Mini Project Article

Caleb Woo

2024-10-01

Motivation and Goals

In this article, we will explore various string distance metrics and conduct an experiment comparing the efficacy of some popular metrics on the task of matching entity names. We will briefly summarize some metrics that Cohen, Ravikumar, and Fienberg discussed in their paper. Inspired by these authors, we will conduct a similar experiment comparing how well different string distance metrics perform on matching entity names in an example data set.

Cohen, Ravikumar, and Fienberg

Cohen, Ravikumar, and Fienberg published a paper in 2003 called *A Comparison of String Distance Metrics for Name-Matching Tasks*. At the time, they recently implemented an open-source, Java toolkit with a variety of string distance metrics for numerous name-matching methods. They compared how the different methods in their toolkit performed on the task of matching entity names across numerous different data sets. Academic communities such as statistics, databases, and artificial intelligence have proposed different methods for the common problem of matching entity names throughout the years. Cohen, Ravikumar, and Fienberg conducted experiments to provide direct and intuitive comparisons of some of the most popular string distance metrics among the numerous communities.

String Distance Metrics

String distance metrics refer to both distance functions and similarity functions. Although they encode analogous information, their interpretations are reversed. For a given pair of strings that are very similar, they would be assigned a small distance value and a large similarity value. Correspondingly, for very different strings, they would be assigned a large distance value and a small similarity value.

Below, we will explore three broad categories of metrics: edit-distance, token-based, and hybrid. First, we will summarize some popular examples of metrics in these categories that Cohen, Ravikumar, and Fienberg discussed in their paper. Afterwards, we will conduct our own experiment comparing how some of these metrics perform on the task of matching entity names in an example data set.

Edit-distance

Edit-distance metrics are calculated as the "cost" of the best order of edits to one string to make it identical to the other string. Edits can include character insertion, deletion, and substitution where each is assigned a particular cost. The **Levenstein distance** is a simple example which assigns a unit cost to all edits. The **Monger-Elkan** distance is a more complex example which assigns a particular cost $\in [0, 1]$ for each edit where insertions and deletions are assigned relatively lower costs.

Although not an edit-distance metric, the **Jaro** metric takes a broadly similar approach by calculating a score based on the count and sequence of the common characters for a pair of strings. [Jaro metric elaboration?].

The **Jaro-Winkler** metric is an extension which incorporates the size of the longest common prefix for a pair of strings.

Token-based

Token-based metrics treat each string as a multiset of characters or words called tokens. The **Jaccard similarity** is a simple example which is calculated as the size of the intersection of a pair of strings divided by the size of the union. The **cosine similarity** is another popular example. [cosine similarity elaboration?]

Hybrid

Hybrid metrics calculate scores by combining multiple existing string distance metrics. Monge and Elkan's **level two** metric calculates a linear combination of a chosen metric calculated on the sub strings for a pair of strings. Cohen, Ravikumar, and Fienberg experimented with using the Monge-Elkan distance, Jaro metric, and Jaro-Winkler metric in their implementations. **"Soft" cosine similarity** includes similar tokens instead of only identical tokens in the intersection of a pair of strings and combines a secondary distance. Cohen, Ravikumar, and Fienberg used the Jaro-Winkler metric as a secondary distance in their implementation. They also developed their own hybrid metric by training a **binary SVM classifier** with input features being the scores from various existing metrics including the Monge-Elkan distance, the Jaro-Winkler metric, and the cosine similarity.

Experiment

In our experiment below, we will explore 5 metrics using implementations available in the `stringdist` R package for some of the popular metrics discussed above. We will compare how these 5 metrics perform on the task of matching entity names in an example data set from the `RecordLinkage` R package.

2 are edit-distance metrics: - Levenstein distance - Jaro-Winkler distance

2 are token-based metrics: - Jaccard similarity - Cosine similarity

Our 5th metric is a binary classifier using the above 4 metrics as input features

Tentative Plan: - Make an experimental hybrid metric by training some binary classifier (logistic regression, boosting?) using the 4 metrics as features - 5-fold cross validation, in each fold: - Train classifier on other 4 folds - Get classifier predictions on test fold - Get predictions from each of the 4 metrics on test fold - Record true matches in test fold - Compare cross validation results between classifier and the 4 metrics - F1 score - Precision Recall curve

```
library(tidyverse)
library(RecordLinkage)
library(stringdist)
library(caret)

COURSE_NUMBER = 790

data("RLdata500")
temp_data = cbind(RLdata500, id = identity.RLdata500)

rm(RLdata500)
rm(identity.RLdata500)
```

```
temp_data = temp_data %>%
  mutate(
    fname = case_when(
      !is.na(fname_c2) ~ paste0(fname_c1, "-", fname_c2),
      .default = fname_c1
    ),
    lname = case_when(
      !is.na(lname_c2) ~ paste0(lname_c1, "-", lname_c2),
      .default = lname_c1
    ),
    bday = paste0(bm, "-", bd, "-", by)
    )
```

```
n = nrow(temp_data)
ents = paste0(temp_data$fname, " ", temp_data$lname, "; ", temp_data$bday)

FOLDS = 5
set.seed(COURSE_NUMBER)

df = as.data.frame(cbind(
  ent = ents,
  id = temp_data$id,
  fold = createFolds(ents, k = FOLDS, list = F)
  ))
```

```
fold = c()
ns_fold = c()
precision_fold = c()
recall_fold = c()
f1_fold = c()

for (f in 1:FOLDS) {
  temp_df = df %>%
    filter(fold == f)

  n_fold = nrow(temp_df)
  scores = c()
  responses = c()

  for (i in 1:n_fold) {
    for (j in 1:nrow(df)) {
      score = round(stringdist(temp_df[i, "ent"], df[j, "ent"], method = "jaccard"))
      response = as.integer(temp_df[i, "id"] == df[j, "id"])

      scores = c(scores, score)
      responses = c(responses, response)
    }
  }

  fold = c(fold, f)
  ns_fold = c(ns_fold, n_fold)

  precision_fold = c(precision_fold, precision(factor(scores), factor(responses)))
```

```
  recall_fold = c(recall_fold, recall(factor(scores), factor(responses)))
  f1_fold = c(f1_fold, F_meas(factor(scores), factor(responses)))
}
```

```
as.data.frame(cbind(
  Fold = fold,
  n = ns_fold,
  Precision = precision_fold,
  Recall = recall_fold,
  F1 = f1_fold
  ))
```

```
##   Fold   n Precision    Recall        F1
## 1    1 107 0.9947088 0.4402623 0.6103716
## 2    2  94 0.9941798 0.4153265 0.5858920
## 3    3 105 0.9948926 0.4537210 0.6232217
## 4    4 105 0.9936933 0.3971509 0.5674916
## 5    5  89 0.9944207 0.4295948 0.5999906
```