People Analytics in R - Job Classification 'Revisited'

Lyndon Sundmark, MBA

2016-04-30

Introduction

About a year back I posted a couple of blog articles on Data Driven Job Classification, showing a variety of tools to do this including R, Azure machine Learning and a few others. Its purpose was to encourage HR folks to start thinking about HR's future being more in the use of HR analytics.

https://www.linkedin.com/pulse/data-driven-job-classification-lyndon-sundmark-mba?trk=mp-author-card

As the terminology in People Analytics is continuing to unfold and evolve, it is becoming increasing apparent that one of the best ways to understand People Analytics -both stringently and widely at the same time is to see it as:

- "Data Driven" HR and HR Decision Making
- When Data Science as a process and a field meets the HR context

Seeing it this way helps prevent us from unnecessarily limiting the contribution that it can make, and yet at the same time help prevent proliferation of terminology to the point of meaninglessness. 'Data driven' must be how we conduct HR Management and decision making in the future. Data Science contributes to that goal by itself being a 'data driven' process.

Another way to not artificially limit the application of People Analytics, is to remind ourselves of the potential scope of the relevant HR context. People Analytics can be applied to:

- **information on what is happening to employees in the organization over time.** Typically this is thought of as HR metrics/demographics. Many still see this as the 'extent' of HR analytics.

- **how well the HR department conducts its business and operations.** These are metrics related to process improvement. Right now this is more typically thought of in the 'quality improvement realm. But it's still data driven decision making
- **'direct' embedding of statistical algorithms in our HR methodologies- how we actually 'do' HR.** There is huge application of People Analytics here. HR needs to get out of its traditional non-analytic methodologies paradigm where data science can be brought to bear.

People Analytics in R- Job Classification is an example of embedding statistical algorithms in our HR methodologies.

With that in mind, I thought I would do a **'revisit'** of job classification as an example of People Analytics in R. By revisit, I mean let's restrict the tool to R, and let's apply the data science process/framework to it. This would put into into a format similar to my last 2 blog articles. Additionally, in this article more time will be spent on describing how we generate the data in the first place.

The files for this currrent blog article can be found in this location:

https://onedrive.live.com/redir?resid=4EF2CCBEDB98D0F5!6433&authkey=!ABv-gHg5jVluYpc&ithint=folder%2cxlsx

Applying the Data Science Process

Let's remind ourselves of those steps again.

The Data Science Steps

1. *Define A Goal*
2. *Collect And Manage Data*
3. *Build the Model*
4. *Evaluate And Critique Model*
5. *Present Results and Document*
6. *Deploy Model*

##1. Define A Goal

Ok - what is our goal here? Perhaps a quick primer on job classification would be in order to answer the question.

A Quick Job Classification Primer

Job Classification is at the heart of compensation and salary administration in HR. We desire to pay our employees fairly- both from an external and internal perspective. Salary surveys help us out on the external side. But job classification helps us out on the internal picture. We try to understand the similarities and differences between jobs in the organization.

At a base level, this process starts with documenting job descriptions. We document tasks, knowleges, and skills needed to complete the work of the organization as organized within our jobs. Usually as a result of job descriptions being documented, we design broader categories that the job descriptions fall into. We call these job classifications. They attempt to categorize like with like and distinguish between job descriptions that are different. We often are concerned with the characteristics of how responsibility, accountability, supervision, education level, experience etc vary between these classifications. And to tie into our compensations systems, we often have paygrades assigned to the classification.

So what is our goal in Job Classification? To properly categorize 'job descriptions' into appropriate job classifications. When job classifications are written they are, by definition, of a 'known', 'intended' population. Job descriptions until they are categorized are outside of that population. Once proper categorization is made of a job description, it becomes part of that 'known' population. It is 'unknown' by the population until then. When all the job descriptions are categorized into job classifications, both the job descriptions and the job classifications are part of the known population. Any new job description written in the future is unknown until it is classified as well.

Why is this significant to People Analytics? The above process indicates that we are trying to classify something, or find the right category, that we **don't know** based on how it compares to a population we do **know**. HR job classification is the context here. It just so happens that in data science and statistics there are all sorts of algorithms designed to create categories or find the best fit among known categories.

For decades we have had job classification as a process in HR, and classification algorithms in statistics- but HR,in most organizations, is not recognizing this and the potential contribution it could/can make .

So in People Analytics in R -Job Classification:

*** our primary goal is to classify job descriptions into job classifications using the power of statistical algorithms to assist in prediction of best fit. * our secondary goal might be to help improve the design of our job classification system/framework.**

2.Collect And Manage Data

For purposes of this application of People Analytics, this step in the data science process will take the longest initially. This is because in almost every organization, the existing job classifications or categories, and the job descriptions themselves are not typically represented in numerical format suitable for statistical analysis. Sometimes, that which we are predicting- the pay grade is numeric because point methods are used in evaluation and different paygrades have different point ranges. But more often the job descriptions are narrative as are the job classification specs or summaries. For this blog article, we will assume that and delineate the steps required.

Collecting The Data

The following are typical steps:

1. Gather together the entire set of narrative, written job classification specifications.
2. Review all of them to determine what the common denominators are- what the organization is paying attention to , to differentiate them from each other.
3. For each of the common denominators, pay attention to descriptions of how much of that common denominator exists in each narrative, writing down the phrases that are used.
4. For each common denominator, develop an ordinal scale which assigns numbers and places them in a 'less to more' order
5. Create a datafile where each record (row) is one job classification, and where each column is either a common denominator or the job classification identifier or paygrade.
6. Code each job classification narrative into the datafile recording their common denominator information and other pertinent categorical information.

Gather together the entire set of narrative, written job classification specifications.

This initially represents the 'total' population of what will be a 'known' population. Ones that by definition represent the prescribed intended categories and levels of paygrades. These are going to be used to compare an 'unknown' population- unclassified job descriptions, to determine best fit. But before this can happen, we should have confidence that the job classifications themselves are well designed- since they will be the standard against which all job descriptions will be compared.

Review all of them to determine what the common denominators are

Technically speaking, anything that appears in the narrative could be considered a feature that is a common denominator including the tasks, knowledges described. But few organizations have that level of automation in their job descriptions. So generally broader features are used to describe common denominators. Often they may include the following:

- Education Level
- Experience
- Organizational Impact
- Problem Solving
- Supervision Received
- Contact Level
- Financial Budget Responsibility

To be a common denominator they need to be mentioned or discernable in every job classification specification

Pay attention to the descriptions of how much of that common denominator exists in each narrative

For each of the above common denominators (if these are ones you use), go through each narrative identify where the common denominator is mentioned and write down the words used to describe how much of it exists. Go through you entire set of job classification specs and tabulate these for each common denominator and each class spec.

For each common denominator, develop an ordinal scale

Ordinal means in order. You order the descriptions from less than to more than. Then apply a numerical indicator to it. 0 might mean it doesnt exist in any significant way, 1 might mean something at a low or introductory level, higher numbers meaning more of it. The scale should have as many numbers as distinguishable descriptions.(You may have to merge or collapse descriptions if it's impossible to distinguish order)

Create a datafile

This might be a spreadsheet.

each record(row) will be one job classification, and each column will be either a common denominator or the job classification identifier or paygrade or other categorical information.

Code each job classification narrative into the datafile

Record their common denominator information and other pertinent categorical or identifying information. At the end of this task you will have as many records as you have written job classification specs.

At the end of this effort you will have something that looks like the data found at the following link:

https://onedrive.live.com/redir?resid=4EF2CCBEDB98D0F5!6435&authkey=!AL37Wt0sVLrsUYA&ithint=file%2ctxt

###Manage The Data

In this step we check the data for errors, organize the data for model building, and take an initial look at what the data is telling us.

Check the data for errors

```
library(readr)
#MYdataset <- read.csv("jobclassinfo2.txt")
MYdataset<- read_csv("jobclassinfo2.txt",
    col_types = cols(PG = col_factor(levels = c("PG01",
        "PG02", "PG03", "PG04", "PG05", "PG06",
        "PG07", "PG08", "PG09", "PG10"))))

str(MYdataset,width=80,strict.width ="wrap")
```

```
spc_tbl_ [66 x 14] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
$ ID : num [1:66] 1 2 3 4 5 6 7 8 9 10 ...
$ JobFamily : num [1:66] 1 1 1 1 2 2 2 2 2 3 ...
$ JobFamilyDescription: chr [1:66] "Accounting And Finance" "Accounting And
   Finance" "Accounting And Finance" "Accounting And Finance" ...
$ JobClass : num [1:66] 1 2 3 4 5 6 7 8 9 10 ...
$ JobClassDescription : chr [1:66] "Accountant I" "Accountant II" "Accountant
   III" "Accountant IV" ...
$ PayGrade : num [1:66] 5 6 8 10 1 2 3 4 5 4 ...
$ EducationLevel : num [1:66] 3 4 4 5 1 1 1 4 4 2 ...
$ Experience : num [1:66] 1 1 2 5 0 1 2 0 0 0 ...
$ OrgImpact : num [1:66] 3 5 6 6 1 1 1 1 4 1 ...
$ ProblemSolving : num [1:66] 3 4 5 6 1 1 2 2 3 4 ...
$ Supervision : num [1:66] 4 5 6 7 1 1 1 1 5 1 ...
$ ContactLevel : num [1:66] 3 7 7 8 1 2 3 3 7 1 ...
$ FinancialBudget : num [1:66] 5 7 10 11 1 3 3 5 7 2 ...
$ PG : Factor w/ 10 levels "PG01","PG02",..: 5 6 8 10 1 2 3 4 5 4 ...
- attr(*, "spec")=
.. cols(
..   ID = col_double(),
..   JobFamily = col_double(),
..   JobFamilyDescription = col_character(),
..   JobClass = col_double(),
```

```
..   JobClassDescription = col_character(),
..   PayGrade = col_double(),
..   EducationLevel = col_double(),
..   Experience = col_double(),
..   OrgImpact = col_double(),
..   ProblemSolving = col_double(),
..   Supervision = col_double(),
..   ContactLevel = col_double(),
..   FinancialBudget = col_double(),
..   PG = col_factor(levels = c("PG01", "PG02", "PG03", "PG04", "PG05", "PG06",
    "PG07", "PG08",
..   "PG09", "PG10"), ordered = FALSE, include_na = FALSE)
.. )
- attr(*, "problems")=<externalptr>
```

```r
summary(MYdataset)
```

```
       ID           JobFamily       JobFamilyDescription    JobClass
 Min.   : 1.00   Min.   : 1.000   Length:66            Min.   : 1.00
 1st Qu.:17.25   1st Qu.: 4.000   Class :character     1st Qu.:17.25
 Median :33.50   Median : 7.000   Mode  :character     Median :33.50
 Mean   :33.50   Mean   : 7.606                        Mean   :33.50
 3rd Qu.:49.75   3rd Qu.:11.000                        3rd Qu.:49.75
 Max.   :66.00   Max.   :15.000                        Max.   :66.00

 JobClassDescription    PayGrade       EducationLevel    Experience
 Length:66            Min.   : 1.000   Min.   :1.000   Min.   : 0.000
 Class :character     1st Qu.: 4.000   1st Qu.:2.000   1st Qu.: 0.000
 Mode  :character     Median : 5.000   Median :4.000   Median : 1.000
                      Mean   : 5.697   Mean   :3.167   Mean   : 1.758
                      3rd Qu.: 8.000   3rd Qu.:4.000   3rd Qu.: 2.750
                      Max.   :10.000   Max.   :6.000   Max.   :10.000

   OrgImpact     ProblemSolving   Supervision     ContactLevel
 Min.   :1.000   Min.   :1.000   Min.   :1.000   Min.   :1.000
 1st Qu.:2.000   1st Qu.:3.000   1st Qu.:1.000   1st Qu.:3.000
 Median :3.000   Median :4.000   Median :4.000   Median :6.000
 Mean   :3.348   Mean   :3.606   Mean   :3.864   Mean   :4.758
 3rd Qu.:4.000   3rd Qu.:5.000   3rd Qu.:5.750   3rd Qu.:7.000
 Max.   :6.000   Max.   :6.000   Max.   :7.000   Max.   :8.000
```

```
 FinancialBudget         PG
 Min.   : 1.000   PG05   :15
 1st Qu.: 2.000   PG03   : 7
 Median : 5.000   PG04   : 7
 Mean   : 5.303   PG06   : 7
 3rd Qu.: 7.750   PG08   : 7
 Max.   :11.000   PG09   : 6
                  (Other):17
```

On the surface there doesn't seem to be any issues with data. This gives a summary of the layout of the data and the likely values we can expect. PG is the category we will predict. It's a categorical representation of the numeric paygrade. Education level through Financial Budgeting Responsibility will be the independent variables/measures we will use to predict. The other columns in file will be ignored.

Organize the data

Lets narrow down the information to just the data used in the model.

```
MYnobs <- nrow(MYdataset) # 66 observations
MYsample <- MYtrain <- sample(nrow(MYdataset), 0.7*MYnobs) # 46 observations
MYvalidate <- sample(setdiff(seq_len(nrow(MYdataset)), MYtrain), 0.14*MYnobs) # 9 observat
MYtest <- setdiff(setdiff(seq_len(nrow(MYdataset)), MYtrain), MYvalidate) # 11 observation



#============================================================
# Rattle timestamp: 2016-04-27 12:43:48 x86_64-w64-mingw32

# Note the user selections.

# The following variable selections have been noted.

MYinput <- c("EducationLevel", "Experience", "OrgImpact", "ProblemSolving",
     "Supervision", "ContactLevel", "FinancialBudget")

MYnumeric <- c("EducationLevel", "Experience", "OrgImpact", "ProblemSolving",
     "Supervision", "ContactLevel", "FinancialBudget")

MYcategoric <- NULL

MYtarget  <- "PG"
```

```
MYrisk    <- NULL
MYident   <- "ID"
MYignore  <- c("JobFamily", "JobFamilyDescription", "JobClass", "JobClassDescription", "Pa
MYweights <- NULL
```

We are predominantly interested in MYinput and MYtarget because they represent the predictors and what is to be predicted respectively. You will notice for the time being that we are not partitioning the data. This will be elaborated upon in model building.

What the data is initially telling us

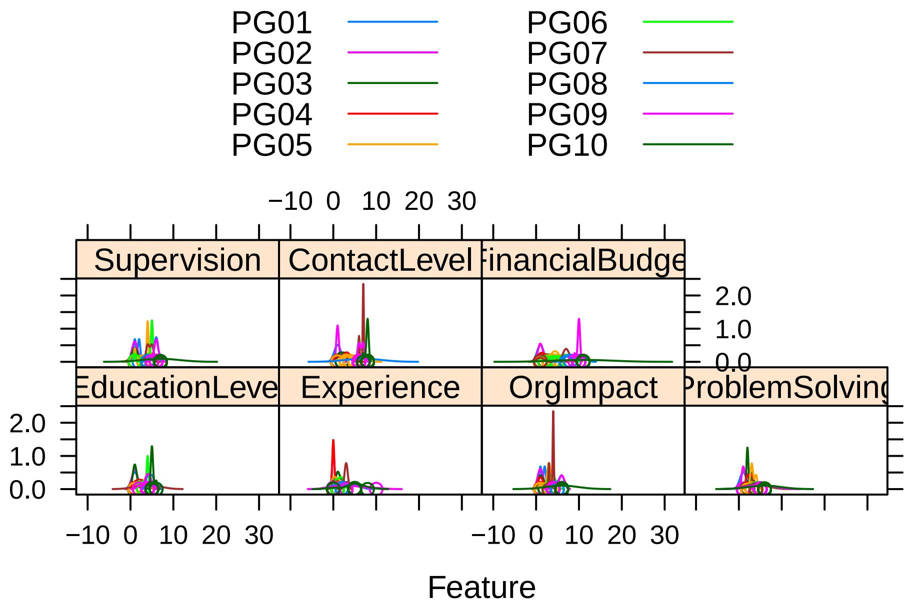
Lets use the caret library again for some graphical representations of this data.

```
library(caret)
```

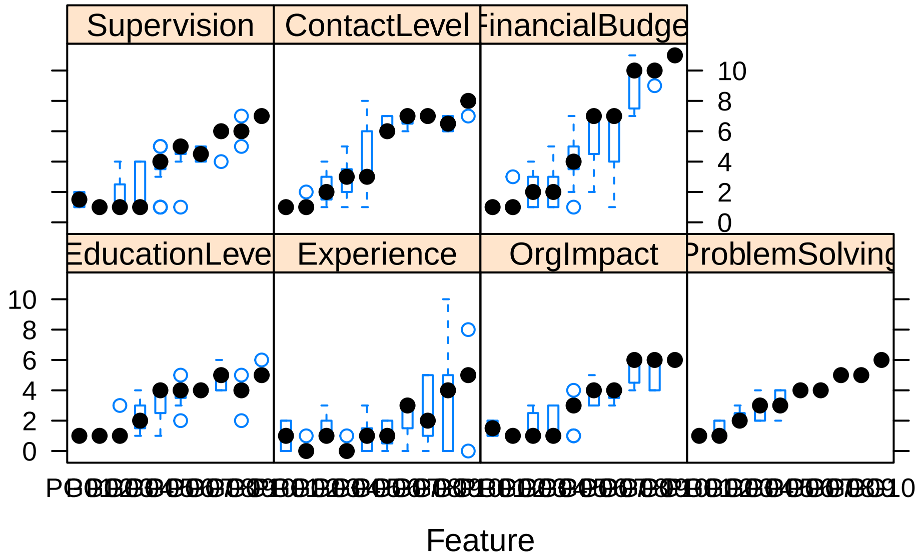
```
Loading required package: ggplot2
```

```
Loading required package: lattice
```

```
featurePlot(x=MYdataset[,7:13],y=MYdataset$PG,plot="density",auto.key = list(columns = 2))
```



```
featurePlot(x=MYdataset[,7:13],y=MYdataset$PG,plot="box",auto.key = list(columns = 2))
```



The first set of charts show the distribution of the independent variable values(predictors) by PG.

The second set of charts show the range of values of the predictors by PG. PG is ordered left to right in ascending order from PG1 to PG10. In each of the predictors we would expect increasing levels as we move up the paygrades and from left to right (or at least not dropping from previous paygrade).

This is the first indication by a graphic 'visual' that we 'may' have problems in the data or the interpretation of the coding of the information. Then again the coding may be accurate based on our descriptions and our assumptions false. We will probably want to recheck our coding from the job description to make sure.

3.Build The Model

Lets use the rattle library to efficiently generate the code to run the following classification algorithms against our data:

- Decision Tree
- Random Forest
- Support Vector Machines
- Linear

Decision Tree

```r
library(rattle)
```

```
Loading required package: tibble

Loading required package: bitops

Rattle: A free graphical interface for data science with R.
Version 5.5.1 Copyright (c) 2006-2021 Togaware Pty Ltd.
Type 'rattle()' to shake, rattle, and roll your data.
```

```r
#============================================================
# Rattle timestamp: 2016-04-27 12:51:16 x86_64-w64-mingw32

# Decision Tree

# The 'rpart' package provides the 'rpart' function.

library(rpart, quietly=TRUE)

# Reset the random number seed to obtain the same results each time.
#crv$seed <- 42
#set.seed(crv$seed)

# Build the Decision Tree model.

MYrpart <- rpart(PG ~ .,
    data=MYdataset[, c(MYinput, MYtarget)],
    method="class",
    parms=list(split="information"),
      control=rpart.control(minsplit=10,
           minbucket=2,
           maxdepth=10,
        usesurrogate=0,
        maxsurrogate=0))

# Generate a textual view of the Decision Tree model.

print(MYrpart)
```

```
n= 66

node), split, n, loss, yval, (yprob)
      * denotes terminal node

 1) root 66 51 PG05 (0.03 0.076 0.11 0.11 0.23 0.11 0.061 0.11 0.091 0.091)
   2) ProblemSolving< 4.5 47 32 PG05 (0.043 0.11 0.15 0.15 0.32 0.15 0.085 0 0 0)
     4) ContactLevel< 5.5 32 21 PG05 (0.062 0.16 0.22 0.22 0.34 0 0 0 0 0)
       8) EducationLevel< 1.5 15  9 PG03 (0.13 0.33 0.4 0.13 0 0 0 0 0 0)
        16) ProblemSolving< 1.5 5  2 PG02 (0.4 0.6 0 0 0 0 0 0 0 0) *
        17) ProblemSolving>=1.5 10  4 PG03 (0 0.2 0.6 0.2 0 0 0 0 0 0)
          34) Experience< 0.5 3  1 PG02 (0 0.67 0 0.33 0 0 0 0 0 0) *
          35) Experience>=0.5 7  1 PG03 (0 0 0.86 0.14 0 0 0 0 0 0) *
       9) EducationLevel>=1.5 17  6 PG05 (0 0 0.059 0.29 0.65 0 0 0 0 0)
        18) Experience< 0.5 8  3 PG04 (0 0 0 0.62 0.37 0 0 0 0 0) *
        19) Experience>=0.5 9  1 PG05 (0 0 0.11 0 0.89 0 0 0 0 0) *
     5) ContactLevel>=5.5 15  8 PG06 (0 0 0 0 0.27 0.47 0.27 0 0 0)
      10) Experience< 2.5 12  5 PG06 (0 0 0 0 0.33 0.58 0.083 0 0 0)
        20) ContactLevel>=6.5 8  4 PG05 (0 0 0 0 0.5 0.38 0.13 0 0 0) *
        21) ContactLevel< 6.5 4  0 PG06 (0 0 0 0 0 1 0 0 0 0) *
      11) Experience>=2.5 3  0 PG07 (0 0 0 0 0 0 1 0 0 0) *
   3) ProblemSolving>=4.5 19 12 PG08 (0 0 0 0 0 0 0 0.37 0.32 0.32)
     6) ProblemSolving< 5.5 13  6 PG08 (0 0 0 0 0 0 0 0.54 0.46 0)
      12) ContactLevel>=6.5 10  3 PG08 (0 0 0 0 0 0 0 0.7 0.3 0) *
      13) ContactLevel< 6.5 3  0 PG09 (0 0 0 0 0 0 0 0 1 0) *
     7) ProblemSolving>=5.5 6  0 PG10 (0 0 0 0 0 0 0 0 0 1) *
```

```r
printcp(MYrpart)
```

```
Classification tree:
rpart(formula = PG ~ ., data = MYdataset[, c(MYinput, MYtarget)],
    method = "class", parms = list(split = "information"), control = rpart.control(minsplit =
        minbucket = 2, maxdepth = 10, usesurrogate = 0, maxsurrogate = 0))

Variables actually used in tree construction:
[1] ContactLevel   EducationLevel Experience     ProblemSolving

Root node error: 51/66 = 0.77273

n= 66
```

```
        CP nsplit rel error  xerror     xstd
1 0.137255      0   1.00000 1.00000 0.066756
2 0.117647      1   0.86275 0.96078 0.069660
3 0.088235      2   0.74510 0.90196 0.073207
4 0.058824      4   0.56863 0.84314 0.075902
5 0.039216      7   0.39216 0.78431 0.077835
6 0.019608      9   0.31373 0.74510 0.078728
7 0.010000     10   0.29412 0.68627 0.079501
```

```
cat("\n")
```

```
# Time taken: 0.02 secs
```

Random Forest

```
#============================================================
# Rattle timestamp: 2016-04-27 12:51:16 x86_64-w64-mingw32

# Random Forest

# The 'randomForest' package provides the 'randomForest' function.

library(randomForest, quietly=TRUE)
```

```
randomForest 4.7-1.1

Type rfNews() to see new features/changes/bug fixes.


Attaching package: 'randomForest'

The following object is masked from 'package:rattle':

    importance

The following object is masked from 'package:ggplot2':

    margin
```

```r
# Build the Random Forest model.

#set.seed(crv$seed)
MYrf <- randomForest::randomForest(PG ~ .,
      data=MYdataset[,c(MYinput, MYtarget)],
      ntree=500,
      mtry=2,
      importance=TRUE,
      na.action=randomForest::na.roughfix,
      replace=FALSE)

# Generate textual output of 'Random Forest' model.

MYrf
```

```
Call:
 randomForest(formula = PG ~ ., data = MYdataset[, c(MYinput,      MYtarget)], ntree = 500, m
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 2

        OOB estimate of  error rate: 40.91%
Confusion matrix:
     PG01 PG02 PG03 PG04 PG05 PG06 PG07 PG08 PG09 PG10 class.error
PG01    0    2    0    0    0    0    0    0    0    0   1.0000000
PG02    0    4    1    0    0    0    0    0    0    0   0.2000000
PG03    0    0    5    1    1    0    0    0    0    0   0.2857143
PG04    0    0    1    2    4    0    0    0    0    0   0.7142857
PG05    0    0    0    4    9    2    0    0    0    0   0.4000000
PG06    0    0    0    0    1    5    1    0    0    0   0.2857143
PG07    0    0    0    0    0    3    1    0    0    0   0.7500000
PG08    0    0    0    0    0    0    0    5    2    0   0.2857143
PG09    0    0    0    0    0    0    0    3    2    1   0.6666667
PG10    0    0    0    0    0    0    0    0    0    6   0.0000000
```

```r
# List the importance of the variables.

rn <- round(randomForest::importance(MYrf), 2)
rn[order(rn[,3], decreasing=TRUE),]
```

```
                 PG01  PG02  PG03 PG04 PG05  PG06  PG07  PG08  PG09  PG10
EducationLevel   2.01 15.23 13.17 4.01 6.00  0.00  5.50  3.24 -4.07  6.53
ProblemSolving   3.80  7.11 10.86 2.59 6.54 12.05  8.33 13.54  9.97 17.95
Experience      -3.51 10.76  8.78 6.54 0.75  3.15  8.04 -4.90  6.35  1.69
Supervision     -3.40  8.75  7.19 3.67 7.25  5.53  1.98  3.75  2.20 15.90
FinancialBudget  2.25  8.30  4.39 1.77 9.77 -0.86 -2.07 -4.32 11.90 15.11
ContactLevel     2.46 12.57  4.09 1.13 3.39  9.30  3.18 11.28 -0.70 10.99
OrgImpact       -1.95 12.07  3.76 1.39 9.55  0.91  3.70 -0.32  4.79  9.92
                MeanDecreaseAccuracy MeanDecreaseGini
EducationLevel                 18.28             4.26
ProblemSolving                 23.43             6.14
Experience                     14.24             4.29
Supervision                    17.91             4.04
FinancialBudget                15.12             5.36
ContactLevel                   15.00             4.99
OrgImpact                      14.02             3.51
```

```
# Time taken: 0.06 secs
```

Support Vector Machine

```
#============================================================
# Rattle timestamp: 2016-04-27 12:51:16 x86_64-w64-mingw32

# Support vector machine.

# The 'kernlab' package provides the 'ksvm' function.

library(kernlab, quietly=TRUE)
```

```
Attaching package: 'kernlab'

The following object is masked from 'package:ggplot2':

    alpha
```

```
# Build a Support Vector Machine model.

#set.seed(crv$seed)
MYksvm <- ksvm(as.factor(PG) ~ .,
      data=MYdataset[,c(MYinput, MYtarget)],
      kernel="rbfdot",
      prob.model=TRUE)

# Generate a textual view of the SVM model.

MYksvm
```

```
Support Vector Machine object of class "ksvm"

SV type: C-svc  (classification)
 parameter : cost C = 1

Gaussian Radial Basis kernel function.
 Hyperparameter : sigma =  0.183309272040963

Number of Support Vectors : 64

Objective Function Value : -3.8147 -3.6222 -2.8441 -2.2133 -1.5242 -1.334 -1.4231 -1.5539 -1
Training error : 0.287879
Probability model included.
```

```
# Time taken: 0.43 secs
```

Linear Model

```
#============================================================
# Rattle timestamp: 2016-04-27 12:51:17 x86_64-w64-mingw32

# Regression model

# Build a multinomial model using the nnet package.

library(nnet, quietly=TRUE)
```

```r
# Summarise multinomial model using Anova from the car package.

library(car, quietly=TRUE)

# Build a Regression model.

MYglm <- multinom(PG ~ ., data=MYdataset[,c(MYinput, MYtarget)], trace=FALSE, maxit=1000)

# Generate a textual view of the Linear model.

rattle.print.summary.multinom(summary(MYglm,
                              Wald.ratios=TRUE))
```

```
Warning in sqrt(diag(vc)): NaNs produced
```

```
Call:
multinom(formula = PG ~ ., data = MYdataset[, c(MYinput, MYtarget)],
    trace = FALSE, maxit = 1000)

n=66

Coefficients:
     (Intercept) EducationLevel  Experience  OrgImpact ProblemSolving
PG02   31570.140     -29546.246 -10054.5402 -31742.949       22887.11
PG03    5173.408     -13991.742  17611.5597 -23434.513       27580.11
PG04  -12639.313       3742.145  -5763.6011 -11968.730       18528.68
PG05  -29223.399       5164.052   1451.4069 -12579.427       21540.91
PG06  -64936.850       5381.517   1332.3066 -10153.015       27846.30
PG07  -55186.714       5379.689   1334.2282 -10155.643       25405.97
PG08 -124914.252       6197.617    572.4667 -13900.558       42485.72
PG09  -97759.116     -14428.508   8698.6998  -8681.908      -21901.43
PG10 -201459.455       9187.762    765.1991 -15483.566       54919.25
     Supervision ContactLevel FinancialBudget
PG02   -3826.376   -3466.3364       14125.347
PG03   -7979.596    4054.9295       -2131.931
PG04   -1202.967   -4060.4297        6156.244
PG05   -3137.087    -986.4325        5756.414
PG06   -3540.354    -746.1415        6259.916
PG07   -3539.699    -742.8932        6259.885
PG08   -1862.503   -1725.1769        7267.505
PG09   21068.207  -12739.5580       34119.536
```

```
PG10   -2773.094    1152.7658        5960.684

Std. Errors:
      (Intercept) EducationLevel   Experience    OrgImpact ProblemSolving
PG02 1.749631e-01   1.749631e-01 0.000000e+00 1.749631e-01   1.749631e-01
PG03 0.000000e+00   0.000000e+00 0.000000e+00 0.000000e+00   0.000000e+00
PG04 0.000000e+00            NaN          NaN 1.776635e-14   7.258525e-15
PG05 4.576029e-16   2.976903e-15 2.398744e-16 1.189550e-16   1.716121e-16
PG06 3.859338e-01   1.397271e+00 6.650459e-01 1.366421e+00   1.543735e+00
PG07 3.859338e-01   1.397271e+00 6.650459e-01 1.366421e+00   1.543735e+00
PG08 0.000000e+00   0.000000e+00 0.000000e+00 0.000000e+00   0.000000e+00
PG09 0.000000e+00   0.000000e+00 0.000000e+00 0.000000e+00   0.000000e+00
PG10 0.000000e+00   0.000000e+00 0.000000e+00 0.000000e+00   0.000000e+00
      Supervision ContactLevel FinancialBudget
PG02 1.749631e-01 1.749631e-01    1.749631e-01
PG03 0.000000e+00 0.000000e+00    0.000000e+00
PG04 4.074351e-15          NaN             NaN
PG05 7.348316e-17 2.343827e-15    4.111322e-17
PG06 8.809021e-01 1.298034e+00    3.216738e-01
PG07 8.809021e-01 1.298034e+00    3.216738e-01
PG08 0.000000e+00 0.000000e+00    0.000000e+00
PG09 0.000000e+00 0.000000e+00    0.000000e+00
PG10 0.000000e+00 0.000000e+00    0.000000e+00

Value/SE (Wald statistics):
       (Intercept) EducationLevel   Experience     OrgImpact ProblemSolving
PG02  1.804388e+05  -1.688713e+05         -Inf -1.814265e+05   1.308111e+05
PG03           Inf           -Inf          Inf          -Inf            Inf
PG04          -Inf            NaN          NaN -6.736740e+17   2.552679e+18
PG05 -6.386192e+19   1.734706e+18 6.050696e+18 -1.057495e+20   1.255209e+20
PG06 -1.682590e+05   3.851448e+03 2.003330e+03 -7.430373e+03   1.803826e+04
PG07 -1.429953e+05   3.850140e+03 2.006220e+03 -7.432296e+03   1.645746e+04
PG08          -Inf            Inf          Inf          -Inf            Inf
PG09          -Inf           -Inf          Inf          -Inf           -Inf
PG10          -Inf            Inf          Inf          -Inf            Inf
       Supervision  ContactLevel FinancialBudget
PG02 -2.186961e+04 -1.981181e+04    8.073327e+04
PG03          -Inf           Inf            -Inf
PG04 -2.952536e+17           NaN             NaN
PG05 -4.269123e+19 -4.208639e+17    1.400137e+20
PG06 -4.019010e+03 -5.748241e+02    1.946045e+04
PG07 -4.018266e+03 -5.723216e+02    1.946035e+04
PG08          -Inf          -Inf             Inf
```

```
PG09           Inf          -Inf              Inf
PG10          -Inf           Inf              Inf

Residual Deviance: 13.0907
AIC: 157.0907
```

```r
cat(sprintf("Log likelihood: %.3f (%d df)
", logLik(MYglm)[1], attr(logLik(MYglm), "df")))
```

```
Log likelihood: -6.545 (72 df)
```

```r
if (is.null(MYglm$na.action)) omitted <- TRUE else omitted <- -MYglm$na.action
cat(sprintf("Pseudo R-Square: %.8f

",cor(apply(MYglm$fitted.values, 1, function(x) which(x == max(x))),
as.integer(MYdataset[omitted,]$PG))))
```

```
Pseudo R-Square: 0.99516038
```

```r
cat('==== ANOVA ====
')
```

```
==== ANOVA ====
```

```r
print(Anova(MYglm))
```

```
Analysis of Deviance Table (Type II tests)

Response: PG
               LR Chisq Df Pr(>Chisq)
EducationLevel  14.3433  9   0.110626
Experience      24.2064  9   0.003987 **
OrgImpact        1.6140  9   0.996209
ProblemSolving  21.1081  9   0.012179 *
Supervision      2.9187  9   0.967430
```

```
ContactLevel      4.8563  9   0.846653
FinancialBudget   5.5476  9   0.784206
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
print("
")
```

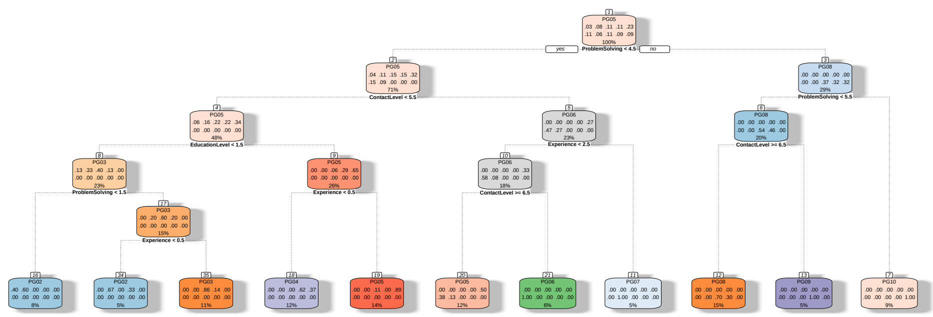
```
[1] "\n"
```

Now lets plot the Decision Tree

Decision Tree Plot

```
# Time taken: 0.16 secs

#=============================================================
# Rattle timestamp: 2016-04-27 12:51:52 x86_64-w64-mingw32

# Plot the resulting Decision Tree.

# We use the rpart.plot package.

fancyRpartPlot(MYrpart, main="Decision Tree MYdataset $ PG")
```

Decision Tree MYdataset $ PG



Rattle 2024−Jan−19 15:33:48 lyndonsundmark

A readable view of the decision tree can be found at the following pdf:

https://onedrive.live.com/redir?resid=4EF2CCBEDB98D0F5!6449&authkey=!ACgJAX95 1UZuo4s&ithint=file%2cpdf

##4.Evaluate And Critique Model

###Evaluate

Because we have multiple categories to be predicted, the only evaluation used is Error Matricies.

Lets see how well the models performed.

Decision Tree

```
#==============================================================
# Rattle timestamp: 2016-04-27 13:11:52 x86_64-w64-mingw32

# Evaluate model performance.

# Generate an Error Matrix for the Decision Tree model.

# Obtain the response from the Decision Tree model.

MYpr <- predict(MYrpart, newdata=MYdataset[,c(MYinput, MYtarget)], type="class")
```

```r
# Generate the confusion matrix showing counts.

table(MYdataset[,c(MYinput, MYtarget)]$PG, MYpr,
        dnn=c("Actual", "Predicted"))
```

```
      Predicted
Actual PG01 PG02 PG03 PG04 PG05 PG06 PG07 PG08 PG09 PG10
  PG01    0    2    0    0    0    0    0    0    0    0
  PG02    0    5    0    0    0    0    0    0    0    0
  PG03    0    0    6    0    1    0    0    0    0    0
  PG04    0    1    1    5    0    0    0    0    0    0
  PG05    0    0    0    3   12    0    0    0    0    0
  PG06    0    0    0    0    3    4    0    0    0    0
  PG07    0    0    0    0    1    0    3    0    0    0
  PG08    0    0    0    0    0    0    0    7    0    0
  PG09    0    0    0    0    0    0    0    3    3    0
  PG10    0    0    0    0    0    0    0    0    0    6
```

```r
# Generate the confusion matrix showing proportions.

pcme <- function(actual, cl)
{
  x <- table(actual, cl)
  nc <- nrow(x)
  tbl <- cbind(x/length(actual),
               Error=sapply(1:nc,
                 function(r) round(sum(x[r,-r])/sum(x[r,]), 2)))
  names(attr(tbl, "dimnames")) <- c("Actual", "Predicted")
  return(tbl)
}
per <- pcme(MYdataset[,c(MYinput, MYtarget)]$PG, MYpr)
round(per, 2)
```

```
      Predicted
Actual PG01 PG02 PG03 PG04 PG05 PG06 PG07 PG08 PG09 PG10 Error
  PG01    0 0.03 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00  1.00
  PG02    0 0.08 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00  0.00
  PG03    0 0.00 0.09 0.00 0.02 0.00 0.00 0.00 0.00 0.00  0.14
  PG04    0 0.02 0.02 0.08 0.00 0.00 0.00 0.00 0.00 0.00  0.29
  PG05    0 0.00 0.00 0.05 0.18 0.00 0.00 0.00 0.00 0.00  0.20
```

```
  PG06     0 0.00 0.00 0.00 0.05 0.06 0.00 0.00 0.00 0.00  0.43
  PG07     0 0.00 0.00 0.00 0.02 0.00 0.05 0.00 0.00 0.00  0.25
  PG08     0 0.00 0.00 0.00 0.00 0.00 0.00 0.11 0.00 0.00  0.00
  PG09     0 0.00 0.00 0.00 0.00 0.00 0.00 0.05 0.05 0.00  0.50
  PG10     0 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.09  0.00
```

```
# Calculate the overall error percentage.

cat(100*round(1-sum(diag(per), na.rm=TRUE), 2))
```

23

```
# Calculate the averaged class error percentage.

cat(100*round(mean(per[,"Error"], na.rm=TRUE), 2))
```

28

Random Forest

```
# Generate an Error Matrix for the Random Forest model.

# Obtain the response from the Random Forest model.

MYpr <- predict(MYrf, newdata=na.omit(MYdataset[,c(MYinput, MYtarget)]))

# Generate the confusion matrix showing counts.

table(na.omit(MYdataset[,c(MYinput, MYtarget)])$PG, MYpr,
        dnn=c("Actual", "Predicted"))
```

```
      Predicted
Actual PG01 PG02 PG03 PG04 PG05 PG06 PG07 PG08 PG09 PG10
  PG01    1    1    0    0    0    0    0    0    0    0
  PG02    0    5    0    0    0    0    0    0    0    0
  PG03    0    0    7    0    0    0    0    0    0    0
  PG04    0    0    0    7    0    0    0    0    0    0
  PG05    0    0    0    0   15    0    0    0    0    0
```

```
PG06    0    0    0    0    0    7    0    0    0    0
PG07    0    0    0    0    0    0    4    0    0    0
PG08    0    0    0    0    0    0    0    7    0    0
PG09    0    0    0    0    0    0    0    0    6    0
PG10    0    0    0    0    0    0    0    0    0    6
```

```
# Generate the confusion matrix showing proportions.

pcme <- function(actual, cl)
{
  x <- table(actual, cl)
  nc <- nrow(x)
  tbl <- cbind(x/length(actual),
              Error=sapply(1:nc,
                function(r) round(sum(x[r,-r])/sum(x[r,]), 2)))
  names(attr(tbl, "dimnames")) <- c("Actual", "Predicted")
  return(tbl)
}
per <- pcme(na.omit(MYdataset[,c(MYinput, MYtarget)])$PG, MYpr)
round(per, 2)
```

```
      Predicted
Actual PG01 PG02 PG03 PG04 PG05 PG06 PG07 PG08 PG09 PG10 Error
  PG01 0.02 0.02 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00   0.5
  PG02 0.00 0.08 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00   0.0
  PG03 0.00 0.00 0.11 0.00 0.00 0.00 0.00 0.00 0.00 0.00   0.0
  PG04 0.00 0.00 0.00 0.11 0.00 0.00 0.00 0.00 0.00 0.00   0.0
  PG05 0.00 0.00 0.00 0.00 0.23 0.00 0.00 0.00 0.00 0.00   0.0
  PG06 0.00 0.00 0.00 0.00 0.00 0.11 0.00 0.00 0.00 0.00   0.0
  PG07 0.00 0.00 0.00 0.00 0.00 0.00 0.06 0.00 0.00 0.00   0.0
  PG08 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.11 0.00 0.00   0.0
  PG09 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.09 0.00   0.0
  PG10 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.09   0.0
```

```
# Calculate the overall error percentage.

cat(100*round(1-sum(diag(per), na.rm=TRUE), 2))
```

```
2
```

```
# Calculate the averaged class error percentage.

cat(100*round(mean(per[,"Error"], na.rm=TRUE), 2))
```

```
5
```

Support Vector Machine

```
# Generate an Error Matrix for the SVM model.

# Obtain the response from the SVM model.

MYpr <- kernlab::predict(MYksvm, newdata=na.omit(MYdataset[,c(MYinput, MYtarget)]))

# Generate the confusion matrix showing counts.

table(na.omit(MYdataset[,c(MYinput, MYtarget)])$PG, MYpr,
        dnn=c("Actual", "Predicted"))
```

```
      Predicted
Actual PG01 PG02 PG03 PG04 PG05 PG06 PG07 PG08 PG09 PG10
  PG01    0    1    1    0    0    0    0    0    0    0
  PG02    0    5    0    0    0    0    0    0    0    0
  PG03    0    2    3    1    1    0    0    0    0    0
  PG04    0    0    0    6    1    0    0    0    0    0
  PG05    0    0    0    1   12    2    0    0    0    0
  PG06    0    0    0    0    1    6    0    0    0    0
  PG07    0    0    0    0    0    4    0    0    0    0
  PG08    0    0    0    0    0    0    0    7    0    0
  PG09    0    0    0    0    0    0    0    4    2    0
  PG10    0    0    0    0    0    0    0    0    0    6
```

```
# Generate the confusion matrix showing proportions.

pcme <- function(actual, cl)
{
  x <- table(actual, cl)
  nc <- nrow(x)
  tbl <- cbind(x/length(actual),
```

```
                 Error=sapply(1:nc,
                   function(r) round(sum(x[r,-r])/sum(x[r,]), 2)))
  names(attr(tbl, "dimnames")) <- c("Actual", "Predicted")
  return(tbl)
}
per <- pcme(na.omit(MYdataset[,c(MYinput, MYtarget)])$PG, MYpr)
round(per, 2)
```

```
      Predicted
Actual PG01 PG02 PG03 PG04 PG05 PG06 PG07 PG08 PG09 PG10 Error
  PG01    0 0.02 0.02 0.00 0.00 0.00    0 0.00 0.00 0.00  1.00
  PG02    0 0.08 0.00 0.00 0.00 0.00    0 0.00 0.00 0.00  0.00
  PG03    0 0.03 0.05 0.02 0.02 0.00    0 0.00 0.00 0.00  0.57
  PG04    0 0.00 0.00 0.09 0.02 0.00    0 0.00 0.00 0.00  0.14
  PG05    0 0.00 0.00 0.02 0.18 0.03    0 0.00 0.00 0.00  0.20
  PG06    0 0.00 0.00 0.00 0.02 0.09    0 0.00 0.00 0.00  0.14
  PG07    0 0.00 0.00 0.00 0.00 0.06    0 0.00 0.00 0.00  1.00
  PG08    0 0.00 0.00 0.00 0.00 0.00    0 0.11 0.00 0.00  0.00
  PG09    0 0.00 0.00 0.00 0.00 0.00    0 0.06 0.03 0.00  0.67
  PG10    0 0.00 0.00 0.00 0.00 0.00    0 0.00 0.00 0.09  0.00
```

```
# Calculate the overall error percentage.

cat(100*round(1-sum(diag(per), na.rm=TRUE), 2))
```

29

```
# Calculate the averaged class error percentage.

cat(100*round(mean(per[,"Error"], na.rm=TRUE), 2))
```

37

Linear Model

```
# Generate an Error Matrix for the Linear model.
```

```r
# Obtain the response from the Linear model.

MYpr <- predict(MYglm, newdata=MYdataset[,c(MYinput, MYtarget)])

# Generate the confusion matrix showing counts.

table(MYdataset[,c(MYinput, MYtarget)]$PG, MYpr,
        dnn=c("Actual", "Predicted"))
```

```
      Predicted
Actual PG01 PG02 PG03 PG04 PG05 PG06 PG07 PG08 PG09 PG10
  PG01    1    1    0    0    0    0    0    0    0    0
  PG02    0    5    0    0    0    0    0    0    0    0
  PG03    0    0    7    0    0    0    0    0    0    0
  PG04    0    0    0    7    0    0    0    0    0    0
  PG05    0    0    0    0   15    0    0    0    0    0
  PG06    0    0    0    0    0    6    1    0    0    0
  PG07    0    0    0    0    0    2    2    0    0    0
  PG08    0    0    0    0    0    0    0    7    0    0
  PG09    0    0    0    0    0    0    0    0    6    0
  PG10    0    0    0    0    0    0    0    0    0    6
```

```r
# Generate the confusion matrix showing proportions.

pcme <- function(actual, cl)
{
  x <- table(actual, cl)
  nc <- nrow(x)
  tbl <- cbind(x/length(actual),
               Error=sapply(1:nc,
                 function(r) round(sum(x[r,-r])/sum(x[r,]), 2)))
  names(attr(tbl, "dimnames")) <- c("Actual", "Predicted")
  return(tbl)
}
per <- pcme(MYdataset[,c(MYinput, MYtarget)]$PG, MYpr)
round(per, 2)
```

```
      Predicted
Actual PG01 PG02 PG03 PG04 PG05 PG06 PG07 PG08 PG09 PG10 Error
  PG01 0.02 0.02 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00  0.50
```

```
PG02 0.00 0.08 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00  0.00
PG03 0.00 0.00 0.11 0.00 0.00 0.00 0.00 0.00 0.00 0.00  0.00
PG04 0.00 0.00 0.00 0.11 0.00 0.00 0.00 0.00 0.00 0.00  0.00
PG05 0.00 0.00 0.00 0.00 0.23 0.00 0.00 0.00 0.00 0.00  0.00
PG06 0.00 0.00 0.00 0.00 0.00 0.09 0.02 0.00 0.00 0.00  0.14
PG07 0.00 0.00 0.00 0.00 0.00 0.03 0.03 0.00 0.00 0.00  0.50
PG08 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.11 0.00 0.00  0.00
PG09 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.09 0.00  0.00
PG10 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.09  0.00
```

```r
# Calculate the overall error percentage.

cat(100*round(1-sum(diag(per), na.rm=TRUE), 2))
```

```
6
```

```r
# Calculate the averaged class error percentage.

cat(100*round(mean(per[,"Error"], na.rm=TRUE), 2))
```

```
11
```

Critique

It turns out that:

- The model that performed best was Random Forests at 2% error
- The linear model was next at 6% error.

- Support Vector Machines performed less well at 18% error.
- And Decision trees, while being able to give us 'visual' on what rules are being used, performed worst of all at 23% error.

Earlier I said ,that for purposes of this analysis, we would not have training and test datasets. This is because the total population of our data is only 66 records which are scattered among up to 10 categories. Unless we took special care, if we randomly created test and training datasets, we could not guarantee that each of these datasets would have all 10 groups represented. So we use all the data to predict itself.(usually not recommended if lots of data)

If as an organization , we were just starting out with this kind of endeavor, at this point you would only have the job classification specs coded into the dataset. it would be the only ‘known’ population. In some ways it is useful to do this here- to gauge how well designed our paygrades are ‘differentiated’ from each other. Even though we know that the known population predicting itself will give lower error rates, suffice to say that here, the hypothetical organization is seeing results that would make it worthwhile to expand the coding effort. If at this stage, the data could not reliably predict itself, we might to rethink our approach.

In most organizations who have written job descriptions and job classification specs, their job descriptions **are** already classified as well. So we arent necessarily restricted to just coding the job class specs. We could go ahead and code the job descriptions on the same common denominator features.(outside the context of this blog article) This would make the ‘known’ population quite a bit bigger. On a bigger population too we could **also** have the population predict itself but additionally do cross validation and have both training and test datasets.

While the above results were found on just the job class specs, it would be wise to have a much larger population before deciding on which model is best to deploy in real life.

One other observation- you noticed in the results of the various models, that some model had predictions that were one or two paygrades off ‘higher or lower’ than the actual existing paygrade.

In a practical sense this might mean:

- these might be candidates for determining whether criteria/features for these pay grades should be redefined
- and or whether there are ,in reality, fewer categories needed.

We could extend our analysis and modelling to ‘cluster’ analysis.This would create a newer grouping based on the existing characteristics, and then the classification algorithms could be rerun to see if there was any improvement.

Some articles on People Analytics suggest that on a ‘maturity level’ basis, the step/stage beyond prediction is ‘experimental design’. If we are using our results to modify our design of our systems to predict better, that might be an example of this.

5.Present Results And Document

As with previous blog articles, a good way of carrying out this step is this the .rmd file which is used to create this blog article. R Markdown language is used to create this narrative as well as allow for ‘inline’ inclusion of the R program and its output. The rmd file can be found here:

https://onedrive.live.com/redir?resid=4EF2CCBEDB98D0F5!6467&authkey=!AE4IyMNEaoLgqnw&ithint=fil

##6.Deploy The Model

In R the easiest form of deploying the model, is to run your unknown data against the model . Put the data in a separate dataset and run the following R commands:

Here is dataset:

https://onedrive.live.com/redir?resid=4EF2CCBEDB98D0F5!6478&authkey=!ALYidIIpaCrfnf4&ithint=file%2ccsv

```
#DeployDataset <- read.csv("~/Documents/OneDrive/Public/R Job Classification Example/Deplo
DeployDataset <- read.csv("Deploydata.csv")
DeployDataset
```

```
  EducationLevel Experience OrgImpact ProblemSolving Supervision ContactLevel
1              2          0         3              4           1            5
  FinancialBudget
1               4
```

```
PredictedJobGrade <- predict(MYrf, newdata=DeployDataset)
PredictedJobGrade
```

```
   1
PG05
Levels: PG01 PG02 PG03 PG04 PG05 PG06 PG07 PG08 PG09 PG10
```

The DeployDataset represemts the information coded from a single job description (paygrade not known). PredictedJobGrade compares the coded values against the MYrf (random forest model) and the prediction is determined. In this case - PG05.

Final Comments

This is the third in a series of blog articles I have written to illustrate the use of People Analytics- **data driven** decision making for HR.

These articles have had the intention of providing real tangible examples of the application of data science to HR, to illustrate the data science process in the HR context, and to show that the scope mentioned previously in this article, isnt just theoretical- its real.

None of the articles is intended to illustrate necessarily best practices, but rather to show a structured process of thinking and analysis. The intention has also been to encourage more of being 'data driven' in HR, making People Analytics not an addon to HR but rather THE way we conducted HR decision making in the future, where applicable, human judgement is 'added on' to a rigorous analysis of the data in the first place.'