Jiamin Li

SDS 136: Communicating with Data

Professor Amelia McNamara

December 22, 2017

How to Become YouTube Famous: Final Project Summary

Does clickbait work? *How to Become YouTube Famous* is a project that I originally started to see if there was any correlation between key words in video titles and video popularity. As I dug further into the data, however, the project quickly became more of an exploratory analysis to extract the most statistically interesting results. The final product thus became a series of word cloud visualizations that showcases that iterative process.

The data that I used comes from Mitchell J's Trending YouTube Video Statistics datasets, which is a collection of (up to) 200 listed trending YouTube videos every day in the United States, United Kingdom, Canada, Germany and France. The datasets are updated every few days and includes information such as video titles, channel name, number of views, number of comments, number of likes and dislikes, etc. For the sake of simplifying the data, I chose to only look at the US dataset for videos featured during the period between November 14, 2017 and December 12, 2017.

There are two fundamental ways of representing data. In *Intro to Functional Art*, Alberto Cairo makes a subtle distinction between infographics and visualizations. He argues that while both present data and allow a certain degree of exploration, infographics tend to be data-heavy while visualizations are more concerned with data exploration (Cairo 3). To represent the text data that I had, I used three different metrics of popularity – views, comments, and likes – to generate

pairs of word clouds. For each metric of popularity, the frequency of certain words would be reflected in the word clouds. Although using word clouds may skew the visual decoding of the data from the observer's perspective and contain many other weaknesses, they are impactful and familiar to many people. They are great storytelling tools to get people to start thinking about the data in meaningful ways. In other words, my project was more about allowing readers to explore and engage with otherwise boring data rather than delivering pre-identified trends in the data.

In *Computational Information Design*, Benjamin Fry summarizes the process of creating visualizations as: acquiring, parsing, filtering, mining, representing, refining, and interacting. The problem, however, is that these vital steps are often treated as isolated parts of the same problem. To create a good visualization, a single person needs to have adequate knowledge in each of the fields required to complete all of the steps (Fry 13). As someone who is not a statistician, I found the steps of filtering and mining the data to be very challenging. I often questioned my own methods, wondering how I could better represent the data so that it could be more accurate while maintaining its insightfulness. The following describes the methods I used to create my visualizations:

1. *Acquire* – I acquired the dataset from Mitchell J's Trending YouTube Video Statistics on Kaggle. The dataset is in a CSV file and contains at least 6,000 entries. Of these 6,000 entries, many appear more than once because they were featured more than once.
2. *Parse & Mine* – I parsed the CSV file using my own Python code. The code arbitrarily removes duplicate titles, filters out some stop words, bins the data into categories of low and high popularity (e.g. below/above average or top/bottom 25th quartile), and writes the parsed data into a new file.

```
def stopWords():
    with open('stopWords.txt') as textFile:
        lines = textFile.readlines()
        lines = [x.strip() for x in lines]

    for word in lines:
        stopwords.add(word)
    return

def openFile( fnR, fnW ):
    f = open(fnW, "wb")
    non_bmp_map = dict.fromkeys(range(0x10000, sys.maxunicode + 1), 0xfffd)

    with codecs.open(fnR, 'r', encoding='utf-8') as csvfile:
        next(csvfile) # skip header
        readCSV = csv.reader(csvfile, delimiter=',')

        for row in readCSV:
            vid = row[0] # video id
            if vid not in seen:
                title = row[2].translate(non_bmp_map)
                view = int(row[7]) #.translate(non_bmp_map)
                likes = int(row[8])
                comments = int(row[10])
                # get ratio of comments-to-views, convert to %, and round to 2 d
                commentsToViews = round((comments/view)*(100), 2)
                ctv.append(commentsToViews)
                 # get ratio of likes-to-views
                likesToViews = round((likes/view)*100, 2)
                ltv.append(likesToViews)

                title_view_entry = [title, view] # entry for a list of (titles,
                titlesViews.append(title_view_entry)

                title_ctv_entry = [title, commentsToViews] # entry for a list of
                titlesCVratio.append(title_ctv_entry)

                title_ltv_entry = [title, likesToViews] # entry for a list of (t
                titlesLVratio.append(title_ltv_entry)
```

```
def bin( type, average, list1, list2 ):
    list = []
    if type == "views":
        list = titlesViews
    if type == "ctv":
        list = titlesCVratio
    if type == "ltv":
        list = titlesLVratio

    for each in list:
        num = int(each[1])
        title = each[0]
        pair = [title, num]
        if num < average:
            list1.append(pair)
        else:
            list2.append(pair)

    return

def countWords( dirtyList ):
    words = []
    table = str.maketrans('', '', string.punctuation)
    for each in dirtyList:
        words.extend(each[0].split()) # splits each line into words and then
                                      # add each element of the result to words *NOT .append
        stripped = [w.translate(table) for w in words]

    stripped = list(filter(None, stripped))
    for word in stripped:
        if word in stopwords:
            stripped.remove(word)

    return stripped
```

3. *Filter* – Although Python filtered some of the data, I realized that it was not efficient at filtering out large amounts of data. rStudio has libraries specifically for text processing and mining, so I further filtered the data to remove numbers, punctuations, extra white spaces, and additional stop words. Furthermore, to preserve case sensitivity in my word clouds, I filtered out the title case and uppercase versions of stop words.

```
text <- readLines(file.choose())
docs <- Corpus(VectorSource(text))
toSpace <- content_transformer(function (x , pattern ) gsub(pattern, " ", x))
docs <- tm_map(docs, toSpace, "/")
docs <- tm_map(docs, toSpace, "@")
docs <- tm_map(docs, toSpace, "\\|")
docs <- tm_map(docs, toSpace, "-")
docs <- tm_map(docs, removeNumbers)
docs <- tm_map(docs, removeWords, stopwords("english"))
docs <- tm_map(docs, removeWords, stopwordsuc)
docs <- tm_map(docs, removeWords, stopwordstc)
docs <- tm_map(docs, removePunctuation)
docs <- tm_map(docs, stripWhitespace)
df <- data.frame( text = get("content", docs))
>write.table(df, file = "top25Likes-tidy.txt")
```

a. Using a variety of online word processing tools, I further cleaned the exported text files to remove empty lines and spaces as well as unnecessary quotation marks. The end product was a cleaned text file, with a list of words, one per line.

4. *Represent* – I used amueller's Python word cloud generator to create my word clouds. Amueller's algorithm uses the size of the text (not color saturation) to represent word frequency, and is more customizable than rStudio's so that you can apply masks and customized colors to the word clouds.

```
from os import path
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt

from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator

d = path.dirname(__file__)
font_path = "C:/Users/hyukw/Desktop/SDS136Final/Fonts/Oswald-Regular.ttf"

# Read the whole text.
below_text = open(path.join(d, './text/bottom25Comments-tidy.txt'), encoding='ut
above_text = open(path.join(d, './text/top25Comments-tidy.txt'), encoding='utf-8
# read the mask image
# taken from
# http://www.stencilry.org/stencils/movies/alice%20in%20wonderland/255fk.jpg
below_coloring = np.array(Image.open(path.join(d, "logo-blue.png")))
above_coloring = np.array(Image.open(path.join(d, "logo.png")))

stopwords = set(STOPWORDS)
stopwords.update(['Official', 'OFFICIAL', 'official', 'VIDEO', 'video', 'Trump',


below_wc = WordCloud(font_path=font_path, background_color="white", max_words=20
               stopwords=stopwords)
above_wc = WordCloud(font_path=font_path, background_color="white", max_words=20
               stopwords=stopwords)

# generate word clouds
below_wc.generate(below_text)
bimage_colors = ImageColorGenerator(below_coloring)

above_wc.generate(above_text)
aimage_colors = ImageColorGenerator(above_coloring)

# store to file
below_wc.recolor(color_func=bimage_colors)
below_wc.to_file(path.join(d, "./wc/bottom25Comments.png"))
```

```
above_wc.recolor(color_func=aimage_colors)
above_wc.to_file(path.join(d, "./wc/top25Comments.png"))

# show
plt.imshow(wc, interpolation='bilinear')
plt.axis("off")
plt.figure()
plt.imshow(wc.recolor(color_func=bimage_colors), interpolation='bilinear')
plt.axis("off")
plt.figure()
plt.imshow(alice_coloring, cmap=plt.cm.gray, interpolation="bilinear")
plt.axis("off")
plt.show()
```

a. To showcase the word clouds, I used simple CSS/HTML styling to create a website.

5. *Refine* – During this step, I considered what types of visual attributes I could use to help observers easily distinguish between the least and most popular videos. Color seemed like the most obvious choice, but I struggled on deciding whether I wanted a change in hue or saturation to define such differences. As discussed in class, we often use different hues to represent categorical variables and different saturations/lightness for ordinal variables. In this case, I thought popularity could be seen as both a categorical and ordinal variable. I

finally decided to change the hues because changing the saturation/lightness made the word clouds look too similar and difficult for comparison purposes.

a. I also revisited the filtering step during this step because I realized that some words appeared so frequently in both clouds that they were not meaningful (e.g. "official"). In the second set of word clouds under "What is Popular?", I show what the word clouds look like after filtering out these non-meaningful words.

titles with < average # of views

titles with > titles average # of views

titles with < average # of views (filtered)

titles with > average # of views (filtered)

b. In “What Are People Talking About?” and “What Do People Like?”, I first experimented with strictly using the ratio of comments to views and likes to views to determine level of popularity. The word clouds that were generated were far more interesting (e.g. greater variety of words) compared to the previously generated word clouds. It turns out that the cause of such dramatic difference was because I was basing the word clouds strictly on their ratios. For example, videos with fewer views could have very high comments-to-views ratios, therefore placing them in a higher rank, while videos with higher views could have very low comments-to-views ratios, resulting in lower ranks. I soon realized that if I wanted to take into consideration both a video’s number of views as well as its number of comments (or likes), I had to use a scalar. In the second set of word clouds under “What Are People Talking About?” and “What Do People Like?”, I used a 50-50 scalar, where both number of likes and number of comments (or likes) were weighted equally. This required me to revisit the mining step.

titles with < average comments to views ratio

titles with > average comments to views ratio

titles with < average # of comments + views

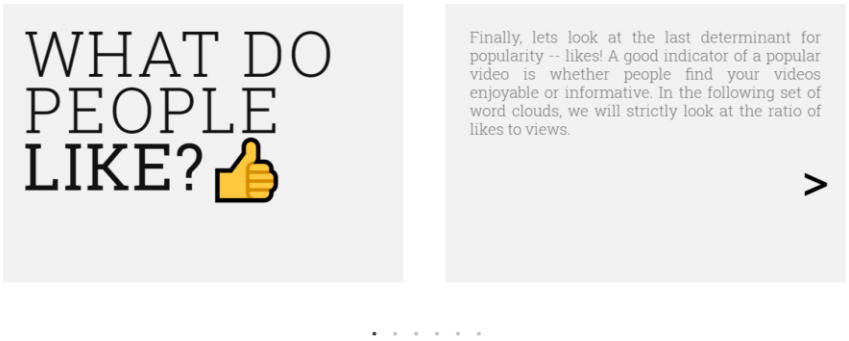
titles with > average # of comments + views

c. Using the average as the threshold for the upper and lower bounds did not seem to produce very interesting results, so I tried tightening the bounds by discarding all the video titles in the 2nd quartile. This required me to revisit the mining step. In the final set of word clouds under "What is Popular", "What Are People Talking About?", and "What Do People Like?", I show only the words in the bottom 25th and top 25th percentile. The result was that all three sets of word clouds looked very similar, if not nearly identical.

words in the bottom 25th percentile of # of views (filtered)

words in the top 25th percentile of # of views (filtered)

6. *Interact* – The main goal of adding an interactive element was to promote cross-comparison across different sets of word clouds. Rather than creating a static website that showed the literal progression of my iterative process (e.g. infinite scrolling), I decided to use a jQuery carousel script (slick) to implement horizontal sliders. This way, readers can scroll to different sets of word clouds and compare across different metrics for popularity.



I encountered many problems in the process of creating my visualizations. For example, I am unsure why some words (e.g. “official”) appear multiple times in the same word cloud, and amueller’s website does not explain what algorithm he used. In addition, it was challenging defining an appropriate threshold between levels of popularity since all the videos in the dataset would be considered “popular.” Thus, popularity could only be measured relative to the other videos in the data set.

Another problem I encountered was that the data set represents a very small fraction of the total number of videos on YouTube during the period of November and December. After parsing out duplicate titles in the dataset, about 2000 entries remained, meaning that the sample size was not large enough and the term frequencies may not have been meaningful.

These problems reminded me of Kosara’s talk in which he argues that sometimes it is more important to think about the big picture of a visualization than all the small details behind its construction. Increasing accessibility to data and engaging the audience is sometimes more important than the finer details. Yet, in the process of refining my visualizations, I continually wondered to myself: how can I strike a healthy balance between accuracy, readability, and visualization’s “stickiness” value (how memorable it is)? After all, when you start sacrificing accuracy for memorability or readability, then at some point the data visualization only holds entertainment value and becomes untrue. When I first started this project, I did not think the process of creating data visualizations would be so challenging and fraught with critical decisions. It is as Nathan Yau says:

> *It’s easy to think that this process is instant because software enables you to plug data in, and you get something back instantly, but there are steps and choices in between. What shape should you choose to encode your data? What color is most appropriate for the purpose and message? […] In many ways, visualization is like cooking. You are the chef, and datasets, geometry, and color are your ingredients. A skilled chef…is likely to prepare a delicious meal. A less skilled*

> *cook, who heads to the local freezer section to see what microwave dinners look good, might nuke a less savory meal (Yau, 92).*

Perhaps it is because I am a "unskilled cook" that I was unable to produce the statistically interesting results that I wanted. More importantly, this experience made me realize first-hand that every data visualization contains bias because they reflect judgments about what the computer or the programmer considers the most important features. Perhaps the only difference between the skilled and unskilled chef is not that the skilled chef lacks bias, but rather that he is better at concealing his bias.

Works Cited

Cairo, Alberto. *The Functional Art: An Introduction to Information Graphics and Visualization.* New Riders, 2013.

Fry, Benjamin Jotham. *Computational Information Design*. 2004.

Yau, Nathan. *Data Points: Visualization That Means Something*. John Wiley & Sons, Inc., 2013.