ICS Fall 2016

Final Exam

- This exam has **3 parts**
- You have **3 hours** to finish everything.
- You need to **download** the necessary files from NYU Classes
- **Upload** all your files to NYU classes at the end of the exam.

TURN OFF YOUR WIFI NOW. IF YOUR WIFI IS FOUND TO BE ON DURING THE EXAM, YOU WILL BE ASKED TO LEAVE AND YOU WILL RECEIVE 0 POINTS.

READ EACH QUESTION CAREFULLY BEFORE YOU PROCEED WITH SOLVING IT.

The Programming part has multiple questions with increasing difficulties. You do NOT have to finish one segment before moving on to the next; choose your best strategy.

Good luck!

Q1: Short Lecture Questions

Note: save your answers in **answers.txt**

1. Describe its rough steps of K-nearest neighbors; pseudo code preferred.

2. We have a dataset contains salary, age, educational background and many other parameters. We want to train the computer to intelligently guess new users' salary, given their other data, which method (regression, classification and clustering) should we use and why?

3. Give an example of a greedy algorithm, and explain why this algorithm might not be optimal.

Q2: Programming Questions

Q2.1: Recursion

Part 1.

The input `n` is a nonnegative integer, write a recursive function `sum_num(n)` to return the sum of all digits. I.e.:

```
$ python3 sum_num.py
345 :  12
89 :  17
```

Part 2.

The input `x` is a ***string*** made up by 1-digit integers and letters, write a recursive function `analyze(x, num=0, letters='')` to return the sum of contained 1-digit integers and a string that formed by all contained letters concatenated together. Example:

```
$ python3 analyze.py
1a2b3c: (6, 'abc')
4Shang56hai: (15, 'Shanghai')
```

Q2.2: OOP - Twitter Redesign

Design a simplified version of Twitter where users can post tweets, follow/unfollow another user and is able to see the 10 most recent tweets from each person that a user is following. Under the Twitter class, there is a all_users list that keeps track of the number of users in the system, and a all_tweets dictionary that saves all the tweets that each user posts.

The result might look like this

User class includes the following methods. We implemented some of them already.

1. `post_tweet(self, tweet)`: Allows a user to post a new tweet.
2. `get_news_feed(self)`: Retrieve the 10 most recent tweets from each person that a user is following. Each item in the news feed must be posted by users who the user followedf. Remember to handle the possible bug that might arise, if a person that the user follows does not have any tweets. Tweets must be *ordered from most recent to least recent*.
3. `follow(self, other)`: A user follows another user
4. `unfollow(self, other)`: A user unfollows another user.

Twitter class includes the following methods. We implemented some of them already.

1. `get_users(self)`: Return the list of all the users already in the Twitter class
2. `get_num_users(self)`: Return the number of users
3. `get_tweets(self, following)`: Return all the tweets posted by users in the following list passed into the method
4. `add_user(self,new)`: Add a new user to the Twitter class
5. `add_tweet(self, user, tweet)`: Add a new tweet under the user passed into the method

Your tasks are as following:

(1) Implement the `add_tweet` method under Twitter class
(2) Implement the `post_tweet` method under User class
(3) Implement the `get_news_feed` method under User class and the `get_tweets` method under Twitter class

Q3. Comprehensive programming

Q3.1 Epsilon-greedy tweet reader

Among the most useful strategies to deal with uncertainty is to combine exploration and exploitation strategically. This is the core of the ε-greedy algorithm; ε is a positive value smaller than 1:

With probability ε we pick randomly, otherwise we pick the current best choice

(The reinforcement Q learning in-class exercise actually uses this strategy.)

There are 3 twitter accounts besides you in Twitter. We provide a list of probabilities called the ground truth “interestingness” but it’s hidden from you. The “interestingness” level represents the probability that you find the account interesting. In the warm-up phase that we implemented in the code, we use a 50% “interestingness” as the *initial but wrong* estimation.

During each iteration, you will read the tweets from ***only one*** account, and update your estimation of its “interestingness” level. The “interestingness” of an account is how many times you find it interesting divided by the number of times you read it.

You’ll apply the ε-greedy algorithm to estimate the ground truth interestingness level of every account. To do that, you throw a dice and decide whether to 1) randomly pick an account to read, or 2) compute the “interestingness” of all accounts and pick the one account with the highest “interestingness” level.

Complete the code in `tweet_read.py`. Here is the output:

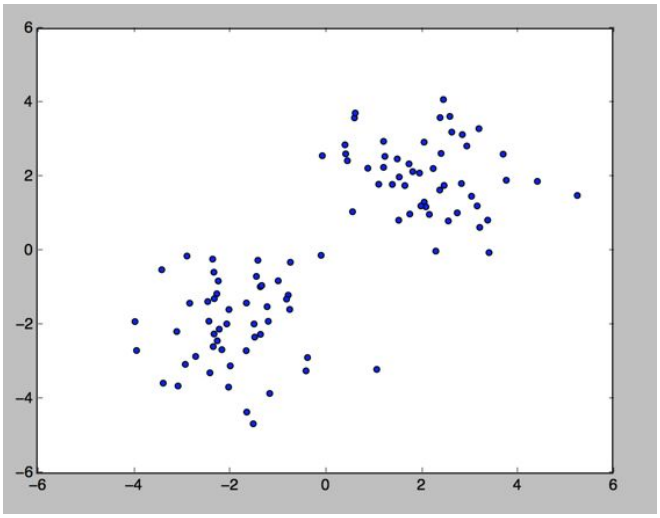
```
$ python3 tweet_read.py
+++++++ initial feeds
{'reads': 100, 'likes': 50.0}
{'reads': 100, 'likes': 50.0}
{'reads': 100, 'likes': 50.0}
------- ground truth interests:
[0.1, 0.6, 0.4]


++++++++ after reading  100000
0 {'reads': 3412, 'likes': 377.0}
1 {'reads': 93427, 'likes': 56195.0}
2 {'reads': 3461, 'likes': 1411.0}
-------- estimated interests
0 0.11049237983587339
1 0.6014856518993439
2 0.40768563998844265
```

Q3.2: Group the twitters

Note: part 1 and part 2 are must. Part 3 *might* be bonus, depending the performance of the class

Earlier we have asked you to implement a simple twitter community. Based the tweets, we can cluster the users according to their "features." The feature is an array of values representing the interests of a user.

There are 100 users, and we represent each with two features, as shown below:



Part 1: Parse the features

The file `tweet_features.txt` lists features of our users. Each line is the two features of a user. Complete the code in `group_tweet.py` to read the file, and build a dictionary `df`. `df[i]` is a two-element tuple containing the two features.

Here is the output of the first 10 users after passing the file.

```
$ python3 group_tweets.py
(2.9417154046806644, 0.679283792665065)
(0.6034218952988502, 0.5909683286784129)
(1.3202855519215788, 2.070938016993632)
(2.37050356746066, 3.6200914287046837)
(0.983651105811929, 2.5230956316925224)
(1.9278799772149287, 2.473787812435009)
(2.179196487274857, 2.2486021956535267)
(1.1689007847290118, 1.135860519977629)
(0.6909626355406413, 2.1853131806254416)
(2.193887741249104, 1.5784189790950336)
```

Part 2: how to cluster?

Describe in English: How will you apply K-means to cluster the users based on the features.

Part 3: clustering

Use K-means to cluster the users and set K as 2.