Project 4 - Entity Resolution

Team 11

04/14/2017

Summary:

In this project, we implemented, evaluated and compared the algorithms in paper 1 : Information Processing and Management (Kang 2009), and paper 5: Author Disambiguation using Error-driven Machine Learning with a Ranking Loss Function(Culotta 2007) for Entity Resolution. We created an author disambiguation system that divides the same-name author occurrences in citation data into different clusters, each of which are expected to correspond to a real individual. We used hierarchical clustering for both papers. In addition, we implemented Cluster Scoring Function, Error-driven Online Training, and Ranking MIRA. After comparing these two methods, we find out that for large dataset, paper 5 performed better(higher f1 and accuracy) than Paper1 did. For instance, algorithm in paper 5 offered much better results than paper 1 when tackling datasets of which author names are 'CChen', 'JLee', 'JSmith', and 'SLee'with the number of observations 801, 1419, 927, and 1464 respectively.

Note: This is the final result notebook. If you want to read all the codes, please go to doc/TrainAndPredict.ipynb and lib/.

Step 0: Load the packages, specify directories

```
if (!require("pacman")) install.packages("pacman")
```

```
## Loading required package: pacman
```

```
pacman::p_load(text2vec, plyr,qlcMatrix, kernlab, knitr)
setwd("~/Spr2017-proj4-team-11/doc")
library(ggplot2)
```

```
##
## Attaching package: 'ggplot2'

## The following object is masked from 'package:kernlab':
##
##     alpha
```

Step 1: Load and process the data

For each record in the dataset, there are some information we want to extract and store them in a regular form: canonical author id, coauthors, paper title, publication venue title.

After generated a list of 14 elements using Professor Zheng's code, we reorganizes it into a list of 14 dataframes for easier access and processing.

```
source("../lib/dataclean.R")
```

Step 2: Feature design

Paper 1 : Following the section 5.2, that each name occurrence is represented by a set of his/her coauthor names. We count the number of matched coauthors between two authors.

Paper 5: We want to use coauthors, paper titles and journey titles to design features for citations.

- **TF-IDF and cosine similarity: Term Frequency/Inverse Document Frequency weighting**

- **Same-Coauthor occurrences**

- **Edit distance: Compute the approximate string distance between character vectors. The distance is a generalized Levenshtein (edit) distance, giving the minimal possibly weighted number of insertions, deletions and substitutions needed to transform one string into another**

- **Bigram and Trigram: Count the Frequency of Pairs/Triple characters**

- **Journey Title Similarity**

```
# source("../lib/coauthormatrix.R")
load("../data/sim_matrix.RData")
```

For paper 5, we have r2py.RMD and PreprocessData.ipynb in doc/ to process RData. R files for feature extractions are in lib/

Step 3: Clustering

We used a hierarchical clustering method for both paper 1 and paper 5. The algorithm also follows section 5.2 in paper 1.

Algorithm 2

Agglomerative Clustering for Same-name Author Occurrences

Input:
$a_1,\ldots,a_n$; same-name author occurrences
$a_i = \{v_{i1},\ldots,v_{im}\}$; each name occurrence a_i has a set of m ($m \geqslant 0$) his/her coauthor names
θ; a cluster-merging threshold

Initialize:
$c_i = \{a_i\}$; consider each name occurrence a_i as an element of cluster c_i

Loop:
DO
  For each cluster-pair (c_i, c_j), calculate $CSim(c_i,c_j)$
    $CSim(c_i,c_j) = \max(ASim(a_x, a_y))$, $\forall a_x \in c_i$, $\forall a_y \in c_j$
    $ASim(a_x,a_y) = |a_x \cap a_y|$
  Find the most similar cluster-pair (c_u,c_v)
    $(c_u,c_v) = \text{argmax } CSim(c_i, c_j)$
  IF $CSim(c_u,c_v) \geqslant \theta$ THEN
    $c_{u\_v} = c_u \cup c_v$; merge c_u and c_v into a new larger cluster c_{u_v}
  ENDIF
WHILE $(CSim(c_u,c_v) \geqslant \theta)$

Output:
Clusters of author occurrences: $\{c_k\}$

We set the number of overlapping coauthors to 1.

```
source("../lib/singlelink.R")
start.time <- Sys.time()
cluster_temp.list <- NULL
cluster_temp.list <- llply(simmatrix.list,singlecluster,theta=1)
cluster.combined <- NULL
cluster.combined <- llply(cluster_temp.list,comninecluster)
cluster.notcombined <- NULL
cluster.notcombined <- llply(cluster_temp.list,splitcluster)
end.time <- Sys.time()
time_scluter <- end.time - start.time
time_scluter
```

```
## Time difference of 17.73878 mins
```

We also considered two scenarios : all the single-element cluster are combined; or we don't combine them. Here, I only showed the cluster result of a subset of the data, "AGupta.txt" to illustrate the difference between these two scenarios.

```
# combind cluster table for AGupta
table(cluster.combined[[1]])
```

```
##
##   0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
##  62 114  58  24  24  24  23  22  19  16  15  14  13  12  11   9   7   7
##  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35
##   7   6   5   5   4   4   4   4   3   3   3   3   3   3   3   3   2   2
##  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53
##   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
```

```
# do not combine single-element cluster
table(cluster.notcombined[[1]])
```

```
##
##   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
## 114  58  24  24  24  23  22  19  16  15  14  13  12  11   9   7   7   7
##  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36
##   6   5   5   4   4   4   4   3   3   3   3   3   3   3   3   2   2   2
##  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54
##   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   1
##  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72
##   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
##  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90
##   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
##  91  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106 107 108
##   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
## 109 110 111 112 113 114 115
##   1   1   1   1   1   1   1
```

In first table, cluster "0" has 62 elements, and it means we combined 62 single-element clusters together.

Paper 5:

Procedure:

- **An initial guess, λ, assigns weights to each feature. We used odds ratio as our initial guess.**

	coauthor	journal	tfidf_sml	edit_dist	edit_dist_sml	bigram	trigram
is_same							
0	0.001182	0.069144	0.019433	0.302339	0.211198	0.000049	0.000007
1	0.170841	0.169042	0.086875	0.286961	0.247866	0.005399	0.003545

- **Error-driven Trainging: We use hierarchical clustering. After first iteration, we have a partition T hat. We calculate a score S and a true socre(accuracy) for our partition. T star is a partition with higher true score in the neigborhood of T hat, and the existence of T star means that T hat is not the best partition and therefore we need to update λ.**

- **Ranking MIRA: We update λ with three constratins. It is also the tuning part that we choose τ to be 0.04 and margin to be 0.0001. The optimization is solved using Python through the quadratic program.**

- **After iterations and updates, we have our final λ**

Step 4: Evaluation

To evaluate the performance of the method, it is required to calculate the degree of agreement between a set of system-output partitions and a set of true partitions. In general, the agreement between two partitioins is measured for a pair of entities within partitions. The basic unit for which pair-wise agreement is assessed is a pair of entities (authors in our case) which belongs to one of the four cells in the following table (Kang et at.(2009)):

Matching matrix for the agreement between two sets of clusters

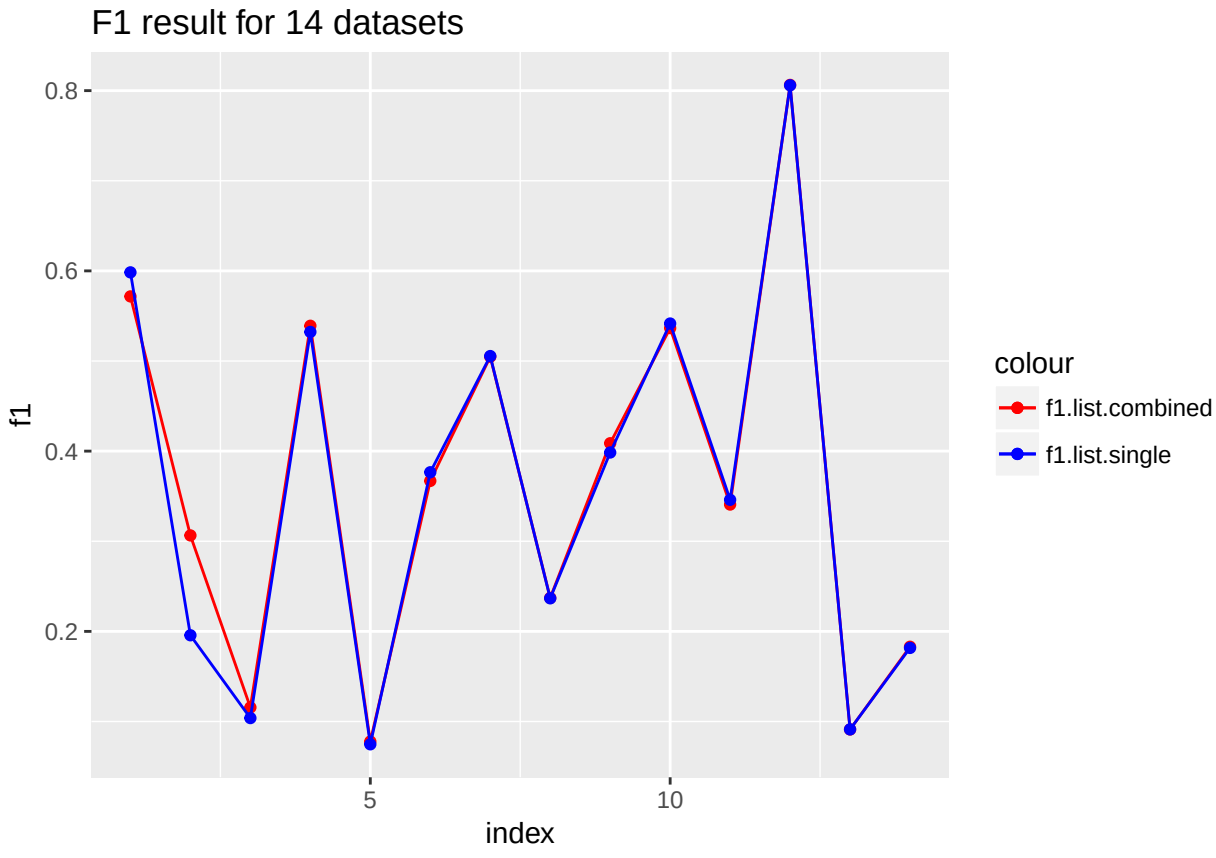
		Gold standard clusters (G)	
		Match	Mismatch
Machine-generated clusters (M)	Match	a	b
	Mismatch	c	d

Let M be the set of machine-generated clusters, and G the set of gold standard clusters. Then. in the table, for example, a is the number of pairs of entities that are assigned to the same cluster in each of M and G. Hence, a and d are interpreted as agreements, and b and c disagreements. When the table is considered as a confusion matrix for a two-class prediction problem, the standard "Precision", "Recall","F1", and "Accuracy" are defined as follows.

$$\begin{aligned}
\text{Precision} &= \frac{a}{a+b} \\
\text{Recall} &= \frac{a}{a+c} \\
\text{F1} &= \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \\
\text{Accuracy} &= \frac{a+d}{a+b+c+d}
\end{aligned}$$

```
source("../lib/evaluation_measures.R")

#### paper 1
matching_matrix_single <- NULL
matching_matrix_combined <- NULL
for (i in 1:14){
  matching_matrix_single[[i]] <- matching_matrix(data[[i]],cluster.notcombined[[i]])
  matching_matrix_combined[[i]] <- matching_matrix(data[[i]],cluster.combined[[i]])
}

f1.list.single <- NULL
accuracy.list.single <- NULL
f1.list.combined <- NULL
accuracy.list.combined <- NULL
clustering_errors_single <- NULL
clustering_errors_combined <- NULL
for (i in 1:14){
  f1.list.single[i] <- performance_statistics(matching_matrix_single[[i]])$f1
  f1.list.combined[i] <- performance_statistics(matching_matrix_combined[[i]])$f1
  accuracy.list.single[i] <- performance_statistics(matching_matrix_single[[i]])$accuracy
  accuracy.list.combined[i] <- performance_statistics(matching_matrix_combined[[i]])$accuracy
}
```

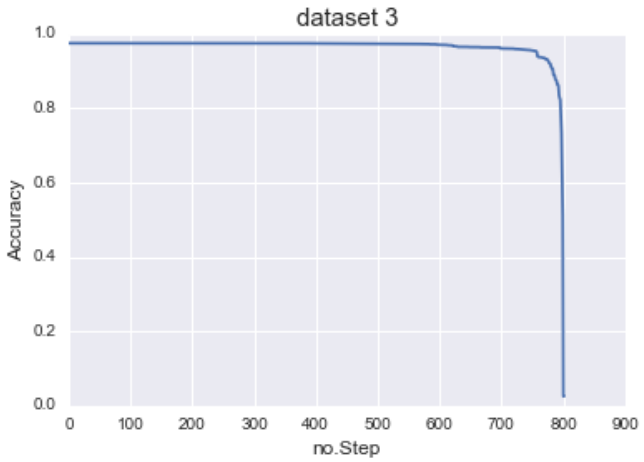
```
f1.combined <- as.data.frame(f1.list.combined)
f1.combined$index <-c(1:14)
f1.single <- as.data.frame(f1.list.single)
f1.single$index <- c(1:14)
acc.combined <- as.data.frame(accuracy.list.combined)
acc.combined$index <-c(1:14)
acc.single <- as.data.frame(accuracy.list.single)
acc.single$index <- c(1:14)

# f1 result
ggplot()+
  geom_point(mapping=aes(x=index,y=f1.list.combined,colour="f1.list.combined"), data=f1.combined)+
  geom_line(mapping=aes(x=index,y=f1.list.combined,colour="f1.list.combined"), data=f1.combined)+
  geom_point(mapping=aes(x=index,y=f1.list.single,colour="f1.list.single"),
             data=f1.single)+
  geom_line(mapping=aes(x=index,y=f1.list.single, colour="f1.list.single"),
            data=f1.single)+
  labs(title="F1 result for 14 datasets",
       x="index",
       y="f1")+
  scale_color_manual(values=c(f1.list.combined="red",f1.list.single="blue"))
```

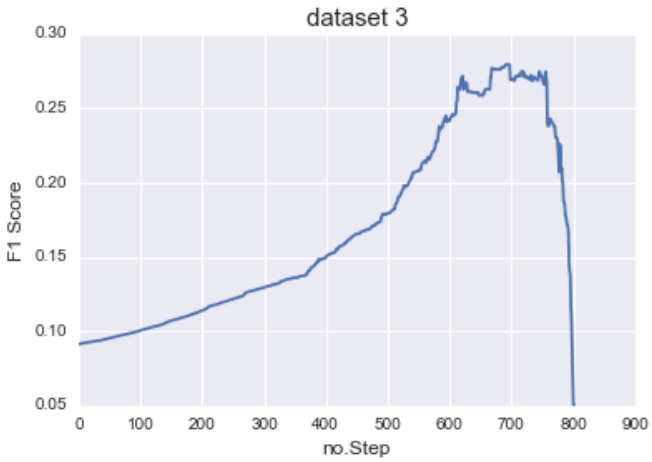


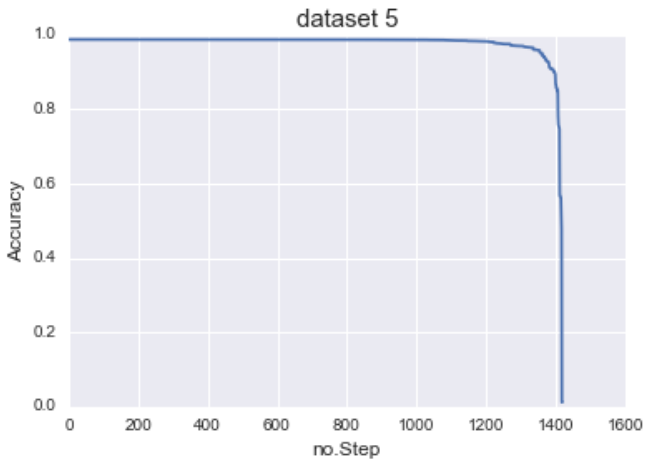

```
# Accuracy result
ggplot()+
  geom_point(mapping=aes(x=index,y=accuracy.list.combined,colour="accuracy.list.combined"), data=acc.co
  geom_line(mapping=aes(x=index,y=accuracy.list.combined,colour="accuracy.list.combined"), data=acc.com
```

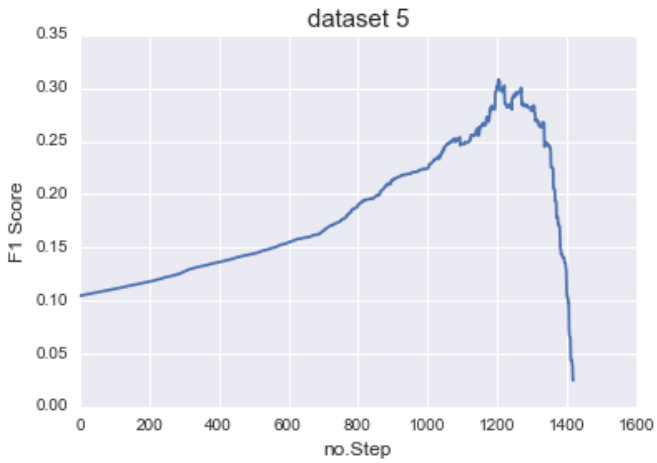
```
geom_point(mapping=aes(x=index,y=accuracy.list.single,colour="accuracy.list.single"),
           data=acc.single)+
geom_line(mapping=aes(x=index,y=accuracy.list.single, colour="accuracy.list.single"),
          data=acc.single)+
labs(title="Accuracy result for 14 datasets",
     x="index",
     y="accuracy")+
scale_color_manual(values=c(accuracy.list.combined="red",accuracy.list.single="blue"))
```

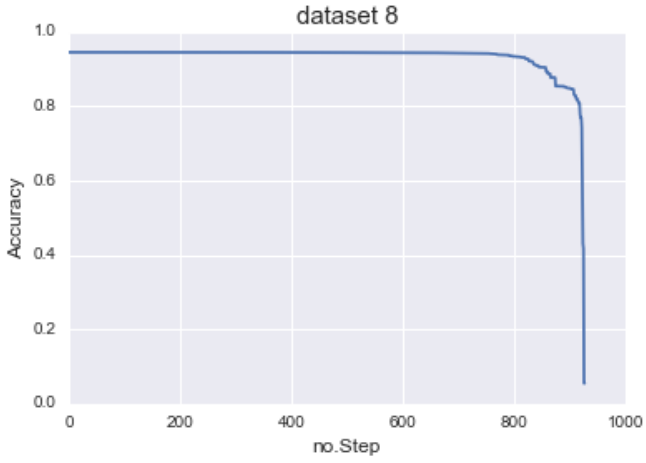
Two scenarios actually look similar. #### Paper 3,5,8 and 13 have low accuracy and f1. So, we use these papers as test sets for algorithm in paper 5. And here are the f1 and accuracy results for these papers.

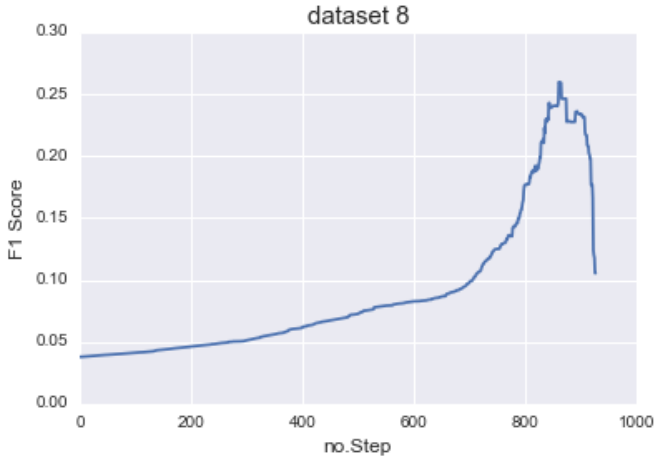


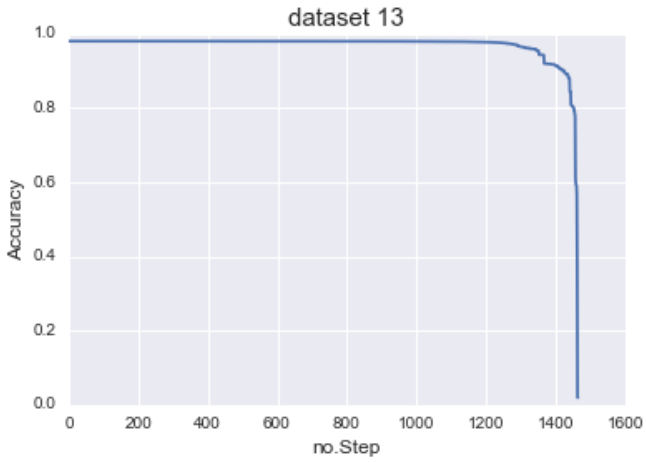


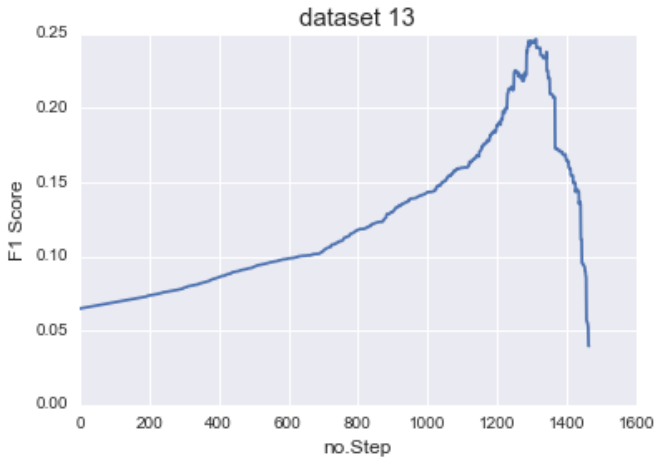












Compared two models

In method 2 (paper 5), the time for updating λ is too long, so we only allow the program to update λ 200 times(it takes about 1 minute). For our traning set, we estimate that it will take 5 mintues.

```r
# Data from python notebook, filename TrainingAndPredict
a <- mean(accuracy.list.single[[3]],accuracy.list.single[[5]],accuracy.list.combined[[8]],accuracy.list
f1.unlist <- unlist(f1.list.single)
bb <- sd(c(f1.unlist[3],f1.unlist[5],f1.unlist[8],f1.unlist[13]))
b <- mean(f1.list.single[[3]],f1.list.single[[5]],f1.list.single[[8]],f1.list.single[[13]])

time_scluter <- time_scluter/14
aa = c(0.964,0.967,0.849,0.919)
bbb = c(0.27,0.268,0.234,0.172)
cccc = 5
compare_df <- data.frame(method=c("singlelink","hcluster"),
                         accuracy=c(a,mean(aa)),
                         f1_avgerage=c(b,mean(bbb)),
                         f1_variance=c(bb,sd(bbb)),
                         traning_time=c(time_scluter,cccc),
                         predict_time=c("NA","0.896s"))

kable(compare_df,caption="Comparision of performance for two clustering methods", digits=2)
```

Table 1: Comparision of performance for two clustering methods

method	accuracy	f1_avgerage	f1_variance	traning_time	predict_time
singlelink	0.68	0.10	0.07	1.267056 mins	NA
hcluster	0.92	0.24	0.05	5.000000 mins	0.896s

Compared two models

In method 2 (paper 5), the time for updating λ is too long, so we only allow the program to update λ 200 times(it takes about 1 minute). For our traning set, we estimate that it will take 5 mintues.

Obviously, method 2 increases accuracy. But the f1 score is still not so good and that is due to the fact that we choose the four datasets with relatively low f1. In addition, the structure of our 14 dataests are not the same.

Conclusion

Algorithm in paper 1 is easy to follow, but it depends heavily on the structure of dataset. In other words, it is not consistent. However, method in paper 5 is hard to follow, but performs better. Considering the executation time, method 1 takes about 1 minute/dataest. Algorithm 2 without updating λ only needs 0.896s to run the training set. However, during training part, we need to upate the parameter, and it takes a relatively long time.