Apple Case Study

Shane Lanan

January 18, 2019

Extract, Transform, and Load

A Python 3.7 backend was developed to: - Pull the data from its sources - Prepare for a load into Sqlite - Commit records and report upload status

In a production environment: - Use an enterprise database like Oracle, SQL Server, MySQL, etc. - Schedule the script in Unix via /etc/crontab

```
#!/usr/bin/python3
import urllib.request
import pandas as pd
import io
import sqlite3
from datetime import datetime
from tqdm import tqdm
dataUrl = "http://archive.ics.uci.edu/ml/machine-learning-databases/secom/secom.data"
labelUrl = "http://archive.ics.uci.edu/ml/machine-learning-databases/secom/secom_labels.data"
vendorUrl = "./data/vendordata.json"
# column names for sqlite
newCols = {
    "datetime": "MFG_DATE",
    "mat vendor": "MAT_VENDOR",
    "part vendor": "PART_VENDOR",
    "sil vendor": "SIL_VENDOR",
    "adhs vendor": "ADHS_VENDOR",
    "sop vendor": "SOP_VENDOR",
}
def dfUpload(df, con, table, timeStamp=True, clearTable=False, debug=False):
    if timeStamp:
        df['INSERTED_ON'] = datetime.now()
    df = df.where(pd.notnull(df), None)  # convert NaN to None, for SQL Nulls
    # just to fix pd.NaT to insert as NULLS
    for col in df.columns:
        if df[col].dtype.kind == 'M':
            df[col] = df[col].astype(object).where(df[col].notnull(), None)
            df[col] = df[col].dt.strftime('%Y-%m-%d %h:%m:%s')
    sqlColumns = '(' + ','.join([col for col in df.columns]) + ')'
    sqlValues = '(' + ','.join([':' + str(x + 1) for x in list(range(len(df.columns)))]) + ')'
    sqlInsert = "INSERT INTO %s %s VALUES %s" % (table, sqlColumns, sqlValues)
    crsr = con.cursor()
    # uploading
    if clearTable:
        crsr.execute("DELETE FROM %s" % table)
    for row in tqdm(df.values.tolist(), desc="Uploading data", unit="row"):
        if debug:
            try:
                crsr.executemany(sqlInsert, [row])
            except:
```

```
                print(row)
                pass
        else:
            crsr.executemany(sqlInsert, [row])
    con.commit()
    crsr.close()
def main():
    # tried pd.read_html(), but no tables found?
    def PandasFromUrl(url):
        return pd.read_csv(io.BytesIO(urllib.request.urlopen(url).read()),
                           encoding="utf8", sep=" ", header=None)
    print("Fetching data from web and formatting...")
    data = PandasFromUrl(dataUrl)
    data.columns = ["F" + str(i) for i in range(len(data.columns))]  # prefix feature columns with "F"
    data['PASS_FAIL'] = PandasFromUrl(labelUrl)[0]
    vendors = pd.read_json(vendorUrl).sort_index()
    df = data.merge(vendors, left_index=True, right_index=True)
    df.rename(index=str, columns=newCols, inplace=True)
    df['ID'] = list(range(len(df)))
    print("Connecting to Sqlite...")
    con = sqlite3.connect("warehouse.db")
    print("Clearing table and inserting records...")
    dfUpload(df, con, "SAMPLE", clearTable=True)
    print("Disconnecting from Sqlite...")
    con.close()
    print("Done!")


if __name__ == '__main__':
    main()
```

```
## Fetching data from web and formatting...
## Connecting to Sqlite...
## Clearing table and inserting records...
## Disconnecting from Sqlite...
## Done!
##
##
Uploading data:   0%|          | 0/1567 [00:00<?, ?row/s]
Uploading data:  35%|###5      | 555/1567 [00:00<00:00, 5532.57row/s]
Uploading data:  65%|######5   | 1025/1567 [00:00<00:00, 5247.87row/s]
Uploading data:  94%|#########4| 1475/1567 [00:00<00:00, 4993.17row/s]
Uploading data: 100%|##########| 1567/1567 [00:00<00:00, 5015.41row/s]
```

Prepare Environment

Loading required libraries, clearing cache, and defining a helper function

```
library(DBI)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
```

```
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(broom)
library(ROCR)
```

```
## Loading required package: gplots
```

```
##
## Attaching package: 'gplots'
```

```
## The following object is masked from 'package:stats':
##
##     lowess
```

```
library(extraTrees)
```

```
## Loading required package: rJava
```

```
rm(list = ls()) # clear all data
try(dev.off(),silent=TRUE) # clear all plots

# helper function to print huge dataframe
printWideDataFrame <- function(df, n){
  head(df[c(1:n,(ncol(df)-n):ncol(df))])
}
```

Fetch from Database

Since the data has already been normalized into Sqlite, a SELECT statement can be used to pull the table into RAM.

In a production environment: - For ODBC/JDBC, pass connection credentials to the connection object - For REST API "GET", call web service for response object

```
# connect to db, fetch table into RAM, disconnect from db
con <- dbConnect(RSQLite::SQLite(), "warehouse.db")
df_orig <- dbGetQuery(con, "select * from sample") %>%
  mutate(INSERTED_ON = as.POSIXct(INSERTED_ON), MFG_DATE = as.POSIXct(MFG_DATE))
dbDisconnect(con)

# preview dataframe
printWideDataFrame(df_orig, 20)
```

```
##   ID         INSERTED_ON            MFG_DATE PASS_FAIL MAT_VENDOR
## 1  0 2019-01-16 01:11:52 2008-07-19 11:55:00        -1        ddd
## 2  1 2019-01-16 01:11:52 2008-07-19 12:32:00        -1        eee
## 3  2 2019-01-16 01:11:52 2008-07-19 13:17:00         1        fff
## 4  3 2019-01-16 01:11:52 2008-07-19 14:43:00        -1        ccc
## 5  4 2019-01-16 01:11:52 2008-07-19 15:22:00        -1        ccc
## 6  5 2019-01-16 01:11:52 2008-07-19 17:53:00        -1        eee
##   PART_VENDOR SIL_VENDOR ADHS_VENDOR SOP_VENDOR      F0      F1       F2
## 1         aaa        ddd         bbb        eee 3030.93 2564.00 2187.733
## 2         ccc        ddd         aaa        aaa 3095.78 2465.14 2230.422
## 3         aaa        eee         aaa        jjj 2932.61 2559.94 2186.411
```

```
## 4         ccc        hhh         aaa        eee 2988.72 2479.90 2199.033
## 5         bbb        aaa         bbb        iii 3032.24 2502.87 2233.367
## 6         aaa        hhh         bbb        eee 2946.25 2432.84 2233.367
##          F3     F4  F5       F6     F7     F8      F9     F10    F569
## 1 1411.1265 1.3602 100  97.6133 0.1242 1.5005  0.0162 -0.0034      NA
## 2 1463.6606 0.8294 100 102.3433 0.1247 1.4966 -0.0005 -0.0148      NA
## 3 1698.0172 1.5102 100  95.4878 0.1241 1.4436  0.0041  0.0013 68.8489
## 4  909.7926 1.3204 100 104.2367 0.1217 1.4882 -0.0124 -0.0033 25.0363
## 5 1326.5200 1.5334 100 100.3967 0.1235 1.5031 -0.0031 -0.0072      NA
## 6 1326.5200 1.5334 100 100.3967 0.1235 1.5287  0.0167  0.0055 22.5598
##       F570   F571  F572   F573   F574   F575   F576    F577   F578   F579
## 1 533.8500 2.1113  8.95 0.3157 3.0624 0.1026 1.6765 14.9509     NA     NA
## 2 535.0164 2.4335  5.92 0.2653 2.0111 0.0772 1.1065 10.9003 0.0096 0.0201
## 3 535.0245 2.0293 11.21 0.1882 4.0923 0.0640 2.0952  9.2721 0.0584 0.0484
## 4 530.5682 2.0253  9.33 0.1738 2.8971 0.0525 1.7585  8.5831 0.0202 0.0149
## 5 532.0155 2.0275  8.83 0.2224 3.1776 0.0706 1.6597 10.9698     NA     NA
## 6 534.2091 2.3236  8.91 0.3201 2.2598 0.0899 1.6679 13.7755 0.0342 0.0151
##     F580     F581   F582   F583   F584    F585   F586   F587   F588
## 1     NA       NA 0.5005 0.0118 0.0035  2.3630     NA     NA     NA
## 2 0.0060 208.2045 0.5019 0.0223 0.0055  4.4447 0.0096 0.0201 0.0060
## 3 0.0148  82.8602 0.4958 0.0157 0.0039  3.1745 0.0584 0.0484 0.0148
## 4 0.0044  73.8432 0.4990 0.0103 0.0025  2.0544 0.0202 0.0149 0.0044
## 5     NA       NA 0.4800 0.4766 0.1045 99.3032 0.0202 0.0149 0.0044
## 6 0.0052  44.0077 0.4949 0.0189 0.0044  3.8276 0.0342 0.0151 0.0052
##       F589
## 1       NA
## 2 208.2045
## 3  82.8602
## 4  73.8432
## 5  73.8432
## 6  44.0077
```

Clean Data

Using dplyr commands to: - Force response variable to binary - Add dummy variables to all strings/factors - Remove columns no longer required in calculations

```
# massage for statistics
df_stats <- df_orig %>%
  mutate(PASS_FAIL = ifelse(PASS_FAIL==1,0,1)) %>%  # 1 = pass, 0 = fail
  fastDummies::dummy_cols() %>%  # add dummy variables for all string columns
  select(-c(ID, INSERTED_ON, MFG_DATE, MAT_VENDOR, PART_VENDOR, SIL_VENDOR, ADHS_VENDOR, SOP_VENDOR))

# preview dataframe
printWideDataFrame(df_stats, 20)
```

```
##   PASS_FAIL      F0      F1       F2        F3     F4  F5       F6     F7
## 1         1 3030.93 2564.00 2187.733 1411.1265 1.3602 100  97.6133 0.1242
## 2         1 3095.78 2465.14 2230.422 1463.6606 0.8294 100 102.3433 0.1247
## 3         0 2932.61 2559.94 2186.411 1698.0172 1.5102 100  95.4878 0.1241
## 4         1 2988.72 2479.90 2199.033  909.7926 1.3204 100 104.2367 0.1217
## 5         1 3032.24 2502.87 2233.367 1326.5200 1.5334 100 100.3967 0.1235
## 6         1 2946.25 2432.84 2233.367 1326.5200 1.5334 100 100.3967 0.1235
##       F8      F9     F10    F11      F12 F13     F14      F15     F16
## 1 1.5005  0.0162 -0.0034 0.9455 202.4396   0  7.9558 414.8710 10.0433
```

```
## 2 1.4966 -0.0005 -0.0148 0.9627 200.5470   0 10.1548 414.7347  9.2599
## 3 1.4436  0.0041  0.0013 0.9615 202.0179   0  9.5157 416.7075  9.3144
## 4 1.4882 -0.0124 -0.0033 0.9629 201.8482   0  9.6052 422.2894  9.6924
## 5 1.5031 -0.0031 -0.0072 0.9569 201.9424   0 10.5661 420.5925 10.3387
## 6 1.5287  0.0167  0.0055 0.9699 200.4720   0  8.6617 414.2426  9.2441
##      F17      F18 SIL_VENDOR_eee SIL_VENDOR_hhh SIL_VENDOR_aaa
## 1 0.9680 192.3963              0              0              0
## 2 0.9701 191.2872              0              0              0
## 3 0.9674 192.7035              1              0              0
## 4 0.9687 192.1557              0              1              0
## 5 0.9735 191.6037              0              0              1
## 6 0.9747 191.2280              0              1              0
##   SIL_VENDOR_ggg SIL_VENDOR_bbb SIL_VENDOR_ccc SIL_VENDOR_iii
## 1              0              0              0              0
## 2              0              0              0              0
## 3              0              0              0              0
## 4              0              0              0              0
## 5              0              0              0              0
## 6              0              0              0              0
##   SIL_VENDOR_fff ADHS_VENDOR_bbb ADHS_VENDOR_aaa SOP_VENDOR_eee
## 1              0               1               0              1
## 2              0               0               1              0
## 3              0               0               1              0
## 4              0               0               1              1
## 5              0               1               0              0
## 6              0               1               0              1
##   SOP_VENDOR_aaa SOP_VENDOR_jjj SOP_VENDOR_iii SOP_VENDOR_ddd
## 1              0              0              0              0
## 2              1              0              0              0
## 3              0              1              0              0
## 4              0              0              0              0
## 5              0              0              1              0
## 6              0              0              0              0
##   SOP_VENDOR_ccc SOP_VENDOR_kkk SOP_VENDOR_hhh SOP_VENDOR_bbb
## 1              0              0              0              0
## 2              0              0              0              0
## 3              0              0              0              0
## 4              0              0              0              0
## 5              0              0              0              0
## 6              0              0              0              0
##   SOP_VENDOR_ggg SOP_VENDOR_fff
## 1              0              0
## 2              0              0
## 3              0              0
## 4              0              0
## 5              0              0
## 6              0              0
```

Model 1: Logistic Regression

Starting with a simple approach due to binary response variable.

Filling NAs with column medians.

Stepping through to find lowest AIC.

```r
# create copy
df1 <- df_stats

# impute NAs in dataframe with column medians
for(col in names(df1)) {
  # feature columns only (they start with "F" and the other digits are numeric)
  if((substring(col,1,1) == "F") && !is.na(as.numeric(substring(col,2)))) {
    df1[is.na(df1[,col]), col] <- median(df1[,col], na.rm = TRUE)
  }
}

# preview dataframe
printWideDataFrame(df1, 20)
```

```
##   PASS_FAIL      F0      F1       F2        F3     F4  F5       F6     F7
## 1         1 3030.93 2564.00 2187.733 1411.1265 1.3602 100  97.6133 0.1242
## 2         1 3095.78 2465.14 2230.422 1463.6606 0.8294 100 102.3433 0.1247
## 3         0 2932.61 2559.94 2186.411 1698.0172 1.5102 100  95.4878 0.1241
## 4         1 2988.72 2479.90 2199.033  909.7926 1.3204 100 104.2367 0.1217
## 5         1 3032.24 2502.87 2233.367 1326.5200 1.5334 100 100.3967 0.1235
## 6         1 2946.25 2432.84 2233.367 1326.5200 1.5334 100 100.3967 0.1235
##       F8      F9     F10    F11      F12 F13     F14      F15     F16
## 1 1.5005  0.0162 -0.0034 0.9455 202.4396   0  7.9558 414.8710 10.0433
## 2 1.4966 -0.0005 -0.0148 0.9627 200.5470   0 10.1548 414.7347  9.2599
## 3 1.4436  0.0041  0.0013 0.9615 202.0179   0  9.5157 416.7075  9.3144
## 4 1.4882 -0.0124 -0.0033 0.9629 201.8482   0  9.6052 422.2894  9.6924
## 5 1.5031 -0.0031 -0.0072 0.9569 201.9424   0 10.5661 420.5925 10.3387
## 6 1.5287  0.0167  0.0055 0.9699 200.4720   0  8.6617 414.2426  9.2441
##      F17      F18 SIL_VENDOR_eee SIL_VENDOR_hhh SIL_VENDOR_aaa
## 1 0.9680 192.3963              0              0              0
## 2 0.9701 191.2872              0              0              0
## 3 0.9674 192.7035              1              0              0
## 4 0.9687 192.1557              0              1              0
## 5 0.9735 191.6037              0              0              1
## 6 0.9747 191.2280              0              1              0
##   SIL_VENDOR_ggg SIL_VENDOR_bbb SIL_VENDOR_ccc SIL_VENDOR_iii
## 1              0              0              0              0
## 2              0              0              0              0
## 3              0              0              0              0
## 4              0              0              0              0
## 5              0              0              0              0
## 6              0              0              0              0
##   SIL_VENDOR_fff ADHS_VENDOR_bbb ADHS_VENDOR_aaa SOP_VENDOR_eee
## 1              0               1               0              1
## 2              0               0               1              0
## 3              0               0               1              0
## 4              0               0               1              1
## 5              0               1               0              0
## 6              0               1               0              1
##   SOP_VENDOR_aaa SOP_VENDOR_jjj SOP_VENDOR_iii SOP_VENDOR_ddd
## 1              0              0              0              0
## 2              1              0              0              0
## 3              0              1              0              0
## 4              0              0              0              0
```

```
## 5              0              0              1              0
## 6              0              0              0              0
##   SOP_VENDOR_ccc SOP_VENDOR_kkk SOP_VENDOR_hhh SOP_VENDOR_bbb
## 1              0              0              0              0
## 2              0              0              0              0
## 3              0              0              0              0
## 4              0              0              0              0
## 5              0              0              0              0
## 6              0              0              0              0
##   SOP_VENDOR_ggg SOP_VENDOR_fff
## 1              0              0
## 2              0              0
## 3              0              0
## 4              0              0
## 5              0              0
## 6              0              0
```

```
# # initialize models
# m1_full = glm(PASS_FAIL ~ ., data=df1, family=binomial(), control = list(maxit = 50))
# m1_null = glm(PASS_FAIL ~ 1, data=df1, family=binomial(), control = list(maxit = 50))
#
# # down-select variables
# # m1_bwd = step(m1_full, direction="backward"), backward not a good choice for high dimensionality pr
# m1_fwd = step(m1_null, scope=list(lower=m1_null, upper=m1_full), direction="forward")
# m1_both = step(m1_null, scope = list(upper=m1_full), direction="both")
#
# # compare methods
# if(m1_fwd$aic<m1_both$aic){
#   print("Step forward selection chosen")
#   m1_varModel = m1_fwd
# }else{
#   print("Step both selection chosen")
#   m1_varModel = m1_both
# }
# m1_formula <- m1_varModel$formula
# m1_formula

# # BOTH SELECTED, TODO
m1_formula <- as.formula(
"PASS_FAIL ~ SIL_VENDOR_eee + F103 + F59 + F21 + F73 + F428 +
F569 + F64 + F75 + F129 + F433 + F365 + F9 + F443 + F473 +
F500 + F368 + F488 + SOP_VENDOR_ggg + F411 + F476 + F38 +
F87 + F104 + F484 + F349 + F84 + F72 + F56 + F554 + F131 +
F511 + F545 + F470 + F410 + F419 + F418 + F32 + SIL_VENDOR_ccc +
SOP_VENDOR_aaa + F320 + F66 + F321 + F94 + F132 + F575"
)

m1_base = glm(m1_formula, data=df1, family=binomial(), control = list(maxit = 50))
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
summary(m1_base)
```

```
##
## Call:
```

```
## glm(formula = m1_formula, family = binomial(), data = df1, control = list(maxit = 50))
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
##  -8.49    0.00    0.00    0.00    8.49
##
## Coefficients:
##                   Estimate Std. Error    z value Pr(>|z|)
## (Intercept)     1.118e+17  9.574e+08  116800273   <2e-16 ***
## SIL_VENDOR_eee -1.860e+15  4.823e+06 -385570582   <2e-16 ***
## F103           -6.645e+15  6.460e+08  -10285437   <2e-16 ***
## F59            -5.886e+13  2.506e+05 -234897325   <2e-16 ***
## F21            -3.362e+11  2.914e+03 -115374448   <2e-16 ***
## F73             2.029e+13  3.648e+05   55628443   <2e-16 ***
## F428            1.586e+13  1.654e+05   95899494   <2e-16 ***
## F569           -2.292e+13  1.853e+05 -123660497   <2e-16 ***
## F64            -7.830e+13  4.308e+05 -181743602   <2e-16 ***
## F75            -1.049e+16  8.162e+07 -128475389   <2e-16 ***
## F129           -1.745e+14  1.553e+06 -112342200   <2e-16 ***
## F433           -5.630e+11  7.974e+03  -70601180   <2e-16 ***
## F365           -8.186e+16  8.225e+08  -99530315   <2e-16 ***
## F9              1.254e+16  1.150e+08  109020153   <2e-16 ***
## F443           -1.769e+15  1.240e+07 -142638391   <2e-16 ***
## F473           -6.100e+12  8.710e+04  -70034373   <2e-16 ***
## F500           -4.088e+11  5.341e+03  -76547229   <2e-16 ***
## F368            1.267e+17  1.051e+09  120583482   <2e-16 ***
## F488            6.391e+11  6.916e+03   92421264   <2e-16 ***
## SOP_VENDOR_ggg  1.388e+14  6.077e+06   22839541   <2e-16 ***
## F411           -1.353e+14  3.194e+06  -42363755   <2e-16 ***
## F476            2.204e+12  1.388e+05   15876370   <2e-16 ***
## F38             1.098e+14  4.245e+06   25853822   <2e-16 ***
## F87            -7.567e+15  1.347e+08  -56161019   <2e-16 ***
## F104            1.794e+17  2.053e+09   87379983   <2e-16 ***
## F484            3.712e+11  8.278e+03   44844919   <2e-16 ***
## F349           -9.179e+15  1.631e+08  -56282408   <2e-16 ***
## F84             2.020e+16  3.411e+08   59217406   <2e-16 ***
## F72            -6.039e+12  3.585e+05  -16846010   <2e-16 ***
## F56            -4.287e+16  2.950e+08 -145330525   <2e-16 ***
## F554           -2.064e+14  3.163e+06  -65255420   <2e-16 ***
## F131           -6.563e+16  7.862e+08  -83486519   <2e-16 ***
## F511           -2.169e+11  5.224e+03  -41515405   <2e-16 ***
## F545            5.685e+13  1.397e+06   40683253   <2e-16 ***
## F470            5.863e+13  4.735e+05  123817519   <2e-16 ***
## F410            2.752e+13  8.189e+05   33607936   <2e-16 ***
## F419           -1.676e+11  5.271e+03  -31803275   <2e-16 ***
## F418            1.247e+11  5.980e+03   20843840   <2e-16 ***
## F32            -8.543e+13  8.762e+05  -97495392   <2e-16 ***
## SIL_VENDOR_ccc -8.954e+14  5.923e+06 -151161347   <2e-16 ***
## SOP_VENDOR_aaa -1.246e+14  5.819e+06  -21415935   <2e-16 ***
## F320           -3.402e+15  7.863e+07  -43265617   <2e-16 ***
## F66            -1.821e+13  2.287e+05  -79618705   <2e-16 ***
## F321            7.536e+13  1.190e+06   63318441   <2e-16 ***
## F94             7.806e+17  1.007e+10   77480257   <2e-16 ***
## F132            1.846e+15  3.585e+07   51494879   <2e-16 ***
```

```
## F575            9.544e+14  2.577e+07   37037828   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance:  765.15  on 1566  degrees of freedom
## Residual deviance: 5406.55  on 1520  degrees of freedom
## AIC: 5500.5
##
## Number of Fisher Scoring iterations: 46
```

Model 2: Extremely Random Trees

Handles NAs gracefully Handles noisey data and high dimensionality

```
# create copy
df2 <- df_stats

# preview dataframe
printWideDataFrame(df2, 20)
```

```
##   PASS_FAIL      F0      F1       F2        F3     F4  F5       F6     F7
## 1         1 3030.93 2564.00 2187.733 1411.1265 1.3602 100  97.6133 0.1242
## 2         1 3095.78 2465.14 2230.422 1463.6606 0.8294 100 102.3433 0.1247
## 3         0 2932.61 2559.94 2186.411 1698.0172 1.5102 100  95.4878 0.1241
## 4         1 2988.72 2479.90 2199.033  909.7926 1.3204 100 104.2367 0.1217
## 5         1 3032.24 2502.87 2233.367 1326.5200 1.5334 100 100.3967 0.1235
## 6         1 2946.25 2432.84 2233.367 1326.5200 1.5334 100 100.3967 0.1235
##       F8      F9     F10    F11      F12 F13     F14      F15     F16
## 1 1.5005  0.0162 -0.0034 0.9455 202.4396   0  7.9558 414.8710 10.0433
## 2 1.4966 -0.0005 -0.0148 0.9627 200.5470   0 10.1548 414.7347  9.2599
## 3 1.4436  0.0041  0.0013 0.9615 202.0179   0  9.5157 416.7075  9.3144
## 4 1.4882 -0.0124 -0.0033 0.9629 201.8482   0  9.6052 422.2894  9.6924
## 5 1.5031 -0.0031 -0.0072 0.9569 201.9424   0 10.5661 420.5925 10.3387
## 6 1.5287  0.0167  0.0055 0.9699 200.4720   0  8.6617 414.2426  9.2441
##      F17      F18 SIL_VENDOR_eee SIL_VENDOR_hhh SIL_VENDOR_aaa
## 1 0.9680 192.3963              0              0              0
## 2 0.9701 191.2872              0              0              0
## 3 0.9674 192.7035              1              0              0
## 4 0.9687 192.1557              0              1              0
## 5 0.9735 191.6037              0              0              1
## 6 0.9747 191.2280              0              1              0
##   SIL_VENDOR_ggg SIL_VENDOR_bbb SIL_VENDOR_ccc SIL_VENDOR_iii
## 1              0              0              0              0
## 2              0              0              0              0
## 3              0              0              0              0
## 4              0              0              0              0
## 5              0              0              0              0
## 6              0              0              0              0
##   SIL_VENDOR_fff ADHS_VENDOR_bbb ADHS_VENDOR_aaa SOP_VENDOR_eee
## 1              0               1               0              1
## 2              0               0               1              0
## 3              0               0               1              0
## 4              0               0               1              1
```

```
## 5              0               1               0              0
## 6              0               1               0              1
##   SOP_VENDOR_aaa SOP_VENDOR_jjj SOP_VENDOR_iii SOP_VENDOR_ddd
## 1              0              0              0              0
## 2              1              0              0              0
## 3              0              1              0              0
## 4              0              0              0              0
## 5              0              0              1              0
## 6              0              0              0              0
##   SOP_VENDOR_ccc SOP_VENDOR_kkk SOP_VENDOR_hhh SOP_VENDOR_bbb
## 1              0              0              0              0
## 2              0              0              0              0
## 3              0              0              0              0
## 4              0              0              0              0
## 5              0              0              0              0
## 6              0              0              0              0
##   SOP_VENDOR_ggg SOP_VENDOR_fff
## 1              0              0
## 2              0              0
## 3              0              0
## 4              0              0
## 5              0              0
## 6              0              0
```

```
# run model and summarize
m2_base = extraTrees(df2 %>% select(-PASS_FAIL),df2$PASS_FAIL,numRandomCuts=1,na.action="fuse")
m2_base
```

```
## ExtraTrees:
##  - # of trees: 500
##  - node size:  5
##  - # of dim:   622
##  - # of tries: 207
##  - type:       numeric (regression)
##  - multi-task: no
```

```
# extraTrees does not have a variable importance function
```

Cross Validation

Run 5 fold cross validation for both models, generate all ROC graphs, and export to pdf.

```
# create copy
df <- df_stats

n = 5  # number of folds
df = df[sample(nrow(df)),]  # Randomly shuffle the data
folds = cut(seq(1,nrow(df)),breaks=n,labels=FALSE)  # Create 5 equally size folds

# create empty matrix for accuracy and precision
accuracy = matrix(data=NA,nrow=n,ncol=2)
precision = matrix(data=NA,nrow=n,ncol=2)
cutoff = 0.50

pdf(file='./docs/Rplots.pdf',width=10,height=7.5)  # begin pdf writer
```

```
# Perform 5 fold cross validation
for(i in 1:n){
  # Segment the data by fold using the which() function
  testIndexes = which(folds==i,arr.ind=TRUE)
  testData = df[testIndexes, ]
  trainData = df[-testIndexes, ]

  # model 1: logistic regression
  m1 = glm(m1_formula,data=trainData,family='binomial',control=list(maxit=50))
  p1 = predict(m1,newdata=testData,type='response')
  pr1 = prediction(p1,testData$PASS_FAIL)
  prf1 = performance(pr1,measure="tpr",x.measure="fpr")
  prec1 = performance(pr1,measure="prec")
  acc1 = performance(pr1,measure="acc")
  auc1 = performance(pr1,measure="auc")

  # model 2: extremely random forest
  m2 = extraTrees(trainData %>% select(-PASS_FAIL),trainData$PASS_FAIL,numRandomCuts=1,na.action="fuse")
  p2 = predict(m2,testData %>% select(-PASS_FAIL))
  pr2 = prediction(p2,testData$PASS_FAIL)
  prf2 = performance(pr2,measure="tpr",x.measure="fpr")
  prec2 = performance(pr2,measure="prec")
  acc2 = performance(pr2,measure="acc")
  auc2 = performance(pr2,measure="auc")

  # graph results
  par(pty="s")
  plot(prf1,main=paste('ROC: Fold ',i,sep=''),xaxs='i',yaxs='i',asp=1)
  lines(prf2@x.values[[1]],prf2@y.values[[1]],col='red')
  abline(a=0,b=1,lty=2)
  legend('bottomright',
         c(paste('Model 1 | AUC=',format(round(auc1@y.values[[1]],3),3),sep=''),
           paste('Model 2 | AUC=',format(round(auc2@y.values[[1]],3),3),sep='')),
         col=c('black','red'),lty=c(1,1))

  par(pty="m")
  plot(prec1,main=paste('Precision: Fold ',i,sep=''),ylim=c(0.4,1))
  lines(prec2@x.values[[1]],prec2@y.values[[1]],col='red')
  abline(v=0.5,lty=2)
  legend('topleft',c('Model 1','Model 2'),col=c('black','red'),lty=c(1,1))

  plot(acc1,main=paste('Accuracy: Fold ',i,sep=''),ylim=c(0.4,1))
  lines(acc2@x.values[[1]],acc2@y.values[[1]],col='red')
  abline(v=0.5,lty=2)
  legend('topleft',c('Model 1','Model 2'),col=c('black','red'),lty=c(1,1))

  accuracy[i,1] = acc1@y.values[[1]][max(which(acc1@x.values[[1]]>=cutoff))]
  accuracy[i,2] = acc2@y.values[[1]][max(which(acc2@x.values[[1]]>=cutoff))]

  precision[i,1] = prec1@y.values[[1]][max(which(prec1@x.values[[1]]>=cutoff))]
  precision[i,2] = prec2@y.values[[1]][max(which(prec2@x.values[[1]]>=cutoff))]

}
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
dev.off()  # close pdf writer
```

```
## pdf
##   2
```

Conclusion

```
# defined as null hypothesis: m1-m2=0
accuracy_test = t.test(accuracy[,1],accuracy[,2],conf.level=0.95,paired=T)
precision_test = t.test(precision[,1],precision[,2],conf.level=0.95,paired=T)

accuracy
```

```
##           [,1]      [,2]
## [1,] 0.9574468 0.9522184
## [2,] 0.9278351 0.9503817
## [3,] 0.9203540 0.9247312
## [4,] 0.9285714 0.9522059
## [5,] 0.9259259 0.9343629
```

```
accuracy_test
```

```
##
##  Paired t-test
##
## data:  accuracy[, 1] and accuracy[, 2]
## t = -1.9528, df = 4, p-value = 0.1226
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.026042000  0.004535236
## sample estimates:
## mean of the differences
##             -0.01075338
```

```
if(accuracy_test$p.value>0.05){
  print("Model 1 and Model 2 accuracies are not significantly different.")
}else if(mean(accuracy[,1])>mean(accuracy[,2])){
  print("Model 1 is statistically more accurate than Model 2.")
}else{
  print("Model 2 is statistically more accurate than Model 1.")
}
```

```
## [1] "Model 1 and Model 2 accuracies are not significantly different."
```

```
precision
```

```
##           [,1]      [,2]
```

```
## [1,] 0.9887640 0.9744526
## [2,] 0.9764706 0.9593496
## [3,] 0.9800000 0.9427481
## [4,] 0.9438202 0.9686275
## [5,] 0.9861111 0.9558233
```

```
precision_test
```

```
##
##  Paired t-test
##
## data:  precision[, 1] and precision[, 2]
## t = 1.3777, df = 4, p-value = 0.2404
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.01505936  0.04472535
## sample estimates:
## mean of the differences
##                0.014833
```

```
if(precision_test$p.value>0.05){
  print("Model 1 and Model 2 precisions are not significantly different.")
}else if(mean(precision[,1])>mean(precision[,2])){
  print("Model 1 is statistically more precise than Model 2.")
}else{
  print("Model 2 is statistically more precise than Model 1.")
}
```

```
## [1] "Model 1 and Model 2 precisions are not significantly different."
```