```
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_hamlet['Author'] = no_hamlet['Author'].map({'Shakespeare':1, 'Other':0})
no_mcbeth['Author'] = no_mcbeth['Author'].map({'Shakespeare':1, 'Other':0})
```

```
#Train/Test and Vectorizing  for Hamlet
from sklearn.model_selection import train_test_split


h_X = no_hamlet.Text
h_y = no_hamlet.Author
print(h_X.shape, h_y.shape)

h_X_train, h_X_test, h_y_train, h_y_test = train_test_split(h_X, h_y, random_state=1)
print(h_X_train.shape, h_y_train.shape)
print(h_X_test.shape, h_y_test.shape)

#Train/Test and Vectorizing for McBeth

#m_X = no_mcbeth.Text
#m_y = no_mcbeth.Author
#print(m_X.shape, m_y.shape)

#m_X_train, m_X_test, m_y_train, m_y_test = train_test_split(m_X, m_y, random_state=1)
#print(m_X_train.shape, m_y_train.shape)
#print(m_X_test.shape, m_y_test.shape)
```

```
(6472,) (6472,)
(4854,) (4854,)
(1618,) (1618,)
(7405,) (7405,)
(5553,) (5553,)
(1852,) (1852,)
```

```
# pre processing

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

```
from keras.preprocessing.text import Tokenizer

#Hamlet
tokenizer_h = Tokenizer(num_words=MAX_NUM_WORDS)
tokenizer_h.fit_on_texts(h_X_train)
train_sequences_h = tokenizer_h.texts_to_sequences(h_X_train) #Converting text to a vector of word indexes
test_sequences_h = tokenizer_h.texts_to_sequences(h_X_test)
word_index_h = tokenizer_h.word_index
print('Found %s unique tokens.' % len(word_index_h))

#Macbeth
#tokenizer_m = Tokenizer(num_words=MAX_NUM_WORDS)
#tokenizer_m.fit_on_texts(m_X_train)
#train_sequences_m = tokenizer_m.texts_to_sequences(m_X_train) #Converting text to a vector of word indexes
#test_sequences_m = tokenizer_m.texts_to_sequences(m_X_test)
#word_index_m = tokenizer_m.word_index
#print('Found %s unique tokens.' % len(word_index_m))
```

```
Found 10471 unique tokens.
Found 11156 unique tokens.
```

```
from sklearn.preprocessing import LabelEncoder

#Hamlet
le = LabelEncoder()
le.fit(h_y_train)
train_labels_h = le.transform(h_y_train)
test_labels_h = le.transform(h_y_test)

#Macbeth
#le = LabelEncoder()
#le.fit(m_y_train)
#train_labels_m = le.transform(m_y_train)
#test_labels_m = le.transform(m_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_h = pad_sequences(train_sequences_h, maxlen=MAX_SEQUENCE_LENGTH)
test_data_h = pad_sequences(test_sequences_h, maxlen=MAX_SEQUENCE_LENGTH)
trainvalid_labels_h = to_categorical(train_labels_h)

test_labels_h = to_categorical(np.asarray(test_labels_h), num_classes= trainvalid_labels_h.shape[1])

# split the training data into a training set and a validation set
indices_h = np.arange(trainvalid_data_h.shape[0])
np.random.shuffle(indices_h)
trainvalid_data_h = trainvalid_data_h[indices_h]
trainvalid_labels_h = trainvalid_labels_h[indices_h]
num_validation_samples_h = int(VALIDATION_SPLIT * trainvalid_data_h.shape[0])
x_train_h = trainvalid_data_h[:-num_validation_samples_h]
y_train_h = trainvalid_labels_h[:-num_validation_samples_h]
x_val_h = trainvalid_data_h[-num_validation_samples_h:]
y_val_h = trainvalid_labels_h[-num_validation_samples_h:]
#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
```

```
#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_m = pad_sequences(train_sequences_m, maxlen=MAX_SEQUENCE_LENGTH)
#test_data_m = pad_sequences(test_sequences_m, maxlen=MAX_SEQUENCE_LENGTH)
#trainvalid_labels_m = to_categorical(train_labels_m)

#test_labels_m = to_categorical(np.asarray(test_labels_m), num_classes= trainvalid_labels_m.shape[1])

# split the training data into a training set and a validation set
#indices_m = np.arange(trainvalid_data_m.shape[0])
#np.random.shuffle(indices_m)
#trainvalid_data_m = trainvalid_data_m[indices_m]
#trainvalid_labels_m = trainvalid_labels_m[indices_m]
#num_validation_samples_m = int(VALIDATION_SPLIT * trainvalid_data_m.shape[0])
#x_train_m = trainvalid_data_m[:-num_validation_samples_m]
#y_train_m = trainvalid_labels_m[:-num_validation_samples_m]
#x_val_m = trainvalid_data_m[-num_validation_samples_m:]
#y_val_m = trainvalid_labels_m[-num_validation_samples_m:]
#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

#Hamlet
#embedding matrix

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

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_h[word] = coefs

print('Found %s word vectors in Glove embeddings.' % len(embeddings_index_h))
#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_h = min(MAX_NUM_WORDS, len(word_index_h)) + 1
embedding_matrix_h = np.zeros((num_words_h, EMBEDDING_DIM))
```

```
for word, i in word_index_h.items():
    if i > MAX_NUM_WORDS:
        continue
    embedding_vector_h = embeddings_index_h.get(word)
    if embedding_vector_h is not None:
        # words not found in embedding index will be all-zeros.
        embedding_matrix_h[i] = embedding_vector_h

# 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_h = Embedding(num_words_h,
                            EMBEDDING_DIM,
                            embeddings_initializer = Constant(embedding_matrix_h),
                            input_length=MAX_SEQUENCE_LENGTH,
                            trainable=False)
print("Preparing of embedding matrix is done")
```

```
Preparing embedding matrix.
Found 41545 word vectors in Glove embeddings.
Preparing of embedding matrix is done
```

```
#Macbeth
#embedding matrix

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

#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_m[word] = coefs

#print('Found %s word vectors in Glove embeddings.' % len(embeddings_index_m))
#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_m = min(MAX_NUM_WORDS, len(word_index_m)) + 1
#embedding_matrix_m = np.zeros((num_words_m, EMBEDDING_DIM))
#for word, i in word_index_m.items():
    #if i > MAX_NUM_WORDS:
        continue
    #embedding_vector_m = embeddings_index_m.get(word)
    #if embedding_vector_m is not None:
        # words not found in embedding index will be all-zeros.
        #embedding_matrix_m[i] = embedding_vector_m

# 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_m = Embedding(num_words_m,
                            #EMBEDDING_DIM,
                            #embeddings_initializer=Constant(embedding_matrix_m),
                            #input_length=MAX_SEQUENCE_LENGTH,
                            #trainable=False)
#print("Preparing of embedding matrix is done")
```

```
Preparing embedding matrix.
Found 46429 word vectors in Glove embeddings.
Preparing of embedding matrix is done
```

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

#hamlet
cnnmodel_h = Sequential()
cnnmodel_h.add(embedding_layer_h)
cnnmodel_h.add(Conv1D(128, 5, activation='relu'))
cnnmodel_h.add(MaxPooling1D(5))
cnnmodel_h.add(Conv1D(128, 5, activation='relu'))
cnnmodel_h.add(MaxPooling1D(5))
cnnmodel_h.add(Conv1D(128, 5, activation='relu'))
cnnmodel_h.add(GlobalMaxPooling1D())
cnnmodel_h.add(Dense(128, activation='relu'))
```

```
cnnmodel_h.add(Dense(len(trainvalid_labels_h[0]), activation='softmax'))

cnnmodel_h.compile(loss='categorical_crossentropy',
              optimizer='rmsprop',
              metrics=['acc'])

cnnmodel_h.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 300, 100)          1047200

 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: 1,292,194
Trainable params: 244,994
Non-trainable params: 1,047,200
_________________________________________________________________
```

```
#macbeth

#cnnmodel_m = Sequential()
#cnnmodel_m.add(embedding_layer_m)
#cnnmodel_m.add(Conv1D(128, 5, activation='relu'))
#cnnmodel_m.add(MaxPooling1D(5))
#cnnmodel_m.add(Conv1D(128, 5, activation='relu'))
#cnnmodel_m.add(MaxPooling1D(5))
#cnnmodel_m.add(Conv1D(128, 5, activation='relu'))
#cnnmodel_m.add(GlobalMaxPooling1D())
#cnnmodel_m.add(Dense(128, activation='relu'))
#cnnmodel_m.add(Dense(len(trainvalid_labels_m[0]), activation='softmax'))

#cnnmodel_m.compile(loss='categorical_crossentropy',
#              optimizer='rmsprop',
#              metrics=['acc'])

#cnnmodel_m.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, 300, 100)          1115700

 conv1d_3 (Conv1D)           (None, 296, 128)          64128

 max_pooling1d_2 (MaxPooling  (None, 59, 128)          0
 1D)

 conv1d_4 (Conv1D)           (None, 55, 128)           82048

 max_pooling1d_3 (MaxPooling  (None, 11, 128)          0
 1D)

 conv1d_5 (Conv1D)           (None, 7, 128)            82048

 global_max_pooling1d_1 (Glo  (None, 128)              0
 balMaxPooling1D)

 dense_2 (Dense)             (None, 128)               16512
```

```
 dense_3 (Dense)             (None, 2)                 258

=================================================================
Total params: 1,360,694
Trainable params: 244,994
Non-trainable params: 1,115,700
_________________________________________________________________
```

```
#Hamlet
#Train the model. Tune to validation set.
cnnmodel_h.fit(x_train_h, y_train_h,
          batch_size=128,
          epochs=3, validation_data=(x_val_h, y_val_h))
#Evaluate on test set:
score_h, acc_h = cnnmodel_h.evaluate(test_data_h, test_labels_h)
print('Test accuracy with CNN for No Hamlet Model:', acc_h)
```

```
Epoch 1/3
27/27 [==============================] - 25s 846ms/step - loss: 0.5742 - acc: 0.7322 - val_loss: 0.4365 - val_acc: 0.8166
Epoch 2/3
27/27 [==============================] - 13s 483ms/step - loss: 0.3505 - acc: 0.8670 - val_loss: 0.3773 - val_acc: 0.8599
Epoch 3/3
27/27 [==============================] - 13s 469ms/step - loss: 0.2944 - acc: 0.8896 - val_loss: 0.2425 - val_acc: 0.9032
51/51 [==============================] - 3s 56ms/step - loss: 0.2569 - acc: 0.8912
Test accuracy with CNN for No Hamlet Model: 0.8912237286567688
```

```
#Predicting with a CNN - Hamlet

#experimental set
hamlet_tok = tokenizer_h.texts_to_sequences(hamlet_tok)
hamlet_exp = pad_sequences(hamlet_tok, maxlen=MAX_SEQUENCE_LENGTH)
is_hamlet = cnnmodel_h.predict(hamlet_exp)
```

```
74/74 [==============================] - 3s 36ms/step
```

```
print(is_hamlet)

#what does this mean???
```

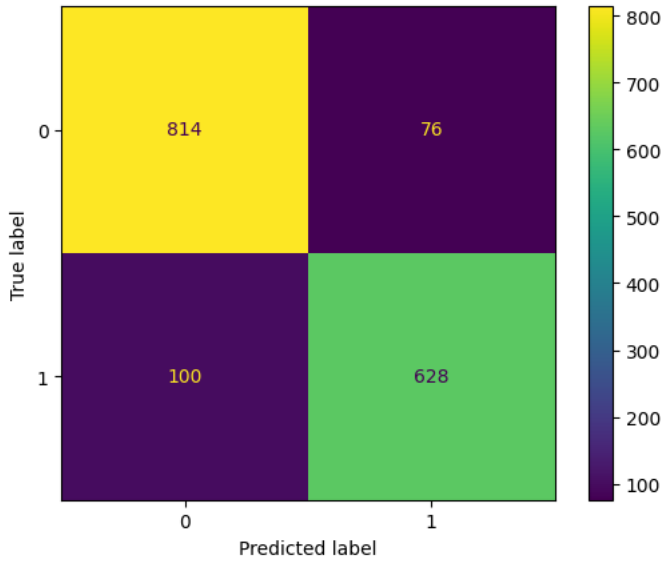
```
[[0.39699972 0.6030002 ]
 [0.19515932 0.8048407 ]
 [0.08507244 0.9149275 ]
 ...
 [0.73228854 0.26771143]
 [0.19515932 0.8048407 ]
 [0.1397319  0.86026806]]
```

```
#Confusion matrix test_data_h, test_labels_h

from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

class_labels = [0, 1]

Y_test_preds_h = cnnmodel_h.predict(test_data_h)
rounded_labels_h = np.argmax(test_labels_h, axis=1)
Y_test_preds_h = np.argmax(Y_test_preds_h, axis=1)

print("Test Accuracy : {}".format(accuracy_score(rounded_labels_h, Y_test_preds_h)))
print("\nConfusion Matrix : ")
print(confusion_matrix(rounded_labels_h, Y_test_preds_h, labels=[0, 1]))
print("\nClassification Report :")
```

```
51/51 [==============================] - 2s 36ms/step
Test Accuracy : 0.8912237330037083

Confusion Matrix :
[[814  76]
 [100 628]]

Classification Report :
```

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

cm = confusion_matrix(rounded_labels_h, Y_test_preds_h, labels=class_labels)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
```

```
                              display_labels=class_labels)
disp.plot()
plt.show()
```




```
#DF Analysis

def count_if_1(series):
  count = 0

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

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

is_hamlet_df = pd.DataFrame(is_hamlet)
is_hamlet_df['Text'] = hamlet_df['Text']

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

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

```
2027
```

```
is_hamlet_df
```

	Other	Shakespeare	Text
0	0.397000	0.603000	the tragedie of hamlet by william shakespeare ...
1	0.195159	0.804841	scoena prima
2	0.085072	0.914927	enter barnardo and francisco two centinels
3	0.195159	0.804841	barnardo
4	0.195159	0.804841	whos there
...	...	...	...
2348	0.046989	0.953011	go bid the souldiers shoote
2349	0.195159	0.804841	exeunt
2350	0.732289	0.267711	marching after the which a peale of ordenance ...
2351	0.195159	0.804841	finis
2352	0.139732	0.860268	the tragedie of hamlet prince of denmarke

2353 rows × 3 columns

```
MacBeth CNN
```

```
#Macbeth
Train the model. Tune to validation set.
cnnmodel_m.fit(x_train_m, y_train_m,
          #batch_size=128,
          #epochs=3, validation_data=(x_val_m, y_val_m))
Evaluate on test set:
score_m, acc_m = cnnmodel_m.evaluate(test_data_m, test_labels_m)
print('Test accuracy with CNN for No Macbeth:', acc_m)
```

```
#Macbeth

macbeth_tok = tokenizer_m.texts_to_sequences(mcbeth_tok)
macbeth_exp = pad_sequences(macbeth_tok, maxlen=MAX_SEQUENCE_LENGTH)
is_macbeth = cnnmodel_m.predict(macbeth_exp)
```

```
print(is_macbeth)
```

```
# Confusion matrix test_data_m, test_labels_m

class_labels = [0, 1]

Y_test_preds_m = cnnmodel_m.predict(test_data_m)
rounded_labels_m = np.argmax(test_labels_m, axis=1)
Y_test_preds_m = np.argmax(Y_test_preds_m, axis=1)

print("Test Accuracy : {}".format(accuracy_score(rounded_labels_m, Y_test_preds_m)))
print("\nConfusion Matrix : ")
print(confusion_matrix(rounded_labels_m, Y_test_preds_m, labels=[0, 1]))
print("\nClassification Report :")

cm_m = confusion_matrix(rounded_labels_m, Y_test_preds_m, labels=class_labels)
disp_m = ConfusionMatrixDisplay(confusion_matrix=cm,
                                display_labels=class_labels)
disp_m.plot()
plt.show()
```

```
# DF Analysis

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

is_macbeth_df = pd.DataFrame(is_macbeth)
is_macbeth_df['Text'] = mcbeth_df['Text']

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

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

```
is_macbeth_df
```

Predicting other Shakespeares

```
s1 = "To be, or not to be: that is the question."
s2 = "The lady doth protest too much, methinks"
s3 = "Good night, good night! Parting is such sweet sorrow."
s4 = "If you prick us do we not bleed? If you tickle us do we not laugh? If you poison us do we not die? And if you wrong us, shall we not
s5 = "All the world’s a stage, and all the men and women merely players. They have their exits and their entrances; And one man in his tim
s6 = "Romeo, Romeo! Wherefore art thou Romeo?"
s7 = "If music be the food of love play on."
s8 = "What’s in a name? A rose by any other name would smell as sweet."
s9 = "Shall I compare thee to a summer’s day? Thou art more lovely and more temperate."
s10 = "How sharper than a serpent’s tooth it is to have a thankless child!"
```

```
sentences = []

sentences.append([s1])
```

```
sentences.append([s2])
sentences.append([s3])
sentences.append([s4])
sentences.append([s5])
sentences.append([s6])
sentences.append([s7])
sentences.append([s8])
sentences.append([s9])
sentences.append([s10])
```

```
sentences
```

```
[['To be, or not to be: that is the question.'],
 ['The lady doth protest too much, methinks'],
 ['Good night, good night! Parting is such sweet sorrow.'],
 ['If you prick us do we not bleed? If you tickle us do we not laugh? If you poison us do we not die? And if you wrong us, shall we not revenge?'],
 ['All the world’s a stage, and all the men and women merely players. They have their exits and their entrances; And one man in his time plays many parts.'],
 ['Romeo, Romeo! Wherefore art thou Romeo?'],
 ['If music be the food of love play on.'],
 ['What’s in a name? A rose by any other name would smell as sweet.'],
 ['Shall I compare thee to a summer’s day? Thou art more lovely and more temperate.'],
 ['How sharper than a serpent’s tooth it is to have a thankless child!']]
```

```
sentences_df = pd.DataFrame(sentences)
```

```
sentences_df
sentences_df = sentences_df.rename(columns={0: 'Line'})
sentences_df
```

	Line
0	To be, or not to be: that is the question.
1	The lady doth protest too much, methinks
2	Good night, good night! Parting is such sweet ...
3	If you prick us do we not bleed? If you tickle...
4	All the world’s a stage, and all the men and w...
5	Romeo, Romeo! Wherefore art thou Romeo?
6	If music be the food of love play on.
7	What’s in a name? A rose by any other name wou...
8	Shall I compare thee to a summer’s day? Thou a...
9	How sharper than a serpent’s tooth it is to ha...

```
#To be, or not to be: that is the question.

sentence_tok = tokenizer_h.texts_to_sequences(sentences_df.Line)
sentence_exp = pad_sequences(sentence_tok, maxlen=MAX_SEQUENCE_LENGTH)
sentence_shakes = cnnmodel_h.predict(sentence_exp)
print(sentence_shakes)
```

```
1/1 [==============================] - 0s 100ms/step
[[0.47284698 0.527153  ]
 [0.10213784 0.8978621 ]
 [0.05841608 0.9415839 ]
 [0.03079555 0.96920437]
 [0.9019413  0.0980587 ]
 [0.1211498  0.87885016]
 [0.16905494 0.830945  ]
 [0.8127134  0.1872865 ]
 [0.7245256  0.2754744 ]
 [0.24459441 0.75540555]]
```

```
sentence_shakes_df = pd.DataFrame(sentence_shakes)
print(sentence_shakes_df)
```

```
          0         1
0  0.472847  0.527153
1  0.102138  0.897862
2  0.058416  0.941584
```

```
3  0.030796  0.969204
4  0.901941  0.098059
5  0.121150  0.878850
6  0.169055  0.830945
7  0.812713  0.187286
8  0.724526  0.275474
9  0.244594  0.755406
```

```
1 sentence_shakes_df['Line'] = sentences_df.Line
2 print(sentence_shakes_df)
```

```
           0         1                                               Line
0  0.472847  0.527153         To be, or not to be: that is the question.
1  0.102138  0.897862           The lady doth protest too much, methinks
2  0.058416  0.941584  Good night, good night! Parting is such sweet ...
3  0.030796  0.969204  If you prick us do we not bleed? If you tickle...
4  0.901941  0.098059  All the world’s a stage, and all the men and w...
5  0.121150  0.878850            Romeo, Romeo! Wherefore art thou Romeo?
6  0.169055  0.830945              If music be the food of love play on.
7  0.812713  0.187286  What’s in a name? A rose by any other name wou...
8  0.724526  0.275474  Shall I compare thee to a summer’s day? Thou a...
9  0.244594  0.755406  How sharper than a serpent’s tooth it is to ha...
```