```
#Importing all the needed libraries
import pandas as PD
import matplotlib.pyplot as plt
import seaborn as SB
import numpy as NP

#NLTK imports
import csv
import nltk
nltk.download('all')
from nltk.corpus import wordnet
from nltk.corpus import stopwords

#For Naive Bayas
from sklearn.naive_bayes import *

#For vectorization
from sklearn.feature_extraction.text import TfidfVectorizer

#Imports of neural networks
import tensorflow as tf
from tensorflow.keras import datasets, layers, models, preprocessing
from keras.models import Sequential
from keras import layers

from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier

#stopword shortcut
stopwords = stopwords.words('english')
```

```
#read file
filename = 'metacritic_game_user_comments.csv'

#We want these specific columns
col_names = ['Title', 'Platform' ,'Userscore', 'Comment']

DF = PD.read_csv(filename, index_col= 0)
DF = PD.DataFrame(DF, columns= col_names)

print(DF)
DF['Platform'] = DF.Platform.astype("category")

#here are the platforms we are dealing w/
print("Here are all the platforms we are dealing with:\n")
DF['Platform'].value_counts()
```

```
                                          Title    Platform  Userscore  \
Num
0             The Legend of Zelda: Ocarina of Time  Nintendo64         10
1             The Legend of Zelda: Ocarina of Time  Nintendo64         10
2             The Legend of Zelda: Ocarina of Time  Nintendo64         10
3             The Legend of Zelda: Ocarina of Time  Nintendo64         10
4             The Legend of Zelda: Ocarina of Time  Nintendo64         10
...                                            ...         ...        ...
283978  Etrian Odyssey Untold: The Millennium Girl         3DS          7
283979  Etrian Odyssey Untold: The Millennium Girl         3DS          0
283980  Etrian Odyssey Untold: The Millennium Girl         3DS          9
283981  Etrian Odyssey Untold: The Millennium Girl         3DS          8
283982  Etrian Odyssey Untold: The Millennium Girl         3DS          9

                                                  Comment
Num
0       Everything in OoT is so near at perfection, it...
1       I won't bore you with what everyone is already...
2       Anyone who gives the masterpiece below a 7 or ...
3       I'm one of those people who think that this is...
4        This game is the highest rated game on Metacr...
...                                                   ...
283978  Extremely similar to EO:4, which obviously isn...
283979  Typical overrated Atlus trash. A game i should...
283980  While I find the story mode to have annoying c...
283981  Pretty good, but it certainly lacks the visual...
283982  As my first game from the "Etrian series" i ha...
```

```
[283983 rows x 4 columns]
Here are all the platforms we are dealing with:

PC                118936
Xbox360            37420
PlayStation4       33547
PlayStation3       32430
PlayStation2       10637
XboxOne             9137
Switch              5922
Wii                 5232
3DS                 4812
Xbox                4713
WiiU                4420
GameCube            3479
DS                  3047
Nintendo64          2401
PlayStation         2396
PSP                 1732
GameBoyAdvance      1588
PlayStationVita     1555
Dreamcast            456
not specified        123
Name: Platform, dtype: int64
```

Let's filter out our data and sentence tokenize our comment column

---

Default title text

```
from nltk.corpus.reader.tagged import word_tokenize
#@title Default title text
import re
#Finding any NAs
DF_nulls = PD.isnull(DF)
print(DF_nulls)
```

```
          Title  Platform  Userscore  Comment
Num
0       False     False      False    False
1       False     False      False    False
2       False     False      False    False
3       False     False      False    False
4       False     False      False    False
...       ...       ...        ...      ...
283978  False     False      False    False
283979  False     False      False    False
283980  False     False      False    False
283981  False     False      False    False
283982  False     False      False    False

[283983 rows x 4 columns]
```

We will cover these three targets:

1. Target: Game Platform

   Domain: Game title

2. Target: Game title

   Domain: Game Platform

3. Target: Game title

   Domain: Comments

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split

#Method call for Tfid vectorizer
vectorizer = TfidfVectorizer()

#Lets first see a relation between the game platform and game title
X = DF.Title      #feature
Y = DF.Platform   #target
```

```
10
11 #Converting Platform column to number value
12 from sklearn.preprocessing import LabelEncoder
13 LB = LabelEncoder()
14 Y = LB.fit_transform(Y)
15
16 # train-test split
17 X_train, X_test, y_train, y_test = train_test_split(X, Y,
18 test_size=0.2, train_size=0.8, random_state=1234)
19
20 #Removing the stopwords from the text. All to fit the training data
21 vector = TfidfVectorizer(stop_words= stopwords)
22 X_train_vectored = vector.fit_transform(X_train)
23 X_test_vectored = vector.transform(X_test)
24
25 #making to array
26 X_train_vectored = X_train_vectored.toarray()
27 X_test_vectored = X_test_vectored.toarray()
28
29 #Displaying the results
30 print("The train shape after vectorization is:",X_train_vectored.shape)
31 print("The test shape after vectorization is:",X_test_vectored.shape)
```

```
The train shape after vectorization is: (227186, 2291)
The test shape after vectorization is: (56797, 2291)
```

```
1 from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix
2
3 #Lets see the results in Naive Bayes model
4 #Getting MultinomialNB started
5 MNB = MultinomialNB()
6 MNB.fit(X_train_vectored, y_train)
7
8 # make predictions on the test data
9 pred = MNB.predict(X_test_vectored)
10
11 # print confusion matrix
12 print(confusion_matrix(y_test, pred))
13
14 #the MNB score
15 MNBscore = MNB.score(X_test_vectored, y_test)
16
17 print('\nNaive Bayes accuracy is:', MNBscore)
18 print('\nNaive Bayes precision score: ', precision_score(y_test, pred, average='micro'))
19 print('\nNaive Bayes recall score: ', recall_score(y_test, pred, average='micro'))
20 print('\nNaive Bayes f1 score: ', f1_score(y_test, pred, average='micro'))
21
22
```

```
[[  839     7     0     0     0     0     3     0     2    11    32    28
     11     9     3    44     0     0     0     0]
 [   11   531     0     0     2     0    22     8     0     3     0     5
      0     0     3    20     2     2     0     0]
 [    0     3    43     0     7     0     5     0     3     8    17     0
      0     0     0     0    10     3     1     0]
 [   37     0     0   253     3     0     3     0     0     2     0     8
      0     0    12     7     4     1     0     0]
 [   12     0     0     0   531     5    11     0     0    79    11     1
      0     0    21    18    12    17     0     0]
 [    0     4     0     0     2   431     2     2     0     7     0     5
      3     0    23     0     0    13     0     0]
 [    3    10     0     0    22     0 20140     0     4   210   366   858
      1    32     0    17   203  1666   225     0]
 [    0     3     0     0     0     0     3   245     0    31    11     5
      0     0     1     0     0    14     0     0]
 [    3     0     0     0    20     0     6     6   321    50    56    16
      0     0     0     0     2     6     0     0]
 [    1     0     0     0    53     0   103     2     8  1450   108   271
      0     0     7     0    58    84     0     0]
 [    3     2     0     0    33     0  1607     7     2    34  2988   265
      1     0     7    24     2  1535     0     0]
 [    0     0     0     0     7     0  1271     0     2     5    61  4745
      5    35    12     7     0   435    99     0]
 [    4     0     0     0     1     0    25     0     0     0    17    56
    227     0     0     0     0     0     0     0]
 [    1     1     0     0     2     0    71     0     0     1     2    34
      1   965     4    74     0     4     4     0]
 [    0     0     0     0     0    19    46     0     0    26     0     7
      0    44   812    58     0    23     0     0]
```

```
[    0     0     0     0     0     0    23     0     0     0    14    50
     0   158    96   519     0    11     0     0]
[    0     0     0     0     4     0   124     2     3   198    30     3
     0     0     0     0   400   185    13     0]
[    0     5     0     0    26     1  2170     6     6    56   628    99
     0     0    13    34    53  4267    97     0]
[    0     0     0     0     0     4   421     0     0     6     3   442
     0     6     3     4     0   199   726     0]
[    0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     4     0    23]]

Naive Bayes accuracy is: 0.712291142137789

Naive Bayes precision score:  0.712291142137789

Naive Bayes recall score:  0.712291142137789

Naive Bayes f1 score:  0.7122911421377891
```

Now lets see what deep learning model can give us.

1st on to the plate is Basic Keras Model

```
maxlen = 50

#loading in the target
input_train = X_train_vectored.shape  # Number of different trains
input_test = X_test_vectored.shape  #Number of different test

print("This is the size of our train data:", input_train)
print("This is the size of our test data:", input_test)
```

```
This is the size of our train data: (227186, 2291)
This is the size of our test data: (56797, 2291)
```

```
model = models.Sequential()
# Layer 1
model.add(layers.Dense(750, input_dim=2291, activation='relu'))
# Layer 2
model.add(layers.Dense(500, activation='sigmoid'))
# Layer 3
model.add(layers.Dense(200, activation='sigmoid'))
# Layer 6
model.add(layers.Dense(100, activation='sigmoid'))
# Layer 7
model.add(layers.Dense(50, activation='sigmoid'))
# Layer 8
model.add(layers.Dense(10, activation='sigmoid'))
# Layer 9
model.add(layers.Dense(1, activation='relu'))


#compiling our layers
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc'])
model.summary()

history = model.fit(X_train_vectored, y_train,
                    epochs=10,
                    batch_size=45,
                    validation_split = 0.3
                   )
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 750)               1719000

 dense_1 (Dense)             (None, 500)               375500

 dense_2 (Dense)             (None, 200)               100200

 dense_3 (Dense)             (None, 100)               20100

 dense_4 (Dense)             (None, 50)                5050

 dense_5 (Dense)             (None, 10)                510
```

```
 dense_6 (Dense)             (None, 1)                 11

=================================================================
Total params: 2,220,371
Trainable params: 2,220,371
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
3534/3534 [==============================] - 103s 29ms/step - loss: 145.6916 - acc: 0.0170 - val_loss: 145.8305 - val_acc: 0.0165
Epoch 2/10
3534/3534 [==============================] - 101s 28ms/step - loss: 145.6915 - acc: 0.0170 - val_loss: 145.8305 - val_acc: 0.0165
Epoch 3/10
3534/3534 [==============================] - 99s 28ms/step - loss: 145.6915 - acc: 0.0170 - val_loss: 145.8305 - val_acc: 0.0165
Epoch 4/10
3534/3534 [==============================] - 96s 27ms/step - loss: 145.6915 - acc: 0.0170 - val_loss: 145.8305 - val_acc: 0.0165
Epoch 5/10
3534/3534 [==============================] - 95s 27ms/step - loss: 145.6915 - acc: 0.0170 - val_loss: 145.8305 - val_acc: 0.0165
Epoch 6/10
3534/3534 [==============================] - 103s 29ms/step - loss: 145.6914 - acc: 0.0170 - val_loss: 145.8305 - val_acc: 0.0165
Epoch 7/10
3534/3534 [==============================] - 100s 28ms/step - loss: 145.6913 - acc: 0.0170 - val_loss: 145.8305 - val_acc: 0.0165
Epoch 8/10
3534/3534 [==============================] - 99s 28ms/step - loss: 145.6914 - acc: 0.0170 - val_loss: 145.8305 - val_acc: 0.0165
Epoch 9/10
3534/3534 [==============================] - 92s 26ms/step - loss: 145.6915 - acc: 0.0170 - val_loss: 145.8305 - val_acc: 0.0165
Epoch 10/10
3534/3534 [==============================] - 93s 26ms/step - loss: 145.6912 - acc: 0.0170 - val_loss: 145.8305 - val_acc: 0.0165
```

> CNN time and a new target

This section of the code will cover the target column platform and feature comment

---

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split

#Method call for Tfid vectorizer
vectorizer = TfidfVectorizer()

#Lets first see a relation between the game platform and game title
X = DF.Platform    #feature
Y = DF.Title   #target

#Converting Platform column to number value
from sklearn.preprocessing import LabelEncoder
LB = LabelEncoder()
Y = LB.fit_transform(Y)

# train-test split
X_train, X_test, y_train, y_test = train_test_split(X, Y,
test_size=0.15, train_size=0.85, random_state=1234)

#Removing the stopwords from the text. All to fit the training data
vector = TfidfVectorizer(stop_words= stopwords)
X_train_vectored = vector.fit_transform(X_train)
X_test_vectored = vector.transform(X_test)

#making to array
X_train_vectored = X_train_vectored.toarray()
X_test_vectored = X_test_vectored.toarray()

#Displaying the results
print("The train shape after vectorization is:",X_train_vectored.shape)
print("The test shape after vectorization is:",X_test_vectored.shape)
```

```
The train shape after vectorization is: (241385, 20)
The test shape after vectorization is: (42598, 20)
```

Lets see what we get with the feature being the Platform

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix

#Lets see the results in Naive Bayes model
#Getting MultinomialNB started
MNB = MultinomialNB()
MNB.fit(X_train_vectored, y_train)
```

```
 7
 8 # make predictions on the test data
 9 pred = MNB.predict(X_test_vectored)
10
11 # print confusion matrix
12 print(confusion_matrix(y_test, pred))
13
14 #the MNB score
15 MNBscore = MNB.score(X_test_vectored, y_test)
16
17 print('\nNaive Bayes accuracy is:', MNBscore)
18 print('\nNaive Bayes precision score: ', precision_score(y_test, pred, average='micro'))
19 print('\nNaive Bayes recall score: ', recall_score(y_test, pred, average='micro'))
20 print('\nNaive Bayes f1 score: ', f1_score(y_test, pred, average='micro'))
```

```
[[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]

Naive Bayes accuracy is: 0.07401755950983614

Naive Bayes precision score:  0.07401755950983614

Naive Bayes recall score:  0.07401755950983614

Naive Bayes f1 score:  0.07401755950983614
```

Now lets see what deep learning model can give us.

1st on to the plate is CNN

```
1 #loading in the target
2 input_train = X_train_vectored.shape  # Number of different trains
3 print("This is the size of our train data:", input_train)
```

```
This is the size of our train data: (241385, 20)
```

```
1 max_features = 15000
2 maxlen = 20
3
4 model = models.Sequential()
5 model.add(layers.Embedding(max_features, 120, input_length=maxlen))
6 model.add(layers.Conv1D(55, 10, activation='relu'))
7 model.add(layers.MaxPooling1D(5))
8 model.add(layers.Conv1D(55, 2, activation='relu'))
9 model.add(layers.Conv1D(55, 1, activation='relu'))
10 model.add(layers.GlobalMaxPooling1D())
11 model.add(layers.Dense(10))
12 model.add(layers.Dense(1))
13
14 model.summary()
15
16 #compiling our layers
17 model.compile(optimizer=tf.keras.optimizers.RMSprop(lr=1e-4),  # set learning rate
18               loss='binary_crossentropy',
19               metrics=['accuracy'])
20
21 history = model.fit(X_train_vectored, y_train,
22                     epochs=10,
23                     batch_size=125,
24                     validation_data = (X_test_vectored, y_test)
25                     )
26
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, 20, 120)           1800000

 conv1d_3 (Conv1D)           (None, 11, 55)            66055

 max_pooling1d_1 (MaxPooling  (None, 2, 55)            0
```

```
 1D)

 conv1d_4 (Conv1D)           (None, 1, 55)             6105

 conv1d_5 (Conv1D)           (None, 1, 55)             3080

 global_max_pooling1d_1 (Glo  (None, 55)               0
 balMaxPooling1D)

 dense_9 (Dense)             (None, 10)                560

 dense_10 (Dense)            (None, 1)                 11

=================================================================
Total params: 1,875,811
Trainable params: 1,875,811
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
1932/1932 [==============================] - 85s 43ms/step - loss: 15998.0947 - accuracy: 6.2141e-05 - val_loss: 16082.0752 - val_accur
Epoch 2/10
1932/1932 [==============================] - 76s 39ms/step - loss: 15998.0869 - accuracy: 6.2141e-05 - val_loss: 16082.0752 - val_accur
Epoch 3/10
1932/1932 [==============================] - 75s 39ms/step - loss: 15998.0986 - accuracy: 6.2141e-05 - val_loss: 16082.0752 - val_accur
Epoch 4/10
1932/1932 [==============================] - 70s 36ms/step - loss: 15998.0811 - accuracy: 6.2141e-05 - val_loss: 16082.0752 - val_accur
Epoch 5/10
1932/1932 [==============================] - 69s 36ms/step - loss: 15998.1094 - accuracy: 6.2141e-05 - val_loss: 16082.0752 - val_accur
Epoch 6/10
1932/1932 [==============================] - 74s 38ms/step - loss: 15998.1035 - accuracy: 6.2141e-05 - val_loss: 16082.0752 - val_accur
Epoch 7/10
1932/1932 [==============================] - 77s 40ms/step - loss: 15998.0938 - accuracy: 6.2141e-05 - val_loss: 16082.0752 - val_accur
Epoch 8/10
1932/1932 [==============================] - 77s 40ms/step - loss: 15998.0781 - accuracy: 6.2141e-05 - val_loss: 16082.0752 - val_accur
Epoch 9/10
1932/1932 [==============================] - 81s 42ms/step - loss: 15998.0928 - accuracy: 6.2141e-05 - val_loss: 16082.0752 - val_accur
Epoch 10/10
1932/1932 [==============================] - 77s 40ms/step - loss: 15998.1104 - accuracy: 6.2141e-05 - val_loss: 16082.0752 - val_accur
```

✓ 13m 23s completed at 3:09 AM