Panda Internet Radio

Introduction

Have you ever used Pandora Internet Radio? It's free. Go on, give it a go here (https://www.pandora.com).

The web app plays an endless parade of music (with occasional ads, of course), selecting the next song based on your responses ("likes" and "dislikes") to songs played previously. To make this work, the Pandora team manually curates a vast library of music as part of their Music Genome Project, labeling each song with some *genes* (from a set of 450 *genes* they deemed important, such as "Folk influences" and "Female lead vocalist"). By quantifying music via these *genes*, Pandora is able to create a meaningful comparison between any two songs and define a measure of similarity between music you seem to like and music you have not yet heard. You can read more about the Music Genome Project here (https://en.wikipedia.org/wiki/Music_Genome_Project). It's interesting stuff!

In this tutorial Lab, you will complete a music player reminiscent of Pandora: the Panda Internet Radio. We, the friendly 6.009 staff, have admittedly gotten carried away preparing a pretty user interface and curating a music genome project of 100 songs with our own simplistic "genes" (such as "Armando liked this song" and "Adam overheard it somewhere"). We seem to have neglected to implement the business logic of the player: given the user's "likes" and "dislikes," and our music genome, which song should play next? This is your lab assignment.

`lab.py`

You are expected to implement the entire lab in a single file, `lab.py`. The business logic of the Panda Internet Radio is accessed via a single Python function: `next_song(likes, dislikes, music)`, which takes a list of song IDs the user has liked, a list of song IDs the user disliked, and the music genome.

For example, consider a toy genome of 4 songs with 3 genes:

```
[[0,1,0],[0,1,1],[1,0,1],[1,1,0]]
```

If the first and currently playing song is 0 (the first song), and the user disliked it, your code will be invoked with:

```
next_song(likes=[],
          dislikes=[0],
          music=[[0,1,0],[0,1,1],[1,0,1],[1,1,0]])
```

This document describes in detail what `next_song` should do in order to correctly select the next best song from songs `{1,2,3}`, as `0` has already played. Which do you think should be selected? (Hint: it's 2.)

Your code will be loaded into a tiny web server (`server.py`) which, when running, serves the Panda Internet Radio user interfaces from your very own computer acting as a web server (at localhost:8000 (http://localhost:8000) - your computer's own address at port 8000).

Run `server.py` and go to the url http://localhost:8000 (http://localhost:8000) on Chrome.

To be clear, you are only implementing the code that gets the next song for Panda Radio. You are not responsible for any functions involved with UI. For example, when you click the "like" button for a given song, some complex machinery which we've created eventually queries your code via your `next_song` function.

Auto-grader (unit tester)

To verify the correctness of your implementation, we provide a little script `test.py` that loads your `lab.py` code, calls `next_song` with a few different inputs, and checks the output for correctness. The behavior of `next_song` is defined to be unambiguous (deterministic), meaning we can hard-code the expected outputs and compare against them in the unit tester. We encourage you to create your own tests to help you track down errors in your implementation.

In-Browser UI

The in-browser user interface (we use a browser to make sure everything works and looks nice on your machine, whatever it may be, within reason) knows how to play songs given your implementation of `next_song` and the library of songs in `resources/music.json`. You, the user, can "like" (and then skip) songs or "dislike" them, informing the inputs to `next_song`. Refreshing the page clears your history of "likes" and "dislikes," and re-loads your code in `lab.py`. The web UI will not function if your `next_song` crashes or returns incomprehensible nonsense.

The 6.009 Music Genome

We found a small pile of music (we hope you find something you like there!) and annotated it with our hastily conceived "genes." We weren't fancy: we used the approximate genre of each song, the decade in which it was released, as well as some arbitrary features such as "emo high school." While we certainly don't claim to be as good at suggesting music as the good people at Pandora, it seems to work quite well.

The music genome is a list of *genomes*, one genome per song. Each genome is an array of N *genes*, each of which is an integer in {0,1} where 1 denotes the presence of a gene and 0 denotes its absence. Each song's genome has the same number of entries in the genome array. A toy genome of two songs and three genes may look something like `[[1,0,0], [0,1,1]]`, where each of the 3 genes is some feature we've attributed to the songs. The user interface works with `static/music.json` that endows each song genome with a title and a YouTube (https://www.youtube.com/) video ID in order to play it through the browser UI.

Distance Function

Much like Pandora, we take care to define a notion of `distance` for two song genomes in order to compare songs quantitatively. Any given song's genome is an array of N elements, each in `{0,1}`, denoting the presence or absence of each gene.

Given two genomes, we define *distance* to be the sum of the absolute-value difference in each of their individual genes. For example, the distance between `[0,1]` and `[0,0]` is `1`, and the distance between `[1,0]` and `[0, 1]` is `2`.

When considering the distance between a song `s` and a set of "liked" songs L, we compute the average (mean) of the individual distances `distance(s,l)` for each song `l` in `L`. The same holds for the set of user "dislikes."

Selecting the Next Song: The "Goodness" Function

How do we select the next song to play? The project managers upstairs decided that we should select the song with a maximum *goodness* value given a set of "likes," and a set of "dislikes". Given our music genome, we define *goodness* of a song `s` as:

```
goodness(s, likes, dislikes) =
  average_distance(s, dislikes) - average_distance(s, likes)
```

In other words, we favor songs that have genomes dissimilar to the songs the user has disliked and similar to the set of songs the user liked. We select the song `s` from `(all_music - likes - dislikes)` that maximizes this notion of *goodness*. In case of multiple songs with equal and greatest *goodness*, select the one occurring earliest (by order in the array) in our music genome. For example, if all songs have the same genes, it would be appropriate to select the first unplayed song in the music genome list.

Troubleshooting

The in-browser UI for Panda Internet Radio streams songs from YouTube, and it may lag a second or two before the song plays. The artist and song title will display near the "like" and "dislike" buttons. YouTube being a commercial product, there may be an occasional advertisement break (guess what, Pandora has these too!).

If, for some reason, port `8000` is not available on your computer, you may change that number in `server.py:7` to any available port. If, for example, you change the port to `50005`, your new URL will become localhost:50005 (http://localhost:50005). Go wild!

When debugging your code, you may wish to familiarize yourself with Python's powerful debugger (PDB). You may find the PDB documentation (https://docs.python.org/2/library/pdb.html) helpful as reference, but it is not exactly a tutorial. By adding `import pdb` to the top of the `lab.py`, and placing the `pdb.set_trace()` statement wherever you wish to pause your program, the server will stop whenever this statement is encountered, allowing you to print variables, modify state, and evaluate arbitrary Python statements. This is the *debugger console*. Use `print myVariable` to inspect your code, and `next` and `step` to execute your program line by line. The "Debugger Commands" documentation section (https://docs.python.org/2/library/pdb.html#debugger-commands) describes how to interact with this console at length (the commands we find most helpful are `next`, `step`, `continue`, and `print`).

You may also wish to add your own tests for the various components of your Panda implementation, such as the distance function. Hard-code some expected results and run the test occasionally to make sure you don't break anything as you build your lab.

Do Something Cool

Once you've aced this laboratory, satisfying both the autograder and the LAs, why not improve your Panda Internet Radio? Consider investigating `resources/make_genome.py` and `resources/music.csv` to add your own songs to our genome or add your own genes. Consider also the goodness function: ours is quite simplistic, and Panda may become a better radio with a more sophisticated notion of "goodness." For example, how should Panda react to changing music preferences? We'd love to see what you come up with. (Do keep in mind, though, that our autograder will only mark answers as correct when they follow exactly the rules given in this document; so we suggest experimenting with improvements only *after* getting full credit in the autograder.)

Good luck!

Issues to Ponder in the Future:

- what happens after all played? specify
- Specify average distance to an empty set

In [ ]: