3460-676

**Data Mining**

Final Report

**Recommendit: A Recommendation Engine for Identifying Subreddits of Interest**

**Students to conduct the work:**

JD Kilgallin

Suphanut Jamonnak

07/27/2015

Department of Computer Science

The University of Akron

**Background**

Reddit is a social network and content aggregation site begun in 2005. Users post text, photos, or links to a “subreddit” - a forum on the site centered around a single topic. As of April 2015, Reddit had over eight thousand subreddits, ranging across a wide variety of popular topics such as “jokes”, “art”, “world news”, “gaming”, and “recipes”, as well as more esoteric subjects like “haskell”, “doctorwho”, and “cyber laws”. Users may view posts recently submitted to a subreddit, and can also contribute to a crowd-based ranking system by voting posts up or down to indicate approval or disapproval. A user may also comment on a post and perform the same actions on the comments that can be done on the post itself. Figure 1 below illustrates the view of a single subreddit, with the sorted posts, and Figure 2 illustrates the view of a single post, which appears if the user clicks on a post’s “comments” link in the subreddit list. Both figures are cropped slightly for readability in the limited space of a single page. Of course, though, the interactive nature of the site can best be illustrated by visiting the site directly - for example, at “www.reddit.com/r/python”, though note that some actions reqire account creation and login.

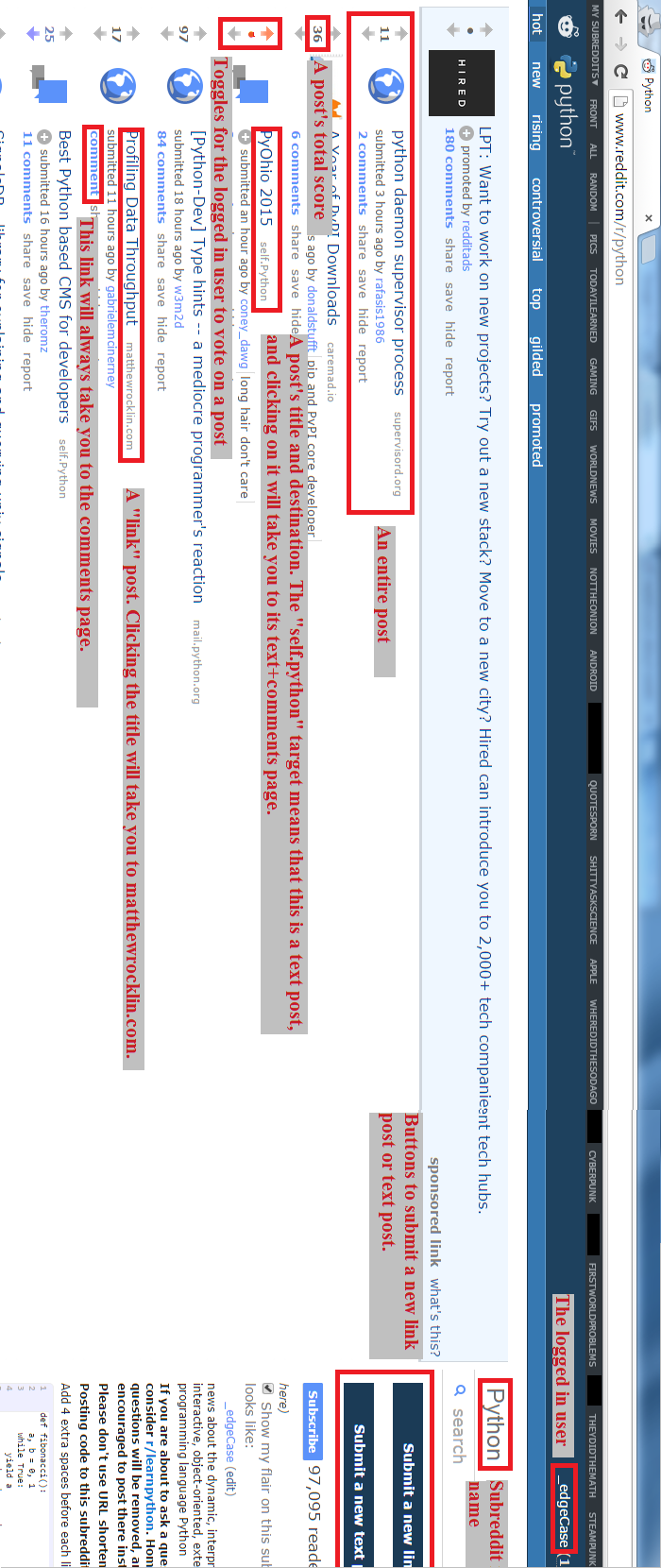


Figure 1: A subreddit page

Figure 2: A page for a single post

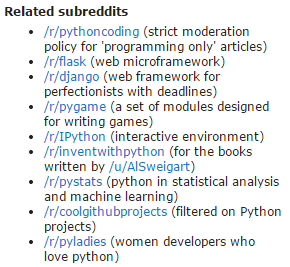
**Subreddit Discovery**

The reddit homepage consists of a feed of popular posts across a set of different subreddits. The nature of the collection depends on whether a user is logged in or not. A user browsing anonymously will see posts from among a select group of “default” subreddits, which are hand-picked by the site administrator according to perceived quality and general applicability. If a user is logged in, however, the front page will show primarily content from subreddits that the user has subscribed to. A user can subscribe or unsubscribe from any subreddit from either the subreddit itself or from a list of subreddits through personal settings. A particularly popular and recent post from a subreddit the user has not subscribed to may also appear on their front page. The process of discovering subreddits requires a user to actively search and browse, or to come across links to new subreddits more organically through other links on the site or elsewhere. Such links are regularly found within comments to a post in a related subreddit. Even a subreddit’s main page may list related subreddits, as determined by the moderators of that subreddit. For example, toward the bottom of the r/python page, a list is included as shown in Figure 3 below. With this system, a user could go months or years on the site unaware of the existence of a subreddit pertinent to the user, and in any case relies on direction from other users, which may at any time be incomplete or out of date.

Some third-party sites aim to facilitate this discovery process. For example, the site “subreddits.org/” and other third-party sites group subreddits by category; again, these are built by hand by the site development team. At least a few projects aim to programmatically build recommendation engines. In one case, Sundaresan et al build a graph to identify communities using weighted graphs, based on two similar, earlier approaches [1]. A github project also called “Recommendit” by “ben444422”, with no updates in 2 years, appears to have begun on a similar path to the one we intend, but did not seem to get very far [2]. Our goal with this project is to facilitate this process of discovering pertinent subreddits.

**Formulation**

Of the thousands of subreddits, many address related topics and the problem of identifying such relations is not solved. As a site with hundreds of millions of visitors per month, with content partitioned so distinctly by topic, such a solution could improve engagement and growth of the site. To accomplish this, we propose an association-rule based recommendation engine based on the set of subreddits that a single user posts to in common. With this approach, an “item” is a subreddit and a transaction is a set of subreddits that a given user frequently posts to. By mining reddit for many users’ post history, we identify and associate subreddits that have several joint users, using the a priori algorithm with appropriate support and confidence thresholds. From the rules produced, we develop an application that allows a user to input a subreddit or set of subreddits of interest, and returns related subreddits that may be of interest to the user as well.

Figure 3: Subreddits related to r/python

**Process**

To implement this approach to identifying similar subreddits, we perform four steps:

1. For each of the top 2500 subreddits by subscriber (obtained from redditlist.com [3]), we scrape reddit to obtain the set of users that have made a popular post to the subreddit this year. This is done with a Python program using reddit’s API [4] and stored in a separate CSV file for each subreddit. This is augmented with the top 1000 posts of all time as of August 2014, downloaded from [5]. See "Data Collection" below for more information on this part.
2. We strip the irrelevant attributes from the data collected from step 1, then group it by user to form a list of transactions as described in our formulation above. This is done by loading the CSV files into Sqlite and using SQL queries to transform the data.
3. We pass the list of transactions to an implementation of the a priori algorithm in Python [6], in order to find the list of association rules.
4. Using the association rules discovered, we host a simple webapp, also written in Python and using the CherryPy library [7], that allows a user to query for the set of subreddits associated with a given input subreddit.

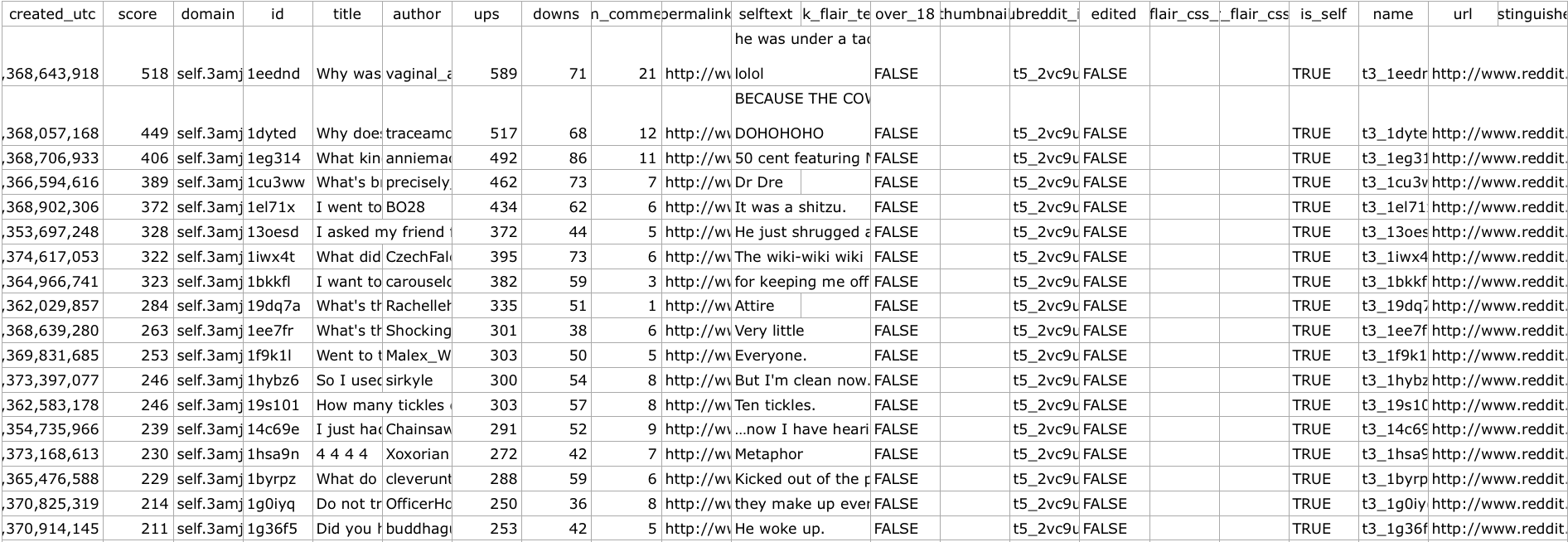
This process, taken together, allows users to find subreddits of interest. Due to the growing and shifting nature of the reddit communities, steps 1 to 3 should be repeated periodically - perhaps on the order of once a month

**Data Collection**

The data we require consists of the author and subreddit for as many posts as we can collect, store, and process. There are many ways to obtain such data - from reddit itself we can scrape the HTML site or use their provided API. In this case, we have used a Python library called PRAW, an acronym for “Python Reddit API Wrapper” [8]. PRAW is a python package that allows for simple access to reddit’s API. It aims to be easy to use for us and is also designed to respect all of reddit’s API rules. Another approach, which we also make use of, is to rely on data previously collected by others and hosted elsewhere. Fortunately, an existing github project provides a 6-month old copy of exactly the data we are interested in - post information from the top 2500 subreddits by subscriber count [5].

So, we created a crawling data application implemented in python called “subrtime.py [appendix 1]”. subrtime.py will get the input as a subreddit’name, start and end timestamp. In addition, it will crawl historical subreddit’s post using PRAW, which allows a crawling interval to be specified. subrtime.py will crawl all of subreddit’s posts in the range of timestamp, and return the data and attributes shown in the example below:

Table 1: raw data table, attributes and data



The above table represents all of the attributes we crawled at the first time. Each individual post consists of title, author, created time, score, subreddit name, and so on. In addition, we have selected top 1,000 posts from popular 2,500 subreddit ranked by its subscribers.

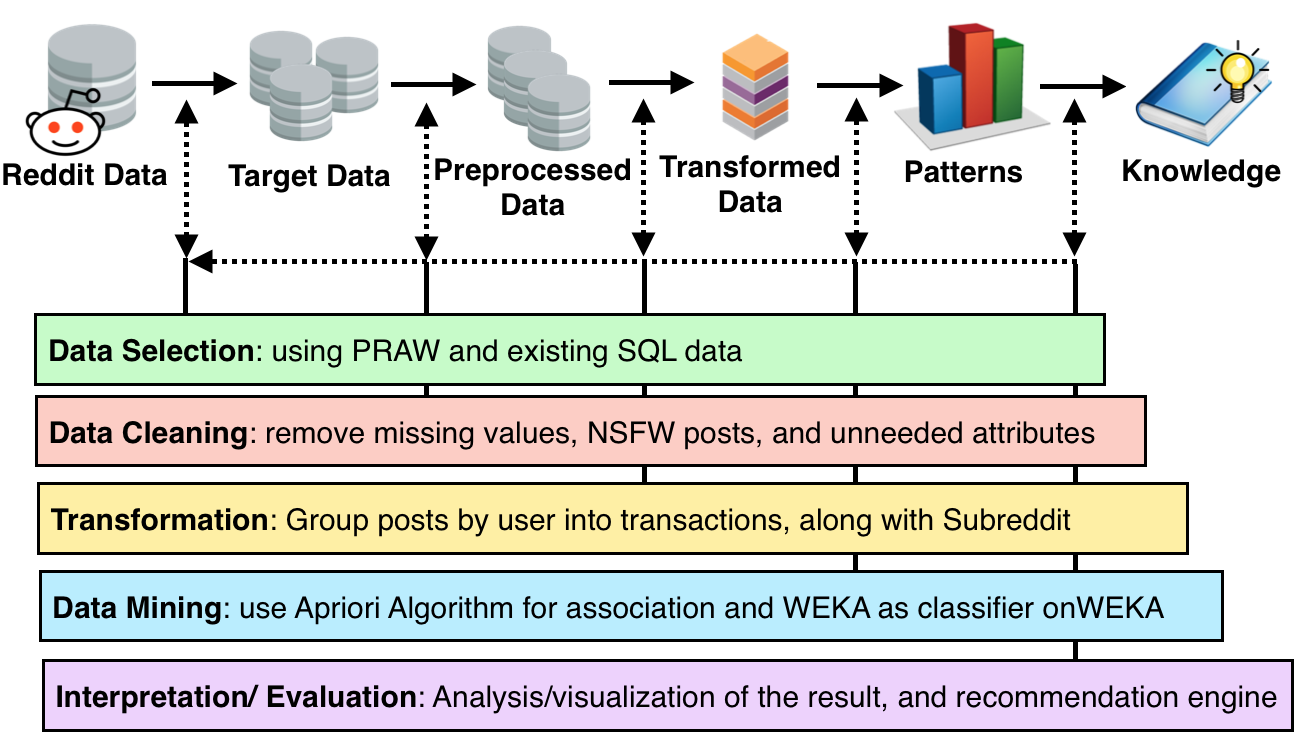


Figure 4: Data mining process for recommendit system

**Data Selection**

According to collected data in table1, we have selected attributes that we are interested in transforming in order to apply the Aprori algorithm. In particular, we have selected author name, subreddit name, created time, nsfw, score, and num\_comments for the further experiments. This is all done simply using SQL queries for reducing the columns. and getting all of the data and attributes we need. Here is the sample dataset:

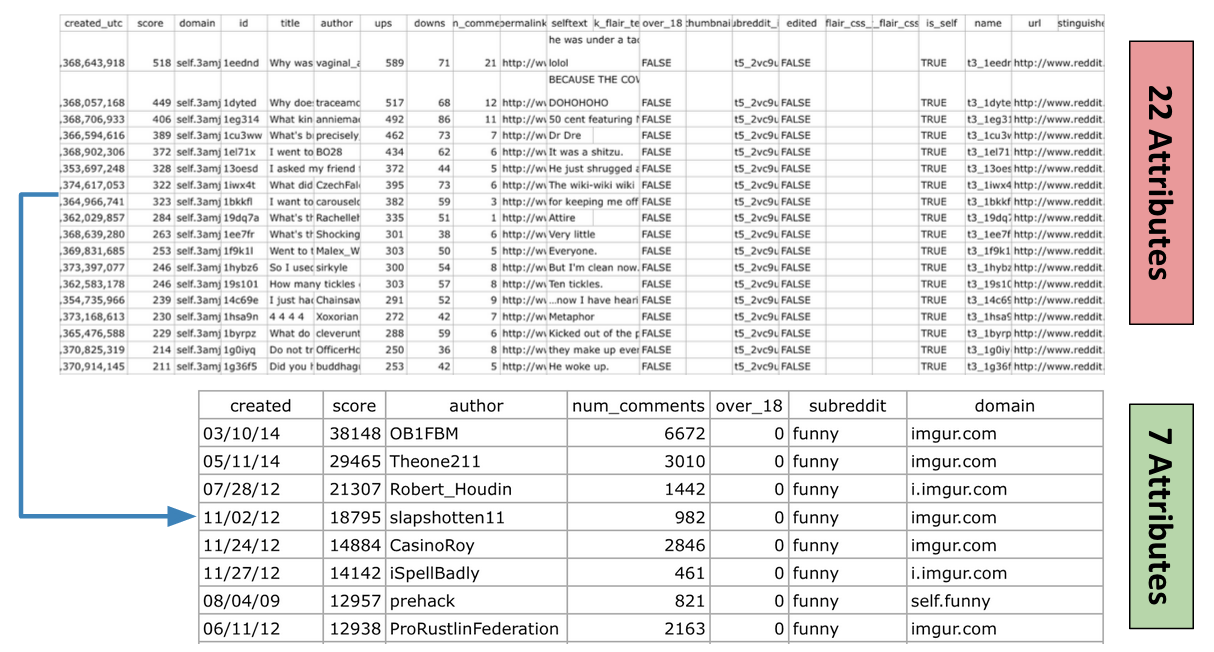


Figure 5: Illustrate selected raw data reduced from 22 attributes to 7 attributes

As you seen from the table above, [delete] author, means that author name does not exist or has been removed by reddit, but that post still exists. Also “NSFW” - Not Safe for Work - attribute consists of True or False value. A “True” value indicates that the post is explicit or inappropriate, or links to explicit or inappropriate content. We remove these entries as well, during the preprocessing process in the next step.

**Data Preprocessing**

In this process, we have used the preprocessing concepts for getting consistent data such as data cleaning, removing missing value or outlier detection techniques. Specifically, in order to create the recommendation system, Recommendit requires the author name in order to compute a transaction set for Apriori algorithm. Thus, We have to remove author with none or NULL values. In this case, all of the records will filtered and deleted author will be removed. Moreover, we are also removing NSFW post, which represents “not safe for work”, or posts with explicit material, which in general the site allows. We have removed the deleted author and inappropriate post "NSFW" by writing another python script "transubreddit.py" to filter the data in the SQL library. This decreased the number of records in from 2.5 millions to 2.4 millions respectively.

**Data Transformation**

**Introducing Association Rules**

Discovering interesting relations between variables in large databases, is a popular and well researched method in data mining. These techniques are very popular in several fields such as marketing strategy, data analytics, machine learning and so on. For instances, in marketing strategy, association rule mining may be used for discovering regularities between products in large scale transaction data recorded by Point-Of-Sale (POS) systems in supermarket. For example, {onion, potatoes} => burger found in sales data of supermarket can indicate that, if a customer buys onion and potatoes together, they may be also likely to buy a burger. Such information can be used and adapted for several marketing activities such as promotional pricing, product placements, and so on.

Apriori is a classic algorithm for solving and finding association rules, introduced by Rakesh and Ramakrishnan in 1994 [9], its also popular in both computer science and data mining. Apriori is not the only algorithm for solving this problem, but it is sufficient for our problem. There are plenty of algorithm outs there such as finding association rules in data having no transaction (Winepi and Minepi), or having no timestamps (DNA sequencing). But in Recommedit, in order to find association rules, we focused particularly on Apriori Algorithms.

**Association Rules Concepts**

In Agrawal [9], he has originally defined the association rule as:

Let **I = {i1, i2, ..., im}** be a set of m binary attributes called *items*

Let **T = {t1, t2, …, tn}** be a set of transactions called the *database*.

Each transaction in T has a unique transaction ID and contains a subset of the items in I. A *rule* is defined as an implication of the form **X→Y** where **X, Y ⊆ I** and **∩ = ∅**. The sets of items (for short itemsets) X and Y are called antecedent (left-hand-side or LHS) and consequent (right-hand-side or RHS) of the rule respectively. To illustrate this concept, we use this concept with our data from reddit. The set of subreddit items is I = {python, compsci, django, java} and a small database coded with 0 as absence and 1 is presence. From figure below, we can predicted that redditor or user who interested in python, and compsci also interested in django as well.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Transaction ID | python | compsci | java | django |
| Author 1 | 0 | 1 | 0 | 1 |
| Author 2 | 1 | 1 | 0 | 1 |
| Author 3 | 1 | 1 | 0 | 1 |
| Author 4 | 0 | 0 | 1 | 1 |
| Author 5 | 1 | 1 | 0 | 1 |
| Author 6 | 1 | 1 | 1 | 0 |

Table 2: Illustrate of transactions and itemsets database and structure

The above example is very small. In association rules mining, we needs a support of several hundred transactions before It can be considered significant. Our dataset also contain of thousands or millions of transactions. As a result, we could select interesting rules from a set of possible rules, constraints on various measures of significance and interest can be used. [10] The best-known constraints are minimum thresholds on **support** and **confidence**:

***Support***

The support supp(X) of an itemset X is defined as the proportion of transactions in the data set which contain the itemset.

***supp(X)= no. of transactions which contain the itemset X / total no. of transactions***

In the example database, the itemset {python, compsci, django} has a support of 3 /6 = 0.50 since it occurs in 60% of all transactions.

***Confidence***

The confidence of a rule is defined:

***conf(X → Y) = supp(X ∪ Y) / supp(X)***

For the rule **{python, compsci} => {django}** we have the following confidence:

***supp*(**{python, compsci, django}**) */ supp*(**{python, compsci}**)** = 0.50 / 0.66 = **0.75**

This means that for 75% of the transactions containing python and compsci the rule is correct.

**Transformation**

In order to using concepts stated above, we have to transform our dataset by using association rules with set of items and transactions, as a template. In addition, we have to transform our dataset according to the structure below:

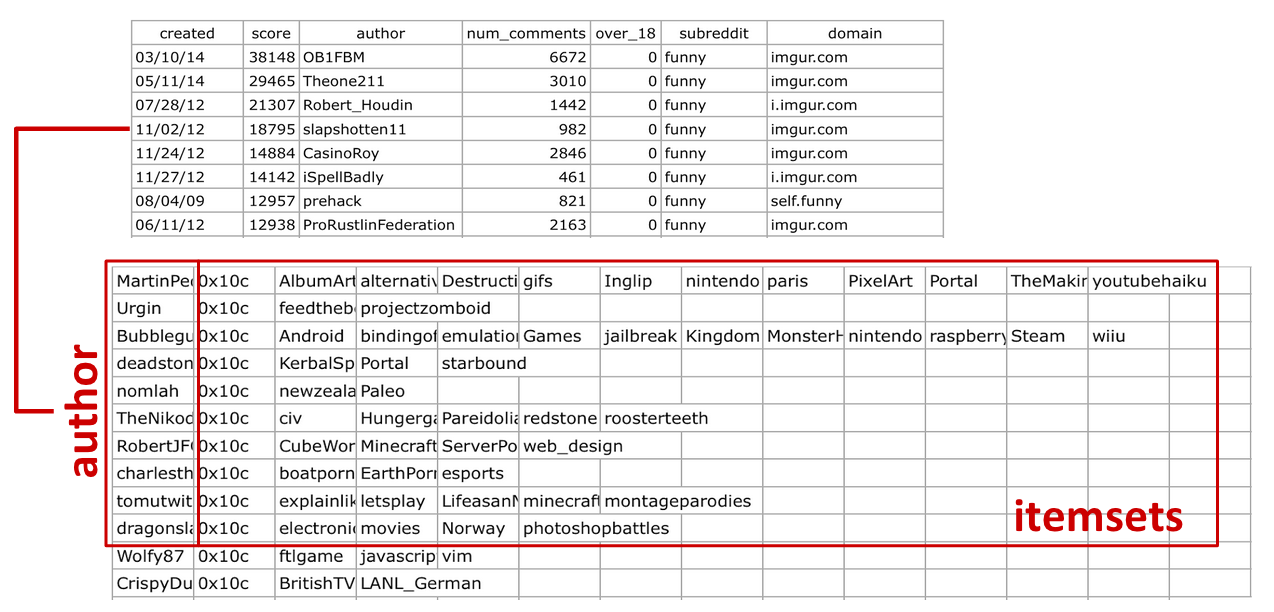


Figure 6: transformed data for data preparation

I = {S1, S2, S3, …, Sm} be a set of subreddit itemsets, where m is binary attributes of subreddit item appeared in our database

T = {A1, A2, A3, …, An} be a set of author, who contributed to those items.

In this case, the unique transaction ID are all of the unique author’ name, and subreddit, which author posted to. We group all of the items as subreddit’ itemsets by unique authors, implemented with python and SQL library, in order to achieve process. The figure shown below, can illustrate this methods:

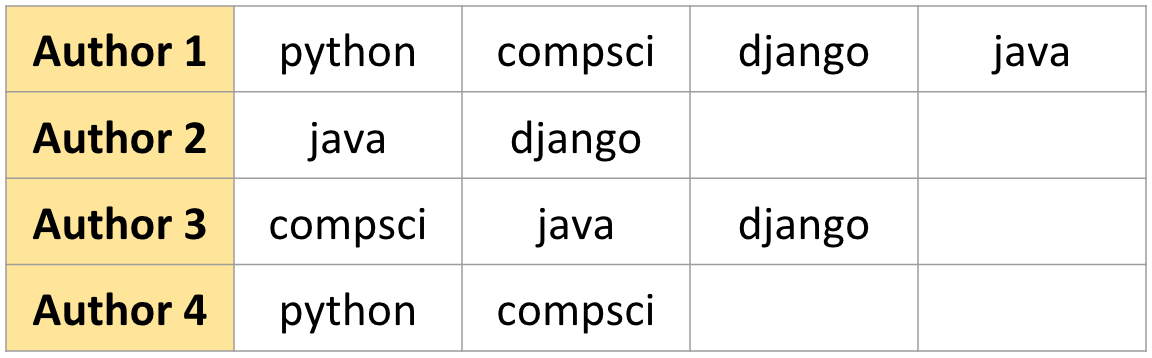






Figure 7 : Illustrate of example dataset applying transformation to association rules

After we transform to a new dataset in .csv file extension. The next step is to apply Apriori for the association rule for the recommendation engine.

**Apriori Algorithm**

Generating association rules**,** Apriori has split up into 2 steps:

* Step 1: Finding all frequent itemset and ***minimum support*** of given databases.
  + This step seem to be very challenging for us, because finding all frequent itemsets in a large database is difficult to searching all possible itemsets (~2.5millions- records). The set of possible itemsets is the power set over I and has size 2n -1 (excluding author with only 1 subreddit in the itemsets), efficient search is possible using the downward-closure property of support, which guarantees that for a frequent itemset.
* Step 2: Using frequent itemset and ***minimum confidence*** to form a rules
  + This step will generating an association rules along with the recommendation results

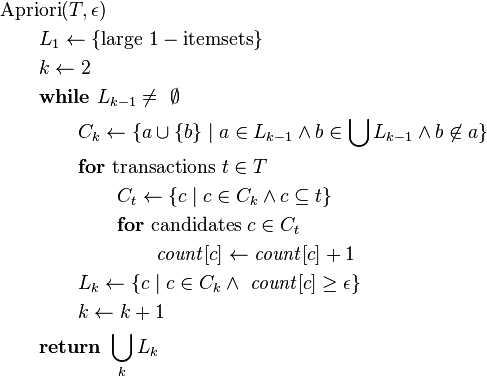
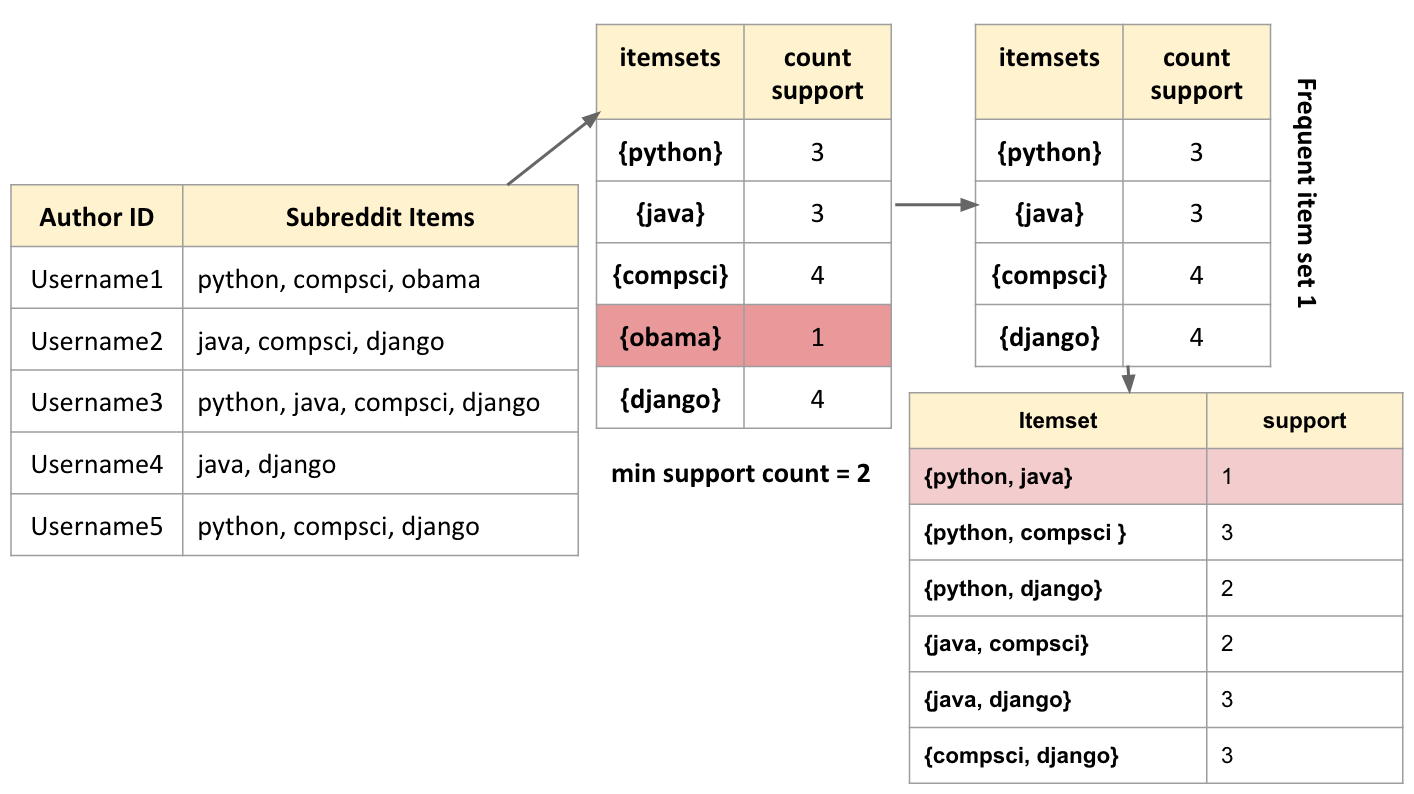


Figure 8: Apriori Algorithm Pseudocode [11]

Apriori uses BFS or breadth-first search and a tree structure to count candidate itemsets efficiently. By searching all of the item in the transaction database, its generates candidate itemsets of length K from itemsets of length K-1. Then it prunes the candidates which have an infrequent sub pattern. In addition, candidate generation generates a large numbers of subsets, and explore any maximal subset S only after all 2 |S| - 1 of its proper subsets.



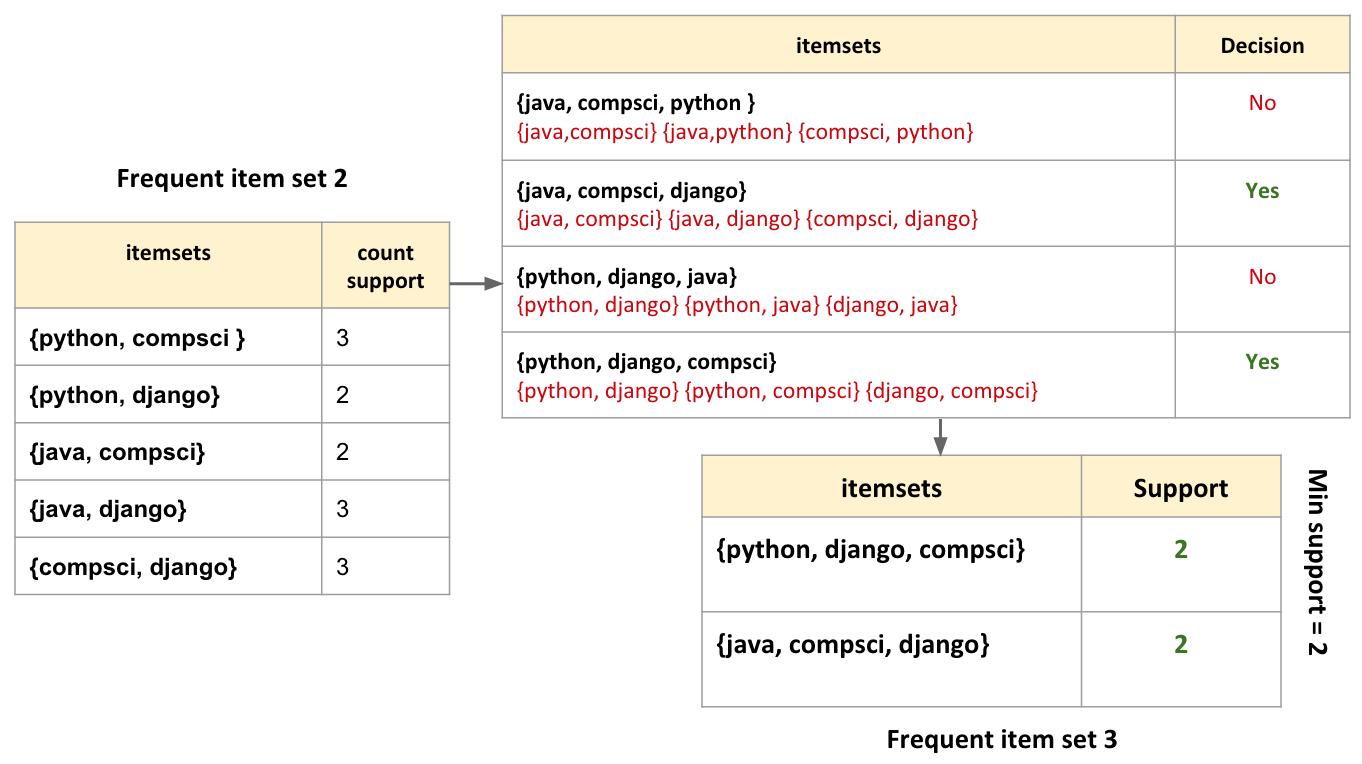


Figure 9: Illustrate workflow of Apriori Algorithm applied with our sample data

**Result and Knowledge**

The output of apriori now gives us the data we need to host Recommendit as a Python app, as shown in Figure 10. A barebones online version is, at the time of this writing, hosted at “kilgallin.com/?key=python” for example. This version is suitable to be used as a web service or integrated into a richer application. Its also important to note that the underlying data will quickly become stale as the Reddit communities evolve, and repeated mining would be necessary to maintain such a service.

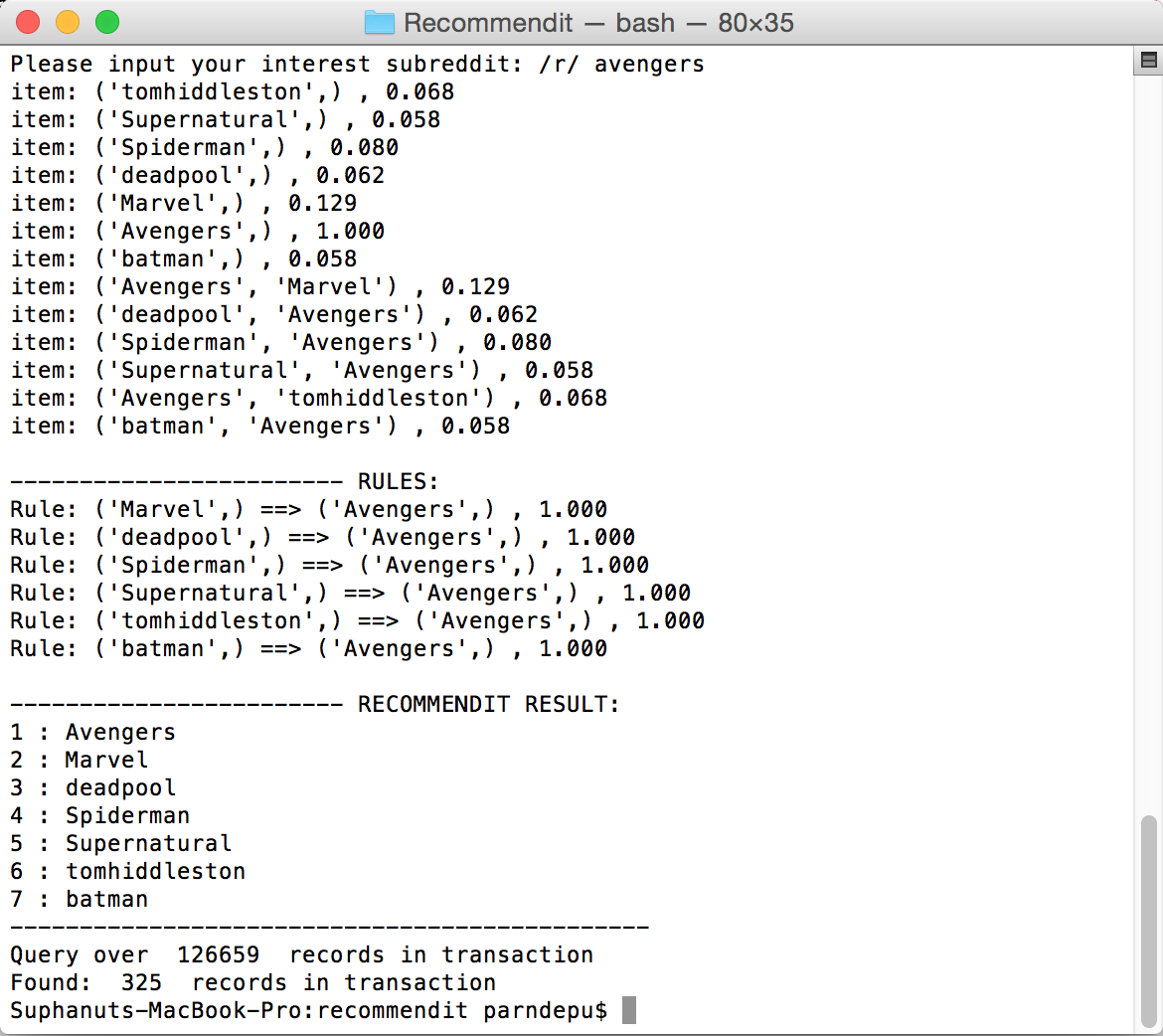


Figure 10: Recommendit in Python

**Discussion and Conclusion**

While our process appears to be sound for the data we collect, there are some limitations that could be overcome in future work. First, by only looking at the 2500 largest subreddits, we miss out on the ability to suggest new or small subreddits. Since these are the subreddits that are typically the hardest to discover on one’s own, this is a potentially valuable application of our program - perhaps even one of the most valuable from a discovery viewpoint. However, since the 2500th subreddit has only 0.01% the user count of the top few subreddits, the next 6000 subreddits will, cumulatively, only correspond to the number of users in the top 4-6 subreddits. The overhead to scrape and store the data for these subreddits is thus not justified for the scope of this project. Second, our web app architecture as written is limited in its ability to serve a large amount of traffic. Since reddit has several million unique daily visitors and hundreds of millions of unique monthly visitors, were this application to gain traction among the userbase, this would be a significant problem that would require, at the least, many servers and a load-balancing solution. Again, though, this concern is beyond the scope of our project as it stands. Within the scope of our project, though, we have achieved positive outcomes and would like to see it used more widely, as we believe it has potential to be of use or interest to many communities and reddit users.

**References**

[1] Sundaresan et al. *Subreddit Recommendations within Reddit Communitie*s. Stanford University. http://web.stanford.edu/class/cs224w/projects/cs224w-16-final.pdf

[2] “ben444422”. github.com. https://github.com/ben444422/Recommendit/

[3] Reddit list. redditlist.com

[4] Reddit API documentation. https://www.reddit.com/dev/api.

[5] “Umbrae”. *Reddit List of Top 2.5 Million Posts.* https://github.com/umbrae/reddit-top-2.5-million

[6] “asaini”. github.com. https://github.com/asaini/Apriori.

[7] CherryPy. http://cherrypy.org/

[8] PRAW. praw.org

[9] Rakesh Agrawal and Ramakrishnan Srikant. *Fast algorithms for mining association rules in large databases*. Proceedings of the 20th International Conference on Very Large Data Bases, VLDB, pages 487-499, Santiago, Chile, September 1994.

[10] Mining Frequent Itemssets, Laboratory module 8 http://software.ucv.ro/~cmihaescu/ro/teaching/AIR/docs/Lab8-Apriori.pdf

[11] Apriori Algorithm. http://en.wikipedia.org/wiki/Apriori\_algorithm

**Appendix I**

“””

subrtime.py  
“””

import praw

import csv

import time

import sqlite3

user\_agent = "Get top subreddit / Suphanut & JD"

r = praw.Reddit(user\_agent=user\_agent)

databasename = 'alltimepost\_data.db'

sql = sqlite3.connect(databasename)

cur = sql.cursor()

cur.execute(('CREATE TABLE IF NOT EXISTS reddit\_post(created TEXT, score INT, author TEXT, num\_comments INT,'

'over\_18 INT, subreddit TEXT, domain TEXT)'))

def insert\_toDB(created, author, subreddit, score, nsfw, num\_comment, url):

postdata = [created,score,author,num\_comment,nsfw,subreddit,url]

print postdata

cur.execute('INSERT INTO reddit\_post VALUES(?, ?, ?, ?, ?, ?, ?)', postdata)

sql.commit()

def searchtop(header):

post\_count = 0

# edit limit to 100 if you want to

while True:

try:

submissions = r.get\_subreddit(header).get\_top\_from\_all(limit=1000)

except:

traceback.print\_exc()

print('resuming in 5...')

time.sleep(5)

continue

break

for post in submissions:

if not post.author:

name = '[deleted]'

else:

name = post.author.name

created\_time = time.strftime("%D", time.localtime(float(post.created\_utc)))

author = name

subreddit = post.subreddit.display\_name

score = post.score

nsfw = post.over\_18

num\_comment = post.num\_comments

url = post.domain

insert\_toDB(created\_time, author, subreddit, score, nsfw, num\_comment, url)

post\_count+=1

print "getting total: ", post\_count, " post"

def main():

r = praw.Reddit(user\_agent = user\_agent)

# edit your redditlist file

with open('redditlist.txt') as f:

subreddit\_list = f.readlines()

#count = 0

for x in subreddit\_list:

subreddit\_name = x.split()

header = subreddit\_name[0]

print "Start searching top 1000 post in /r/", header

searchtop(header)

if \_\_name\_\_ == '\_\_main\_\_':

main()

**Appendix II**

"""

recommedit.py

"""

import sys

import csv

import errno

import apriori

def addto\_resultfile(result):

with open("test.txt", "a") as myfile:

myfile.write(result)

def compute\_list(records):

item\_list = []

#print "test apriori"

for data in records:

if data is not None:

item\_list.append(data)

return item\_list

def spliting\_result(result):

strResult = ""

for subr in result:

strResult+= subr+","

return strResult

def query\_subreddit(subreddit, records):

subreddit\_set = set()

for x in records:

subreddit\_set.add(x)

if subreddit in map(str.lower, subreddit\_set):

return True

else:

return False

def write\_tempfile(data,temp\_filename):

#print author

#print subreddit\_list

streddit = ""

f = open(temp\_filename,"a") #opens file with name of "test.txt"

for x in data:

streddit += x + ","

#print streddit

f.write(streddit+"\n")

f.close()

def delete\_tempfile(temp\_filename):

f = open(temp\_filename, "w")

f.truncate()

f.close()

def main():

""" start calling function """

# file path specification

"""

print "Please input your interest subreddit: /r/",

subreddit = raw\_input().lower()"""

with open('redditlist.txt') as f:

subreddit\_list = f.readlines()

#count = 0

for x in subreddit\_list:

subreddit\_name = x.split()

subreddit = subreddit\_name[0].lower()

print subreddit

filename = 'transaction.csv'

items\_data = []

try:

# create temporary file for user

temp\_filename = 'transaction\_temp.csv'

with open(filename, 'rb') as content:

reader = csv.reader(content)

content\_list = list(reader)

count = 0

found\_count = 0

# create csv file here

for records in content\_list:

count+=1

if query\_subreddit(subreddit, records):

temp = compute\_list(records)

items\_data.append(temp)

found\_count+=1

# writing temporary file for calculation

for data in items\_data:

write\_tempfile(data,temp\_filename)

content.close()

# test on priori

""" SET THE SUPPORT AND CONFIDENCE HERE!

DEFAULT: support = 0.15 confidence = 0.6

runApriori(temp\_filename, support, confidence)

"""

items, rules = apriori.runApriori(apriori.dataFromFile(temp\_filename), 0.05, 0.6)

lastresult = apriori.printResults(items, rules)

print "\n------------------------ RECOMMENDIT RESULT:"

index = 1

final = ""

final += subreddit + " ::: " + spliting\_result(lastresult) + "\n----------------------------------------------\n"

addto\_resultfile(final)

#print strResult

#print "----------------------------------------------"

print "Query over ", count," records in transaction"

print "Found: ",found\_count, " records in transaction"

# delete csv file here

delete\_tempfile(temp\_filename)

except IOError as exc:

if exc.errno != errno.EISDIR: # Do not fail if a directory is found, just ignore it.

print filename," not found"

raise # Propagate other kinds of IOError.

if \_\_name\_\_ == '\_\_main\_\_':

main()

**Apriori.py**

"""

Description : Simple Python implementation of the Apriori Algorithm

Usage:

$python apriori.py -f DATASET.csv -s minSupport -c minConfidence

$python apriori.py -f DATASET.csv -s 0.15 -c 0.6

"""

import sys

from itertools import chain, combinations

from collections import defaultdict

from optparse import OptionParser

def add\_Recommendation(itemset,recommendation\_set,seen\_recommendation):

""" FOR RECOMMENDIT ENGINE """

if len(itemset) >= 2:

for subreddit in itemset:

if subreddit not in seen\_recommendation:

recommendation\_set.append(subreddit)

seen\_recommendation.add(subreddit)

return recommendation\_set

def subsets(arr):

""" Returns non empty subsets of arr"""

return chain(\*[combinations(arr, i + 1) for i, a in enumerate(arr)])

def returnItemsWithMinSupport(itemSet, transactionList, minSupport, freqSet):

"""calculates the support for items in the itemSet and returns a subset

of the itemSet each of whose elements satisfies the minimum support"""

\_itemSet = set()

localSet = defaultdict(int)

for item in itemSet:

for transaction in transactionList:

if item.issubset(transaction):

freqSet[item] += 1

localSet[item] += 1

for item, count in localSet.items():

support = float(count)/len(transactionList)

if support >= minSupport:

\_itemSet.add(item)

#print \_itemSet

return \_itemSet

def joinSet(itemSet, length):

"""Join a set with itself and returns the n-element itemsets"""

return set([i.union(j) for i in itemSet for j in itemSet if len(i.union(j)) == length])

def getItemSetTransactionList(data\_iterator):

transactionList = list()

itemSet = set()

for record in data\_iterator:

transaction = frozenset(record)

transactionList.append(transaction)

for item in transaction:

itemSet.add(frozenset([item])) # Generate 1-itemSets

return itemSet, transactionList

def runApriori(data\_iter, minSupport, minConfidence):

"""

run the apriori algorithm. data\_iter is a record iterator

Return both:

- items (tuple, support)

- rules ((pretuple, posttuple), confidence)

"""

itemSet, transactionList = getItemSetTransactionList(data\_iter)

freqSet = defaultdict(int)

largeSet = dict()

# Global dictionary which stores (key=n-itemSets,value=support)

# which satisfy minSupport

assocRules = dict()

# Dictionary which stores Association Rules

oneCSet = returnItemsWithMinSupport(itemSet,

transactionList,

minSupport,

freqSet)

currentLSet = oneCSet

k = 2

while(currentLSet != set([])):

largeSet[k-1] = currentLSet

currentLSet = joinSet(currentLSet, k)

currentCSet = returnItemsWithMinSupport(currentLSet,

transactionList,

minSupport,

freqSet)

currentLSet = currentCSet

k = k + 1

def getSupport(item):

"""local function which Returns the support of an item"""

return float(freqSet[item])/len(transactionList)

toRetItems = []

for key, value in largeSet.items():

toRetItems.extend([(tuple(item), getSupport(item))

for item in value])

toRetRules = []

for key, value in largeSet.items()[1:]:

for item in value:

\_subsets = map(frozenset, [x for x in subsets(item)])

for element in \_subsets:

remain = item.difference(element)

if len(remain) > 0:

confidence = getSupport(item)/getSupport(element)

if confidence >= minConfidence:

toRetRules.append(((tuple(element), tuple(remain)),

confidence))

return toRetItems, toRetRules

def printResults(items, rules):

"""prints the generated itemsets and the confidence rules"""

recommendation\_set = []

seen\_recommendation = set()

for item, support in items:

add\_Recommendation(item,recommendation\_set,seen\_recommendation)

#print "item: %s , %.3f" % (str(item), support)

#print "\n------------------------ RULES:"

for rule, confidence in rules:

pre, post = rule

return recommendation\_set

#print "Rule: %s ==> %s , %.3f" % (str(pre), str(post), confidence)

def dataFromFile(fname):

"""Function which reads from the file and yields a generator"""

file\_iter = open(fname, 'rU')

for line in file\_iter:

line = line.strip().rstrip(',') # Remove trailing comma

record = frozenset(line.split(','))

yield record

if \_\_name\_\_ == "\_\_main\_\_":

optparser = OptionParser()

optparser.add\_option('-f', '--inputFile',

dest='input',

help='filename containing csv',

default=None)

optparser.add\_option('-s', '--minSupport',

dest='minS',

help='minimum support value',

default=0.15,

type='float')

optparser.add\_option('-c', '--minConfidence',

dest='minC',

help='minimum confidence value',

default=0.6,

type='float')

(options, args) = optparser.parse\_args()

inFile = None

if options.input is None:

inFile = sys.stdin

elif options.input is not None:

inFile = dataFromFile(options.input)

else:

print 'No dataset filename specified, system with exit\n'

sys.exit('System will exit')

minSupport = options.minS

minConfidence = options.minC

items, rules = runApriori(inFile, minSupport, minConfidence)

printResults(items, rules)