
rf_predict <- ml_predict(rf_model, partitions$test) %>%  
  ft_string_indexer("is_churn", "is_churn_idx") %>% collect  
table(rf_predict$is_churn_idx, rf_predict$prediction)
```

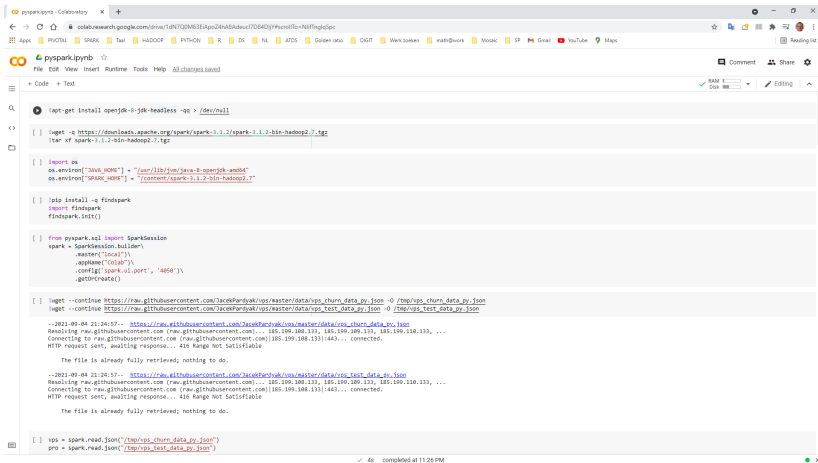
```
##  
##      0  1  
##    0 38 23  
##    1 24 28
```

```
ml_evaluate(rf_model, partitions$test)
```

```
## # A tibble: 1 x 1  
##   Accuracy  
##   <dbl>  
## 1    0.584
```

PySpark

PySpark



```
!apt-get install openjdk-8-jdk-headless -qq > /dev/null

!wget -q https://download.apache.org/spark/spark-3.1.2/spark-3.1.2-bin-hadoop2.7.tgz
!tar xf spark-3.1.2-bin-hadoop2.7.tgz

[ ] import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"] = "/content/spark-3.1.2-bin-hadoop2.7"

[ ] !pip install -q findspark
import findspark
findspark.init()

[ ] from pyspark.sql import SparkSession
spark = SparkSession.builder\
    .master("local")\
    .appName("Colab")\
    .config('spark.ui.port', '4050')\
    .getOrCreate()

[ ] !wget --continue https://raw.githubusercontent.com/JacekPardyak/vps/master/data/vps_churn_data_py.json -O /tmp/vps_churn_data_py.json
!wget --continue https://raw.githubusercontent.com/JacekPardyak/vps/master/data/vps_test_data_py.json -O /tmp/vps_test_data_py.json

--2021-09-04 21:24:57-- https://raw.githubusercontent.com/JacekPardyak/vps/master/data/vps_churn_data_py.json
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.132, 185.199.108.132, 185.199.110.132, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.132|:443... connected.
HTTP request sent, awaiting response... 416 Range Not Satisfiable

The file is already fully retrieved; nothing to do.

--2021-09-04 21:24:57-- https://raw.githubusercontent.com/JacekPardyak/vps/master/data/vps_test_data_py.json
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.132, 185.199.108.132, 185.199.110.132, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.132|:443... connected.
HTTP request sent, awaiting response... 416 Range Not Satisfiable

The file is already fully retrieved; nothing to do.

[ ] vps = spark.read.json("/tmp/vps_churn_data_py.json")
pro = spark.read.json("/tmp/vps_test_data_py.json")
```

✓ 48 - completed at 11:26 PM

Figure 1: In this Colaboratory notebook you will find a step-by-step guide on how to use PySpark for the classification of VPS customers with Random Forest Classifier.

<https://github.com/JacekPardyak/vps/blob/master/pyspark.ipynb>

Further steps

Further steps

- ▶ discuss with domain experts: outliers detected, crossing variables for feature engineering