**RECOMMNEDATION SYSTEMS**

While listening to spotify today, I realized that the song recommendation system isn't good and that there's lots of room for improvement. So I want to explore the topic from scratch. Hopefully someone will get inspired to write a better recommendation system.

What is a recommendation system fundamentally?

To define this, lets make some observations:

-There is a finite amount of options. Movies, books, blogs, etc.

-You cannot recommend the same thing twice, for obvious reasons.

-Basically everyone gets different recommendations. If we look at 2 people's youtube feed, there is no way that the recommended videos and the order (very important) of the recommended videos will be the same. This tells us that the output is highly dependent on previous actions.

So, lets build one.

Rather than taking the conventional approach by building a movie recommender, I am choosing to build a "Would you rather?" questions recommender because it is different and interesting.

Firstly, we steal a dataset from kaggle.

It is called all\_unique.csv, so I run this code to read it

import csv

with open('all\_unique.csv', mode ='r')as file:

csvFile = csv.reader(file)

data = list(csvFile)

data = data

print(data)

we get an output:

[['option\_a','votes\_a','option\_b','votes\_b'], ['Be painted by Van Gogh', '462554', 'Be painted by Da Vinci', '1121946'], ['Run 26 miles', '915383', 'Swim 5 miles', '901412'], ['Your partner have a homosexual affair', '985582', 'Your partner have a heterosexual affair ', '675683'], ['Have a Pepsi', '637479', 'Have a Coke', '1184397'], ['Watch only dramas for the rest of your life', '254102', 'Watch only comedies for the rest of your life', '1510175'], ['Burn to death', '554364', 'Drown ', '1210010'], ...]

Ok, we can exclude the 1st line; it just explains the format of the data. To recommend questions, we need to classify them and it can be quite hard to classify things based on words (but python libraries such as nltk make this remarkable easy, so interested readers are welcome to implement this). Luckily, we also have numbers, which gave me an idea to choose questions based on 2 features: controversy and popularity. We can define controversy as the difference in the % of votes\_a and votes\_b i.e.

controversy = abs(votes\_a-votes\_b)/(votes\_a+votes\_b). #abs to keep controversy positive

We can plot the controversy of every question on the y axis.

For popularity, we can just plot the ranks on the x axis with a 1 to 1 correspondence. Never mind. When I went deeper into the problem, I realised that this is not good (try to reason why, after you read the distance section). Instead, we could just use a 6 to 1 similarity for simplicity.

Note: I updated the code for data to

data = data [1:1201]

because we don't need the 1st line, and the number of votes drops drastically after the 1204th question which will mess with our algorithm. 1201 because 1200 is a nice, divisible number.

Now, if a user likes some question, we need to recommend them a question of similar nature. How to do this? This brings us to the heart of our system: distance. What is distance? It is basically how similar 2 values are. We can just use the following function:

from math import sqrt

def distance(points): #def similarity is a more apt name, so we'll

dist =0 #use that for the final code

for i in range(len(points[0])):

dist+=(points[0][i]-points[1][i])\*\*2

return sqrt(dist)

Note: The code is pretty self explanatory if you know that the distance function in 2d is root((x2-x1)^2+(y2-y1)^2), and in 3d, it is ((x2-x1)^2+(y2-y1)^2+(z2-z1)^2). The format of points is ((x11, x12, ... x1n), (x21, x22, ... x2n)) where n is the number of dimensions. For our code, points will just be ((x1, y1), (x2, y2)). The subscripts were included because we run out of alphabets.

I think we're almost done, we have clustered the points based on their features and we know how to find the similarity between 2 questions. We just need to find the next question based on our current question. One thing we could do is calculate the similarity to every other question and pick the minimum. That is too expensive though. 1202\*2 subtractions, 1202 additions, 1202\*2 multiplications and 1202 square roots for just changing the question once! Actually, I reckon this will be done instantly but this approach won't scale if we add more features or solve a more complex problem.

A simple approach would be to look at all data points at a set x interval. Say, you are at x=5, check only the points at x=4, x=5 & x=6. The size of the iterval depends on the other feature’s size compared to x which just gave me the idea of weights! Since the x value is 6x higher, it will dominate the similarity function. We like controversy more than popularity & since the domain of controversy is [0:1], we will multiply it by 12x.

To quote Suyog, “Talk is cheap, show me the code”.

def controversy(vote):

return 12\*abs(vote[0]-vote[1])/(vote[0]+vote[1])

#not to microoptimize, but I think (vote[0]+vote[1]) is faster than sum(vote)

votes = [(int(val[1]), int(val[3])) for val in data]

y = [controversy(vote) for vote in votes]

x = [item for sublist in ([i]\*6 for i in range(int(len(data)/6))) for item in sublist]

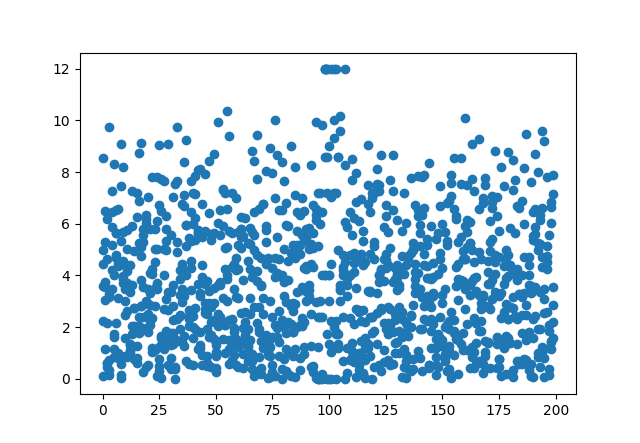
#x is just [0, 0,..., 6,6,..,..] it required some fancy list comprehension to do this #unfortuantely

import matplotlib.pyplot as plt

plt.scatter(x, y)

plt.show()

We get the following figure:



Now, for the final piece of the code:

def nextQuestion(currX, currY, userRating): #userRating is boolean

currVote = votes[currY]

distances = []

loc = currX #we need this a lot so might as well create a variable

for v in range(loc-6, loc+13): #I am aware of the edgecases...

if votes[v]!=currVote:

distances.append(similarity((currVote, votes[v])))

if(userRating):

nextQuestion = loc-6+distances.index(min(distances))

else:

nextQuestion = loc-6+distances.index(max(distances)) #if user doesn't like, pick the least similar

votes.pop(loc) #removing the last question to not ask it again

data.pop(loc)

x.pop(loc)

y.pop(loc)

return nextQuestion

If you’ve been paying attention so far, you could probably understand this yourself.

To test my code, I ran this, & we got pretty reasonable results:

def wouldYouRather():

Ques = 356

print("Would you rather ", data[Ques][0], " or ", data[Ques][2], "?")

for i in range(5):

print("Controversy: ", y[Ques], "\nPopularity: ", x[Ques])

Ques = nextQuestion(Ques, Ques, 1)

print("Would you rather ", data[Ques][0], " or ", data[Ques][2], "?")

wouldYouRather()

output:  
Would you rather Wear summer clothes in the winter or Wear winter clothes in the summer ?

Controversy: 4.209450627134632

Popularity: 59

Would you rather Be known for interrupting Taylor Swift or Be known for smacking Rihanna ?

Controversy: 4.432480821664424

Popularity: 58

Would you rather Appear on Oprah or Appear on Jerry Springer ?

Controversy: 4.2477250913055995

Popularity: 60

Would you rather Laugh uncontrollably at insurance commercials or Become wildly aroused at car commercials ?

Controversy: 3.170280158126059

Popularity: 59

Would you rather Be close with only one person, and only see them on Sundays or Know many people and see them every day, but not be particularly close with any ?

Controversy: 3.471197367577049

Popularity: 58

Would you rather Have to get around on all fours all the time or Only make animal noises ?