Final Project by Donato Scarano

PART 1. DEFINE THE PROBLEM AND DATA

I. INTRODUCTION

TEMPLATE: DEEP LEARNING ON A PUBLIC DATASET

OBJECTIVE: DEVELOP A DEEP LEARNING MODEL ON A PUBLIC DATASET

WHY THIS PROJECT?

Classification problems are everywhere today:

- `Customer Behaviour Prediction`
- `Document or Media Classification`
- `Product Categorization`
- `Sentiment Analysis`
- `Fraud Detection`
- `Text Generation`

Deep Learning for Natural Language processing has many documented projects on Kaggle (www.kaggle.com) related to small databases parsed from open databases as Project Gutenberg, IMDB, Reuters.

Can we expand and generalise to a very large dataset of heterogeneous text?

Can we gain further insights and develop a better model?

Based on a text input can we do text classification and text generation?

MOTIVATIONS

- `Master Advanced Machine Learning Techniques`
- `Personal curiosity in the subject and desire to gain insights from data`
- `Data is the new oil and it has become one of the most precious commodities`
- `Gaining insights from data and put them to purpose`
- `Doing and Learning are the drivers for this project`
- `Hands-on lab approach at crunching data`
- `Learn text generation and text classification techniques`

WORKFLOW METHOD

I have used the Universal Workflow detailed in Francois Chollet, Deep Learning with Python [1].

The steps that will be undertaken are the following:

- `Define the problem and data`
- `Collect the data`
- `Choose the measure of success`
- `Choose the metrics to monitor`
- `Determine the evaluation protocol`
- `Develop a first model`
- `Regularize the model and fine tune`

II. LITERATURES REVIEW

SOURCES

Kaggle (www.kaggle.com) has been described as a loose association of geniuses who compete against each other to raise the limits of Data Science. Kaggle was founded in 2010 by Anthony Goldbloom a former data scientist with a prize competion of 1000 dollars. Since then the site has exploded with over 8 million users by 2021. I have explored many different public datasets with deep learning classification problems to get a grasp on the variety of projects undertaken by the Kaggle community and the wide variety of analysis that can be undertaken on a public dataset. Kaggle has many documented projects related to classification and a wide variety of public datasets available for free and that can be used to experiment and compete.

Francois Chollet's book Deep Learning with Python [1] is also a source of inspiration with his advance deep learning techniques described in Chapter 7.

ACM: (https://cacm.acm.org) catalogue has a wide collection of reference literature within deep learning and machine learning fields.

IEEE Xplore site: (https://ieeexplore.ieee.org/Xplore/home.jsp) is another useful site to explore books, articles, conferences and more on technology research Google Scholar: (https://scholar.google.com/schhp?hl=en) is a useful search engine to explore articles and case law on a wide variety of subjects.

University of London Summon: (https://scholar.google.com/schhp?hl=en) is also another search engine for books, e-books, articles and more.

Some of the projects examples found:

Classifying movie reviews with the IMDB Dataset

The IMDB dataset is a set of 50000 reviews from the Internet Movie Database. The reviews are split into a training and testing dataset of 25000 reviews each. Each set contains a 50% of negative and 50% of positive reviews. The dataset comes also packaged with Keras. It is a two class classification problem which is one of the most widely applied type of problems in machine learning. The movie review can be either positive or negative.

Classifying newswires with the Reuters Dataset

In this example our classification problem become more complex, in the IMDB example there were only two classes, here we have 46 different topics to which each newswire in the dataset belongs to. The dataset comes also packaged with Keras. AS with the IMDB dataset most of the analysis restricts the data to a limited number of words.

Classifying documents within the Gutenberg dataset

Project Gutenberg is the oldest digital library founded in 1971 by American writer Michael S. Hart. It is a collection of books in the public domain. All books are free and as of 2021 there were over 65000 items in the collection. The collection come packaged in the NLTK library. NLTK (Natural Language toolkit) is an open source platform for natural language processing and available in Python.

Classifying images using the MNIST dataset.

MNIST is a classic dataset in the machine learning community. Again it is a limited set of 60000 training images and 10000 test images. It was originally assembled by the National Institute of Standards and Technology (NIST) in the 80s. The dataset comes also packaged with Keras.

Can we expand and generalise to a very large dataset of heterogeneous data and obtain the same results, what further insights can we gain by developing a better model on a large dataset?

Can we do text generation using Keras and Tensorflow?

TOOLS AND TECHNIQUES

I have used Jupyter notebooks.

Python and its libraries for machine learning, deep learning, matplotlib and NLTK. Tensorflow and Keras.

I have used the Universal Workflow as described by Frederic Chollet in his book Deep Learning with Python. [1]

Its stages and method are described below:

- `DEFINE THE PROBLEM AND DATA`
- `COLLECT THE DATA`
- `CHOOSE THE MEASURE OF SUCCESS`
- `CHOOSE THE METRICS TO MONITOR`
- `DETERMINE THE EVALUATION PROTOCOL`
- `DEVELOP A FIRST MODEL`
- `REGULARIZE THE MODEL AND FINE TUNE`

Some of the techniques I had been considering for use in my project were the following:

```
Binary and Multiclass Classification
Confusion Matrix and Classifiers
Bayes' Theorem
RNN
```

Further techniques and tools were found needed along the road such as GPT-2 and Bokeh.

It is important to note that model performance and related tools are dictated by the characteristics of the data.

There is no single silver bullet that work on all the problems.

Determining the best model for a given problem is still today more an art than a science.

Some metrics (accuracy, precision and recall for example) provided us with a way to qualify the model but testing and fine tuning has been a constant necessity along the way and at each stage to find the best characteristics and features.

STUDIES AND RESEARCH

There is a wide variety of studies that have tried and address the deep learning problem.

As mentioned above the lack of a single model appropriate for all datasets has given the rise to a large number of deep learning techniques.

Some of the most commonly used which I had been considering for my project were the following:

Linear Classifiers

Used to find a linear combination of features that describe or separate two or more classes. Fisher's Linear Discriminant and LDA (Linear Discriminant Analysis) are sometimes used interchangeably but in reality LDA is a generalization of Fisher's Linear Discriminant first developed by Sir Ronald Fisher in 1936 [2]

SVM(Support Vector Machines)

It is a supervised learning model used for classification and regression analysis. It was first developed at At&T Bell Laboratories by Vladimir Vapnik and colleagues (Boser et al., 1992, Guyon et al., 1993, Vapnik et al., 1997) [3]

K-Nearest Neighbor

Also known as k-NN is a classification method developed by Evelyn Fix and Joseph Hodges in 1951. [4] The input is the k closest training examples in a dataset. The output is a class membership, the object is assigned to the class most common amongst its k nearest Neighbors.

Decision Trees

Decision trees where the target variable can take discrete set of values are called Classification Trees they are tree structures where the leaves represent class labels and the branches represent a combination of features that output those class labels. Decision trees where instead the target variable can take continuous values are called Regression Trees. Decision Trees are one of the most popular algorithms since they are very simple and clear to understand [5] I was planning to use the implementation available with scikit-learn (https://scikitlearn.org/stable/index.html) library in Python.

Neural Networks

Inspired by our brain functioning, Neural Networks (NN) or Artificial Neural Network (ANN) are a collection of connected nodes called artificial neurons. Each connection works like a synapse in our brain sending signals to other neurons. The system learn by example without being programmed with specific rules.

Warren McCulloch and Walter Pitts in 1943 were the first to deal with the subject by creating a computational model for neural networks. The first pattern recognizer to achieve human levels or superior was built by Ciresan and his colleagues [6] LSTM was introduced by Hochreiter & Schmidhuber in 1997 [7].

References:

```
[1] François Chollet. 2021. Deep Learning with Python. Simon and Schuster, 2021
[2] Fisher, Ronald A. "The use of multiple measurements in taxonomic problems." Annals of eugenics 7.2 (1936
[3] Cortes, Corinna, and Vladimir Vapnik. "Support-vector networks." Machine learning 20.3 (1995): 273-297.
[4] Fix, Evelyn, and Joseph Lawson Hodges. "Discriminatory analysis. Nonparametric discrimination: Consisten
[5] Wu, Xindong, et al. "Top 10 algorithms in data mining." Knowledge and information systems 14.1 (2008): 1
[6] Ciregan, Dan, Ueli Meier, and Jürgen Schmidhuber. "Multi-column deep neural networks for image classific
[7] Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." Neural computation 9.8 (1997): 1735-
```

III. DESIGN

DOMAIN AND USERS

The domain of the project is documents (it can be media, images, text) but in this case I have focused on text models. I have used the Gutenberg Dataset which come packaged with NLTK library.

The users that I am targeting are personas concerned with extracting value out of those datasets but more in general everyone interested in machine learning and its advanced techniques.

I wanted to experiment on the dataset to see the difference of context and results using different models and what we can learn from it. Develop a deep learning model on a public dataset that is able to generalise to different datasets and situations and develop the capability to do text generation.

As humans we are able to abstract and reason on hypothetical models to expand our mental capabilities we can adapt to new situations with little or no data at all. Francois Chollet in his book Deep Learning with Python call this extreme generalisation.

Deep learning in contrast is a local generalization where an input/output model can fail if the input change from previous experience. Deep learning can provide huge business value to a company, examples of this are the recent development of speech recognition, smart assistants, image classification, translation and text generation. Deep learning is extremely valuable in regards to the insights it can obtain but still unable of abstraction and reasoning.

Machine learning models can be defined as learnable programs. So far we can only learn programs related to a narrow subset, the next step would be to learn any program in a modular and reusable way that can be used to tackle any challenge. This would be the future of deep learning and the reason I want to investigate how to use models in different contexts.

DESIGN

The design starting point is based on the universal machine learning workflow.

I have used Jupyter notebooks to structure the workflow and have a clear outline easily interpretable for all the users interested to the analysis.

The notebook is structured in the following sections:

1. `DEFINE THE PROBLEM AND DATA`
2. `COLLECT THE DATA`
3. `CHOOSE THE MEASURE OF SUCCESS`
4. `CHOOSE THE METRICS TO MONITOR`
5. `DETERMINE THE EVALUATION PROTOCOL`
6. `DEVELOP A FIRST MODEL`
7. `REGULARIZE, TEST THE MODEL AND FINE TUNE`

I have added extensive notes at each level to be safe that every user is able to follow and understand each step of the process. As mentioned above the project is catering to both personas interested to the subject of the analysis and to general machine learning public. We have to be cautious in considering that not necessarily all of them will understand machine learning processes and are familiar with Python therefore we will adapt and incorporate plain language commenting which will allow us to reach the widest public possible and use of visual aids where possible to easily interpret the results.

TECHNOLOGIES AND METHODS

The main technologies, method and tools used are the following:

- `Jupyter Notebook`
- `Tensorflow`
- `Tensorboard`
- `Keras`
- `GPT-2`
- `TF-IDF`
- `Gensim`
- `Word2Vec`
- `Bokeh`
- `RNN/LSTM`
- `Python`
- `Matplotlib Library`
- `sklearn Library`
- `Kaggle.com`
- `Github`
- `Universal Workflow Model for Deep Learning`

PROJECT PLAN

I am using Microsoft Project to keep cadence and outline the different steps to undertake with relevant prerequisites and timelines.

TESTING PLAN

As mentioned above the notebook will be structured in the following sections:

• DEFINE THE PROBLEM AND DATA

• COLLECT THE DATA

• CHOOSE THE MEASURE OF SUCCESS

• CHOOSE THE METRICS TO MONITOR

• DETERMINE THE EVALUATION PROTOCOL

• DEVELOP A FIRST MODEL

• REGULARIZE, TEST THE MODEL AND FINE TUNE

A lot of fine tuning and testing has been employed to find the best possible model. I have enacted some basic unit testing by using the unittest library in Python and tf.test in Tensorflow.

A lot of the testing I have conducted though is human centred rather than code centred since the final objective is for anyone interested in the subject to gain insights from the data, I want to be safe that format, model and design chosen is human friendly and able to reach the set objectives.

Insights mean little if the intended recipient does not get it, therefore an illustrative and educational approach must be pursued.

For this I have been using different channels to interact with colleagues and friends.

Further details are available in the Evaluation section at Chapter 5.

PART 2. COLLECT THE DATA

```
#import the Gutenberg dataset from the NLTK corpus and print the books contained
import nltk
nltk.download('gutenberg')
from nltk.corpus import gutenberg
gutenberg.fileids()
```

```
[nltk_data] Downloading package gutenberg to /root/nltk_data...
[nltk_data]   Unzipping corpora/gutenberg.zip.
['austen-emma.txt',
 'austen-persuasion.txt',
 'austen-sense.txt',
 'bible-kjv.txt',
 'blake-poems.txt',
 'bryant-stories.txt',
 'burgess-busterbrown.txt',
 'carroll-alice.txt',
 'chesterton-ball.txt',
 'chesterton-brown.txt',
 'chesterton-thursday.txt',
 'edgeworth-parents.txt',
 'melville-moby_dick.txt',
 'milton-paradise.txt',
 'shakespeare-caesar.txt',
 'shakespeare-hamlet.txt',
 'shakespeare-macbeth.txt',
 'whitman-leaves.txt']
```

```
#for each book of the collection we collect some basic info such as number of characters, words, sen
#we also print the lexical diversity for the text in question.
import pandas as pd
nltk.download('punkt')
for fileid in gutenberg.fileids():
    num_chars = len(gutenberg.raw(fileid))
    num_words = len(gutenberg.words(fileid))
    num_sents = len(gutenberg.sents(fileid))
    num_vocab = len(set(w.lower() for w in gutenberg.words(fileid)))
    lex_diver = len(set(fileid))/len(fileid)
    print(fileid, "Nr. Characters:", num_chars, "Nr. Words:", num_words, "Nr. Sentences:", num_sents
    print(fileid, "Average Word Length: ", round(num_chars/num_words), "Average Sentence Length: ",
    print(fileid, "Lexical Diversity: ", lex_diver)
```

```
     [nltk_data] Downloading package punkt to /root/nltk_data...
     [nltk_data]   Unzipping tokenizers/punkt.zip.
     austen-emma.txt Nr. Characters: 887071 Nr. Words: 192427 Nr. Sentences: 7752 Nr. Vocabularies:
     austen-emma.txt Average Word Length:  5 Average Sentence Length:  25 Vocabulary Occurences:  2
     austen-emma.txt Lexical Diversity:  0.6666666666666666
     austen-persuasion.txt Nr. Characters: 466292 Nr. Words: 98171 Nr. Sentences: 3747 Nr. Vocabula
     austen-persuasion.txt Average Word Length:  5 Average Sentence Length:  26 Vocabulary Occurenc
     austen-persuasion.txt Lexical Diversity:  0.6190476190476191
     austen-sense.txt Nr. Characters: 673022 Nr. Words: 141576 Nr. Sentences: 4999 Nr. Vocabularies
     austen-sense.txt Average Word Length:  5 Average Sentence Length:  28 Vocabulary Occurences:
     austen-sense.txt Lexical Diversity:  0.5625
     bible-kjv.txt Nr. Characters: 4332554 Nr. Words: 1010654 Nr. Sentences: 30103 Nr. Vocabularies
     bible-kjv.txt Average Word Length:  4 Average Sentence Length:  34 Vocabulary Occurences:  79
     bible-kjv.txt Lexical Diversity:  0.8461538461538461
     blake-poems.txt Nr. Characters: 38153 Nr. Words: 8354 Nr. Sentences: 438 Nr. Vocabularies: 153
     blake-poems.txt Average Word Length:  5 Average Sentence Length:  19 Vocabulary Occurences:  5
     blake-poems.txt Lexical Diversity:  0.8666666666666667
     bryant-stories.txt Nr. Characters: 249439 Nr. Words: 55563 Nr. Sentences: 2863 Nr. Vocabularie
     bryant-stories.txt Average Word Length:  4 Average Sentence Length:  19 Vocabulary Occurences:
     bryant-stories.txt Lexical Diversity:  0.7222222222222222
     burgess-busterbrown.txt Nr. Characters: 84663 Nr. Words: 18963 Nr. Sentences: 1054 Nr. Vocabul
     burgess-busterbrown.txt Average Word Length:  4 Average Sentence Length:  18 Vocabulary Occure
     burgess-busterbrown.txt Lexical Diversity:  0.5652173913043478
     carroll-alice.txt Nr. Characters: 144395 Nr. Words: 34110 Nr. Sentences: 1703 Nr. Vocabularies
     carroll-alice.txt Average Word Length:  4 Average Sentence Length:  20 Vocabulary Occurences:
     carroll-alice.txt Lexical Diversity:  0.6470588235294118
     chesterton-ball.txt Nr. Characters: 457450 Nr. Words: 96996 Nr. Sentences: 4779 Nr. Vocabulari
     chesterton-ball.txt Average Word Length:  5 Average Sentence Length:  20 Vocabulary Occurences
     chesterton-ball.txt Lexical Diversity:  0.7368421052631579
     chesterton-brown.txt Nr. Characters: 406629 Nr. Words: 86063 Nr. Sentences: 3806 Nr. Vocabular
     chesterton-brown.txt Average Word Length:  5 Average Sentence Length:  23 Vocabulary Occurence
     chesterton-brown.txt Lexical Diversity:  0.65
     chesterton-thursday.txt Nr. Characters: 320525 Nr. Words: 69213 Nr. Sentences: 3742 Nr. Vocabu
     chesterton-thursday.txt Average Word Length:  5 Average Sentence Length:  18 Vocabulary Occure
     chesterton-thursday.txt Lexical Diversity:  0.6521739130434783
     edgeworth-parents.txt Nr. Characters: 935158 Nr. Words: 210663 Nr. Sentences: 10230 Nr. Vocabu
     edgeworth-parents.txt Average Word Length:  4 Average Sentence Length:  21 Vocabulary Occurenc
     edgeworth-parents.txt Lexical Diversity:  0.7142857142857143
     melville-moby_dick.txt Nr. Characters: 1242990 Nr. Words: 260819 Nr. Sentences: 10059 Nr. Voca
     melville-moby_dick.txt Average Word Length:  5 Average Sentence Length:  26 Vocabulary Occuren
     melville-moby_dick.txt Lexical Diversity:  0.7272727272727273
     milton-paradise.txt Nr. Characters: 468220 Nr. Words: 96825 Nr. Sentences: 1851 Nr. Vocabulari
```

```
milton-paradise.txt Average Word Length:  5 Average Sentence Length:  52 Vocabulary Occurences
milton-paradise.txt Lexical Diversity:  0.7894736842105263
shakespeare-caesar.txt Nr. Characters: 112310 Nr. Words: 25833 Nr. Sentences: 2163 Nr. Vocabul
shakespeare-caesar.txt Average Word Length:  4 Average Sentence Length:  12 Vocabulary Occuren
shakespeare-caesar.txt Lexical Diversity:  0.5454545454545454
shakespeare-hamlet.txt Nr. Characters: 162881 Nr. Words: 37360 Nr. Sentences: 3106 Nr. Vocabul
shakespeare-hamlet.txt Average Word Length:  4 Average Sentence Length:  12 Vocabulary Occuren
shakespeare-hamlet.txt Lexical Diversity:  0.5909090909090909
shakespeare-macbeth.txt Nr. Characters: 100351 Nr. Words: 23140 Nr. Sentences: 1907 Nr. Vocabu
shakespeare-macbeth.txt Average Word Length:  4 Average Sentence Length:  12 Vocabulary Occure
shakespeare-macbeth.txt Lexical Diversity:  0.6086956521739131
whitman-leaves.txt Nr. Characters: 711215 Nr. Words: 154883 Nr. Sentences: 4250 Nr. Vocabulari
whitman-leaves.txt Average Word Length:  5 Average Sentence Length:  36 Vocabulary Occurences:
whitman-leaves.txt Lexical Diversity:  0.7777777777777778
```

Let us start delving into the text processing.

SENTENCE SEGMENTATION

```
# we tokenize the sentences of the book
msents = gutenberg.sents('shakespeare-macbeth.txt')
msents
```

```
[['[', 'The', 'Tragedie', 'of', 'Macbeth', 'by', 'William', 'Shakespeare', '1603', ']'], ['Act
```

```
# let us print some of the sentences
for m in msents:
    print(m[0:5])
```

```
['Macb', '.']
['Why', 'should', 'I', 'play', 'the']
['whiles', 'I', 'see', 'liues', ',']
['Enter', 'Macduffe', '.']
['Macd', '.']
['Turne', 'Hell', '-', 'hound', ',']
['Macb', '.']
['Of', 'all', 'men', 'else', 'I']
['Macd', '.']
['I', 'haue', 'no', 'words', ',']
['Fight', ':', 'Alarum']
['Macb', '.']
['Thou', 'loosest', 'labour', 'As', 'easie']
['Macd', '.']
['Dispaire', 'thy', 'Charme', ',', 'And']
['Macb', '.']
['Accursed', 'be', 'that', 'tongue', 'that']
['Ile', 'not', 'fight', 'with', 'thee']
['Macd', '.']
['Then', 'yeeld', 'thee', 'Coward', ',']
['Wee', "'", 'l', 'haue', 'thee']
['Macb', '.']
['I', 'will', 'not', 'yeeld', 'To']
['Though', 'Byrnane', 'wood', 'be', 'come']
['Before', 'my', 'body', ',', 'I']
['Exeunt', '.']
```

```
['fighting', '.']
['Alarums', '.']
['Enter', 'Fighting', ',', 'and', 'Macbeth']
['Retreat', ',', 'and', 'Flourish', '.']
['Enter', 'with', 'Drumme', 'and', 'Colours']
['Mal', '.']
['I', 'would', 'the', 'Friends', 'we']
['Sey', '.']
['Some', 'must', 'go', 'off', ':']
['Mal', '.']
['Macduffe', 'is', 'missing', ',', 'and']
['Rosse', '.']
['Your', 'son', 'my', 'Lord', ',']
['Sey', '.']
['Then', 'he', 'is', 'dead', '?']
['Rosse', '.']
['I', ',', 'and', 'brought', 'off']
['Sey', '.']
['Had', 'he', 'his', 'hurts', 'before']
['Rosse', '.']
['I', ',', 'on', 'the', 'Front']
['Sey', '.']
['Why', 'then', ',', 'Gods', 'Soldier']
['Mal', '.']
['Hee', "'", 's', 'worth', 'more']
['Sey', '.']
['He', "'", 's', 'worth', 'no']
['Here', 'comes', 'newer', 'comfort', '.']
['Enter', 'Macduffe', ',', 'with', 'Macbeths']
['Macd', '.']
['Haile', 'King', ',', 'for', 'so']
['Behold', 'where', 'stands', 'Th', "'"]
```

```
# we have 1907 sentences
len(msents)
```

```
1907
```

WORD TOKENIZATION

```
#we subdivide the text into its component words (Tokens)
mwords = gutenberg.words('shakespeare-macbeth.txt')
print("Total Words: ", len(mwords))
print("Unique Words: ", len(set(mwords)))
```

```
Total Words:  23140
Unique Words:  4017
```

TEXT NORMALIZATION

Both stemming and lemmatization are text normalizing procedures

Stemming is the process of reducing inflection toward their root forms for example play is the root base word for "playing", "played" etc. Stemming slice "playing" removing the suffix "-ing"

Lemmatization is the process of switching any word to its base root It is similar to stemming but has a linguistically principled analysis and therefore identify the correct lemma of each word. For example "troubled" is solved by lemmatization with "trouble" identifying the correct form Stemming would have solved "troubled" by slicing "-ed" with a base root "troubl"

Stemming is faster but less accurate chopping words without context can create non existing words it can be useful though in a context where the meaning of the word is not important such as Spam Detection.

Lemmatization is slower but more accurate and give meaningful words it is used in contexts such as Queries and Answers where the meaning of the word is important.

We use here mltk PorterStemmer and WordnetLemmatizer which use Wordnet's built-in morphy functions and check the word towards WordNet.

```
import nltk
from nltk.stem import PorterStemmer, WordNetLemmatizer
from nltk import wordnet
nltk.download('wordnet')
stemmer = PorterStemmer()
wnl = WordNetLemmatizer()

# stemming
swords = [stemmer.stem(word) for word in mwords]
swords
```

```
      'self',
      'haue',
      'all',
      'the',
      'other',
      ',',
      'and',
      'the',
      'veri',
      'port',
      'they',
      'blow',
      ',',
      'all',
      'the',
      'quarter',
      'that',
      'they',
      'know',
      ',',
      'I',
      "'",
      'th',
      "'",
      'ship',
      '-',
      'man',
      'card',
      '.',
      'ile',
      'dreyn',
      'him',
      'drie',
      'as'
```

```
        'as',
        'hay',
        ':',
        'sleep',
        'shall',
        'neyther',
        'night',
        'nor',
        'day',
        'hang',
        'vpon',
        'hi',
        'pent',
        '-',
        'hous',
        'lid',
        ':',
        'He',
        'shall',
        'liue',
        'a',
        'man',
        'forbid',
        ':',
        'weari',
        'seu'.
```

```
#lemmatization
nltk.download('omw-1.4')
lwords = [wnl.lemmatize(word, pos = 'v') for word in mwords]
lwords
```

```
        'selfe',
        'haue',
        'all',
        'the',
        'other',
        ',',
        'And',
        'the',
        'very',
        'Ports',
        'they',
        'blow',
        ',',
        'All',
        'the',
        'Quarters',
        'that',
        'they',
        'know',
        ',',
        'I',
        "'",
        'th',
        "'",
        'Ship',
        '-',
        'man',
        'Card',
        '.',
        'Ile',
        'dreyne'
```

```
     'him',
     'drie',
     'as',
     'Hay',
     ':',
     'Sleepe',
     'shall',
     'neyther',
     'Night',
     'nor',
     'Day',
     'Hang',
     'vpon',
     'his',
     'Pent',
     '-',
     'house',
     'Lid',
     ':',
     'He',
     'shall',
     'liue',
     'a',
     'man',
     'forbid',
     ':',
     'Wearie',
```

STOP WORDS

In a narrative text such as Macbeth we have a lot of high frequency words (such as: and, the, of, as, ...) that bring little lexical value. The issue is that they can add noise to our tasks.

We can preprocess and filter them, NLTK has built-in lists for many languages that can help in the process.

```
#import stopwords list and print the english version
from nltk.corpus import stopwords

nltk.download('stopwords')
stopwords = nltk.corpus.stopwords.words("english")
print(stopwords)
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'
```

```
# filter the Macbeth text and remove punctuation
import string
mwords2 = [''.join(letter for letter in word if letter not in string.punctuation) for word in mwords

while '' in mwords2:
  mwords2.remove('')

print(mwords2)
```

```
['The', 'Tragedie', 'of', 'Macbeth', 'by', 'William', 'Shakespeare', '1603', 'Actus', 'Primus'
```

```
#filter the Macbeth text and remove stopwords
mwords2 = [w for w in mwords2 if not w in stopwords]
print(mwords2)
```

```
['The', 'Tragedie', 'Macbeth', 'William', 'Shakespeare', '1603', 'Actus', 'Primus', 'Scoena',
```

```
print("Total Words: ", len(mwords2))
print("Unique Words: ", len(set(mwords2)))
```

```
Total Words:  12049
Unique Words:  3898
```

FREQUENCY DISTRIBUTIONS

```
#frequency distribution of words in the text
f = nltk.FreqDist(mwords2)
```

```
# top 10 frequent words
f.most_common(10)
```

```
[('I', 333),
 ('And', 170),
 ('Macb', 137),
 ('The', 118),
 ('haue', 117),
 ('Enter', 80),
 ('That', 80),
 ('What', 74),
 ('To', 73),
 ('thou', 63)]
```

```
# we plot the 20 most frequent words
f.plot(20,cumulative=False)
```

```
from wordcloud import WordCloud
import matplotlib.pyplot as plt
#generate word cloud to visualize in a more intuite way the most frequent words
#WordCloud use an in-built Stopword list to remove stopwords
wordcloud = WordCloud(width = 5000, height = 3000, random_state=1, background_color='blue', colorma
plt.figure()
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
plt.show()
```

Macb
haue
come
thou
King
Macd
thee
hath
Night
Rosse
make
yet
Lady
time
Enter
thy
vpon
Macbeth
Banquo
hand
good
well
must
tis
see
vs
great
know
Wife
feare
selfe
doe
speake
Ile
Let
th
say
still
giue
may
blood
Death
Noble
man
thing
done
Lord
Sir
Mal
Lenox
Life
sleepe
Nature
one
take
day
loue
Heauen
La
call
Doct
looke
Fatherheart

COLLOCATIONS

Collocations are expressions of multiple words that co-occur frequently. They are words that stick together inside the text.

PMI (pointwise mutual information) is a frequency based technique that give us the probability of one word showing up in the text given that some word has shown up before it. In short it measures the conditional probability of a word x given a word y.

```
import ipywidgets as widgets
from IPython import display

# we use collocations in NLTK to extract the most common bigram,trigram and fourgram
from nltk.collocations import *
from nltk.metrics.association import QuadgramAssocMeasures


bigram_measures = nltk.collocations.BigramAssocMeasures()
trigram_measures = nltk.collocations.TrigramAssocMeasures()
#fourgram_measures = nltk.collocations.QuadgramAssocMeasures()
finderb = BigramCollocationFinder.from_words(mwords2)
findert = TrigramCollocationFinder.from_words(mwords2)
#finderf = QuadgramCollocationFinder.from_words(mwords2)
#we print the 10 most common n-grams
import pandas as pd
print("BIGRAMS")
# filter based on frequency to ignore all n-grams occuring less than 3 times
finderb.apply_freq_filter(3)
print(pd.DataFrame(finderb.nbest(bigram_measures.pmi,10), columns = ["Word 1","Word 2"]))
print("TRIGRAMS")
findert.apply_freq_filter(3)
print(pd.DataFrame(findert.nbest(trigram_measures.pmi,10), columns = ["Word 1","Word 2","Word 3"]))
#print("FOURGRAMS")
#finderf.apply_freq_filter(3)
#print(pd.DataFrame(finderf.nbest(fourgram_measures.pmi,10), columns = ["Word 1","Word 2","Word 3",

#PMI (pointwise mutual information) is a frequency based technique that give us the probability of c
#create a PMI table
bigramPMITable = pd.DataFrame(list(finderb.score_ngrams(bigram_measures.pmi)), columns=['bigram','PM
trigramPMITable = pd.DataFrame(list(findert.score_ngrams(trigram_measures.pmi)), columns=['trigram',

# create output widgets
widget1 = widgets.Output()
widget2 = widgets.Output()

# render in output widgets
with widget1:
    display.display(bigramPMITable)
with widget2:
    display.display(trigramPMITable)

# create HBox
hbox = widgets.HBox([widget1, widget2])

# render hbox
display.display(hbox)
```

```
BIGRAMS
     Word 1    Word 2
0      Fire     burne
1       ten  thousand
2  Cauldron    bubble
3     burne  Cauldron
4        dy        de
5   trouble      Fire
6   weyward   Sisters
7    Double    double
8      Drum   Colours
9       Wee         l
TRIGRAMS
    Word 1    Word 2    Word 3
0     Fire     burne  Cauldron
1    burne  Cauldron    bubble
2  trouble      Fire     burne
3      All    Double    double
4    Knock     Knock     Knock
5   Exeunt     Scena   Secunda
6   Exeunt     Scena   Quarta
7   Exeunt     Scena    Tertia
8    three   Witches         1
9    Enter     three   Witches
```

	bigram	PMI
0	(Fire, burne)	11.556626
1	(ten, thousand)	11.141588
2	(Cauldron, bubble)	10.819660
3	(burne, Cauldron)	10.819660
4	(dy, de)	10.819660

	trigram	PMI
0	(Fire, burne, Cauldron)	22.791323
1	(burne, Cauldron, bubble)	22.376286
2	(trouble, Fire, burne)	22.305897
3	(All, Double, double)	18.943327
4	(Knock, Knock, Knock)	18.013115

PART 3. MEASURES OF SUCCESS

Based on initial collection and exploration of the data we want establish our target and measures of success.

If we go back to the motivations for this project I want to master advanced techniques such as Callbacks, Tensorboard, Tensorflow, LSTM, Keras and GPT-2 as described as well in Chollet's Deep Learning with Python.

I have experimented with multiple techniques and solutions to compare results and expand my knowledge to advanced deep learning techniques that are a starting point for a future career in machine learning.

My first objective is to develop a model that allow me to understand the relationship between words using BOW(Bag of Words), TF-IDF and Word Embeddings and then train a classifier on the dataset that predict accurately if word A occurs near Word B.

The second objective is to learn how to do text generation where given an input sequence of text we can predict the output using LSTM and GPT-2. Here the measure of success is the quality of the output. Is the text generation of high quality and semantically meaningful.

PART 4. METRICS

Some metrics were set at start, initial metrics such as frequency and text statistics were the base for further research on the dataset.

Accuracy and precision are some important metrics that I have considered for the classifier since we wanted to consider the accuracy of prediction and the similarity between words.

For the text generation models instead I have considered the quality of the output and iterative progress in fine-tuning the model.

I have incorporated callbacks and Tensorboards in the model measurement which are advanced tools and techniques to measure, monitor and inspect your model success.

Tensorboard is Tensorflow visualization framework that allows to get rich and frequent feedback on what is happening inside the model and its performance.

With Tensorboard I have been able to measure the metrics during the training, the model architecture, visualizations of activations and gradients and also embeddings in 3D.

PART 5. EVALUATION PROTOCOL

Since I knew my objective I had to understand the most appropriate way to measure my current progress in achieving those objectives.

Testing and training set division and composition had to be chosen based on initial exploration.

Simple hold out validation has been the choice here based on the quality of the dataset.

PART 6. DEVELOP A FIRST MODEL

TEXT FEATURES

We cannot use the text itself with deep learning so we need to convert it to numeric values.

There are different approaches to this and I have experimented with them, for example Bag-of-Words, TF-IDF and word embeddings using Gensim library.

Ve want to vectorize the data based on word count, this is also known as **BOW (Bag of Words).**

```
#vectorization by word count
from sklearn.feature_extraction.text import CountVectorizer
vec = CountVectorizer()
X = vec.fit_transform(mwords2)
```

```
#sparse matrix converted to dataframe
import pandas as pd
pd.DataFrame(X.toarray(), columns = vec.get_feature_names())
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Functio
  warnings.warn(msg, category=FutureWarning)
       1603  abhorred  abide  abiure  aboue  abound  abroad  absence  absent  absolute  ...  y
    0     0         0      0       0      0       0       0        0       0         0  ...
    1     0         0      0       0      0       0       0        0       0         0  ...
    2     0         0      0       0      0       0       0        0       0         0  ...
    3     0         0      0       0      0       0       0        0       0         0  ...
    4     0         0      0       0      0       0       0        0       0         0  ...
  ...   ...       ...    ...     ...    ...     ...     ...      ...     ...       ...  ...
12044     0         0      0       0      0       0       0        0       0         0  ...
12045     0         0      0       0      0       0       0        0       0         0  ...
12046     0         0      0       0      0       0       0        0       0         0  ...
12047     0         0      0       0      0       0       0        0       0         0  ...
12048     0         0      0       0      0       0       0        0       0         0  ...

12049 rows × 3417 columns
```

The above method does not take into account the relative importance of a term in the document.

To have a better measure we use here **TF-IDF (Term Frequency-Inverse Document Frequency)** to weight the word counts by a measure of how often they appear in the document.

TF-IDF score represents the relative importance of a term in the document and the entire text. TF-IDF score is composed by two terms:

- the first computes the normalized Term Frequency (TF),
- the second term is the Inverse Document Frequency (IDF), computed as the logarithm of the number of the documents in the corpus divided by the number of documents where the specific term appears.

TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document)

IDF(t) = log_e(Total number of documents / Number of documents with term t in it)

```
#vectorization with TF-IDF
from sklearn.feature_extraction.text import TfidfVectorizer
vec = TfidfVectorizer()
X = vec.fit_transform(mwords2)
pd.DataFrame(X.toarray(), columns = vec.get_feature_names())
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Functio
  warnings.warn(msg, category=FutureWarning)
```

	1603	abhorred	abide	abiure	aboue	abound	abroad	absence	absent	absolute	...	y
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	
...	...	...	...	...	...	...	...	...	...	...	...	
12044	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	
12045	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	
12046	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	
12047	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	
12048	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	

12049 rows × 3417 columns

Word Embeddings is a numeric vector that represents a word in a lower-dimensional space.

It allows words with similar meaning to have a similar representation.

One of the advantages is to reduce the dimensionality using a word to predict the words around it and therefore capturing also the semantics between words.

We train below a classifier on the dataset to predict if word A occurs near word B.

We use Word2Vec that use a shallow neural network to embed words in a lower-dimensional vector space.

```
import gensim
from gensim.models import Word2Vec

model = gensim.models.Word2Vec(msents)

#we save the model for future reference
model.save('gutenberg.embeddings')

# words in our model
model.wv.index2word
```

```
 'lye',
 'little',
 'English',
 'rise',
 'Those',
 'Colours',
 'fled',
 'sit',
 'better',
 'Wood',
 'Tyrants',
 'Seyton',
 'Actus',
 'Prima',
 'faire',
 'ayre',
 'Secunda',
 'Say',
 'together',
 'Fortune',
 'bad',
 'comfort',
 'ouer',
 'another',
 'helpe',
 'Goe',
 'get',
 'terrible',
 'former',
 'Title',
 'Tertia',
 'Husband',
 'Charme',
 'farre',
 'Earth',
 'sound',
 'indeed',
 'Water',
 'stay',
 'Can',
 'Wee',
 'Onely',
 'Haue',
 'home',
 'Crowne',
 'harme',
 'honest',
 'Heart',
 'Against',
 'whose',
 'toward',
 'Flourish',
 'Attendants',
 'dy',
 'part',
 'safe',
 'hither',
 'sorrow',
```

```
#number of dimensions
model.wv['Macbeth']
```

```
    array([-0.35443422,  0.17828825, -0.09376097, -0.1880054 , -0.29076445,
            0.191049  , -0.04607759, -0.08554105, -0.16691715, -0.4843835 ,
            0.13086379, -0.11853001,  0.17701855,  0.05152978,  0.00834565,
            0.4532589 ,  0.04276754, -0.28456172,  0.22778673, -0.0251052 ,
            0.08147204,  0.22064416, -0.22629543, -0.03828051, -0.03008088,
            0.16627757, -0.14552045,  0.06207381, -0.2822905 ,  0.11444974,
            0.23169331, -0.16043739,  0.58737147,  0.09234127,  0.00094342,
         -0.36678854,  0.11826986, -0.13590477,  0.34952605,  0.1928778 ,
          0.28359824, -0.2337844 , -0.1565219 , -0.05358644,  0.07020001,
         -0.04285696,  0.15301515,  0.40693197, -0.18382466, -0.35118952,
         -0.01581182,  0.02455212, -0.16354774,  0.16552691,  0.14915606,
          0.19756605, -0.25773227, -0.44121897, -0.00126442,  0.0452926 ,
          0.041987  , -0.02613256,  0.01808384,  0.25815594,  0.36536098,
         -0.09578561,  0.02042233, -0.01368931,  0.25400108, -0.02626381,
          0.16253757,  0.32092893, -0.02615316,  0.2443197 , -0.18624465,
         -0.5051063 ,  0.37365237, -0.02907391,  0.38392562,  0.02010341,
          0.17091101, -0.06454128, -0.10008939, -0.07976638, -0.23050818,
          0.23365356, -0.11953653, -0.23572087,  0.1672621 , -0.20575519,
         -0.24199766,  0.13944161,  0.01387719, -0.17551744,  0.15557583,
          0.17259991, -0.00079818,  0.02202378,  0.02710741, -0.27982014],
        dtype=float32)
```

```
#similarity between words
model.wv.similarity('Macbeth','Thane')
```

```
     0.9996848
```

```
#top similar terms
model.wv.most_similar('Macbeth', topn = 10)
```

```
     [('vs', 0.9998487234115601),
      ('they', 0.9998390078544617),
      ('be', 0.9998345375061035),
      ('haue', 0.9998307228088379),
      ('to', 0.999830424785614),
      (',', 0.9998299479484558),
      ('do', 0.9998297691345215),
      ('d', 0.9998294115066528),
      ('with', 0.9998282790184021),
      ('the', 0.9998273849487305)]
```

```
#reduce word vector dimensionality with t-SNE (t-Distributed Stochastic Name Embedding)
from sklearn.manifold import TSNE
X = model[model.wv.vocab]
tsne = TSNE(n_components=2, n_iter=250)
X_2d = tsne.fit_transform(X)
```

```
     /usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: DeprecationWarning: Call to de
       This is separate from the ipykernel package so we can avoid doing imports until
     /usr/local/lib/python3.7/dist-packages/sklearn/manifold/_t_sne.py:783: FutureWarning: The defa
       FutureWarning,
     /usr/local/lib/python3.7/dist-packages/sklearn/manifold/_t_sne.py:793: FutureWarning: The defa
       FutureWarning,
```

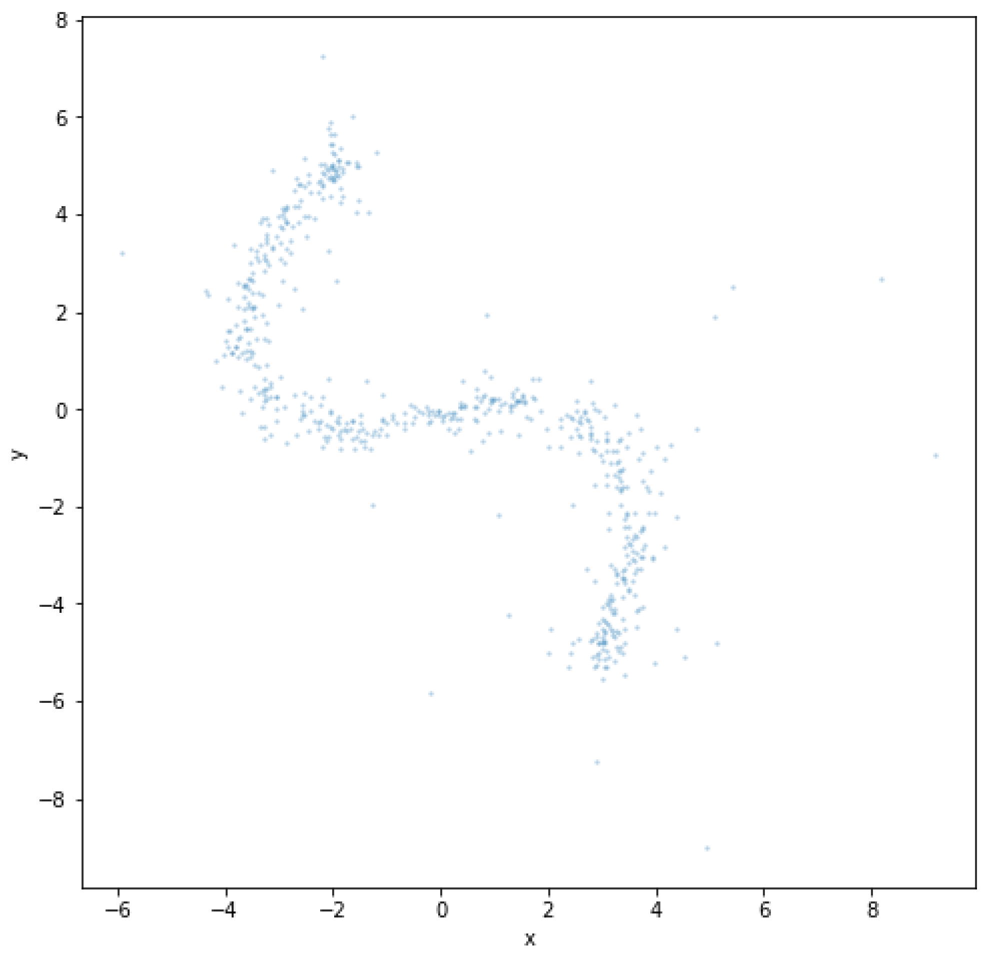
```
#use pandas to create a dataframe
coords_df = pd.DataFrame(X_2d, columns=['x', 'y'])
coords_df['token'] = model.wv.vocab.keys()
```

```
#output the first 5 values
coords_df.head()
```

	x	y	token
0	-1.957978	5.648552	The
1	-1.530583	4.989043	of
2	-2.919196	4.061176	Macbeth
3	-2.943707	3.699504	by
4	3.056102	-5.139109	Actus

```
#output the dataframe to csv file
coords_df.to_csv('clean_gutenberg_tsne.csv', index=False)
```

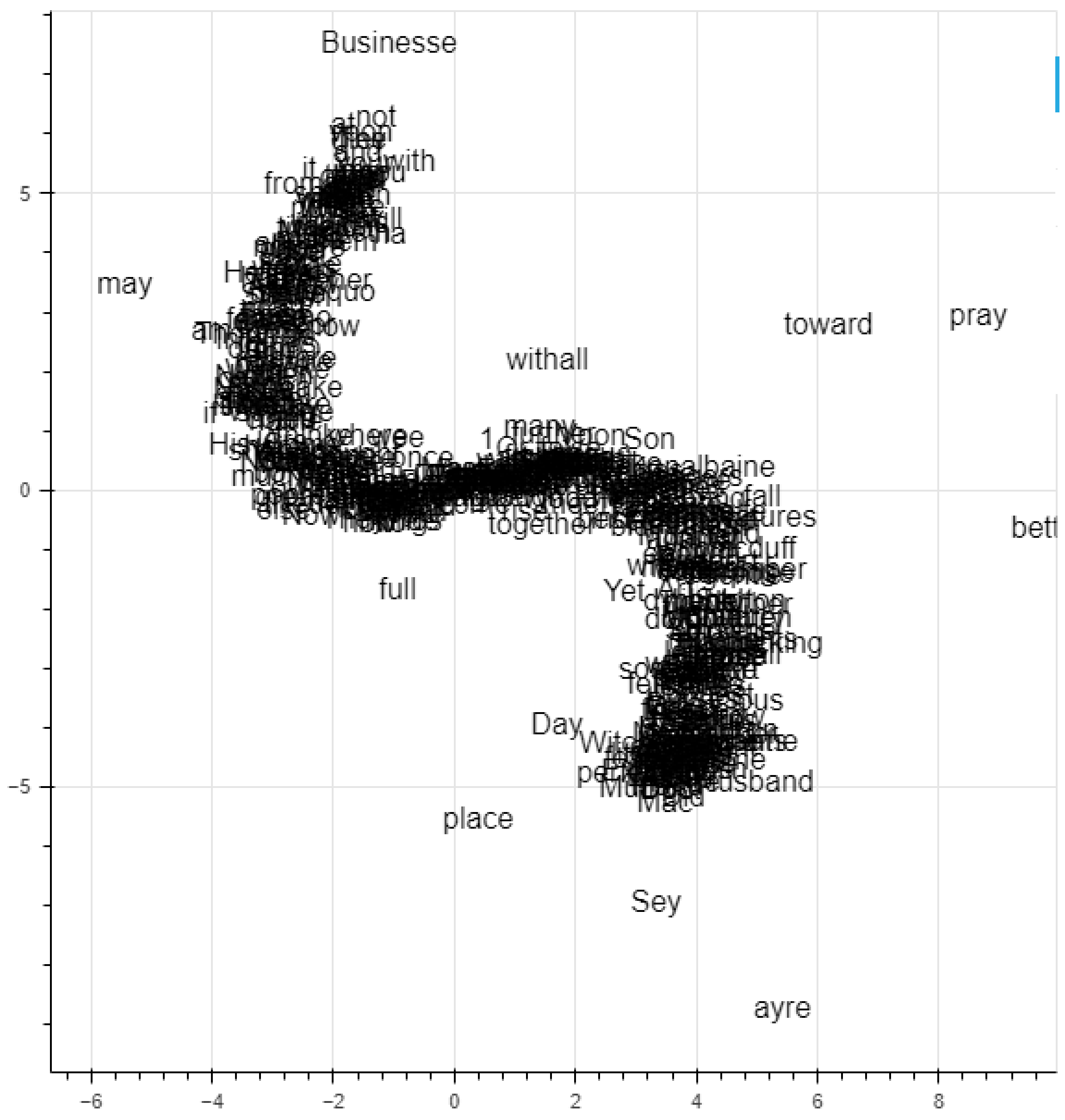
```
#use a scatterplot to render the dataframe
_ = coords_df.plot.scatter('x', 'y', figsize=(8,8), marker='.', s=10, alpha=0.2)
```



```
from bokeh.io import output_notebook
output_notebook()
```

```
#use bokeh to plot the dataframe and its value and understand the outliers.
from bokeh.plotting import show, figure
subset_df = coords_df.sample(n=500)
p = figure(plot_width=600, plot_height=600)
_ = p.text(x=subset_df.x, y=subset_df.y, text=subset_df.token)
```

```
show(p)
```



KERAS

Text generation is one of the most important applications of NLP. Deep learning techniques are used for a variety of tasks from writing poetry to generating scripts and much more. In this I want to experiment with Keras how to create a prediction model where given an input text (I will continue using Macbeth of Shakespeare) we can predict the next word.

Deep Learning models are based on statistical algorithms, therefore to work with them we need to convert first the words to numbers.

```
import numpy as np
import keras
from keras.models import Sequential, load_model
from keras.layers import Dense, Embedding, LSTM, Dropout
from tensorflow.keras.utils import to_categorical, plot_model
from random import randint
import re
```

```
macbeth_text = nltk.corpus.gutenberg.raw('shakespeare-macbeth.txt')
print(macbeth_text[:500])
```

```
     [The Tragedie of Macbeth by William Shakespeare 1603]


     Actus Primus. Scoena Prima.

     Thunder and Lightning. Enter three Witches.

       1. When shall we three meet againe?
     In Thunder, Lightning, or in Raine?
       2. When the Hurley-burley's done,
     When the Battaile's lost, and wonne

        3. That will be ere the set of Sunne

        1. Where the place?
       2. Vpon the Heath

        3. There to meet with Macbeth

        1. I come, Gray-Malkin

        All. Padock calls anon: faire is foule, and foule is faire,
     Houer through
```

To make it easier the conversion we can clean up the data by removing useless components such as punctuations and special characters. Let us clean them.

```
#we create a function to process and clean the data

def preprocess_text(sen):
    #remove punctuations and numbers
    sentence = re.sub('[^a-zA-Z]', ' ', sen)

    #single character removal
    sentence = re.sub(r"\s+[a-zA-Z]\s+", ' ', sentence)

    #removing multiple spaces
    sentence = re.sub(r'\s+', ' ', sentence)



    return sentence.lower()


macbeth_text = preprocess_text(macbeth_text)


macbeth_text[:500]
```

```
' the tragedie of macbeth by william shakespeare actus primus scoena prima thunder and lightn
ing enter three witches when shall we three meet againe in thunder lightning or in raine when
the hurley burley done when the battaile lost and wonne that will be ere the set of sunne whe
```

```python
from nltk.corpus import stopwords
#tokenize the text in the dataset
from nltk.tokenize import word_tokenize
macbeth_text_words = (word_tokenize(macbeth_text))

#remove stopwords
macbeth_text_words2 = [w for w in macbeth_text_words if not w in stopwords.words('english')]

n_words = len(macbeth_text_words2)
unique_words = len(set(macbeth_text_words2))

print('Total Words: %d' % n_words)
print('Unique Words: %d' % unique_words)
```

```
Total Words: 10044
Unique Words: 3335
```

```python
#convert tokenized words to numbers
from keras.preprocessing.text import Tokenizer
tokenizer = Tokenizer(num_words=3336)
tokenizer.fit_on_texts(macbeth_text_words2)


#access the dictionary that contain the words and their relevant index.
vocab_size = len(tokenizer.word_index) + 1
word_2_index = tokenizer.word_index



#we can print the first element and we can verify the index associated
print(macbeth_text_words2[0])
print(word_2_index[macbeth_text_words2[0]])
```

```
tragedie
775
```

LSTM (LONG SHORT-TERM MEMORY NETWORK)

LSTM is a variation of RNN (Recurring Neural Network) and is used normally to solve sequence problems.

Text classification is normally a Many-To-One problem where we have an input of many words and the prediction is a single output.

```python
#we modify the shape of the input sequences and of the outputs

input_sequence = []
output_words = []
input_seq_length = 100

for i in range(0, n_words - input_seq_length , 1):
    in_seq = macbeth_text_words2[i:i + input_seq_length]
    out_seq = macbeth_text_words2[i + input_seq_length]
    input_sequence.append([word_2_index[word] for word in in_seq])
    output_words.append(word_2_index[out_seq])

#print the value of the first sequence
print(input_sequence[0])
```

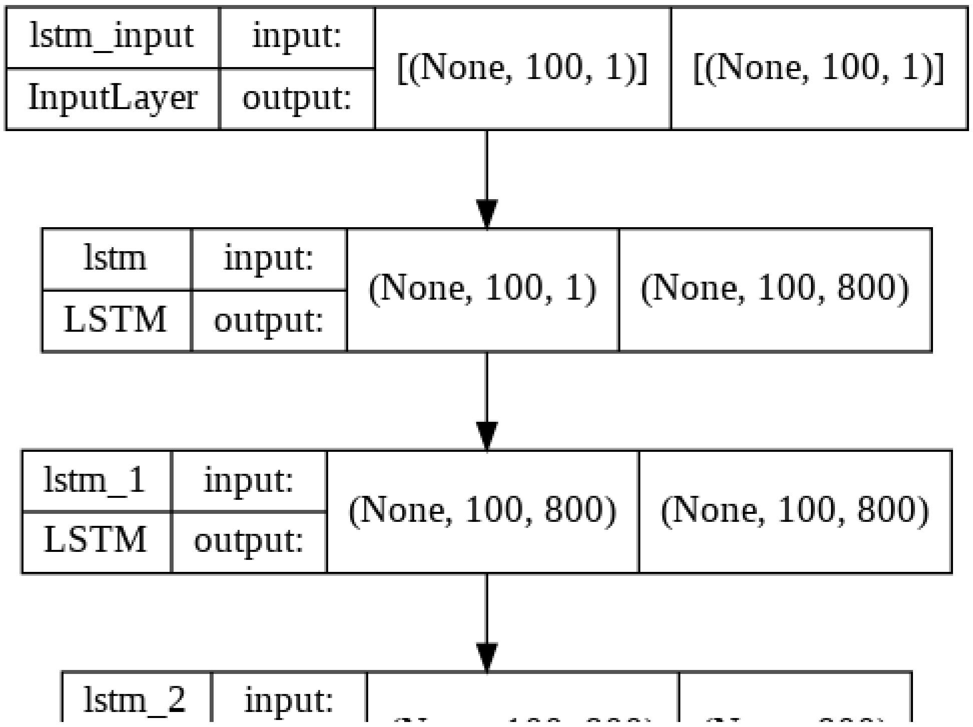
```
     [775, 6, 1260, 1261, 319, 1262, 1263, 320, 185, 776, 4, 117, 209, 5, 117, 241, 62, 185, 776, 1
```

```
#normalize our input sequences by dividing the integers in the sequence by the largest value and cor
X = np.reshape(input_sequence, (len(input_sequence), input_seq_length, 1))
X = X/float(vocab_size)
Y = to_categorical(output_words)


#print the shape of the inputs and outputs
print('X Shape: ', X.shape)
print('Y Shape: ', Y.shape)
```

```
     X Shape:  (9944, 100, 1)
     Y Shape:  (9944, 3336)
```

MODEL TRAINING

```
#train the model with 3 LSTM layers with 800 neurons each
#the final layer with 1 neuron is added to predict the index of the next word
model = Sequential()
model.add(LSTM(800, input_shape = (X.shape[1], X.shape[2]), return_sequences = True))
model.add(LSTM(800, return_sequences = True))
model.add(LSTM(800))

model.add(Dense(Y.shape[1], activation = 'softmax'))

model.summary()

model.compile(loss = 'categorical_crossentropy', optimizer = 'adam')

#use the fit method to train the model
model.fit(X, Y, batch_size = 64, epochs = 10, verbose = 1)



#for a visual overview of our model we can plot it
plot_model(model, show_shapes=True, to_file='model.png')
```



```
#randomly select a sequence of integers and print it
random_seq_index = np.random.randint(0,len(input_sequence)-1)
random_seq = input_sequence[random_seq_index]

index_2_word = dict(map(reversed, word_2_index.items()))

word_sequence = [index_2_word[value] for value in random_seq]

print(' '.join(word_sequence))
```

```
point second meeting doe finde patience predominant nature let goe gospell pray good man issue
```

```
#print the next 100 words that follow the previously generated sequence
#word_sequence contains the input sequence and the next 100 predicted words
for i in range(100):
    int_sample = np.reshape(random_seq, (1,len(random_seq), 1))
    int_sample = int_sample / float(vocab_size)

    predicted_word_index = model.predict(int_sample, verbose = 0)

    predicted_word_id = np.argmax(predicted_word_index)
    seq_in = [index_2_word[index] for index in random_seq]

    word_sequence.append(index_2_word[predicted_word_id])

    random_seq.append(predicted_word_id)
    random_seq = random_seq[1:len(random_seq)]



#we join the words in the list to get the final output
final_output = ""
for word in word_sequence:
    final_output = final_output + " " + word

print(final_output)
```

```
point second meeting doe finde patience predominant nature let goe gospell pray good man issu
```

PART 7. REGULARIZE THE MODEL AND FINE TUNE

The output is not very good, however we are able to create a text generation model with Keras.

we can to try and improve the results by modifying the hyper parameters and the size and number of LSTM layers and epochs. Furthermore the longer we will train the network the better the generated text will be.

we could increase the number of training epochs to 20 to improve the network performance and we could use a deeper neural network(more layers) or a wider network(larger number of neurons).

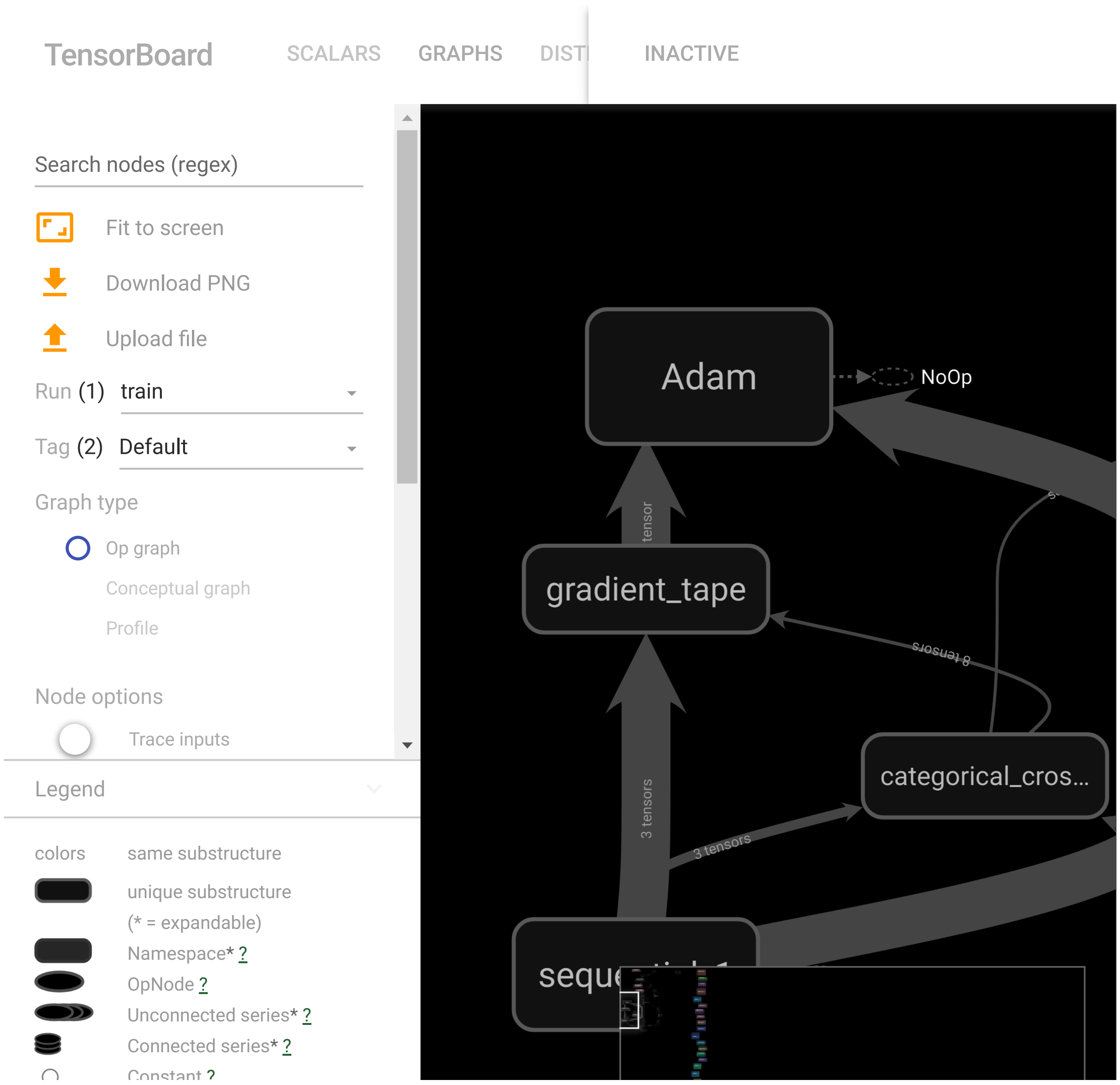
Due to resources constraints I am unable to run 20 epochs and will proceed with model analysis using Tensorboard and Callbacks.

Making progress on our model is an iterative process and we need to understand the results of our model. Tensorboard help to visualize and monitor everything that happen inside your model. Keras callbacks it is another advanced technique that allow to act on a model during the training.

```
import tensorflow as tf
import datetime, os
from tensorboard import notebook


callbacks = [
keras.callbacks.TensorBoard(
log_dir='my_log_dir',
histogram_freq=1,
embeddings_freq=1,
)
]
history = model.fit(X, Y,
epochs=10,
batch_size=64,
validation_split=0.2,
callbacks=callbacks)
```

```
125/125 [==============================] - 2112s 17s/step - loss: 7.3873 - val_loss: 7.4319
```

```
%load_ext tensorboard
%tensorboard --logdir=my_log_dir --load_fast true
```



GPT2

The above text generation approach is not the best output possible, we can try GPT2. GPT-2 is an AI based text generation model based on the Transformer architecture and trained on massive amount of text from internet.

The text generation results are much better then our first approach.

```
#install and import requirements
!pip install -q gpt-2-simple
import gpt_2_simple as gpt2
from datetime import datetime
from google.colab import files
from google.colab import drive
import tensorflow as tf
```

```
#there are 4 releases of GPT-2 (124M is the default and small model, 355M is the medium model, 774M
#due to resources constraints will use the small model
gpt2.download_gpt2(model_name="124M")
```

```
     Fetching checkpoint: 1.05Mit [00:00, 244Mit/s]
     Fetching encoder.json: 1.05Mit [00:00, 2.15Mit/s]
     Fetching hparams.json: 1.05Mit [00:00, 132Mit/s]
     Fetching model.ckpt.data-00000-of-00001: 498Mit [00:20, 24.7Mit/s]
     Fetching model.ckpt.index: 1.05Mit [00:00, 220Mit/s]
     Fetching model.ckpt.meta: 1.05Mit [00:00, 3.06Mit/s]
     Fetching vocab.bpe: 1.05Mit [00:00, 3.07Mit/s]
```

```
#mount Google Drive to access files
gpt2.mount_gdrive()
```

```
     Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/co
```

```
#IMPORTANT! file has been uploaded to sample_data folder (it will be erased when runtime is reset)
# To upload it again you can use the Files section on the left sidebar of Google Colaboratory.
# another solution for larger files is to upload the file to Google Drive root folder first and ther
# gpt2.copy_file_from_gdrive(file_name)
file_name = "shakespeare-macbeth.txt"
run_name = 'run1'


#TensorFlow session is created and training is run for the number of steps indicated
#IMPORTANT! to rerun the cell a Restart of Runtime is required together with the rerun of the impor
# restore_from is set to Fresh to start the training from the base GPT-2 if you want to start from a


tf.compat.v1.reset_default_graph()
sess = gpt2.start_tf_sess()

gpt2.finetune(sess,
              dataset=file_name,
              model_name='124M',
              steps=1000,
              restore_from='fresh',
              run_name='run1',
              print_every=10,
              sample_every=200,
              save_every=500
              )



#checkpoint folder is copied to Google Drive
gpt2.copy_checkpoint_to_gdrive(run_name='run1')


#if you want to load a trained model from Google Drive you can use command below:
#gpt2.load_gpt2(sess, run_name='run1')
```

```
#generate text from the trained model
#length is the number of tokens generated (default 1023)
#temperature is the creativity setting (the higher the crazier, default is 0.7)

gpt2.generate(sess,
              length=300,
              temperature=0.7,
              nsamples=5,
              batch_size=5,
              run_name='run1'
              )
```

```
     Eugene, a.d.

     Towards midnight, I was the first.
     The young woman sconceed in, and fell in from the shadow,
     Remembered the sleep, and fell asleep.


     Exit.




     Eugene was eight o'clock, and Sir Launcelot at once
     for the first time.
     The three women, who were in the way,
     lou'd so late in the night.
     In the evening, they were gone.
     Enter Sir W. Batten, and Lady Frere.
     Enter Sir W.


     Enter,
     Mother, and Master.



     Enter, and C.


     Enter, Sir W.
```

CONCLUSIONS

I have used Jupyter notebooks and the approach was to follow the Universal Workflow blueprint as described by Frederic Chollet in his book Deep Learning with Python. [1]

I managed to successfully execute text generation using two different methods (LSTM and GPT-2) and also did text classification with Word2Vec to measure similarity between 2 words in the text.

A possible evolution of this project that I am considering in the future is to create some kind of web interface where provided an input sequence of words we can predict the next word and also to integrate this process in the logic of text generation. I have used 2 of the most frequently used methods for text generation such as LSTM and GPT-2, there are many more methods that I would like to experiment with such as Markov, BART, Meteor and much more. GPT-3 is also a method I would like to experiment with in the future. The main obstacle I found in this project which curtailed my ambitions is the processing power of my computers any further development will require a lot more processing power than the one I used for this project. Google Colaboratory has helped a little but I could not process a larger dataset as I wished initially. I will also consider to use this model to evolve into a text generation app which could write poetry, generate scripts or even write a book in the future.

If the project and its subject is of your interest, please do not hesitate to contact me at: ds349@student.london.ac.uk for further information on this project or to provide any suggestions or comments.

Appendix A: List of References:

[1] François Chollet. 2021. Deep Learning with Python. Simon and Schuster, 2021

[2] Fisher, Ronald A. "The use of multiple measurements in taxonomic problems." Annals of eugenics 7.2 (1936): 179-188.

[3] Cortes, Corinna, and Vladimir Vapnik. "Support-vector networks." Machine learning 20.3 (1995): 273-297.

[4] Fix, Evelyn, and Joseph Lawson Hodges. "Discriminatory analysis. Nonparametric discrimination: Consistency properties." International Statistical Review/Revue Internationale de Statistique 57.3 (1989): 238-247.

[5] Wu, Xindong, et al. "Top 10 algorithms in data mining." Knowledge and information systems 14.1 (2008): 1-37.

[6] Ciregan, Dan, Ueli Meier, and Jürgen Schmidhuber. "Multi-column deep neural networks for image classification." 2012 IEEE conference on computer vision and pattern recognition. IEEE, 2012.

[7] Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." Neural computation 9.8 (1997): 1735-1780.