In [4]:
```
# This Python 3 environment comes with many helpful analytics libraries ins
# It is defined by the kaggle/python Docker image: https://github.com/kaggl
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        os.path.join(dirname, filename)

# You can write up to 20GB to the current directory (/kaggle/working/) that
# You can also write temporary files to /kaggle/temp/, but they won't be sa
```

Importing Libraries

In [5]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn.preprocessing as preprocessing
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from sklearn.preprocessing import StandardScaler
import nltk
import re
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle
from tqdm import tqdm
```

In [6]:
```
df_train = pd.read_csv("../input/aid-escalating-internet-coverage/train.csv
df_test = pd.read_csv("../input/aid-escalating-internet-coverage/test.csv")

y = df_train["label"]
df_train.drop("label",axis = 1,inplace = True)
df = pd.concat([df_train,df_test],ignore_index=True)
```

Text Processing

In [7]:
```
df_page = df.loc[:,"page_description"]

for i in range(df_page.shape[0]):
    df_page.iloc[i] = re.sub(r'[^\w\s]','',df_page.iloc[i])
for i in range(df_page.shape[0]):
     df_page.iloc[i] = re.sub(r'[\d]','',df_page.iloc[i])
```

```
/opt/conda/lib/python3.7/site-packages/pandas/core/indexing.py:1732: Settin
gWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  self._setitem_single_block(indexer, value, name)
```

Tokenize

In [8]:

```
token = [[]] * df.shape[0]
for i in range(df.shape[0]):
    token[i] = nltk.word_tokenize(df_page.iloc[i].lower())
```

Remove Stop Words

In [9]:

```
from nltk.corpus import stopwords
stop_words = stopwords.words('english')
stop_words.extend(['i','my','if','oh','yes','yeah','no','cuz','us','also','
word_list=[]
for text in token:
    no_stopwords = [word for word in text if word not in stop_words]
    word_list.append(no_stopwords)
```

Lemmatization

In [10]:

```
nltk.download('omw-1.4')
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()

lemmatize_list = []
for word in word_list:
    filter_data = []
    for data in word:
        filter_data.append(lemmatizer.lemmatize(data))
    lemmatize_list.append(filter_data)
```

```
[nltk_data] Downloading package omw-1.4 to /usr/share/nltk_data...
```

Unique Values

In [11]:

```
unique_list=[]
for data in lemmatize_list:
    set_data = set(data)
    temporary=[]
    for word in set_data:
        temporary.append(word)
    unique_list.append(temporary)
```

In [12]:

```
clean_pagedesc=[]
for item in unique_list:
```

```
    sr=" "
    clean_pagedesc.append(sr.join(item))
df["clean_page_description"]=clean_pagedesc
```

Word2Vec Implementation

In [13]:
```
list_of_sentance=[]
for sentance in clean_pagedesc:
    list_of_sentance.append(sentance.split())
```

In [14]:
```
w2v_model = Word2Vec(list_of_sentance,min_count = 5,vector_size = 50, worke
w2v_words = list(w2v_model.wv.index_to_key)
```

In [15]:
```
feature_extraction = TfidfVectorizer()
feature_extraction.fit(df['clean_page_description'])
a=feature_extraction.get_feature_names_out()
b=list(feature_extraction.idf_)
```

In [16]:
```
dictionary = {}
for i in range(len(a)):
    dictionary[a[i]]=b[i]
```

In [17]:
```
tfidf_feat = feature_extraction.get_feature_names_out()

tfidf_sent_vectors = []
row=0
for sent in tqdm(list_of_sentance):
    sent_vec = np.zeros(50)
    weight_sum =0
    for word in sent:
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors.append(sent_vec)
    row += 1
```

```
100%|██████████| 7395/7395 [1:19:20<00:00,  1.55it/s]
```

In [18]:
```
len(tfidf_sent_vectors)
x_df = pd.DataFrame(tfidf_sent_vectors)
x_df.shape
```

Out[18]:
```
(7395, 50)
```

Predicting "alchemy_score"

In [19]:
```
cat_df=pd.DataFrame()
cat_df['alchemy_category']=df['alchemy_category']
```

```
cat_df=pd.concat([x_df,cat_df],axis=1)
dict={'arts_entertainment':1, 'recreation':2, 'business':3, 'sports':4, '?'
'science_technology':10, 'gaming':11, 'law_crime':12, 'unknown':13, 'weathe

for i in range(cat_df.shape[0]):
    cat_df.loc[i,"alchemy_category"]=dict[cat_df.loc[i,"alchemy_category"]]
cat_df["alchemy_category"]=cat_df['alchemy_category'].astype(np.int64)
```

In [20]:

```
cat_test=cat_df[cat_df['alchemy_category']==5]
cat_train=cat_df[cat_df['alchemy_category']!=5]

test_X=cat_test.drop("alchemy_category",axis=1)
test_Y=cat_test["alchemy_category"]

train_X=cat_train.drop("alchemy_category",axis=1)
train_Y=cat_train["alchemy_category"]
```

In [21]:

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier

train_cat_X, test_cat_X, train_cat_y, test_cat_y = train_test_split(train_X

decision_tree_model = DecisionTreeClassifier(max_depth = 10).fit(train_cat_
yhat = decision_tree_model.predict(test_cat_X)
yt=pd.DataFrame(yhat)
yt.value_counts()
```

Out[21]:

```
2     289
1     232
3     196
9     113
4      76
7      45
10     31
6      24
11      5
dtype: int64
```

In [22]:

```
accuracy_score(test_cat_y,yhat)
```

Out[22]:

```
0.32146389713155293
```

In [23]:

```
decision_tree = DecisionTreeClassifier(max_depth = 10).fit(train_X, train_Y
category_y = decision_tree.predict(test_X)
category=pd.DataFrame(category_y)
```

In [24]:

```
k = 0
for i in range(cat_df.shape[0]):
    if cat_df.loc[i,"alchemy_category"] == 5:
        cat_df.loc[i,"alchemy_category"] = category[0][k]
        k += 1
```

Predict "alchemy_category_score"

In [25]:
```
cat_score_df=pd.DataFrame()

cat_score_df['alchemy_category_score']=df['alchemy_category_score']
cat_score_df=pd.concat([x_df,cat_score_df],axis=1)


cat_score_test=cat_score_df[cat_score_df['alchemy_category_score'] == "?"]
cat_score_train=cat_score_df[cat_score_df['alchemy_category_score'] != "?"]

test_x=cat_score_test.drop("alchemy_category_score",axis=1)
test_y=cat_score_test["alchemy_category_score"]

train_x=cat_score_train.drop("alchemy_category_score",axis=1)
train_y=cat_score_train["alchemy_category_score"].astype(np.float64)
```

In [26]:
```
train_cat_score_X, test_cat_score_X, train_cat_score_y, test_cat_score_y =
```

In [27]:
```
from sklearn.linear_model import LinearRegression

lr_model = LinearRegression()
lr_model.fit(train_cat_score_X, train_cat_score_y)
score_y=lr_model.predict(test_cat_score_X)

score=pd.DataFrame(score_y).astype(np.float64)
```

In [28]:
```
import math
from sklearn.metrics import mean_squared_error
rmse = math.sqrt(mean_squared_error(test_cat_score_y,score))
rmse
```

Out[28]:
```
0.21090979661118997
```

In [29]:
```
from sklearn.linear_model import LinearRegression

lr_model = LinearRegression()
lr_model.fit(train_x, train_y)
cat_score_y=lr_model.predict(test_x)
cat_score=pd.DataFrame(cat_score_y).astype(np.float64)
```

In [30]:
```
k = 0
for i in range(cat_score_df.shape[0]):
    if cat_score_df.loc[i,"alchemy_category_score"] == "?":
        cat_score_df.loc[i,"alchemy_category_score"] = cat_score[0][k]
        k += 1
```

Selecting relevant features

In [31]:
```
df_new=df.drop(["link_id","alchemy_category","alchemy_category_score","link
df_new["alchemy_category"]= cat_df["alchemy_category"]
df_new["alchemy_category_score"]= cat_score_df["alchemy_category_score"].as
df_new_columns=df_new.columns
```

Standardization

In [32]:
```
scaler = StandardScaler()
for i in df_new_columns:
    df_new[i] = scaler.fit_transform(df_new[i].to_numpy().reshape(-1, 1)).r
```

Outlier Removal

In [33]:
```
class OutlierRemoval:
    def __init__(self, lower_quartile, upper_quartile):
        self.lower_whisker = lower_quartile - 1.5*(upper_quartile - lower_q
        self.upper_whisker = upper_quartile + 1.5*(upper_quartile - lower_q

    def removeOutlier(self, x):
        return (x if x <= self.upper_whisker and x >= self.lower_whisker el
```

In [34]:
```
for i in df_new_columns:
    score=df_new[i]
    score_outlier_remover = OutlierRemoval(score.quantile(0.25), score.quar
    outlier_removed_score = score.apply(score_outlier_remover.removeOutlier
    df_new[i]=outlier_removed_score
```

Splitting into Test and Train

In [35]:
```
Train=pd.concat([df_new.iloc[:4437,:],x_df.iloc[:4437,:]],ignore_index=True
Test=pd.concat([df_new.iloc[4437:,:],x_df.iloc[4437:,:]],ignore_index=True,
```

In [36]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, f1_score
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(Train, y, test_size=0.2
```

XGBoost (Hyper Parameter Optimization)

In [37]:
```
params={
 "learning_rate"    : [0.05, 0.10, 0.15, 0.20, 0.25, 0.30 ] ,
 "max_depth"        : [ 3, 4, 5, 6, 8, 10, 12, 15],
 "min_child_weight" : [ 1, 3, 5, 7 ],
 "gamma"            : [ 0.0, 0.1, 0.2 , 0.3, 0.4 ],
 "colsample_bytree" : [ 0.3, 0.4, 0.5 , 0.7 ]
}
```

In [38]:
```
from sklearn.model_selection import RandomizedSearchCV
import xgboost
classifier=xgboost.XGBClassifier()
```

In [39]:
```
random_search=RandomizedSearchCV(classifier,param_distributions=params,n_it
```

```
random_search.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 5 candidates, totalling 25 fits
RandomizedSearchCV(cv=5,
                   estimator=XGBClassifier(base_score=None, booster=None,
                                           callbacks=None,
                                           colsample_bylevel=None,
                                           colsample_bynode=None,
                                           colsample_bytree=None,
                                           early_stopping_rounds=None,
                                           enable_categorical=False,
                                           eval_metric=None, gamma=None,
                                           gpu_id=None, grow_policy=None,
                                           importance_type=None,
                                           interaction_constraints=None,
                                           learning_rate=None, max_bin=Non
e,...
                                           n_estimators=100, n_jobs=None,
                                           num_parallel_tree=None,
                                           predictor=None, random_state=Non
e,
                                           reg_alpha=None, reg_lambda=None,
...),
                   n_iter=5, n_jobs=-1,
                   param_distributions={'colsample_bytree': [0.3, 0.4, 0.5,
                                                             0.7],
                                        'gamma': [0.0, 0.1, 0.2, 0.3, 0.4],
                                        'learning_rate': [0.05, 0.1, 0.15,
0.2,
                                                          0.25, 0.3],
                                        'max_depth': [3, 4, 5, 6, 8, 10, 1
2,
                                                      15],
                                        'min_child_weight': [1, 3, 5, 7]},
                   scoring='roc_auc', verbose=3)
```

In [40]:
```
random_search.best_estimator_
```

Out[40]:
```
XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=0.
4,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, gamma=0.0, gpu_id=-1, grow_policy='depthwis
e',
              importance_type=None, interaction_constraints='',
              learning_rate=0.1, max_bin=256, max_cat_to_onehot=4,
              max_delta_step=0, max_depth=3, max_leaves=0, min_child_weight
=7,
              missing=nan, monotone_constraints='()', n_estimators=100,
              n_jobs=0, num_parallel_tree=1, predictor='auto', random_state
=0,
              reg_alpha=0, reg_lambda=1, ...)
```

In [41]:
```
random_search.best_params_
```

Out[41]:
```
{'min_child_weight': 7,
 'max_depth': 3,
 'learning_rate': 0.1,
 'gamma': 0.0,
 'colsample_bytree': 0.4}
```

In [42]:
```
y_pred = random_search.predict_proba(X_test)
```

```
roc_auc_score(y_test,y_pred[:,1])
```

Out[42]:
```
0.8630953589925865
```