Final Project ChE 395

by **Spencer Hong**

"Trump speaks at level of 8-year-old, new analysis finds."
Is this true? Can we prove it?

Let's play around with the data

In [1]:
```
import pandas as pd
```

In [2]:
```
tweets = pd.read_csv('data/trumptweets-1515775693.tweets.csv', low_memory = False)
tweets.head()
```

Out[2]:

	status_id	created_at	user_id	screen_name	text	source	display_text_width	reply_to_status_id	reply_
0	x1864367186	2009-05-20 22:29:47	x25073877	realDonaldTrump	Read a great interview with Donald Trump that ...	Twitter Web Client	112	NaN	
1	x9273573134835712	2010-11-29 15:52:46	x25073877	realDonaldTrump	Congratulations to Evan Lysacek for being nomi...	Twitter Web Client	127	NaN	
2	x29014512646	2010-10-28 18:53:40	x25073877	realDonaldTrump	I was on The View this morning. We talked abou...	Twitter Web Client	139	NaN	
3	x7483813542232064	2010-11-24 17:20:54	x25073877	realDonaldTrump	Tomorrow night's episode of The Apprentice del...	Twitter Web Client	140	NaN	
4	x5775731054	2009-11-16 21:06:10	x25073877	realDonaldTrump	Donald Trump Partners with TV1 on New Reality ...	Twitter Web Client	116	NaN	

5 rows × 68 columns

What is the range of dates on this dataset?

In [3]:
```
tweets[['created_at']].sort_values('created_at').iloc[[0, -1]]
```

Out[3]:

	created_at
18	2009-05-04 18:54:25
32644	2018-01-12 13:48:49

Our analysis will pertain to Donald Trump tweets from 2009 to 2018.

How often does Donald Trump retweet vs. spit out a tweet of his own?

In [4]:
```
len(tweets[tweets.is_retweet])/len(tweets) * 100
```

Out[4]:
```
1.5627855967830377
```

He only retweets 1.6% of the time. This means we have a good enough dataset to analyze his own writing. Let's omit retweets from the dataset. We're also only worried about the text, so let's just keep the text and the date, and drop everything else.

In [3]:
```
tweets = tweets[~tweets.is_retweet][['created_at','text']]
tweets.head()
```

Out[3]:

	created_at	text
0	2009-05-20 22:29:47	Read a great interview with Donald Trump that ...
1	2010-11-29 15:52:46	Congratulations to Evan Lysacek for being nomi...
2	2010-10-28 18:53:40	I was on The View this morning. We talked abou...
3	2010-11-24 17:20:54	Tomorrow night's episode of The Apprentice del...
4	2009-11-16 21:06:10	Donald Trump Partners with TV1 on New Reality ...

If you look closely, even if we take out the retweets, there are some weird tweets (looks like not a first-person type of writing). See below for examples.

In [6]:
```
tweets.iloc[0].text
```

Out[6]:
```
'Read a great interview with Donald Trump that appeared in The New York Times Magazine: http://tinyurl.com/qsx4
o6'
```

In [7]:
```
tweets.iloc[4].text
```

Out[7]:
```
"Donald Trump Partners with TV1 on New Reality Series Entitled, Omarosa's Ultimate Merger: http://tinyurl.com/y
k5m3lc"
```

after some research, it is clear that even the non-retweets may not come from himself -- and to get to the heart of the question, we want to only get the tweets that represent Donald Trump as himself, no one else. Let's also avoid all the media links. Trump also sometimes retweets other accounts using "RT @account" method. Let's avoid this as well.

In [4]:
```
tweets = tweets[~(tweets.text.str.contains('http') | tweets.text.str.contains('RT @'))]
print(f'we have {tweets.shape[0]} tweets after preprocessing')
tweets.head(2)
```

```
we have 24669 tweets after preprocessing
```

Out[4]:

	created_at	text
1	2010-11-29 15:52:46	Congratulations to Evan Lysacek for being nomi...
2	2010-10-28 18:53:40	I was on The View this morning. We talked abou...

Okay, now we're ready.

Question 1

What distribution does Donald Trump's word frequency follow? For now, just plot, and find a distribution that might fit this. No parameters are needed yet, just a visual approximation.

Answer 1

We first have to tokenize the text. This means to split the text into individual word(s). We are going to use unigram-approach, which means that each word is one word. Let's also get rid of punctuations.

In [5]:
```
# importing an useful tokenizer
from nltk import word_tokenize
# this is regex!
import re
```

In [6]:
```
# we split tweets into individual words while also getting rid of anything at is not a letter or space.
tweets['words'] = tweets['text'].apply(lambda x: word_tokenize(re.sub(r'[^\w\s]', '', x)))
tweets[['words']].head(1).words.values
```

Out[6]:
```
array([list(['Congratulations', 'to', 'Evan', 'Lysacek', 'for', 'being', 'nominated', 'SI', 'sportsman', 'of',
'the', 'year', 'Hes', 'a', 'great', 'guy', 'and', 'he', 'has', 'my', 'vote', 'EvanForSI'])],
      dtype=object)
```

Now, we have to go through every tweet and start counting. Python provides an excellent built-in collection tool that allows this.

In [7]:
```
from collections import Counter
```

In [64]:
```
all_words = [word for templist in tweets[['words']].values.tolist() for words in templist for word in words  ]
len(all_words)
```

Out[64]:
```
421566
```

there are more than 400,000 words (not unique) in his tweets from 2009 to 2017.

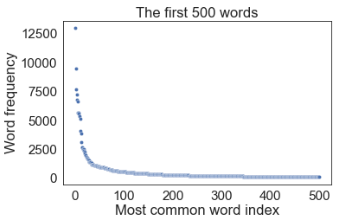
In [65]:
```
word_frequency = Counter(all_words)
sorted(word_frequency, key=word_frequency.get, reverse=True)[:10]
```

Out[65]:
```
['the', 'to', 'realDonaldTrump', 'a', 'is', 'and', 'I', 'of', 'you', 'in']
```

It makes sense that the top 10 words include "the", "to", his own Twitter name, and "a".

In [8]:
```
# plotting tools
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
sns.set_style("white")
```

In [68]:
```
word_frequency = Counter(all_words)
word_frequency = word_frequency.most_common()
frequency_values = [x[1] for x in word_frequency]

fig, axes = plt.subplots(1, 1, figsize=(8, 5), dpi = 50)
sns.set(font_scale = 2)
sns.scatterplot(x = np.arange(0, len(word_frequency))[:500], y = frequency_values[:500], ax= axes)

plt.title('The first 500 words')
axes.set_xlabel('Most common word index')
axes.set_ylabel('Word frequency')
plt.show()
```



This fits the Zipf's law.

$$f(k; s, N) = \frac{\frac{1}{k^s}}{\sum_1^N \frac{1}{n^s}}$$

where k is the rank, N is the number of elements, and s is the parameter that characterizes the distribution.

Question 2

Obviously, we're going to need a more rigorous way to find the distribution. First, find the maximum likelihood estimator for the parameter of the Zipf distribution. Zipf distribution only requires 1 parameter, s, the shape parameter.

Answer 2

The probability mass function using Zipf distribution (from equation 1) can be used to calculate the log-likelihood $L(s)$.

$L(s) = Log(\frac{1}{k^s * HarmonicNumber(N,s)})$

Because of the Harmonic Numbers, it will be extremely difficult to find an analytical solution for the estimator. Therefore, we use optimization methods to find an estimate.

In [66]:
```
from scipy.optimize import minimize_scalar
from scipy.stats import zipf, norm
def neg_log_likelihood(s):
    N = sum(frequency_values)
    probs = np.log(frequency_values**(-1 * s) / np.sum(np.arange(1,  N+1)**(-1 * s)))
    summed = sum(probs * frequency_values)
    return -1 * summed
```

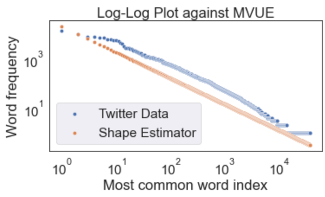
In [69]:
```
frequency_values = np.array(frequency_values)
s_best = minimize_scalar(neg_log_likelihood, [0, 6.0] )
print(s_best.x)
```

```
1.0472055615708566
```

The best estimate for the s parameter was 1.0472.

In [18]:
```
sns.set_style('white')
fig, axes = plt.subplots(1, 1, figsize=(8, 5), dpi = 50)
sns.set(font_scale = 2)
plt.yscale('log')
```

```
plt.xscale('log')
sns.scatterplot(x = np.arange(1, len(word_frequency)+1), y =frequency_values, ax= axes, label = 'Twitter Data')
plt.title('Log-Log Plot against MVUE')
axes.set_xlabel('Most common word index')
axes.set_ylabel('Word frequency')

# here, we are fitting the estimate we found using Scipy's built-in zipf PMF function.
fitted = np.array([zipf.pmf(n, s_best.x) for n in np.arange(1, len(frequency_values) + 1)]) * sum(frequency_val

pl = sns.scatterplot(x = np.arange(1, len(word_frequency) + 1), y = fitted, ax = axes, label = 'Shape Estimator
pl.legend(frameon = True)
plt.tight_layout()
plt.show()
```



The estimate we found is a pretty good estimate to the 400,000 tweet data.

Question 3

Using the MLE from question 2, bootstrap 10000 times and calculate this estimator 1000 times. What is the mean? the variance? Does this match with analytical solution?

Answer 3

Python's Scikit-Learn has a great bootstrap API that we will use. (We also could have used Pandas resample feature, which actually would have been easier since our data is already in dataframes, but oh well. too lazy to change it.)

In [10]:

```
from sklearn.utils import resample
```

In [11]:

```
list_to_sample_from = [x[0] for x in tweets[['words']].values]
```

In [44]:

```
len(resample(list_to_sample_from, replace=True, n_samples=100, random_state=42))
```

Out[44]:

```
100
```

The following step will take a while since we are optimizing to find estimator for s 10000 times. We'll use a time-tracking library to keep track of the progress.

In [55]:

```
from tqdm.notebook import tqdm
tqdm()
```

Out[55]:

```
0it [00:00, ?it/s]
```

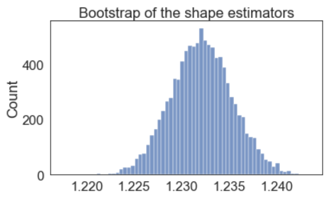
In [61]:

```
global word_frequency_temp, frequency_values_temp, temp_words
def neg_log_likelihood_temp(s):
    N = sum(frequency_values_temp)
    probs = np.log(frequency_values_temp**(-1 * s) / np.sum(np.arange(1,  N+1)**(-1 * s)))
    summed = sum(probs * frequency_values_temp)
    return -1 * summed
bootsamples = []
for i in tqdm(range(0, 10000)):
    sample = resample(list_to_sample_from, replace=True, n_samples=1000)

    temp_words = [word for words in sample for word in words]
    word_frequency_temp = Counter(temp_words)
    word_frequency_temp = word_frequency_temp.most_common()
    frequency_values_temp = [x[1] for x in word_frequency_temp]
    frequency_values_temp = np.array(frequency_values_temp)

    s_best = minimize_scalar(neg_log_likelihood_temp, [0, 6.0] )

    del word_frequency_temp, frequency_values_temp, temp_words
    bootsamples.append(s_best.x)
```

Let's plot our estimators.

In [62]:

```
sns.set_style('white')
fig, axes = plt.subplots(1, 1, figsize=(8, 5), dpi = 50)
sns.set(font_scale = 2)
sns.histplot(bootsamples, ax = axes)

plt.title('Bootstrap of the shape estimators')

plt.tight_layout()
plt.show()
```



In [63]:

```
print(f'The mean of the estimator from the bootstrap population is: {np.round(np.mean(bootsamples), 3)}')
print(f'The variance of the estimator from the bootstrap population is: {np.round(np.var(bootsamples), 5)}')
```

```
The mean of the estimator from the bootstrap population is: 1.232
The variance of the estimator from the bootstrap population is: 1e-05
```

Notice that when we only look at 1000 tweets at a time, it does not come close to knowing the full picture of 400,000 tweets.

Question 4

Using our hypothesis testing knowledge from class, determine if Donald Trump's tweets follow the distribution with the shape estimator from the bootstrap in question 3. You will need to set up a null/alternative hypotheses and a log-likelihood ratio statistic. Plot the p-values from the 1000 bootstraps 100 times.

Answer 4

The null hypothesis is that the s parameter from the bootstrapped distribution fits the overall, underlying distribution of 400,000 tweets. The alternative hypothesis is that the s parameter does not fit the underlying distribution.

$H_o : s = s_{bootstrap}$
$H_a : s \neq s_{bootstrap}$

$D = -2 * log\frac{L(\theta_o)}{L(\theta_a)}$
where $L(\theta_o)$ is the likelihood if the null hypothesis is true (bootstrap s estimator) and
$L(\theta_a)$ is the maximum lieklihood for the alternative hypothesis.

$L(\theta_o) = \Pi_i^n f(s_{bootstrap}, i)$
where f is the Zipf probability mass function.

$L(\theta_a) = \Pi_i^n f(s_{data}, i)$
where s_{data} represents the fitted s parameter for the sample, and not the bootstrapped parameter.

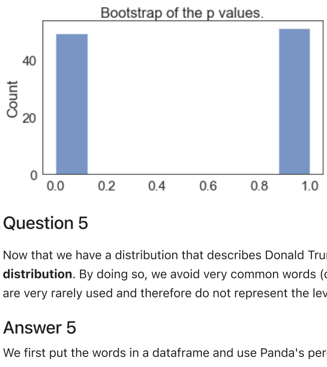
$D = \sum_i^n -2log(\frac{f(s_{bootstrap}, i)}{f(s_{data}, i)}) \approx \chi^2(n)$
since the null hypothesis has 1 parameter and the alternative hypothesis has 1 parameter (1-1 = 0).

In [71]:

```
from scipy.stats import chi2

bootstrap_s =  1.232
test_statistics = []
p_values = []
global word_frequency_temp, frequency_values_temp, temp_words
def neg_log_likelihood_temp(s):
    N = sum(frequency_values_temp)
    probs = np.log(frequency_values_temp**(-1 * s) / np.sum(np.arange(1,  N+1)**(-1 * s)))
    summed = sum(probs * frequency_values_temp)
    return -1 * summed

for i in tqdm(range(0, 100)):
    sample = resample(list_to_sample_from, replace=True, n_samples=1000)

    temp_words = [word for words in sample for word in words]
    word_frequency_temp = Counter(temp_words)
    word_frequency_temp = word_frequency_temp.most_common()
    frequency_values_temp = [x[1] for x in word_frequency_temp]
    frequency_values_temp = np.array(frequency_values_temp)

    s_best = minimize_scalar(neg_log_likelihood_temp, [0, 6.0] )

    log0 = np.log(np.array([zipf.pmf(n, bootstrap_s) for n in np.arange(1, len(frequency_values_temp) + 1)]))
    log1 = np.log(np.array([zipf.pmf(n, s_best.x) for n in np.arange(1, len(frequency_values_temp) + 1)]))
    D = np.sum(-2*(log0 - log1))

    test_statistics.append(D)

    p_values.append(1 - chi2.cdf(D, 1))
    del word_frequency_temp, frequency_values_temp, temp_words
```

In [74]:

```
sns.set_style('white')
fig, axes = plt.subplots(1, 1, figsize=(8, 5), dpi = 50)
sns.set(font_scale = 2)
sns.histplot(p_values, ax = axes)

plt.title('Bootstrap of the p values.')

plt.tight_layout()
plt.show()
```



Question 5

Now that we have a distribution that describes Donald Trump's "speaking" (writing) quality, **get the top 20% of words in this distribution**. By doing so, we avoid very common words (called stop words) such as "the", "for", "and", but we also avoid words that are very rarely used and therefore do not represent the level of speaker well. Give us 10 words that fall in this section.

Answer 5

We first put the words in a dataframe and use Panda's percentile function to get the top 20%.

In [11]:

```
all_words = [word for templist in tweets[['words']].values.tolist() for words in templist for word in words  ]
word_frequency = Counter(all_words)
word_frequency = word_frequency.most_common()
frequency_values = [x[1] for x in word_frequency]
word_frequency = [x[0] for x in word_frequency]
```

In [12]:

```
percentile = pd.DataFrame({'words':word_frequency, 'frequency':frequency_values})
percentile.head()
```

Out[12]:

	words	frequency
0	the	12968
1	to	9506
2	realDonaldTrump	7663
3	a	7257
4	is	6753

In [87]:

```
percentile.quantile([1, 0.8])
```

Out[87]:

	frequency
1.0	12968.0
0.8	3.0

The 80th percentile for frequency values is 3. We'll just grab the data points with frequency higher than or equal to 3.

In [95]:

```
percentile = percentile.loc[percentile.frequency >= 3]
percentile.head(10)
```

Out[95]:

	words	frequency
0	the	12968
1	to	9506
2	realDonaldTrump	7663
3	a	7257
4	is	6753
5	and	6611
6	I	5678
7	of	5648
8	you	5597
9	in	5378

Question 6

In natural langauge processing, a method to test the reading level of a writing is called the **Flesch Kincaid Grade Level Test**. Let's apply this test to the middle 50% of words that we've identified. Give a histogram of the results.

Answer 6

We use a Python library called py-readability-metrics (https://github.com/cdimascio/py-readability-metrics).

In [13]:

```
from readability import `Readability
```

because one tweet is too short to be tested, we combine 10 tweets together (considering one tweet to be around 10 words).

In [23]:

```
readinglevel = (tweets.groupby(tweets.index // 10).agg(' '.join))
readinglevel.head(2)
```

Out[23]:

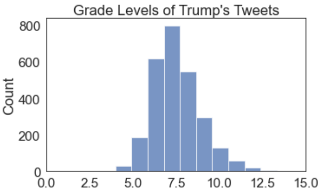
	created_at	text
0	2010-11-29 15:52:46 2010-10-28 18:53:40 2010-1...	Congratulations to Evan Lysacek for being nomi...
1	2010-12-10 14:42:15 2010-12-16 14:30:23 2010-1...	Congratulations to Brandy as our new Apprentic...

In [29]:

```
grade_levels = []
grade_scores = []
for index, row in readinglevel.iterrows():
    # we put a try statement in case there are still tweets than are <100 words.
    try:
        r = Readability(row['text'])
        fk = r.flesch_kincaid()
        grade_scores.append(fk.score)
        grade_levels.append(fk.grade_level)
    except:
        pass
grade_levels = [ int(x) for x in grade_levels ]
```

In [49]:

```
sns.set_style('white')
fig, axes = plt.subplots(1, 1, figsize=(8, 5), dpi = 50)
sns.histplot(sorted(grade_levels, reverse = False), ax = axes, bins = 30)
plt.title('Grade Levels of Trump\'s Tweets')
sns.set(font_scale = 2)
axes.set_xlim([0, 15])
plt.tight_layout()
plt.show()
```



In [51]:

```
np.mean(grade_levels)
```

Out[51]:

```
7.338757396449704
```

Donald Trump's average reading level is 7th grade, which is around 12 years old.

Question 7

Calculate and present a 95% confidence interval of the grade results per tweet.

Answer 7

We make use of bootstrapping method again to get the sample mean and use a normal distribution citing the Central Limit Theorem.

In [52]:

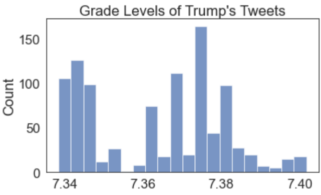
```
def bootstrap():
    temp = tweets.sample(1000)
    readinglevel = (temp.groupby(temp.index // 10).agg(' '.join))
    for index, row in readinglevel.iterrows():
        # we put a try statement in case there are still tweets than are <100 words.
        try:
            r = Readability(row['text'])
            fk = r.flesch_kincaid()
            grade_scores.append(int(fk.score))
            grade_levels.append(int(fk.grade_level))
        except:
            pass

    return np.mean(grade_levels)
```

In [56]:

```
averaged_grades = []
for i in tqdm(range(0, 1000)):
    averaged_grades.append(bootstrap())
```

In [60]:

```
sns.set_style('white')
fig, axes = plt.subplots(1, 1, figsize=(8, 5), dpi = 50)
sns.histplot(sorted(averaged_grades, reverse = False), ax = axes, bins = 20)
plt.title('Grade Levels of Trump\'s Tweets')
sns.set(font_scale = 2)
plt.tight_layout()
plt.show()
```



In [61]:

```
mean_grade = np.mean(averaged_grades)
std_grade = np.std(averaged_grades)
```

The confidence interval is $\mu - 1.96 * \frac{\sigma}{\sqrt{n}} < \mu < \mu + 1.96 * \frac{\sigma}{\sqrt{n}}$

In [62]:

```
lower_bound = mean_grade - 1.96 * std_grade/np.sqrt(1000)
upper_bound = mean_grade + 1.96 * std_grade/np.sqrt(1000)

print(lower_bound)
print(upper_bound)
```

```
7.362628739609287
7.364733810344065
```

Donald Trump speaks at a 7th grade level.