

# COMP 4433 individual report

Weblog mining  
XU Dunyuan 17083297D

## Table of Content:

<b>COMP 4433 INDIVIDUAL REPORT .....</b>	<b>1</b>
<b>CHAPTER1 DEAL WITH THE ORIGINAL DATA .....</b>	<b>2</b>
<b>1.1 Transfer form .txt file to .csv file (translate.py).....</b>	<b>2</b>
<b>1.2 Remove noise data (clean_noisy_data.py) .....</b>	<b>2</b>
<b>CHAPTER2 CLUSTERING DATA.....</b>	<b>3</b>
<b>2.1 K-means .....</b>	<b>3</b>
2.1.1 Principal Component Analysis (PCA) method (PcA+Kmeans.py) .....	3
2.1.2 tSEN method (tSNE+Kmeans.py).....	3
2.1.3 Apply K-means on the new data .....	4
<b>CHAPTER3 PREDICTION &amp; EVALUATION.....</b>	<b>5</b>
<b>3.1 Collaborative Filtering Algorithms (data_analyze.py) .....</b>	<b>5</b>
3.1.1 Difference between clustering and non-clustering .....	5
<b>3.2 Apriori (Apriori.py) .....</b>	<b>6</b>
3.2.1 Evaluate the Apriori's accuracy .....	6
<b>3.3 ID3 .....</b>	<b>7</b>
3.3.1 Build decision tree without clustering (id3.py) .....	7
3.3.2 Build decision tree with clustering (id3_with_clustering.py).....	7
<b>3.4 N-grams (n_gram.py) .....</b>	<b>8</b>
3.4.1 Generate prediction based on N-grams algorithm.....	8
3.4.2 Evaluate the accuracy of prediction (check_ngrams_accuracy.py) .....	9
<b>CHAPTER4 CONCLUSION &amp; TIME COMPLEXITY COMPARISION.....</b>	<b>9</b>
<b>CITATION LIST: .....</b>	<b>10</b>

# Chapter1 Deal with the original data

## 1.1 Transfer from .txt file to .csv file (translate.py)

The original data was stored in .txt format, it be more convenient for further analyzing if we transfer it into .csv format. So, I wrote a python program to complete this task.

In this program, we first read the .txt file, if the line starts with 'A', we record this Web's Id.

Continue reading lines, if the line starts with 'C', we record the user's ID. Then we create a list full of 0 (at the length of number of attributes) for this user, while the following lines start with 'V', we add 1 at the corresponding index at this list.

Then create a .csv file, write each user's list into this .csv file.

The result will look like this:



(0 means this user never visit this website, 1 means this user has visited this website)

## 1.2 Remove noise data (clean\_noisy\_data.py)

Write a simple program we can discover that there are some websites that have been visited quite few times (only 0, 1 or 2 times among all users). I regarded these attributes (sum  $\leq 5$ ) as noisy data and remove them.

```
check_accuracy_with_clustering.py x data_analyze.py x check_accuracy.py x test.py x
1 import pandas as pd
2 train = pd.read_csv('/Users/xudunyan/Desktop/data mining individual/web/trainData.csv')
3 a = []
4 labels = list(train.columns.values) #this is the first line of the csv file
5 total_column = train.shape[0]
6 total_row = train.shape[1]
7 for i in range(1, len(labels)):
8     column = train[labels[i]].tolist()
9     Num_votes = 0
10    for j in range(0, len(column)):
11        if column[j] == 1:
12            Num_votes = Num_votes + 1
13    a.append(Num_votes)
14 print(a)

/Users/xudunyan/.bash_profile: line 1: /Users/xudunyan/.profile: No such file or directory
[912, 4451, 749, 2968, 8463, 42, 135, 865, 10836, 4628, 698, 179, 44, 61, 728, 79, 287, 5108, 5330, 111, 1087, 380,
325, 191, 521, 2123, 3228, 507, 93, 132, 1115, 574, 1446, 26, 9383, 1791, 759, 1160, 1110, 345, 1506, 1500, 281,
224, 168, 474, 636, 210, 343, 186, 86, 842, 670, 338, 264, 276, 195, 672, 258, 391, 269, 141, 113, 324, 323, 82,
548, 198, 227, 602, 187, 128, 284, 584, 396, 444, 155, 462, 136, 121, 215, 241, 105, 186, 86, 22, 189, 237, 157,
187, 69, 97, 65, 14, 102, 214, 56, 98, 120, 291, 14, 118, 36, 35, 183, 16, 11, 47, 59, 46, 36, 128, 181, 48, 15,
31, 9, 172, 365, 1, 63, 13, 372, 222, 199, 27, 132, 1, 7, 395, 148, 23, 69, 162, 115, 181, 123, 33, 18, 118, 36,
19, 60, 36, 20, 79, 86, 96, 12, 93, 21, 52, 8, 67, 52, 75, 124, 90, 41, 36, 16, 48, 25, 49, 38, 33, 72, 93, 44, 16,
47, 45, 3, 10, 6, 63, 43, 2, 11, 9, 12, 7, 167, 57, 24, 46, 38, 94, 51, 48, 4, 7, 25, 26, 15, 1, 32, 18, 1, 18, 38,
4, 55, 30, 24, 29, 12, 34, 9, 5, 16, 25, 3, 3, 45, 30, 13, 18, 4, 23, 10, 11, 29, 16, 7, 14, 32, 13, 4, 21, 19, 4,
1, 8, 9, 10, 3, 4, 3, 11, 9, 4, 4, 4, 2, 10, 3, 1, 2, 13, 11, 3, 5, 1, 3, 3, 5, 3, 1, 1, 2, 4, 2, 4, 3, 2, 5, 1, 2,
1, 1, 1, 1, 1, 1, 3, 1, 2, 1, 2, 1, 2, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 716, 0]
[Finished in 3.75s]
```

# Chapter2 clustering data

## 2.1 K-means

K-means starts by randomly defining  $k$  centroids. From there, it works in repetitive steps to perform two tasks:

1. Assign each data point to the closest corresponding centroid, using the standard Euclidean distance.
2. For each centroid, calculate the mean of the values of all the points belonging to it. The mean value becomes the new value of the centroid.

Once the new centroid has no change, which means we have finished this procedure. And all points now have been accurately grouped.

However, it works for 2-D or 3-D data, but the data's dimension I have now is much higher than 3, so I need to find some method to reduce their dimension.

### 2.1.1 Principal Component Analysis (PCA) method (PcA+Kmeans.py)

Principal component analysis (PCA) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables (entities each of which takes on various numerical values) into a set of values of linearly uncorrelated variables called principal components.[1]

There is a module called PCA in python's library sklearn, it's very convenience for using. The data after reduce dimension is:

```
pca=PCA(n_components=2)
newData=pca.fit_transform(total_num)

In [3]: newData
Out[3]: array([[ -0.45392971,  0.08874764],
               [-0.49669083,  0.35170869],
               [-0.80321623,  0.6480309 ],
               ...,
               [ 0.41756315, -0.27407425],
               [-0.49669083,  0.35170869],
               [-0.37253411,  0.80220042]])
```

### 2.1.2 tSEN method (tSNE+Kmeans.py)

tSNE is non-linear and probabilistic technique. What this means tSNE can capture non-linear pattern in the data.

There is a module called TSNE in python's library sklearn, it's also very convenience for using. The data after reduce dimension is:

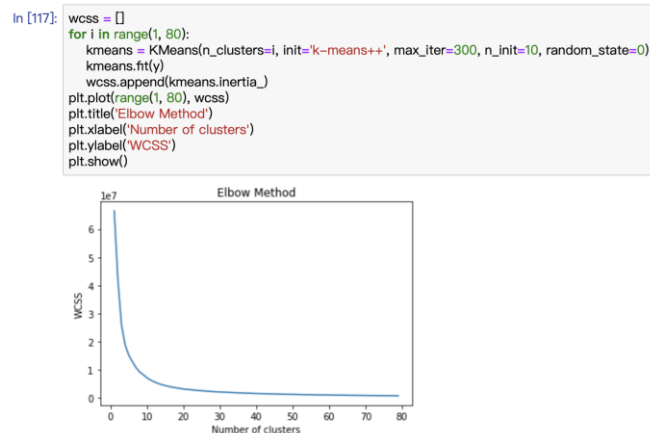
```
In [79]: from sklearn.manifold import TSNE
import numpy as np
X = np.array(total_num)
tsne = TSNE(n_components=2)
y = tsne.fit_transform(X)
print(tsne.embedding_)

[[ -6.195653  47.16304 ]
 [  7.6250005 61.88289 ]
 [ 13.579866  35.785366 ]
 ...
 [ 27.080004 -17.03326 ]
 [  7.5852456 61.836727 ]
 [ 21.293022  64.4112  ]]
```

### 2.1.3 Apply K-means on the new data

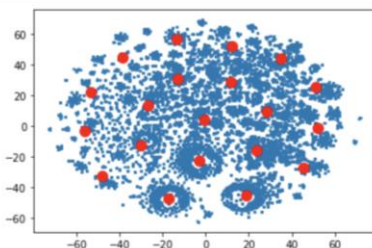
First, I need to know the number of clusters that can provide the best result. Here I use the `kmeans.inertia_` to represent the result of clustering. And also use matplotlib to draw the picture for more obvious viewing.

By observing this picture, I found that the result is better when number of clusters is increasing. When it reached 20, it changes a little as going on. So, I set the number of clusters as 20.



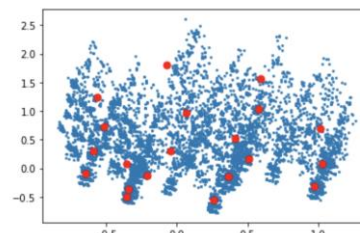
Then I draw the points distribution picture for both tSEN and PCA method.

```
import sys
kmeans = KMeans(n_clusters=20, init='k-means++', max_iter=300, n_init=10, random_state=0)
colors = ['b','g','r','orange']
pred_y = kmeans.fit_predict(y)
plt.scatter(y[:,0], y[:,1], s=3)
plt.scatter(kmeans.cluster_centers[:, 0], kmeans.cluster_centers[:, 1], s=100, c='red')
fig = plt.figure(figsize=(20,20))
plt.show()
```



This is K-means for tSEN

```
import sys
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=20, init='k-means++', max_iter=300, n_init=10, random_state=0)
colors = ['b','g','r','orange']
pred_y = kmeans.fit_predict(newData)
plt.scatter(newData[:,0], newData[:,1], s=3)
plt.scatter(kmeans.cluster_centers[:, 0], kmeans.cluster_centers[:, 1], s=50, c='red')
fig = plt.figure(figsize=(20,20))
plt.show()
```



This is K-means for PCA

# Chapter3 prediction & evaluation

## 3.1 Collaborative Filtering Algorithms (data\_analyze.py)

According to paper Empirical Analysis of Predictive Algorithms for Collaborative Filtering [2], I could predict the probability of each user visit the specific website. And we could also evaluate the accuracy of prediction using these probabilities.

The probability equation is:  $p_{a,j} = \bar{v}_a + \kappa \sum_{i=1}^n w(a,i)(v_{i,j} - \bar{v}_i)$

K is a normalizing factor.

$\bar{v}_i$  is the mean vote for user i. It's the number of websites that has been visited by user i divide the number of total websites.

$w(a, i)$  is the weight between user a and user i. The calculating process is shown at the Figure1.

$f_i$  means the inverse user frequency. It's the number of users has visited this website divide the total number of users, then take log of it.

So, I wrote a python program to calculate the probability. The predicting result is like:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	0.24957388	0.66894556	0.21426983	0.27773799	0.28018841	-0.0051108	0.00543172	-0.0016708	0.23450211	0.15994916	0.00652715	0.01002742	-0.0015938	0.00022654	0.09469357	-0.0018156	0.007425
2	0.04680298	0.79052323	0.03501653	0.64585851	0.36194654	-0.0090206	-0.0049266	-0.0061631	0.27103308	0.18031313	0.00457786	-0.0016485	-0.0076279	-0.0010834	0.01667017	-0.0069133	0.000856
3	0.03993117	0.57375625	0.03389231	0.46835424	0.60029845	-0.0044099	-0.0002262	4.52E-05	0.25808428	0.17092349	0.01047183	0.00234683	-0.0033293	0.00146461	0.01818102	-0.0021817	0.004204
4	-0.0344264	-0.139918	-0.0184143	-0.0840135	-0.196582	0.24122002	0.0076604	0.00906186	-0.2361188	-0.1006721	-0.014481	0.00218845	0.01560643	0.01614607	-0.0265029	0.00918113	-0.00387
5	0.04285741	0.00849591	0.02089463	-0.0448552	-0.0040454	0.00234682	0.48878628	0.01387838	-0.0051104	-0.0233257	-0.0091615	0.00200794	0.00373395	0.01136038	-0.0208494	0.00306104	-0.00568
6	0.0266195	0.35244222	0.02509687	0.50259329	0.73115279	-0.0071123	-0.0041555	-0.0023596	0.25913217	0.16485426	0.00882685	0.00034792	-0.0057495	-0.002475	0.0157479	-0.0051601	0.001302
7	-0.0029315	-0.0103925	-0.0049782	-0.0120803	0.02216087	-8.35E-05	0.00283316	0.85824939	0.3429268	0.02585371	-0.0022521	-0.0051777	-0.0037023	-0.0036027	-0.0074707	-0.0038876	-0.00425
8	0.01466269	0.17333242	0.01864883	0.13826675	0.97910728	-0.0087177	-0.0045011	-0.0016532	0.21609557	0.13755379	0.00793328	-0.0032108	-0.0082032	-0.0065479	0.00790773	-0.0063466	-0.00252
9	0.01536751	0.12774119	0.02302378	0.10253304	0.20862042	-0.0066001	-0.0031446	0.01808297	0.78969613	0.55154512	0.01387597	-0.0032105	-0.0076729	-0.0067851	0.01432728	-0.0057837	0.007960
10	0.37739958	0.10362864	0.01627541	0.08125872	0.09580695	0.01068215	0.01625986	0.00625867	0.12094994	0.05341545	0.23907888	0.13720709	0.05479395	0.04913846	0.35038863	0.01378	0.029091

Figure 1

### 3.1.1 Difference between clustering and non-clustering

After predicting, I need to evaluate the accuracy of prediction.

I use the formula:  $S_a = \frac{1}{m_a} \sum_{j \in P_a} |p_{a,j} - v_{a,j}|$ , here  $m_a$  is the number of predicted visiting.

I have evaluated the prediction both on clustering data and non-clustering data. The results are:

```
/Users/xudunyan/.bash_profile: line 1: /Users/xudunyan/.profile: No such file or directory
final score: 2.75856444104169
[Finished in 1.4s]
```

This is the score without clustering (check\_accuracy.py)

```
/Users/xudunyan/.bash_profile: line 1: /Users/xudunyan/.profile: No such file or directory
final score: 2.0659046782605675
[Finished in 2.3s]
```

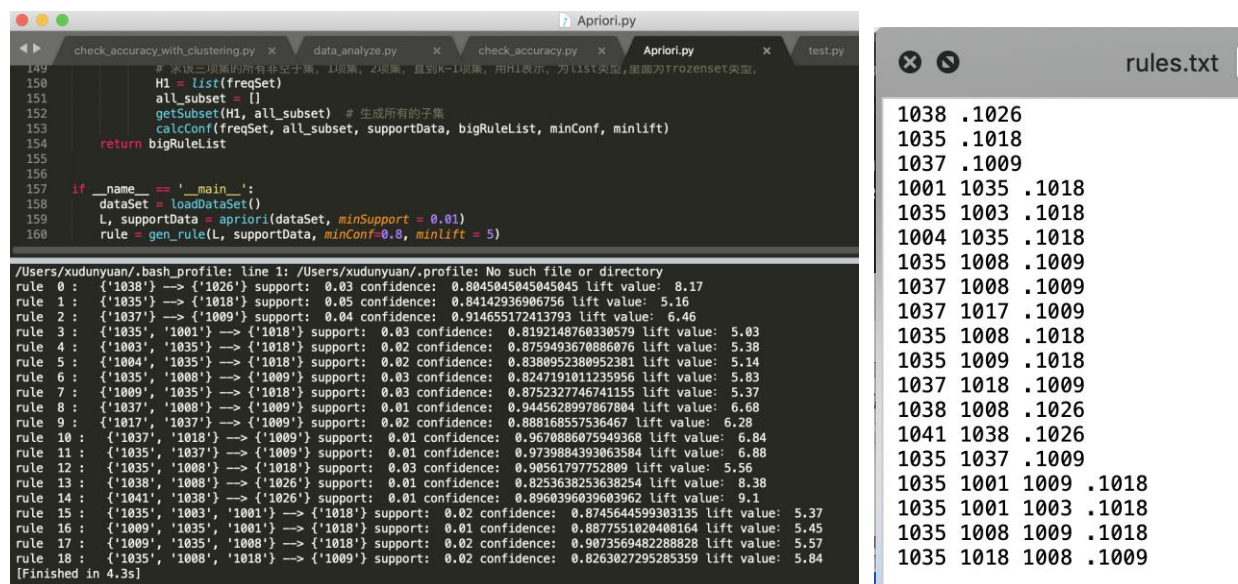
This is the score with clustering (tSEN) (check\_accuracy\_with\_clustering.py)

As we can observe that the prediction with clustering is better than prediction without clustering.

## 3.2 Apriori (Apriori.py)

Apriori is an algorithm for frequent item set mining and association rule learning over relational databases [3]. It has been discussed in class, so here I just show the screenshot of the running result of this program. And using a .txt file to contain all these rules.

I set the support  $\geq 0.01$ , confidence  $\geq 0.8$ , lift value  $\geq 5$ . (There are totally 19 rules).



The screenshot shows the output of the Apriori algorithm. The left pane displays the code execution in a terminal, showing the generation of 19 rules based on the specified parameters (support  $\geq 0.01$ , confidence  $\geq 0.8$ , lift value  $\geq 5$ ). The right pane shows the content of the rules.txt file, which lists the generated rules in a tab-separated format.

```
rule 0 : {'1038'} -> {'1026'} support: 0.03 confidence: 0.8045045045045 lift value: 8.17
rule 1 : {'1035'} -> {'1018'} support: 0.05 confidence: 0.84142036906756 lift value: 5.16
rule 2 : {'1037'} -> {'1009'} support: 0.04 confidence: 0.914655172413793 lift value: 6.46
rule 3 : {'1035', '1001'} -> {'1018'} support: 0.03 confidence: 0.8192148760330579 lift value: 5.03
rule 4 : {'1003', '1035'} -> {'1018'} support: 0.02 confidence: 0.8759493670886076 lift value: 5.38
rule 5 : {'1004', '1035'} -> {'1018'} support: 0.02 confidence: 0.8380952380952381 lift value: 5.14
rule 6 : {'1035', '1008'} -> {'1009'} support: 0.03 confidence: 0.8247191011235956 lift value: 5.83
rule 7 : {'1009', '1035'} -> {'1018'} support: 0.03 confidence: 0.8752327746741155 lift value: 5.37
rule 8 : {'1037', '1008'} -> {'1009'} support: 0.01 confidence: 0.9445628997867804 lift value: 6.68
rule 9 : {'1017', '1037'} -> {'1009'} support: 0.02 confidence: 0.888168557536467 lift value: 6.28
rule 10 : {'1037', '1018'} -> {'1009'} support: 0.01 confidence: 0.9670886075949368 lift value: 6.84
rule 11 : {'1035', '1037'} -> {'1009'} support: 0.01 confidence: 0.9739884393063584 lift value: 6.88
rule 12 : {'1035', '1008'} -> {'1018'} support: 0.03 confidence: 0.90561977752809 lift value: 5.56
rule 13 : {'1038', '1008'} -> {'1026'} support: 0.01 confidence: 0.8253638253638254 lift value: 8.38
rule 14 : {'1041', '1038'} -> {'1026'} support: 0.01 confidence: 0.8960396039603962 lift value: 9.1
rule 15 : {'1035', '1003', '1001'} -> {'1018'} support: 0.02 confidence: 0.8745644599383135 lift value: 5.37
rule 16 : {'1009', '1035', '1001'} -> {'1018'} support: 0.01 confidence: 0.8877551020408164 lift value: 5.45
rule 17 : {'1009', '1035', '1008'} -> {'1018'} support: 0.02 confidence: 0.9073569482288828 lift value: 5.57
rule 18 : {'1035', '1008', '1018'} -> {'1009'} support: 0.02 confidence: 0.8263027295285359 lift value: 5.84
```

rules.txt

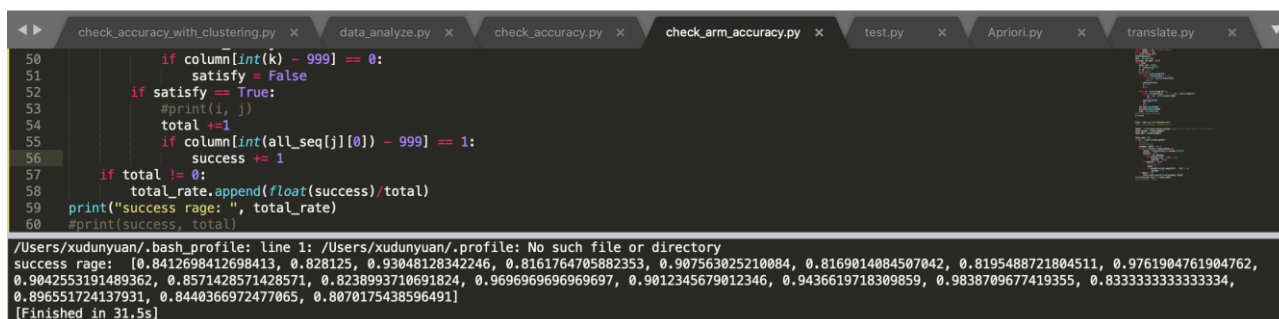
```
1038 .1026
1035 .1018
1037 .1009
1001 1035 .1018
1035 1003 .1018
1004 1035 .1018
1035 1008 .1009
1037 1008 .1009
1037 1017 .1009
1035 1008 .1018
1035 1009 .1018
1037 1018 .1009
1038 1008 .1026
1041 1038 .1026
1035 1037 .1009
1035 1001 1009 .1018
1035 1001 1003 .1018
1035 1008 1009 .1018
1035 1018 1008 .1009
```

### 3.2.1 Evaluate the Apriori's accuracy

After constructing rules, I wrote a python program to evaluate the accuracy of these rules.

This program first read the .txt file to get the rules, then test these rules in test\_data.

For example, there is a rule says  $1038 \rightarrow 1026$ , if there are 10 users in test\_data visited 1038, but only 8 of them also visited 1026, the accuracy of this rule is  $8/10 = 0.8$



The screenshot shows the execution of the check\_arm\_accuracy.py program. The code calculates the accuracy of the rules by comparing the predicted results with the actual results in the test\_data. The output shows the success rate for each rule, which is higher than 0.8 for all rules.

```
success rage: [0.8412698412698413, 0.828125, 0.93048128342246, 0.8161764705882353, 0.907563025210084, 0.8169014084507042, 0.8195488721804511, 0.9761904761904762, 0.9042553191489362, 0.8571428571428571, 0.8238993710691824, 0.9696969696969697, 0.9012345679012346, 0.9436619718309859, 0.9838709677419355, 0.8333333333333334, 0.896551724137931, 0.8440366972477065, 0.8070175438596491]
```

It can be observed that all rules' accuracy are higher than 0.8, which means the rules constructed using Apriori algorithm is credible.



## 3.3 ID3

### 3.3.1 Build decision tree without clustering (id3.py)

ID3 is an algorithm used to generate decision trees based on dataset [5].

Normally speaking, building a decision tree requires us to choose a specific target for predicting. In this problem, there seems no such target. But after some analyzing and calculating, I found that there is one website which has been visited by around 1/3 users, (whose ID is '1008', named as "Free Downloads"). So, I thought I could set this website as the final target and other websites as the attributes.

For accomplishing this decision tree, I import tree library form sklearn. And using GridSearchCV function to find the best parameter pair. Dividing the data into 2 parts, one for training, the other for testing the accuracy.

There is another problem, too many attributes. It will result that the decision tree become too high and to complicated. So, I decide to use 10 most visited websites to be the attributes.

After training, the score is shown in Figure2.

I can also output the picture of this tree, which is shown in Figure3.

Each root of the decision tree and its subtrees is chosen based on the Gini value and the entropy.

```
tr = tree.DecisionTreeClassifier()
gsearch = GridSearchCV(tr, parameters)
gsearch.fit(X_train, y_train)
model = gsearch.best_estimator_
score = model.score(X_test, y_test)
score
```

/Users/xudunyan/anaconda3/lib/python3.7/site-packages/sklearn/exceptions.py:101: FutureWarning: The attribute `score` is deprecated in version 0.22. Specify it explicitly using `score` or `score\_loss`.

0.7213816292220694

Figure2

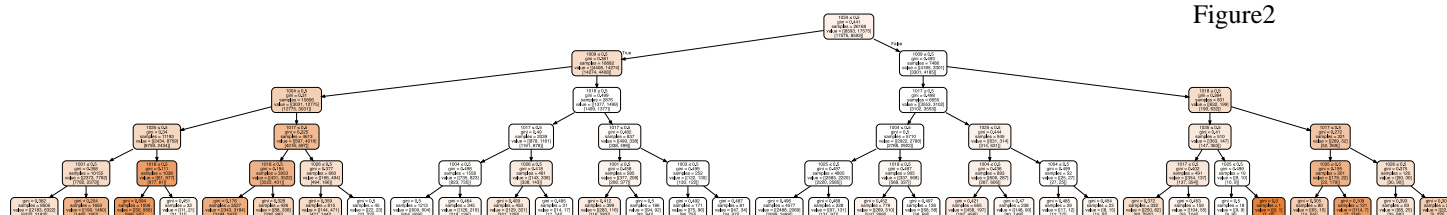


Figure3

### 3.3.2 Build decision tree with clustering (id3\_with\_clustering.py)

I have discussed the clustering method at chapter2. I have also tried the decision tree's score inside each cluster. And the result is better. Some cluster even could reach 1, which means there is a rule that must happen in this cluster.

The method of dividing clusters is the same as what I have done in chapter2. Using tSEN and K-means to divide it into 20 clusters.

The 20 clusters' scores are shown in Figure4. And one decision tree example is shown in Figure5, which is much smaller than the original decision tree.

```
for i in scores:
    print(i, end = " , ")
```

1.0 , 0.8166666666666667 , 1.0 , 1.0 , 0.9927360774818402 , 0.8257575757575758 , 0.827027027027027 , 1.0 , 0.575 , 0.9629629629629629 , 1.0 , 0.7676767676767676 , 1.0 , 0.9814814814814815 , 1.0 , 0.7236180904522613 , 0.8373983739837398 , 0.7488789237668162 , 1.0 , 0.7608695652173914 ,

Figure4

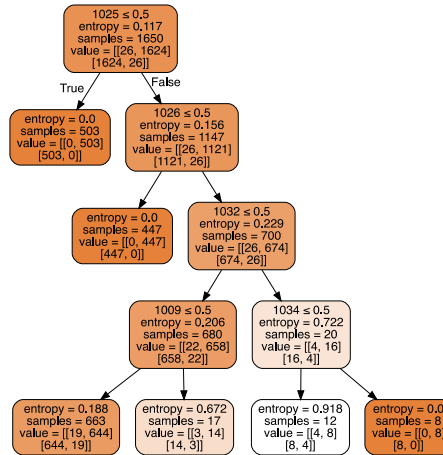


Figure5

## 3.4 N-grams (n\_gram.py)

### 3.4.1 Generate prediction based on N-grams algorithm

Normally, N-grams algorithm is used for text mining in order to predict what words will be typed next. But according to Fu et al. [5], N-grams algorithm can also be used in prediction web browsing. Because the users may waste a lot of time in useless websites until they get to the final page. For example, the web browsing order is A,B,C,D,E,F,G, but what user wants is to get to page G from page C, that means page D,E,F are useless pages.

So, I followed this idea and found that the data of users visiting history is disordered, I thought it may be the order of browsing. First is to set the number of grams (for testing, I set it to be 3-grams). Then scan all users' browsing history to get all grams (the last page in browsing history is assumed to be the target page). Finally, I need to calculate the probabilities for each rule mined.

In this program, I used Counter module from collection library to help me finish the calculating task.



There are 4 parameters for n\_gram function: first is the number of grams, second is the number of minimum times that gram appears, third is the number of minimum times target page appears inside that gram, fourth is the minimum probability of that rule. The result is shown below:

```

50
51 def n_gram(grams, rules_num, preds_num, min_prob): ...
97
98 n_gram(3, 100, 10, 0.3)
99
/Users/xudunyan/.bash_profile: line 1: /Users/xudunyan/.profile: No such file or directory
rules: (('1035', '1001', '1003'), '-->', '1018') . prob: 0.7310924369747899
rules: (('1001', '1003', '1018'), '-->', '1004') . prob: 0.3963963963963964
rules: (('1035', '1008', '1009'), '-->', '1018') . prob: 0.3629032258064516
rules: (('1001', '1003', '1004'), '-->', '1018') . prob: 0.6306306306306306
rules: (('1008', '1035', '1009'), '-->', '1018') . prob: 0.41935483870967744
rules: (('1035', '1037', '1009'), '-->', '1017') . prob: 0.3316062176165803
rules: (('1017', '1001', '1003'), '-->', '1004') . prob: 0.3577981651376147
rules: (('1038', '1026', '1041'), '-->', '1034') . prob: 0.39285714285714285

```

### 3.4.2 Evaluate the accuracy of prediction (check\_ngrams\_accuracy.py)

Because in the paper, it only mentioned about the evaluation on testing server and client response time. So, in this experiment I used the method like the algorithm evaluating Apriori. First read the test.txt and generate the 3-grams rules. If the grams are in the prediction rules, then check if the result is the same as the prediction result. And then calculate the percentage of correct production. The result is shown below:

```

42
43 def test_gram(): ...
96
97 test_gram()
98
/Users/xudunyan/.bash_profile: line 1: /Users/xudunyan/.profile: No such file or directory
[0.125, 0.05263157894736842, 0.047619047619047616, 0.07142857142857142, 0.022727272727272728, 0.02857142857142857, 0.05, 0.045454545454545456]
[Finished in 6.6s]

```

And I observed that the result is not so satisfied...

## Chapter4 Conclusion & Time complexity comparision

In order to finish this project, I separate it into 3 parts. First, dealing with data. Second, generating clusters. Third, using different models to predict. And among all these 4 models I have used, the most time consuming one is the Collaborative Filtering, it takes 100s to predict 10 users. I think maybe it's due to the formula. When calculating the probability for each user, we need to know the weight between this user and all rest users, it's such a huge task. Decision Tree model, because there are too many attributes, the height and width of the tree will be quite large. So, I restricted the max

depth and only use the top 10 most visited websites. The time consuming of Decision Tree is acceptable because the reduction on data. The worst prediction result may be generated by N-grams algorithm, its success rate is too low. The time consuming is about 10 second including predicting(5s) and checking accuracy(5s). And Apriori may be the best method I have used among these four. The time consuming is about 20s including generating rules(12s) and checking accuracy(8s).

## Citation list:

- [1] W. contributors. (2019, December 11). Principal component analysis. In *Wikipedia, The Free Encyclopedia*. Retrieved 12:49, December 18, 2019, from [https://en.wikipedia.org/w/index.php?title=Principal\\_component\\_analysis&oldid=930241703](https://en.wikipedia.org/w/index.php?title=Principal_component_analysis&oldid=930241703)
- [2] Breese, J. S., Heckerman, D., & Kadie, C. (1998, July). Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence* (pp. 43-52). Morgan Kaufmann Publishers Inc..
- [3] W. contributors. (2019, September 22). Apriori algorithm. In *Wikipedia, The Free Encyclopedia*. Retrieved 12:50, December 18, 2019, from [https://en.wikipedia.org/w/index.php?title=Apriori\\_algorithm&oldid=917123976](https://en.wikipedia.org/w/index.php?title=Apriori_algorithm&oldid=917123976)
- [4] Wikipedia contributors. (2019, November 30). ID3 algorithm. In *Wikipedia, The Free Encyclopedia*. Retrieved. 17:12, December 19, 2019, from [https://en.wikipedia.org/w/index.php?title=ID3\\_algorithm&oldid=928640290](https://en.wikipedia.org/w/index.php?title=ID3_algorithm&oldid=928640290)
- [5] Fu, Y., Paul, H., & Shetty, N. (2007). Improving Mobile Web Navigation Using N-Grams Prediction Models. *International Journal of Intelligent Information Technologies (IJIIT)*, 3(2), 51-64.