```
import nltk
nltk.download('gutenberg')
nltk.download('punkt')
import re
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize
nltk.download('omw-1.4')
nltk.download('wordnet')
import pandas as pd
import string
```

```
[nltk_data] Downloading package gutenberg to /root/nltk_data...
[nltk_data]   Unzipping corpora/gutenberg.zip.
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
[nltk_data] Downloading package wordnet to /root/nltk_data...
```

```
nltk.corpus.gutenberg.fileids()
```

```
['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']
```

```
#preprocessing
#grabbing the Shakespeare works

caesar = nltk.corpus.gutenberg.raw('shakespeare-caesar.txt')
hamlet = nltk.corpus.gutenberg.raw('shakespeare-hamlet.txt')
mcbeth = nltk.corpus.gutenberg.raw('shakespeare-macbeth.txt')

#lowercasing everything

c_lower = caesar.lower()
h_lower = hamlet.lower()
m_lower = mcbeth.lower()

#tokenizing the text blobs

from nltk.tokenize import sent_tokenize
caesar_tok = sent_tokenize(c_lower)
hamlet_tok = sent_tokenize(h_lower)
mcbeth_tok = sent_tokenize(m_lower)
```

```
def remove_punct(str_list):
  no_punct = []

  for sent in str_list:
    sent = re.sub('[%s]' % re.escape(string.punctuation), '', sent)

    no_punct.append(sent)
  return no_punct
```

```
caesar_tok = remove_punct(caesar_tok)
hamlet_tok = remove_punct(hamlet_tok)
mcbeth_tok = remove_punct(mcbeth_tok)

#Make Shakespeare plays into dataframes
import pandas as pd
ceasar_df = pd.DataFrame(caesar_tok)
```

```
author = 'Shakespeare'
ceasar_df['Author'] = author
ceasar_df = ceasar_df.rename(columns={0: 'Text'})

hamlet_df = pd.DataFrame(hamlet_tok)
hamlet_df['Author'] = author
hamlet_df = hamlet_df.rename(columns={0: 'Text'})

mcbeth_df = pd.DataFrame(mcbeth_tok)
mcbeth_df['Author'] = author
mcbeth_df = mcbeth_df.rename(columns={0: 'Text'})

#creating a non-Shakeaspeare corpus

bible = nltk.corpus.gutenberg.raw('bible-kjv.txt')
milton = nltk.corpus.gutenberg.raw('milton-paradise.txt')
blake = nltk.corpus.gutenberg.raw('blake-poems.txt')

b_lower = bible.lower()
mil_lower = milton.lower()
bl_lower = blake.lower()

#tokeninzing the text-blob

bible_tok = sent_tokenize(b_lower)
milton_tok = sent_tokenize(mil_lower)
blake_tok = sent_tokenize(bl_lower)

bible_tok = remove_punct(bible_tok)
milton_tok = remove_punct(milton_tok)
blake_tok = remove_punct(blake_tok)

#making the non-Shakespeare all one dataframe.

author_2 = 'Other'

bible_df = pd.DataFrame(bible_tok)
bible_df['Author'] = author_2
bible_df = bible_df.rename(columns={0: 'Text'})

milton_df = pd.DataFrame(milton_tok)
milton_df['Author'] = author_2
milton_df = milton_df.rename(columns={0: 'Text'})

blake_df = pd.DataFrame(blake_tok)
blake_df['Author'] = author_2
blake_df = blake_df.rename(columns={0: 'Text'})

noshakespear = bible_df.append([milton_df, blake_df], ignore_index=True)

#subsampling No Shakespeare

import random
random.seed(1)

noshakes_sub = noshakespear.sample(n = 3500, random_state=1)

#composite data frames

no_ceasar = hamlet_df.append([noshakes_sub, mcbeth_df], ignore_index=True)
no_hamlet = ceasar_df.append([noshakes_sub, mcbeth_df], ignore_index=True)
no_mcbeth = hamlet_df.append([noshakes_sub, ceasar_df], ignore_index=True)
```

```
<ipython-input-5-d1a1d18a8dd7>:56: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future vers
  noshakespear = bible_df.append([milton_df, blake_df], ignore_index=True)
<ipython-input-5-d1a1d18a8dd7>:67: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future vers
  no_ceasar = hamlet_df.append([noshakes_sub, mcbeth_df], ignore_index=True)
<ipython-input-5-d1a1d18a8dd7>:68: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future vers
  no_hamlet = ceasar_df.append([noshakes_sub, mcbeth_df], ignore_index=True)
<ipython-input-5-d1a1d18a8dd7>:69: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future vers
  no_mcbeth = hamlet_df.append([noshakes_sub, ceasar_df], ignore_index=True)
```

```
no_ceasar['Author'] = no_ceasar['Author'].map({'Shakespeare':1, 'Other':0})
print(no_ceasar)
```

```
                                                  Text  Author
0     the tragedie of hamlet by william shakespeare ...       1
1                                        scoena prima       1
2            enter barnardo and francisco two centinels       1
3                                          barnardo       1
4                                       whos there       1
...                                                ...     ...
7268  this and what need full else\nthat calls vpon ...       1
7269                                        flourish       1
7270                                      exeunt omnes       1
7271                                             finis       1
7272                          the tragedie of macbeth       1

[7273 rows x 2 columns]
```

```
#Test Train - The old Fashioned Way

from sklearn.model_selection import train_test_split

c_X = no_ceasar.Text
c_y = no_ceasar.Author
print(c_X.shape, c_y.shape)

c_X_train, c_X_test, c_y_train, c_y_test = train_test_split(c_X, c_y, random_state=1)
print(c_X_train.shape, c_y_train.shape)
print(c_X_test.shape, c_y_test.shape)
```

```
(7273,) (7273,)
(5454,) (5454,)
(1819,) (1819,)
```

```
no_ceasar_labels = no_ceasar['Author']
no_ceasar_text = no_ceasar['Text']
```

```
# pre processing

MAX_SEQUENCE_LENGTH = 300
MAX_NUM_WORDS = 20000
EMBEDDING_DIM = 100
VALIDATION_SPLIT = 0.3
```

```
from keras.preprocessing.text import Tokenizer
```

```
tokenizer = Tokenizer(num_words=MAX_NUM_WORDS)
tokenizer.fit_on_texts(c_X_train)
train_sequences = tokenizer.texts_to_sequences(c_X_train) #Converting text to a vector of word indexes
test_sequences = tokenizer.texts_to_sequences(c_X_test)
word_index = tokenizer.word_index
print('Found %s unique tokens.' % len(word_index))
```

```
Found 11460 unique tokens.
```

```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
le.fit(c_y_train)
train_labels = le.transform(c_y_train)
test_labels = le.transform(c_y_test)
```

```
from tensorflow.keras.preprocessing.sequence import pad_sequences
from keras.utils import to_categorical
import numpy as np

#Converting this to sequences to be fed into neural network. Max seq. len is 1000 as set earlier
 #initial padding of 0s, until vector is of size MAX_SEQUENCE_LENGTH
trainvalid_data = pad_sequences(train_sequences, maxlen=MAX_SEQUENCE_LENGTH)
test_data = pad_sequences(test_sequences, maxlen=MAX_SEQUENCE_LENGTH)
trainvalid_labels = to_categorical(train_labels)

test_labels = to_categorical(np.asarray(test_labels), num_classes= trainvalid_labels.shape[1])

# split the training data into a training set and a validation set
indices = np.arange(trainvalid_data.shape[0])
```

```
np.random.shuffle(indices)
trainvalid_data = trainvalid_data[indices]
trainvalid_labels = trainvalid_labels[indices]
num_validation_samples = int(VALIDATION_SPLIT * trainvalid_data.shape[0])
x_train = trainvalid_data[:-num_validation_samples]
y_train = trainvalid_labels[:-num_validation_samples]
x_val = trainvalid_data[-num_validation_samples:]
y_val = trainvalid_labels[-num_validation_samples:]
#This is the data we will use for CNN training
print('Splitting the train data into train and valid is done')
```

```
Splitting the train data into train and valid is done
```

```
import os

BASE_DIR = os.getcwd()
GLOVE_DIR = os.path.join(BASE_DIR, 'glove.6B')
```

```
from keras.layers import Conv1D, MaxPooling1D, Embedding
from keras.initializers import Constant

#embedding matrix

print('Preparing embedding matrix.')
# first, build index mapping words in the embeddings set
# to their embedding vector
embeddings_index = {}

with open( 'glove.6B.100d.txt',encoding="utf8") as f:
    for line in f:
        values = line.split()
        word = values[0]
        coefs = np.asarray(values[1:], dtype='float32')
        embeddings_index[word] = coefs

print('Found %s word vectors in Glove embeddings.' % len(embeddings_index))
#print(embeddings_index["google"])

# prepare embedding matrix - rows are the words from word_index, columns are the embeddings of that word from glove.
num_words = min(MAX_NUM_WORDS, len(word_index)) + 1
embedding_matrix = np.zeros((num_words, EMBEDDING_DIM))
for word, i in word_index.items():
    if i > MAX_NUM_WORDS:
        continue
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        # words not found in embedding index will be all-zeros.
        embedding_matrix[i] = embedding_vector

# load these pre-trained word embeddings into an Embedding layer
# note that we set trainable = False so as to keep the embeddings fixed
embedding_layer = Embedding(num_words,
                            EMBEDDING_DIM,
                            embeddings_initializer=Constant(embedding_matrix),
                            input_length=MAX_SEQUENCE_LENGTH,
                            trainable=False)
print("Preparing of embedding matrix is done")
```

```
Preparing embedding matrix.
Found 2449 word vectors in Glove embeddings.
Preparing of embedding matrix is done
```

```
from keras.models import Model, Sequential
from keras.layers import Dense, Input, GlobalMaxPooling1D

cnnmodel = Sequential()
cnnmodel.add(embedding_layer)
cnnmodel.add(Conv1D(128, 5, activation='relu'))
cnnmodel.add(MaxPooling1D(5))
cnnmodel.add(Conv1D(128, 5, activation='relu'))
cnnmodel.add(MaxPooling1D(5))
cnnmodel.add(Conv1D(128, 5, activation='relu'))
cnnmodel.add(GlobalMaxPooling1D())
cnnmodel.add(Dense(128, activation='relu'))
cnnmodel.add(Dense(len(trainvalid_labels[0]), activation='softmax'))
```

```
14
15 cnnmodel.compile(loss='categorical_crossentropy',
16               optimizer='rmsprop',
17               metrics=['acc'])
18
19 cnnmodel.summary()
20
21 #you can replace softmax by hyperbolic
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 300, 100)          1146100

 conv1d (Conv1D)             (None, 296, 128)          64128

 max_pooling1d (MaxPooling1D  (None, 59, 128)          0
 )

 conv1d_1 (Conv1D)           (None, 55, 128)           82048

 max_pooling1d_1 (MaxPooling  (None, 11, 128)          0
 1D)

 conv1d_2 (Conv1D)           (None, 7, 128)            82048

 global_max_pooling1d (Globa  (None, 128)              0
 lMaxPooling1D)

 dense (Dense)               (None, 128)               16512

 dense_1 (Dense)             (None, 2)                 258

=================================================================
Total params: 1,391,094
Trainable params: 244,994
Non-trainable params: 1,146,100
_________________________________________________________________
```

```
1 #Train the model. Tune to validation set.
2 cnnmodel.fit(x_train, y_train,
3           batch_size=128,
4           epochs=3, validation_data=(x_val, y_val))
5 #Evaluate on test set:
6 score, acc = cnnmodel.evaluate(test_data, test_labels)
7 print('Test accuracy with CNN:', acc) #consider decreasing batch size and increasing epoch size
```

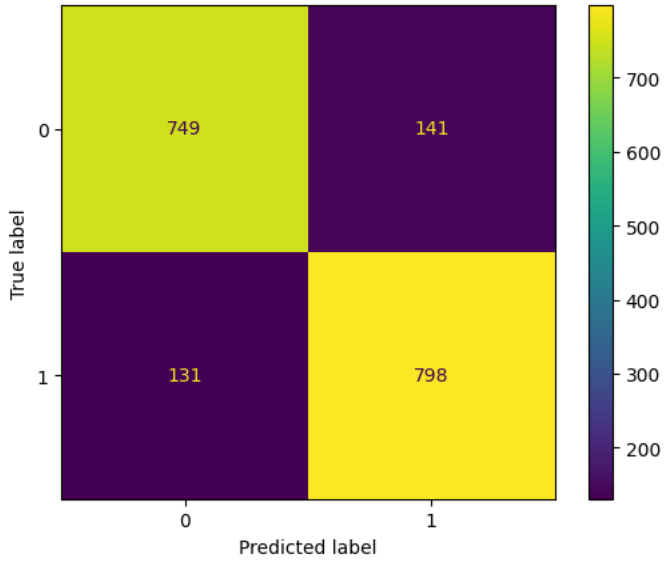
```
Epoch 1/3
30/30 [==============================] - 25s 807ms/step - loss: 0.5859 - acc: 0.7391 - val_loss: 0.4406 - val_acc: 0.8081
Epoch 2/3
30/30 [==============================] - 20s 663ms/step - loss: 0.4406 - acc: 0.8070 - val_loss: 0.3986 - val_acc: 0.8417
Epoch 3/3
30/30 [==============================] - 15s 500ms/step - loss: 0.3783 - acc: 0.8363 - val_loss: 0.3378 - val_acc: 0.8570
57/57 [==============================] - 4s 62ms/step - loss: 0.3408 - acc: 0.8505
Test accuracy with CNN: 0.8504672646522522
```

Confusion Matrix

```
1 #x_val, y_val, test_data, test_labels
2
3 from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
4
5 class_labels = [0, 1]
6
7 Y_test_preds = cnnmodel.predict(test_data)
8 rounded_labels = np.argmax(test_labels, axis=1)
9 Y_test_preds = np.argmax(Y_test_preds, axis=1)
10
11 print("Test Accuracy : {}".format(accuracy_score(rounded_labels, Y_test_preds)))
12 print("\nConfusion Matrix : ")
13 print(confusion_matrix(rounded_labels, Y_test_preds, labels=[0, 1]))
14 print("\nClassification Report :")
15 #print(classification_report(rounded_labels, Y_test_preds, target_names=class_labels))
16
17 #https://stackoverflow.com/questions/54589669/confusion-matrix-error-classification-metrics-cant-handle-a-mix-of-multilabel
18 #https://coderzcolumn.com/tutorials/artificial-intelligence/lime-explain-keras-image-classification-network-predictions
```

```
57/57 [==============================] - 2s 37ms/step
Test Accuracy : 0.8504672897196262

Confusion Matrix :
[[749 141]
 [131 798]]

Classification Report :
```

```
from sklearn.metrics import ConfusionMatrixDisplay
import matplotlib.pyplot as plt

cm = confusion_matrix(rounded_labels, Y_test_preds, labels=class_labels)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                              display_labels=class_labels)
disp.plot()
plt.show()

#https://stackoverflow.com/questions/54589669/confusion-matrix-error-classification-metrics-cant-handle-a-mix-of-multilabel
```




Predicting with CNN

```
#Predicting with a CNN

#experimental set
#caesar_experiment = vect.transform(caesar_tok)

caesar_tok = tokenizer.texts_to_sequences(caesar_tok)
ceasar = pad_sequences(caesar_tok, maxlen=MAX_SEQUENCE_LENGTH)


#https://medium.com/analytics-vidhya/multi-class-classification-using-cnn-for-custom-dataset-7759865bd19
#https://towardsdatascience.com/how-to-predict-an-image-with-keras-ca97d9cd4817
#https://www.analyticsvidhya.com/blog/2021/12/intent-classification-with-convolutional-neural-networks/
```

```
is_ceasar = cnnmodel.predict(ceasar)
```

```
49/49 [==============================] - 2s 36ms/step
```

```
print(is_ceasar)
```

```
[[0.6988547  0.30114526]
 [0.11961149 0.88038856]
 [0.17559522 0.8244048 ]
 ...
 [0.11961149 0.88038856]
 [0.11961149 0.88038856]
 [0.21503216 0.7849678 ]]
```

```
#DF Analysis
```

```
ceasar_df['Author'] = author
ceasar_df = ceasar_df.rename(columns={0: 'Text'})

is_ceasar_df = pd.DataFrame(is_ceasar)
is_ceasar_df['Text'] = ceasar_df['Text']

is_ceasar_df = is_ceasar_df.rename(columns={0: 'Other'})
is_ceasar_df = is_ceasar_df.rename(columns={1: 'Shakespeare'})
```

```
def count_if_1(series):
  count = 0

  for cell in series:
    if cell >= 0.5:
      count += 1
  return count
```

```
count_if_1(is_ceasar_df['Shakespeare'])
```

```
1380
```

```
is_ceasar_df
```

| | Other | Shakespeare | Text |
|---|---|---|---|
| **0** | 0.698855 | 0.301145 | the tragedie of julius caesar by william shake... |
| **1** | 0.119611 | 0.880389 | scoena prima |
| **2** | 0.175595 | 0.824405 | enter flauius murellus and certaine commoners ... |
| **3** | 0.119611 | 0.880389 | flauius |
| **4** | 0.057073 | 0.942927 | hence home you idle creatures get you home\nis... |
| **...** | ... | ... | ... |
| **1547** | 0.177149 | 0.822851 | within my tent his bones to night shall ly\nmo... |
| **1548** | 0.119611 | 0.880389 | exeunt |
| **1549** | 0.119611 | 0.880389 | omnes |
| **1550** | 0.119611 | 0.880389 | finis |
| **1551** | 0.215032 | 0.784968 | the tragedie of ivlivs caesar |

1552 rows × 3 columns