Twitch Chat Interpreter, demo

members:

Nate Yeo, 42099639, yeon@uci.edu

Jingtian Li, 46918353, jingtil2@uci.edu

Duo Chen, 92789693, duoc3@uci.edu

This is the demonstration for part of our twitch chat interpreter project. This demo will be a streamline of what we have accomplished. For time saving, the demonstrated part will only be what we feel worth showing. Outputs of parts not included are presented as screen snippets in the project report.

Project Purpose & First part

We aim to make a system that can automically clip from a stream vod by using chat, thus saving editors' time in finding potential clips. Selected stream vods are downloaded by using TwitchDownloader (third party software), files are in the project folder named "chatjsonfiles".

Our first approach was to gather stream clips based on chat speed. This is done using the Clipper.py module, whose terminal outputs are in report's screen snippets and whose files outputs are in the project folder "clip_data".

We want to train a learner to distinguish between good/bad clips from a collection of clips. To do this, we used labeler to manually label those clips. Terminal outputs are in project report and file outputs are in project folder "labeled_clip_data".

To numericalize string texts, we made a Embedding module by using gensim python package. We then trained it on the json files we have. Here is part of its results. NOTE: to run the cell for live demo of Embedding (second next), you will need the gensim package, numpy, scipy and nltk package.

In [1]:
```
# For this cell, you only need the python pickle package
# This cell demonstrates three words we have chosen to present their similar words and consine similarity
# vector size is 300 (too big to show), thus we will not show vectors in this cell output
import pickle
with open('data/EmbdDemo.pkl','rb') as f:
    demo=pickle.load(f)

print(f"Demonstrating [{len(demo)}] chosen words and 5 of their most similar words: ")
for word in demo.keys():
    print(f"For word [{word}]")
    for w in demo[word]: print(f">> [{w[0]}]:[{w[1]}]")
    print()

# Somethings about those words:
# lulw, omegalul, lmao, etc are variants of the word lol commonly used on Twitch
# kekw is a Twitch emote, if you want to know what it looks like, google image "kekw"
# pog is an emote on Twitch to express amazement
# pog has variants such as pogu, pogcrew etc.
# Many of these words are Twitch emotes, feel free to look them up.
```

```
Demonstrating [3] chosen words and 5 of their most similar words:
For word [lol]
>> [lulw]:[0.9854356050491333]
>> [fuck]:[0.9765660762786865]
>> [haha]:[0.974907398223877]
>> [omegalul]:[0.9736191630363464]
>> [lmao]:[0.9641276597976685]

For word [haha]
>> [lmao]:[0.9905922412872314]
>> [lulw]:[0.9809660315513611]
>> [omegalul]:[0.9793339967727661]
>> [lol]:[0.974907398223877]
>> [kekw]:[0.9713457822799683]

For word [pog]
>> [pogu]:[0.9807208180427551]
>> [pogcrew]:[0.9568219780921936]
>> [katielpog]:[0.9555957317352295]
>> [clutch]:[0.9483774900436401]
>> [kreygasm]:[0.9365337491035461]
```

In [2]:
```
# For this cell, you will need the gensim python package and many other packages specified above
# we want to demonstrate the complete w2v model we have, including some functions in Embedding for better rea
dibility
# How to use the terminal UI is somewhat included in itself, feel free to play around with it.
# If you want to save time in not installing gensim, just skip this cell
import Embedding
kv=Embedding.Load_wv('data/teo.kv')
Embedding.Check_trained_model(kv)
```

```
---------------------------------------------------------------
Vocabulary size is: [1804]
Enter either one word to find most similar or two to find similarity, enter 0 to exit
Enter here: lol
How many similar words do you want to see?: 5
Do you want to see the vector? (y/n): n
---------------------------------------------------------------
5 most similar words of [lol] are:
>>[lulw]: 0.9854356050491333
>>[fuck]: 0.9765660762786865
>>[haha]: 0.974907398223877
>>[omegalul]: 0.9736191630363464
>>[lmao]: 0.9641276597976685

---------------------------------------------------------------
Vocabulary size is: [1804]
Enter either one word to find most similar or two to find similarity, enter 0 to exit
Enter here: lol lmao
---------------------------------------------------------------
[lol]:[lmao] has similarity 0.9641276597976685
---------------------------------------------------------------
Vocabulary size is: [1804]
Enter either one word to find most similar or two to find similarity, enter 0 to exit
Enter here: nice
How many similar words do you want to see?: 5
Do you want to see the vector? (y/n): n
---------------------------------------------------------------
5 most similar words of [nice] are:
>>[lukas]: 0.951810896396637
>>[poggers]: 0.9405299425125122
>>[kreygasm]: 0.9384805560112
>>[ace]: 0.9324921369552612
>>[wow]: 0.927863359451294

---------------------------------------------------------------
Vocabulary size is: [1804]
Enter either one word to find most similar or two to find similarity, enter 0 to exit
Enter here: 0
```

In [3]:
```
# This cell demonstrate some word purification processes done in LogReg
# to run this cell, you need the python package "enchant" of spell checking
# simple purification process does not remove repeating letters while heavy purification does.
# for example, "Hello" would be changed to "Helo" by heavy purification.
# simple purification process does not edit numbers,
# while heavy purification returns you a special words if it receives a float number.
import log_regression_demo
cmd = input("what do you want to test? 1 -> simple purification; 2 -> heavy purification; enter other to qui
t")
while (cmd in ["1", "2"]):
    be_converted = input("please enter the string you wish to convert: ")
    print()
    print(log_regression_demo.demo_convert(be_converted, cmd))
    cmd = input("what do you want to test? 1 -> simple purification; 2 -> heavy purification; enter other to
 quit")
```

```
what do you want to test? 1 -> simple purification; 2 -> heavy purification; enter other to quit1
please enter the string you wish to convert: niiiiiceeeee

nice
what do you want to test? 1 -> simple purification; 2 -> heavy purification; enter other to quit2
please enter the string you wish to convert: pooooog

pog
what do you want to test? 1 -> simple purification; 2 -> heavy purification; enter other to quit0
```

Second part

Models we have tried are LogReg, MLP, RNN, GRU. Because we implemented those python modules to train AND test models in one go, we do not have enough time to re-write codes for this demonstration. But the typical outputs are included in project report.

We found that learner models does not have significant increase compared to default accuracy, and that many of our clips made from chat speed are noisy in terms of chat content.

This lead to the making of ClipperV2.py. In this approach, we identify clips by locating intervals of chats whose tokens have similar vector directions. Ideally, this should find clips where chats have homogeneous reactions.

Because it was close to the end of the quarter, we did not make a terminal UI based python model. Thus running this py file will directly output terminal results. The ClipperV2 used in demo is different from the one in project, thus the only additional packages needed for this part should be nltk, numpy and scipy.

In [4]:
```
%run ClipperV2.py
```

```
processing data
clipping vod
clip for file [data/teo7.pkl]
number of clips found: [29]
Clip number [1] is: [0:3:19] -> [0:5:14]
Clip number [2] is: [0:7:34] -> [0:8:56]
Clip number [3] is: [0:13:25] -> [0:14:29]
Clip number [4] is: [0:15:10] -> [0:16:8]
Clip number [5] is: [0:16:34] -> [0:17:32]
Clip number [6] is: [0:23:43] -> [0:24:41]
Clip number [7] is: [0:25:40] -> [0:27:38]
Clip number [8] is: [0:34:55] -> [0:36:2]
Clip number [9] is: [0:42:31] -> [0:43:29]
Clip number [10] is: [0:56:52] -> [0:57:50]
Clip number [11] is: [1:31:52] -> [1:32:59]
Clip number [12] is: [1:39:4] -> [1:40:2]
Clip number [13] is: [1:48:40] -> [1:49:38]
Clip number [14] is: [1:49:58] -> [1:50:56]
Clip number [15] is: [1:51:19] -> [1:52:59]
Clip number [16] is: [1:54:4] -> [1:55:2]
Clip number [17] is: [1:56:25] -> [1:57:23]
Clip number [18] is: [2:9:1] -> [2:10:14]
Clip number [19] is: [3:21:52] -> [3:22:50]
Clip number [20] is: [3:29:43] -> [3:30:41]
Clip number [21] is: [3:32:28] -> [3:33:26]
Clip number [22] is: [3:36:55] -> [3:37:53]
Clip number [23] is: [3:54:52] -> [3:55:50]
Clip number [24] is: [4:32:31] -> [4:33:32]
Clip number [25] is: [4:37:37] -> [4:38:35]
Clip number [26] is: [5:3:22] -> [5:4:20]
Clip number [27] is: [5:4:31] -> [5:5:47]
Clip number [28] is: [5:26:55] -> [5:27:53]
Clip number [29] is: [5:51:25] -> [5:52:59]
total time to edit is [0:32:41]
original vod duration is [5:53:9]
```

This brings back the project purpose, which is pre-processing stream vods into collection of clips. As you can see, a vod of 5 hours can be shrunken down to less than 1 hour (whose content we have checked) in unsupervised clipping. Hence, this approach is viable.

The vod URL is https://www.twitch.tv/videos/945481267 (https://www.twitch.tv/videos/945481267) However, Twitch delete out-dated vod to save storage. This vod is likely deleted by the time you are reading this.

In this vod, some clips (such as the ones in the beginning) are filled with greetings. Some other (such as clip number 19) is watch worthy. But the major improvement is that these are short and less noisy segments.

```
In [ ]:
```