In [1]:

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# 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 list all files under the input directory

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 gets preserved as output when you create a version
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session
```

Importing Libraries

In [2]:

```
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 [3]:

```
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)
```

Data Preprocessing

In [4]:

```
df.isna().sum()
```

Out[4]:

```
link                                  0
link_id                               0
page_description                      0
alchemy_category                      0
alchemy_category_score                0
avg_link_size                         0
common_word_link_ratio_1              0
common_word_link_ratio_2              0
common_word_link_ratio_3              0
common_word_link_ratio_4              0
compression_ratio                     0
embed_ratio                           0
frame_based                           0
frame_tag_ratio                       0
has_domain_link                       0
html_ratio                            0
image_ratio                           0
is_news                               0
lengthy_link_domain                   0
link_word_score                       0
news_front_page                       0
non_markup_alphanumeric_characters    0
count_of_links                        0
number_of_words_in_url                0
parametrized_link_ratio               0
spelling_mistakes_ratio               0
dtype: int64
```

In [5]:

```
(df == '?').sum()
```

Out[5]:

```
link                                     0
link_id                                  0
page_description                         0
alchemy_category                      2342
alchemy_category_score                2342
avg_link_size                            0
common_word_link_ratio_1                 0
common_word_link_ratio_2                 0
common_word_link_ratio_3                 0
common_word_link_ratio_4                 0
compression_ratio                        0
embed_ratio                              0
frame_based                              0
frame_tag_ratio                          0
has_domain_link                          0
html_ratio                               0
image_ratio                              0
is_news                               2843
lengthy_link_domain                      0
link_word_score                          0
news_front_page                       1248
non_markup_alphanumeric_characters       0
count_of_links                           0
number_of_words_in_url                   0
parametrized_link_ratio                  0
spelling_mistakes_ratio                  0
dtype: int64
```

In [6]:

```
#df["alchemy_category"].replace("?","recreation",inplace = True)
```

In [11]:

```
# df["alchemy_category_score"].replace("?",0,inplace = True)
# df["alchemy_category_score"] = pd.to_numeric(df["alchemy_category_score"])
# df["alchemy_category_score"].replace(0,df["alchemy_category_score"].sum()/5053,inplace = True)
```

In [12]:

```
df.duplicated().sum()
```

Out[12]:

```
0
```

NLP (Data Cleaning)

Text Processing

In [13]:

```
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])
```

Tokenize

In [14]:

```
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 [15]:

```
from nltk.corpus import stopwords
stop_words = stopwords.words('english')
stop_words.extend(['i','my','if','oh','yes','yeah','no','cuz','us','also','un','put','get','got','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 [16]:

```
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 [17]:

```
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 [18]:

```
clean_pagedesc=[]
for item in unique_list:
    sr=" "
    clean_pagedesc.append(sr.join(item))
df["clean_page_description"]=clean_pagedesc
```

Applying TF-IDF

In [19]:

```
from sklearn.decomposition import TruncatedSVD
feature_extraction = TfidfVectorizer(max_df=0.9,min_df = 5,max_features=3000)
page_feature = feature_extraction.fit_transform(df['clean_page_description'])
x_df = pd.DataFrame(page_feature.toarray())
```

PCA(SVD)

In [20]:

```
svd = TruncatedSVD(n_components=1000)
x_df = svd.fit_transform(x_df)
x_df=pd.DataFrame(x_df)
```

Word2Vec Implementation

In [21]:

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

In [22]:

```
# w2v_model = Word2Vec(list_of_sentance,min_count = 5,vector_size = 50, workers = 4,window = 5, sg = 1,negative =20)
# w2v_words = list(w2v_model.wv.index_to_key)
```

In [23]:

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

In [24]:

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

In [25]:

```
# 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
```

In [26]:

```
# len(tfidf_sent_vectors)
# x_df = pd.DataFrame(tfidf_sent_vectors)
# x_df.shape
```

Predicting "alchemy_score"

In [27]:

```
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, '?':5,'culture_politics':6, 'computer_internet':7, 're
'science_technology':10, 'gaming':11, 'law_crime':12, 'unknown':13, 'weather':14}

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 [28]:

```
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 [29]:

```
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, train_Y, train_size=0.8)

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

Out[29]:

```
1     316
2     273
3     153
9      94
4      80
7      31
10     29
6      26
8       4
11      4
12      1
dtype: int64
```

In [30]:

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

Out[30]:

```
0.334322453016815
```

In [31]:

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

In [32]:

```
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
```

Predicting "alchemy_category_score"

In [33]:

```
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 [34]:

```
train_cat_score_X, test_cat_score_X, train_cat_score_y, test_cat_score_y = train_test_split(train_x, train_y, train_size=0.8)
```

In [35]:

```
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 [36]:

```
import math
from sklearn.metrics import mean_squared_error
rmse = math.sqrt(mean_squared_error(test_cat_score_y,score))
rmse
```

Out[36]:

```
0.22366917184708088
```

In [37]:

```
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 [38]:

```
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 [39]:

```
df_new=df.drop(["link_id","alchemy_category","alchemy_category_score","link","clean_page_description","page_description","fram
df_new["alchemy_category"]= cat_df["alchemy_category"]
df_new["alchemy_category_score"]= cat_score_df["alchemy_category_score"].astype(np.float64)
df_new_columns=df_new.columns
```

Exploratory Data Analysis

Standardization

In [40]:

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

Outlier Removal

In [41]:

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

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

In [42]:

```
for i in df_new_columns:
    score=df_new[i]
    score_outlier_remover = OutlierRemoval(score.quantile(0.25), score.quantile(0.75))
    outlier_removed_score = score.apply(score_outlier_remover.removeOutlier)
    df_new[i]=outlier_removed_score
```

Skewness

In [43]:

```
# plt.hist(df["alchemy_category_score"], bins=20)
# plt.show()
```

Splitting into Test and Train

In [60]:

```
# Train=pd.concat([df_new.iloc[:4437,:],x_df.iloc[:4437,:]],ignore_index=True,axis=1)
# Test=pd.concat([df_new.iloc[4437:,:],x_df.iloc[4437:,:]],ignore_index=True,axis=1)

Train=pd.concat([x_df.iloc[:4437,:]],ignore_index=True,axis=1)
Test=pd.concat([x_df.iloc[4437:,:]],ignore_index=True,axis=1)
```

In [61]:

```
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, random_state=42)
```

Training Models

Logestic Regression

In [62]:

```
from sklearn.linear_model import LogisticRegression

logistic_regression_model = LogisticRegression(max_iter=700)
logistic_regression_model.fit(X_train,y_train)
log_roc=logistic_regression_model.predict_proba(X_test)
roc_auc_score(y_test,log_roc[:,1])
```

Out[62]:

```
0.8762871940692594
```

XGBoost with Hyper-Parameter Optimization

In [63]:

```
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 [64]:

```
from sklearn.model_selection import RandomizedSearchCV
import xgboost
classifier=xgboost.XGBClassifier()
```

In [65]:

```
random_search=RandomizedSearchCV(classifier,param_distributions=params,n_iter=5,scoring='roc_auc',n_jobs=-1,cv=5,verbose=3)
random_search.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 5 candidates, totalling 25 fits
```

Out[65]:

```
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=None,...
                                           n_estimators=100, n_jobs=None,
                                           num_parallel_tree=None,
                                           predictor=None, random_state=None,
                                           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, 12,
                                                      15],
                                        'min_child_weight': [1, 3, 5, 7]},
                   scoring='roc_auc', verbose=3)
```

In [66]:

```
random_search.best_params_
```

Out[66]:

```
{'min_child_weight': 1,
 'max_depth': 15,
 'learning_rate': 0.05,
 'gamma': 0.4,
 'colsample_bytree': 0.7}
```

In [67]:

```
y_pred = random_search.predict_proba(X_test)
roc_auc_score(y_test,y_pred[:,1])
```

Out[67]:

```
0.8629481060221387
```

K-Nearest Neighbors with Hyper-Parameter Optimization

In [68]:

```
# Import necessary modules
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import RandomizedSearchCV

param_grid = {
    'n_neighbors': [1, 3, 5, 7, 9, 11,13,15,17,19,21,23,25,27],
    'weights': ['uniform', 'distance'],
    'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute']
}

knn = KNeighborsClassifier()

knn_search = RandomizedSearchCV(
    estimator=knn,
    param_distributions=param_grid,
    n_iter=10,
    cv=5,
    scoring='roc_auc'
)

knn_search.fit(X_train, y_train)
print(knn_search.best_params_)
```

```
{'weights': 'uniform', 'n_neighbors': 25, 'algorithm': 'brute'}
```

In [69]:

```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(weights = "uniform", n_neighbors = 25, algorithm = "brute")
knn.fit(X_train, y_train)

y_pred = knn.predict_proba(X_test)
roc_auc_score(y_test,y_pred[:,1])
```

Out[69]:

```
0.8604041840154362
```

Support Vector Machine

In [70]:

```
from sklearn.svm import SVC

# Define the SVM classifier
clf = SVC(kernel='rbf',C=1,gamma=0.01)
clf.probability=True
# Fit the classifier on the training data
clf.fit(X_train, y_train)

# Use the classifier to make predictions on the test data
y_pred = clf.predict_proba(X_test)

# Evaluate the predictions using roc_auc as the metric
roc_auc_score(y_test,y_pred[:,1])
```

Out[70]:

```
0.858063369554179
```

Voting Ensemble Classifier (Logistic Regression, SVM, Decision Tree)

In [71]:

```
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split

clf1 = LogisticRegression(max_iter=700)
clf2 = SVC(kernel='rbf')
clf3 = DecisionTreeClassifier(max_depth=9)
clf1.probability=True
clf2.probability=True
clf3.probability=True

ensemble = VotingClassifier(
    estimators=[('lr', clf1), ('svc', clf2), ('dt', clf3)],
    voting='soft'
)
ensemble.fit(X_train, y_train)

y_pred = ensemble.predict_proba(X_test)

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

Out[71]:

```
0.8536864019498326
```

```
[CV 1/5] END colsample_bytree=0.3, gamma=0.3, learning_rate=0.1, max_depth=4, min_child_weight=5;, score=0.851 t
otal time=  16.9s
[CV 1/5] END colsample_bytree=0.7, gamma=0.1, learning_rate=0.15, max_depth=15, min_child_weight=5;, score=0.852
total time=  50.7s
[CV 1/5] END colsample_bytree=0.4, gamma=0.3, learning_rate=0.1, max_depth=4, min_child_weight=7;, score=0.852 t
otal time=  20.3s
[CV 4/5] END colsample_bytree=0.4, gamma=0.3, learning_rate=0.1, max_depth=4, min_child_weight=7;, score=0.864 t
otal time=  20.5s
[CV 2/5] END colsample_bytree=0.7, gamma=0.3, learning_rate=0.15, max_depth=12, min_child_weight=5;, score=0.826
total time=  49.9s
[CV 2/5] END colsample_bytree=0.7, gamma=0.4, learning_rate=0.05, max_depth=15, min_child_weight=1;, score=0.842
total time= 1.5min
[CV 4/5] END colsample_bytree=0.3, gamma=0.3, learning_rate=0.1, max_depth=4, min_child_weight=5;, score=0.872 t
otal time=  17.2s
[CV 3/5] END colsample_bytree=0.7, gamma=0.1, learning_rate=0.15, max_depth=15, min_child_weight=5;, score=0.863
total time=  49.6s
[CV 5/5] END colsample_bytree=0.7, gamma=0.1, learning_rate=0.15, max_depth=15, min_child_weight=5;, score=0.876
total time=  52.1s
[CV 4/5] END colsample_bytree=0.7, gamma=0.3, learning_rate=0.15, max_depth=12, min_child_weight=5;, score=0.865
total time=  50.1s
[CV 3/5] END colsample_bytree=0.7, gamma=0.4, learning_rate=0.05, max_depth=15, min_child_weight=1;, score=0.855
total time= 1.4min
[CV 3/5] END colsample_bytree=0.3, gamma=0.3, learning_rate=0.1, max_depth=4, min_child_weight=5;, score=0.853 t
otal time=  16.6s
[CV 5/5] END colsample_bytree=0.3, gamma=0.3, learning_rate=0.1, max_depth=4, min_child_weight=5;, score=0.868 t
otal time=  16.0s
[CV 4/5] END colsample_bytree=0.7, gamma=0.1, learning_rate=0.15, max_depth=15, min_child_weight=5;, score=0.869
total time=  49.8s
[CV 3/5] END colsample_bytree=0.4, gamma=0.3, learning_rate=0.1, max_depth=4, min_child_weight=7;, score=0.858 t
otal time=  20.3s
[CV 1/5] END colsample_bytree=0.7, gamma=0.3, learning_rate=0.15, max_depth=12, min_child_weight=5;, score=0.843
total time=  49.1s
[CV 5/5] END colsample_bytree=0.7, gamma=0.3, learning_rate=0.15, max_depth=12, min_child_weight=5;, score=0.869
total time=  48.9s
[CV 4/5] END colsample_bytree=0.7, gamma=0.4, learning_rate=0.05, max_depth=15, min_child_weight=1;, score=0.870
total time= 1.3min
[CV 2/5] END colsample_bytree=0.3, gamma=0.3, learning_rate=0.1, max_depth=4, min_child_weight=5;, score=0.842 t
otal time=  17.0s
[CV 2/5] END colsample_bytree=0.7, gamma=0.1, learning_rate=0.15, max_depth=15, min_child_weight=5;, score=0.822
total time=  50.8s
[CV 2/5] END colsample_bytree=0.4, gamma=0.3, learning_rate=0.1, max_depth=4, min_child_weight=7;, score=0.836 t
otal time=  20.3s
[CV 5/5] END colsample_bytree=0.4, gamma=0.3, learning_rate=0.1, max_depth=4, min_child_weight=7;, score=0.870 t
otal time=  20.5s
[CV 3/5] END colsample_bytree=0.7, gamma=0.3, learning_rate=0.15, max_depth=12, min_child_weight=5;, score=0.855
total time=  48.5s
[CV 1/5] END colsample_bytree=0.7, gamma=0.4, learning_rate=0.05, max_depth=15, min_child_weight=1;, score=0.850
total time= 1.5min
[CV 5/5] END colsample_bytree=0.7, gamma=0.4, learning_rate=0.05, max_depth=15, min_child_weight=1;, score=0.872
total time= 1.0min
```

Prediction on Test Data

In [72]:

```
train_yhat_probability1=logistic_regression_model.predict_proba(Test)
train_yhat_probability2=random_search.predict_proba(Test)
train_yhat_probability3=knn.predict_proba(Test)
train_yhat_probability4=clf.predict_proba(Test)
train_yhat_probability5=ensemble.predict_proba(Test)
```

In [76]:

```
# submission=pd.DataFrame(df_test["link_id"])
# temp=avgt[:,1]
# submission.insert(1,"label",temp,True)

# submission.to_csv('avg_submission.csv',index = False)
```