Phish Setlist Generator

By Kenny Callaghan

Introduction: Problem

Phish is an American rock band formed in 1983 in Burlington, VT. Over the course of their 30+ years of prolific touring, the band has gained a reputation for their whimsical sense of humor, whacky onstage antics, and virtuistic instrumental improvisation. All of this has underscored the quintessential aspect of a Phish concert: every show is unique, and no two shows are quite the same. The easiest way to ensure this is to develop a unique setlist (ordered set) of songs for each of their 1700+ shows. When a show begins, no one can be sure what songs will be played - not even the members of the band themselves! What if there was a way to predict a setlist before-hand?

Solution

We would like to create an algorithm that can create Phish setlists from scratch - without any input from the band. We could do this by assembling a random list of songs, but we can do better. We can inform what song comes next in a setlist by looking at historical setlist data and extrapolating from there. We can start with a randomly-chosen song, either from a list of all previously-played songs, or from a smaller pool of all songs that have been used before as a set-opener. Then we can use a Natural Language Processing machine learning algorithm to predict the next songs that can be played in the setlist - similar to the word prediction system that is used by Google and other services to autocomplete searches or messages. We can do this a number of times until it reaches a natural conclusion.

Data:

The dataset that we are going to be working on is a list of every pair of songs ever performed together, formatted: "song_title1"+"song_title2" (song titles are in quotation marks, separated by a '+'). In the event that a given song is a set-opener (meaning it is the first song performed in a set), the first element in the pair will be "*****". If a song is a set-closer (the last performed song in a set), that string will occupy the second space in the pair. In order to format the data in this way, we begin with phish.net, which is a fan-run website run by a dedicated group of individuals whom catalogue everything Phish-related. All setlist data was scraped from this site and put in a sqlite database in third normal form. The code for performing this task can be found at this public github repository: https://github.com/kcallaghan1/Phish-DB.

In [20]:
```python
# This code will help for later

# Scientific and vector computation for python
import numpy as np

# Plotting library
from matplotlib import pyplot

import tensorflow as tf
from nltk.tokenize import RegexpTokenizer
from keras.models import Sequential, load_model
from keras.layers import LSTM
from keras.layers.core import Dense, Activation
from keras.optimizers import RMSprop
import pickle
import heapq
```

Pre-processing to desired format:

First, we need to get a list of songs. Running the following commands from a sqlite terminal will give us a list of Phish sets as well as the songs contained in those sets.

In [ ]:
```
.open Phish.db
```

In [ ]:
```
.mode csv
```

In [ ]:
```
.headers on
```

In [ ]:
```
.output songs.txt
```

In [ ]:
```sql
select setNumber, songName from (select * from songs join setlists using(songID));
```

Then, we can use the following code to create a new document that will include set-closers:

In [44]:
```python
def peek_line(f):
    pos = f.tell()
    line = f.readline()
    f.seek(pos)
    return line

def count_lines(f):
    pos = f.tell()
    count = len(f.readlines())
    f.seek(pos)
    return count

def create_pairs():
    songs = open("songs.txt", "r")
    pairs = open("pairs_old.txt", "w")
    line_count = count_lines(songs)

    #First two line of file are not useful
    songs.readline()
    line_count -= 1

    for i in range(line_count - 1): # Offset by 2 because first 2 lines are headers
        line = songs.readline()
        next_line = peek_line(songs)
        if(next_line):
            song = line.split(",", 1)[1]
            song = song[0: len(song) - 1]
            if("\"" not in song):
                song = "\"" + song + "\""


            nextSong = next_line.split(",", 1)[1]
            nextSong = nextSong[0: len(nextSong) - 1]
            if("\"" not in nextSong):
                nextSong = "\"" + nextSong + "\""

            if(line.split(",",1)[0] != next_line.split(",",1)[0]):
                nextSong = "\"*****\""
        pair = song + "+" + nextSong
        pairs.write(pair + "\n")

    #Final line:
    line = songs.readline()
    song = line.split(",", 1)[1]
    song = song[0: len(song) - 1]
    pairs.write("\"" + song + "\"" + "+" + "\"*****\"\n")
    songs.close()
    pairs.close()
```

In [45]:
```python
# Run the following code:
create_pairs()
```

This sets us up with set-closers, and this function will give us set-openers as well:

In [19]:
```python
def add_openers():
    pairs = open("pairs_old.txt", "r")
    pairs2 = open("pairs.txt", "w")

    lines = pairs.readlines()

    #Add first song as opener:
    pairs2.write("\"*****\"" + "+" + lines[0].split("+",1)[0] + "\n")

    for i in range(len(lines)):
        line = lines[i]
        one = line.split("+",1)[0]
        two = line.split("+",1)[1]
        pairs2.write(one + "+" + two)
        if(two == "\"*****\"\n"):
            if(i < len(lines) - 1):
                nextline = lines[i+1]
                pairs2.write("\"*****\"" + "+" + nextline.split("+",1)[0] + "\n")

    pairs.close()
    pairs2.close()
```

In [20]:
```python
#Run this code as well:
add_openers()
```

And now we have the data in the desired format in the pairs.txt file:

In [8]:
```python
def sampleData():
    pairs = open("pairs.txt", "r")
    lines = pairs.readlines()
    pairs.close()
    rand = np.random.randint(len(lines) - 1)

    for i in range(rand, rand+10):
        print(str(i) + ": " + lines[i][0:len(lines[i]) - 1])
```

In [25]:
```python
sampleData()
```

```
21279: "*****"+"Ya Mar"
21280: "Ya Mar"+"Julius"
21281: "Julius"+"Fee"
21282: "Fee"+"Taste"
21283: "Taste"+"Cavern"
21284: "Cavern"+"Stash"
21285: "Stash"+"The Lizards"
21286: "The Lizards"+"Free"
21287: "Free"+"Johnny B. Goode"
21288: "Johnny B. Goode"+"*****"
```

We will populate the list with an ordered set of all songs performed:

In [7]:
```python
pairs = open("pairs.txt", "r")
lines = pairs.readlines()
pairs.close()
songs = []
for i in range(0, len(lines), 2):
    songs.append(lines[i].split("+", 1)[0])
    songs.append(lines[i].split("+", 1)[1][0: len(lines[i].split("+", 1)[1]) - 1])
songs.append("\"*****\"")
```

Now, we will create a sorted dictionary of unique songs:

In [9]:
```python
unique_songs = np.unique(songs)
unique_song_index = dict((c, i) for i, c in enumerate(unique_songs))
```

In [112…
```python
SONG_LENGTH = 10
prev_songs = []
next_songs = []
for i in range(len(songs) - SONG_LENGTH):
    prev_songs.append(songs[i:i+SONG_LENGTH])
    next_songs.append(songs[i + SONG_LENGTH])
print(prev_songs[0])
print(next_songs[0])
```

```
['"*****"', '"Long Cool Woman in a Black Dress"', '"Proud Mary"', '"In the Midnight Hour"', '"Squeeze Box"',
'"Roadhouse Blues"', '"Happy Birthday to You"', '"*****"', '"Scarlet Begonias"', '"Fire on the Mountain"']
"*****"
```

In [113…
```python
X = np.zeros((len(prev_songs), SONG_LENGTH, len(unique_songs)), dtype=bool)
Y = np.zeros((len(next_songs), len(unique_songs)), dtype=bool)

for i, each_songs in enumerate(prev_songs):
    for j, each_song in enumerate(each_songs):
        X[i, j, unique_song_index[each_song]] = 1
    Y[i, unique_song_index[next_songs[i]]] = 1
```

In [13]:
```python
print(X[0][0])
```

```
[ True False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False]
```

In [114…
```python
model = Sequential()
model.add(LSTM(128, input_shape=(SONG_LENGTH, len(unique_songs))))
model.add(Dense(len(unique_songs)))
model.add(Activation("softmax"))
```

In [115…
```python
optimizer = RMSprop(lr=0.01)
model.compile(loss="categorical_crossentropy", optimizer=optimizer, metrics=["accuracy"])
history = model.fit(X, Y, validation_split=0.05, batch_size=128, epochs=2, shuffle=True).history
```

```
Epoch 1/2
296/296 [==============================] - 23s 57ms/step - loss: 5.0621 - accuracy: 0.1365 - val_loss: 5.451
8 - val_accuracy: 0.1530
Epoch 2/2
296/296 [==============================] - 15s 52ms/step - loss: 4.0542 - accuracy: 0.2113 - val_loss: 5.434
3 - val_accuracy: 0.1580
```

In [116…
```python
model.save("keras_next_song_model.h10")
pickle.dump(history, open("history.p", "wb"))

model = load_model("keras_next_song_model.h10")
history = pickle.load(open("history.p", "rb"))
```

```
WARNING:absl:Found untraced functions such as lstm_cell_7_layer_call_and_return_conditional_losses, lstm_cel
l_7_layer_call_fn, lstm_cell_7_layer_call_fn, lstm_cell_7_layer_call_and_return_conditional_losses, lstm_cel
l_7_layer_call_and_return_conditional_losses while saving (showing 5 of 5). These functions will not be dire
ctly callable after loading.
WARNING:absl:Found untraced functions such as lstm_cell_7_layer_call_and_return_conditional_losses, lstm_cel
l_7_layer_call_fn, lstm_cell_7_layer_call_fn, lstm_cell_7_layer_call_and_return_conditional_losses, lstm_cel
l_7_layer_call_and_return_conditional_losses while saving (showing 5 of 5). These functions will not be dire
ctly callable after loading.
INFO:tensorflow:Assets written to: keras_next_song_model.h10\assets
INFO:tensorflow:Assets written to: keras_next_song_model.h10\assets
```

In [117…
```python
def prepare_input(text):
    x = np.zeros((1, SONG_LENGTH, len(unique_songs)))
    for t, song in enumerate(text):
        #print(song)
        x[0, t, unique_song_index[song]] = 1
    return x

prepare_input(songs[0:5])
```

Out[117…
```
array([[[1., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        ...,
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.]]])
```

In [118…
```python
def sample(preds, top_n=3):
    preds = np.asarray(preds).astype("float64")
    preds = np.log(preds)
    exp_preds = np.exp(preds)
    preds = exp_preds / np.sum(exp_preds)

    return heapq.nlargest(top_n, range(len(preds)), preds.take)
```

In [119…
```python
def predict_completions(text, n=3):
    if(text == ""):
        return("0")
    x = prepare_input(text)
    preds = model.predict(x, verbose=0)[0]
    next_indices = sample(preds, n)
    return [unique_songs[idx] for idx in next_indices]
```

In [168…
```python

```

In [193…
```python
def generate_setlist():
    setlist = []
    #setlist.append(songs[np.random.randint(len(songs))])
    setlist.append("\"*****\"")
    go = True
    num = 1
    setEnt = 1
    idx = 1
    while(num < 4 and idx < 25):
        rand = np.random.randint(SONG_LENGTH)
        if(idx <= SONG_LENGTH):
            new_song = predict_completions(setlist[0:idx], SONG_LENGTH)[rand]
        else:
            new_song = predict_completions(setlist[idx - SONG_LENGTH: idx], SONG_LENGTH)[rand]
        if(new_song == "\"*****\"" and (setEnt > 6 or num >= 2)):
            num += 1
            setlist.append(new_song)
            setEnt = 0
        elif(new_song in setlist):
            idx -= 1
        else:
            setlist.append(new_song)
        idx += 1
        setEnt += 1
    return setlist
```

In [198…
```python
setlist = generate_setlist()
print(setlist)
```

```
['"*****"', '"Frankenstein"', '"Good Times Bad Times"', '"Lawn Boy"', '"Chalk Dust Torture"', '"Maze"', '"Ru
n Like an Antelope"', '"Mike\'s Song"', '"Harpua"', '"Weekapaug Groove"', '"The Horse"', '"Cold as Ice"',
'"Love You"', '"*****"', '"Sweet Adeline"', '"Big Black Furry Creature from Mars"', '"Runaway Jim"', '"Amazi
ng Grace"', '"Llama"', '"Sparkle"', '"The Landlady"', '"Suzy Greenberg"', '"It\'s Ice"', '"Horn"', '"Rift"']
```

In [ ]:
```

```