NLP FINAL PROJECT REPORT

(DSCI-6004-01-S20)

On

Text Classification, Summarization and Generation

By

RAHUL AKKINENI (ID: 00689658)

Under the guidance of

KEITH DILLON, Ph.D.

Tagliatela College of Engineering

Department of Data Science

300 Boston Post Rd, West Haven, CT 06516

ACKNOWLEDGEMENT

My most sincere and grateful acknowledgement is due to this sanctum, **University Of Newhaven,** for giving us the opportunity to fulfill my aspirations and for good career.

I express my heartfelt gratitude to my guide **Mr. Keith Dillon, Ph.D,** for his esteemed guidance. I am indebted for his guidelines that proved to be very much helpful in completing my project successfully in time.

My special thanks to my classmates and seniors for providing their special guidance and support for the completion of my project successfully.

Submitted By

Rahul Akkineni

(00689658)

1. THE DATA AND THE BACKGROUND

1.1 THE DATA:

The data here is taken from the Gutenberg Online : https://www.gutenberg.org/wiki/Main_Page.

The data I have taken is from crime and horror genres. I have manually downloaded each and every book in the respective genres and made two folders for both the genres and moved the files to the respective folders. The crime genre contains 29 books and the horror genre contains 41 books. The folders be like: crime/crime1.txt, crime/crime2.txt, ---- and horror/horror1.txt, horror2.txt, --- etc.,

Each text file contains a different story.

1.2 METHODS I AM TRYING TO SOLVE:

- Text data
- Scikit-learn and Keras for Text Classification
- Text Summarization using words and sentences
- Text Generation using Multilayer Neural Networks

1.3 THE BACKGROUND:

Generating text with semantic meaning is a very difficult process. The Neural Networks are the best approach for this problem. The solution is predicting the next word given a sequence of previous words. In this project I have trained a model to classify the given text to identify the genre to which it belongs. Then summarize the text and send it into the model trained for generating the text. Then use the generated text to find the similarity between the generated text and the original text. The purpose is to create new and improved stories or texts for testing a person (who doesn't know if the text is generated or original) if he can tell whether the text is computer generated or written by a writer and see if he can understand the text like he did if he was given the normal texts and check if he gives the text a better rating so that we can sell the text by making it into a book and make money.

3. APPROACHES

CLASSIFICATION

3.1. TEXT DATA:

First of all I have loaded all the text data in crime books to one string and all the horror books into another string.

```
In [160]:
text1 = ''
for i in range(1, 30):
    t = 'F://Rahul Subjects/Unstructured Data and NLP/NLP Project Books/crime/crime'
    j = str(i)
    f = open(t + j + '.txt', encoding="utf8")
    text1 = text1 + '\n\n--------------------------------------------------------------------------------\n\n' + f.read()
```

```
In [161]:
text2 = ''
for i in range(1, 42):
    t = 'F://Rahul Subjects/Unstructured Data and NLP/NLP Project Books/horror/horror'
    j = str(i)
    f = open(t + j + '.txt', encoding="utf8")
    text2 = text2 + '\n\n--------------------------------------------------------------------------------\n\n' + f.read()
```

3.2. PREPROCESSING DATA:

Since this is text data we need to first clean all the lines so that we can separate the sentences and tokenize them for building models.

```
In [3]:
replacements = [':', ';', '"', "'", ',', '-', '”', '“', '***', '\r', '\n', '  ']
def sentencer(string):
    for i in range(len(replacements)):
        string = string.replace(replacements[i],' ')
        string = string.lower()
    return string
```

```
In [4]:
for r in replace:
    my_str1 = text1.replace(r, '.')

sen1 = my_str1.split('.')
for i in sen1:
    print('Sentnece( '+i+') \n')
```

```
The Project Gutenberg Etext of An African Millionaire, by Grant Allen
#4 in our series by Grant Allen

Copyright laws are changing all over the world)

Sentnece(  Be sure to check the
copyright laws for your country before downloading or redistributing
this or any other Project Gutenberg file)
```

```
In [5]:
sen11 = sen1
```

```
In [6]:
 for i in range(len(sen1)):
        sen11[i] = sentencer(sen1[i])
```

```
In [7]:
len(sen11)
```

```
Out[7]: 162630
```

In [10]:
```
sen22 = sen2
```

In [11]:
```
for i in range(len(sen2)):
        sen22[i] = sentencer(sen2[i])
```

In [12]:
```
len(sen22)
```

Out[12]: 143972

3.2 TRAINING AND TESTING DATA:

Now we will combine the data from both genres and create a dataframe.

In [13]:
```
X = sen11 + sen22
y = []
```

In [14]:
```
for i in range(len(X)):
    if i<len(sen11):
        y.append(0)
    else:
        y.append(1)
```

In [15]:
```
data = []
for i in range(len(X)):
    row = []
    row.extend((X[i], y[i]))
    data.append(row)
```

In [18]:
```
df = pd.DataFrame(data,columns=['Text','Corpus'])
df.head()
```

Out[18]:

	Text	Corpus
0	the little woman stood a moment pensive and t...	1
1	my first idea now was mere surprise at the re...	1
2	the history of the gables seemed to be suscep...	0
3	she soon found one and armed with candle and ...	1
4	overhead the stars were brilliant in a sky qu...	1

3.2.1 Scikit-learn:

Now as we do always we split the data into training and testing.

In [21]:
```
from sklearn.model_selection import train_test_split
```

In [22]:
```
xtr1, xts1, ytr1, yts1 = train_test_split(df['Text'], df['Corpus'], test_size = 0.2)
```

Now we create a corpus to contain all words in the train data.

In [26]:
```
corpus=''
for i in range(len(xtr1)):
    corpus = corpus + xtr1.iloc[i]
```

Now we create bag of words.

In [29]:
```
bow = []
for i in range(len(d)):
    if d[i][1] >= 15:
        bow.append(d[i][0])
```

In [30]:

```python
def bagofwords(data, bow):
    dbow = []
    for x in range(len(data)):
        row = []
        for i in range(len(bow)):
            row.append(data.iloc[x].count(' '+bow[i]+' '))
        dbow.append(row)
    dbow = np.asarray(dbow)
    return dbow
```

Now we use the bag of words to create a matrix of 0's and 1's (vectors) that represent the sentences (data) in both the train and test sets.

In [31]:

```python
xb = bagofwords(df['Text'], bow)
```

In [32]:

```python
xtrb = bagofwords(xtr1, bow)
```

In [33]:

```python
xtsb = bagofwords(xts1, bow)
```

Now we use different models from scikit-learn to build the models.

Linear Regression:

In [34]:

```python
from sklearn.linear_model import LinearRegression
reg = LinearRegression().fit(xtrb, ytr1)
reg.score(xtsb, yts1)
```

Out[34]: -0.04945137295685797

The accuracy is too low (even in negative?) to use this model.

Logistic Regression:

In [35]:

```python
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression().fit(xtrb, ytr1)
logreg.score(xtsb, yts1)
```

```
c:\users\rahul\appdata\local\programs\python\python37\lib\s
g: Default solver will be changed to 'lbfgs' in 0.22. Speci
  FutureWarning)
```

Out[35]: 0.54

Better than previous but still not enough.

Decision Tree:

```
In [36]:
from sklearn import tree
clf = tree.DecisionTreeClassifier().fit(xtrb, ytr1)
clf.score(xtsb, yts1)

Out[36]: 0.58
```

Random Forest:

```
In [37]:
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(max_depth=2, random_state=0).fit(xtrb, ytr1)
clf.score(xtsb, yts1)

c:\users\rahul\appdata\local\programs\python\python37\lib\site-packages\sklearn\e
default value of n_estimators will change from 10 in version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)

Out[37]: 0.55
```

As we use models on and on from scikit-learn the accuracy is still around 60's. So we can't use the sklearn for text data.

3.2. MULTILAYER NUERAL NETWORK(KERAS):

First we need to tokenize and pad our data so that we can use them to build models.

```
In [41]:
X_train, X_test, y_train, y_test = train_test_split(df['Text'], df['Corpus'])

In [39]:
maxlen = 150
embedding_dim = 50

In [42]:
tokenizer = Tokenizer(num_words=5000)
tokenizer.fit_on_texts(X_train)

In [43]:
X_train = tokenizer.texts_to_sequences(X_train)
X_test = tokenizer.texts_to_sequences(X_test)

In [44]:
vocab_size = len(tokenizer.word_index) + 1
vocab_size

Out[44]: 3242

In [45]:
X_train = pad_sequences(X_train, padding = 'post', maxlen = maxlen)
X_test = pad_sequences(X_test, padding = 'post', maxlen = maxlen)
```

Now creating models from Keras.

First we will build a normal model to see the accuracy.

In [46]:
```
model = Sequential()
model.add(layers.Embedding(vocab_size, embedding_dim, input_length = maxlen, trainable = True))
model.add(layers.GlobalMaxPool1D())
model.add(layers.Dense(10, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.summary()
```

```
WARNING:tensorflow:From c:\users\rahul\appdata\local\programs\python\python37\lib\site-packages\tensorflow\python\ops\nn_im
pl.py:180: add_dispatch_support.<locals>.wrapper (from tensorflow.python.ops.array_ops) is deprecated and will be removed i
n a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 150, 50)           162100
_________________________________________________________________
global_max_pooling1d_1 (Glob (None, 50)                0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                510
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 11
=================================================================
Total params: 162,621
Trainable params: 162,621
Non-trainable params: 0
_________________________________________________________________
```

In [47]:
```
history = model.fit(X_train, y_train, epochs = 10, validation_split = 0.2, batch_size = 10)
```

```
WARNING:tensorflow:From c:\users\rahul\appdata\local\programs\python\python37\lib\site-packages\keras\backend\tensorflow_ba
ckend.py:422: The name tf.global_variables is deprecated. Please use tf.compat.v1.global_variables instead.

Train on 600 samples, validate on 150 samples
Epoch 1/10
600/600 [==============================] - 1s 1ms/step - loss: 0.6910 - accuracy: 0.5267 - val_loss: 0.6966 - val_accuracy:
0.4533
Epoch 2/10
600/600 [==============================] - 0s 518us/step - loss: 0.6807 - accuracy: 0.5717 - val_loss: 0.6933 - val_accurac
y: 0.4800
Epoch 3/10
600/600 [==============================] - 0s 503us/step - loss: 0.6574 - accuracy: 0.6383 - val_loss: 0.6828 - val_accurac
y: 0.5733
```

In [48]:
```
loss, accuracy = model.evaluate(X_test, y_test)
print("Loss: ", loss)
print("Accuracy: ", accuracy)
```

```
250/250 [==============================] - 0s 42us/step
Loss:  0.766281424999237
Accuracy:  0.5799999833106995
```

The accuracy seems to be same with scikit-learn when using base model.

Now we will use embedding and pooling to increase the accuracy.

In [31]:
```
model = Sequential()
model.add(layers.Embedding(input_dim = vocab_size, output_dim = embedding_dim, input_length = maxlen))
model.add(layers.Flatten())
model.add(layers.Dense(10, activation = 'relu'))
model.add(layers.Dense(1, activation = 'sigmoid'))
model.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 150, 50)           417150
_________________________________________________________________
flatten_1 (Flatten)          (None, 7500)              0
_________________________________________________________________
dense_5 (Dense)              (None, 10)                75010
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 11
=================================================================
Total params: 492,171
Trainable params: 492,171
Non-trainable params: 0
_________________________________________________________________
```

```
In [32]: history = model.fit(X_train, y_train, epochs = 10, validation_split = 0.2, batch_size = 10)
```

```
Train on 5695 samples, validate on 1424 samples
Epoch 1/10
5695/5695 [==============================] - 8s 1ms/step - loss: 0.5287 - accuracy: 0.7152 - val_loss: 0.3554 - val_accurac
y: 0.8357
Epoch 2/10
5695/5695 [==============================] - 8s 1ms/step - loss: 0.2488 - accuracy: 0.8904 - val_loss: 0.3259 - val_accurac
y: 0.8574
Epoch 3/10
5695/5695 [==============================] - 8s 1ms/step - loss: 0.1541 - accuracy: 0.9336 - val_loss: 0.3855 - val_accurac
y: 0.8525
```

```
In [33]:
loss, accuracy = model.evaluate(X_test, y_test)
print("Loss: ", loss)
print("Accuracy: ", accuracy)
```

```
2373/2373 [==============================] - 0s 55us/step
Loss:  0.4330313197681284
Accuracy:  0.8605141043663025
```

We can see that the accuracy has increase by a lot. So, now we use this model to predict the data.

```
In [88]:
def predict(word):
    word = tokenizer.texts_to_sequences(word)
    word = pad_sequences(word, padding = 'post', maxlen = maxlen)
    result = model.predict(word)
    if result[0][0]>0.4:
        #print(result[0][0])
        print('Crime genre')
    else:
        #print(result[0][0])
        print('Horror genre')
```

Predicting sentences using the model.

horror10.txt

```
In [83]: predict('All of them stared at the statuette as if they expected it to move again forthwith, under their very eyes.')
```

```
Horror genre
```

crime20.txt

```
In [85]: predict("It was rather less than two hours earlier on the same evening that Quentin Gray came out of the confectioner's
```

```
Crime genre
```

Predicting files using the model.

horror10.txt

```
In [81]:
f = open("F://Rahul Subjects/Unstructured Data and NLP/NLP Project Books/horror/horror10.txt").read()
predict(f)
```

```
Horror genre
```

crime10.txt

```
In [86]:
f = open("F://Rahul Subjects/Unstructured Data and NLP/NLP Project Books/crime/crime10.txt").read()
predict(f)
```

```
Crime genre
```

SUMMARIZATION

Now that we made the classification we need to do the summarization. First we will make sentences and then tokenize them.

In [147]:
```
def tokenizer(s):
    tokens = []
    replace = ['.', '?', '!']
    replacements = [':', ';', '"', "'", ',', '-', '”', '“', '***', '\r', '\n', '  ']
    for r in replace:
        s = s.replace(r, '.')
    for r in replacements:
        s = s.replace(r,' ')
    for word in s.split(' '):
        tokens.append(word.strip().lower())
    return tokens

def sent_tokenizer(s):
    sents = []
    for sent in s.split('.'):
        sents.append(sent.strip())
    return sents
```

In [148]:
```
text = open("F://Rahul Subjects/Unstructured Data and NLP/NLP Project Books/horror/horror10.txt", encoding="utf-8").read
```

In [149]:
```
tokens = tokenizer(text)
sents = sent_tokenizer(text)

print(tokens)
print(sents)
```

```
atsoever.', 'you', 'may', 'copy', 'it', 'give', 'it', 'away', 'or', 're', 'use', 'it', 'under', 'the', 'terms', 'of', 'the'
'project', 'gutenberg', 'license', 'included', 'with', 'this', 'ebook', 'or', 'online', 'at', 'www.gutenberg.net', '', 'tit
e', 'the', 'night', 'land', 'author', 'william', 'hope', 'hodgson', 'release', 'date', 'january', '9', '2004', '[ebook', '#
662]', 'language', 'english', '', '', 'start', 'of', 'this', 'project', 'gutenberg', 'ebook', 'the', 'night', 'land', '', '
'', 'produced', 'by', 'suzanne', 'shell', 'maria', 'khomenko', 'and', 'pg', 'distributed', 'proofreaders', '', '', 'the', '
```

Now we make word counts,

In [150]:
```
def count_words(tokens):
    word_counts = {}
    for token in tokens:
        if token not in stop_words and token not in punctuation:
            if token not in word_counts.keys():
                word_counts[token] = 1
            else:
                word_counts[token] += 1
    return word_counts

word_counts = count_words(tokens)
word_counts
```

```
'spirit': 223,
'live': 69,
'natural': 94,
'holiness': 14,
'beloved': 20,
'bodies': 10,
'sweet': 150,
```

Now we will make word frequency distribution,

In [151]:
```
def word_freq_distribution(word_counts):
    freq_dist = {}
    max_freq = max(word_counts.values())
    for word in word_counts.keys():
        freq_dist[word] = (word_counts[word]/max_freq)
    return freq_dist

freq_dist = word_freq_distribution(word_counts)
freq_dist
```

```
'shall': 0.31849912739965097,
'never': 0.11867364746945899,
'lost': 0.07504363001745201,
'lovely': 0.04799301919720768,
```

Next we will score the sentences to use in summarization.

In [152]:
```
def score_sentences(sents, freq_dist, max_len=40):
    sent_scores = {}
    for sent in sents:
        words = sent.split(' ')
        for word in words:
            if word.lower() in freq_dist.keys():
                if len(words) < max_len:
                    if sent not in sent_scores.keys():
                        sent_scores[sent] = freq_dist[word.lower()]
                    else:
                        sent_scores[sent] += freq_dist[word.lower()]
    return sent_scores

sent_scores = score_sentences(sents, freq_dist)
sent_scores
```

```
 'And so shall you picture us wandering and having constant speech, so\nth
nowledge and sweet\nfriendship of the other': 1.112565445026178,
 'And we all that time a-wander together in happy forgetfulness': 0.432809
 'And this was the way of our meeting and the growing of our acquaintance,
h the Beautiful': 1.674520069808028,
 'Now, from that time onward, evening by evening would I go a-wander along
state to the estate of Sir\nJarles': 1.3106457242582896,
 'And I had a sudden thought, and came up\nto them to see them more anigh;
Lady Mirdath': 1.5209424083769636,
 'And the two to join the dance, and danced very hearty; but had only each
to avoid the torches': 0.330715532286213,
```

Finally, we summarize the text,

In [153]:
```
def summarize(sent_scores, k):
    top_sents = Counter(sent_scores)
    summary = ''
    scores = []

    top = top_sents.most_common(k)
    for t in top:
        summary += t[0].strip()+'. '
        scores.append((t[1], t[0]))
    return summary[:-1], scores
```

In [154]:
```
summary, summary_sent_scores = summarize(sent_scores, 3)
print(summary)
```

```
Now, a great time I walked, and made a halt upon every sixth hour, and
did eat and drink, and look a little unto the monstrous towering of the
Great Redoubt; and afterwards make strong mine heart, and go forward
again. And oft I did pause, and made a watching upon the monster; but truly it
moved not, save as I have told; and I kept a great heed upon the Maid,
that she follow alway close unto my feet. And I made no great haste now to go unto that place; but went down
sudden into the bushes, and lay upon my belly, and had a new great fear
upon my spirit.
```

Now we will save the summary to a file,

```
In [155]:
f = open('F://Rahul Subjects/Unstructured Data and NLP/NLP Project Books/summary_text/summary_text.txt','w')
f.write(summary)
f.close()
```

TEXT GENERATION

Here we will read the file which we summarized in the previous section.

```
In [2]:
text=(open("F://Rahul Subjects/Unstructured Data and NLP/NLP Project Books/summary_text/summary_text.txt").read())
text=text.lower()
```

Now we need to create word mappings where we map all the distinct words to a number, so that we can build model since they can only understand numbers.

```
In [3]:
characters = sorted(list(set(text)))
n_to_char = {n:char for n, char in enumerate(characters)}
char_to_n = {char:n for n, char in enumerate(characters)}
```

Now we need to use our data to predict the next words.

```
In [5]:
X = []
Y = []
length = len(text)
seq_length = 100
for i in range(0, length-seq_length, 1):
    sequence = text[i:i + seq_length]
    label =text[i + seq_length]
    X.append([char_to_n[char] for char in sequence])
    Y.append(char_to_n[label])
```

Here we need to set a sequence length to predict the word considering that length (just like an N-gram model.)

Now we need to shape the data to send into the model for training.

```
In [6]:
X_modified = np.reshape(X, (len(X), seq_length, 1))
X_modified = X_modified / float(len(characters))
Y_modified = np_utils.to_categorical(Y)
```

Now comes our model,

```
In [8]:
model = Sequential()
model.add(LSTM(700, input_shape=(X_modified.shape[1], X_modified.shape[2]), return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(700, return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(700))
model.add(Dropout(0.2))
model.add(Dense(Y_modified.shape[1], activation='softmax'))
model.summary()
model.compile(loss='categorical_crossentropy', optimizer='adam')
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_4 (LSTM)                (None, 100, 700)          1965600
_________________________________________________________________
dropout_4 (Dropout)          (None, 100, 700)          0
_________________________________________________________________
lstm_5 (LSTM)                (None, 100, 700)          3922800
_________________________________________________________________
dropout_5 (Dropout)          (None, 100, 700)          0
_________________________________________________________________
lstm_6 (LSTM)                (None, 700)               3922800
_________________________________________________________________
dropout_6 (Dropout)          (None, 700)               0
_________________________________________________________________
dense_2 (Dense)              (None, 28)                19628
=================================================================
Total params: 9,830,828
Trainable params: 9,830,828
Non-trainable params: 0
_________________________________________________________________
```

For starters let’s start with one epoch,

```
In [19]:
model.fit(ds, steps_per_epoch=(len(encoded_text) - sequence_length) // BATCH_SIZE, epochs=EPOCHS)
```

```
Epoch 1/30
WARNING:tensorflow:From c:\users\rahul\appdata\local\programs\python\python37\lib\site-packages\tensorflow
grad.py:1250: add_dispatch_support.<locals>.wrapper (from tensorflow.python.ops.array_ops) is deprecated a
d in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
  69/7654 [..............................] - ETA: 15:40:50 - loss: 2.9296
```

```
In [31]:
for i in range(1000):
    x = numpy.reshape(pattern, (1, len(pattern), 1))
    x = x / float(vocab_len)
    prediction = model.predict(x, verbose=0)
    index = numpy.argmax(prediction)
    result = num_to_char[index]
    seq_in = [num_to_char[value] for value in pattern]

    sys.stdout.write(result)

    pattern.append(index)
    pattern = pattern[1:len(pattern)]
```

```
re shall shall shall shall shall shall shall shall shall shall shall shall shall shall shall shall shall shall shall shall
shall shall shall shall shall shall shall shall shall shall shall shall shall shall shall shall shall shall shall shall sha
ll shall shall shall shall shall shall shall shall shall shall shall shall shall shall shall shall shall shall shall shall
shall shall shall shall shall shall shall shall shall shall shall shall shall shall shall shall shall shall shall shall sha
ll shall shall shall shall shall shall shall shall shall shall shall shall shall shall shall shall shall shall shall shall
shall shall shall shall shall shall shall shall shall shall shall shall shall shall shall shall shall shall shall shall sha
ll shall shall shall shall shall shall shall shall shall shall shall shall shall shall shall shall shall shall shall shall
shall shall shall shall shall shall shall shall shall shall shall shall shall shall shall shall shall shall shall shall sha
ll shall s
```

As we can see if we only train on single epoch the result is the above one.

But the training time for each epoch as you can see is 15 hours. So, what I did is use a pre-trained model and save the file. We get,

```
or truly I made the
way with us.

Now, presently, I perceived the rove, which did by ligttle, because, I had
come as within the rong. And truly, she
had last rearings.

And
beenly, she might strace to the
Teatres; and I knew in the memory, and acto did
no
null make; and I to met nigh arast downward
```

Now but if we use a pre-trained model we don't know if it fits our data or not. So, what I did is use a module called '**textgenrnn'.** This module can only be used in the google colab since it requires tensorflow version>2 which I don't have in my computer.

This module is used because my computer doesn't have the compuatation power to run the model.

I am gonna add the code file of that when submitting.

SIMILARITY

Now we need to see the similarity between the text generated from the model and the original text. We will use the cosine similarity to measure the similarity.

In [14]:
```
a = "F://Rahul Subjects/Unstructured Data and NLP/NLP Project Books/horror/horror10.txt.txt"
b = "F://Rahul Subjects/Unstructured Data and NLP/NLP Project Books/colaboratory_gentext_20200516_234554.txt"
a = open(a, 'r').read()
b = open(b, 'r').read()
```

In [15]:
```
X_list = word_tokenize(a)
Y_list = word_tokenize(b)

# sw contains the list of stopwords
sw = stopwords.words('english')
l1 =[];l2 =[]

# remove stop words from string
X_set = {w for w in X_list if not w in sw}
Y_set = {w for w in Y_list if not w in sw}

# form a set containing keywords of both strings
rvector = X_set.union(Y_set)
for w in rvector:
    if w in X_set: l1.append(1) # create a vector
    else: l1.append(0)
    if w in Y_set: l2.append(1)
    else: l2.append(0)
c = 0

# cosine formula
for i in range(len(rvector)):
        c+= l1[i]*l2[i]
cosine = c / float((sum(l1)*sum(l2))**0.5)
print("similarity: ", cosine)
```

```
similarity:  0.4125607579337458
```

Even though the similarity is too less we can say that it is good for a model built with minimum data (only a single file).

Almost forgot to measure the similarity between the crime and horror genre books.

In [94]:

```
X_list = word_tokenize(X)  #crime genre
Y_list = word_tokenize(Y)  #horror genre

list1 = []
list2 = []

#removing stop words
X_set = {word for word in X_list if not word in stop_words}
Y_set = {word for word in Y_list if not word in stop_words}

# form a set containing keywords of both strings
key_words = X_set.union(Y_set)
for word in key_words:
    if word in X_set: list1.append(1) # create a vector
    else: list1.append(0)
    if word in Y_set: list2.append(1)
    else: list2.append(0)
c = 0

# cosine formula
for i in range(len(key_words)):
        c += list1[i]*list2[i]
cosine = c / float((sum(list1)*sum(list2))**0.5)
print("similarity: ", cosine)
```

```
similarity:  0.5248169062997158
```

Even though they are not that similar it is a significant number.

4. CONCLUSION

1. I have created a model to classify the text into genre.
2. Summarized the text to send into the model.
3. Sent the summarized text into the model to build the story from a small summary.
4. Compared the generated text with the original text.

5. FUTURE STUDY

- We can use these methods to classify more genres and summarize them and send them to a model specific to a genre for text generation. This way we can generate text more similar to that genre.
- Also, we can use the text generation for auto generation of text while writing a document or a blog etc.,
- We can use the obtained models to write stories like Harry Potter which sell well.

6. REFERENCES

- Geeksforgeeks.com (cosine similarity)
- Analyticsvidhya.com (text generation)
- Kdnuggets.com (text summarization)
- Gutenberg.org (data)