Massive Data Storage & Retrieval Final Project Report

Title: Text & Author Analysis

Name: Eunbin Ko

Section 1. Map Reduce analysis

Data Collection

```
/* The books are collected from “Project Gutenberg” website. They are free and available in text files.
The text files are edited manually to contain only the content of the book, since they have some unneccesary
information that is not useful for analysis.

There are three books for each of the author.

Jonathan Swift's books are:
"A modest proposal", "Gulliver's Travels into Several Remote Nations of the World",  "The Tale of a Tub and
The History of Martin"

Jane Austen's books are:
"Pride and Prejudice", "Emma", "Sense and Sensibility"

Mary Shelly's books are:
"Frankenstein", "The Last Man", "Tales and Stories"

By anlayzing their text with Map Reduce, the top most frequent words will be shown. Later in the section
2 of the project the classification of the each text's author will be shown.

For this first section, let's see how the word usage is similar or different for each author by looking
at the top word frequency for each of the books. */
```

Import packages

```
import sqlContext.implicits._
import org.apache.spark.mllib.feature.{Word2Vec, Word2VecModel}
```

```
import sqlContext.implicits._
import org.apache.spark.mllib.feature.{Word2Vec, Word2VecModel}
```

Text processing with MapReduce

```
/* By creating a MapReduce function, it is convenient to import any the textfile and then compute their
word frequency. While creating this MapReduce function, all the natural language processing part has been
included since the book textfile needs to be processed.

First, all the punctuations in the text will be removed. Then remove the digits, and trim the text since
there will be empty spaces.And and then change the words into lowercase. Then split the word using regex then
filter the stopwords where I found it from website “https://www.ranks.nl/stopwords”. Now, the flatMap
function can be applied for the MapReduce job. First map each word with 1, and then reduce them by key
and then the MapReduce will be done.

I have save the data into the Dataframe so that it is easier to look at. After putting MapReduce into
dataframe, there will be a empty row that has been reduced. So removed the empty word as well as single
character since they are not meaningful. Then renamed the second column with "count" since it gives the
word frequency. */
```

MapReduce function

```
def MapReduce(text:String) = {
    val book = sc.textFile(text)
    val stopWords = sc.textFile("/Users/eunbinko/documents/MassiveData/stopwords.txt")
    val words = book.map(_.replaceAll(
```

```
                   """[\p{Punct}&&[^.]]""", "")
                   .replaceAll("\\d", "")
                   .trim
                   .toLowerCase)
                   .filter(!_.isEmpty)
                   .flatMap(line => line.split("\\W"))
                   .subtract(stopWords)
                   .flatMap(_.split(" "))
                   .map((_, 1))
                   .reduceByKey(_ + _)
                   .toDS()
                   .filter(($"_1" =!= ""))
                   .filter(length($"_1") > 2)
                   .withColumnRenamed("_2", "count")
    words
}
```

```
MapReduce: (text: String)org.apache.spark.sql.DataFrame
```

Import text within MapReduce function

```
// create each book's MapReduce to see the top frequency words in each book

val swift_1 = MapReduce("/Users/eunbinko/documents/MassiveData/books/Swift_1.txt")
val swift_2 = MapReduce("/Users/eunbinko/documents/MassiveData/books/Swift_2.txt")
val swift_3 = MapReduce("/Users/eunbinko/documents/MassiveData/books/Swift_3.txt")
val austen_1 = MapReduce("/Users/eunbinko/documents/MassiveData/books/Austen_1.txt")
val austen_2 = MapReduce("/Users/eunbinko/documents/MassiveData/books/Austen_2.txt")
val austen_3 = MapReduce("/Users/eunbinko/documents/MassiveData/books/Austen_3.txt")
val shelley_1 = MapReduce("/Users/eunbinko/documents/MassiveData/books/Shelley_1.txt")
val shelley_2 = MapReduce("/Users/eunbinko/documents/MassiveData/books/Shelley_2.txt")
val shelley_3 = MapReduce("/Users/eunbinko/documents/MassiveData/books/Shelley_3.txt")
```

```
swift_1: org.apache.spark.sql.DataFrame = [_1: string, count: int]
swift_2: org.apache.spark.sql.DataFrame = [_1: string, count: int]
swift_3: org.apache.spark.sql.DataFrame = [_1: string, count: int]
austen_1: org.apache.spark.sql.DataFrame = [_1: string, count: int]
austen_2: org.apache.spark.sql.DataFrame = [_1: string, count: int]
austen_3: org.apache.spark.sql.DataFrame = [_1: string, count: int]
shelley_1: org.apache.spark.sql.DataFrame = [_1: string, count: int]
shelley_2: org.apache.spark.sql.DataFrame = [_1: string, count: int]
shelley_3: org.apache.spark.sql.DataFrame = [_1: string, count: int]
```

Change column name and show the top 10 most frequently used words in each book

```
// Change column name for Jonathan Swift's first book

swift_1.withColumnRenamed("_1", "swift_1")
       .orderBy($"count".desc).show(10)
```

```
+--------+-----+
| swift_1|count|
+--------+-----+
|    will|   36|
|children|   18|
|     one|   15|
|thousand|   15|
| kingdom|   15|
|    upon|   13|
| country|   12|
|  number|   11|
|     may|   11|
|   great|   10|
+--------+-----+
only showing top 10 rows
```

```
// Change column name for Jonathan Swift's second book

swift_2.withColumnRenamed("_1", "swift_2")
       .orderBy($"count".desc).show(10)
```

```
+-------+-----+
|swift_2|count|
+-------+-----+
|   upon|  383|
|  great|  285|
|    one|  273|
```

```
|    two|  249|
|   made|  223|
|   much|  205|
|country|  199|
|several|  176|
|   time|  165|
|  three|  163|
+-------+-----+
only showing top 10 rows
```

```
// Change column name for Jonathan Swift's third book

swift_3.withColumnRenamed("_1", "swift_3")
       .orderBy($"count".desc).show(10)
```

```
+-------+-----+
|swift_3|count|
+-------+-----+
|   upon|  220|
|   will|  185|
|  great|  154|
|    one|  116|
|   much|   98|
|    now|   90|
|  peter|   89|
|    may|   83|
|   well|   83|
|certain|   83|
+-------+-----+
only showing top 10 rows
```

```
// Change column name for Jane Austen's first book

austen_1.withColumnRenamed("_1", "austen_1")
       .orderBy($"count".desc).show(10)
```

```
+--------+-----+
|austen_1|count|
+--------+-----+
|    emma|  770|
|     mrs|  688|
|    miss|  589|
|    must|  567|
|    will|  555|
|    said|  481|
|    much|  477|
|     one|  426|
|   every|  423|
| harriet|  403|
+--------+-----+
only showing top 10 rows
```

```
// Change column name for Jane Austen's second book

austen_2.withColumnRenamed("_1", "austen_2")
       .orderBy($"count".desc).show(10)
```

```
+---------+-----+
| austen_2|count|
+---------+-----+
|elizabeth|  595|
|     will|  411|
|     said|  401|
|    darcy|  371|
|      mrs|  343|
|     much|  327|
|     must|  308|
|   bennet|  294|
|     miss|  283|
|      one|  265|
+---------+-----+
```

```
only showing top 10 rows
```

```
// Change column name for Jane Austen's thrid book

austen_3.withColumnRenamed("_1", "austen_3")
       .orderBy($"count".desc).show(10)
```

```
+--------+-----+
|austen_3|count|
+--------+-----+
|  elinor|  616|
|     mrs|  529|
|marianne|  489|
|    said|  388|
|   every|  374|
|    will|  353|
|     one|  317|
|    much|  287|
|    must|  282|
|    time|  237|
+--------+-----+
only showing top 10 rows
```

```
// Change column name for Mary Shelley's first book

shelley_1.withColumnRenamed("_1", "shelley_1")
       .orderBy($"count".desc).show(10)
```

```
+---------+-----+
|shelley_1|count|
+---------+-----+
|      one|  206|
|     will|  194|
|      now|  155|
|      yet|  152|
|      man|  136|
|   father|  134|
|     upon|  126|
|     life|  115|
|    every|  109|
|    might|  108|
+---------+-----+
only showing top 10 rows
```

```
// Change column name for Mary Shelley's second book

shelley_2.withColumnRenamed("_1", "shelley_2")
       .orderBy($"count".desc).show(10)
```

```
+---------+-----+
|shelley_2|count|
+---------+-----+
|      one|  486|
|      now|  458|
|     will|  358|
|  raymond|  334|
|     life|  307|
|   adrian|  285|
|      yet|  275|
|     even|  275|
|    heart|  268|
|     love|  262|
+---------+-----+
only showing top 10 rows
```

-

```
// Change column name for Mary Shelley's thrid book
```

```
shelley_3.withColumnRenamed("_1", "shelley_3")
       .orderBy($"count".desc).show(10)
```

```
+---------+-----+
|shelley_3|count|
+---------+-----+
|      one|  385|
|      now|  302|
|     will|  255|
|     said|  240|
|    heart|  229|
|      yet|  211|
|   father|  191|
|     upon|  176|
|     love|  174|
|     life|  160|
+---------+-----+
only showing top 10 rows
```

Conclusion

READY

```
/* Top 10 words of each book has been shown using the dataframe. It is interesting that for the same author
their word usage are very similar though the content of the books are very different. Looking at the top 10
words of Jonathan Swift's book, it is assumable that this author likes to use some words that describes the
with time, such as  "upon", "will" and "time". Also, Jonathan Swift uses numbers frequently since "one",
"two", and "three" are showing as the most frequent words in his books.

Similary, for Mary Shelley, her books contain similar words with Jonathan Swift's. She uses some words that
describing with time too, such as "will", "yet", "upon", and "now". Interestingly, the very most frequent
word for all of Mary Shelley's books are all "one".

In contrast, the top frequently used words for Jane Austen's books are different. She uses a lot of pronoun
in her books, such as names. All of her books' first top frequently used words are seem to be the name of the
protagonist of the novel.

MayReduce function helped to analyze the each author's writing style in terms of the word usage. In the
next section of this project, Neural Network is applied to classify the texts from these books for the authors.
*/
```

Section 2. Model Classification for different authors' books

This section of the project will use the LSTM Neural Network to classify the authors using their books.

First, import the necessary packages to use.

In [27]:
```
import pandas as pd
import json
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Dense, Flatten, LSTM, Conv1D, MaxPooling1D, Dropout, Activation
from keras.layers.embeddings import Embedding
import nltk
import string
import numpy as np
import pandas as pd
from nltk.corpus import stopwords
from sklearn.model_selection import train_test_split
import re
from nltk.stem.snowball import SnowballStemmer
from keras.constraints import max_norm
import tensorflow as tf
```

Data Collection

Three authors of the books are: Jonathan Swift, Jane Austen, and Mary Shelley. The three popular books are collected for each of the authors, so there are total of 9 books in the dataset. For the ease of model fitting, each input unit will be the each paragraph of the book.

Below is the function of generating dataframe for each of the authors.

In [3]:
```
# import text file and create dataframe
def Swift(textfile):
    # textfile has been edited so that it contains only body of the text
    with open(textfile) as f:
        lines = f.read()
    book = lines.split("\n\n") #split by paragraph
    text = pd.Series(book, index = range(len(book)))
    author = pd.Series(['Jonathan_Swift'] * len(book), index = range(len(book)))
    df = pd.DataFrame({'author':author,
                'text':text})
    return df
```

In [4]:
```
def Austen(textfile):
    # textfile has been edited so that it contains only body of the text
    with open(textfile) as f:
        lines = f.read()
    book = lines.split("\n\n") #split by paragraph
    text = pd.Series(book, index = range(len(book)))
    author = pd.Series(['Jane_Austen'] * len(book), index = range(len(book)))
    df = pd.DataFrame({'author':author,
                'text':text})
    return df
```

In [5]:
```
def Shelley(textfile):
    # textfile has been edited so that it contains only body of the text
    with open(textfile) as f:
        lines = f.read()
    book = lines.split("\n\n") #split by paragraph
    text = pd.Series(book, index = range(len(book)))
    author = pd.Series(['Mary_Shelley'] * len(book), index = range(len(book)))
    df = pd.DataFrame({'author':author,
                'text':text})
    return df
```

```
In [30]: swift_1 = Swift('Swift_1.txt')
         swift_2 = Swift('Swift_2.txt')
         swift_3 = Swift('Swift_3.txt')
```

```
In [31]: austen_1 = Austen('Austen_1.txt')
         austen_2 = Austen('Austen_2.txt')
         austen_3 = Austen('Austen_3.txt')
```

```
In [32]: shelley_1 = Shelley('Shelley_1.txt')
         shelley_2 = Shelley('Shelley_2.txt')
         shelley_3 = Shelley('Shelley_3.txt')
```

Then, concatenate all the data for different authors into one large dataframe. There are total of 10818 rows, which means there are 10818 paragraphs in total. The example of the dataframe is shown below.

```
In [33]: df = pd.concat([swift_1,swift_2,swift_3,austen_1,austen_2,austen_3,shelley_1,shelley_2,shelley_3],ignore_inde
```

```
In [34]: df.shape
```

```
Out[34]: (10818, 2)
```

```
In [35]: df.head()
```

Out[35]:

	author	text
0	Jonathan_Swift	It is a melancholy object to those, who walk t...
1	Jonathan_Swift	I think it is agreed by all parties, that this...
2	Jonathan_Swift	But my intention is very far from being confin...
3	Jonathan_Swift	As to my own part, having turned my thoughts f...
4	Jonathan_Swift	There is likewise another great advantage in m...

Data Cleaning

After the text of the data has been stored, it needs to be processed. Below function will replace all the punctuations, symbols and also take out the stopwords. The text will all be transformed into lower case and there will be no digits.

```
In [12]: df = df.reset_index(drop=True)
         REPLACE_BY_SPACE_RE = re.compile('[/(){}\[\]\|@,;]')
         BAD_SYMBOLS_RE = re.compile('[^0-9a-z #+_]')
         STOPWORDS = set(stopwords.words('english'))

         def clean_text(text):
             """
                 text: a string

                 return: modified initial string
             """
             text = text.lower()
             text = REPLACE_BY_SPACE_RE.sub(' ', text)
             text = BAD_SYMBOLS_RE.sub('', text)
             text = text.replace('x', '')
             text = ' '.join(word for word in text.split() if word not in STOPWORDS)
             return text
```

```
In [13]: df['text'] = df['btext'].apply(clean_text)
         df['text'] = df['text'].str.replace('\d+', '')
```

Now, the below 'text' is the cleaned version of the dataframe. We can compare this dataframe with above original text. There are only useful words contain in this cleaned dataframe. Next these text will be tokenized and use word embedding for the model fitting.

In [14]:
```
df.head()
```

Out[14]:

	author	text
0	Jonathan_Swift	melancholy object walk great town travel count...
1	Jonathan_Swift	think agreed parties prodigious number ofchild...
2	Jonathan_Swift	intention far confined provide thechildren pro...
3	Jonathan_Swift	part turned thoughts many years upon thisimpor...
4	Jonathan_Swift	likewise another great advantage scheme willpr...

The maximum number of words to be used is set as 5000, and it is the most frequent words showing in the data. Max number of words in each complaint is set as 500, since one paragraph will not be too long. The embedding dimension is set to be 100.

'Tokenizer' method will split text into words are generate word vectors. The number of unique words that are in these books are 67501.

In [15]:
```
MAX_NB_WORDS = 5000
MAX_SEQUENCE_LENGTH = 500
EMBEDDING_DIM = 100
tokenizer = Tokenizer(num_words=MAX_NB_WORDS, filters='!"#$%&()*+,-./:;<=>?@[\]^_`{|}~', lower=True)
tokenizer.fit_on_texts(df['text'].values)
word_index = tokenizer.word_index
print('Found %s unique tokens.' % len(word_index))
```

```
Found 67501 unique tokens.
```

Since there are too many words, input vector needs to be padded into the maximum sequence where I set to be 500.

In [16]:
```
X = tokenizer.texts_to_sequences(df['text'].values)
X = pad_sequences(X, maxlen=MAX_SEQUENCE_LENGTH)
print('Shape of data tensor:', X.shape)
```

```
Shape of data tensor: (10818, 500)
```

The Y vector are now the names of the authors. It needs to be converted into vectors as well. 'get_dummies' function will automatically generate vector for those authors.

In [40]:
```
Y = pd.get_dummies(df['author']).values
print('Shape of label tensor:', Y.shape)
Y
```

```
Shape of label tensor: (10818, 3)
```

Out[40]:
```
array([[0, 1, 0],
       [0, 1, 0],
       [0, 1, 0],
       ...,
       [0, 0, 1],
       [0, 0, 1],
       [0, 0, 1]], dtype=uint8)
```

Now, the dataset is ready to split into training and testing sets. The size of the training set is 90% of the total dataset, and the rest of the dataset is used for testing set. The 'random_state' will set seed to these testing and training set so that everytime running this code will have the same training and testing set.

In [18]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size = 0.10, random_state = 42)
print(X_train.shape,Y_train.shape)
print(X_test.shape,Y_test.shape)
```

```
(9736, 500) (9736, 3)
(1082, 500) (1082, 3)
```

In [19]:
```
X.shape[1]
```

Out[19]:
```
500
```

Baseline model

Below is the baseline for the LSTM Neural Network model. The activation function is 'tanh' for the LSTM Network. The baseline model has added the dropout of 0.5 since LSTM Network can easily get into overfitting model. The dense unit would be 3 at the end since we have three different authors that will be classified. The activation function of Dense function is a 'softmax' function. It will be complied into the 'categorical crossentropy' since there are three categories, which are three authors.

In [20]:
```
model = Sequential()
model.add(Embedding(MAX_NB_WORDS, EMBEDDING_DIM, input_length=X.shape[1]))
model.add(LSTM(32, activation = 'tanh'))
model.add(Dropout(0.5))
model.add(Dense(3, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

epochs = 10
batch_size = 128

history = model.fit(X_train, Y_train, epochs=epochs, batch_size=batch_size,validation_split=0.1)
```

```
WARNING:tensorflow:From /anaconda3/lib/python3.7/site-packages/tensorflow/python/ops/resource_variable_ops.
py:435: colocate_with (from tensorflow.python.framework.ops) is deprecated and will be removed in a future
version.
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From /anaconda3/lib/python3.7/site-packages/tensorflow/python/ops/math_ops.py:3066: to_i
nt32 (from tensorflow.python.ops.math_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
Train on 8762 samples, validate on 974 samples
Epoch 1/10
8762/8762 [==============================] - 27s 3ms/step - loss: 0.8667 - accuracy: 0.6303 - val_loss: 0.5
422 - val_accuracy: 0.8101
Epoch 2/10
8762/8762 [==============================] - 26s 3ms/step - loss: 0.4258 - accuracy: 0.8341 - val_loss: 0.3
117 - val_accuracy: 0.8522
Epoch 3/10
8762/8762 [==============================] - 25s 3ms/step - loss: 0.2249 - accuracy: 0.9217 - val_loss: 0.2
217 - val_accuracy: 0.9168
```

In [21]:
```
accr = model.evaluate(X_test,Y_test)
print('Test set\n  Loss: {:0.3f}\n  Accuracy: {:0.3f}'.format(accr[0],accr[1]))
```

```
1082/1082 [==============================] - 1s 1ms/step
Test set
  Loss: 0.272
  Accuracy: 0.918
```

In [22]:
```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 500, 100)          500000
_________________________________________________________________
lstm_1 (LSTM)                (None, 32)                17024
_________________________________________________________________
dropout_1 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_1 (Dense)              (None, 3)                 99
=================================================================
Total params: 517,123
Trainable params: 517,123
Non-trainable params: 0
_________________________________________________________________
```

From the above result, the model is overfitting according to the accuracy of validation dataset.

LSTM Neural Network model

Here, the model constraints has been added in order to fix the overfitting. Instead of using the dropout method, this model used to specify the maximum number of the norm of kernel vector, reccurent vector, and bias vector. I used 3 as the maximun number of norm of the kernel vector, norm of reccurent vector, and norm of bias vector. If the number of norm of the vector exceed 3, then it will be dropped.

In [23]:
```
model = Sequential()
model.add(Embedding(MAX_NB_WORDS, EMBEDDING_DIM, input_length=X.shape[1]))
model.add(LSTM(32, kernel_constraint=max_norm(3), recurrent_constraint=max_norm(3),
               bias_constraint=max_norm(3)))
model.add(Dense(3, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

epochs = 6
batch_size = 128

history = model.fit(X_train, Y_train, epochs=epochs, batch_size=batch_size,validation_split=0.1)
```

```
Train on 8762 samples, validate on 974 samples
Epoch 1/6
8762/8762 [==============================] - 27s 3ms/step - loss: 0.8405 - accuracy: 0.6258 - val_loss: 0.5
465 - val_accuracy: 0.7721
Epoch 2/6
8762/8762 [==============================] - 26s 3ms/step - loss: 0.4326 - accuracy: 0.8350 - val_loss: 0.3
554 - val_accuracy: 0.8470
Epoch 3/6
8762/8762 [==============================] - 26s 3ms/step - loss: 0.2587 - accuracy: 0.8797 - val_loss: 0.2
572 - val_accuracy: 0.8881
Epoch 4/6
8762/8762 [==============================] - 25s 3ms/step - loss: 0.1535 - accuracy: 0.9482 - val_loss: 0.2
109 - val_accuracy: 0.9168
Epoch 5/6
8762/8762 [==============================] - 25s 3ms/step - loss: 0.1079 - accuracy: 0.9563 - val_loss: 0.1
956 - val_accuracy: 0.9230
Epoch 6/6
8762/8762 [==============================] - 29s 3ms/step - loss: 0.0913 - accuracy: 0.9618 - val_loss: 0.2
005 - val_accuracy: 0.9230
```

In [24]:
```
accr = model.evaluate(X_test,Y_test)
print('Test set\n  Loss: {:0.3f}\n  Accuracy: {:0.3f}'.format(accr[0],accr[1]))
```

```
1082/1082 [==============================] - 2s 1ms/step
Test set
  Loss: 0.220
  Accuracy: 0.915
```

In [25]:
```
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 500, 100)          500000
_________________________________________________________________
lstm_2 (LSTM)                (None, 32)                17024
_________________________________________________________________
dense_2 (Dense)              (None, 3)                 99
=================================================================
Total params: 517,123
Trainable params: 517,123
Non-trainable params: 0
_________________________________________________________________
```

Now, above model has the good performance without any overfitting problem. Test accuracy is 0.915, where training accuracy is 0.9618.

Confusion Matrix

After the model, look at the confusion matrix of the model to see the detailed prediction accuracy for each of the output vector.

In [26]:
```
Y_pred = model.predict_classes(X_test)
sum(Y_pred == Y_test.argmax(axis = 1))
con_mat = tf.confusion_matrix(labels=Y_test.argmax(axis = 1), predictions=Y_pred)
sess = tf.Session()
with sess.as_default():
        print(sess.run(con_mat))
```

```
WARNING:tensorflow:From /anaconda3/lib/python3.7/site-packages/tensorflow/python/ops/confusion_matrix.py:19
3: to_int64 (from tensorflow.python.ops.math_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
[[615   1  47]
 [  7  78  16]
 [ 17   4 297]]
```

Above confusion matrix is in the order of Jonathan Swift, Jane Austen, and Mary Shelley. For Jane Austen, the number of paragraphs are a lot shorter than other two authors', so that there are only 5 paragraphs has been misclassified. Jonathan Swift's paragraphs are classified more accurate than Mary Shelley's. Overall, the performance is good for all of the three author's paragraphs.

resources for LSTM model: https://towardsdatascience.com/multi-class-text-classification-with-lstm-1590bee1bd17 (https://towardsdatascience.com/multi-class-text-classification-with-lstm-1590bee1bd17)