Project 491
Ruiqi Zhao, Liting Chen

String Similarity Metrics Comparison for Name-matching Task

Abstract:

For name-matching data, we evaluated the performance of Jaro-Winkler distance, Levenshtein distance, and Sorensen-dice coefficient. After using different model to test these three similarity metrics, we found that Jaro-Winkler distance and Levenshtein distance performed better than Sorensen-dice coefficient and with Jaro-Winkler distance performs slightly better than Levenshtein distance. First, we explored record linkage similarity metrics to determine which are suitable for predicting names matches/nonmatches.

Introduction:

There are several record linkage similarity metric. We explored different similarity metric to decide which one is suitable for predicting name matches/nonmatches.
Euclidean distance measures distance between two points, given by Pythagorean formula.

$$\mathrm{d}(\mathbf{p},\mathbf{q}) = \mathrm{d}(\mathbf{q},\mathbf{p}) = \sqrt{(q_1-p_1)^2+(q_2-p_2)^2+\cdots+(q_n-p_n)^2} = \sqrt{\sum_{i=1}^{n}(q_i-p_i)^2}.$$

Euclidean space becomes a metric space by using this formula.
Cosine similarity measures similarity between two vectors in an inner product space. Cosine similarity is independent of input's magnitude. It is used in positive space, with outcome range from 0 to 1. The formula is as follows:

$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}}$$

The similarity ranges from -1 to 1. -1 represents exactly opposite; 1 represents exactly same; 0 represents in dependence. More closer to value 1, more similarity.
Cosine similarity is most commonly used in high-dimensional positive spaces. In the field of text matching and informational retrieval, vector A and B are term frequency vectors of the documents. In a document is assigned with vectors where each dimension represents the frequency of terms appear in the document. Cosine similarity measures how similar two documents in terms of their term frequency.

Jaccard Index is also known as the Jaccard similarity coefficient. Jaccard index is used for comparing the similarity and diversity of sample sets.

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}.$$

Jaccard distance measures how dissimilar two sample sets are. As the formula show, Jaccard distance is the complementary to the Jaccard coefficient.

$$d_J(A,B) = 1 - J(A,B) = \frac{|A \cup B| - |A \cap B|}{|A \cup B|}.$$

Jaccard index and Jaccard distance measure the overlap of binary attributes in A and B.
M_{11} represents the total number of attributes where A and B both have a value of 1.
M_{01} represents the total number of attributes where the attribute of A is 0 and the attribute of B is 1.
M_{10} represents the total number of attributes where the attribute of A is 1 and the attribute of B is 0.
M_{00} represents the total number of attributes where A and B both have a value of 0.

The Jaccard similarity coefficient, *J*, is given as

$$J = \frac{M_{11}}{M_{01} + M_{10} + M_{11}}.$$

The Jaccard distance, *J'*, is given as

$$J' = \frac{M_{01} + M_{10}}{M_{01} + M_{10} + M_{11}}.$$

String kernel measures how similar two texts are by operating kernel function on strings. The more similar pairs of strings in two texts, the higher the value of a string kernel will be. Algorithm with kernel function prediction allows direct string comparison. It helps avoid converting strings to fixed-length or feature vectors.
Hamming distance measures the minimum number of substitutions needed to change one string into the other for two strings to equal length.
Jaro-Winkler distance measures the distance between two strings, s1 and s2. In the formula below, m is the number of matching characters, t is half the number of transpositions.

$$d_j = \begin{cases} 0 & \text{if } m = 0 \\ \frac{1}{3}\left(\frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m-t}{m}\right) & \text{otherwise} \end{cases}$$

dj is the Jaro distance for strings s1 and s2. l is the length of common prefix for a maximum of 4 characters. P is the constant scaling factor for giving favourable ratings to strings that matches from the beginning than toward the end. The common value used for p is 0.1.

$$d_w = d_j + (\ell p(1 - d_j))$$

Levenshtein distance is the minimum number of character edits one word needed to change into the other. Character edit includes insertion, deletion, and substitution.

Sørensen–Dice coefficient can be used to compare sample data set. The set operations can be expressed in terms of vector operations over binary vector A and B. Sorenson-Dice coefficient can also be used to compare string similarity.

$$s = \frac{2n_t}{n_x + n_y}$$

nt is the number of bigram element in common in string A and B. nx is the number of bigrams in A and ny is the number of bigrams in B.

After exploring this similarity distances and metrics, we chose to use Jaro-Winkler metrics, Levenstein distance, and Sorensen-Dice coefficient for predicting real names matches from a names list. These metrics take input as strings while metrics, such as string kernel does text comparison. In addition, the three metrics we chose measure similarities in different methods.

Data:

The dataset we are using are subset of RLdata500 and RLdata10000 from the R data library. These tables contain artificial personal data for the evaluation of Record Linkage procedures. The RLdata500 has 500 records and the RLdata10000 has 10000 records. The variables we are interested in here are first name first component, last name first component, year of birth, month of birth, day of birth and their ID numbers. Our train data is from RLdata10000. Since the data set is not in specific order, we take a subset containing the first 2000 records. There are 184 matches in this subset. We mixed the 184 matches with 300 nonmatches to get our 484 records train data. To ensure out-of-sample data testing, we pick our test data from RLdata500. We take all the 100 matches from it and mix it with randomly generated 100 nonmathces from the data. The size of our test data is 200 records. We use randomly generated data sets form test data as we want to reduce coincident patterns when testing the models.

Methods:

We fit four different models to our similarity matrices to see which similarity metric outperform others across all models. Logistic Regression, Classification Tree, Fellegi-Sunter and Random Forest are the models we chose.

Logistic regression:

JW	Estimate	Std. Error	Z value	P-value
fname_c1.comp	94.96	69.61	1.36	0.17
lname_c1.comp	41.72	30.07	1.39	0.16
by.comp	48.97	16.62	2.94	0.003
bm.comp	5.38	2.86	1.88	0.060
bd.comp	9.55	4.14	2.31	0.021

Levenshtein	Estimate	Std. Error	Z value	P-value
fname_c1.comp	63.00	912.26	0.069	0.94
lname_c1.comp	76.81	1108.27	0.069	0.94
by.comp	108.17	1435.49	0.075	0.94
bm.comp	28.63	540.36	0.053	0.96

bd.comp	51.88	769.12	0.067	0.95

****** not converge

Dice	Estimate	Std. Error	Z value	P-value
fname_c1.comp	83.54	797.02	0.104	0.91
lname_c1.comp	83.62	805.18	0.103	0.92
by.comp	130.80	1420.61	0.092	0.92
bm.comp	32.09	390.30	0.082	0.93
bd.comp	70.55	678.88	0.109	0.92

****** not converge

Figure 1: Tables for the output of logistic regression model

	True NM	True M
predicted NM	19850	2
predicted M	0	48

Figure 2: Jaro-Winkler logistic regression error summary

We fit logistic regression model to the train data of Jaro-Winkler, Levenshtein, Dice similarities scores. Summary output is shown in **Figure 1**. Logistic regression model does not converge for Dice coefficient and Levenshtein similarities scores. This is probably because the train data does not have variables different enough to fit a logistic regression model. For Jaro-Winkler scores, we used trained logistic model to predict matches/nonmatches in train data. **Figure 2** shows the logistic regression error summary. It has 2 false negative error.

Classification tree:

We derive a Classification Tree model with the train data for each of the similarity techniques and then test out the model with test data. We choose 0.9 to be the probability threshold for predicting matches/nonmatches. As classification tree plot in **Figure 3, a, b ,c** shows, the probability is either closed to 0 or 1. We believe probability greater than 0.9 is large enough to predict the real matches. We trained classification tree model for three similarities scores. Error summary table is shows in **Figure 4.** Error summary are the same for three similarities scores. The model predictions has 8 false negative. The sameness of prediction error is probably due to the similarity in classification tree plot.

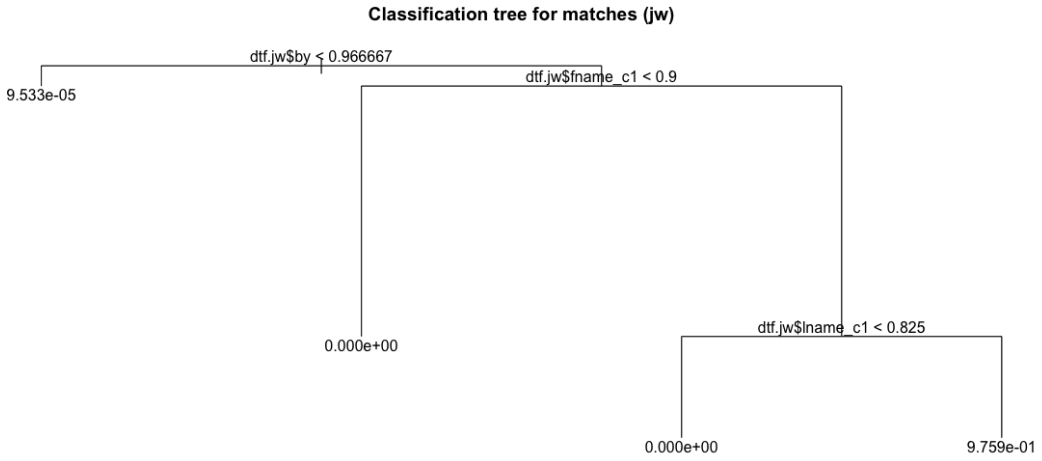

Figure 3.a: Classification tree plot for matches for Jaro-Winkler model

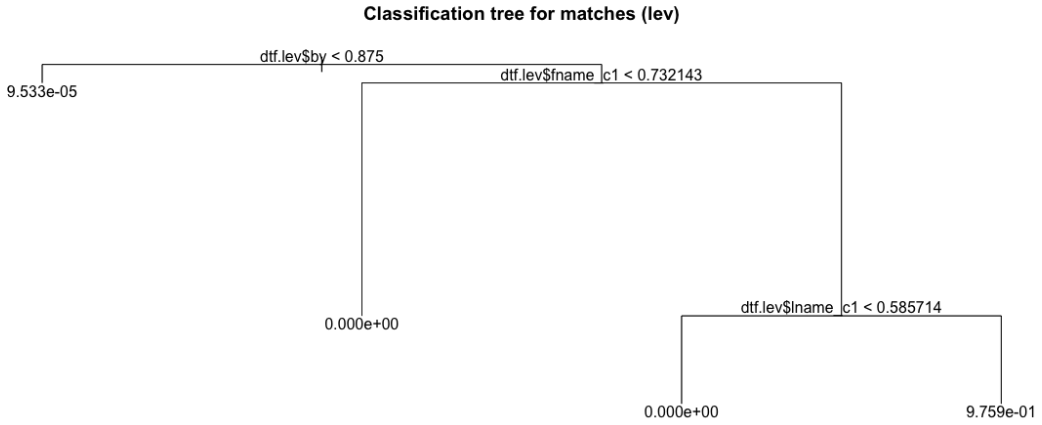

Figure 3.b: Classification tree plot for matches for Levenshtein model

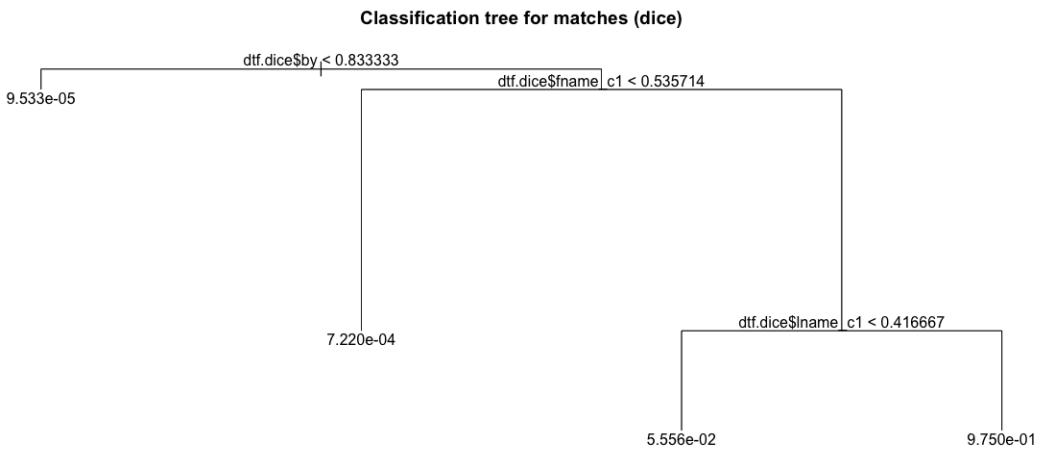

Figure 3.c: Classification tree plot for matches for Dice coefficient model

	True NM	True M
predicted NM	19850	8
predicted M	0	42

Figure 4: Error summary table of classification tree model for JW, Lev, Dice.

Fellegi-Sunter:

We define agreement as having a similarity score above 0.9. By using the first three fields (first name, last name, and birth year), we found the probabilities of the eight agreement patterns in the group of matches and in the group of non-matches with the help of known labels that is being derived from ID list. We are looking at the probability of the joint vector of match (1) and nonmathes (0) for all three fields at once. We then calculate the corresponding R values for each of the eight agreement patterns. Upper bound and lower bound are set to be 10 and 0.1, respectively. 10 is large enough to be the upper bound because it shows the probability of match given a specific pattern is 10 times of the probability of non-match. Similarly, 0.1 is low enough to be the lower bound as it shows the probability of nonmatch given a specific pattern is 10 times of the probability of match for that pattern. We use the thresholds to label our record pairs matches and non-matches, and compare them with the true labels. We considered assuming conditional independence for the agreement patterns in our study; however, it does not improve our model by much. Therefore, we continue with our model without this assumption.

For JW model, **Figure 5** shows that category 1-1-1 yield extremely large R-values and therefore would be categorized as matches. There is no clerical review in JW model. By comparing our assigned label with real label from train data, there are 3 false negative errors and 37 false positive errors. The false negative errors come from the 1-1-0 pattern (first name and last name match; birth year nonmatch) while the false positive errors come from 1-1-1 pattern (match for all three fields). The JW similarity model with test

data under Fellegi-Sunter has single false negative errors occurred at 1-1-0 and 4 false positive errors occurred at 1-1-1. The errors from the test data are of the same type as train data and they are resulted from the same patterns as those from train. A summary of the comparison is shown in **Figure 6a, b**.

-JW train:

	ratio	prob|NM	prob|M
0-0-0	0.00	0.10	0.00
0-0-1	0.00	0.00	0.00
0-1-0	0.00	0.86	0.00
0-1-1	0.00	0.02	0.00
1-0-0	0.00	0.00	0.00
1-0-1	0.00	0.00	0.00
1-1-0	1.94	0.02	0.03
1-1-1	3053.66	0.00	0.97

Figure 5: Agreement pattern of first name, last name, and birth year. A 1 indicates the field matches with at least 0.9 similarity, and 0 indicates other wise.

	predicted NM	predicted M
true NM	116757	37
true M	3	89

Figure 6a: Comparing our assigned label with real label for train data.

	predicted NM	predicted M
true NM	19846	4
true M	1	49

Figure 6b: Comparing our assigned label with real label for test data.

For Levenshtein model, **figure 7** shows that category 0-1-1, 1-1-0, and 1-1-1 yield large R-values (above 10) and therefore would be categorized as matches. There is no clerical review in this model. The Levenshtein similarity model with train data under Fellegi-Sunter has two false negative errors occurred at 0-1-0 (last name match, first name and birth year nonmatch) and 3057 false positive errors occurred at 0-1-1 (first name nonmatch, last name and birth year match), 1-1-0 (names match, birth year nonmatch), and 1-1-1(all match). The Levenshtein similarity model with test data under Fellegi-Sunter has single false negative error occurred at 0-1-0 and 419 false positive errors occurred at 0-1-1, 1-1-0, and 1-1-1. The errors from the test data are of the same type as train data and they are resulted from the same patterns as those from train. A summary of the comparison is shown in **Figure 8 a, b**. Note that the amount of false positive errors is probably due to the upper bound of 10. The false positive error would reduce if we raise the upper bound.

	ratio.lev	Prob| NM	Prob|M
0-0-0	0.00	0.10	0.00
0-0-1	0.00	0.00	0.00
0-1-0	0.03	0.87	0.02
0-1-1	29.71	0.01	0.37
1-0-0	0.00	0.00	0.00
1-0-1	0.00	0.00	0.00
1-1-0	16.75	0.01	0.23
1-1-1	3702.71	0.00	0.38

Figure7 : Agreement pattern of first name, last name, and birth year. A 1 indicates the field matches with at least 0.9 similarity, and 0 indicates other wise.

	predicted NM	predicted M
true NM	113737	3057
true M	2	90

Figure 8a: Comparing our assigned label with real label for train data.

	predicted NM	predicted M
true NM	19431	419
true M	1	49

Figure 8b: Comparing our assigned label with real label for test data.

For Dice model, **figure 9** shows that category, 0-1-1, 1-1-0, and 1-1-1 yield large R-values (above 10) and therefore would be categorized as matches. There is no clerical review in this model. The dice similarity model with train data under Fellegi-Sunter has two false negative error occurred at 0-1-0 (last name match, first name, birth year nonmatch) and 3153 false positive errors occurred at 0-1-1 (first name nonmatch, last name and birth year match), 1-1-0 (first and last name match, birth year nonmatch), and 1-1-1(all match). The test for dice similarity model under Fellegi-Sunter has one false negative error occurred at 0-1-0 and 435 false positive errors occurred at 0-1-1, 1-1-0, and 1-1-1. The errors from the test data are of the same type as train data and they are resulted from the same patterns as those from train. A summary of the comparison is shown in **Figure 10 a, b**.

	ratio.lev	Prob| NM	Prob|M
0-0-0	0.00	0.10	0.00
0-0-1	0.00	0.00	0.00
0-1-0	0.03	0.87	0.02
0-1-1	26.52	0.01	0.35
1-0-0	0.00	0.00	0.00
1-0-1	0.00	0.00	0.00
1-1-0	16.58	0.01	0.23
1-1-1	3613.19	0.00	0.40

Figure 9: Agreement pattern of first name, last name, and birth year. A 1 indicates the field matches with at least 0.9 similarity, and 0 indicates other wise.

	predicted NM	predicted M
true NM	113641	3153
true M	2	90

Figure 10a: Comparing our assigned label with real label for train data.

Dice test

	predicted NM	predicted M
true NM	19415	435
true M	1	49

Figure 10b: Comparing our assigned label with real label for train data.

When we compare across all three similarity metrics, it is not hard to tell JW outperform the other two with both the train data and test data. The errors from JW predictions are smaller than those of the other two similarity metrics. Therefore, the Fellegi-Sunter model suggests JW as the best model.

Random Forest:

	True NM	True M
predicted NM	19847	11
predicted M	3	39

Figure 11a: error summary of Jaro-Winkler for train data

	True NM	True M
predicted NM	19848	11
predicted M	2	39

Figure 11b: error summary of Levenshtein for train data

	True NM	True M
predicted NM	19848	14
predicted M	2	36

Figure 11c: error summary of dice-coefficient for train data

We trained random forest model by using train data of three similarity scores. We used the trained model to predict matches/nonmatches in test data. **Figure 11 a , b, c** shows the error summary output for each similarity metric. We used threshold of 0.8 when predicting matches/nonmatches in test data. For Jaro-Winkler, it has 11 false negatives and 3 false positives. For Levenshtein, it has 11 false negatives and 2 false positives. For dice coefficient, it has 14 false negatives and 2 false positives. Thus, Jaro-Winkler and Levenshtein predict the test data almost the same accurately.

Result:

	JW	Lev	Dice
Logistics	False negative: 2	Not converge	Not converge
Classification Tree	False negative: 8	False negative: 8	False negative: 8
Fellegi-Sunter	Train data- false negative: 3 false positive: 37 test data- false negative:1 false positive: 4	Train data- false negative: 2 false positive: 3057 test data- false negative:1 false positive: 419	Train data- false negative: 2 false positive: 3157 test data- false negative:1 false positive: 435
Random Forest	False negative: 11 False positive: 3	False negative: 11 False positive: 2	False negative: 14 False positive: 2

Figure 12: error summary of model prediction for JW, Lev, and Dice.

Fellegi-Sunter model suggests JW as the best model. One potential reason for this may be the 0.9 thresholds we chose for matches. The false positive error of Fellegi-Sunter model by Leventein and Dice are probably due to the upper bound of 10. The false positive error would reduce if you raise the upper bound. Since we use the same upper and lower bound for Fellegi-Sunter ratio, this error rate would not interfere with our purpose to compare similarity metrics. The Dice similarity scores are relatively lower when taking numbers as strings. For logistic regression, one potential explanation is that it takes bigrams while both birth month and birth date has only two digits. This may make the dice similarity score calculated for birth month/date matches inaccurate.

Discussion/Conclusions:

Through comparing error type of logistics regression, classification tree, Fellegi-Sunter, Random Forest test models under Jaro-Winkler, Leveshtein, and Dice coefficient similarity scores, we conclude that Jaro-Winkler distance and Levenshtein distance perform better than Sorensen dice-coefficient for our name-matching data. Jaro-Winkler distance performs slightly better than Levenshtein distance.

There are some areas we can improve in future studies. First, we may consider exact match for the birth year, month and dates. In addition, we can apply our models on other datasets to see how they perform over other relevant info for personal record, for example, home address.

References:

http://en.wikipedia.org/wiki/Category:String_similarity_measures
http://www.cs.columbia.edu/~cleslie/cs4761/papers/string-kernel-slides.pdf
http://en.wikipedia.org/wiki/S%C3%B8rensen%E2%80%93Dice_coefficient
http://journal.r-project.org/archive/2010-2/RJournal_2010-2_Sariyar+Borg.pdf
www.cs.cmu.edu/~wcohen/Matching-1.ppt

Sam Ventura
Prof. Nugent
Prof. Fienberg

Code:

```
library(RecordLinkage)
source("calc.pcs.R")
data(RLdata10000)
data(RLdata500)
true.10000 = identity.RLdata10000
true.500 = identity.RLdata500
my.data10000 = cbind(RLdata10000, true.10000)
my.data.sample = my.data10000[1000:3000, ]
my.data500 = cbind(RLdata500, true.500)

#################### Generate train data from RL500 and test data from RL10000
############
#################### Size of test data is 200 records with 100 matches ###############
###################### Size of the train data is 484 records with 184 matches ############
################################################################################
##############
############################ Test data ########################################
dup.or.not.500<-duplicated(true.500) | duplicated(true.500[length(true.500):1])[length(true.500):1]
match.500<- my.data500[which(dup.or.not.500==1), ]
nomatch.500<- my.data500[which(dup.or.not.500==0), ]
nomat.500<- nomatch.500[sample(seq(1:nrow(nomatch.500)),100, replace=FALSE), ]
#nomat.500<- nomatch.500[1:100, ]
data.test<- rbind(match.500, nomat.500)
colnames(data.test)<- colnames(data.train)
############################ Train data ###################
dup.or.not<-duplicated(my.data.sample$true) |
  duplicated(my.data.sample$true[length(my.data.sample$true):1])[length(my.data.sample$true):1
]
match<- my.data.sample[which(dup.or.not==1), ]
nomatch<- my.data.sample[which(dup.or.not==0), ]
#nomat<- nomatch[sample(seq(1:nrow(nomatch)),500, replace=FALSE), ]
nomat<- nomatch[1:300,]
data.train<- rbind(match, nomat)

################################################################################
################################################################################
dtf.lev = calc.pcs(data.train, type = c(rep("l", ncol(data.train)-1), "e"))

dtf.dice = calc.pcs(data.train, type = c(rep("d", ncol(data.train)-1), "e"))

dtf.jw = calc.pcs(data.train, type = c(rep("j", ncol(data.train)-1), "e"))

dtf.lev.test = calc.pcs(data.test, type = c(rep("l", ncol(data.train)-1), "e"))
dtf.dice.test = calc.pcs(data.test, type = c(rep("d", ncol(data.train)-1), "e"))
dtf.jw.test = calc.pcs(data.test, type = c(rep("j", ncol(data.train)-1), "e"))

#################### logistic regression ######
lr.lev<-
glm(true.10000.comp~fname_c1.comp+lname_c1.comp+by.comp+bm.comp+bd.comp,data=dtf.l
ev,family="binomial")
summary(lr.lev)$coef
# not converge
```

```
lr.dice<-
glm(true.10000.comp~fname_c1.comp+lname_c1.comp+by.comp+bm.comp+bd.comp,data=dtf.d
ice,family="binomial")
summary(lr.dice)$coef
# not converge
lr.jw<-
glm(true.10000~fname_c1.comp+lname_c1.comp+by.comp+bm.comp+bd.comp,data=dtf.jw,famil
y="binomial")
summary(lr.jw)$coef
length(predict(lr.jw, data=dtf.jw.test))

lr.jw.pred<-ifelse(exp(predict(lr.jw, data=dtf.jw.test))>0.5,1,0)
table(lr.jw.pred)
table(lr.jw.pred, dtf.jw.test$true)

#################### Classification tree ##############################
library(tree)
tree.jw<- tree(dtf.jw$true~dtf.jw$fname_c1+dtf.jw$lname_c1+dtf.jw$by+dtf.jw$bm+dtf.jw$bd)
plot(tree.jw); text(tree.jw); title("Classification tree for matches (jw)")
tree.lev<- tree(dtf.lev$true~dtf.lev$fname_c1+dtf.lev$lname_c1+dtf.lev$by+dtf.lev$bm+dtf.lev$bd)
plot(tree.lev); text(tree.lev); title("Classification tree for matches (lev)")
tree.dice<-
tree(dtf.dice$true~dtf.dice$fname_c1+dtf.dice$lname_c1+dtf.dice$by+dtf.dice$bm+dtf.dice$bd)
plot(tree.dice); text(tree.dice); title("Classification tree for matches (dice)")

pred.tree.jw<-predict(tree.jw,data=dtf.jw.test)
match.model.jw<-ifelse(pred.tree.jw>=0.9,1,0)
#error.tree.jw<- (sum(match.model.jw)-sum(dtf.jw.test$true))/sum(dtf.jw.test$true)
table(match.model.jw)


pred.tree.lev<-predict(tree.lev,data=dtf.lev.test)
match.model.lev<-ifelse(pred.tree.lev>=0.9,1,0)
#error.tree.lev<- (sum(match.model.lev)-sum(dtf.lev.test$true))/sum(dtf.lev.test$true)

pred.tree.dice<-predict(tree.dice,data=dtf.dice.test)
match.model.dice<-ifelse(pred.tree.dice>=0.9,1,0)
#error.tree.dice<- (sum(match.model.dice)-sum(dtf.dice.test$true))/sum(dtf.dice.test$true)

################## Fellegi Suntner assuming conditional independence ##################
library(xtable)
#for jw
agree <- matrix(NA, nrow(dtf.jw), 3)
for (jj in 1:3) agree[, jj] <- 1 * (dtf.jw[, jj] > 0.9)
patterns <- apply(agree, 1, paste, collapse = "-")
match <- dtf.jw$true
probs <- table(patterns, match)
probs[, 1] <- probs[, 1]/sum(probs[, 1])
probs[, 2] <- probs[, 2]/sum(probs[, 2])
ratio <- round(probs[, 2]/probs[, 1], 4)
colnames(probs) <- c("prob | NM", "prob | M")
xtable(cbind(ratio, round(probs, 4)))
### only 1-1-1 yield extreme values; so they are assigned match
decision <- ifelse(patterns == "1-1-1", 1, 0)
xt1 <- table(match, decision)
rownames(xt1) <- c("true NM", "true M")
```

```
colnames(xt1) <- c("predicted NM", "predicted M")
xtable(xt1)
table(match, decision, patterns)
#this breaks it down by pattern and shows where the
# errors occur

#test
agree.t <- matrix(NA, nrow(dtf.jw.test), 3)
for (jj in 1:3) agree.t[, jj] <- 1 * (dtf.jw.test[, jj] > 0.9)
patterns.t <- apply(agree.t, 1, paste, collapse = "-")
match.t <- dtf.jw.test$true
probs.t <- table(patterns.t, match.t)
probs.t[, 1] <- probs.t[, 1]/sum(probs.t[, 1])
probs.t[, 2] <- probs.t[, 2]/sum(probs.t[, 2])
ratio.t <- round(probs.t[, 2]/probs.t[, 1], 4)
colnames(probs.t) <- c("prob | NM", "prob | M")
xtable(cbind(ratio.t, round(probs.t, 4)))
### only 1-1-1 yield extreme values; so they are assigned match
decision.t <- ifelse(patterns.t == "1-1-1", 1, 0)
xt1.t <- table(match.t, decision.t)
rownames(xt1.t) <- c("true NM", "true M")
colnames(xt1.t) <- c("predicted NM", "predicted M")
xtable(xt1.t)
table(match.t, decision.t, patterns.t)


# lev
agree.lev <- matrix(NA, nrow(dtf.lev), 3)
for (jj in 1:3) agree.lev[, jj] <- 1 * (dtf.lev[, jj] > 0.9)
patterns.lev <- apply(agree.lev, 1, paste, collapse = "-")
match.lev <- dtf.lev$true
probs.lev <- table(patterns.lev, match.lev)
probs.lev[, 1] <- probs.lev[, 1]/sum(probs.lev[, 1])
probs.lev[, 2] <- probs.lev[, 2]/sum(probs.lev[, 2])
ratio.lev <- round(probs.lev[, 2]/probs.lev[, 1], 4)
colnames(probs.lev) <- c("prob | NM", "prob | M")
xtable(cbind(ratio.lev, round(probs.lev, 4)))
### only 1-1-1 yield extreme values; so they are assigned match
decision.lev <- ifelse(patterns.lev == "0-1-1"|patterns.lev == "1-1-0"|patterns.lev == "1-1-1", 1, 0)
xt1.lev <- table(match.lev, decision.lev)
rownames(xt1.lev) <- c("true NM", "true M")
colnames(xt1.lev) <- c("predicted NM", "predicted M")
xtable(xt1.lev)
table(match.lev, decision.lev, patterns.lev)
#test
agree.lev.t <- matrix(NA, nrow(dtf.lev.test), 3)
for (jj in 1:3) agree.lev.t[, jj] <- 1 * (dtf.lev.test[, jj] > 0.9)
patterns.lev.t <- apply(agree.lev.t, 1, paste, collapse = "-")
match.lev.t <- dtf.lev.test$true
probs.lev.t <- table(patterns.lev.t, match.lev.t)
probs.lev.t[, 1] <- probs.lev.t[, 1]/sum(probs.lev.t[, 1])
probs.lev.t[, 2] <- probs.lev.t[, 2]/sum(probs.lev.t[, 2])
ratio.lev.t<- round(probs.lev.t[, 2]/probs.lev.t[, 1], 4)
colnames(probs.lev.t) <- c("prob | NM", "prob | M")
xtable(cbind(ratio.lev.t, round(probs.lev.t, 4)))
### only 1-1-1 yield extreme values; so they are assigned match
```

```
decision.lev.t <- ifelse(patterns.lev.t == "0-1-1"|patterns.lev.t == "1-1-0"|patterns.lev.t == "1-1-1",
1, 0)
xt1.lev.t <- table(match.lev.t, decision.lev.t)
rownames(xt1.lev.t) <- c("true NM", "true M")
colnames(xt1.lev.t) <- c("predicted NM", "predicted M")
xtable(xt1.lev.t)
table(match.lev.t, decision.lev.t, patterns.lev.t)

# Dice
agree.dice <- matrix(NA, nrow(dtf.dice), 3)
for (jj in 1:3) agree.dice[, jj] <- 1 * (dtf.dice[, jj] > 0.9)
patterns.dice <- apply(agree.dice, 1, paste, collapse = "-")
match.dice <- dtf.dice$true
probs.dice <- table(patterns.dice, match.dice)
probs.dice[, 1] <- probs.dice[, 1]/sum(probs.dice[, 1])
probs.dice[, 2] <- probs.dice[, 2]/sum(probs.dice[, 2])
ratio.dice <- round(probs.dice[, 2]/probs.dice[, 1], 4)
colnames(probs.dice) <- c("prob | NM", "prob | M")
xtable(cbind(ratio.dice, round(probs.dice, 4)))
decision.dice <- ifelse(patterns.dice == "0-1-1"|patterns.dice == "1-1-0"|patterns.dice == "1-1-1",
1, 0)
xt1.dice <- table(match.dice, decision.dice)
rownames(xt1.dice) <- c("true NM", "true M")
colnames(xt1.dice) <- c("predicted NM", "predicted M")
xtable(xt1.dice)
table(match.dice, decision.dice, patterns.dice)

#test
agree.dice.t <- matrix(NA, nrow(dtf.dice.test), 3)
for (jj in 1:3) agree.dice.t[, jj] <- 1 * (dtf.dice.test[, jj] > 0.9)
patterns.dice.t <- apply(agree.dice.t, 1, paste, collapse = "-")
match.dice.t <- dtf.dice.test$true
probs.dice.t <- table(patterns.dice.t, match.dice.t)
probs.dice.t[, 1] <- probs.dice.t[, 1]/sum(probs.dice.t[, 1])
probs.dice.t[, 2] <- probs.dice.t[, 2]/sum(probs.dice.t[, 2])
ratio.dice.t <- round(probs.dice.t[, 2]/probs.dice.t[, 1], 4)
colnames(probs.dice.t) <- c("prob | NM", "prob | M")
xtable(cbind(ratio.dice.t, round(probs.dice.t, 4)))
decision.dice.t <- ifelse(patterns.dice.t == "0-1-1"|patterns.dice.t == "1-1-0"|patterns.dice.t == "1-
1-1", 1, 0)
xt1.dice.t <- table(match.dice.t, decision.dice.t)
rownames(xt1.dice.t) <- c("true NM", "true M")
colnames(xt1.dice.t) <- c("predicted NM", "predicted M")
xtable(xt1.dice.t)
table(match.dice.t, decision.dice.t, patterns.dice.t)


# assuming conditional independence
ap.jw<-dtf.jw; ap.lev<-dtf.lev; ap.dice<-dtf.dice
ap.jw$fname_c1 <-ifelse(ap.jw$fname_c1 >= 0.90, 1, 0)
ap.jw$lname_c1 <-ifelse(ap.jw$lname_c1 >= 0.90, 1, 0); ap.jw$by <-ifelse(ap.jw$by >= 0.90, 1, 0)

ap.lev$fname_c1 <-ifelse(ap.lev$fname_c1 >= 0.90, 1, 0)
ap.lev$lname_c1 <-ifelse(ap.lev$lname_c1 >= 0.90, 1, 0); ap.lev$by <-ifelse(ap.lev$by >= 0.90,
1, 0)
```

```
ap.dice$fname_c1 <-ifelse(ap.dice$fname_c1 >= 0.90, 1, 0)
ap.dice$lname_c1 <-ifelse(ap.dice$lname_c1 >= 0.90, 1, 0); ap.dice$by <-ifelse(ap.dice$by >=
0.90, 1, 0)

Rs<-function(matrix){
  g.m<-matrix[matrix$true==1,];m<-nrow(g.m)
  g.u<-matrix[matrix$true==0,];u<-nrow(g.u)
  R111<-((length(which(g.m$fname_c1 == 1))/m)*(length(which(g.m$lname_c1 ==
1))/m)*(length(which(g.m$by == 1))/m))/
    ((length(which(g.u$fname_c1 == 1))/u)*(length(which(g.u$lname_c1 ==
1))/u)*(length(which(g.u$by == 1))/u))
  R110<-((length(which(g.m$fname_c1 == 1))/m)*(length(which(g.m$lname_c1 ==
1))/m)*(length(which(g.m$by == 0))/m))/
    ((length(which(g.u$fname_c1 == 1))/u)*(length(which(g.u$lname_c1 ==
1))/u)*(length(which(g.u$by == 0))/u))
  R101<-((length(which(g.m$fname_c1 == 1))/m)*(length(which(g.m$lname_c1 ==
0))/m)*(length(which(g.m$by == 1))/m))/
    ((length(which(g.u$fname_c1 == 1))/u)*(length(which(g.u$lname_c1 ==
0))/u)*(length(which(g.u$by == 1))/u))
  R100<-((length(which(g.m$fname_c1 == 1))/m)*(length(which(g.m$lname_c1 ==
0))/m)*(length(which(g.m$by == 0))/m))/
    ((length(which(g.u$fname_c1 == 1))/u)*(length(which(g.u$lname_c1 ==
0))/u)*(length(which(g.u$by == 0))/u))
  R011<-((length(which(g.m$fname_c1 == 0))/m)*(length(which(g.m$lname_c1 ==
1))/m)*(length(which(g.m$by == 1))/m))/
    ((length(which(g.u$fname_c1 == 0))/u)*(length(which(g.u$lname_c1 ==
1))/u)*(length(which(g.u$by == 1))/u))
  R010<-((length(which(g.m$fname_c1 == 0))/m)*(length(which(g.m$lname_c1 ==
1))/m)*(length(which(g.m$by == 0))/m))/
    ((length(which(g.u$fname_c1 == 0))/u)*(length(which(g.u$lname_c1 ==
1))/u)*(length(which(g.u$by == 0))/u))
  R001<-((length(which(g.m$fname_c1 == 0))/m)*(length(which(g.m$lname_c1 ==
0))/m)*(length(which(g.m$by == 1))/m))/
    ((length(which(g.u$fname_c1 == 0))/u)*(length(which(g.u$lname_c1 ==
0))/u)*(length(which(g.u$by == 1))/u))
  R000<-((length(which(g.m$fname_c1 == 0))/m)*(length(which(g.m$lname_c1 ==
0))/m)*(length(which(g.m$by == 0))/m))/
    ((length(which(g.u$fname_c1 == 0))/u)*(length(which(g.u$lname_c1 ==
0))/u)*(length(which(g.u$by == 0))/u))
  R<- c(R111,R110,R101,R100,R011,R010,R001,R000); plot(R);hist(R)
  return (R)
}

R.jw<-Rs(ap.jw)
R.lev<-Rs(ap.lev)
R.dice<-Rs(ap.dice)

label<-function(r, lo, up){
  label=NA
  if (r<=lo){label=0}
  else if (r>=up){label=1}
  else if (r>lo & r<up) {label="CR"}
  return (label)
}

up<-10; lo<- 0.1
```

```
lab.type<-function(R){
  R111.lab<-label(R[1], lo, up);R110.lab<-label(R[2], lo, up); R101.lab<-label(R[3], lo, up)
  R100.lab<-label(R[4], lo, up);R010.lab<-label(R[5], lo, up); R011.lab<-label(R[6], lo, up)
  R001.lab<-label(R[7], lo, up);R000.lab<-label(R[8], lo, up)
  R.lab<- c(R111.lab, R110.lab, R101.lab, R100.lab, R010.lab, R011.lab, R001.lab,R000.lab)
  return (R.lab)
}

jw.lab<-lab.type(R.jw); lev.lab<-lab.type(R.lev); dice.lab<-lab.type(R.dice)

vec.label<-function(matrix, R.lab){
  nn<-nrow(matrix)
  vec<- rep(NA, nn)
  for (i in 1:nn){
    if (matrix$fname_c1[i]==1 & matrix$lname_c1[i]==1 & matrix$by[i]==1){vec[i]=R.lab[1]}
    else if (matrix$fname_c1[i]==1 & matrix$lname_c1[i]==1 & matrix$by[i]==0){vec[i]=R.lab[2]}
    else if (matrix$fname_c1[i]==1 & matrix$lname_c1[i]==0 & matrix$by[i]==1){vec[i]=R.lab[3]}
    else if (matrix$fname_c1[i]==1 & matrix$lname_c1[i]==0 & matrix$by[i]==0){vec[i]=R.lab[4]}
    else if (matrix$fname_c1[i]==0 & matrix$lname_c1[i]==1 & matrix$by[i]==0){vec[i]=R.lab[5]}
    else if (matrix$fname_c1[i]==0 & matrix$lname_c1[i]==1 & matrix$by[i]==1){vec[i]=R.lab[6]}
    else if (matrix$fname_c1[i]==0 & matrix$lname_c1[i]==0 & matrix$by[i]==1){vec[i]=R.lab[7]}
    else if (matrix$fname_c1[i]==0 & matrix$lname_c1[i]==0 & matrix$by[i]==0){vec[i]=R.lab[8]}
  }
  return (vec)
}

jw.vec.lab<-vec.label(ap.jw, jw.lab)
lev.vec.lab<-vec.label(ap.lev,lev.lab)
dice.vec.lab<-vec.label(ap.dice, dice.lab)
###### error rate (without CR)
table.jw<-table(jw.vec.lab[jw.vec.lab!="CR"] == dtf.jw$true[jw.vec.lab!="CR"])
table(jw.vec.lab[jw.vec.lab!="CR"], dtf.jw$true[jw.vec.lab!="CR"])
jw.error<-table.jw[1]/(table.jw[1]+table.jw[2])
table(jw.vec.lab,dtf.jw$true)
table(jw.vec.lab == dtf.jw$true)

table.lev<-table(lev.vec.lab[lev.vec.lab!="CR"] == dtf.lev$true[lev.vec.lab!="CR"])
table(lev.vec.lab[lev.vec.lab!="CR"], dtf.lev$true[lev.vec.lab!="CR"])
lev.error<-table.lev[1]/(table.lev[1]+table.lev[2])
table(lev.vec.lab,dtf.lev$true)
table(lev.vec.lab == dtf.lev$true)

table.dice<-table(dice.vec.lab[dice.vec.lab!="CR"] == dtf.dice$true[dice.vec.lab!="CR"])
table(dice.vec.lab[dice.vec.lab!="CR"], dtf.dice$true[dice.vec.lab!="CR"])
dice.error<-table.dice[1]/(table.dice[1]+table.dice[2])
table(dice.vec.lab, dtf.dice$true)
table(dice.vec.lab == dtf.dice$true)

####### predict #################
ap.jw.test<-dtf.jw.test; ap.lev.test<-dtf.lev.test; ap.dice.test<-dtf.dice.test
ap.jw.test$fname_c1 <-ifelse(ap.jw.test$fname_c1 >= 0.90, 1, 0)
ap.jw.test$lname_c1 <-ifelse(ap.jw.test$lname_c1 >= 0.90, 1, 0)
ap.jw.test$by <-ifelse(ap.jw.test$by >= 0.90, 1, 0)

ap.lev.test$fname_c1 <-ifelse(ap.lev.test$fname_c1 >= 0.90, 1, 0)
```

```
ap.lev.test$lname_c1 <-ifelse(ap.lev.test$lname_c1 >= 0.90, 1, 0)
ap.lev.test$by <-ifelse(ap.lev.test$by >= 0.90, 1, 0)

ap.dice.test$fname_c1 <-ifelse(ap.dice.test$fname_c1 >= 0.90, 1, 0)
ap.dice.test$lname_c1 <-ifelse(ap.dice.test$lname_c1 >= 0.90, 1, 0)
ap.dice.test$by <-ifelse(ap.dice.test$by >= 0.90, 1, 0)

jw.vec.lab.test<-vec.label(ap.jw.test, jw.lab)
lev.vec.lab.test<-vec.label(ap.lev.test,lev.lab)
dice.vec.lab.test<-vec.label(ap.dice.test, dice.lab)

table.jw.test<-table(jw.vec.lab.test[jw.vec.lab.test!="CR"] ==
dtf.jw.test$true[jw.vec.lab.test!="CR"])
table(jw.vec.lab.test[jw.vec.lab.test!="CR"], dtf.jw.test$true[jw.vec.lab.test!="CR"])
jw.error.test<-table.jw.test[1]/(table.jw.test[1]+table.jw.test[2])

table.lev.test<-table(lev.vec.lab.test[lev.vec.lab.test!="CR"] ==
dtf.lev.test$true[lev.vec.lab.test!="CR"])
table(lev.vec.lab.test[lev.vec.lab.test!="CR"], dtf.lev.test$true[lev.vec.lab.test!="CR"])
lev.error.test<-table.lev.test[1]/(table.lev.test[1]+table.lev.test[2])

table.dice.test<-table(dice.vec.lab.test[dice.vec.lab.test!="CR"] ==
dtf.dice.test$true[dice.vec.lab.test!="CR"])
table(dice.vec.lab.test[dice.vec.lab.test!="CR"], dtf.dice.test$true[dice.vec.lab.test!="CR"])
dice.error.test<-table.dice.test[1]/(table.dice.test[1]+table.dice.test[2])

#Random Forest
data.train=my.data.sample
dtf.lev = calc.pcs(data.train, type = c(rep("l", ncol(data.train)-1), "e"))
nm.row = which(dtf.lev$true.10000.comp == 0)   # sam's code on random generating data
##nm.row.sample = sample(nm.row, 100)
nm.row.sample = nm.row[1:100]
m.row = which(dtf.lev$true.10000.comp == 1)
new.rows = c(m.row, nm.row.sample)
dtf.lev = dtf.lev[new.rows,]
table(dtf.lev$true.10000.comp)

dtf.dice = calc.pcs(data.train, type = c(rep("d", ncol(data.train)-1), "e"))
dtf.dice = dtf.dice[new.rows,]
table(dtf.dice$true.10000.comp)

dtf.jw = calc.pcs(data.train, type = c(rep("j", ncol(data.train)-1), "e"))
dtf.jw = dtf.jw[new.rows,]
table(dtf.jw$true.10000.comp)

library(randomForest)
dtf.jw$true.10000.comp = as.factor(dtf.jw$true.10000.comp)
ran.f.jw<-randomForest(true.10000.comp~
  fname_c1.comp+lname_c1.comp, data=dtf.jw,
                  na.action=na.omit, mtry = 2, importance = TRUE)
preds.jw = as.numeric(predict(ran.f.jw, newdata = dtf.jw.test, type = "prob")[,2])
print(ran.f.jw)
plot(ran.f.jw)

dtf.lev$true.10000.comp = as.factor(dtf.lev$true.10000.comp)
ran.f.lev<-randomForest(true.10000.comp~
```

```
  fname_c1.comp+lname_c1.comp, data=dtf.lev,
                       na.action=na.omit, mtry = 2, importance = TRUE)
preds = as.numeric(predict(ran.f.lev, newdata = dtf.lev.test, type = "prob")[,2])
print(ran.f.lev)
plot(ran.f.lev)

dtf.dice$true.10000.comp = as.factor(dtf.dice$true.10000.comp)
ran.f.dice<-randomForest(true.10000.comp~
  fname_c1.comp+lname_c1.comp, data=dtf.dice,
                        na.action=na.omit, mtry = 2, importance = TRUE)
preds.dice = as.numeric(predict(ran.f.dice, newdata = dtf.dice.test, type = "prob")[,2])
print(ran.f.dice)
plot(ran.f.dice)
```