NLP Guide Book

Release 0.0.1

Nishant Baheti

Jan 03, 2024

CONTENTS

How to contact me?

Please feel free to reach out to me if something is wrong in this doc or if it was helpful for you or you want to have a discussion on something.

email	nishantbaheti.it19@gmail.com
github	https://www.github.com/nishantbaheti
linkedin	https://www.linkedin.com/in/nishantbaheti/
instagram	@_cranky_panda_

CHAPTER

ONE

NATURAL LANGUAGE PROCESSING

Combining Computer Science, Data Science and linguisitics for computer to understand human languages.

CHAPTER

TWO

CORPUS

```
[1]: import nltk
```

Corpus is latin for 'body', plural is Corpora.

2.1 Text Corpora /Text Corpus

- Gutenberg Corpus
- Web and Chat Text
- Brown Corpus
- Reuters Corpus
- Inaugural Address Corpus
- Annotated Text Corpora

```
>>> nltk.download('punkt')
>>> nltk.download('book')
```

will be downloaded in the user directory

2.1.1 Gutenberg Corpus

NLTK includes a small selection of texts from the Project Gutenberg electronic text archive, which contains some 25,000 free electronic books, hosted at.

http://www.gutenberg.org/

2.1.2 Brown

The Brown Corpus was the first million-word electronic corpus of English,

2.1.3 Inaugural

US presidential speeches

2.1.4 Popular Text Corpora

stopwords : Collection of stop words. reuters : Collection of news articles. cmudict : Collection of CMU Dictionary words. movie_reviews : Collection of Movie Reviews. np_chat : Collection of chat text. names : Collection of names associated with males and females. state_union : Collection of state union address. wordnet : Collection of all lexical entries. words : Collection of words in Wordlist corpus.

2.1.5 Text Corpus Structure

A text corpus is organized into any of the following four structures.

Isolated - Holds Individual text collections. Categorized - Each text collection tagged to a category. Overlapping - Each text collection tagged to one or more categories, and Temporal - Each text collection tagged to a period, date, time, etc.

[2]:
```
from nltk.corpus import genesis

genesis.fileids()
```

[2]:
```
['english-kjv.txt',
 'english-web.txt',
 'finnish.txt',
 'french.txt',
 'german.txt',
 'lolcat.txt',
 'portuguese.txt',
 'swedish.txt']
```

[3]:
```
from prettytable import PrettyTable

x = PrettyTable()
x.field_names = ["average word length","average sentence length","fileids"]
for fileid in genesis.fileids():
    n_chars = len(genesis.raw(fileid))
    n_words = len(genesis.words(fileid))
    n_sents = len(genesis.sents(fileid))


    x.add_row([int(n_chars/n_words), int(n_words/n_sents), fileid])

print(x)
```

```
+--------------------+------------------------+-----------------+
| average word length | average sentence length |     fileids     |
+--------------------+------------------------+-----------------+
|          4         |           30           | english-kjv.txt |
|          4         |           19           | english-web.txt |
|          5         |           15           |   finnish.txt   |
|          4         |           23           |    french.txt   |
```

(continues on next page)

(continued from previous page)

```
|          4          |            23           |    german.txt   |
|          4          |            20           |    lolcat.txt   |
|          4          |            27           |  portuguese.txt |
|          4          |            30           |   swedish.txt   |
+---------------------+-------------------------+-----------------+
```

[4]:
```
from nltk.corpus import inaugural
int(len(inaugural.words('1789-Washington.txt')) / len(set(inaugural.words('1789-
↪Washington.txt'))))
```

[4]:
```
2
```

CHAPTER

THREE

TOKENIZATION

- Chopping down a sentence into individual words/ group of words or `tokens`.
- Removing punctuations, special characters.
- Technique to simplify a corpus to prepare it for next stage of processing.

[1]:
```
import nltk

text = "Hello! I am Nishant. I am an engineer, and I like to build things."
```

3.1 sentence tokenizer

[2]:
```
sentences = nltk.sent_tokenize(text)
sentences
```

[2]:
```
['Hello!', 'I am Nishant.', 'I am an engineer, and I like to build things.']
```

3.2 word tokenizer

[3]:
```
words = nltk.word_tokenize(text)

words
```

[3]:
```
['Hello',
 '!',
 'I',
 'am',
 'Nishant',
 '.',
 'I',
 'am',
 'an',
 'engineer',
 ',',
 'and',
 'I',
 'like',
 'to',
```

(continues on next page)

(continued from previous page)

```
 'build',
 'things',
 '.']
```

3.3 Word Frequencies

[4]:
```
wordFreq = nltk.FreqDist(words)
print(f"""
    Word Frequencies : {wordFreq.elements}
    2 Most Common    : {wordFreq.most_common(2)}
""")
```

```
    Word Frequencies : <bound method Counter.elements of FreqDist({'I': 3, 'am': 2, '.': 2, 'Hello': 1, '!': 1, 'Nishant': 1, 'an': 1, 'engineer': 1, ',': 1, 'and': 1, ...})>
    2 Most Common    : [('I', 3), ('am', 2)]
```

3.4 Importing Items of book

[5]:
```
from nltk.book import *
```

```
*** Introductory Examples for the NLTK Book ***
Loading text1, ..., text9 and sent1, ..., sent9
Type the name of the text or sentence to view it.
Type: 'texts()' or 'sents()' to list the materials.
text1: Moby Dick by Herman Melville 1851
text2: Sense and Sensibility by Jane Austen 1811
text3: The Book of Genesis
text4: Inaugural Address Corpus
text5: Chat Corpus
text6: Monty Python and the Holy Grail
text7: Wall Street Journal
text8: Personals Corpus
text9: The Man Who Was Thursday by G . K . Chesterton 1908
```

3.4.1 find

[6]:
```
print(text1.findall("<tri.*r>"))
```

```
triangular; triangular; triangular; triangular
None
```

3.4.2 word count

[7]:
```
print(len(text1))
```
```
260819
```

3.4.3 unique word count

[8]:
```
print(len(set(text1)))
```
```
19317
```

3.4.4 transforming words

[9]:
```
print(len(set([word.lower() for word in set(text1)])))
```
```
17231
```

3.4.5 word coverage

[10]:
```
print(len(text1) / len(set(text1)))
```
```
13.502044830977896
```

3.4.6 filtering

[12]:
```
[word for word in set(text1) if word.startswith('Sun')]
```
[12]:
```
['Sunda', 'Sunday', 'Sunset']
```

CHAPTER

FOUR

STEMMING

Removing all fixes - prefixes, affixes and suffixes.

CHAPTER

FIVE

CO-OCCURANCE MARIX

[25]:
```
import re
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.sparse import csr_matrix
```

[26]:
```
strings = [
    "I like deep learning",
    "I like NLP",
    "I enjoy flying",
    "they enjoy flying",
    "I love eating cake",
    "I enjoy good cake",
    "I like coding",
    "they like coding"
]

p_str= '[a-zA-Z]{1,}'
pattern = re.compile(p_str, re.M)
groups = [re.findall(pattern,i) for i in strings]
groups
```

[26]:
```
[['I', 'like', 'deep', 'learning'],
 ['I', 'like', 'NLP'],
 ['I', 'enjoy', 'flying'],
 ['they', 'enjoy', 'flying'],
 ['I', 'love', 'eating', 'cake'],
 ['I', 'enjoy', 'good', 'cake'],
 ['I', 'like', 'coding'],
 ['they', 'like', 'coding']]
```

[27]:
```
bow = []
for i in groups:
    bow += i

bow = list(set(bow))
bow
```

[27]:
```
['deep',
 'cake',
 'good',
 'love',
 'like',
 'I',
 'flying',
 'NLP',
 'they',
 'enjoy',
 'eating',
 'learning',
 'coding']
```

[28]:
```
word_map = dict(enumerate(bow))
word_inv_map = { word_map[i]:i for i in word_map }
```

[29]:
```
word_map
```

[29]:
```
{0: 'deep',
 1: 'cake',
 2: 'good',
 3: 'love',
 4: 'like',
 5: 'I',
 6: 'flying',
 7: 'NLP',
 8: 'they',
 9: 'enjoy',
 10: 'eating',
 11: 'learning',
 12: 'coding'}
```

5.1 Context words association matrix

One step words

[30]:
```
bigrams_maps = []
for row in groups:
    length = len(row)
    for i in range(length - 1):
        w1 = word_inv_map[row[i]]
        w2 = word_inv_map[row[i+1]]
        bigrams_maps.append([w1, w2])
        bigrams_maps.append([w2, w1])
bigrams_maps = np.array(bigrams_maps)
```

[31]:
```
mat = csr_matrix((np.ones((bigrams_maps.shape[0])) ,
           (bigrams_maps[..., 0], bigrams_maps[..., 1]))).toarray()
```

[32]:
```
words:list = list(word_map.values())
groups, words
```

[32]:
```
([['I', 'like', 'deep', 'learning'],
  ['I', 'like', 'NLP'],
  ['I', 'enjoy', 'flying'],
  ['they', 'enjoy', 'flying'],
  ['I', 'love', 'eating', 'cake'],
  ['I', 'enjoy', 'good', 'cake'],
  ['I', 'like', 'coding'],
  ['they', 'like', 'coding']],
 ['deep',
  'cake',
  'good',
  'love',
  'like',
  'I',
  'flying',
  'NLP',
  'they',
  'enjoy',
  'eating',
  'learning',
  'coding'])
```

[33]:
```
df = pd.DataFrame(mat, columns = words)
df.index = words
df
```

[33]:

	deep	cake	good	love	like	I	flying	NLP	they	enjoy	eating	learning	coding
deep	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
cake	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
good	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
love	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
like	1.0	0.0	0.0	0.0	0.0	3.0	0.0	1.0	1.0	0.0	0.0	0.0	2.0
I	0.0	0.0	0.0	1.0	3.0	0.0	0.0	0.0	0.0	2.0	0.0	0.0	0.0
flying	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
NLP	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
they	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
enjoy	0.0	0.0	1.0	0.0	0.0	2.0	2.0	0.0	1.0	0.0	0.0	0.0	0.0
eating	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
learning	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
coding	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

(continues on next page)

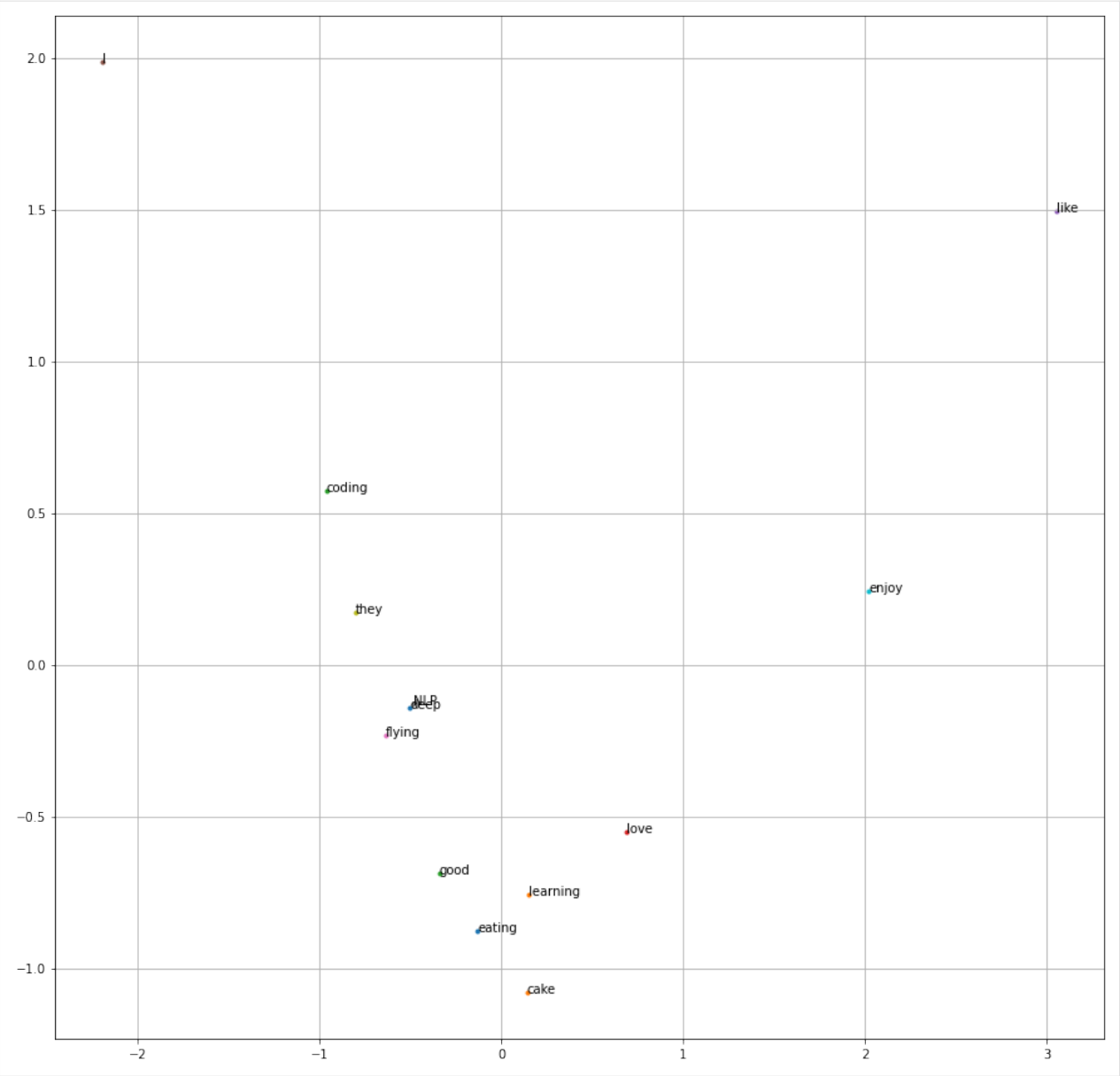
(continued from previous page)

```
NLP             0.0     0.0
they            0.0     0.0
enjoy           0.0     0.0
eating          0.0     0.0
learning        0.0     0.0
coding          0.0     0.0
```

5.2 Plotting word vectors

[34]:
```
from sklearn.decomposition import PCA
pc = PCA(n_components=2).fit_transform(mat)
```

[35]:
```
fig,ax = plt.subplots(1,1,figsize=(15,15))
for i in range(len(pc)):
    ax.scatter(pc[i,0],pc[i,1],marker='.')
    ax.text(pc[i,0],pc[i,1],words[i])

plt.grid()
plt.show()
```



[36]:
```
from scipy.spatial.distance import cosine
```

[37]:
```
def cosine_distance(word1, word2):
    return cosine(df.loc[word1].values, df.loc[word2].values)
```

[38]:
```
cosine_distance('like','enjoy')
```

[38]:
```
0.4466014094705336
```

[39]:
```
cosine_distance('like', 'love')
```

[39]:
```
0.4696699141100893
```

```
[40]: cosine_distance('enjoy', 'love')
[40]: 0.5527864045000421

[41]: cosine_distance('like', 'learning')
[41]: 0.75

[42]: cosine_distance('flying', 'learning')
[42]: 1.0

[43]: cosine_distance('I', 'they')
[43]: 0.05508881747693195
```

CHAPTER

SIX

COUNT BASED VECTORIZER

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.feature_extraction.text import (
    TfidfVectorizer, CountVectorizer, TfidfTransformer
)
```

6.1 References

https://en.wikipedia.org/wiki/Tf%E2%80%93idf

6.2 Count Vectorizer

frequency based on the set of words

[2]:
```
corpus = [
    'this is the first document',
    'this document is the second document',
    'and this is the third one'
]
```

[3]:
```
count_vec = CountVectorizer()
count_vec.fit(corpus)
```

[3]:
```
CountVectorizer()
```

[4]:
```
count_vec.get_feature_names()
```

[4]:
```
['and', 'document', 'first', 'is', 'one', 'second', 'the', 'third', 'this']
```

[5]:
```
count_data = count_vec.transform(corpus).toarray()
count_data
```

[5]:
```
array([[0, 1, 1, 1, 0, 0, 1, 0, 1],
       [0, 2, 0, 1, 0, 1, 1, 0, 1],
       [1, 0, 0, 1, 1, 0, 1, 1, 1]])
```

[6]:
```
pd.DataFrame(
    data=count_data,
    columns=count_vec.get_feature_names()
)
```

[6]:

	and	document	first	is	one	second	the	third	this
0	0	1	1	1	0	0	1	0	1
1	0	2	0	1	0	1	1	0	1
2	1	0	0	1	1	0	1	1	1

So the first line is `this is the first document`. and then the first vector is reflecting as document is 1, first is 1, is is 1 etc.

6.3 Term Frequency Transformer

Number of times term t, appear in the document d.

:nbsphinx-math:`begin{align}
tf_{t,d} &= frac{n_{t,d}}{sum_k{n_{k,d}}}

end{align}`

[7]:
```
tf_transformer = TfidfTransformer(use_idf=False)
tf_transformer.fit(count_data)
```

[7]:
```
TfidfTransformer(use_idf=False)
```

[8]:
```
tf_data = tf_transformer.transform(count_data).toarray()
tf_data
```

[8]:
```
array([[0.        , 0.4472136 , 0.4472136 , 0.4472136 , 0.        ,
        0.        , 0.4472136 , 0.        , 0.4472136 ],
       [0.        , 0.70710678, 0.        , 0.35355339, 0.        ,
        0.35355339, 0.35355339, 0.        , 0.35355339],
       [0.40824829, 0.        , 0.        , 0.40824829, 0.40824829,
        0.        , 0.40824829, 0.40824829, 0.40824829]])
```

[9]:
```
pd.DataFrame(
    data=tf_data,
    columns=count_vec.get_feature_names()
)
```

[9]:

	and	document	first	is	one	second	the	third	this
0	0.000000	0.447214	0.447214	0.447214	0.000000	0.000000	0.447214	0.000000	0.447214
1	0.000000	0.707107	0.000000	0.353553	0.000000	0.353553	0.353553	0.000000	0.353553
2	0.408248	0.000000	0.000000	0.408248	0.408248	0.000000	0.408248	0.408248	0.408248

6.4 Inverse Document Frequency Transformer

Document Frequency : number of ducuments that the term appears / number of ducuments

:nbsphinx-math:`begin{align}
probability = df_{t,d,D} &= frac{d in D : t in d}{D}

end{align}`

The log of the number of documents D divided by the number of documents that contain the word t. Inverse data frequency determines the weight of rare words across all documents in the corpus.

:nbsphinx-math:`begin{align}
idf &= -log(p)\ idf &= log(frac{1}{p})\ idf_{t,d,D} &= log(frac{D}{d in D : t in d})

end{align}`

document d is in all the documents D, and term t is in the document d.

[10]:
```
tfidf_transformer = TfidfTransformer()
tfidf_transformer.fit(count_data)
```

[10]:
```
TfidfTransformer()
```

[11]:
```
tfidf_data = tfidf_transformer.transform(count_data).toarray()
tfidf_data
```

[11]:
```
array([[0.        , 0.46941728, 0.61722732, 0.3645444 , 0.        ,
        0.        , 0.3645444 , 0.        , 0.3645444 ],
       [0.        , 0.7284449 , 0.        , 0.28285122, 0.        ,
        0.47890875, 0.28285122, 0.        , 0.28285122],
       [0.49711994, 0.        , 0.        , 0.29360705, 0.49711994,
        0.        , 0.29360705, 0.49711994, 0.29360705]])
```

[12]:
```
pd.DataFrame(
    data=tfidf_data,
    columns=count_vec.get_feature_names()
)
```

[12]:

	and	document	first	is	one	second	the	\
0	0.00000	0.469417	0.617227	0.364544	0.00000	0.000000	0.364544	
1	0.00000	0.728445	0.000000	0.282851	0.00000	0.478909	0.282851	
2	0.49712	0.000000	0.000000	0.293607	0.49712	0.000000	0.293607	

	third	this
0	0.00000	0.364544
1	0.00000	0.282851
2	0.49712	0.293607

6.5 TFIDF Vectorizer

Count Vectorizer –> Tfidf Transformer = Tfidf Vectorizer

$tfidf_{t,d,D} = tf_{t,d}.idf_{t,d,D}$

[13]:
```
tfidf_vec = TfidfVectorizer()
tfidf_vec.fit(corpus)
```

[13]:
```
TfidfVectorizer()
```

[14]:
```
tfidf_data = tfidf_vec.transform(corpus).toarray()
```

[15]:
```
tfidf_vec.get_feature_names()
```

[15]:
```
['and', 'document', 'first', 'is', 'one', 'second', 'the', 'third', 'this']
```

[16]:
```
pd.DataFrame(
    data=tfidf_data,
    columns=tfidf_vec.get_feature_names()
)
```

[16]:

	and	document	first	is	one	second	the	third	this
0	0.00000	0.469417	0.617227	0.364544	0.00000	0.000000	0.364544	0.00000	0.364544
1	0.00000	0.728445	0.000000	0.282851	0.00000	0.478909	0.282851	0.00000	0.282851
2	0.49712	0.000000	0.000000	0.293607	0.49712	0.000000	0.293607	0.49712	0.293607

CHAPTER

SEVEN

WORD2VEC ALGORITHM

[ ]:

CHAPTER

EIGHT

WORD2VEC EXAMPLE (AMAZON PRODUCTS REVIEWS)

[1]:
```
import numpy as np
import pandas as pd
import gensim
```

8.1 Loading data

The dataset we are using here is a subset of Amazon reviews from the Cell Phones & Accessories category. The data is stored as a JSON file and can be read using pandas.

Link to the Dataset: http://snap.stanford.edu/data/amazon/productGraph/categoryFiles/reviews_Cell_Phones_and_Accessories_5.json.gz

[2]:
```
df = pd.read_json("./datasets/Cell_Phones_and_Accessories_5.json",\
    lines=True)
df.head()
```

[2]:

	reviewerID	asin	reviewerName	helpful	reviewText	overall	summary	unixReviewTime	reviewTime
0	A30TL5EWN6DFXT	120401325X	christina	[0, 0]	They look good and stick good! I just don't li...	4	Looks Good	1400630400	05 21, 2014
1	ASY55RVNIL0UD	120401325X	emily l.	[0, 0]	These stickers work like the review says they ...	5	Really great product.	1389657600	01 14, 2014
2	A2TMXE2AFO7ONB	120401325X	Erica	[0, 0]	These are awesome and make my phone look so st...	5	LOVE LOVE LOVE	1403740800	06 26, 2014
3	AWJ0WZQYMYFQ4	120401325X	JM	[4, 4]	Item arrived in great time and was in perfect ...	4	Cute!	1382313600	10 21, 2013
4	ATX7CZYFXI1KW	120401325X	patrice m rogoza	[2, 3]	awesome! stays on, and looks great. can be use...	5	leopard home button sticker for iphone 4s	1359849600	02 3, 2013

[3]:
```
df.shape
```

[3]:
```
(194439, 9)
```

8.2 Extract Reviews

Extract review texts, apply simple preprocessing and tokenization.

[4]:
```
review_text = df.reviewText.apply(gensim.utils.simple_preprocess)
```

[5]:
```
review_text
```

[5]:
```
0         [they, look, good, and, stick, good, just, don...
1         [these, stickers, work, like, the, review, say...
2         [these, are, awesome, and, make, my, phone, lo...
3         [item, arrived, in, great, time, and, was, in,...
4         [awesome, stays, on, and, looks, great, can, b...
                                ...
194434    [works, great, just, like, my, original, one, ...
194435    [great, product, great, packaging, high, quali...
194436    [this, is, great, cable, just, as, good, as, t...
194437    [really, like, it, becasue, it, works, well, w...
194438    [product, as, described, have, wasted, lot, of...
Name: reviewText, Length: 194439, dtype: object
```

[6]:
```
review_text.loc[0]
```

[6]:
```
['they',
 'look',
 'good',
 'and',
 'stick',
 'good',
 'just',
 'don',
 'like',
 'the',
 'rounded',
 'shape',
 'because',
 'was',
 'always',
 'bumping',
 'it',
 'and',
 'siri',
 'kept',
 'popping',
 'up',
 'and',
 'it',
 'was',
 'irritating',
```

(continues on next page)

(continued from previous page)

```
 'just',
 'won',
 'buy',
 'product',
 'like',
 'this',
 'again']
```

8.3 Model Building

https://radimrehurek.com/gensim/models/word2vec.html

param	description
window = 10	10 words before the current word and 10 words after
min_count = 2	every sentence should have atleast two words
workers = 4	number of threads in CPUs

8.3.1 Continuous Bag of words model

[24]:
```
cbow_model = gensim.models.Word2Vec(
    window=10,
    min_count=2,
    workers=4,
    sg=0
)
```

[25]:
```
cbow_model.build_vocab(review_text, progress_per=1000)
```

[26]:
```
cbow_model.corpus_count, cbow_model.epochs
```

[26]:
```
(194439, 5)
```

[27]:
```
cbow_model.train(review_text, total_examples=cbow_model.corpus_count, epochs=cbow_model.
↪epochs)
```

[27]:
```
(61506525, 83868975)
```

[28]:
```
cbow_model.wv.most_similar("bad")
```

[28]:
```
[('terrible', 0.6839468479156494),
 ('shabby', 0.6071913838386536),
 ('horrible', 0.6051773428916931),
 ('good', 0.5834843516349792),
 ('awful', 0.5723922848701477),
 ('legit', 0.5297592878341675),
 ('funny', 0.5213862657546997),
 ('disappointing', 0.5114020705223083),
 ('ok', 0.5099763870239258),
 ('poor', 0.507624089717865)]
```

[29]:
```
cbow_model.wv.similarity(w1="cheap", w2="inexpensive")
```

[29]:
```
0.5118147
```

8.3.2 Skip-grams model

[31]:
```
sg_model = gensim.models.Word2Vec(
    window=10,
    min_count=2,
    workers=4,
    sg=1
)
```

[32]:
```
sg_model.build_vocab(review_text, progress_per=1000)
```

[33]:
```
sg_model.corpus_count, sg_model.epochs
```

[33]:
```
(194439, 5)
```

[34]:
```
sg_model.train(review_text, total_examples=sg_model.corpus_count, epochs=sg_model.epochs)
```

[34]:
```
(61501632, 83868975)
```

[35]:
```
sg_model.wv.most_similar("bad")
```

[35]:
```
[('terrible', 0.7510119676589966),
 ('horrible', 0.7072789669036865),
 ('good', 0.6595520377159119),
 ('soso', 0.6567561030387878),
 ('ok', 0.6563253402709961),
 ('okay', 0.6561095118522644),
 ('bleached', 0.6534101366996765),
 ('trilled', 0.6488171815872192),
 ('poor', 0.6452155113220215),
 ('stinks', 0.6401873230934143)]
```

[36]:
```
sg_model.wv.similarity(w1="cheap", w2="inexpensive")
```

[36]:
```
0.6635918
```

8.4 Compare Models

[42]:
```
import matplotlib.pyplot as plt
import seaborn as sns
```

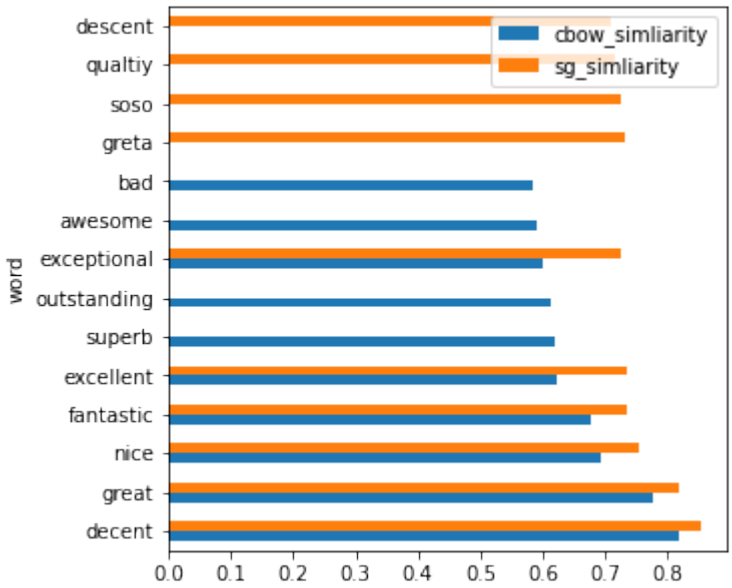
[150]:
```
def plot_word_similarities(word):
    cbow_sims = pd.DataFrame(cbow_model.wv.most_similar(word),\
                             columns=['word','cbow_simliarity'])
    sg_sims = pd.DataFrame(sg_model.wv.most_similar(word),\
```

(continues on next page)

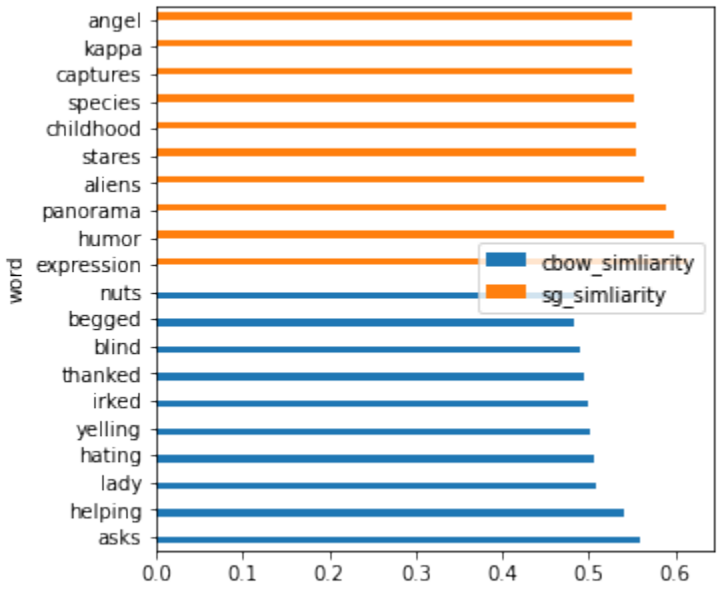
(continued from previous page)

```
                          columns=['word','sg_simliarity'])

    fig, ax = plt.subplots(1, 1, figsize=(5, 5))
    cbow_sims.join(sg_sims.set_index('word'), on='word', \
                   how='outer').plot.barh(x='word', ax=ax)
    plt.show()
```

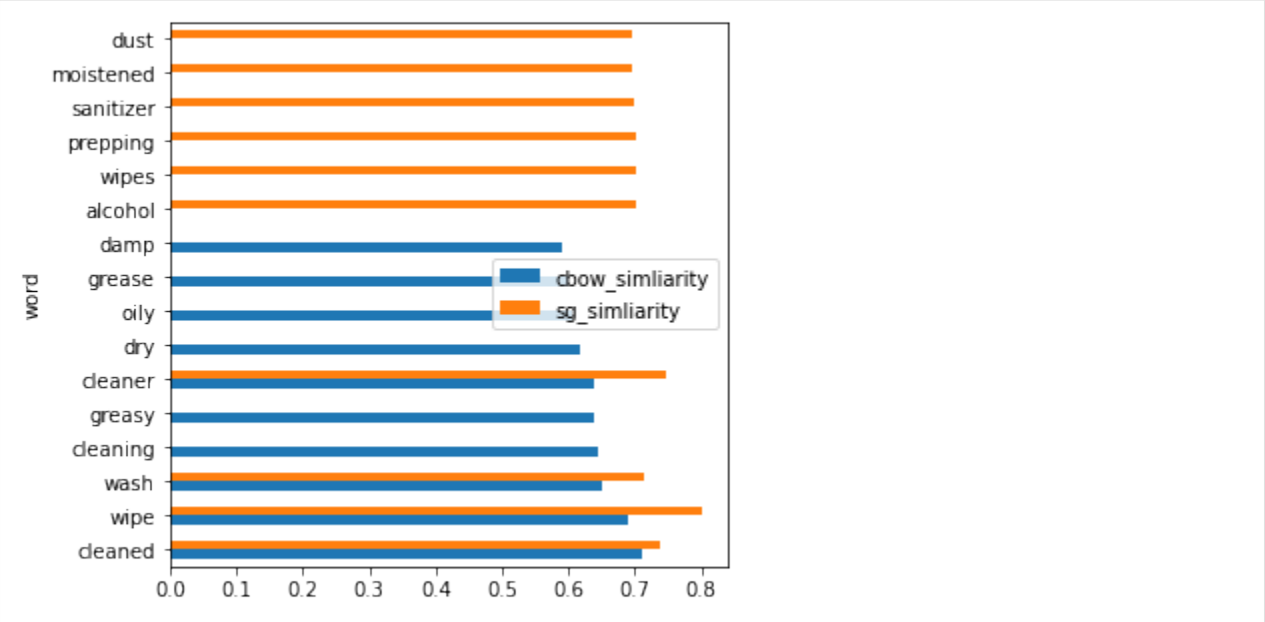
[151]:
```
plot_word_similarities('good')
```



[152]:
```
plot_word_similarities('smile')
```



[153]:
```
plot_word_similarities('clean')
```


dust
moistened
sanitizer
prepping
wipes
alcohol
damp
grease
oily
dry
cleaner
greasy
cleaning
wash
wipe
cleaned
word
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
cbow_simliarity
sg_simliarity

CHAPTER

NINE

BERT TRANSFORMER

`Bidirectional Encoder Representations from Transformers (BERT)`

Issue with Word2Vec, In these two sentences the word 'fair' has different meaning.

- It was not a `fair` game.
- The town `fair` was really fun.

But word2vec will only generate a single vector for scenarios. There is no context.

This contextualization meaning of the word will be provided by BERT.

The original English-language BERT has two models:

1. the BERTBASE: 12 encoders with 12 bidirectional self-attention heads
2. the BERTLARGE: 24 encoders with 16 bidirectional self-attention heads.

Both models are pre-trained from unlabeled data extracted from the BooksCorpus with 800M words and English Wikipedia with 2,500M words.

BERT was pretrained on two tasks 1. language modelling (15% of tokens were `masked`(removed) and BERT was trained to predict them from context) 2. next sentence prediction (BERT was trained to predict if a chosen next sentence was probable or not given the first sentence).

As a result of the training process, BERT learns contextual embeddings for words. After pretraining, which is computationally expensive, BERT can be finetuned with less resources on smaller datasets to optimize its performance on specific tasks.

9.1 References

https://en.wikipedia.org/wiki/BERT_(language_model)

http://jalammar.github.io/illustrated-bert/

https://www.tensorflow.org/hub

https://tfhub.dev/google/collections/bert/1

https://tfhub.dev/tensorflow/bert_en_uncased_L-12_H-768_A-12/4

https://www.youtube.com/c/codebasics

CHAPTER

TEN

BERT DEMO

10.1 References

https://www.tensorflow.org/hub

https://tfhub.dev/google/collections/bert/1

https://tfhub.dev/tensorflow/bert_en_uncased_L-12_H-768_A-12/4

https://www.youtube.com/c/codebasics

10.2 Loading Libraries

[1]:
```
import tensorflow_hub as hub
import tensorflow_text as text
```

10.3 Loading Urls

encoder and preprocessing url

[2]:
```
encoder_url = "https://tfhub.dev/tensorflow/bert_en_uncased_L-12_H-768_A-12/4"
preprocessing_url = "https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3"
```

10.4 Loading BERT Preprocessor

[3]:
```
preprocessor = hub.KerasLayer(preprocessing_url)
```

10.4.1 testing preprocessor

[5]:
```
test_text = [
    "this is first document",
    "I love pasta"
]

test_dict = preprocessor(test_text)
test_dict.keys()
```

[5]:
```
dict_keys(['input_word_ids', 'input_mask', 'input_type_ids'])
```

Preprocessor returns a dictionary with 3 values.

- input_word_ids : has the token ids of the input sequences.
- input_mask : has value 1 at the position of all input tokens present before padding and value 0 for the padding tokens.
- input_type_ids : has the index of the input segment that gave rise to the input token at the respective position. The first input segment (index 0) includes the start-of-sequence token and its end-of-segment token. The second segment (index 1, if present) includes its end-of-segment token. Padding tokens get index 0 again.

every sentence/row has prefix `CLS` and suffix `SEP`.

and total size of vector for 1 row will be `128`.

[9]:
```
test_dict['input_word_ids'].shape
```

[9]:
```
TensorShape([2, 128])
```

```
CLS this is first document SEP
```

[13]:
```
test_dict['input_word_ids'][0]
```

[13]:
```
<tf.Tensor: shape=(128,), dtype=int32, numpy=
array([ 101, 2023, 2003, 2034, 6254,  102,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0], dtype=int32)>
```

[14]:
```
test_dict['input_mask'].shape
```

[14]:
```
TensorShape([2, 128])
```

[15]:
```
test_dict['input_mask'][0]
```

[15]:
```
<tf.Tensor: shape=(128,), dtype=int32, numpy=
array([1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=int32)>
```

[16]:
```
test_dict['input_type_ids'].shape
```

[16]:
```
TensorShape([2, 128])
```

[20]:
```
test_dict['input_type_ids'][0]
```

[20]:
```
<tf.Tensor: shape=(128,), dtype=int32, numpy=
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=int32)>
```

10.5 Loading Encoder

[26]:
```
encoder_model = hub.KerasLayer(encoder_url)
```

10.5.1 testing encoder

[28]:
```
encoded_dict = encoder_model(test_dict)
```

[29]:
```
encoded_dict.keys()
```

[29]:
```
dict_keys(['pooled_output', 'sequence_output', 'encoder_outputs', 'default'])
```

[30]:
```
encoded_dict['pooled_output']
```

[30]:
```
<tf.Tensor: shape=(2, 768), dtype=float32, numpy=
array([[-0.8564542 , -0.20264098,  0.44491994, ...,  0.18142326,
        -0.52242935,  0.839577  ],
       [-0.8231588 , -0.19543926,  0.4399575 , ...,  0.3558046 ,
        -0.5919596 ,  0.8541847 ]], dtype=float32)>
```

Pooled output gives embedding for the sentences/ rows. we have 2 sentences in input text and two pooled outputs representing the embedding.

[ ]:
```
encoded_dict['sequence_output'].shape
```

```
<tf.Tensor: shape=(2, 128, 768), dtype=float32, numpy=
array([[[-2.23829597e-01,  2.58546323e-01,  1.60378978e-01, ...,
```

(continues on next page)

(continued from previous page)

```
         -1.75076678e-01,  2.57635832e-01,  3.58420372e-01],
        [-5.81663609e-01, -2.24008322e-01,  6.07548207e-02, ...,
         -4.35946345e-01,  7.22116292e-01,  2.05105364e-01],
        [-5.88114560e-01, -3.27617466e-01,  6.20402157e-01, ...,
         -4.78888750e-01,  5.28291821e-01,  6.42452359e-01],
        ...,
        [-2.36085236e-01,  1.05683051e-01,  5.52026212e-01, ...,
          1.31120771e-01,  4.79894549e-01,  3.75098825e-01],
        [-2.66874582e-01,  8.43454301e-02,  5.48812032e-01, ...,
          1.94243729e-01,  4.29042369e-01,  3.45293581e-01],
        [-3.05021435e-01,  4.44966406e-02,  5.43690860e-01, ...,
          2.48683482e-01,  4.10975337e-01,  2.66701370e-01]],

       [[ 8.02702308e-02,  2.39341095e-01,  6.48294538e-02, ...,
         -2.08112523e-01,  1.62344888e-01,  2.63515770e-01],
        [ 1.01238646e-01,  2.56397724e-01,  1.76448345e-01, ...,
         -3.21156919e-01,  7.11108208e-01,  3.40716913e-02],
        [ 1.00183558e+00,  7.70643294e-01,  6.90153003e-01, ...,
         -2.05520004e-01,  4.99139577e-01, -4.82482314e-02],
        ...,
        [ 1.50605410e-01,  2.65507936e-01,  6.01852715e-01, ...,
          2.56534293e-03, -7.26580620e-04,  2.05264628e-01],
        [ 7.44732320e-02,  1.70520827e-01,  4.65080142e-01, ...,
          6.46677464e-02, -2.52081901e-02,  1.48678273e-01],
        [-3.44444752e-01, -7.48260766e-02,  3.19036841e-01, ...,
          2.54593313e-01, -5.94146326e-02,  1.00927249e-01]]],
      dtype=float32)>
```

[35]:
```
enc_output = encoded_dict['encoder_outputs']
```

[36]:
```
len(enc_output)
```

[36]:
```
12
```

As we are using BERT small and it has 12 encoder layers. and the outputs mentioned in this is output of each individual encoder layer.

[38]:
```
enc_output[0].shape
```

[38]:
```
TensorShape([2, 128, 768])
```

as we have 2 sentences, so first shape is 2, then 128 is padding and 768 sized embedding.

[39]:
```
enc_output[-1] == encoded_dict['sequence_output']
```

[39]:
```
<tf.Tensor: shape=(2, 128, 768), dtype=bool, numpy=
array([[[ True,  True,  True, ...,  True,  True,  True],
        [ True,  True,  True, ...,  True,  True,  True],
        [ True,  True,  True, ...,  True,  True,  True],
        ...,
        [ True,  True,  True, ...,  True,  True,  True],
```

(continues on next page)

(continued from previous page)

```
        [ True,  True,  True, ...,  True,  True,  True],
        [ True,  True,  True, ...,  True,  True,  True]],

       [[ True,  True,  True, ...,  True,  True,  True],
        [ True,  True,  True, ...,  True,  True,  True],
        [ True,  True,  True, ...,  True,  True,  True],
        ...,
        [ True,  True,  True, ...,  True,  True,  True],
        [ True,  True,  True, ...,  True,  True,  True],
        [ True,  True,  True, ...,  True,  True,  True]]])>
```

so the last encoder output is actually sequence output.. Obviously.

[41]:
```
encoded_dict['default'] == encoded_dict['pooled_output']
```

[41]:
```
<tf.Tensor: shape=(2, 768), dtype=bool, numpy=
array([[ True,  True,  True, ...,  True,  True,  True],
       [ True,  True,  True, ...,  True,  True,  True]])>
```

CHAPTER

ELEVEN

BERT EXAMPLE (SPAM OR HAM CLASSIFICATION)

[1]:
```
!pip install -q tensorflow-text
```

```
     || 4.9 MB 6.7 MB/s
```

[4]:
```
import pandas as pd
import numpy as np
import tensorflow as tf
import tensorflow_hub as hub
import tensorflow_text as text
import csv
import matplotlib.pyplot as plt
import datetime

%matplotlib inline
%load_ext tensorboard
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```

[5]:
```
def check_gpu():
    device_name = tf.test.gpu_device_name()
    if device_name != '/device:GPU:0':
        return 'GPU device not found'
    return 'Found GPU at: {}'.format(device_name)
check_gpu()
```

[5]:
```
'Found GPU at: /device:GPU:0'
```

[7]:
```
log_dir = "logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1)
```

11.1 Loading URLs

[8]:
```
encoder_url = "https://tfhub.dev/tensorflow/bert_en_uncased_L-12_H-768_A-12/4"
preprocessing_url = "https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3"
```

11.2 Loading data

[9]:
```
df = pd.read_csv("./SMSSpamCollection.csv", sep='\t', \
                 quoting=csv.QUOTE_NONE, names=["label", "message"])
df.head()
```

[9]:
```
  label                                            message
0   ham  Go until jurong point, crazy.. Available only ...
1   ham                      Ok lar... Joking wif u oni...
2  spam  Free entry in 2 a wkly comp to win FA Cup fina...
3   ham  U dun say so early hor... U c already then say...
4   ham  Nah I don't think he goes to usf, he lives aro...
```

11.3 Checking Class Count

[10]:
```
df['label'].value_counts()
```

[10]:
```
ham     4827
spam     747
Name: label, dtype: int64
```

11.4 Treating Imbalanced Class

Looks like there is an imbalance. There are two ways to handle it, either upsampling and down sampling.

[11]:
```
spam_df = df[df.label == 'spam']
ham_df = df[df.label == 'ham']
```

[12]:
```
ham_df = ham_df.sample(spam_df.shape[0])
```

[13]:
```
new_dataset = pd.concat([spam_df, ham_df]).sample(frac=1)

new_dataset.head()
```

[13]:
```
     label                                            message
1134   ham  As I entered my cabin my PA said, '' Happy B'd...
5173   ham                        Oh k. . I will come tomorrow
4077  spam  87077: Kick off a new season with 2wks FREE go...
3189  spam  This is the 2nd time we have tried 2 contact u...
4834  spam  New Mobiles from 2004, MUST GO! Txt: NOKIA to ...
```

[14]:
```
new_dataset.label.value_counts()
```

[14]:
```
ham     747
spam    747
Name: label, dtype: int64
```

11.5 Target Class 1 hot encoding

[15]:
```
new_dataset['spam'] = new_dataset.label.apply(lambda x: 1 if x=='spam' else 0)
new_dataset['ham'] = new_dataset.label.apply(lambda x: 1 if x=='ham' else 0)
new_dataset.head()
```

[15]:

	label	message	spam	ham
1134	ham	As I entered my cabin my PA said, '' Happy B'd...	0	1
5173	ham	Oh k. . I will come tomorrow	0	1
4077	spam	87077: Kick off a new season with 2wks FREE go...	1	0
3189	spam	This is the 2nd time we have tried 2 contact u...	1	0
4834	spam	New Mobiles from 2004, MUST GO! Txt: NOKIA to ...	1	0

11.6 Loading BERT Preprocessor

[16]:
```
preprocessor = hub.KerasLayer(preprocessing_url)
bert_encoder = hub.KerasLayer(encoder_url)
```

[17]:
```
def get_embeddings(sentences):
    preprocessed_values = preprocessor(sentences)
    return bert_encoder(preprocessed_values)['pooled_output']
```

[18]:
```
embeddings = get_embeddings([
    'I am an artist',
    'He was writing',
    'I paint everyday'
])
```

[19]:
```
from sklearn.metrics.pairwise import cosine_similarity
```

[20]:
```
cosine_similarity([embeddings[0]],[embeddings[1]])
```

[20]:
```
array([[0.7917898]], dtype=float32)
```

[21]:
```
cosine_similarity([embeddings[1]],[embeddings[2]])
```

[21]:
```
array([[0.770146]], dtype=float32)
```

[22]:
```
cosine_similarity([embeddings[0]],[embeddings[2]])
```

[22]:
```
array([[0.99068]], dtype=float32)
```

11.7 Building Model

Using tensorflow functional apis -

The Keras functional API is a way to create models that are more flexible than the tf.keras.Sequential API. The functional API can handle models with non-linear topology, shared layers, and even multiple inputs or outputs.

[23]:
```
text_input = tf.keras.layers.Input(shape=(), dtype=tf.string, name='Input-Text')
preprocessed_text = preprocessor(text_input)
encoded_text = bert_encoder(preprocessed_text)


dropout_output = tf.keras.layers.Dropout(0.2, name='Dropout-Layer')(encoded_text['pooled_
↪output'])
relu_output = tf.keras.layers.Dense(units=64, activation='relu', name='Relu-Layer
↪')(dropout_output)
output = tf.keras.layers.Dense(units=2, activation='softmax', name='Output-Layer')(relu_
↪output)

model = tf.keras.Model(inputs=[text_input], outputs=[output])
```

[24]:
```
model.summary()
```

```
Model: "model"
_____________________________________________________________________________________________
↪_________
 Layer (type)                 Output Shape              Param #     Connected to
==============================================================================================
 Input-Text (InputLayer)      [(None,)]                 0           []

 keras_layer (KerasLayer)     {'input_word_ids':        0           ['Input-Text[0][0]']
                              (None, 128),
                               'input_type_ids':
                              (None, 128),
                               'input_mask': (Non
                              e, 128)}

 keras_layer_1 (KerasLayer)   {'encoder_outputs':       109482241   ['keras_layer[0][0]',
                               [(None, 128, 768),                    'keras_layer[0][1]',
                               (None, 128, 768),                     'keras_layer[0][2]']
                               (None, 128, 768),
                               (None, 128, 768),
                               (None, 128, 768),
                               (None, 128, 768),
                               (None, 128, 768),
                               (None, 128, 768),
                               (None, 128, 768),
                               (None, 128, 768),
                               (None, 128, 768),
                               (None, 128, 768),
                               (None, 128, 768)],
                               'default': (None,
                              768),
                               'pooled_output': (
```

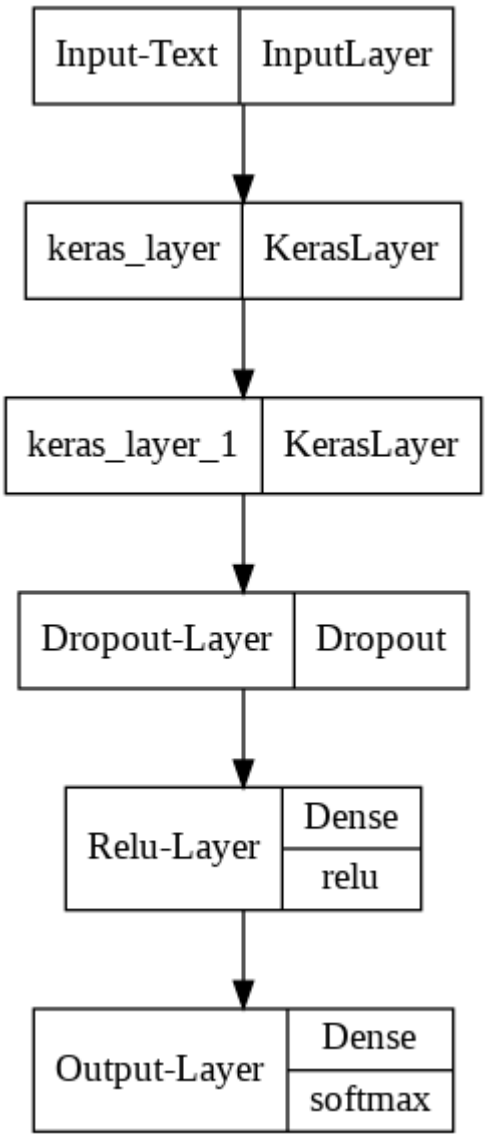
(continues on next page)

(continued from previous page)

```
                                None, 768),
                                 'sequence_output':
                                 (None, 128, 768)}

 Dropout-Layer (Dropout)        (None, 768)          0           ['keras_layer_1[0][13]']

 Relu-Layer (Dense)             (None, 64)           49216       ['Dropout-Layer[0][0]']

 Output-Layer (Dense)           (None, 2)            130         ['Relu-Layer[0][0]']

==================================================================================================
Total params: 109,531,587
Trainable params: 49,346
Non-trainable params: 109,482,241
__________________________________________________________________________________________________
↪__________
```

[25]:
```
tf.keras.utils.plot_model(model,show_layer_activations=True)
```

[25]:



[26]:
```
METRICS = [
    tf.keras.metrics.CategoricalAccuracy(name='accuracy'),
    tf.keras.metrics.Precision(name='precision'),
    tf.keras.metrics.Recall(name='recall')
]

model.compile(
    optimizer='adam',
    loss=tf.keras.losses.CategoricalCrossentropy(),
    metrics=METRICS
)
```

[27]:
```
%tensorboard --logdir logs/fit
```

```
<IPython.core.display.Javascript object>
```

[28]:
```
history = model.fit(new_dataset['message'], new_dataset[['spam','ham']], epochs=30,␣
↪validation_split=0.1,\
                    shuffle=True, batch_size=32, callbacks=[tensorboard_callback])
```

```
Epoch 1/30
42/42 [==============================] - 47s 898ms/step - loss: 0.5015 - accuracy: 0.
↪7500 - precision: 0.7500 - recall: 0.7500 - val_loss: 0.2910 - val_accuracy: 0.9133 -␣
↪val_precision: 0.9133 - val_recall: 0.9133
Epoch 2/30
42/42 [==============================] - 37s 902ms/step - loss: 0.3127 - accuracy: 0.
↪8705 - precision: 0.8705 - recall: 0.8705 - val_loss: 0.2676 - val_accuracy: 0.9067 -␣
↪val_precision: 0.9067 - val_recall: 0.9067
Epoch 3/30
42/42 [==============================] - 34s 823ms/step - loss: 0.2545 - accuracy: 0.
↪9010 - precision: 0.9010 - recall: 0.9010 - val_loss: 0.1959 - val_accuracy: 0.9333 -␣
↪val_precision: 0.9333 - val_recall: 0.9333
Epoch 4/30
42/42 [==============================] - 38s 911ms/step - loss: 0.2192 - accuracy: 0.
↪9204 - precision: 0.9204 - recall: 0.9204 - val_loss: 0.1943 - val_accuracy: 0.9400 -␣
↪val_precision: 0.9400 - val_recall: 0.9400
Epoch 5/30
42/42 [==============================] - 37s 890ms/step - loss: 0.2235 - accuracy: 0.
↪9167 - precision: 0.9167 - recall: 0.9167 - val_loss: 0.1607 - val_accuracy: 0.9400 -␣
↪val_precision: 0.9400 - val_recall: 0.9400
Epoch 6/30
42/42 [==============================] - 35s 851ms/step - loss: 0.1903 - accuracy: 0.
↪9278 - precision: 0.9278 - recall: 0.9278 - val_loss: 0.1431 - val_accuracy: 0.9467 -␣
↪val_precision: 0.9467 - val_recall: 0.9467
Epoch 7/30
42/42 [==============================] - 38s 906ms/step - loss: 0.1804 - accuracy: 0.
↪9338 - precision: 0.9338 - recall: 0.9338 - val_loss: 0.1351 - val_accuracy: 0.9467 -␣
↪val_precision: 0.9467 - val_recall: 0.9467
Epoch 8/30
42/42 [==============================] - 38s 914ms/step - loss: 0.1627 - accuracy: 0.
↪9427 - precision: 0.9427 - recall: 0.9427 - val_loss: 0.1531 - val_accuracy: 0.9533 -␣
↪val_precision: 0.9533 - val_recall: 0.9533
Epoch 9/30
42/42 [==============================] - 37s 894ms/step - loss: 0.1891 - accuracy: 0.
↪9308 - precision: 0.9308 - recall: 0.9308 - val_loss: 0.1272 - val_accuracy: 0.9533 -␣
↪val_precision: 0.9533 - val_recall: 0.9533
Epoch 10/30
42/42 [==============================] - 39s 930ms/step - loss: 0.1659 - accuracy: 0.
↪9382 - precision: 0.9382 - recall: 0.9382 - val_loss: 0.1206 - val_accuracy: 0.9667 -␣
↪val_precision: 0.9667 - val_recall: 0.9667
Epoch 11/30
42/42 [==============================] - 37s 896ms/step - loss: 0.1520 - accuracy: 0.
↪9494 - precision: 0.9494 - recall: 0.9494 - val_loss: 0.1295 - val_accuracy: 0.9600 -␣
↪val_precision: 0.9600 - val_recall: 0.9600
Epoch 12/30
42/42 [==============================] - 40s 964ms/step - loss: 0.1570 - accuracy: 0.
↪9479 - precision: 0.9479 - recall: 0.9479 - val_loss: 0.1130 - val_accuracy: 0.9600 -␣
↪val_precision: 0.9600 - val_recall: 0.9600
Epoch 13/30
```

(continues on next page)

(continued from previous page)

```
42/42 [==============================] - 40s 963ms/step - loss: 0.1440 - accuracy: 0.
↪9516 - precision: 0.9516 - recall: 0.9516 - val_loss: 0.1106 - val_accuracy: 0.9800 -␣
↪val_precision: 0.9800 - val_recall: 0.9800
Epoch 14/30
42/42 [==============================] - 40s 962ms/step - loss: 0.1413 - accuracy: 0.
↪9524 - precision: 0.9524 - recall: 0.9524 - val_loss: 0.1045 - val_accuracy: 0.9733 -␣
↪val_precision: 0.9733 - val_recall: 0.9733
Epoch 15/30
42/42 [==============================] - 38s 928ms/step - loss: 0.1528 - accuracy: 0.
↪9494 - precision: 0.9494 - recall: 0.9494 - val_loss: 0.1398 - val_accuracy: 0.9667 -␣
↪val_precision: 0.9667 - val_recall: 0.9667
Epoch 16/30
42/42 [==============================] - 39s 954ms/step - loss: 0.1319 - accuracy: 0.
↪9494 - precision: 0.9494 - recall: 0.9494 - val_loss: 0.1044 - val_accuracy: 0.9800 -␣
↪val_precision: 0.9800 - val_recall: 0.9800
Epoch 17/30
42/42 [==============================] - 40s 977ms/step - loss: 0.1409 - accuracy: 0.
↪9509 - precision: 0.9509 - recall: 0.9509 - val_loss: 0.1031 - val_accuracy: 0.9733 -␣
↪val_precision: 0.9733 - val_recall: 0.9733
Epoch 18/30
42/42 [==============================] - 40s 958ms/step - loss: 0.1509 - accuracy: 0.
↪9487 - precision: 0.9487 - recall: 0.9487 - val_loss: 0.1307 - val_accuracy: 0.9600 -␣
↪val_precision: 0.9600 - val_recall: 0.9600
Epoch 19/30
42/42 [==============================] - 37s 894ms/step - loss: 0.1427 - accuracy: 0.
↪9487 - precision: 0.9487 - recall: 0.9487 - val_loss: 0.0957 - val_accuracy: 0.9800 -␣
↪val_precision: 0.9800 - val_recall: 0.9800
Epoch 20/30
42/42 [==============================] - 39s 945ms/step - loss: 0.1364 - accuracy: 0.
↪9479 - precision: 0.9479 - recall: 0.9479 - val_loss: 0.1113 - val_accuracy: 0.9667 -␣
↪val_precision: 0.9667 - val_recall: 0.9667
Epoch 21/30
42/42 [==============================] - 38s 930ms/step - loss: 0.1402 - accuracy: 0.
↪9516 - precision: 0.9516 - recall: 0.9516 - val_loss: 0.1447 - val_accuracy: 0.9667 -␣
↪val_precision: 0.9667 - val_recall: 0.9667
Epoch 22/30
42/42 [==============================] - 40s 955ms/step - loss: 0.1229 - accuracy: 0.
↪9591 - precision: 0.9591 - recall: 0.9591 - val_loss: 0.1055 - val_accuracy: 0.9667 -␣
↪val_precision: 0.9667 - val_recall: 0.9667
Epoch 23/30
42/42 [==============================] - 40s 969ms/step - loss: 0.1334 - accuracy: 0.
↪9516 - precision: 0.9516 - recall: 0.9516 - val_loss: 0.0972 - val_accuracy: 0.9733 -␣
↪val_precision: 0.9733 - val_recall: 0.9733
Epoch 24/30
42/42 [==============================] - 41s 990ms/step - loss: 0.1255 - accuracy: 0.
↪9568 - precision: 0.9568 - recall: 0.9568 - val_loss: 0.0941 - val_accuracy: 0.9800 -␣
↪val_precision: 0.9800 - val_recall: 0.9800
Epoch 25/30
42/42 [==============================] - 40s 959ms/step - loss: 0.1286 - accuracy: 0.
↪9531 - precision: 0.9531 - recall: 0.9531 - val_loss: 0.0980 - val_accuracy: 0.9800 -␣
↪val_precision: 0.9800 - val_recall: 0.9800
Epoch 26/30
```

(continues on next page)

(continued from previous page)

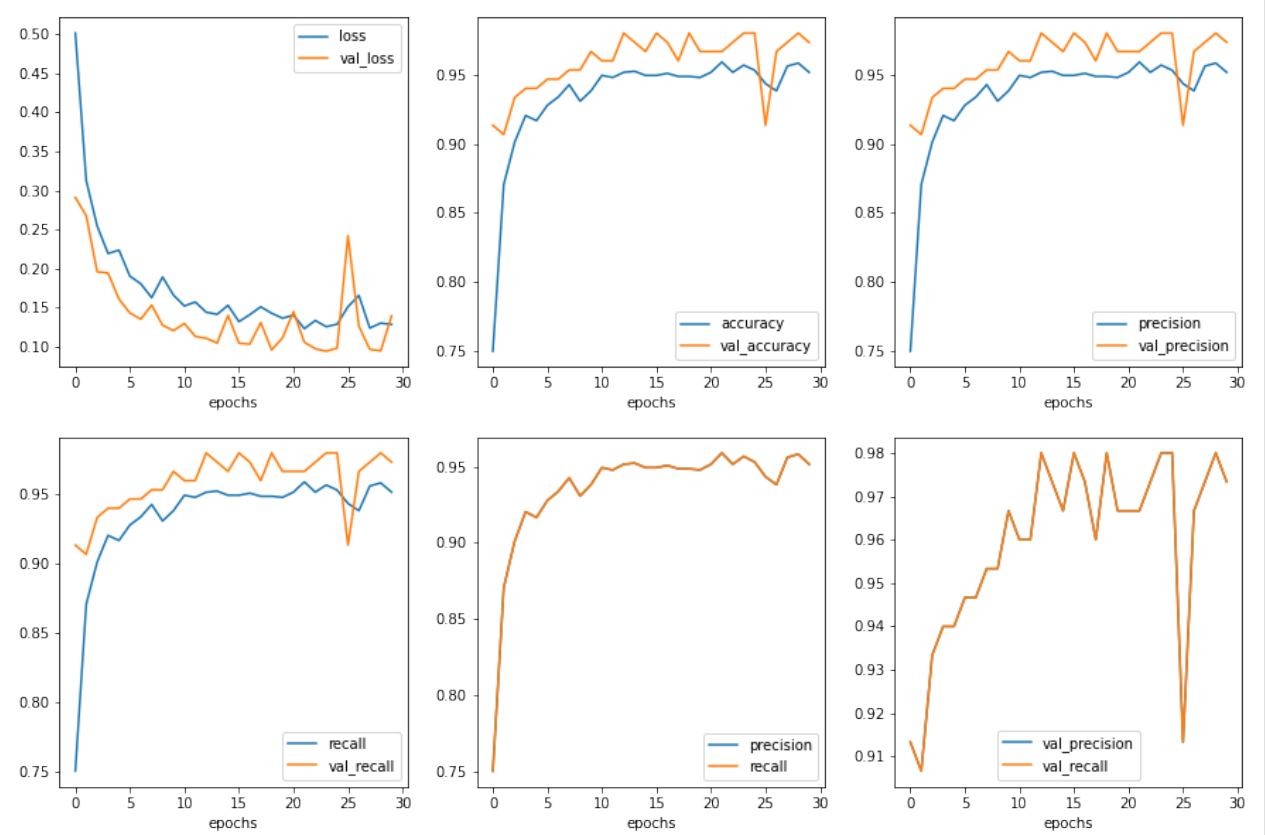
```
42/42 [==============================] - 38s 927ms/step - loss: 0.1511 - accuracy: 0.
↪9435 - precision: 0.9435 - recall: 0.9435 - val_loss: 0.2420 - val_accuracy: 0.9133 -␣
↪val_precision: 0.9133 - val_recall: 0.9133
Epoch 27/30
42/42 [==============================] - 37s 904ms/step - loss: 0.1655 - accuracy: 0.
↪9382 - precision: 0.9382 - recall: 0.9382 - val_loss: 0.1260 - val_accuracy: 0.9667 -␣
↪val_precision: 0.9667 - val_recall: 0.9667
Epoch 28/30
42/42 [==============================] - 38s 916ms/step - loss: 0.1237 - accuracy: 0.
↪9561 - precision: 0.9561 - recall: 0.9561 - val_loss: 0.0966 - val_accuracy: 0.9733 -␣
↪val_precision: 0.9733 - val_recall: 0.9733
Epoch 29/30
42/42 [==============================] - 38s 908ms/step - loss: 0.1300 - accuracy: 0.
↪9583 - precision: 0.9583 - recall: 0.9583 - val_loss: 0.0945 - val_accuracy: 0.9800 -␣
↪val_precision: 0.9800 - val_recall: 0.9800
Epoch 30/30
42/42 [==============================] - 39s 936ms/step - loss: 0.1284 - accuracy: 0.
↪9516 - precision: 0.9516 - recall: 0.9516 - val_loss: 0.1394 - val_accuracy: 0.9733 -␣
↪val_precision: 0.9733 - val_recall: 0.9733
```

[29]:
```
history_df = pd.DataFrame(history.history)
history_df['epochs'] = history.epoch
```

[30]:
```
history_df.head(3)
```

[30]:
```
       loss  accuracy  precision  ...  val_precision  val_recall  epochs
0  0.501539  0.750000   0.750000  ...       0.913333    0.913333       0
1  0.312725  0.870536   0.870536  ...       0.906667    0.906667       1
2  0.254531  0.901042   0.901042  ...       0.933333    0.933333       2

[3 rows x 9 columns]
```

[31]:
```
fig, ax = plt.subplots(2,3, figsize=(15,10))
history_df.plot(x='epochs', y=['loss', 'val_loss'], ax=ax[0][0])
history_df.plot(x='epochs', y=['accuracy', 'val_accuracy'], ax=ax[0][1])
history_df.plot(x='epochs', y=['precision','val_precision'], ax=ax[0][2])
history_df.plot(x='epochs', y=['recall','val_recall'], ax=ax[1][0])
history_df.plot(x='epochs', y=['precision','recall'], ax=ax[1][1])
history_df.plot(x='epochs', y=['val_precision','val_recall'], ax=ax[1][2])

plt.show()
```



[34]:
```
reviews = [
    'Enter a chance to win $5000, hurry up, offer valid until march 31, 2021',
    'You are awarded a SiPix Digital Camera! call 09061221061 from landline. Delivery␣
↪within 28days. T Cs Box177. M221BP. 2yr warranty. 150ppm. 16 . p p^Af3.99',
    'it to 80488. Your 500 free text messages are valid until 31 December 2005.',
    'Hey Sam, Are you coming for a cricket game tomorrow',
    "Why don't you wait 'til at least wednesday to see if you get your ."
]
preds = model.predict(reviews)
```

[36]:
```
reviews_df = pd.DataFrame(preds, columns=['spam','ham'])
reviews_df['message'] = reviews
reviews_df
```

[36]:

	spam	ham	message
0	0.724841	0.275158	Enter a chance to win $5000, hurry up, offer v...
1	0.985587	0.014413	You are awarded a SiPix Digital Camera! call 0...
2	0.512872	0.487128	it to 80488. Your 500 free text messages are v...
3	0.003944	0.996056	Hey Sam, Are you coming for a cricket game tom...
4	0.000492	0.999508	Why don't you wait 'til at least wednesday to ...

[45]:
```
predictions = np.argmax(model.predict(new_dataset.message), axis=1)
```

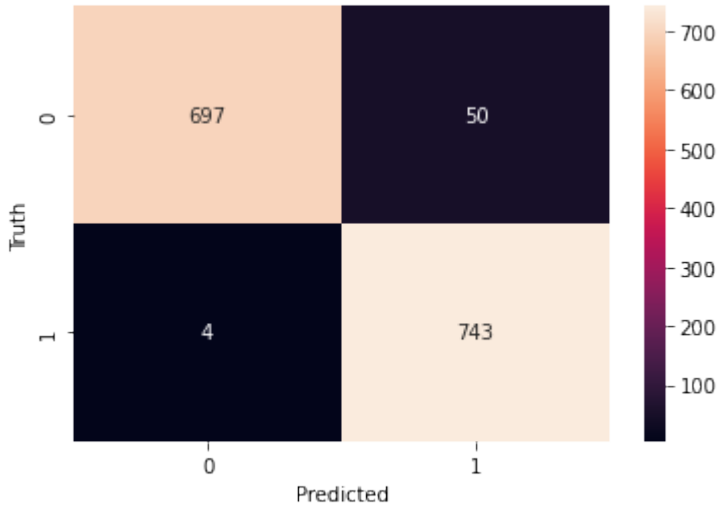
[47]:
```
original = np.argmax(new_dataset[['spam','ham']].values, axis=1)
```

1 is ham and 0 is spam

[53]:
```
from sklearn.metrics import confusion_matrix
import seaborn as sns
```

[58]:
```
cm = confusion_matrix(original, predictions)
sns.heatmap(cm, annot=True, fmt='d')
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

[58]:
```
Text(33.0, 0.5, 'Truth')
```



CHAPTER

TWELVE

COMMON FREQUENCY DISTRIBUTION METHODS

Method	Discription
fdist = nltk.FreqDist(text)	freq. dist. object
fdist.pprint()	print
fdist[‘exmple’]	get count
fdist.freq(‘example’)	get freq
fdist.N()	Total number of samples
fdist.keys()	keys in desc order of freq
for text in fdist	iterate
fdist.max()	key with max freq
fdist.tabulate()	tabulate
fdist.plot()	plot of freq dist
fdist.plot(cumulative=True)	cumulative plot of freq dist
fdist1 < fdist2	compare

1. tokenize
2. FreqDist
3. findall
4. pprint
5. freq
6. plot
7. Text Corpora / Corpus
8. pretty table

[12]:
```
## frequency distribution
fdist = nltk.FreqDist(text1)

## print([(text,fdist[text]) for text in fdist])
```

[13]:
```
fdist.pprint()
```

```
FreqDist({',': 18713, 'the': 13721, '.': 6862, 'of': 6536, 'and': 6024, 'a': 4569, 'to': 
↪4542, ';': 4072, 'in': 3916, 'that': 2982, ...})
```

[14]:
```
fdist['Sunday']
```

[14]:
```
7
```

[15]:
```
fdist.freq('Sunday')
```

[15]:
```
2.683853553613809e-05
```

[16]:
```
fdist.N()
```

[16]:
```
260819
```

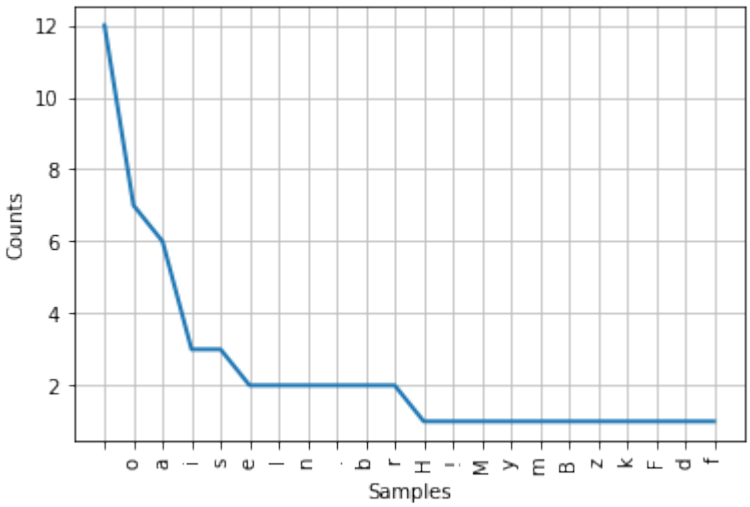
[17]:
```
## fdist.keys()
```

[18]:
```
fdist.max()
```

[18]:
```
','
```

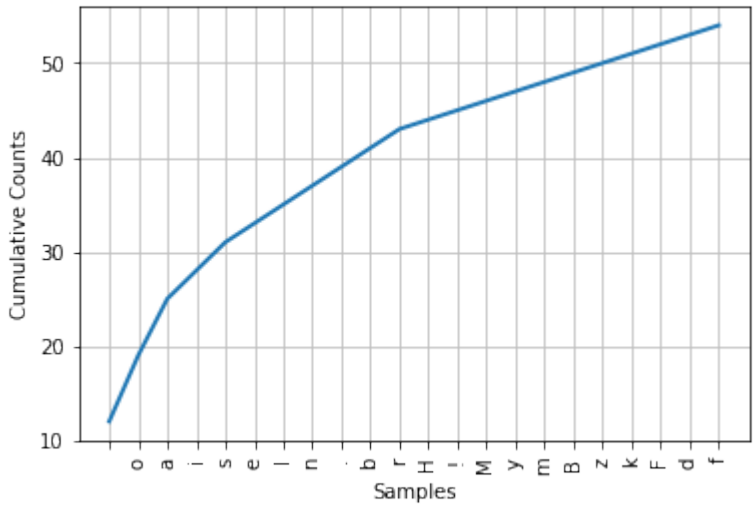
[19]:
```
## fdist.tabulate()
```

[20]:
```


fdist = nltk.FreqDist(text)
fdist.plot()
```



[20]:
```
<AxesSubplot:xlabel='Samples', ylabel='Counts'>
```

[21]:
```
fdist.plot(cumulative=True)
```



[21]:
```
<AxesSubplot:xlabel='Samples', ylabel='Cumulative Counts'>
```

[22]:
```
## compare
## fdist1 < fdist2
```

CHAPTER

THIRTEEN

CONDITIONAL FREQUENCY

```
Conditional Frequency is Frequency Distribution based on conditions.

CFD : Conditional Frequency Distribution
```

[26]:
```
cItems = [
    ('F','apple'),
    ('F','apple'),
    ('F','kiwi'),
    ('V','cabbage'),
    ('V','cabbage'),
    ('V','potato')
]
cfd = nltk.ConditionalFreqDist(cItems)
```

[27]:
```
cfd.conditions()
```

[27]:
```
['F', 'V']
```

[28]:
```
cfd['F']
```

[28]:
```
FreqDist({'apple': 2, 'kiwi': 1})
```

[29]:
```
cfd['V']
```

[29]:
```
FreqDist({'cabbage': 2, 'potato': 1})
```

Method	Description
cfdist = ConditionalFreqDist(pairs)	create
cfdist.conditions()	show conditions
cfdist[condition]	freq distribution for the condition
cfdist[condition][sample]	freq for the given condition
cfdist.tabulate()	tabulate
cfdist.plot()	plot of freq dist
cfdist.plot(cumulative=True)	cumulative plot of freq dist
cfdist1 < cfdist2	compare

CHAPTER

FOURTEEN

PROJECT : SPAM SMS DATA ANALYTICS

WORK IN PROGRESS

[2]:
```
import pandas as pd
import numpy as np
import csv

from sklearn.model_selection import StratifiedShuffleSplit

from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier,␣
↪GradientBoostingClassifier
from sklearn.linear_model import SGDClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC,LinearSVC
from sklearn.multiclass import OneVsRestClassifier


from sklearn.metrics import classification_report
```

[3]:
```
df = pd.read_csv("./SMSSpamCollection.csv", sep='\t', quoting=csv.QUOTE_NONE, names=[
↪"label", "message"])

df.head()
```

[3]:
```
  label                                            message
0   ham  Go until jurong point, crazy.. Available only ...
1   ham                      Ok lar... Joking wif u oni...
2  spam  Free entry in 2 a wkly comp to win FA Cup fina...
3   ham  U dun say so early hor... U c already then say...
4   ham  Nah I don't think he goes to usf, he lives aro...
```

[6]:
```
df.label.value_counts()
```

[6]:
```
ham     4827
spam     747
Name: label, dtype: int64
```

14.1 Loading NLTK (Natural Language Toolkit)

[ ]:
```
import nltk
nltk.download('all')

from nltk.tokenize import word_tokenize
from nltk.stem.wordnet import WordNetLemmatizer
from nltk.corpus import stopwords
```

14.2 Tokenization

[8]:
```
def splitIntoTokens(text):
    text = text.lower()
    tokens = word_tokenize(text)
    return tokens
```

[ ]:
```
df['tokenized_message'] = df['message'].apply(splitIntoTokens)
```

14.3 Lemmatization (convert a word into its base form)

[9]:
```

def getLemmas(tokens):
    lemmas = []
    lemmatizer = WordNetLemmatizer()
    for token in tokens:
        lemmas.append(lemmatizer.lemmatize(token))
    return lemmas
```

[ ]:
```

df['lemmatized_message'] = df['tokenized_message'].apply(getLemmas)
```

14.4 Removing Stop Words

[10]:
```
stopWords = set(stopwords.words('english'))
def removeStopWords(lemmas):
    filteredSentence = []
    filteredSentence = ' '.join([word for word in lemmas if word not in stopWords])
    return filteredSentence
```

[ ]:
```

df['filtered_message'] = df['lemmatized_message'].apply(removeStopWords)
```

14.5 TFIDF Matrix

- The Term Document Matrix (TDM) is a matrix that contains the frequency of occurrence of terms in a collection of documents.
- In a Term Frequency Inverse Document Frequency (TFIDF) matrix, the term importance is expressed by Inverse Document Frequency (IDF)
- IDF diminishes the weight of the most commonly occurring words and increases the weightage of rare words.

[11]:
```
from sklearn.feature_extraction.text import TfidfVectorizer

tfidfVectorizer = TfidfVectorizer(
    ngram_range=(1, 2),
    min_df = 1/df.shape[0],
    max_df = 0.7
)
```

[12]:
```


tfidfModel = tfidfVectorizer.fit(df['filtered_message'])
```

[13]:
```
X = tfidfModel.transform(df['filtered_message']).toarray()
y = df['label'].values

print(X.shape, y.shape)
```

```
(5574, 40373) (5574,)
```

14.6 Stratified Shuffle Sampling

[87]:
```
data_gen = StratifiedShuffleSplit(n_splits=1, test_size=0.3, random_state=7)
```

14.7 Models Pool

[ ]:
```
classifiers = [
    DecisionTreeClassifier(),
    GaussianNB(),
    SGDClassifier(loss='modified_huber', shuffle=True),
    SVC(kernel="linear", C=0.025),
    KNeighborsClassifier(),
    OneVsRestClassifier(LinearSVC()),
    RandomForestClassifier(max_depth=5, n_estimators=100, max_features=10),
    AdaBoostClassifier(),
]

models_params = []
for model in classifiers:
```

(continues on next page)

(continued from previous page)

```
    for train_index, test_index in data_gen.split(X,y):
        X_train, X_test = X[train_index], X[test_index]
        y_train, y_test = y[train_index], y[test_index]
        model.fit(X_train, y_train)

        y_pred = model.predict(X_test)
        report = classification_report(y_test, y_pred, output_dict=True)

        model_report = report['spam']
        model_report.update({
            'model' : type(model).__name__,
            'accuracy' : report['accuracy']
        })

        models_params.append(model_report)
```

[96]:
```
report_df = pd.DataFrame(models_params)

report_df
```

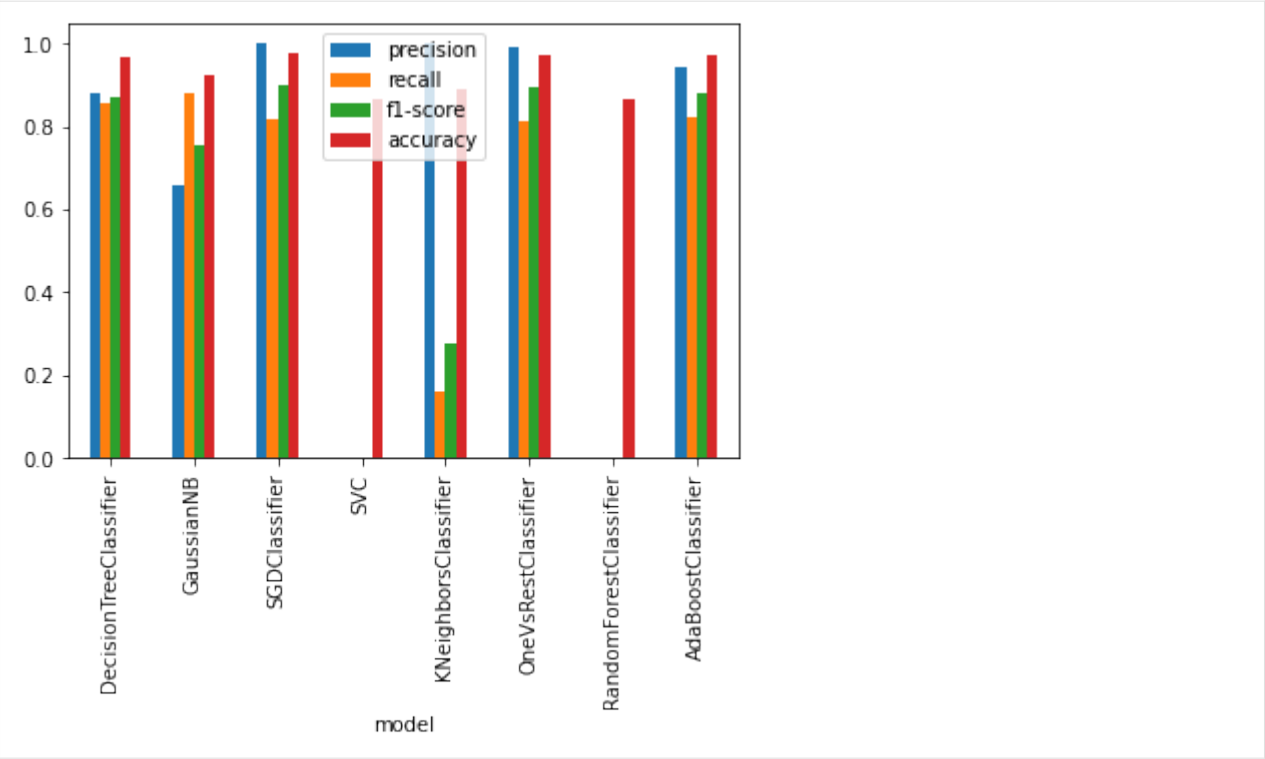
[96]:

	precision	recall	f1-score	support	model	accuracy
0	0.880734	0.857143	0.868778	224	DecisionTreeClassifier	0.965332
1	0.658863	0.879464	0.753346	224	GaussianNB	0.922893
2	1.000000	0.816964	0.899263	224	SGDClassifier	0.975493
3	0.000000	0.000000	0.000000	224	SVC	0.866109
4	1.000000	0.160714	0.276923	224	KNeighborsClassifier	0.887627
5	0.989130	0.812500	0.892157	224	OneVsRestClassifier	0.973700
6	0.000000	0.000000	0.000000	224	RandomForestClassifier	0.866109
7	0.943590	0.821429	0.878282	224	AdaBoostClassifier	0.969516

[100]:
```
del report_df['support']
```

[101]:
```
report_df.plot.bar(x='model')
```

[101]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9f850c3b10>
```


precision
recall
f1-score
accuracy
1.0
0.8
0.6
0.4
0.2
0.0
DecisionTreeClassifier
GaussianNB
SGDClassifier
SVC
KNeighborsClassifier
OneVsRestClassifier
RandomForestClassifier
AdaBoostClassifier
model

CHAPTER

FIFTEEN

INDICES AND TABLES

- genindex
- modindex
- search

INDEX