CS300 – Spring 2020-2021 - Sabancı University

Homework #3 – Personalized Recommendation System for TV Show Streaming Platforms

Due: 8 May 2021, 23:50

Brief description

In this homework we, as a software developer for an online TV show streaming platform, we ask you to develop a tool which will help us to improve our personalized recommendation system. We will provide you the viewing history of our customers as an input file and your tool needs to extract all the preference relations between the TV shows. For this purpose, you are required to use a **templated** hash table with **separate chaining**.

Overview

When we watch TV shows or movies, we generally have a genre or type which we prefer to watch more frequently. These preferences can differ for every customer based on their taste and personality. A customer might prefer to watch sitcoms like Friends, HIMYM or Seinfeld for most of the time. But another customer might prefer to watch the Sci-Fi series like Fringe or Person of Interest. Understanding these watching preferences can assist to the streaming platform in a few ways.

For instance, if X and Y are frequently watched by the same customers:

- X and Y can be both placed into the same categories or lists in the platform, so that customers who watched one of them would be prompted to watch the other TV show.
- Y could be in the recommendation list for customers who have already watched X.
- If the personalized recommendation list for the customers suits their tastes and preferences more, they will be more satisfied from the streaming platform and they will subscribe to the platform for longer periods.

The Goal

The goal of this homework, for a given list of TV shows from the viewing history, is to extract all possible implications. For instance;

```
Fleabag → Normal People
```

implies that, for the customers who watched Fleabag most likely watched Normal People, too. Or, as another example, consider the following implication;

```
Fleabag, After Life → BoJack Horseman
```

implies that, the customers who watched Fleabag and After Life most likely watched `BoJack Horseman`, too.

Input File Format

Your program needs to ask two numbers (ranging from 0 and 1) for given support and confidence threshold values (see their definitions in the document for details below) from the user. Then, your program needs to ask the name of the input file. You can decide on the appropriate display messages for them.
The input file contains the list of TV shows from the viewing history of our customers. In each line, there are several TV shows that are watched by the customer on the streaming platform. For a viewing history, there are no repeated TV shows and the number of TV shows for every customer are not fixed. An example of input file is given as below:

input.txt

```
friends himym  simpsons south_park
six_feet_under  peaky_blinders westworld american_gods
walking_dead  lost    fringe
fleabag normal_people bojack_horseman
office seinfeld big_bang_theory
rick_and_morty avatar south_park true_detective
```

In the input file above, there are no empty input lines, and all of the words are given in **lowercase**. The words in the same line are separated with one or more empty spaces or tabs. An item will be **only one word**, so if the name of the TV show contains more than one word, it is written without any spaces and separated by an underscore "`_`". Therefore, you will not see an item containing multiple words such as "`Six Feet Under`". It will be written in the file as "`six_feet_under`".

Output File Format

The output file format needs to be **exactly the same** as described here. You need to write your outputs to a file named **`results.txt`**. In each line, there will be only one rule. For each rule, you need to put commas between the TV shows on the same side of the implication and you need to put a semicolon (**`;`**) instead of the implication symbol (→). So, basically, what we expect from you is to generate the file as given in the second column of the below table if the extracted implications are as given in the first column of the table. There will be no space or tab character in the output file. Moreover, you need to give the confidence values of rules in the output file. Round your confidence values having 2 decimals, i.e., having 2 digits after the dot as below. Continue reading this document for more details on confidence values.

Extracted Rules with Confidences		**Output File Format**
A → B	confidence = 0.30123	A;B;0.30
{ B, C } → D	confidence = 0.12139	B,C;D;0.12
{ B, C } → { A, D }	confidence = 0.32125	B,C;A,D;0.32

Background Information

Before going into detail with the algorithm, we need to define two concepts: *support* and *confidence*.

Support describes how an item (or set of items) are popular among the whole item set. Hence, the support value of an item (or set of items) is the ratio of the number of times the item appeared among all the transactions. For instance, if there are 10 customers viewing history in the list and Fleabag is purchased 4 times, the support value of Fleabag is 0.4. On the other hand, if you need to calculate the support value of several items together, such as Fleabag and Normal People, you need to find the number of lines which contain both Fleabag and Normal People together to find the support value of them.

```
support(X)= number of lines containing X / number of all lines

support(X,Y)= number of lines containing X and Y together/number of all lines
```

Confidence value describes how likely item Y is watched when item X is watched, expressed as `X->Y`. This is calculated as:

```
confidence(X -> Y) = support(X, Y) / support(X)
```

For an implication having multiple items, such as `{X, Y} -> {Z, W}`, confidence is calculated as:

```
confidence({X, Y} -> {Z, W}) = support(X, Y, Z, W) / support(X, Y)
```

For instance, for the given input file below (input1.txt);

input1.txt

```
friends himym seinfeld
walking_dead got black_mirror
office seinfeld friends
fleabag bojack_horseman normal_people
friends westworld normal_people fleabag
```

Support values are calculated as:

```
support(Friends) = 3/5 = 0.6

support(Seinfeld) = 2/5 = 0.4

support(Friends, Seinfeld) = 2/5 = 0.4
```

The confidence of `{Friends → Seinfeld}` is calculated as:

```
confidence(Friends → Seinfeld) = support(Friends, Seinfeld) /
                                 support(Friends) = 0.4 / 0.6 = 0.66
```

whereas the confidence of `{ Seinfeld → Friends }` becomes,

```
confidence(Seinfeld -> Friends) = support(Seinfeld, Friends) /
                                  support(Seinfeld) = 0.4 / 0.4 = 1.0
```

So, `confidence(Friends → Seinfeld) = 0.66` means that 2/3 of the people who watched Friends also watched Seinfeld.

And, `confidence(Seinfeld → Friends) = 1.0` means that all the people who watched Seinfeld also watched Friends.

The Flow of the Program

The program has 2 main parts: in the first part, you need to find the support value of several item sets such as support({Fleabag})and support({PersonOfInterest, Fringe}).Then, by using those support values, you need to calculate the confidence values and extract the rules.

1) Calculating Support Values

- Find all support values of TV shows from the viewing history list such as, `support({Fleabag })` and `support({NormalPeople})`.
- Store all the TV shows whose support value is greater than or equal to the given support threshold value.
- Find all possible TV show pairs of the TV shows (selected in the previous step by comparing their support value and the given support value).
- Find all support values of TV show pairs in the viewing history list such as, `support({HIMYM, Friends}) and support({Fleabag, Normal People})`.
- Store all the items pairs whose support value is greater than or equal to the given support threshold value.

2) Rule Extraction Process

Up to this point, you have discovered all items and item pairs whose support values are greater than a given support threshold value. Assume that all of these items and item pairs are stored in the **same** hash table named as *lookupTable*.

Using *lookupTable* (a hash table) you need to enumerate all possible 2 way permutations of all the *lookupTable* elements. For example, assume that in the hash table, we have the following elements:

`{A}, {B}, {C}, {A, B}` and `{B, C}`

You need to find all possible 2 length permutations of these elements and construct the rules as below:

```
   A->B
   A->C
A -> {B, C}
   B->A
   B->C
   C->A
   C->B
C -> {A, B}
{A, B} -> C
{B, C} -> A
```

However, note that, there cannot be the same item on both sides of the implication. For example, the following rules are **not valid**:

```
 A -> {A, B}
{A, B} -> {B, C}
 {A, B} -> A
```

After enumerating all possible rules, you need to calculate the confidence values of them. Note that you may need to calculate the support value of item sets having 3 or 4 elements in that process. Simply, make another input file reading and calculate the support values of only those item sets i.e., not every 3 or 4 length item sets. For example, for the rules given above, you need to read the input file again to find the support value of only item set: {A, B , C}.

The output of your program will be the rules whose support values are greater the given confidence threshold value.

AN EXAMPLE EXECUTION OF THE PROGRAM

First, we take support and confidence threshold values from the user:

```
Support threshold = 0.25
Confidence threshold = 0.7
```

Then we read the input file:

```
friends himym seinfeld
walking_dead got black_mirror
office seinfeld friends
fleabag bojack_horseman normal_people
friends westworld normal_people fleabag
```

After that, the support values for all the TV shows will be calculated as follows:

`support(Friends)=0.6 , support(Seinfeld)=0.4 , support(GOT)=0.2 …`

Then we will take the ones whose support value is above the threshold (which is 0.25),

`support(Friends)=0.6 , support(Seinfeld)=0.4,`

`support(Fleabag)=0.4, support(NormalPeople)=0.4`

After that we will calculate the support values for pairs and again take the ones above the threshold value,

`support(Friends,Seinfeld)=0.4, support(Fleabag, NormalPeople)=0.4`

~~support(Friends,Fleabag)=0.2~~ (and others below the threshold)

Then we will find the possible 2-way permutations,

`{Friends} -> {Seinfeld}, {Friends} -> {Fleabag}, {Friends} -> {NormalPeople}`

`{Friends} -> {Fleabag, NormalPeople}, {Seinfeld} -> { Friends },`

`{Seinfeld } -> {Fleabag}, { Seinfeld } -> {NormalPeople},`

`{Seinfeld} -> {Fleabag, NormalPeople}, (and the other 8)`

After that we will calculate the support values for sets with 3 or 4 elements (if it is needed),

`support(Friends, Fleabag, NormalPeople) = 0.2`

`support(Seinfeld, Fleabag, NormalPeople) = 0.0`

`support(Fleabag, Friends,Seinfeld) = 0.0`

`support(NormalPeople, Friends,Seinfeld) = 0.0`

Then we will calculate the confidence values for every permutation found before,

confidence(Friends -> Seinfeld) = 0.66

confidence(Friends -> {Fleabag, NormalPeople}) = 0.33

confidence(Seinfeld -> Friends) = 1.0

Lastly, we will take the ones above confidence threshold value and print as result.txt with the given format,

result.txt

```
seinfeld;friends;1.00
fleabag;normal_people;1.00
normal_people;fleabag;1.00
```

About the Format of the Result File

For the given example result files below, all the are the same will be accepted as true, **(THESE ARE NOT RELATED WITH THE EXAMPLE EXECUTION ABOVE)**

Order of **lines** can be different.

```
normal_people;fleabag;1.00
fleabag;normal_people;1.00
seinfeld;friends;1.00
friends;normal_people,fleabag;0.33
```

=

```
normal_people;fleabag;1.00
friends;normal_people,fleabag;0.33
fleabag;normal_people;1.00
seinfeld;friends;1.00
```

Order of **TV shows in the same set** (items on the same side of the implication) can be different.

```
normal_people;fleabag;1.00
friends;normal_people,fleabag;0.33
fleabag;normal_people;1.00
seinfeld;friends;1.00
```

=

```
normal_people;fleabag;1.00
friends;fleabag,normal_people;0.33
fleabag;normal_people;1.00
seinfeld;friends;1.00
```

However, the order of the **TV shows on the different side of the implication** **cannot** be different.

```
normal_people;fleabag;1.00
fleabag;normal_people;1.00
seinfeld;friends;1.00
friends;normal_people,fleabag;0.33
```

≠

```
normal_people;fleabag;1.00
fleabag;normal_people;1.00
friends;seinfeld;1.00
friends;normal_people,fleabag;0.33
```

Sample Runs

Sample Run 1

input1.txt

```
friends himym seinfeld
walking_dead got black_mirror
office seinfeld friends
fleabag bojack_horseman normal_people
friends westworld normal_people fleabag
```

For the given input file above, the output will be:

```
Please enter the transaction file name: input1.txt
Please enter support and confidence values between 0 and 1: 0.25 0.5
7 rules are written to results.txt
```

result.txt

```
fleabag;friends;0.50
normal_people;friends;0.50
friends;seinfeld;0.67
seinfeld;friends;1.00
fleabag,normal_people;friends;0.50
fleabag;normal_people;1.00
normal_people;fleabag;1.00
```

Sample Run 2

input2.txt

```
lost prison_break fargo
prison_break got sherlock
westworld walking_dead fringe
seinfeld friends breaking_bad dexter
true_detective
lost true_detective prison_break
friends seinfeld hannibal
got mindhunter black_mirror
himym office big_bang_theory
prison_break lost hannibal seinfeld
true_detective
lost himym prison_break
prison_break lost breaking_bad
lost peaky_blinders
rick_and_morty seinfeld true_detective
prison_break lost
```

For the given input file input2.txt above, the output will be:

```
Please enter the transaction file name: input2.txt
Please enter support and confidence values between 0 and 1: 0.10 0.55
24 rules are written to results.txt
```

result.txt

```
true_detective;lost;0.75
true_detective;seinfeld;0.75
seinfeld;true_detective;0.75
true_detective;prison_break;0.75
lost,seinfeld;true_detective;1.00
true_detective;lost,prison_break;0.75
seinfeld,prison_break;true_detective;1.00
friends;seinfeld;1.00
hannibal;seinfeld;1.00
lost;prison_break;0.86
prison_break;lost;0.86
true_detective,seinfeld;lost;0.67
true_detective,prison_break;lost;1.00
seinfeld,prison_break;lost;1.00
true_detective,lost;seinfeld;0.67
true_detective,prison_break;seinfeld;0.67
true_detective,lost;prison_break;1.00
true_detective,seinfeld;prison_break;0.67
lost,seinfeld;prison_break;1.00
true_detective,lost;seinfeld,prison_break;0.67
seinfeld,prison_break;true_detective,lost;1.00
true_detective,seinfeld;lost,prison_break;0.67
true_detective,prison_break;lost,seinfeld;0.67
lost,seinfeld;true_detective,prison_break;1.00
```

Sample Run 3

For the given input file input2.txt above, the output will be:

```
Please enter the transaction file name: input2.txt
Please enter support and confidence values between 0 and 1: 0.20 0.60
14 rules are written to results.txt
```

result.txt

```
true_detective;lost;0.75
true_detective;seinfeld;0.75
seinfeld;true_detective;0.75
true_detective;prison_break;0.75
true_detective;lost,prison_break;0.75
lost;prison_break;0.86
prison_break;lost;0.86
true_detective,seinfeld;lost;0.67
true_detective,prison_break;lost;1.00
true_detective,lost;seinfeld;0.67
true_detective,prison_break;seinfeld;0.67
true_detective,lost;prison_break;1.00
true_detective,seinfeld;prison_break;0.67
true_detective,seinfeld;lost,prison_break;0.67
```

Sample Run 4

For the given input file above, the output will be:

```
Please enter the transaction file name: in1.txt
Please enter support and confidence values between 0 and 1: 0.60 0.45
There is no rule for the given support and confidence values.
```

Frequently Asked Questions

- **How am I going to use the hash table?**

Since the search function takes constant time in the hash table, you will use it to find the number of occurrences of items and item pairs. You will create a hash table like *lookupTable* in the example and use it while calculating confidence values.

- **Can I generate all item pairs while I am reading the input file?**

No. You need to enumerate all item pairs from the items whose support values are greater than the given threshold value.

- **How am I going to find the support value of k items?**

You need to find the number of lines in the input file which contains all of the k items at the same time. Then, divide this number with the number of total lines in input file. Normally, what you have in your hash table contains only items and item pairs.

Thus, you need to read the input file again to find the support values of item sets when needed.

- **Do I need to make input validation?**

No, you can assume that all of the given input values are valid.

- **Do I need to print the rules in the same order as you did in the sample run?**

No, you do not need to do that as long as you find the same rules.

General Rules and Guidelines about Homeworks

The following rules and guidelines will apply to all homework unless otherwise noted.

How to get help?

You may ask questions to TAs (Teaching Assistants) of CS300. Office hours of TAs can be found here. Recitations will partially be dedicated to clarifying the issues related to homework, so it is to your benefit to attend recitations.

What and Where to Submit

Please see the detailed instructions below/on the next page. The submission steps will get natural/easy for later homework.

Grading and Objections

Careful about the semi-automatic grading: Your programs will be graded using a semi-automated system. Therefore, you should follow the guidelines about input and output order; moreover, you should also use the exact same prompts as given in the Sample Runs. Otherwise, the semi-automated grading process will fail for your homework, and you may get a zero, or in the best scenario, you will lose points.

Grading:

- Late penalty is 10% off the full grade and only one late day is allowed.
- **Having a correct program is necessary, but not sufficient to get the full grade. Comments, indentation, meaningful and understandable identifier names, informative introduction and prompts, and especially proper use of required functions, unnecessarily long programs (which is bad) and unnecessary code duplications will also affect your grade.**
- Please submit your own work only (even if it is not working). It is really easy to find out “similar” programs!
- For detailed rules and course policy on plagiarism, please check out http://myweb.sabanciuniv.edu/gulsend/courses/cs201/plagiarism/

Plagiarism will not be tolerated!

Grade announcements: Grades will be posted in SUCourse, and you will get an Announcement at the same time. You will find the grading policy and test cases in that announcement.

Grade objections: Since we will grade your homework with a demo session, there will be very likely no further objection to your grade once determined during the demo.

What and where to submit (IMPORTANT)

Submission guidelines are below. Most parts of the grading process are automatic. Students are expected to strictly follow these guidelines to have a smooth grading process. If you do not follow these guidelines, depending on the severity of the problem created during the grading process, 5 or more penalty points are to be deducted from the grade.
Add your name to the program: It is a good practice to write your name and last name somewhere in the beginning program (as a comment line of course).

Name your submission file:

- Use only English alphabet letters, digits or underscore in the file names. Do not use a blank, Turkish characters or any other special symbols or characters.
- Name your cpp file that contains your program as follows.
  "**SUCourseUserName_yourLastname_yourName_HWnumber.cpp**"
- Your SUCourse user name is your SUNet username which is used for checking sabanciuniv e-mails. Do NOT use any spaces, non-ASCII and Turkish characters in the file name. For example, if your SUCourse user name is cago, name is Çağlayan, and last name is Özbugsızkodyazaroğlu, then the file name must be:
  cago_ozbugsizkodyazaroglu_caglayan_hw2.cpp
- Do not add any other character or phrase to the file name.
- Make sure that this file is the latest version of your homework program.
- You need to submit ALL .cpp and .h files including the data structure files in addition to your main.cpp in your project.

Submission:

Submit via SUCourse+ ONLY! You will receive no credits if you submit by other means (e-mail, paper, etc.).

Successful submission is one of the requirements of the homework. If for some reason, you cannot successfully submit your homework and we cannot grade it, your grade will be 0.

Good Luck!
Efe Öztaban and Gülşen Demiröz