Better Name Matching via Machine Learning

By Jeremy Guinta and Nellie Ponarul

July 31, 2019

ankura
COLLABORATION DRIVES RESULTS

Introduction: Motivation

Task 1

Compare people's names across various data sets to determine if they are the same person or not, accounting for misspellings, nicknames, and other name discrepancies. Do this for hundreds of thousands of name comparisons.

Task 2

Improve upon existing name comparison processes to minimize error. Prior processes included:

- Exact name match (John = John)
- SoundEx / Fuzzy name match
  - John = Jon
  - May not correctly link "Philip" to "Filipe"
  - Only works on anglicized names
- Partial name match (e.g. first initial or first few characters match)
  - **Jon** = **Jon**athan
  - But incorrectly links **Mic**helle to **Mic**hael
- String Distance comparison (Simple string comparisons with a score threshold)

In all of these above methods, we would perform our matching process using the specific process, but we would **NEED** to perform **100% manual review**. to ensure the accuracy of the results.

Introduction: Motivation

Problem

- A 100% manual review is typically not feasible with the number of records we have in our population.
- The final dataset has over 700,000 name comparisons.
- Just using “Levenshtein” string distance comparisons (like we had in the past), was going to produce too much noise and require way too much manual review to get accurate.
- Our other methods even with string distance algorithms were not accurate enough for this exercise.

Introduction: Workflow

Data Sources

We have consumer data compiled from TransUnion, Experian, and MicroBilt.

- Data Aggregators.
- Contained profiles on consumers.
- If you have a cell phone, credit card, any other types of consumer information, you are likely in their database.
- These data sources contain personal information such as phone numbers, addresses and names.
- The initial comparison among the data was performed by various personal identifiable information.

Software

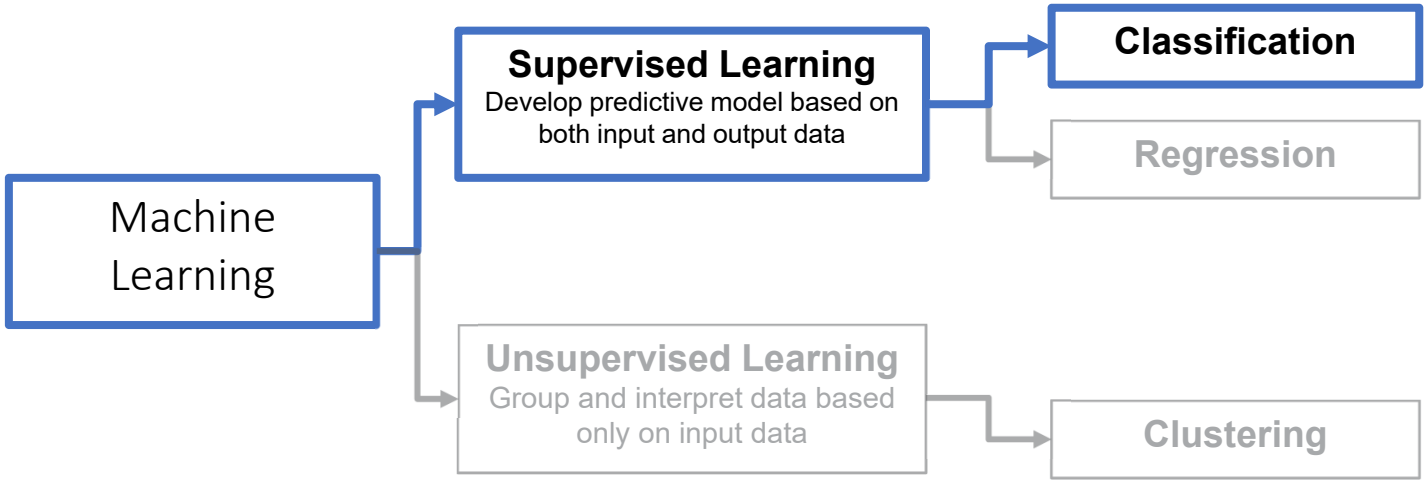
All of our work was performed in R (https://cran.r-project.org/) using Base R and variety of add-on packages.

- data.table (Data Management)
- tidyverse (More Data Management)
- stringdist (String Distance algorithms)
- h2o (Machine Learning modeling)
- stringr (String Parsing)
- sqldf (Soundex Algorithm)

Introduction: Process

What Machine Learning Process did we use?

- Our process was focused on binary classification.
- We are going to discuss a couple of terms that are applicable to Machine Learning and what we performed. There are many more terms and processes under the Machine Learning umbrella that we do not have the time to cover today.



- **Binary Classification** is a method for developing a model that predicts the output as either **Match or No Match.**

Introduction: Process

Supervised Learning - Classification

How is it useful?

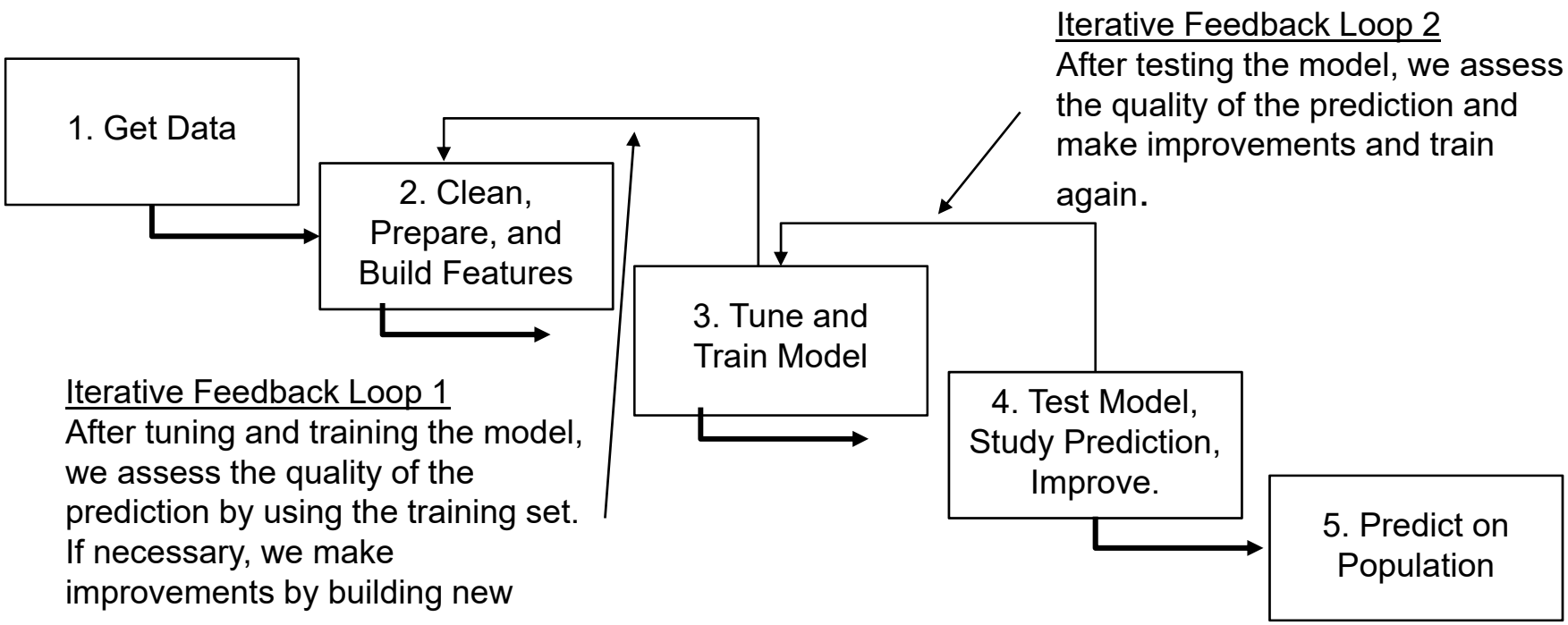
- **Speed**: review a small set of records and the machine does the rest.
- **Power**: the "machine" identifies patterns based upon learning.
- **Accuracy**: the "machine" can be more accurate than a human review. Once the "machine" learns the patterns within the data, it is consistent to the patterns it is taught.

How is it not?

- It takes time to get the model correct.
- We had to review and classify thousands of name comparisons by hand.
- An incorrectly specified model will lead to bad results (i.e. garbage in, garbage out).
- YOU often lose the ability to properly explain WHY decisions were made a certain way (although this is becoming less true with technological advances).
- Accuracy: Once the "machine" learns the patterns within the data, it is consistent to those patterns. If these patterns are not detected or not taught to the machine properly, then the prediction could be wrong.

Introduction: Process

Data Analysis Process



Introduction: Evaluation

How to evaluate the models?

ROCR Curve

- Common technique to evaluate Machine Learning Binary Classifier models
- The ROCR also provides a single measurement of prediction accuracy called AUC (Area Under the Curve).
- The process assigns different thresholds from 0.01 to 0.99 and then checks the classification at each threshold.

Confusion Matrix

- Shows all possible combinations of the prediction.
- Easy to interpret and determine False Positives and True Negatives
- Makes determination based on a 50% threshold

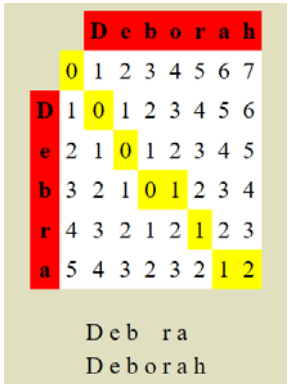
Features: StringDist()

What is String Distance?

- It is a series of mathematical algorithms developed for Natural Language Processing (NLP) that compares two strings and computes the difference between two character strings in a variety of ways.
- We used the package `stringdist` in R to apply these algorithms.
- There are actually many different algorithms:
  - OSA, Levenshtein, Damerau-Levenshtein, LCS,

    Q-Grams, Jaccard, Jaro, Cosine, Soundex
  - Each of the algorithms produces either a number or a proportion.

Levenshtein Distance:



= 2

ankura
COLLABORATION DRIVES RESULTS

Features: StringDist()

"Edit-Based" Distances:

Definition: Count the minimum number of (weighted) basic operations that turns string "s" into string "t."

	Allowed operation			
Distance	substitution	deletion	insertion	transposition
Hamming	✔	✖	✖	✖
LCS	✖	✔	✔	✖
Levenshtein	✔	✔	✔	✖
OSA	✔	✔	✔	✔*
Damerau-Levenshtein	✔	✔	✔	✔

*Substrings may be edited only once.

Features: StringDist()

1. Hamming

Hamming distance is undefined when comparing strings of different length.

```
stringdist("debras","deborah",method = "hamming")

## [1] Inf
```

a. replace the "a" in debra with an "o" [1 pt.]

```
stringdist("debra","debro",method = "hamming")

## [1] 1
```

2. Longest Common Substring (lcs)

a. delete the "o" in deborah [1 pt.]
b. delete the "h" in now debrah [1 pt.]
c. insert an "s" in now debra [1 pt.]

```
stringdist("debras","deborah",method = "lcs")

## [1] 3
```

3.Levenshtein (lv)

a. delete the "o" in deborah [1 pt.]

b. subsitute the "h" in now debrah with a "s" [1 pt.]

```
stringdist("debras","deborah",method = "lv")

## [1] 2
```

4. Optimal String Alignment (osa)

a. transpose the "o" and "r" in deborah [1 pt.]
b. subsitute the "h" in now debroah with a "s" [1 pt.]

```
stringdist("debroas","deborah",method = "osa")

## [1] 2
```

Compared to Levenshtein

a. subsitute the "o" in deborah with an "r" [1 pt.]
b. subsitute the now second "r" in now debrrah with an "r" [1 pt.]
c. subsitute the "o" in now deboras with an "r" [1 pt.]

```
stringdist("debroas","deborah",method = "lv")

## [1] 3
```

5. (Full) Damerau-Levenshtein distance (DL)

a. insert an "s" in deborah [1 pt.]
b. transpose the "r" and "o" (across two locations) in debosrah [1 pt.]

```
stringdist("debrsoah","deborah",method = "dl")

## [1] 2
```

Compared to OSA

a. insert an "s" in deborah [1 pt.]
b. transpose the "o" and "r" in deborsah [1 pt.]
c. transpose the "o" and "s" in debrosah [1 pt.]

```
stringdist("debrsoah","deborah",method = "osa")

## [1] 3
```

Features: StringDist()

"Numerical-Based" Distances:

Definition: Calculated score based on the occurrence of identical string patterns of length *q* (*q-grams)*

```
a <- "maryjane"
b <- "mariejane"

qgrams(a,b, q = 2)
```

```
##    ma ry yj ne ri ar ja ie ej an
## V1  1  1  1  1  0  1  1  0  0  1
## V2  1  0  0  1  1  1  1  1  1  1
```

```
A <- qgrams(a,b, q = 2)[1,]
B <- qgrams(a,b, q = 2)[2,]
```

Features: StringDist()

Cosine distance

Cosine distance is 1 minus the cosine similarity of the two q-gram vectors

Given that the q-gram vector of "maryjane" is $\vec{A}$ and the q-gram vector of "mariejane" is $\vec{B}$, the cosine distance is:

$$1 - \frac{\vec{A} \cdot \vec{B}}{||\vec{A}||||\vec{B}||}$$

```
# Cosine string distance with q-gram length 2
1-((sum(A*B,na.rm=T))/(sqrt(sum(A*A, na.rm=T))*sqrt(sum(B*B, na.rm=T))))
```

```
## [1] 0.3318469
```

Jaccard distance

Jaccard distance is 1 minus the ratio of shared q-grams to total q-grams

$$1 - \frac{\text{Number of shared q-grams}}{\text{Number of all q-grams}}$$

```
a_l <- unlist(strsplit(a,""))
b_l <- unlist(strsplit(b,""))

# m: shared characters
m <- length(intersect(a_l, b_l))
# Jaccard string distance with q-gram length 2
1 - (sum(pmin(A,B))/sum(max(length(A), length(B))))
```

```
## [1] 0.5
```

Features: StringDist()

Jaro-Winkler and Jaro distance

Jaro-Winkler distance is based on a mix of numerical components and edit-based components. It rates two strings closer to a match the longer the beginnings of the strings match.

The Jaro-Winkler distance is based on individual characters (not q-grams).

Characters are *matching* if they are the same character and no more then $\frac{max(|a|,|b|)}{2} - 1$ characters away.

We'll call the total number of matching characters between a and b m.

Transpositions are the the number of necessary transpositions of shared symbols.

The Jaro Similarity is defined as

$$d = \begin{cases} 0, & m = 0 \\ \frac{1}{3}\left(\frac{m}{|a|} + \frac{m}{|b|} + \frac{m-t}{m}\right), & \text{otherwise} \end{cases}$$

```
# Jaccard similarity:
 d <- function(A,B,m,t) {
     1-(1/3)*(m/A + m/B + (m-t)/m);
 }
```

Features: StringDist()

The Jaro-Winkler distance uses the Jaro Similarity and a prefix scale p to give more favorable scores to strings matching on a matching prefix length l.

Jaro-Winkler distance is thus: $d + lp(1 - d)$

```
a <- "maryjane"
b <- "mariejane"
m <- 7
A <- nchar(a)
B <- nchar(b)
t <- 2 # transpose the j by 1, transpose the n by 1
l <- 3 # length of shared prefix "mar"
jaro_sim <- d(A,B,m,t)
jaro_sim*(1-3*.25)
```

```
## [1] 0.05274471
```

Jaro Distance is a special case of the Jaro-Winkler distance where $p = 0$

```
jaro_sim*(1-3*0)
```

```
## [1] 0.2109788
```

Features: Other

We also developed other features

1. Length of the string.
    - Control for longer strings versus shorter strings
    - Short strings can be off by one or two characters, and have low string distance scores.
    - E.g. John to Jon will show a 1 for Levenshtein Distance (which we normalized to 75% matching. (1 – 1/4)

2. First Initial Match of each string – Simple indicator if the first initial of the first name or last name matched.

3. ~~Nick Names – Identified common nicknames of English names.~~
    - Eleanor = Nellie
    - Elizabeth = Betsey

4. Name Component Match
    - New string distance algorithm that we developed.
    - Strips apart all name components (e.g. “Mary Sue” becomes “Mary” and separately “Sue.”
    - Compares all name components to all other name components using simple Levenshtein stringdist().
    - Not order dependent
        - e.g., “Mary Sue” is vastly different than “Sue Mary” when using edit-based string distance algorithms.
        - New method would call this an exact match because we can link “Sue” to “Sue” and “Mary” to “Mary.”

Features: Final

Our analysis dataset contained nearly 200 individual features

1. Normalized String Distance values
   - Normalized indicates that we convert the string distance value into a percentage match for edit based algorithms. We never use the raw count.
   - First Name, Last Name, Middle Name, and Full Name for each string distance algorithm.

2. Other Features already described.
   - First Initial Match.
   - Name Component Match Score.
   - String Length.

Solution: Introduction

Our Solution

- We trained a model, using a human, to make initial name match decisions for over 20,000 randomly selected out of the over 700,000 name comparison pairs.
- Dependent Variables: Match: Label of “Match”, “No Match”
- Independent Variables (“Features”)
  - Normalized computation of every string distance algorithm for first name, last name, and full name.
  - Length of the string.
  - Soundex categorical variable (No Match, Weak Match, Match, Strong Match, Very Strong Match).
  - First Initial Match indicator.
  - Name Component Match.
  - In total there were over 100 independent variable inputs.
- We used a 70% / 30% training, test split.

Solution: Introduction

Our Solution

- We used a binomial regression, random forest, gradient boosted machines, distributed random forest, and deep learning.
- We trained each model hundreds of times using a process called “randomized grid search.”
- Randomized grid search allows us to provide the model with different tuning inputs that will help the model optimize prediction.
- We ensembled the best model from each algorithm and prepared a “Super Learner.”

Solution: Introduction

Our Solution

- Examples of Tuning Parameters
  - Cross Validation “Folds” - Number of times to cross-validate the model.
  - Alpha – Regularization.
  - Lambda – Strength of the regularization.
  - Lambda Search – Let the model find the optimal lambda value.
  - Early Stopping – Should the model continue if little iterative improvement occurs at each step, or wait until convergence.
  - Standardized – Let the model automatically standardize the inputs.
- Randomized Grid Search
  - This process assigns a random value to each parameter above and runs the model.
  - This process is repeated with randomly assigned inputs for as many times as the analyst desires.
  - Accuracy is computed at each iteration of this process and is saved.
  - At the end of the process each of the models generated is ranked from highest accuracy to lowest accuracy. The model with the highest accuracy is selected as the “best” model.
  - The analyst loses control over selecting the tuning inputs, but if randomized grid search is allowed to perform 100s or 1000s of iterations, there should be enough models generated that the “best” model is found.

More here: http://docs.h2o.ai/h2o/latest-stable/h2o-docs/grid-search.html

Solution: HyperTuning

```
#Random Forest
hyper_params_rf <- list(
ntrees = c(5, 10, 25,50,75,100),
max_depth = c(10,15,20),
min_rows = c(5,10,30,70,100),
sample_rate = c(.95, .975,.99,.995,1),
col_sample_rate_per_tree = c(.5,
.6,.7,.8,.9,1),
nbins=c(2,5,10,15,20),
mtries=c(-1,5,10,15,20,25,30,35),
nbins_cats = c(64, 128, 256,
512,1024,1536)
)
```

```
#GBM
hyper_params_gbm <- list(
ntrees = c(5, 10, 25, 50,100,150,200),
max_depth = c(5, 10,15,20),
min_rows = c(2, 5,10,15,20),
learn_rate =
c(.06,.07,.08,.09,.1,.11,.12),
sample_rate = c(.95, .975,.99,.995,1),
col_sample_rate = c(.3,.4,.5,.6,.7),
col_sample_rate_per_tree = c(.5,
.6,.7,.8,.9,1),
nbins_cats = c(32,64,128,256),
learn_rate_annealing=c(0.25,0.5,0.75,
1)
)
```

```
#Binomial - GLM
hyper_params_glm <- list (
alpha =
c(0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1)
)
```

Solution: HyperTuning

```
#Neural Net
hyper_params_nn <- list(
epochs=20,
overwrite_with_best_model=FALSE,
hidden=list(c(32,32,32),c(64,64),
c(128,128,128)),
max_w2=10,
score_duty_cycle=0.025,
activation=c("Rectifier","Tanh","TanhW
ithDropout"),
input_dropout_ratio=c(0,0.05),
score_validation_samples=10000,
l1=c(.00001,.000001,.0000001),
l2=c(.00001,.000001,.0000001),
rho = c(.99,.975,1,0.95),
rate=c(.005,.0005,.00005),
rate_annealing=c(.00000001,.0000001,.0
00001),
momentum_start=c(.5,.1,.01,.05,.005),
momentum_stable=c(0.1, 0.2, 0.3,
0.4,0.5),
momentum_ramp=c(1000000,100000)
)
```

```
#Distributed Random Forest
hyper_params_drf <- list(
ntrees = c(5, 10, 25,50,75,100),
max_depth = c(10,15,20),
min_rows = c(5,10,30,70,100),
sample_rate = c(.95, .975,.99,.995,1),
col_sample_rate_per_tree = c(.5,
.6,.7,.8,.9,1),
nbins=c(2,5,10,15,20),
mtries=c(-1,5,10,15,20,25,30,35),
nbins_cats = c(64, 128, 256,
512,1024,1536)
)
```

More here: https://www.h2o.ai/blog/hyperparameter-optimization-in-h2o-grid-search-random-search-and-the-future/

Solution: Model Execution

Our Solution

```
search_criteria <- list(
  strategy = "RandomDiscrete",
  max_runtime_secs = 30*60,  #30 minutes per run
  max_models = 500
)
xnames<-names(trn)

model<-h2o.grid(algorithm = “<Model Type>”,
         x = xnames, y = “category”,
         training_frame = trn,
         hyper_params =<List of Parameters>,
         search_criteria = <List of Search>,
         stopping_metric = "misclassification", stopping_tolerance = 1e-3,
         stopping_rounds = 3,
         seed = -1,nfolds = 5, fold_assignment = "Modulo",
         keep_cross_validation_predictions = TRUE
)
```

More here: http://docs.h2o.ai/h2o/latest-stable/h2o-docs/index.html

Solution: Model Execution

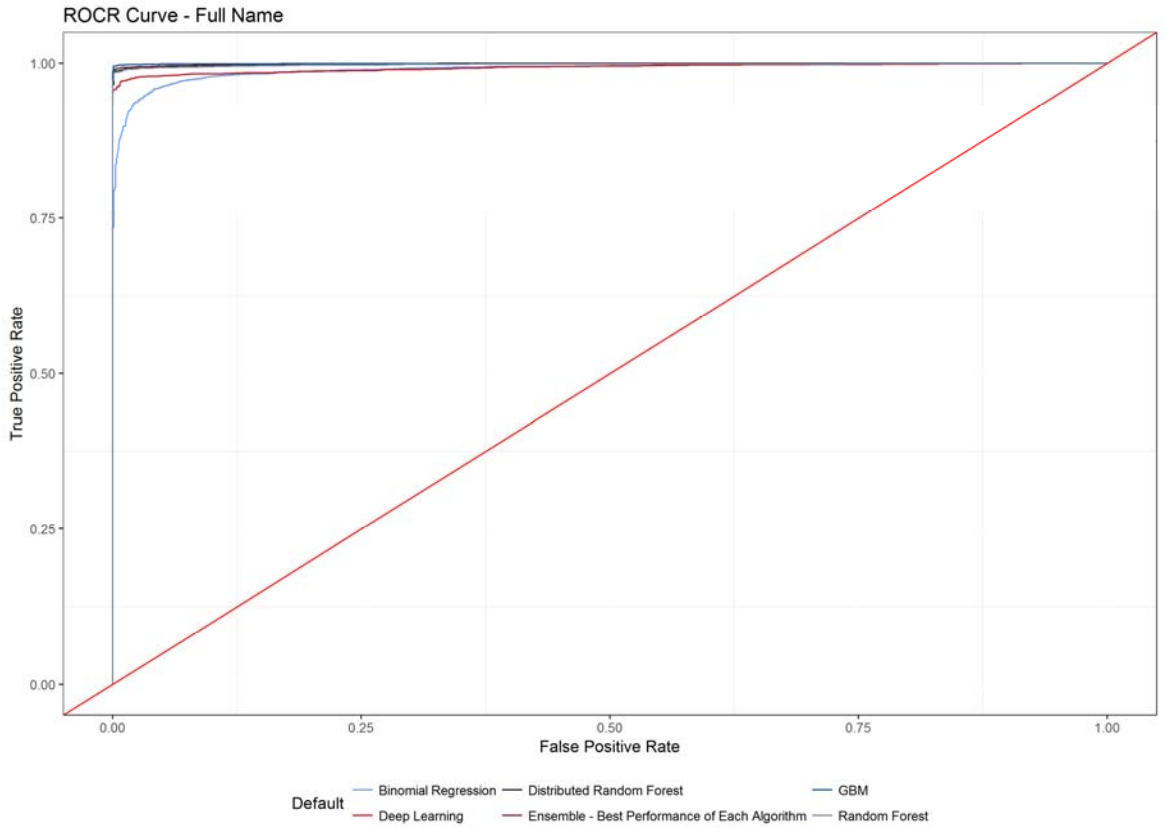
Our Solution

```
model_sort <- h2o.getGrid(grid_id = model@grid_id, sort_by = "err", decreasing =
FALSE)

model_best <- h2o.getModel(model_sort@model_ids[[1]])
summary(model_best)

pred_rf1<-h2o.predict(model_best, newdata = tst)
pref_rf1<-h2o.performance(model_best, newdata=tst)

ens<-h2o.stackedEnsemble(
         x = xnames,
         y="category",
         training_frame = trn,
base_models = list(<List of Best Models>)
)


pred_ens1 <- h2o.predict(ens1, newdata = tst)
pref_ens1 <-h2o.performance(ens1, newdata=tst)
```

More here: http://docs.h2o.ai/h2o/latest-stable/h2o-docs/index.html

Solution: Model Performance

Model Performance – ROCR – Full Name Match



Solution: Model Performance

Model Performance – Confusion Matrix

Model	Match	No Match	Error	Rate
GLM - Binomial	5,166	224	4.2%	=224/5390
	217	5,046	4.1%	=217/5263
	5,383	5,270	4.1%	=441/10653
Gradient Boosted Machines	5,386	4	0.1%	=4/5390
	25	5,238	0.5%	=25/5263
	5,411	5,242	0.3%	=29/10653
Random Forest	5,384	6	0.1%	=6/5390
	77	5,186	1.5%	=77/5263
	5,461	5,192	0.8%	=83/10653
Distributed Random Forest	5,384	6	0.1%	=6/5390
	74	5,189	1.4%	=74/5263
	5,458	5,195	0.8%	=80/10653
Deep Learning	5,348	42	0.8%	=42/5390
	155	5,108	2.9%	=155/5263
	5,503	5,150	1.8%	=197/10653
Ensemble - Best Performance of Each Algorithm	5,375	15	0.3%	=15/5390
	47	5,216	0.9%	=47/5263
	5,422	5,231	0.6%	=62/10653

Solution: Model Performance

Examples of Our Solution Working – Strong Predicted Match, No or Weak String Distance Match.

Full Name 1	Full Name 2	Manual Human Review	Predicted Category	Predicted Match	First Name String Distance	Last Name String Distance
lance dj turner	clarence lee tucker	No Match	No Match	0.1%	62.5%	66.7%
glenda m sapperfield	brenda k satterfield	No Match	No Match	0.1%	66.7%	81.8%
louis vitanzo	doris e vitanza	No Match	No Match	0.1%	60.0%	85.7%
carol sue heyvaert	carla a heytvelt	No Match	No Match	0.1%	60.0%	62.5%
brenda satterfield	glenda m sapperfield	No Match	No Match	0.1%	66.7%	81.8%
kathleen marie charles	katherine p challis	No Match	No Match	0.2%	55.6%	71.4%
christopher j stansa	christine kay stanfa	No Match	No Match	1.0%	63.6%	83.3%
roger albert cote	robert a cote	No Match	No Match	1.0%	66.7%	100.0%
peter ross archer	pete anchor	No Match	No Match	1.4%	80.0%	66.7%
lindsey mmesser	linda c messer	No Match	No Match	2.2%	57.1%	85.7%
sherry french	cherry c fience	No Match	No Match	2.4%	83.3%	66.7%
erin hurlie	erica k hurley	No Match	No Match	3.7%	60.0%	66.7%
amelia dejesus alvarez	melissa alvares	No Match	No Match	4.2%	57.1%	85.7%
chay sliegelman	chaya f fligelman	No Match	No Match	6.3%	80.0%	80.0%
darrel botaw	darren thomas votaw	No Match	No Match	13.9%	83.3%	80.0%
james sheidan	janet sheridan	No Match	No Match	18.4%	60.0%	87.5%
karen leary	lauren k oleary	No Match	No Match	24.2%	66.7%	83.3%
mario reyes	maria elena dereyes	No Match	No Match	28.0%	80.0%	71.4%
kevin f mcandrew	keelin i mcandrew	No Match	No Match	36.4%	66.7%	100.0%
jesse gonzales	jose cruz gonzalez	No Match	No Match	45.9%	60.0%	87.5%

Solution: Model Performance

Examples of Our Solution Working – Strong Predicted No Match, Strong String Distance Match.

Full Name 1	Full Name 2	Manual Human Review	Predicted Category	Predicted Match	First Name String Distance	Last Name String Distance
ondrak keven	kevin ondark	Match	Match	98.4%	0.0%	0.0%
aicrespo@peoplepc.com	a crespo	Match	Match	98.4%	4.8%	0.0%
kappas florence	florance kappes	Match	Match	98.4%	12.5%	12.5%
miceli shirley	shirley miceli	Match	Match	98.4%	14.3%	14.3%
eugene f earl	earl e	Match	Match	98.4%	16.7%	25.0%
deveer thomas van	thomas r vanderveer	Match	Match	98.4%	0.0%	30.0%
s bates	sara ann barnes-bates	Match	Match	98.4%	25.0%	41.7%
g preston-banks	gail preston	Match	Match	98.4%	25.0%	53.8%
lesia wills-johnston	lw johnston	Match	Match	98.0%	20.0%	57.1%
donald john lamey	donl j lmy	Match	Match	97.3%	66.7%	60.0%
thomas fraone	tom farone	Match	Match	98.4%	50.0%	66.7%
chuanchin slu	chuan chin lu	Match	Match	98.2%	55.6%	66.7%
george w clarke	grge w clrk	Match	Match	98.3%	66.7%	66.7%
donl stly	donald d stiely	Match	Match	98.4%	66.7%	66.7%
robert h dewolf	robt h wolf	Match	Match	98.4%	66.7%	66.7%
grge w clrk	george w clarke	Match	Match	98.3%	66.7%	66.7%
danny p st pierre	stdanny pierre	Match	Match	98.4%	71.4%	66.7%
betty & tom lehman	betty lehnian	Match	Match	98.4%	45.5%	71.4%
brad pierson	bradley robert pearson	Match	Match	98.3%	57.1%	71.4%
rich dicuezo	richard dicenzo	Match	Match	98.4%	57.1%	71.4%

Solution: Model Performance

Nobody is Perfect…

Full Name 1	Full Name 2	Manual Human Review	Predicted Category	Predicted Match	First Name String Distance	Last Name String Distance
annemari squatritl	annamarie squafrito	No Match	Match	98.4%	77.8%	77.8%
steve w conner	stephen m oconnor	No Match	Match	98.4%	57.1%	71.4%
katherine challis	kathleen marie charles	No Match	Match	96.3%	55.6%	71.4%
adwar baba	edward aziz baba-ian	No Match	Match	97.7%	66.7%	50.0%
jeffrey widham	jeffery michael wickham	No Match	Match	97.6%	71.4%	71.4%
carol fieldsreed	carolyn z fields	No Match	Match	98.2%	71.4%	60.0%
warren t olsommer	warren disommer	No Match	Match	98.4%	75.0%	75.0%
mp mcgarrity	m garrity	No Match	Match	98.2%	50.0%	77.8%
fred schnieder	fredrick elmer schneider	No Match	Match	98.2%	50.0%	77.8%
clifford colinsjr	c collins	No Match	Match	98.4%	12.5%	62.5%
khanngeun keopradit	khanpoheh keopardit	No Match	Match	98.0%	44.4%	77.8%
margaret argcatoline	maria m catoline	No Match	Match	96.8%	50.0%	72.7%

Conclusion

- There is always a better way…
- Our solution that uses string distance algorithms as features outperformed traditional methods that use string distance algorithms by themselves.
- We are able to match similar looking / sounding names that traditional methods would fail to consider a match.

Questions?

JEREMY GUINTA Jeremy.Guinta@ankura.com

NELLIE PONARUL Nellie.Ponarul@ankura.com