Simon Zuberek
Ling 78100/73800: Methods in Computational Linguistics I
Dr. Gorman
12/23/2019

The Final Project Report

1. Introduction

The Winter Holiday season is upon us, delivering the usual blend of chilly weather, contentious Santa Claus impersonations, and Christmas films. This year's movie list seems to be once again dominated by the Star Wars universe. And although the release frequency of new episodes from the Galaxy far away can be rivaled only by the ubiquity of Mariah Carey's Christmas smasher, the latest installment of the Skywalker Saga promises to break the mold. First and foremost, it is poised to end Star Wars as we know it. It is also expected to tie up any loose plots, introduce a few new characters, retire some old ones, and answer a myriad of questions generated by its eight precursors - all in under three hours. While the life-long Star Wars in me is confident that the movie will meet these expectations, the geek in me has questions. In particular, the relationship between Luke Skywalker and Darth Vader deserves some answers before the Saga's bombastic conclusion.

2. Hypothesis

Luke and Vader, the main protagonist and his dark nemesis, son and father - the cultural preeminence of their tormented kinship elevated it to the status of a meme. They say that every son quotes his father in words and deeds. Yet, it is the actions of the two Skywalkers rather than their dialogue lines that captivated the attention from the media and fans alike. The purpose of this project is to address what has long been neglected, and to investigate what Luke and Vader can tell us about their family dynamic. Put succinctly: Does Luke Skywalker actually speak like his father, Darth Vader? If the two characters are indeed patrilineally related then we can hypothesize that this connection should be reflected in their dialogues, i.e. that Luke's and Vader's language should be similar.

3. Method

a. Obtain the Data

The first step was to procure the data. Luckily, the screenplays for Lucas's Original Trilogy are easily obtainable online. After a bit of searching, I was able to find pre-processed screenplays for all three movies. The files had already been cleaned up, containing only the enumerated dialogue lines for all the characters in each movie. I downloaded the files from kaggle, and renamed them 'ep#.txt', where the # stands for each episode's number.

b. Prepare the Data

Each file contains a list of labelled and numbered dialogue lines. While the character name markers, indicating who says what will be useful down the road, the line numbers will not. The first step was therefore to get rid of them. The first loop does exactly that. It opens the file in the read mode, sets the encoding to UTF-8, and stores the contents in 'input'. The loop iterates over each line in the file, replacing the line-initial numbering with an empty string, and case-folds all characters. Thus "cleaned" lines are then written to the output file 'ep#_out.txt', with a matching encoding. The output file is assigned to the variable 'path'.

c. Isolate the Main Characters

Now that cleaned-up dialogue lines for each character have been saved in a separate file, we need to isolate the lines of the characters relevant to the project. Here we will only look at Luke and Vader, though the script will work for other characters too. We open the file, read it, and store everything in the variable 'lines'. As the file is relatively small I used the '.readlines' method to read all lines at once. For larger files this would need to be done on a per-line basis. I then created a compiler object and stored it in the variable 'vader'. The object uses a regular expression to match every instance of the string '"vader"' followed by one or more instances of any character. The compiler is followed by the 'vader_lines' empty list, which will be used to store everything that Vader said in the film. The for-loop iterates over each line stored in the 'lines' variable, searching for the lines matching the regular expression defined in the compiler. If a matching line is found, it is added to the 'vader_lines' list. Thus created list of strings 'vader_lines' is a local variable. To use it outside of the 'prep_text()' function it needs to be turned global. The next two lines define the 'vader_lines_str' global variable and assign it an empty string. The following loop iterates over 'vader_lines', removing the character marker "vader" as well as all quotation marks and commas. Cleaned lines are then stored in the 'vader_lines_str' string variable.

The above-outlined process is repeated for Luke, with the final product of this function being two global string variables 'vader_lines_str' and "luke_lines_str', containing Vader's and Luke's dialogue lines respectively. To check if everything has processed the way intended, both strings are printed to the console.

d. Process the Main Character Dialogues

Luke's and Vader's strings are assigned to the 'luke' and 'vader' variables respectively. To analyze their dialogue lines, we first need to split them into sentences and then into words. To do that I will use a function named 'word_sentence_tokenize'. To make the code more transparent (and to experiment a bit), I saved it in a separate file, which I then imported into the main project file. The function creates a sentence tokenizer using PunktSentenceTokenizer, imported from the nltk.tokenize library. The tokenizer takes 'text' for an argument and is assigned to the 'sentence_tokenized' variable. It is then used to sentence-tokenize the 'text' and store it in the 'sentence_tokenized' variable. 'Word_tokenized' creates an empty list that will store word-tokenized sentences. The following for-loop iterates over each tokenized sentence stored in 'sentence_tokenized', word-tokenizing it, and appending the result to the 'word_tokenized' list. The function returns a word-tokenized list of all sentences and most importantly, it actually works when imported! Its effectiveness is confirmed by printing a randomly selected sentence to the console. With sentences isolated and words tokenized, we can now move to parts-of-speech tagging. The for-loop iterates over each word-tokenized sentence in 'vader_tokenized', tags it for parts of speech using the 'pos_tag()' function imported from nltk, and appends it to the 'pos_tagged_vader' counter list. The same randomly-selected sentence - this time pos-tagged - is then printed to the console for validation.

At this point, Vader's lines are broken into sentences, tokenized, and tagged for parts of speech. One can iterate over them, count the parts of speech, and juxtapose the results against the parts of speech uttered by Luke. From a lexical standpoint this would a fine comparison, if not necessarily a particularly useful one. For even though we employ words to communicate, we don't communicate words but thoughts and ideas. In that sense, lexical analysis of the dialogues is unlikely to reveal the kind of speech patterns we are trying to access. One way to gain insight into those leads through syntactic analysis. A simple English sentence consists of an implicitly or explicitly stated subject and a verb. These are usually expressed as noun and verb phrases respectively. A noun phrase consists of an

optional determiner, followed by an equally optional adjective or adjectives, followed by a noun. A verb phrase in turn is a noun phrase followed by a verb, which can then be optionally followed by one or more adverbs, another noun phrase, or a prepositional phrase. Lastly, a prepositional phrase is comprised of a preposition and a noun phrase.

Noun phrases are a bit less complex, so let us start with those. The variable 'np_chunk_grammar' stores the grammar chunk 'NP: {<DT>?<JJ.?>*<NN>}' that will be used to isolate noun phrases (NP). True to the definition of a noun phrase, the chunk contains an optional determiner <DT>, followed by zero or more adjectives <JJ.?>, where '.?' accounts for comparative and superlative adjective forms, followed by a noun <NN>. This chunk grammar object is then passed through RegexpParser to create 'np_chunk_parser', which we will use to parse Vader's lines for noun phrases. Analogously, a verb-phrase parser 'vp_chunk_parser' is created to parse for verb phrases (VP). Here the verb-phrase chunk grammar also reflects the description from the previous paragraph, where a verb phrase consists of a noun phrase, followed by a verb <VB.?> (where the '.?' accounts for tenses, participles, and conjugation), followed by zero or more adverbs, or noun phrases, or prepositional <IN> phrases.

With grammar chunks defined, we create two empty list counters, one for noun phrases and the other one for verb phrases. The subsequent for-loop iterates over each sentence in 'pos_tagged_vader' and uses 'np_chunk_parser' to parse it for noun phrases. The results are then appended to the 'np_chunked_vader' list, which at this point should contain all the noun phrases uttered by Vader in the movie. An analogous procedure is followed to compile a list of Vader's verb phrases. We now have two lists: one with all the verb phrases, and one with all the noun phrases; let's count them.

We will use two counter functions: 'np_chunk_counter()' and 'vp_chunk_couner()' for counting most common noun-phrase and verb-phrase chunks respectively. Both functions are stored in 'chunk_couners.py' and imported into the project file. The 'np_chunk_counter' function first creates an empty chunk counter list and assigns it to the 'chunks' variable. The for-loop then iterates through each chunked sentence and extracts all noun-phrase chunks. Writing this function was one of the more challenging parts of this project as the syntax was not covered in class. After a few hours with stack overflow this tutorial the function was ready. A counter object is then created and assigned to the 'chunk_counter' variable. The for-loop iterates over the list of chunks and increases the counter by one each time it detects a noun phrase. The function returns ten most frequently occuring chunks. An analogous function counting verb phrases follows.

The 'np_chunk_counter' function counts the frequencies of noun phrases in the 'np_chunked_vader' list and stores them in the top_np_chunks variable. Likewise, 'vp_chunk_counter' produces a list of most frequently occurring verb-phrase chunks in 'vp_chunked_vader'. The results are then printed to the console. Since we are comparing Lord Vader to Luke Skywalker, an analogous function 'process_luke' is defined and called, returning Luke's most-commonly used verb and noun phrases.

4. Results and Discussion

The analysis of Vader and Luke's dialogues from Star Wars ep. IV "A New Hope" (Fig. 1) shows that the two characters differ in the number and type of noun-phrases and verb-phrases they employ. This, being the first Star Wars film released, is perhaps unsurprising. The unsuspecting viewer has not yet heard the legendary "Luke, I am your father!", and has little reason to anticipate the looming revelation. In fact, at this point it may be in the director's interest to keep the viewer in the dark about the upcoming plot twist.

That being said, the beans are finally spilled in "The Empire Strikes Back" - the second part of the original trilogy. At this point any similarities between the two characters would only serve to heighten

the tension, complicate the plot of the story, and make the movie more entertaining. Lastly, “Return of the Jedi”, the third movie in the trilogy, sets up the final, epic confrontation between Vader and Luke. Here the viewer witnesses Luke’s gradual metamorphosis from a naive apprentice into a Jedi knight. Over the course of his conversion, Luke’s appearance is becoming increasingly resembling of Vader’s, sporting dark clothes, a green lightsaber, and a mechanical hand. It would therefore stand to reason that Luke’s language would become increasingly like to his father’s as well. For better or worse, none of that seems to be the case.

Whereas Luke is becoming more like his infamous father, the transformation is not echoed in his language. We see this illustrated by the analysis of the dialogues from all three episodes (‘new_hope.py’, ‘empire_strikes_back.py’, and ‘return_of_the_jedi.py’, as shown in Fig.2). The reasons for that are many, and as interesting as they may be to investigate, such a task exceeds the scope of this project.

Here it should be mentioned that this project’s scope is limited from the outset. The relatively small corpus we are working with does not lend itself to conclusive analysis, as may be glanced from the presented data. The fact that the majority of the counted noun and verb-phrases occur only once or twice in each movie does not exactly invite firm conclusions. An interesting follow-up project may take a look at the aggregate of all of Luke’s dialogues across the six movies where featuring him, and compare them to the six movies featuring Anakin Skywalker / Darth Vader. A larger corpus should invite results that are more robust and a more conclusive analysis. Another follow-up project could analyze the sentiment expressed by the two characters, by looking into the topics dominating their dialogue lines.

The above are only two ways in which the screenplays can be analyzed. Though very much imperfect, the script is rather robust and it should work with other screenplays from the Star Wars universe. Given appropriately formatted data, it could help unearth unique insights into the Galaxy far, far away.

Fig 1.

Vader				**Luke**			
NPs	**#**	**VPs**	**#**	**NPs**	**#**	**VPs**	**#**
I	11	I want	2	I	76	I’m	8
the force	5	the force is	2	threepio	12	I’ve	5
the ambassador	2	all aboard were	1	all right	11	I was	4
highness	2	a planet is	1	uncle	10	I want	3
this time	2	probe is	1	look	9	I didn’t	2
this ship	2	I told	1	something	8	I’m sorry	2
the rebellion	2	I want every part of this ship	1	a lot	6	I’m never	2
the end	2	I haven’t	1	owen	6	I thought	2
stay	2	I felt	1	unit	5	I think	2
formation	2	Obi-Wan is here	1	hey	4	I’m not	2

Fig. 2

Episode IV: A New Hope (new_hope.py)				Episode V: The Empire Strikes Back (empire_strikes_back.py)				Episode VI: Return of the Jedi (return_of_the_jedi.py)			
Vader		Luke		Vader		Luke		Vader		Luke	
NPs (#)	VPs (#)	NPs (#)	VPs (#)	NPs (#)	VPs (#)	NPs (#)	VPs (#)	NPs (#)	VPs (#)	NPs (#)	VPs (#)
I (11)	I want (2)	I (76)	I'm (8)	I (7)	I am (2)	I (45)	I'm (6)	master (7)	the emperor has (2)	I (45)	I've (5)
the force (5)	the force is (2)	threepio (12)	I've (5)	skywalker (5)	sure Skywalker is (1)	Ben (11)	I've (4)	the emperor (6)	I'm here (1)	father (13)	I have (3)
the ambassador (2)	all aboard were (1)	all right (11)	I was (4)	captain (4)	admiral Ozzel came (1)	Artoo (10)	I want (2)	I (5)	The emperor does not share (1)	Solo (5)	I know (2)
highness (2)	a planet is (1)	uncle (10)	I want (3)	the emperor (4)	nothing gets (1)	something (5)	I'm not (2)	the dark side (4)	The emperor is not as (1)	Leia (4)	I'm (2)
this time (2)	probe is (1)	look (9)	I didn't (2)	master (3)	I want that ship (1)	Leia (5)	I don't (2)	son (3)	I am (1)	Jabba (3)	I am Luke Skywalker Jedi knight (1)
this ship (2)	I told (1)	something (8)	I'm sorry (2)	Obi-Wan (3)	I want every ship (1)	Han (4)	old buddy do (1)	schedule (2)	the rebel fleet massing near sullust (1)	master (3)	I seek an audience (1)
the rebellion (2)	I want every part of this ship (1)	a lot (6)	I'm never (2)	destiny (3)	apology accepted captain Needa (1)	hey (4)	I know (1)	the power (2)	shuttle going (1)	Han (3)	wisdom I (1)
the end (2)	I haven't (1)	owen (6)	I thought (2)	father (3)	I have Skywalker (1)	Dagobah (4)	group use (1)	destiny (2)	force has (1)	Yoda (3)	goodwill I (1)
stay (2)	I felt (1)	unit (5)	I think (2)	Luke (3)	I had (1)	Dack (4)	I see (1)	Obi-Wan (2)	son is (1)	Vader (3)	I warn (1)
formation (2)	Obi-Wan is here (1)	hey (4)	I'm not (2)	the system (2)	this facility is (1)	hang (4)	no I'm not (1)	share (1)	i see (1)	captain (2)	I used (1)