An Analysis of Sentence Structure in Bram Stoker's "Dracula" and Mary Shelley's "Frankenstein"

Main Idea

The purpose of this notebook is to analyze the sentences in Bram Stoker's "Dracula" and Mary Shelley's "Frankenstein." I'm going to take a machine learning approach where I attempt to build a model that uses individual words in a particular sentence taken from one of the two works, and then attempts to predict which of the two works it came from. While there is nothing inherently useful (at least in this case) about being able to predict which work a sentence came from, by examining the way the model solves the problem we can learn a little bit about each of the two works.

Downloading Text and Sentence Tokenizing

I used Project Gutenberg to download the full text of each of the works (see the main MP3 page for details). After loading each work as a string, I used NTLK's sentence tokenizer to chop up each string into sentences (you could do a rough approximation of this by splitting your string on periods, but the NLTK tokenizer is more advanced, e.g., it doesn't get fooled by abbreviations like Ms. or Mr.).

```
import nltk

f = open('dracula.txt')
```

```
dracula_text = ''.join(f.readlines())
f.close()

f = open('frankenstein.txt')
frankenstein_text = ''.join(f.readlines())
f.close()

dracula_sentences = nltk.sent_tokenize(dracula_text)
frankenstein_sentences = nltk.sent_tokenize(frankenstei
print("There are %d sentences in Dracula"% len(dracula_
print("There are %d sentences in Frankenstein"% len(fra
```

```
There are 8569 sentences in Dracula
There are 3199 sentences in Frankenstein
```

Vectorize the Data

In order to create a model to automatically categorize sentences as being from Dracula or Frankenstein, we first need to vectorize our data. The process of vectorization will take the data strings that represent each sentence and convert them into a vector of numbers. The reason that we are doing this is that there are many more examples of machine learning models that can handle numerical data than string data, so we are converting our data from a less usable to a more usable format.

The particular method we are going to be using to vectorize our data is called tf-idf. A full explanation of this is beyond the scope of this notebook, but the basic idea is to convert each sentence into a vector where each element of the resulting vector indicates how prominently a particular word factors in that sentence. In these two texts it turns out there are 12,712 unique words. Therefore, each sentence is represented as a 12,712 dimensional vector. However, Since most words don’t occur at all in a particular sentence, most of these entries are 0.

To really understand these vectors, you should the Wikipedia article or talk to a member of the teaching team.

```
from sklearn.feature_extraction.text import TfidfVector

vectorizer = TfidfVectorizer()
vectorizer.fit(dracula_sentences + frankenstein_sentenc

X_dracula = vectorizer.transform(dracula_sentences)
X_frankenstein = vectorizer.transform(frankenstein_sent

print("The vectorized Dracula data has %d rows and %d c
print("The vectorized Frankenstein data has %d rows and
```

```
The vectorized Dracula data has 8569 rows and 12712 col
The vectorized Frankenstein data has 3199 rows and 1271
```

Splitting the Data into Training and Testing Sets

Before we can do machine learning, we need to do a few things. The first thing is that we have to tell the machine learning algorithm what task we want it to solve. We do this by giving it a bunch of vectors and corresponding labels. The vectors in this case will be the vectorized sentences (i.e. converted into tf-idf features), and the labels will correspond to a 1 if the sentence came from Dracula and a 0 if it came from Frankenstein (which text gets a 1 versus a 0 is arbitrary).

Once we have created the appropriate dataset format, we will randomly split our data into a training and test set. We will use the training set to construct our model (recall that the model's job will be to predict whether a sentence is from Dracula or Frankenstein) and we will use the testing set to evaluate how well the model learned on the training set actually works on new data. We don't want to just test the model on the training set, because that would give us an overinflated sence of how well the model is doing.

```
import numpy as np
```

```
from sklearn.model_selection import train_test_split
from scipy import sparse

X = sparse.vstack((X_dracula, X_frankenstein))
y = np.hstack((np.ones(X_dracula.shape[0]), np.zeros(X_

X_train, X_test, y_train, y_test = train_test_split(X,

print("The training set has %d rows and %d colums" % X_
print("The testing set has %d rows and %d columns" % X_
```

```
The training set has 7884 rows and 12712 colums
The testing set has 3884 rows and 12712 columns
```

Training our Model

For the actual machine learning algorithm we are going to use a technique called multiple logistic regression (you'll also see this called "multivariate logistic regression", or occasionally just "logistic regression"). The basic idea is that our model will fit a set of weights over each entry of the input vectors (remember, each entry in the vector corresponds to a particular word). Each weight will tell us to what degree a particular word is associated with either Frankenstein or Dracula. If the weight is very positive, this means that the corresponding word is associated with the novel Dracula. If the weight is very negative, this means that the corresponding word is associated with the novel Frankenstein. As a side note, the only reason taht positive corresponds to Dracula and negative corresponds to Frankenstein is that when we setup the task we arbitrarily assigned a "1" to sentences from Dracula and a "0" to sentences from Frankenstein.

One we fit the model, we will test how well it does on predicting sentences from the test set. As a baseline, I am printing out the accuracy we would get if we always predicted "Dracula" for every sentence (since there are more sentences of this type).

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

model = LogisticRegression()
model.fit(X_train, y_train)

print("The model achieves an accuracy of %f" %accuracy_
print("Always predicting Dracula gives an accuracy of %
```

```
The model achieves an accuracy of 0.881050
Always predicting Dracula gives an accuracy of 0.737899
```

Interpreting the Model

As I said at the top of this notebook, there isn't really much utility to accurately classifying sentences as being from Dracula or Frankenstein (as a side note, this would be useful for author attribution in cases where authorship is not firmly established, e.g., The Federalist Papers).

In this next cell, we are going to grab the model weights corresponding to the words most highly associated with Frankenstein and those most highly associated with Dracula.

```
# sort the weights (highly negative is associated with
# and highly positive is associated with Dracula)
sorted_weights = np.argsort(model.coef_)

print("Top 50 words associated with Frankenstein")
for ind in sorted_weights[0,:50]:
    print(vectorizer.get_feature_names()[ind])
print()
print("Top 50 words associated with Dracula")
for ind in reversed(sorted_weights[0,-49:]):
    print(vectorizer.get_feature_names()[ind])
```

```
Top 50 words associated with Frankenstein
my
elizabeth
of
miserable
```

```
clerval
father
justine
towards
feelings
but
wretch
thus
which
by
cottage
ice
felix
countenance
human
victor
misery
beheld
wretched
upon
these
although
often
frankenstein
return
peace
fiend
yet
lake
months
tale
should
mountains
entered
while
murderer
every
this
spent
cousin
appeared
```

```
creature
family
and
also
innocence

Top 50 words associated with Dracula
there
we
all
lucy
count
mina
out
back
so
it
is
know
though
van
helsing
come
up
see
he
way
professor
went
arthur
tell
get
too
harker
diary
things
go
think
jonathan
over
must
```

```
got
down
seemed
came
window
quite
help
off
coming
door
here
husband
face
sort
look
```

Interpreting the Results

This is section is a work in progress. These should as hypotheses about the texts that were generated by the analysis above. More work is needed to verify them.

Why is “my” the number one word for Frankenstein

According to Sparknotes the narrator of Frankenstein is:

> *The primary narrator is Robert Walton, who, in his letters, quotes Victor Frankenstein’s first-person narrative at length; Victor, in turn, quotes the monster’s first-person narrative; in addition, the lesser characters Elizabeth Lavenza and Alphonse Frankenstein narrate parts of the story through their letters to Victor.*

According to Sparknotes the narrator of Dracula is:

> *Dracula is told primarily through a collection of journal entries, letters, and telegrams written or recorded by its main characters: Jonathan Harker, Mina Murray, Dr. John Seward, Lucy Westenra, and Dr. Van Helsing.*

This difference in narrators could explain why the word "my" is much more associated with Frankenstein than Dracula.

Explanation of Names and Associations to Frankenstein or Dracula

Some of the top 50 words for each work seem to correspond to names of the characters. Here are a few of the characters that are prominent in each story.

Frankenstein

- Elizabeth Lavenza
- Victor Frankenstein

Dracula

- Jonathan Harker
- Mina Murray
- Dr. John Seward
- Lucy Westenra
- Dr. Van Helsing.

Why is "there" the number one word for Dracula

I’m not totally sure on this one, but I think it has to do with both a stylistic difference between the two writers as well as the structure of the narrative. According to Sparknotes the narrative in Dracula is mostly in the past tense.

> *Though some of the entries record the thoughts and observations of the characters in the present tense, most incidents in the novel are recounted in the past tense.*

In Frankenstein, the quotes from Victor Frankenstein’s personal narrative are often in the present tense.

Another way to examine this is that the phrase “there was” occurs 19 times in Frankenstein and 119 times in Dracula (to be fair, Frankenstein is half as long as Dracula). If we just look at the word “there” it occurs 109 times and 599 times in Frankenstein and Dracula respectively.

Source