Mini Project

Content-based filtering.

It relies on similarities between features of the items. It recommends items to a customer based on previously rated highest items by the same customer. List of features about these items needs to be generated.

- Each item will have an item profile
- A table structure will list these properties
- Comparing what and how many features match and collect scores
- Recommend highest scored item
- Code will be based on an algorithm, by given some item, the most similar item will be found
- Best scoring match will be provided to the user
- This method relies on item features only, and not the user preferences.

Collaborative filtering.

It is a family of algorithms where there are multiple ways to find similar users or items and multiple ways to calculate rating based on

ratings of similar users. Depending on the choices you make, you end up with a type of collaborative filtering approach.
It relies on how other users responded to these same items. It doesn’t rely of features of the item, but the preferences from other users. Similar users survey needs to be done.

- Users will have a table with different rated items of what they choose or liked
- Based on the similarities, prediction can be make of what the user might like, based on what similar users did.
- The list will be filtered and matched to users who used the same items for comparison and recommendations
- Everything will be summed up and highest score will be recommended
- Code will be created based on an algorithm, by given a user x, recommend an item that x might like
- Item with the highest score will be recommended

Dataset:-

Our Million Songs Dataset contains of two files: triplet_file and metadata_file. The triplet_file

contains user_id, song_id and listen time. The metadat_file contains song_id, title, release_by and artist_name. Million Songs Dataset is a mixture of song from various website with the rating that users gave after listening to the song.
Our first job is to integrate our dataset, which is very important every time we want to build a data processing pipeline.To integrate both triplet_file and metadata_file, we are going to use a popular Python library called pandas

```
triplets_file = 'https://static.turi.com/datasets/
millionsong/10000.txt'
songs_metadata_file = 'https://static.turi.com/
datasets/millionsong/song_data.csv'
```

What is the difference between item-to-item collaborative filtering and content-based filtering?

In simple terms Item Based collaboration deals with the other user actions on the item you are looking at or buying. This type of filtering happens generally simultaneously and the attributes of the product doesn't have the importance in recommending . For ex- I am buying a ceiling Fan and then the system starts recommending me to buy a light (this is because many people who buy ceiling fans are also buying lights and not because light and ceiling fan are related , this information is generally extracted from the transcript of users)

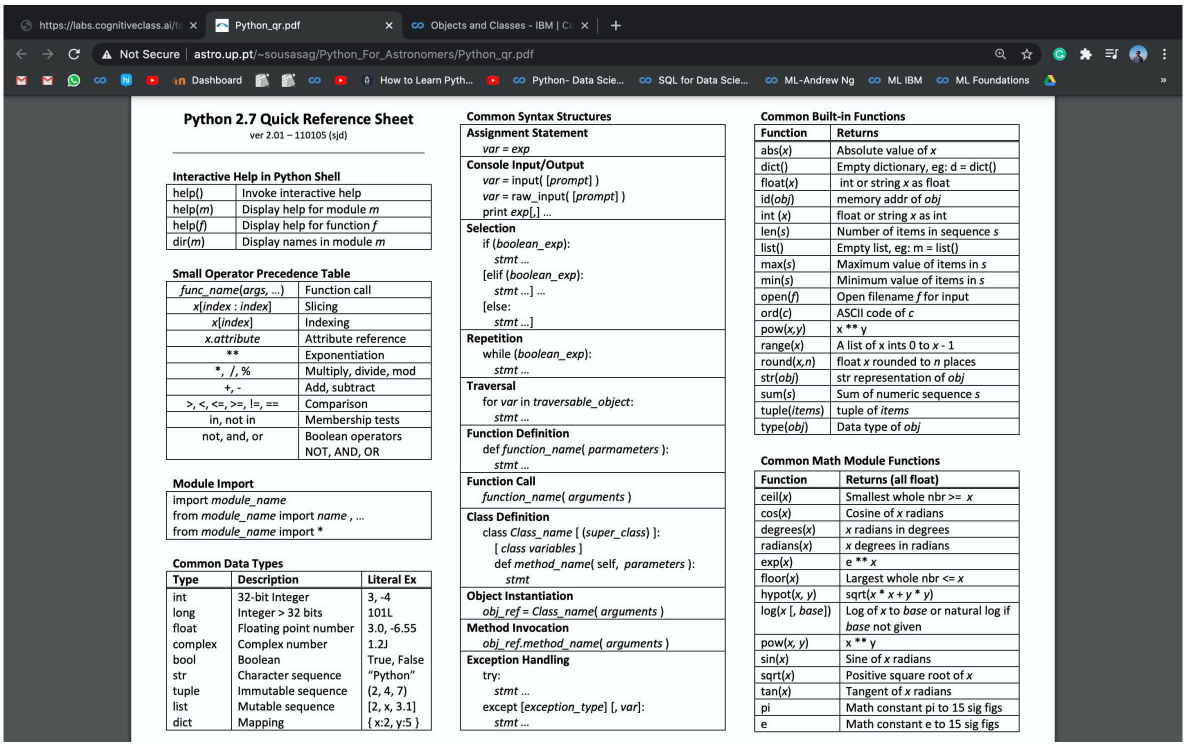
Whereas when we talk about content based filtering , generally the pre-defined attributes of the products are matched and similar products will be recommended . For Ex- When a user buys a Cannon D450 Camera the system starts recommending lenses, other similar model camera (These recommendations are based on the fact that only those products related to the main item in some attributes like model or compatible lens etc . , and also these details about the product are taken from the stored data)

Setup-DATASET:-

```
    import os
    import urllib.request
    from zipfile import ZipFile


    HOME_DIRECTORY = os.path.join('datasets','raw')
    ROOT_URL = 'https://os.unil.cloud.switch.ch/fma/
fma_metadata.zip'


    if not os.path.isdir(HOME_DIRECTORY):
                         os.makedirs(HOME_DIRECTORY)
    zip_path = os.path.join(HOME_DIRECTORY,
'data.zip')
    urllib.request.urlretrieve(ROOT_URL, zip_path)


    with ZipFile(zip_path, 'r') as zip:
          zip.extractall(HOME_DIRECTORY)
          print("Done!")
```

PYTHON IMPORTING AND MODULES(Short notes):

https://www.csee.umbc.edu/courses/331/fall10/notes/python/python3.ppt.pdf



Python 2.7 Quick Reference Sheet
ver 2.01 – 110105 (sjd)

Interactive Help in Python Shell

help()	Invoke interactive help
help(*m*)	Display help for module *m*
help(*f*)	Display help for function *f*
dir(*m*)	Display names in module *m*

Small Operator Precedence Table

func_name(*args, …*)	Function call
x[*index* : *index*]	Slicing
x[*index*]	Indexing
x.attribute	Attribute reference
**	Exponentiation
*, /, %	Multiply, divide, mod
+, -	Add, subtract
>, <, <=, >=, !=, ==	Comparison
in, not in	Membership tests
not, and, or	Boolean operators NOT, AND, OR

Module Import

import *module_name*
from *module_name* import *name* , …
from *module_name* import *

Common Data Types

Type	Description	Literal Ex
int	32-bit Integer	3, -4
long	Integer > 32 bits	101L
float	Floating point number	3.0, -6.55
complex	Complex number	1.2J
bool	Boolean	True, False
str	Character sequence	"Python"
tuple	Immutable sequence	(2, 4, 7)
list	Mutable sequence	[2, x, 3.1]
dict	Mapping	{ x:2, y:5 }

Common Syntax Structures

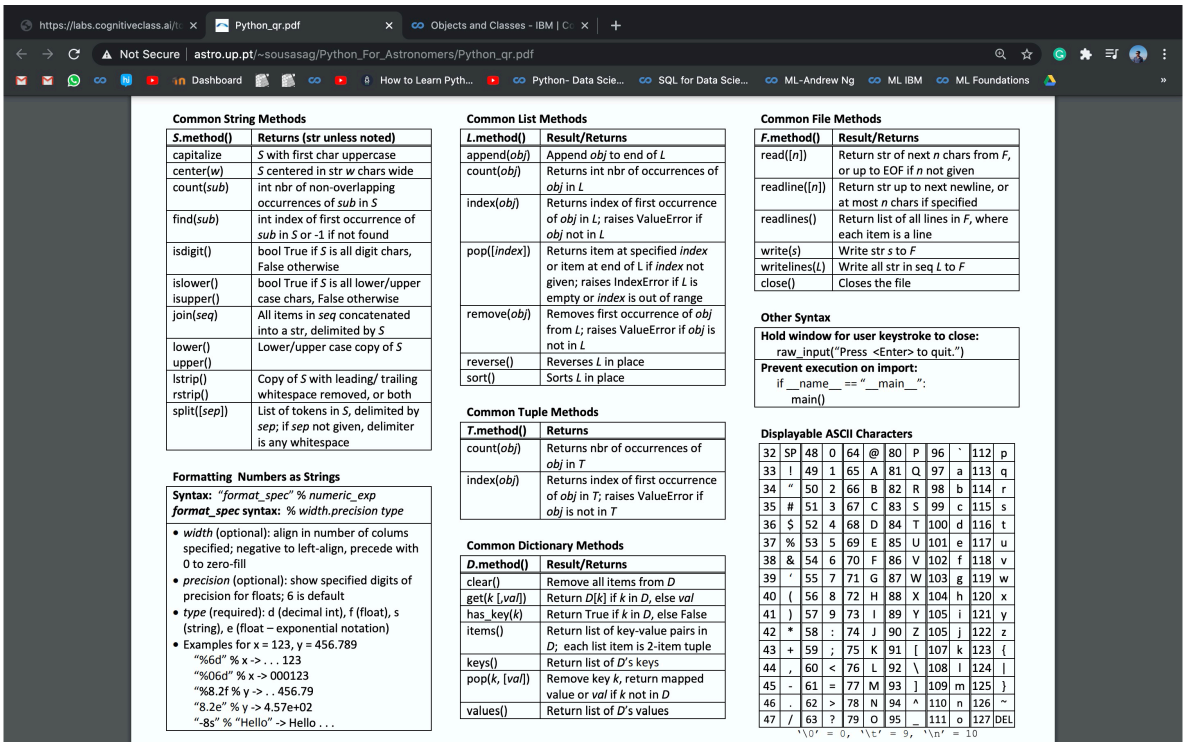
Assignment Statement
```
    var = exp
```
Console Input/Output
```
    var = input( [prompt] )
    var = raw_input( [prompt] )
    print exp[,] …
```
Selection
```
    if (boolean_exp):
        stmt …
    [elif (boolean_exp):
        stmt …] …
    [else:
        stmt …]
```
Repetition
```
    while (boolean_exp):
        stmt …
```
Traversal
```
    for var in traversable_object:
        stmt …
```
Function Definition
```
    def function_name( parmameters ):
        stmt …
```
Function Call
```
    function_name( arguments )
```
Class Definition
```
    class Class_name [ (super_class) ]:
        [ class variables ]
        def method_name( self,  parameters ):
            stmt
```
Object Instantiation
```
    obj_ref = Class_name( arguments )
```
Method Invocation
```
    obj_ref.method_name( arguments )
```
Exception Handling
```
    try:
        stmt …
    except [exception_type] [, var]:
        stmt …
```

Common Built-in Functions

Function	Returns
abs(*x*)	Absolute value of *x*
dict()	Empty dictionary, eg: d = dict()
float(*x*)	int or string *x* as float
id(*obj*)	memory addr of *obj*
int (*x*)	float or string *x* as int
len(*s*)	Number of items in sequence *s*
list()	Empty list, eg: m = list()
max(*s*)	Maximum value of items in *s*
min(*s*)	Minimum value of items in *s*
open(*f*)	Open filename *f* for input
ord(*c*)	ASCII code of *c*
pow(*x,y*)	x ** y
range(*x*)	A list of x ints 0 to *x* - 1
round(*x,n*)	float *x* rounded to *n* places
str(*obj*)	str representation of *obj*
sum(*s*)	Sum of numeric sequence *s*
tuple(*items*)	tuple of *items*
type(*obj*)	Data type of *obj*

Common Math Module Functions

Function	Returns (all float)
ceil(*x*)	Smallest whole nbr >= *x*
cos(*x*)	Cosine of *x* radians
degrees(*x*)	*x* radians in degrees
radians(*x*)	*x* degrees in radians
exp(*x*)	e ** *x*
floor(*x*)	Largest whole nbr <= *x*
hypot(*x, y*)	sqrt(*x* * *x* + *y* * *y*)
log(*x* [, *base*])	Log of *x* to *base* or natural log if *base* not given
pow(*x, y*)	x ** y
sin(*x*)	Sine of *x* radians
sqrt(*x*)	Positive square root of *x*
tan(*x*)	Tangent of *x* radians
pi	Math constant pi to 15 sig figs
e	Math constant e to 15 sig figs



Common String Methods

S.method()	Returns (str unless noted)
capitalize	*S* with first char uppercase
center(*w*)	*S* centered in str *w* chars wide
count(*sub*)	int nbr of non-overlapping occurrences of *sub* in *S*
find(*sub*)	int index of first occurrence of *sub* in *S* or -1 if not found
isdigit()	bool True if *S* is all digit chars, False otherwise

islower()
isupper()
	bool True if *S* is all lower/upper case chars, False otherwise

join(*seq*)	All items in *seq* concatenated into a str, delimited by *S*

lower()
upper()
	Lower/upper case copy of *S*

lstrip()
rstrip()
	Copy of *S* with leading/ trailing whitespace removed, or both

split([*sep*])	List of tokens in *S*, delimited by *sep*; if *sep* not given, delimiter is any whitespace

Formatting Numbers as Strings

Syntax: *"format_spec"* % *numeric_exp*
format_spec **syntax:** % *width.precision type*

- *width* (optional): align in number of colums specified; negative to left-align, precede with 0 to zero-fill
- *precision* (optional): show specified digits of precision for floats; 6 is default
- *type* (required): d (decimal int), f (float), s (string), e (float – exponential notation)
- Examples for x = 123, y = 456.789
  "%6d" % x -> . . . 123
  "%06d" % x -> 000123
  "%8.2f % y -> . . 456.79
  "8.2e" % y -> 4.57e+02
  "-8s" % "Hello" -> Hello . . .

Common List Methods

L.method()	Result/Returns
append(*obj*)	Append *obj* to end of *L*
count(*obj*)	Returns int nbr of occurrences of *obj* in *L*
index(*obj*)	Returns index of first occurrence of *obj* in *L*; raises ValueError if *obj* not in *L*
pop([*index*])	Returns item at specified *index* or item at end of L if *index* not given; raises IndexError if *L* is empty or *index* is out of range
remove(*obj*)	Removes first occurrence of *obj* from *L*; raises ValueError if *obj* is not in *L*
reverse()	Reverses *L* in place
sort()	Sorts *L* in place

Common Tuple Methods

T.method()	Returns
count(*obj*)	Returns nbr of occurrences of *obj* in *T*
index(*obj*)	Returns index of first occurrence of *obj* in *T*; raises ValueError if *obj* is not in *T*

Common Dictionary Methods

D.method()	Result/Returns
clear()	Remove all items from *D*
get(*k* [,*val*])	Return *D*[*k*] if *k* in *D*, else *val*
has_key(*k*)	Return True if *k* in *D*, else False
items()	Return list of key-value pairs in *D*; each list item is 2-item tuple
keys()	Return list of *D*'s keys
pop(*k*, [*val*])	Remove key *k*, return mapped value or *val* if *k* not in *D*
values()	Return list of *D*'s values

Common File Methods

F.method()	Result/Returns
read([*n*])	Return str of next *n* chars from *F*, or up to EOF if *n* not given
readline([*n*])	Return str up to next newline, or at most *n* chars if specified
readlines()	Return list of all lines in *F*, where each item is a line
write(*s*)	Write str *s* to *F*
writelines(*L*)	Write all str in seq *L* to *F*
close()	Closes the file

Other Syntax

Hold window for user keystroke to close:
```
    raw_input("Press  <Enter> to quit.")
```
Prevent execution on import:
```
    if __name__ == "__main__":
        main()
```

Displayable ASCII Characters

32	SP	48	0	64	@	80	P	96	`	112	p
33	!	49	1	65	A	81	Q	97	a	113	q
34	"	50	2	66	B	82	R	98	b	114	r
35	#	51	3	67	C	83	S	99	c	115	s
36	$	52	4	68	D	84	T	100	d	116	t
37	%	53	5	69	E	85	U	101	e	117	u
38	&	54	6	70	F	86	V	102	f	118	v
39	'	55	7	71	G	87	W	103	g	119	w
40	(	56	8	72	H	88	X	104	h	120	x
41	)	57	9	73	I	89	Y	105	i	121	y
42	*	58	:	74	J	90	Z	105	j	122	z
43	+	59	;	75	K	91	[	107	k	123	{
44	,	60	<	76	L	92	\	108	l	124	|
45	-	61	=	77	M	93	]	109	m	125	}
46	.	62	>	78	N	94	^	110	n	126	~
47	/	63	?	79	O	95	_	111	o	127	DEL

'\0' = 0, '\t' = 9, '\n' = 10