Live Nation Project Summary

Gregory Faletto
gfaletto@wustl.edu

During my internship with Live Nation, I created an algorithm to predict the future concert *set lists* of bands using their database on setlist.fm. A *set list* is an ordered list of the songs a band plays at a particular concert. Setlist.fm contains a database of past concert set lists of each band, including the date and the location of the concert.

The basic idea of my algorithm is that for every song the band has ever played in a concert listed on setlist.fm, a *song score* is created by taking a sum of the number of times that the band played the song at concerts in the setlist.fm database—except the sum is weighted by recency of the concert, so that more recent performances count more and less recent ones are discounted. This approach accounts for the fact that a band's new set lists are more likely to be similar to their most recent concerts, but still exploits all of the data in the database to some degree.

The details of how the algorithm works follow. The band must be specified; each band has an identification number called an *mbid* which can be found on the band's profile page on setlist.fm[1]. The user specifies the identification number in the header of the R script I wrote. Some other parameters which will be described later are also specified in the header.

Next the script can be run in R. The script makes a call to the setlist.fm API and downloads JSON files of all of the band's set lists in the database. The script uses the `jsonLite` R package to parse the data and creates an $n \times p$ matrix $\boldsymbol{A}$. Each of the n songs in the database has its own row in $\boldsymbol{A}$, and each of the p columns of $\boldsymbol{A}$ corresponds to a particular concert, ordered from most to least recent. Matrix element A_{ij} equals 1 if song i was played at concert j, and 0 otherwise.

Next a p-dimensional vector of recency weights $\boldsymbol{v}$ is created. Let the difference in time between the date the algorithm is run and concert j be Δt_j. Δt_j is calculated in years (though decimals are allowed) for each concert, and then

[1] One easy way to locate this is by searching on Google for the band's name and "setlist.fm," e.g., "Red Hot Chili Peppers setlist.fm."

v_j is calculated in the following way:

$$v_j = \left(\frac{1}{2}\right)^{\Delta t_j/\lambda} \tag{1}$$

where $\lambda > 0$ is a tuning parameter. Note that as Δt_j increases, v_j decreases; that is, the longer ago concerts have taken place, the less they will be weighted in the song score. Also note that since $0 \leq \Delta t_j < \infty$, $0 < v_j \leq 1$, so weights are always between 0 and 1.

The effect of λ is that for larger values the weights $\boldsymbol{v}$ are slower to decrease in value (so less recent concerts are less discounted). λ can be thought of as the "half-life" of weights for concerts; every λ years, the weight is multiplied by another power of $\frac{1}{2}$. Consider the end behavior of v_j as a function of λ:

$$\lim_{\lambda \to \infty} v_j = 1 \tag{2}$$

so for very large λ, the weights all approximately equal 1 regardless of Δt_j, so all concerts are weighted roughly equally regardless of elapsed time.

On the other hand,

$$\lim_{\lambda \to 0^+} v_j = 0 \text{ if } \Delta t_j > 0 \tag{3}$$

so for very small λ, all but the most recent concert have essentially 0 weight.

$\lambda = 5$ seemed to work fairly well, but this could be fiddled with more.

The total number of times song i was played at every concert in the database for this band is given by

$$\sum_{j=1}^{p} A_{ij} \tag{4}$$

If this sum is weighted by the recency weights $\boldsymbol{v}$, it yields the song score c_i for song i:

$$c_i = \sum_{j=1}^{p} A_{ij} v_j \tag{5}$$

This computation is done by the algorithm for every song i simultaneously by computing the product

$$\boldsymbol{A}\boldsymbol{v} = \boldsymbol{c} \tag{6}$$

where $\boldsymbol{c}$ is an n-dimensional vector of song scores.

Lastly, $\boldsymbol{c}$ is sorted in order of decreasing song score. Then the algorithm returns the names of the first ℓ elements of $\boldsymbol{c}$, which are the ℓ songs with the highest song scores, where ℓ is arbitrary (at the moment it returns 10 songs).

Examples:

```
> source(file="setlist.R")
[1] "Top ten most likely songs for Lorde :"
 [1] "Royals"            "Tennis Court"      "Ribs"
 [4] "Buzzcut Season"    "400 Lux"           "Glory and Gore"
 [7] "A World Alone"     "White Teeth Teens" "Team"
[10] "Biting Down"
> source(file="setlist.R")
[1] "Top ten most likely songs for alt-J :"
 [1] "Tessellate"      "Fitzpleasure"    "Breezeblocks"
 [4] "Matilda"         "Something Good"  "Dissolve Me"
 [7] "Taro"            "Bloodflood"      "❦ (Ripe & Ruin)"
[10] "Left Hand Free"
```

```
> source(file="setlist.R")
[1] "Top ten most likely songs for Red Hot Chili Peppers :"
 [1] "Give It Away"     "By the Way"       "Californication"
 [4] "Under the Bridge" "Can't Stop"       "Scar Tissue"
 [7] "Otherside"        "Right on Time"    "Dani California"
[10] "Around the World"
> source(file="setlist.R")
[1] "Top ten most likely songs for Metallica :"
 [1] "Seek & Destroy"          "Master of Puppets"
 [3] "Enter Sandman"           "One"
 [5] "The Ecstasy of Gold"     "Sad but True"
 [7] "Nothing Else Matters"    "For Whom the Bell Tolls"
 [9] "Creeping Death"          "Fade to Black"
```