LING001_Final_Project

December 20, 2018

1 LING001 Final Project

1.1 Introduction

The Vector Space Model for information retrieval is an interesting linguistic analysis technique. The paramount issue with language processing is representing words mathematically to do relevant computations. So, the Vector Space Model was developed to allow words to be represented as vectors! Using the model, we can find the similarity between a document and a query, which will tell us the relative importance of the query to the document.

I think a good corpus to use is the Harry Potter books since they are a good length and I find them interesting.

Dataset: - SORCERER STONE - CHAMBER OF SECRETS - PRISONER OF AZKABAN
- GOBLET OF FIRE - ORDER OF THE PHOENIX
- HALF-BLOOD PRINCE - DEATHLY HALLOWS (PART 1 AND 2)

1.2 Question/Proposal

How does the cosine similarity of each of the main character's names in all of the Harry Potter books relate to the amount of screen time they have in the Harry Potter movies?

1.3 Hypothesis

If the name of the character has a higher cosine similarity, then the weight of the name in the corpus is higher, meaning the importance of the person to the plot is greater, thus, they should have more screen time in the movie.

1.4 Zipf's Law

Before we begin using the model, let's explore why the model may make sense to use in the first place. What makes it work so well?

Zipf's law states that the rank of an entity - such as word - in a set will be inversely proportional to the number of times the entity shows up. For instance, in English, the most popular word - "the" - accounts for roughly 6% of all the words we say. By Zipf's law, the second most common word - "of" - will occur half as often as "the", which it mysteriously does, and so on for all subsequent words. This property seems to permeate all of nature and is even built into how we think. However, even though there have been many research papers studying Zipf's Law, no conclusive evidence has been discovered.

```
In [17]: HTML("""
         <div align="middle">
         <img width="75%" src="zipf.png">
         </img></div>""")
         #Soure = https://www.ahsthenest.com/archive/2018/03/19/zipfs-law/
```

```
Out[17]: <IPython.core.display.HTML object>
```

$$number\ of\ occurences = \frac{1}{rank}$$

Moreover, this means that a majority of the words we write and say do not have much meaning. They are only there to space out and connect the meaningful/important concepts. This is why the Vector Space Model works so well. Even if a word appears hundreds of thousands of times in a corpus, it's importance will be diminished. In the Harry Potter books, the word "the" occurs all the time, but it gives no information about the plot of the books. The Vector Space Model takes advantage of the Zipf-ian nature of language to extract important terms from a corpus. Below, we use the Vector Space Model to do exactly that. In addition, we show that the Harry Potter books do in fact follow a Pareto distribution, which mathematicallly represent Zipf's law.

```
In [12]: #Frequency of terms in Sorcer of Stone
         import re
         from operator import itemgetter
         from plotly import tools

         datas=[]
         layouts=[]
         for i in range(7):
             file = open(f"B{i+1}.txt", 'rt')
             text = file.read()
             text=nltk.word_tokenize(text)
             frequency={}
             for word in text:
                 if word.isalpha():
                     count = frequency.get(word,0)
                     frequency[word] = count + 1

             frequency = sorted(frequency.items(), key=lambda kv: kv[1], reverse=True)

             words = [item[0] for item in frequency]
             times = [item[1] for item in frequency]
             data = go.Bar(
                         x=words[0:100],
                         y=times[0:100]
                 )

             datas.append(data)
             #layouts.append(layouts)
         fig = tools.make_subplots(rows=4, cols=2,
                                       specs=[[{}, {}],[{}, {}],[{}, {}], [{'colspan': 2}, None]],
                                   subplot_titles=(books[0], books[1], books[2],
                                   books[3],books[4],books[5],books[6]))
         fig.append_trace(datas[0],1,1)
         fig.append_trace(datas[1],1,2)
         fig.append_trace(datas[2],2,1)
         fig.append_trace(datas[3],2,2)
         fig.append_trace(datas[4],3,1)
         fig.append_trace(datas[5],3,2)
         fig.append_trace(datas[6],4,1)

         for i in range(len(books)):
             fig['layout']['xaxis'+str(i+1)].update(title='Word')

         for j in range(len(books)):
             fig['layout']['yaxis'+str(j+1)].update(title='Frequency')

         fig['layout'].update(showlegend= False)

         fig['layout'].update(height=1000, width=1000,
                              title="Zip's law in all Harry Potter books")
         py.iplot(fig, filename='basic-bar')
```

```
This is the format of your plot grid:
[ (1,1) x1,y1 ]  [ (1,2) x2,y2 ]
[ (2,1) x3,y3 ]  [ (2,2) x4,y4 ]
[ (3,1) x5,y5 ]  [ (3,2) x6,y6 ]
[ (4,1) x7,y7           -      ]
```

```
Out[12]: <plotly.tools.PlotlyDisplay object>
```

1.5 Building Blocks

First, let's explore the building blocks that will allow us to determine the importance of a word/phrase to a corpus using the Vector Space Model.

1.5.1 Term Frequency

Term frequency (TF) is the number of times a word appears in a given corpus.
For example, the term frequency of "dope" in "That was dope." is 1.

1.5.2 Inverse Document Frequency

Inverse document frequency (IDF) is a measure of the rareness of a word in a corpus. It is defined as:

2 $idf_t = log(\frac{N}{t_f})$

where N is the number of documents and t_f is the number of documents in the corpus which the term is present in.

The log function looks like:

```
In [2]: from IPython.display import HTML
        HTML("""
        <div align="middle">
        <video width="100%" autoplay loop>
              <source src="VSM.mp4" type="video/mp4">
        </video></div>""")
        #See attached video VSM.mp4 if unable to view

Out[2]: <IPython.core.display.HTML object>
```

From the above graph, we can see that when the term isn't in that many documents, the input to the log function will be high, since N is constant, making the output higher.

2.0.1 TF-IDF

TF-IDF is the term freqency multiplied by the inverse document frequency. This is a measure of the uniqueness of a word in a document that exists in a corpus.

$tf - idf = tf * idf$

2.0.2 Cosine Similarity

The tf-idf values are then put into a vector that represents the word or document in a vector space. The cosine similarity is defined as:

$CosineSimilarity = \frac{a \cdot b}{||a|| ||b||}$ where a and b are two document or word/phrase vectors

```
In [3]: from IPython.display import HTML
        HTML("""
        <div align="middle">
        <video width="100%" autoplay loop>
              <source src="CS.mp4" type="video/mp4">
        </video></div>""")
        #See attached video CS.mp4 if unable to view

Out[3]: <IPython.core.display.HTML object>
```

The higher the cosine similarity, the greater the dot product, since they are more in the same direction i.e. the projection of one vector onto the other is large. Each vector is divided by its magnitude to normalize the distance since we only care about the angle between them, so a unit vector is created in the direction of each vector. The cosine similarity is equal to $cos(\theta)$, shown in the example above.

2.1 Data Exploration

Text files for books was obtained from http://www.glozman.com/textpages.html.

```
In [4]: from sklearn.feature_extraction.text import TfidfVectorizer
        from sklearn.metrics.pairwise import cosine_similarity
        import nltk
        import pandas as pd

        file = open("B1.txt", 'rt')
        text = file.read()
        text=nltk.word_tokenize(text)
        print(text[0:10])
```

```
['Harry', 'Potter', 'and', 'the', 'Sorcerer', "'s", 'Stone', 'CHAPTER', 'ONE', 'THE']
```

The data looks like it is being tokenzied properly. Now, we should find the tf-idf vectors, find the cosine similarity between each of the queries and books, get the screen times of each character in each movie, and see if there is a correlation between the two.

2.2 Data Analysis

```
In [5]: corpus=[]
        books = [
            "SORCERER STONE",
            "CHAMBER OF SECRETS",
            "PRISONER OF AZKABAN",
            "GOBLET OF FIRE",
            "ORDER OF THE PHOENIX",
            "HALF-BLOOD PRINCE",
            "DEATHLY HALLOWS (PART 1 AND 2)"
        ]
        num_books = 7
        for i in range(num_books):
                file = open(f"B{i+1}.txt", 'rt')
                text = file.read()
                corpus.append(text)

        characters=[
            "HARRY POTTER",
            "RON WEASLEY",
            "HERMIONE GRANGER",
            "VOLDEMORT",
            "DUMBLEDORE",
            "HAGRID",
            "DRACO MALFOY",
            "SIRIUS BLACK",
            "NEVILLE LONGBOTTOM"
        ]

        cosine_similarities={}
        for character in characters:
            corpus.append(character)
            vectorizer = TfidfVectorizer()
            tfidf_matrix = vectorizer.fit_transform(corpus)
            character_sim=[]
            for i in range(num_books):
                similarity = float(cosine_similarity(tfidf_matrix[i],tfidf_matrix[-1]))*1000
                sim_rounded = round(similarity,3)
                character_sim.append(sim_rounded)
            cosine_similarities[character]=character_sim

        pd.DataFrame(cosine_similarities, index=books)
```

```
Out[5]:                             HARRY POTTER  RON WEASLEY  HERMIONE GRANGER  \
        SORCERER STONE                   141.351       47.602            29.664
        CHAMBER OF SECRETS               159.369       78.557            31.224
        PRISONER OF AZKABAN              149.567       68.196            44.165
        GOBLET OF FIRE                   103.896       91.440            29.137
        ORDER OF THE PHOENIX             128.195       63.199            39.236
        HALF-BLOOD PRINCE                130.245       49.585            32.546
        DEATHLY HALLOWS (PART 1 AND 2)   124.315       60.461            54.717

                                        VOLDEMORT  DUMBLEDORE  HAGRID  DRACO MALFOY  \
        SORCERER STONE                      5.436      22.853  53.403        13.619
        CHAMBER OF SECRETS                  3.247      20.357  21.093        23.424
        PRISONER OF AZKABAN                 3.532      11.281  27.362        16.343
        GOBLET OF FIRE                      6.507       3.149   1.459         8.282
        ORDER OF THE PHOENIX                5.911      28.657  16.230         6.455
        HALF-BLOOD PRINCE                  15.609      66.981  15.092        20.805
        DEATHLY HALLOWS (PART 1 AND 2)     16.536      33.336  11.902         1.443

                                        SIRIUS BLACK  NEVILLE LONGBOTTOM
        SORCERER STONE                         5.867              12.972
        CHAMBER OF SECRETS                     5.478               3.831
        PRISONER OF AZKABAN                   31.253              11.076
        GOBLET OF FIRE                        10.485               1.214
        ORDER OF THE PHOENIX                  24.405               7.192
        HALF-BLOOD PRINCE                      7.490               3.825
        DEATHLY HALLOWS (PART 1 AND 2)         7.959               0.435
```

*the cosine similarity was multiplied by 1000 for easier viewing and distinction between the values. All calculations past this point will be based off the multiplied cosine similarity values.

2.3 Get screen times

Using IMDB's movie database (https://www.imdb.com/list/ls027460372/), we can get the screen times for each of the actors in the films.

```
In [6]: #screen times are in order of the books as shown above
        screen_time={"HARRY POTTER":[72.75,83.75,74.5,63.25,61.75, 67.00,round(58+8/60,2)],
                     "RON WEASLEY":[28.25,38.25, 21.25,20.5,21.0,21.75,round(30+23/60,2)],
                     "HERMIONE GRANGER":[23.25,15.5,34.75,16.5,23,20,36],
                     "VOLDEMORT":[2,6.75,0,6.5,2.25,4.25,7.75],
                     "DUMBLEDORE":[9.75,10.75,6.25,14.25,7.5,22.25,3.25],
                     "HAGRID":[16.5,8.25,5.25,3.75,2.75,4,round(2+38/60,2)],
                     "DRACO MALFOY":[4.25,7,4,2.25,1.25,8.25,2.33],
                     "SIRIUS BLACK":[0,0,11.5,1.25,7.5,0,0],
                     "NEVILLE LONGBOTTOM":[3.25,1.25,3,4.25,9,1.5,round(7/60,2)]
                    }
        pd.DataFrame(screen_time, index=books)
```

```
Out[6]:                                  HARRY POTTER  RON WEASLEY  HERMIONE GRANGER  \
        SORCERER STONE                          72.75        28.25             23.25
        CHAMBER OF SECRETS                      83.75        38.25             15.50
        PRISONER OF AZKABAN                     74.50        21.25             34.75
        GOBLET OF FIRE                          63.25        20.50             16.50
        ORDER OF THE PHOENIX                    61.75        21.00             23.00
        HALF-BLOOD PRINCE                       67.00        21.75             20.00
        DEATHLY HALLOWS (PART 1 AND 2)          58.13        30.38             36.00

                                        VOLDEMORT  DUMBLEDORE  HAGRID  DRACO MALFOY  \
        SORCERER STONE                       2.00        9.75   16.50          4.25
        CHAMBER OF SECRETS                   6.75       10.75    8.25          7.00
        PRISONER OF AZKABAN                  0.00        6.25    5.25          4.00
        GOBLET OF FIRE                       6.50       14.25    3.75          2.25
        ORDER OF THE PHOENIX                 2.25        7.50    2.75          1.25
        HALF-BLOOD PRINCE                    4.25       22.25    4.00          8.25
        DEATHLY HALLOWS (PART 1 AND 2)       7.75        3.25    2.63          2.33

                                        SIRIUS BLACK  NEVILLE LONGBOTTOM
        SORCERER STONE                          0.00                3.25
        CHAMBER OF SECRETS                      0.00                1.25
        PRISONER OF AZKABAN                    11.50                3.00
        GOBLET OF FIRE                          1.25                4.25
        ORDER OF THE PHOENIX                    7.50                9.00
        HALF-BLOOD PRINCE                       0.00                1.50
        DEATHLY HALLOWS (PART 1 AND 2)          0.00                0.12
```

2.4 Get average cosine similarity and screen time for each character

```
In [8]: average_similarities = {}
        average_screen_times = {}
        for character in characters:
            average_similarities[character]=sum(cosine_similarities[character])\
            /len(cosine_similarities[character])
            average_screen_times[character]=sum(screen_time[character])\
            /len(screen_time[character])

        HTML("""
        <div align="middle">
        <img width="75%" src="average.png">
        </img></div>""")
```

```
Out[8]: <IPython.core.display.HTML object>
```

2.5 Plot on graph to see relationship

```
In [11]: import warnings
         warnings.filterwarnings('ignore')
         import plotly.plotly as py
         import plotly.graph_objs as go
         from scipy import stats
         import numpy as np

         sims = np.array(list((average_similarities.values())))
         times = np.array(list((average_screen_times.values())))

         slope, intercept, r_value, p_value, std_err = stats.linregress(sims,times)
         line = slope*sims+intercept

         trace1 = go.Scatter(
             x = sims,
             y = times,
             mode = 'markers',
             text = characters
         )

         trace2 = trace2 = go.Scatter(
                           x=sims,
                           y=line,
                           mode='lines',
                           marker=go.Marker(color='rgb(31, 119, 180)'),
                           name='Fit'
                           )

         layout= go.Layout(
             title= 'Average screen time vs cosine similarity \
             for harry potter characters over all books and movies',
             hovermode= 'closest',
             xaxis= dict(
                 title= 'Cosine Similarity',
                 ticklen= 5,
                 zeroline= False,
                 gridwidth= 2,
             ),
             yaxis=dict(
                 title= 'Screen Time (minutes)',
                 ticklen= 5,
                 gridwidth= 2,
             ),
             showlegend= False
         )
         data = [trace1, trace2]
         fig= go.Figure(data=data, layout=layout)

         # Plot and embed in ipython notebook!
         py.iplot(fig, filename='basic-scatter')
```

```
Out[11]: <plotly.tools.PlotlyDisplay object>
```

As we can see from the graph above, there is a strong positive correlation between cosine similarity and screen time. (more discussed in conclusion)

2.6 Levels of Linguistic Analysis

2.6.1 Syntax

Word and phrase creation is a difficult task. To make it easier on ourselves, we formulate phrases through a combination of meaningful words and connecting words. This gives us time to think without pausing too often. Beniot Mandelbrot, a famous mathemetician, said that nature always looks for the path of least energy and language may have evolved in a similar process in order to keep language as fluid and understandable as possible.

2.6.2 Semantics

As explored above, the meaning of speech and writing is held in a select few words. These words don't occur often, but that is what makes them influential. The downside to this is that meaning takes long to convey. The Zipf-ian nature of language seems to have caused verboseness to be an inherent part of our language. If every word we said had equally influential meaning, then communication could occur much more efficiently. The question is whether the human brain would be capable of producing coherent information that quickly. Unfortunatly, that is probably not the case. Moreover, for the listener, the meaning is not always easy to find since it exists in a sea of meaningless words. You have to be constantly listening for the important parts because if you lose focus for a brief time, you may miss them. In all, it would make sense for language to evolve for ease of production and retention, but that doesn't seem to be the case. It may be that we are in the nascent stages of language and it may take longer for it to reach its maximum potential.

2.7 Conclusion

In this experiment, I was attempting to uncover the interlinked nature of literature and film through analyzing the content of the Harry Potter books. More specifically, I wanted to see how well the characters in the books were depicted on-screen. To do this, I used the cosine similarity of a character's name and each book as a proxy for the relative importance of the term. I believe this is a fair assumption because, many times, the use of a name indicates that the character had a line of dialogue, which directly relates to them being onscreen. In addition, the use of a character's name in any regard works to increase their importance in the overall scheme of the book as it shows they are more integral to the plot. My hypothesis was that the higher the cosine similarity (loosely translated as the importance of the name in the book), the higher the screen time for that character.

This experiment confirmed my hypothesis and showed that the importance of the characters in the books was accurately portrayed through the film. As seen from the graph above, the relationship is linear with an r^2 value of 0.974, which indicates a strong positive correlation. This implies that the movie closely mirrored the events in the books and each character's contributions. Interestingly, as much as Voldemort is a major component of the plot of many of the films, his screen time doesn't correlate that well with the weight of his name in each of the books. This makes sense though as Voldemort is talked about a lot by other characters, increasing his cosine-similarity, but only present during the tension and action filled scenes towards the end of films.

In all, we explored how Zipf's law characterizes language, allowing for the Vector Space Model to be used effectively, and how different expressions of language - books and movies - relate to one another.

3 Appendix

```
In [ ]: #using manim math engine, code for visualization of log function
        class LOG(GraphScene):
            CONFIG = {
            "x_min" : -5,
            "x_max" : 5,
            "y_min" : 0,
            "y_max" : 25,
            "graph_origin" : DOWN*3,
            "function_color" : RED,
            "axes_color" : GREEN,
            "x_axis_label": None,
            "y_axis_label": None,
            }

            def construct(self):
                self.setup_axes(animate=True)
                graph = self.get_graph(lambda t : np.exp(t),color=self.function_color)
                self.play(ShowCreation(graph))
```

```
In [ ]: #using manim math engine, code for visualization of cosine similarity
        class CS(Scene):
            CONFIG = {
```

```
"x_min" : -10,
"x_max" : 10,
"y_min" : -10,
"y_max" : 10,
"function_color" : RED,
"axes_color" : GREEN,
}

def construct(self):
    plane = NumberPlane()
    vector1=Vector(3*UP+RIGHT)
    vector2=Vector(3*RIGHT+UP)
    group=VGroup(vector1,vector2)
    x_line=Line(ORIGIN,RIGHT, color=RED)
    y_line=Line(RIGHT,3*UP+RIGHT, color=GREEN)
    ordered_pair = TexMobject("[1,3]")
    ordered_pair.next_to(vector1.get_tip())
    self.play(ShowCreation(
        plane,
        submobject_mode = "lagged_start",
        run_time = 3
    ))
    self.play(GrowFromPoint(
        group,
        ORIGIN,
        submobject_mode = "lagged_start"),
        run_time=3)
    self.wait()
    self.play(ShowCreation(
        x_line
    ))
    self.play(ShowCreation(
        y_line
    ))
    self.play(Write(
        ordered_pair
    ))
    self.play(FadeOut(
        x_line
    ),FadeOut(
        y_line
    ))
    x_line=Line(ORIGIN,3*RIGHT, color=RED)
    y_line=Line(3*RIGHT,3*RIGHT+UP, color=GREEN)
    ordered_pair_2 = TexMobject("[3,1]")
    ordered_pair_2.next_to(vector2.get_tip())
    self.play(ShowCreation(
        x_line
```

```
))
self.play(ShowCreation(
    y_line
))
self.play(Write(
    ordered_pair_2
))
self.play(FadeOut(
    x_line
),FadeOut(
    y_line
))
theta = Arc(np.arctan(3)-np.arctan(1/3),start_angle=np.arctan(1/3))
angle = TexMobject("\\theta")
angle.next_to(theta)
angle.shift(UP*.3+LEFT*.3)
equation = TexMobject("cos(\\theta)=\\frac{a\\cdot b}{||a|| \\times ||b||}")
equation.shift(2.5*LEFT+2*DOWN)
equation_2 = TexMobject("cos(\\theta)=\\frac{[1,3]\\cdot \
                        [3,1]}{||[1,3]|| \\times ||[3,1]||}")
equation_2.shift(2.5*LEFT+2*DOWN)
equation_3 = TexMobject("cos(\\theta)=\
                        \\frac{1\\times 3 + 3 \\times 1}{\\sqrt{1^2+3^2} \
                        \\times \\sqrt{3^2+1^2}}")
equation_3.shift(2.5*LEFT+2*DOWN)
equation_4 = TexMobject("cos(\\theta)=\\frac{6}{\\sqrt{10} \\times \\sqrt{10}"
equation_4.shift(2.5*LEFT+2*DOWN)
equation_5 = TexMobject("cos(\\theta)=\\frac{3}{5}=Cosine \\hspace{1.5mm} Simi
equation_5.shift(2.5*LEFT+2*DOWN)
self.play(
    ShowCreation(theta),
    ShowCreation(angle)
)
self.play(Write(equation))
self.wait()
self.play(FocusOn(ordered_pair),FocusOn(ordered_pair_2))
self.wait()
self.play(FadeOut(equation),Write(equation_2), run_time=2)
self.wait()
self.play(FadeOut(equation_2),Write(equation_3), run_time=2)
self.wait()
self.play(FadeOut(equation_3),Write(equation_4), run_time=2)
self.wait()
self.play(FadeOut(equation_4),Write(equation_5), run_time=2)
self.wait()
```