Data Mining and Analysis Course Project

Course Code: 18ECSC301

Team: 5A09

Roll No : 39- Apoorva S Malemath - 01FE16BCS041

Roll No : 44- Arundati Dixit - 01FE16BCS046

Roll No : 45- Ashish Kar - 01FE16BCS047

Roll No : 58- Deepti Nadkarni - 01FE16BCS062

Project ID : 5ADMACP14 Sina Weibo Interaction Prediction Challenge

Predict the forwarding, commenting and liking amount of a Weibo posted by a user based on the historical interaction data on Sina Weibo social platform.

Determining Statistical Factors

Authors: Apoorva Malemath, Arundati Dixit, Ashish Kar, Deepti Nadkarni

----------Understanding The Data----------

Predict Dataset Analysis

Contribution: All

In [ ]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%pylab inline
from googletrans import Translator
header = ['u_id','m_id','time','content']
predict_dataset= pd.read_fwf("G://DMA_PROJECT//weibo_predict_data.txt",header=None,names=header,
                    encoding='utf-8',delimiter="\t")
# fixed width formatted lines.
pedict_dataset.head(5)
translate_dataframe = pd.DataFrame(data=predict_dataset['content'].head(30))
translator = Translator()
translate_dataframe["English_content"] = translate_dataframe['content'].map(lambda x: translator.tr
anslate(x, src="zh-CN", dest="en").text)
print(translate_dataframe)
print("Predict_Dataset has "+str(predict_dataset.shape[0])+" records")
print("Predict_Dataset has "+str(predict_dataset.shape[1])+" attributes")
predict_dataset2=pd.DataFrame(predict_dataset.time.str.split(' ',1).tolist(),columns=['date','new_t
```

```
ime'])
predict_dataset3 = pd.concat([predict_dataset,predict_dataset2], axis=1)
del predict_dataset3['time']
predict_dataset3.rename(columns={'new_time':'time'},inplace=True)
predict_dataset3.head(5)
predict_dataset3.to_csv("G://DMA_PROJECT//weibo_predict.csv",sep=',',index=False,encoding='utf-8')
```

1. Analysis of Train Dataset

Contribution: All

In [ ]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%pylab inline
import copy
from googletrans import Translator
import emoji
import re
Names = ['u_id','m_id','time','content']
train_dataset= pd.read_fwf("G://DMA_PROJECT//weibo_train_data.txt",header=None,names=Names
,encoding='utf-8',delimiter="\t")
# fixed width formatted lines.
train_dataset.head(10)
print("Predict_Dataset has "+str(train_dataset.shape[0])+" records")
print("Predict_Dataset has "+str(train_dataset.shape[1])+" attributes")
train_content_mix=train_dataset['content']
train_content_split = pd.DataFrame(train_content_mix.str.split('\t',expand=True))
print(train_content_split.head(5))
train_dataset2 = pd.concat([train_dataset,train_content_split], axis=1)
print(train_dataset2.head(5))
del train_dataset2['content']
train_dataset2.rename(columns={0:'forward_count'},inplace=True)
train_dataset2.rename(columns={1:'comment_count'},inplace=True)
train_dataset2.rename(columns={2:'like_count'},inplace=True)
train_dataset2.rename(columns={3:'content'},inplace=True)
train_dataset2.head(5)
translate_dataframe = pd.DataFrame(data=train_dataset2['content'].head(10))
translator = Translator()
translate_dataframe["English_content"] = translate_dataframe['content'].map(lambda x: translator.tr
anslate(x, src="zh-CN", dest="en").text)
print(translate_dataframe)

train_dataset2.head(30)
train_dataset2.forward_count.describe()
train_dataset2.like_count.describe()

train_dataset3=pd.DataFrame(train_dataset2.time.str.split(' ',1).tolist(),columns=['date','new_time
'])
train_dataset3.head()
train_dataset4 = pd.concat([train_dataset2,train_dataset3], axis=1)
del train_dataset4['time']
train_dataset4.rename(columns={0:'forward_count'},inplace=True)
train_dataset4.rename(columns={1:'comment_count'},inplace=True)
train_dataset4.rename(columns={2:'like_count'},inplace=True)
train_dataset4.rename(columns={'new_time':'time'},inplace=True)
train_dataset4.head(5)
```

In [ ]:

```
#Month vs Like_Count
train_dataset5=train_dataset4.sort_values('date',ascending=True)
train_dataset5['like_count']=train_dataset5['like_count'].astype(float)
train_dataset5['month']=pd.DatetimeIndex(train_dataset5['date']).month
plt.plot(train_dataset5['month'],train_dataset5['like_count'])
plt.xticks(rotation='vertical')
```

In [ ]:

```
#Month vs Forward_Count
train_dataset5=train_dataset4.sort_values('date',ascending=True)
train_dataset5['forward_count']=train_dataset5['forward_count'].astype(float)
train_dataset5['month']=pd.DatetimeIndex(train_dataset5['date']).month
plt.plot(train_dataset5['month'],train_dataset5['forward_count'])
plt.xticks(rotation='vertical')
```

In [ ]:

```
#Month vs Comment_Count
train_dataset5=train_dataset4.sort_values('date',ascending=True)
train_dataset5['comment_count']=train_dataset5['comment_count'].astype(float)
train_dataset5['month']=pd.DatetimeIndex(train_dataset5['date']).month
plt.plot(train_dataset5['month'],train_dataset5['comment_count'])
plt.xticks(rotation='vertical')
```

In [ ]:

```
#Year vs Like_Count
train_dataset5=train_dataset4.sort_values('date',ascending=True)
train_dataset5['like_count']=train_dataset5['like_count'].astype(float)
train_dataset5['year']=pd.DatetimeIndex(train_dataset5['date']).year
plt.plot(train_dataset5['year'],train_dataset5['like_count'])
plt.xticks(rotation='vertical')
```

In [ ]:

```
#Hour vs Like_Count
train_dataset5=train_dataset4.sort_values('time',ascending=True)
train_dataset5['like_count']=train_dataset5['like_count'].astype(float)
train_dataset5['hour']=pd.DatetimeIndex(train_dataset5['time']).hour
plt.plot(train_dataset5['hour'],train_dataset5['like_count'])
```

In [ ]:

```
#Hour vs forward_Count
train_dataset5=train_dataset4.sort_values('time',ascending=True)
train_dataset5['forward_count']=train_dataset5['forward_count'].astype(float)
train_dataset5['hour']=pd.DatetimeIndex(train_dataset5['time']).hour
plt.plot(train_dataset5['hour'],train_dataset5['forward_count'])
plt.xticks(np.arange(0,23,1),rotation='vertical')
```

In [ ]:

```
#Hour vs comment_Count
train_dataset5=train_dataset4.sort_values('time',ascending=True)
train_dataset5['comment_count']=train_dataset5['comment_count'].astype(float)
train_dataset5['hour']=pd.DatetimeIndex(train_dataset5['time']).hour
plt.plot(train_dataset5['hour'],train_dataset5['comment_count'])
plt.xticks(np.arange(0,23,1),rotation='vertical')
```

Combine Dataset Analysis

Contribution: All

In [ ]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%pylab inline
import copy
from googletrans import Translator
import emoji

Populating the interactive namespace from numpy and matplotlib
df1=pd.read_csv("G://DMA_PROJECT//weibo_train1.csv")
df2=pd.read_csv("G://DMA_PROJECT//weibo_train2.csv")
```

```
df2=pd.read_csv("G://DMA_PROJECT//weibo_train2.csv")

predict_dataset=pd.read_csv("G://DMA_PROJECT//weibo_predict.csv")

frames=[df1,df2]
train_dataset=pd.concat(frames)
train_dataset.head(30)
predict_dataset.head(5)

#translateion
translator = Translator()
predict_dataset["en-content"] = predict_dataset['content'].head(10).map(lambda x: translator.transl
ate(x, src="zh-CN", dest="en").text)
predict_dataset.head(10)

translator = Translator()
train_dataset["en-content"] = train_dataset['content'].head(10).map(lambda x: translator.translate(
x, src="zh-CN", dest="en").text)
train_dataset.head(10)


train_dataset.head(614809).to_csv("G://DMA_PROJECT//weibo_train1_trans.csv",sep=',',index=False,enc
oding='utf-8')
train_dataset.tail(614809).to_csv("G://DMA_PROJECT//weibo_train2_trans.csv",sep=',',index=False,enc
oding='utf-8')
predict_dataset.to_csv("G://weibo_predict_trans.csv",sep=',',index=False,encoding='utf-8')


train_dataset=train_dataset.drop(['uni-content'],axis=1)
tcount=train_dataset.groupby(by='u_id',as_index=False).agg({'content':pd.Series.nunique})
print(tcount)

tcount.to_csv("G://DMA_PROJECT//weibo_count_uid.csv",index=False,encoding='utf-8')
query= "ffdd80d2f1023779e30956c34b044b25"
train_dataset[train_dataset['u_id']==query]
pcount=predict_dataset.groupby(by='u_id',as_index=False).agg({'content':pd.Series.nunique})
print(pcount)

query= "ffdd80d2f1023779e30956c34b044b25"
predict_dataset[predict_dataset['u_id']==query]
tdcount=train_dataset.groupby(by='date',as_index=False).agg({'content':pd.Series.nunique})
print(tdcount)


tdcount.to_csv("G://DMA_PROJECT//weibo_train_date_content.csv",index=False,encoding='utf-8')
ttcount=train_dataset.groupby(by='time',as_index=False).agg({'content':pd.Series.nunique}
print(ttcount)

ttcount.to_csv("G://DMA_PROJECT//weibo_train_time_content.csv",index=False,encoding='utf-8')
pcount.to_csv("G://DMA_PROJECT//weibo_pcount_uid.csv",index=False,encoding='utf-8')
tlcount=train_dataset.groupby(by='u_id',as_index=False).agg({'like_count':pd.Series.nunique})
print(tlcount)

tccount=train_dataset.groupby(by='u_id',as_index=False).agg({'comment_count':pd.Series.nunique})
print(tccount)


tfcount=train_dataset.groupby(by='u_id',as_index=False).agg({'forward_count':pd.Series.nunique})
print(tfcount)


resultcount=pd.merge(tfcount,tccount,on='u_id')
resultcount2=pd.merge(resultcount,tlcount,on='u_id')
print(resultcount2)

resultcount2.to_csv("G://DMA_PROJECT//weibo_tcount_fcl.csv",index=False,encoding='utf-8')

train_dataset['content_media_count']=train_dataset['content'].str.count('http')
predict_dataset['content_media_count']=predict_dataset['content'].str.count('http')
train_dataset.head(5)
```

---------- DATA PRE-PROCESSING----------

Generate Best Statistical Factors

Contribution: Ashish Kar

Information on Loaded Modules

We will find Mean, Median, Max and Min of Forward, Comment and Likes for every unique UID in train dataset for our further statistical analysis

In [ ]:

```
df=pd.read_csv("train_uid_stat.csv")
```

Example For UID stats

Say in train dataset, For UID x there are two MID(ie two posts):

Train Dataset:

UID Stats:

Now Consider that same user has 4 mids in predict dataset, so prediction of FCL by factor "mean" will be as follows:

Predict Dataset

Similary by factor "max" :

Predict Dataset

Predict with fixed Value

1. Default Values

About 80% of the training data are: 0 0 0 (forward_count,comment_count,like_count) and also, 96% of uid in predict dataset is present in train dataset, for remaining 4% which are new, we need some default values. inspired by this, we try some fixed value for all uid:

Function to take Fixed FCL Values, Give Accuracy and Generate Predicted FCL

In [ ]:

```
@runTime
def predict_with_fixed_value(forward,comment,like,submission=True):
 # type check
 if isinstance(forward,int) and isinstance(forward,int) and isinstance(forward,int):
  pass
 else:
  raise TypeError("forward,comment,like should be type 'int' ")

 traindata,testdata = loadData()

 #score on the training set
 train_real_pred = traindata[['forward_count','comment_count','like_count']]
 train_real_pred['fp'],train_real_pred['cp'],train_real_pred['lp'] = forward,comment,like
 print ("Score on the training set:{0:.2f}%".format(precision(train_real_pred.values)*100))

 #predict on the test data with fixed value, generate submission file
 if submission:
  test_pred = testdata[['u_id','m_id']]
```

```
 test_pred['fp'],test_pred['cp'],test_pred['lp'] = forward,comment,like

 result = []
 filename = "weibo_predict_{}_{}_{}.txt".format(forward,comment,like)
 for _,row in test_pred.iterrows():
  result.append("{0}\t{1}\t{2},{3},{4}\n".format(row[0],row[1],row[2],row[3],row[4]))
 f = open(filename,'w')
 f.writelines(result)
 f.close()
 print ('generate submission file "{}"'.format(filename))
```

2. UID Statistics (Mean, Max, Min, Median)

Another wise solution is to predict respectively with uid's statistics(E.g mean,median) , their score on the training data:

Function to take Statistical Factor, Give Accuracy and Generate Predicted FCL

In [ ]:

```
@runTime
def predict_with_stat(stat="median",submission=True):
 """
 stat:
  string
  min,max,mean,median
 """
 stat_dic = genUidStat()
 traindata,testdata = loadData()

 #get stat for each uid
 forward,comment,like = [],[],[]
 for uid in traindata['u_id']:
  if uid in stat_dic:
   forward.append(int(stat_dic[uid]["forward_"+stat]))
   comment.append(int(stat_dic[uid]["comment_"+stat]))
   like.append(int(stat_dic[uid]["like_"+stat]))
  else:
   forward.append(0)
   comment.append(0)
   like.append(0)
 #score on the training set
 train_real_pred = traindata[['forward_count','comment_count','like_count']]
 train_real_pred['fp'],train_real_pred['cp'],train_real_pred['lp'] = forward,comment,like
 print ("Score on the training set:{0:.2f}%".format(precision(train_real_pred.values)*100))

 #predict on the test data with fixed value, generate submission file
 if submission:
  test_pred = testdata[['u_id','m_id']]
  forward,comment,like = [],[],[]
  for uid in testdata['u_id']:
   if uid in stat_dic:
    forward.append(int(stat_dic[uid]["forward_"+stat]))
    comment.append(int(stat_dic[uid]["comment_"+stat]))
    like.append(int(stat_dic[uid]["like_"+stat]))
   else:
    forward.append(0)
    comment.append(0)
    like.append(0)


  test_pred['fp'],test_pred['cp'],test_pred['lp'] = forward,comment,like

  result = []
  filename = "weibo_predict_{}.txt".format(stat)
  for _,row in test_pred.iterrows():
   result.append("{0}\t{1}\t{2},{3},{4}\n".format(row[0],row[1],row[2],row[3],row[4]))
  f = open(filename,'w')
  f.writelines(result)
  f.close()
  print ('generate submission file "{}"'.format(filename))
```

In [ ]:

```
if __name__ == "__main__":
  predict_with_stat(stat="median",submission=True)
```

In [ ]:

```
if __name__ == "__main__":
  predict_with_fixed_value(0,1,1,submission=True)
```

In [ ]:

```
if __name__ == "__main__":
  predict_with_stat(stat="mean",submission=True)
```

In [ ]:

```
if __name__ == "__main__":
  predict_with_stat(stat="max",submission=True)
```

In [ ]:

```
if __name__ == "__main__":
  predict_with_stat(stat="min",submission=True)
```

In [ ]:

```
if __name__ == "__main__":
  predict_with_fixed_value(0,0,0,submission=True)
```

In [ ]:

```
if __name__ == "__main__":
  predict_with_fixed_value(0,0,1,submission=True)
```

In [ ]:

```
if __name__ == "__main__":
  predict_with_fixed_value(0,1,0,submission=True)
```

In [ ]:

```
if __name__ == "__main__":
  predict_with_fixed_value(1,0,0,submission=True)
```

In [ ]:

```
if __name__ == "__main__":
  predict_with_fixed_value(1,0,1,submission=True)
```

In [ ]:

```
if __name__ == "__main__":
  predict_with_fixed_value(1,1,0,submission=True)
```

In [ ]:

```
if __name__ == "__main__":
  predict_with_fixed_value(1,1,1,submission=True)
```

Inference

Generate More Features from Content / Media and Symbol

Generate More Factors from Content (Media and Symbol Counts)

In [ ]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%pylab inline
import copy
from googletrans import Translator
import emoji
import regex

df1=pd.read_csv("G://DMA_PROJECT//weibo_train1.csv")
df2=pd.read_csv("G://DMA_PROJECT//weibo_train2.csv")
frames=[df1,df2]
train_dataset=pd.concat(frames)
predict_dataset=pd.read_csv("G://DMA_PROJECT//weibo_predict.csv")

train_dataset.head(10)

train_dataset['content_media_count'].value_counts()

train_dataset['content_media_count'].value_counts().plot(kind='bar')

train_dataset=train_dataset.sort_values('content_media_count',ascending=True)
train_dataset['like_count']=train_dataset['like_count'].astype(float)
plt.plot(train_dataset['content_media_count'],train_dataset['like_count'])
plt.xticks(np.arange(0,10,1))

train_dataset=train_dataset.sort_values('content_media_count',ascending=True)
train_dataset['forward_count']=train_dataset['forward_count'].astype(float)
plt.plot(train_dataset['content_media_count'],train_dataset['forward_count'])
plt.xticks(np.arange(0,10,1))

train_dataset=train_dataset.sort_values('content_media_count',ascending=True)
train_dataset['comment_count']=train_dataset['comment_count'].astype(float)
plt.plot(train_dataset['content_media_count'],train_dataset['comment_count'])
plt.xticks(np.arange(0,10,1))

train_dataset['content_#_count'].value_counts().plot(kind='bar')

train_dataset['content_#_count']=train_dataset['non_emoji_content'].str.count("#")
train_dataset['content_@_count']=train_dataset['non_emoji_content'].str.count("@")
train_dataset['content_?_count']=train_dataset['non_emoji_content'].str.count("\?")
train_dataset['content_!_count']=train_dataset['non_emoji_content'].str.count("!")

train_dataset.head(10)

train_dataset['content_length']=train_dataset['content'].str.len()

train_dataset.head(10)


train_dataset['emoji_count']=train_dataset['non_emoji_content'].str.count(str(emoji.UNICODE_EMOJI.k
eys()))

train_dataset.head(30)

print(emoji.UNICODE_EMOJI.keys())

import re
train_dataset['emoji']=train_dataset['content_spchar'].str.replace(r'[\U0001F602-
\U0001F64F]','emoticon')

train_dataset.head(30)

train_dataset['emoji_count']=train_dataset['emoji'].str.count("emoticon")

train_dataset.head(30)

train_dataset.drop('content_spchar',axis=1,inplace=True)
train_dataset.drop('non_emoji_content',axis=1,inplace=True)
train_dataset.drop('emoji',axis=1,inplace=True)
```

```
train_dataset.rename(columns={'emoji_count':'content_emoji_count'},inplace=True)

train_dataset.head(30)

train_dataset.head(614809).to_csv("G://DMA_PROJECT//weibo_train1_cp.csv",sep=',',index=False,encoding='utf-8')
train_dataset.tail(614809).to_csv("G://DMA_PROJECT//weibo_train2_cp.csv",sep=',',index=False,encoding='utf-8')
```

Translation

Contribution: Arundati Dixit

In [ ]:

```
train_dataset= pd.read_fwf("G:\\weibo_train_data.txt",header=None,names=Names ,encoding='utf-8',delimiter="\t")
```

In [ ]:

```
emoji_pattern=re.compile("["
                         u"\U0001F600-\U0001F64F"
                         u"\U0001F300-\U0001F5FF"
                         u"\U0001F680-\U0001F6FF"
                         u"\U0001F1E0-\U0001F1FF"
                         "]+",flags=re.UNICODE)
train_dataset2['content']=emoji_pattern.sub(r'',str(train_dataset2['content']))
```

In [ ]:

```
translate_dataframe = pd.DataFrame(data=train_dataset2['content'].head(30))
translator = Translator()
translate_dataframe["English_content"] = translate_dataframe['content'].map(lambda x: translator.translate(x, src="zh-CN", dest="en").text)
```

The data is split into 400 chunks and translated individually and concatenated back

In [ ]:

```
for j in range(0,400):
    filename="G:\\concatfiles\\f"+str(j)+".txt"
    transname="G:\\translated\\ts"+str(j)+".zh-CN.en.txt"
    print(filename)
    print(transname)
    f=pd.read_csv(filename)
    t= pd.read_csv(transname,sep="5A09")
    frames = [f,t]

    df=(pd.concat(frames, join='outer', ignore_index=False,keys=None, levels=None, names=None, verify_integrity=False, copy=True, axis=1))
    translated=translated.append(df)
```

Text Preprocessing

Contribution: Apoorva Malemath

In [ ]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%pylab inline
import copy
from googletrans import Translator
import pandas as pd
```

```
import numpy as np
import csv
import re
import jieba
import time
import json
from sklearn.feature_extraction.text import CountVectorizer
from sklearn import linear_model
from sklearn.externals import joblib
from nltk.corpus import stopwords as e_stopwords
from datetime import datetime, timedelta
import jieba
import sys

from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
```

In [ ]:

```
train1= pd.read_csv("G:\\preprocessed_1.csv")
train2= pd.read_csv("G:\\preprocessed_2.csv")
frames=[train1,train2]
train=pd.concat(frames)
```

TEXT PREPROCESSING

REMOVAL OF NOISE - URL

In [ ]:

```
def remurl(content):
    try:
        URLless_string = re.sub(r'\w+:\/{2}[\d\w-]+(\.[\d\w-]+)*(?:(?:\/[^\s/]*))*', '', content)
        return URLless_string
    except Exception as e:
        print(str(e))
        return content
```

In [ ]:

```
df_urlrem = pd.DataFrame(columns=['en_contenturl','url_rem'])
for i in range(100000):
    non_emo=translated['en_content'].iloc[i]
    content=translated['en_content'].iloc[i]
    new_content=remurl(content)

    df_urlrem = df_urlrem.append({'en_contenturl': non_emo,'url_rem':new_content},
ignore_index=True)
```

Removal of numbers

In [ ]:

```
def rem_num(tokens):
        try:
            for item in tokens:
                if item.isdigit():
                    tokens.remove(item)
            return tokens
        except Exception as e:
            print(str(e))
            return tokens
```

In [ ]:

```
df_remnum = pd.DataFrame(columns=['url_rem',])
for i in range(100000):
    content=df_urlrem['url_rem'].iloc[i]
    nonum=rem_num(df_urlrem['url_rem'].iloc[i])
    list1=[content,nonum]
    df_remnum = df_remnum.append({'url_rem': content, 'no_num': nonum}, ignore_index=True)
```

REMOVAL OF STOPWORDS

In [ ]:

```
remStopword=pd.DataFrame()
```

In [ ]:

```
def removeStopwords(data):
    stopw="STOPWORDS.txt"
    stop_words = set(stopwords.words('english'))
    words = word_tokenize(data)
    wordsFiltered = []
    try:
        for w in words:
            if (w not in stopw or w not in stop_words) :
                wordsFiltered.append(w)
        return wordsFiltered
    except Exception as e:
        print(str(e))
        return data
```

In [ ]:

```
df_new = pd.DataFrame(columns=['no_num','Stopwrod_removed'])
for i in range(100000):
    non_emo=df_remnum['no_num'].iloc[i]
    letters_only = re.sub("[^a-zA-Z]"," ",str(df_remnum['no_num'].iloc[i]))
    remStopword=removeStopwords(letters_only)
    list1=[non_emo,remStopword]
    df_new = df_new.append({'no_num': non_emo, 'Stopword_removed': remStopword}, ignore_index=True)
```

STEMMING

In [ ]:

```
import nltk
from nltk.stem.porter import PorterStemmer
porter_stemmer = PorterStemmer()
```

In [ ]:

```
def stemming(tokens):
    # First Word tokenization
    nltk_tokens =tokens
    stem = []
    #Next find the roots of the word
    try:
        for w in nltk_tokens:
            s=porter_stemmer.stem(w)
            stem.append(s)
        return stem
    except Exception as e:
        print(str(e))
        return tokens
```

In [ ]:

```
df_stem = pd.DataFrame(columns=['en_contentst','Stemming'])
```

```
for i in range(100000):
    content=df_new['Stopword_removed'].iloc[i]
    stem=stemming(df_new['Stopword_removed'].iloc[i])
    list1=[content,stem]
    df_stem = df_stem.append({'en_contentst': content, 'Stemming': stem}, ignore_index=True)
```

LEMMATIZATION

In [ ]:

```
###LEMMATIZATION
import nltk
from nltk.stem import WordNetLemmatizer
```

In [ ]:

```
def lemmatization(tokens):
    wordnet_lemmatizer = WordNetLemmatizer()
    nltk_tokens =tokens
    lem = []
    #Next find the roots of the word
    try:
        for w in nltk_tokens:
            l=wordnet_lemmatizer.lemmatize(w)
            lem.append(l)
        return lem
    except Exception as e:
        print(str(e))
        return tokens
```

In [ ]:

```
df_lem = pd.DataFrame(columns=['Stemmingle','lemmatization'])
for i in range(100000):
    content=df_stem['Stemming'].iloc[i]
    lem=stemming(df_stem['Stemming'].iloc[i])
    list1=[content,lem]
    df_lem = df_lem.append({'Stemmingle': content, 'lemmatization': lem}, ignore_index=True)
```

Converting to lower case

In [ ]:

```
def tolower(tokens):
    try:
        nltk_tokens=tokens
        x = [element.lower() for element in nltk_tokens]
        return x
    except Exception as e:
        print(str(e))
        return tokens
```

In [ ]:

```
df_lower = pd.DataFrame(columns=['lemmatizationtl','lower'])
for i in range(100000):
    content=df_lem['lemmatization'].iloc[i]
    low=tolower(df_lem['lemmatization'].iloc[i])
    list1=[content,low]
    df_lower = df_lower.append({'lemmatizationtl': content, 'lower': low}, ignore_index=True)
```

REMOVE PUNTUATION

In [ ]:

```
def rem_punctuation(tokens):
        try:
            input_text = ' '.join(tokens).lower()
            s = re.sub(r"[-()\"#/@;:<>{}`+=~|.!?,]", "", input_text)
            #print(input_text)
            words = word_tokenize(s)
            return words
        except Exception as e:
            print(str(e))
            return tokens
```

In [ ]:

```
df_rempunc = pd.DataFrame(columns=['lemmatizationtlp','no_punc'])
for i in range(100000):
    content=df_lower['lemmatizationtl'].iloc[i]
    nopun=rem_punctuation(df_lower['lemmatizationtl'].iloc[i])
    list1=[content,nopun]
    df_rempunc = df_rempunc.append({'lemmatizationtlp': content, 'no_punc': nopun}, ignore_index=Tr
ue)
```

In [ ]:

```
frames=[translated,df_urlrem, df_new, df_stem, df_lem, df_lower, df_remnum, df_rempunc]
```

In [ ]:

```
Train=(pd.concat(frames, axis=1))
```

----------MODEL BUILDING----------

BOW

Contribution: Apoorva Malemath

We convert text to a numerical representation called a feature vector. A feature vector can be as simple as a list of numbers.

The bag-of-words model is one of the feature extraction algorithms for text.

1.The first step in this model is defining the vocabulary

2.The second step is to convert sentences into a frequency vector based on the vocabulary.

In [ ]:

```
#Reading data from document
import pandas as pd
df_pre=pd.read_csv("E:\\DMA_PRE\\PREPROCESSED.csv")
df=pd.read_csv("E:\\DMA_PRE\\PREPROCESSED.csv")
#Adjustments to be done for the data
df['content']=df['content'].str.replace(",", "")
df['content']=df['content'].str.replace("'","")
```

In [ ]:

```
#creating a list for all content
l=[]
for i in range(0,10000):
    l.append(df['content'].iloc[i])
l
```

In [ ]:

```
#code for bag of words model
import numpy as np
import re
```

```
import re

#for building vocabulary
def tokenize_sentences(sentences):
    words = []
    for sentence in sentences:
        w = extract_words(sentence)
        words.extend(w)

    words = sorted(list(set(words)))
    return words

def extract_words(sentence):
    ignore_words = ['a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t',
'u','v','w','x','y','z','A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S',
'T','U','V','W','X','Y','Z']
    words = re.sub("[^\w]", " ",  sentence).split() #nltk.word_tokenize(sentence)
    words_cleaned = [w.lower() for w in words if w not in ignore_words]
    return words_cleaned

#function which returns feature vector
def bagofwords(sentence, words):
    sentence_words = extract_words(sentence)
    # frequency word count
    bag = np.zeros(len(words),dtype=int)
    for sw in sentence_words:
        for i,word in enumerate(words):
            if word == sw:
                bag[i] += 1

    return np.array(bag)
```

In [ ]:

```
#building the vocabulary for the list created
vocabulary1 = tokenize_sentences(l)
```

In [ ]:

```
l1 = [x for x in vocabulary1 if not (x.isdigit() or x[1:].isdigit())]
```

In [ ]:

```
b=pd.DataFrame()
```

In [ ]:

```
#constructing bag of words
a=[]

for i in range(0,10000):
    #b.append(bagofwords(df['content'].iloc[i], vocabulary1),ignore_index=True)
    a.append(bagofwords(df['content'].iloc[i], vocabulary1))
```

In [ ]:

```
bow=np.asarray(a)
```

In [ ]:

```
df_pol=pd.read_csv("E:\\DMA_PRE\\weibo_polarity.csv")
```

Linear Regression Model with BOW appened with other features

In [ ]:

```
bow1=np.insert(bow,16792,df_pol["content_media_count"],axis=1)
```

```
bow2=np.insert(bow1,16793,df_pol["forward_median"],axis=1)
bow3=np.insert(bow2,16794,df_pol["comment_median"],axis=1)
bow4=np.insert(bow3,16795,df_pol["like_median"],axis=1)
bow5=np.insert(bow4,16796,df_pol["polarity"],axis=1)

X_train1=train_bow
X_test1=pred_bow
Y_train1=train_df[["forward_count","like_count","comment_count"]]
Y_test1=predict_df[["forward_count"]]

lm=linear_model.LinearRegression()
model=lm.fit(X_train1,Y_train1)
pred1=lm.predict(X_test1)
pred1=pred1.round()
pred1=(np.maximum(pred1,0.))

np.savetxt("E://DMA_PRE//weibo_predict_resultbow.csv",pred1,delimiter=',',header="forward_count,com
ment_count,like_count",comments="")
result=pd.read_csv("E://DMA_PRE//weibo_predict_resultbow.csv")

train_real_pred = Y_test1
train_real_pred['fp']=result['forward_count'].values
train_real_pred['cp']=result['comment_count'].values
train_real_pred['lp']=result['like_count'].values
train_real_pred=train_real_pred.round()
print ("Score on the training set:{0:.2f}%".format(precision(train_real_pred.values)*100))
```

BAG OF WORDS USING COUNTER VECTORIZER

Contribution: Apoorva Malemath

In [ ]:

```
df1=pd.read_csv('E:\\DMA_PRE\\pre_bow.csv')
train_df=df1[0:8000]
train_df.shape
```

In [ ]:

```
train_l=[]
for i in range(0,8000):
    train_l.append(df_pre['content'].iloc[i])
len(train_l)
pred_l=[]
for i in range(8001,10000):
    pred_l.append(df_pre['content'].iloc[i])
len(pred_l)
```

In [ ]:

```
from sklearn.feature_extraction.text import CountVectorizer
```

In [ ]:

```
vect=CountVectorizer()
vect.fit(train_l)
x=vect.transform(train_l)
vect = CountVectorizer(analyzer = "word",   \
                       tokenizer = None,    \
                       preprocessor = None, \
                       stop_words = None,   \
                       max_features = 100)
```

In [ ]:

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn import linear_model
from sklearn.metrics import precision_score
```

Model 1

In [ ]:

```
off_train_data_features = vect.fit_transform(train_l)
off_train_data_features = off_train_data_features.toarray()
off_train_data_forward = train_df.forward_count

off_test_data_features = vect.fit_transform(pred_l)
off_test_data_features = off_test_data_features.toarray()
off_test_data_forward = predict_df.forward_count

X_train1=off_train_data_features
X_test1= off_test_data_features
Y_train1=dftrain[["forward_count","like_count","comment_count"]]
Y_test1=dfcv[["forward_count","like_count","comment_count"]]

lm=linear_model.LinearRegression()
model=lm.fit(X_train1,Y_train1)
pred1=lm.predict(X_test1)
pred1=pred1.round()
pred1=(np.maximum(pred1,0.))
np.savetxt("E://DMA_PRE//weibo_predict_resultbow1.csv",pred1,delimiter=',',header="forward_count,cc
mment_count,like_count",comments="")
result1=pd.read_csv("E://DMA_PRE//weibo_predict_resultbow1.csv")
result1=result1.abs()
result1=result1.astype(int)
train_real_pred = Y_test1
train_real_pred['fp']=result1['forward_count'].values
train_real_pred['cp']=result1['comment_count'].values
train_real_pred['lp']=result1['like_count'].values
train_real_pred=train_real_pred.round()
print ("Score on the training set:{0:.2f}%".format(precision(train_real_pred.values)*100))
```

Model 2

In [ ]:

```
train=df_pol[0:8000]
cv=df_pol[8001:10000]
```

In [ ]:

```
off_train_data_features = vect.fit_transform(train_l)
off_train_data_features = off_train_data_features.toarray()

off_train_data_features1=np.insert(off_train_data_features,100,train["content_media_count"],axis=1
)
off_train_data_features2=np.insert(off_train_data_features1,101,train["forward_median"],axis=1)
off_train_data_features3=np.insert(off_train_data_features2,102,train["comment_median"],axis=1)
off_train_data_features4=np.insert(off_train_data_features3,103,train["like_median"],axis=1)
#off_train_data_features5=np.insert(off_train_data_features4,100,train["polarity"],axis=1)
off_train_data_features6=np.insert(off_train_data_features4,104,train["content_emoji_count"],axis=
1)
#off_train_data_forward = train_df.forward_count

off_test_data_features = vect.fit_transform(pred_l)
off_test_data_features = off_test_data_features.toarray()
off_test_data_features1=np.insert(off_test_data_features,100,cv["content_media_count"],axis=1)
off_test_data_features2=np.insert(off_test_data_features1,101,cv["forward_median"],axis=1)
off_test_data_features3=np.insert(off_test_data_features2,102,cv["comment_median"],axis=1)
off_test_data_features4=np.insert(off_test_data_features3,103,cv["like_median"],axis=1)
#off_test_data_features5=np.insert(off_test_data_features4,100,cv["polarity"],axis=1)
off_test_data_features6=np.insert(off_test_data_features4,104,cv["content_emoji_count"],axis=1)
#off_test_data_forward = predict_df.forward_count

X_train1=off_train_data_features6
X_test1= off_test_data_features6
Y_train1=dftrain[["forward_count","like_count","comment_count"]]
Y_test1=dfcv[["forward_count","like_count","comment_count"]]

lm=linear_model.LinearRegression()
```

```
lm=linear_model.LinearRegression()
model=lm.fit(X_train1,Y_train1)
pred1=lm.predict(X_test1)
pred1=pred1.round()
pred1=(np.maximum(pred1,0.))

np.savetxt("E://DMA_PRE//weibo_predict_resultbow3.csv",pred1,delimiter=',',header="forward_count,cc
mment_count,like_count",comments="")
result3=pd.read_csv("E://DMA_PRE//weibo_predict_resultbow3.csv")
result3=result3.abs()
result3=result3.astype(int)
train_real_pred = Y_test1
train_real_pred['fp']=result3['forward_count'].values
train_real_pred['cp']=result3['comment_count'].values
train_real_pred['lp']=result3['like_count'].values
train_real_pred=train_real_pred.round()
print ("Score on the training set:{0:.2f}%".format(precision(train_real_pred.values)*100))
```

Model 3

In [ ]:

```
off_train_data_features = vect.fit_transform(train_l)
off_train_data_features = off_train_data_features.toarray()

off_train_data_features1=np.insert(off_train_data_features,100,train["content_media_count"],axis=1
)
off_train_data_features2=np.insert(off_train_data_features1,101,train["forward_median"],axis=1)
off_train_data_features3=np.insert(off_train_data_features2,102,train["comment_median"],axis=1)
off_train_data_features4=np.insert(off_train_data_features3,103,train["like_median"],axis=1)
off_train_data_features5=np.insert(off_train_data_features4,104,train["polarity"],axis=1)

#off_train_data_forward = train_df.forward_count

off_test_data_features = vect.fit_transform(pred_l)
off_test_data_features = off_test_data_features.toarray()
off_test_data_features1=np.insert(off_test_data_features,100,cv["content_media_count"],axis=1)
off_test_data_features2=np.insert(off_test_data_features1,101,cv["forward_median"],axis=1)
off_test_data_features3=np.insert(off_test_data_features2,102,cv["comment_median"],axis=1)
off_test_data_features4=np.insert(off_test_data_features3,103,cv["like_median"],axis=1)
off_test_data_features5=np.insert(off_test_data_features4,104,cv["polarity"],axis=1)

#off_test_data_forward = predict_df.forward_count

X_train1=off_train_data_features4
X_test1= off_test_data_features4
Y_train1=dftrain["forward_count"]
Y_test1=dfcv["forward_count"]

lm=linear_model.LinearRegression()
model=lm.fit(X_train1,Y_train1)
pred1=lm.predict(X_test1)
pred1=pred1.round()
pred1=(np.maximum(pred1,0.))

np.savetxt("E://DMA_PRE//weibo_predict_resultbow4.csv",pred1,delimiter=',',header="forward_count",
comments="")
result4=pd.read_csv("E://DMA_PRE//weibo_predict_resultbow4.csv")
result4=result4.abs()
result4=result4.astype(int)

train_real_pred = Y_test1
```

In [ ]:

```
off_train_data_features = vect.fit_transform(train_l)
off_train_data_features = off_train_data_features.toarray()

off_train_data_features1=np.insert(off_train_data_features,100,train["content_media_count"],axis=1
)
off_train_data_features2=np.insert(off_train_data_features1,101,train["forward_median"],axis=1)
off_train_data_features3=np.insert(off_train_data_features2,102,train["comment_median"],axis=1)
off_train_data_features4=np.insert(off_train_data_features3,103,train["like_median"],axis=1)
off_train_data_features5=np.insert(off_train_data_features4,104,train["polarity"],axis=1)
```

```
#off_train_data_forward = train_df.forward_count

off_test_data_features = vect.fit_transform(pred_l)
off_test_data_features = off_test_data_features.toarray()
off_test_data_features1=np.insert(off_test_data_features,100,cv["content_media_count"],axis=1)
off_test_data_features2=np.insert(off_test_data_features1,101,cv["forward_median"],axis=1)
off_test_data_features3=np.insert(off_test_data_features2,102,cv["comment_median"],axis=1)
off_test_data_features4=np.insert(off_test_data_features3,103,cv["like_median"],axis=1)
off_test_data_features5=np.insert(off_test_data_features4,100,cv["polarity"],axis=1)
#off_test_data_forward = predict_df.forward_count

X_train2=off_train_data_features4
X_test2= off_test_data_features4
Y_train2=dftrain["like_count"]
Y_test2=dfcv["like_count"]

lm=linear_model.LinearRegression()
model=lm.fit(X_train1,Y_train1)
pred1=lm.predict(X_test1)
pred1=pred1.round()
pred1=(np.maximum(pred1,0.))

np.savetxt("E://DMA_PRE//weibo_predict_resultbow5.csv",pred1,delimiter=',',header="comment_count",
comments="")
result5=pd.read_csv("E://DMA_PRE//weibo_predict_resultbow5.csv")
result5=result5.abs()
result5=result5.astype(int)
```

In [ ]:

```
off_train_data_features = vect.fit_transform(train_l)
off_train_data_features = off_train_data_features.toarray()

off_train_data_features1=np.insert(off_train_data_features,100,train["content_media_count"],axis=1
)
off_train_data_features2=np.insert(off_train_data_features1,101,train["forward_median"],axis=1)
off_train_data_features3=np.insert(off_train_data_features2,102,train["comment_median"],axis=1)
off_train_data_features4=np.insert(off_train_data_features3,103,train["like_median"],axis=1)
off_train_data_features5=np.insert(off_train_data_features4,100,train["polarity"],axis=1)
#off_train_data_forward = train_df.forward_count

off_test_data_features = vect.fit_transform(pred_l)
off_test_data_features = off_test_data_features.toarray()
off_test_data_features1=np.insert(off_test_data_features,100,cv["content_media_count"],axis=1)
off_test_data_features2=np.insert(off_test_data_features1,101,cv["forward_median"],axis=1)
off_test_data_features3=np.insert(off_test_data_features2,102,cv["comment_median"],axis=1)
off_test_data_features4=np.insert(off_test_data_features3,103,cv["like_median"],axis=1)
off_test_data_features5=np.insert(off_test_data_features4,100,cv["polarity"],axis=1)
#off_test_data_forward = predict_df.forward_count

X_train3=off_train_data_features4
X_test3= off_test_data_features4
Y_train3=dftrain["comment_count"]
Y_test3=dfcv["comment_count"]

lm=linear_model.LinearRegression()
model=lm.fit(X_train1,Y_train1)
pred1=lm.predict(X_test1)
pred1=pred1.round()
pred1=(np.maximum(pred1,0.))

np.savetxt("E://DMA_PRE//weibo_predict_resultbow6.csv",pred1,delimiter=',',header="like_count",com
ments="")
result6=pd.read_csv("E://DMA_PRE//weibo_predict_resultbow6.csv")
result6=result6.abs()
result6=result6.astype(int)
```

In [ ]:

```
train_real_pred = pd.concat([Y_test1,Y_test2,Y_test3],axis=1)
train_real_pred['fp']=result4['forward_count'].values
train_real_pred['cp']=result5['comment_count'].values
train_real_pred['lp']=result6['like_count'].values
train_real_pred=train_real_pred.round()
print ("Score on the training set:{0:.2f}%".format(precision(train_real_pred.values)*100))
```

```
print ('Score on the training set:{0:.2f}%'.format(precision(train_real_pred.values, 100))
```

Polarity

Contribution: Deepti

In [ ]:

```
import pandas as pd
import numpy as np
import re
from sklearn import linear_model
from sklearn.linear_model import Lasso
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from matplotlib import pyplot as plt
from textblob import TextBlob
import statsmodels.api as sm

import import_ipynb
from evaluation import precision
from runTime import runTime

df=pd.read_csv("G://preprocessed1L.csv")
df_new = pd.DataFrame(columns=['pol'])

for i in range(0,100000):
    try:
        a=TextBlob(df['no_punc'].iloc[i]).sentiment
        df_new=df_new.append({'pol':a[0]}, ignore_index=True)
    except Exception as e:
        print(str(e))
        df_new=df_new.append({'pol':999999}, ignore_index=True)

df['polarity']=df_new['pol']
```

Initial Modelling

Contribution: All

In [ ]:

```
df1=pd.read_csv("E:\\5th Sem\\DMA Project\\Model Evaluation\\weibo_train1_cp.csv")
df2=pd.read_csv("E:\\5th Sem\\DMA Project\\Model Evaluation\\weibo_train2_cp.csv")
frames=[df1,df2]
train_dataset=pd.concat(frames)
predict_dataset=pd.read_csv("E:\\5th Sem\\DMA Project\\Model Evaluation\\weibo_predict_cp.csv")
```

In [ ]:

```
train_dataset['date']=pd.to_datetime(train_dataset['date'],errors='coerce')
train_month=[g for n, g in train_dataset.groupby(pd.Grouper(key='date',freq='M'))]
train_dataset['time']=pd.to_datetime(train_dataset['time'],errors='coerce')
train_hour=[g for n, g in train_dataset.groupby(pd.Grouper(key='time',freq='H'))]
```

In [ ]:

```
train_month[0].to_csv("E:\\5th Sem\\DMA Project\\Model Evaluation\\weibo_train_feb_cp.csv",sep=','
,index=False,encoding='utf-8')
train_month[1].to_csv("E:\\5th Sem\\DMA Project\\Model Evaluation\\weibo_train_march_cp.csv",sep='
,',index=False,encoding='utf-8')
train_month[2].to_csv("E:\\5th Sem\\DMA Project\\Model Evaluation\\weibo_train_april_cp.csv",sep='
,',index=False,encoding='utf-8')
train_month[3].to_csv("E:\\5th Sem\\DMA Project\\Model Evaluation\\weibo_train_may_cp.csv",sep=','
,index=False,encoding='utf-8')
train_month[4].to_csv("E:\\5th Sem\\DMA Project\\Model
Evaluation\\weibo_train_june_cp.csv",sep=',',index=False,encoding='utf-8')
train_month[5].to_csv("E:\\5th Sem\\DMA Project\\Model
Evaluation\\weibo_train_july_cp.csv",sep=',',index=False,encoding='utf-8')
```

In [ ]:

```
i=0
for i in range(0,24):
    path="E:\\5th Sem\\DMA Project\\Model Evaluation\\weibo_train_hour_"+str(i)+"_cp.csv"
    train_hour[i].to_csv(path,sep=',',index=False,encoding='utf-8')
```

In [ ]:

```
frames1=[train_month[0],train_month[1],train_month[2],train_month[3],train_month[4]]
train=pd.concat(frames1)
predict=train_month[5]
```

4. Initial Predictions without Model and Analysis

Putting known values of stats in predict dataset without any computation and finding accuracy

Best Statistical Factors and Default Value

Model 1 (Factors: Media, #, @, ?, !, Length, Emoji)

In [ ]:

```
X_train=train[["content_media_count","content_#_count","content_@_count","content_?_count","conten
t_!_count","content_length","content_emoji_count"]]
Y_train=train[["forward_count","comment_count","like_count"]]
X_test=predict[["content_media_count","content_#_count","content_@_count","content_?_count","conte
nt_!_count","content_length","content_emoji_count"]]
Y_test=predict[["forward_count","comment_count","like_count"]]
pd.options.mode.use_inf_as_na = True
X_train.fillna(X_train.max(),inplace=True)
X_test.fillna(X_test.max(),inplace=True)
```

In [ ]:

```
lm=linear_model.LinearRegression()
model=lm.fit(X_train,Y_train)
pred=lm.predict(X_test)
pred=pred.round()
pred=(np.maximum(pred,0.))
np.savetxt("E:\\5th Sem\\DMA Project\\Model Evaluation\\weibo_predict_result2.csv",pred,delimiter=
',',header="forward_count,comment_count,like_count",comments="")
result=pd.read_csv("E:\\5th Sem\\DMA Project\\Model Evaluation\\weibo_predict_result2.csv")
train_real_pred = Y_test
forward=result['forward_count'].values
comment=result['forward_count'].values
like=result['forward_count'].values
train_real_pred['fp'],train_real_pred['cp'],train_real_pred['lp'] = forward,comment,like
print ("Score on the training set:{0:.2f}%".format(precision(train_real_pred.values)*100))
```

Model 2 (Media, Length, Emoji)

In [ ]:

```
X_train=train[["content_media_count","content_length","content_emoji_count"]]
Y_train=train[["forward_count","comment_count","like_count"]]
X_test=predict[["content_media_count","content_length","content_emoji_count"]]
Y_test=predict[["forward_count","comment_count","like_count"]]
pd.options.mode.use_inf_as_na = True
X_train.fillna(X_train.max(),inplace=True)
X_test.fillna(X_test.max(),inplace=True)
lm=linear_model.LinearRegression()
model=lm.fit(X_train,Y_train)
```

```
pred=lm.predict(X_test)
pred=pred.round()
pred=(np.maximum(pred,0.))
np.savetxt("E:\\5th Sem\\DMA Project\\Model Evaluation\\weibo_predict_result3.csv",pred,delimiter=
',',header="forward_count,comment_count,like_count",comments="")
result=pd.read_csv("E:\\5th Sem\\DMA Project\\Model Evaluation\\weibo_predict_result3.csv")
train_real_pred = Y_test
forward=result['forward_count'].values
comment=result['forward_count'].values
like=result['forward_count'].values
train_real_pred['fp'],train_real_pred['cp'],train_real_pred['lp'] = forward,comment,like
print ("Score on the training set:{0:.2f}%".format(precision(train_real_pred.values)*100))
```

Model 3(Media)

In [ ]:

```
X_train=train[["content_media_count"]]
Y_train=train[["forward_count","comment_count","like_count"]]
X_test=predict[["content_media_count"]]
Y_test=predict[["forward_count","comment_count","like_count"]]
print(X_train.shape,Y_train.shape)
print(X_test.shape,Y_test.shape)
pd.options.mode.use_inf_as_na = True
X_train.fillna(X_train.max(),inplace=True)
X_test.fillna(X_test.max(),inplace=True)
lm=linear_model.LinearRegression()
model=lm.fit(X_train,Y_train)
pred=lm.predict(X_test)
pred=pred.round()
pred=(np.maximum(pred,0.))
np.savetxt("E:\\5th Sem\\DMA Project\\Model Evaluation\\weibo_predict_result4.csv",pred,delimiter=
',',header="forward_count,comment_count,like_count",comments="")
result=pd.read_csv("E:\\5th Sem\\DMA Project\\Model Evaluation\\weibo_predict_result4.csv")
train_real_pred = Y_test
train_real_pred['fp']=result['forward_count'].values
train_real_pred['cp']=result['comment_count'].values
train_real_pred['lp']=result['like_count'].values
train_real_pred=train_real_pred.round()
print ("Score on the training set:{0:.2f}%".format(precision(train_real_pred.values)*100))
```

Model 4 (Time) Pre-requisite

In [ ]:

```
X_train=train[["hour","min","sec"]]
Y_train=train[["forward_count","comment_count","like_count"]]
X_test=predict[["hour","min","sec"]]
Y_test=predict[["forward_count","comment_count","like_count"]]
pd.options.mode.use_inf_as_na = True
X_train.fillna(X_train.max(),inplace=True)
X_test.fillna(X_test.max(),inplace=True)
lm=linear_model.LinearRegression()
model=lm.fit(X_train,Y_train)
pred=lm.predict(X_test)
pred=pred.round()
pred=(np.maximum(pred,0.))
np.savetxt("G://DMA_PROJECT//weibo_predict_result5.csv",pred,delimiter=',',header="forward_count,cc
mment_count,like_count",comments="")
result=pd.read_csv("G://DMA_PROJECT//weibo_predict_result5.csv")
train_real_pred = Y_test
train_real_pred['fp']=result['forward_count'].values
train_real_pred['cp']=result['comment_count'].values
train_real_pred['lp']=result['like_count'].values
train_real_pred=train_real_pred.round()
print ("Score on the training set:{0:.2f}%".format(precision(train_real_pred.values)*100))
```

Model 5 (Time: Hour)

In [ ]:

```
X_train=train[["hour"]]
Y_train=train[["forward_count","comment_count","like_count"]]
X_test=predict[["hour"]]
Y_test=predict[["forward_count","comment_count","like_count"]]
print(X_train.shape,Y_train.shape)
print(X_test.shape,Y_test.shape)
pd.options.mode.use_inf_as_na = True
X_train.fillna(X_train.max(),inplace=True)
X_test.fillna(X_test.max(),inplace=True)
lm=linear_model.LinearRegression()
model=lm.fit(X_train,Y_train)
pred=lm.predict(X_test)
pred=pred.round()
pred=(np.maximum(pred,0.))
np.savetxt("G://DMA_PROJECT//weibo_predict_result6.csv",pred,delimiter=',',header="forward_count,cc
mment_count,like_count",comments="")
result=pd.read_csv("G://DMA_PROJECT//weibo_predict_result6.csv")
train_real_pred = Y_test
train_real_pred['fp']=result['forward_count'].values
train_real_pred['cp']=result['comment_count'].values
train_real_pred['lp']=result['like_count'].values
train_real_pred=train_real_pred.round()
print ("Score on the training set:{0:.2f}%".format(precision(train_real_pred.values)*100))
```

Model 6 Time: (Hour, Min, Sec), Media, Length, Emoji

In [ ]:

```
X_train=train[["content_media_count","content_length","content_emoji_count","hour","min","sec"]]
Y_train=train[["forward_count","comment_count","like_count"]]
X_test=predict[["content_media_count","content_length","content_emoji_count","hour","min","sec"]]
Y_test=predict[["forward_count","comment_count","like_count"]]
pd.options.mode.use_inf_as_na = True
X_train.fillna(X_train.max(),inplace=True)
X_test.fillna(X_test.max(),inplace=True)
lm=linear_model.LinearRegression()
model=lm.fit(X_train,Y_train)
pred=lm.predict(X_test)
pred=pred.round()
pred=(np.maximum(pred,0.))
np.savetxt("G://DMA_PROJECT//weibo_predict_result7.csv",pred,delimiter=',',header="forward_count,cc
mment_count,like_count",comments="")
result=pd.read_csv("G://DMA_PROJECT//weibo_predict_result7.csv")
train_real_pred = Y_test
train_real_pred['fp']=result['forward_count'].values
train_real_pred['cp']=result['comment_count'].values
train_real_pred['lp']=result['like_count'].values
train_real_pred=train_real_pred.round()
print ("Score on the training set:{0:.2f}%".format(precision(train_real_pred.values)*100))
```

Model 7 Median,Time: (Hour, Min, Sec), Media, Length, Emoji

Only for Forward Count

In [ ]:

```
X_train=train[["content_media_count","content_length","content_emoji_count","hour","min","sec","for
ward_median"]]
Y_train=train[["forward_count"]]
X_test=predict[["content_media_count","content_length","content_emoji_count","hour","min","sec","fo
rward_median"]]
Y_test=predict[["forward_count"]]
```

In [ ]:

```
pd.options.mode.use_inf_as_na = True
X_train.fillna(X_train.max(),inplace=True)
```

```
X_test.fillna(X_test.max(),inplace=True)
```

In [ ]:

```
lm=linear_model.LinearRegression()
model=lm.fit(X_train,Y_train)
pred=lm.predict(X_test)
pred=pred.round()
pred=(np.maximum(pred,0.))
```

In [ ]:

```
np.savetxt("G://DMA_PROJECT//weibo_predict_result8.csv",pred,delimiter=',',header="forward_count,cc
mment_count,like_count",comments="")
result=pd.read_csv("G://DMA_PROJECT//weibo_predict_result8.csv")
```

In [ ]:

```
train_real_pred=Y_test
train_real_pred['fp']=result['forward_count'].values
train_real_pred=train_real_pred.round()
print ("Score on the training set:{0:.2f}%".format(precision2(train_real_pred.values)*100))
```

Modelling with Polarity

-10000 tuples

Contribution: Deepti

In [ ]:

```
dfpol=pd.read_csv("E:\DMA_PRE\polarity\weibo_polarity.csv")
dfpol['date']=pd.to_datetime(dfpol['date'],errors='coerce')
train_month=[g for n, g in dfpol.groupby(pd.Grouper(key='date',freq='M'))]

train_month[0]=pd.read_csv("E:\DMA_PRE\polarity\weibo_train_feb_cpts10000.csv")
train_month[1]=pd.read_csv("E:\DMA_PRE\polarity\weibo_train_march_cpts10000.csv")
train_month[2]=pd.read_csv("E:\DMA_PRE\polarity\weibo_train_april_cpts10000.csv")
train_month[3]=pd.read_csv("E:\DMA_PRE\polarity\weibo_train_may_cpts10000.csv")
train_month[4]=pd.read_csv("E:\DMA_PRE\polarity\weibo_train_june_cpts10000.csv")
train_month[5]=pd.read_csv("E:\DMA_PRE\polarity\weibo_train_july_cpts10000.csv")


frames1=[train_month[0],train_month[1],train_month[2],train_month[3],train_month[4]]
train=pd.concat(frames1)
predict=train_month[5]

## Model 7: (Factors: Media, Length, Emoji, Median,Polarity)

X_train1=train[["content_media_count","content_length","forward_median","comment_median","like_medi
an","polarity"]]
Y_train1=train[["forward_count","comment_count","like_count"]]
X_test1=predict[["content_media_count","content_length","forward_median","comment_median","like_med
ian","polarity"]]
Y_test1=predict[["forward_count","comment_count","like_count"]]

pd.options.mode.use_inf_as_na = True
X_train1.fillna(X_train1.max(),inplace=True)
X_test1.fillna(X_test1.max(),inplace=True)

lm1=linear_model.LinearRegression()
model1=lm1.fit(X_train1,Y_train1)
pred1=lm1.predict(X_test1)
pred1=pred1.round()
pred1=(np.maximum(pred1,0.))

print(model1.coef_)
print(model1.intercept_)

np.savetxt("E:\DMA_PRE\polarity\weibo_predict_result51.csv",pred1,delimiter=',',header="forward_cou
nt,comment_count,like_count",comments="")
```

```
result1=pd.read_csv("E:\DMA_PRE\polarity\weibo_predict_result51.csv")

print(mean_squared_error(Y_test1,result1))

train_real_pred=Y_test1
train_real_pred['fp']=result1['forward_count']
train_real_pred['cp']=result1['comment_count']
train_real_pred['lp']=result1['like_count']
print("Score:{0:.2f}%".format(precision(train_real_pred.values)*100))
```

Model (Factors: Media, Length, Emoji, Median,Polarity) with OLS

OLS is a type of linear least sqaures methods for estimating parameters in a linear regression model

In [ ]:

```
X_train1=train[["content_media_count","content_length","forward_median","comment_median","like_medi
an","polarity"]]
Y_train1=train[["forward_count","comment_count","like_count"]]
X_test1=predict[["content_media_count","content_length","forward_median","comment_median","like_med
ian","polarity"]]
Y_test1=predict[["forward_count","comment_count","like_count"]]

pd.options.mode.use_inf_as_na = True
X_train1.fillna(X_train1.max(),inplace=True)
X_test1.fillna(X_test1.max(),inplace=True)
model1=sm.OLS(Y_train1,X_train1).fit()
pred1=model1.predict(X_test1)
pred1=pred1.round()
pred1=(np.maximum(pred1,0.))
np.savetxt("C:/Users/user/Downloads/weibo_predict_result52.csv",pred1,delimiter=',',header="forward
_count,comment_count,like_count",comments="")
result1=pd.read_csv("C:/Users/user/Downloads/weibo_predict_result52.csv")

train_real_pred=Y_test1
train_real_pred['fp']=result1['forward_count']
train_real_pred['cp']=result1['comment_count']
train_real_pred['lp']=result1['like_count']
print("Score:{0:.2f}%".format(precision(train_real_pred.values)*100))
```

Model : (Factors: Media, Length, Emoji, Median,Polarity) with Ridge regression

Ridge regression is used to prevent multicollinearity among variables by shrinking the parameters

In [ ]:

```
X_train1=train[["content_media_count","content_length","forward_median","comment_median","like_medi
an","polarity"]]
Y_train1=train[["forward_count","comment_count","like_count"]]
X_test1=predict[["content_media_count","content_length","forward_median","comment_median","like_med
ian","polarity"]]
Y_test1=predict[["forward_count","comment_count","like_count"]]

pd.options.mode.use_inf_as_na = True
X_train1.fillna(X_train1.max(),inplace=True)
X_test1.fillna(X_test1.max(),inplace=True)
lm1=linear_model.Ridge(alpha=3)
model1=lm1.fit(X_train1,Y_train1)
pred1=lm1.predict(X_test1)
pred1=pred1.round()
pred1=(np.maximum(pred1,0.))
np.savetxt("C:/Users/user/Downloads/weibo_predict_result53.csv",pred1,delimiter=',',header="forward
_count,comment_count,like_count",comments="")
result1=pd.read_csv("C:/Users/user/Downloads/weibo_predict_result53.csv")
train_real_pred=Y_test1
```

```
train_real_pred['fp']=result1['forward_count']
train_real_pred['cp']=result1['comment_count']
train_real_pred['lp']=result1['like_count']
print("Score:{0:.2f}%".format(precision(train_real_pred.values)*100))
```

Model (Factors: Media, Length, Emoji, Median,Polarity) with Lasso regression

Lasso regression does automatic feature selection that means if some features are correlated then lasso will pick only one feature

In [ ]:

```
X_train1=train[["content_media_count","content_length","forward_median","comment_median","like_medi
an","polarity"]]
Y_train1=train[["forward_count","comment_count","like_count"]]
X_test1=predict[["content_media_count","content_length","forward_median","comment_median","like_med
ian","polarity"]]
Y_test1=predict[["forward_count","comment_count","like_count"]]

pd.options.mode.use_inf_as_na = True
X_train1.fillna(X_train1.max(),inplace=True)
X_test1.fillna(X_test1.max(),inplace=True)
lm1=Lasso(alpha=0.01)
model1=lm1.fit(X_train1,Y_train1)
pred1=lm1.predict(X_test1)
pred1=pred1.round()
pred1=(np.maximum(pred1,0.))
np.savetxt("C:/Users/user/Downloads/weibo_predict_result54.csv",pred1,delimiter=',',header="forward
_count,comment_count,like_count",comments="")
result1=pd.read_csv("C:/Users/user/Downloads/weibo_predict_result54.csv")
np.savetxt("C:/Users/user/Downloads/weibo_predict_result54.csv",pred1,delimiter=',',header="forward
_count,comment_count,like_count",comments="")
result1=pd.read_csv("C:/Users/user/Downloads/weibo_predict_result54.csv")
```

Modelling with Normalized Polarity

Contribution: Apoorva Malemath

In [ ]:

```
def normalize(df):
    result = df.copy()
    for feature_name in df.columns:

        result[feature_name] = (df[feature_name] - min_value) / (max_value - min_value)
    return result
```

In [ ]:

```
df1=pd.read_csv("E://DMA_PRED//polarityL1.csv")

max_value = df1['polarity'].max()
min_value = df1['polarity'].min()
df1['pnorm'] = (df1['polarity'] - min_value) / (max_value - min_value)
df=df1.drop(['Unnamed: 0'], axis=1)
df = sklearn.utils.shuffle(df)
uid_stat=pd.read_csv("E:\\DMA_PRE\\train_uid_stat.csv")
df = pd.merge(df,uid_stat , on=['u_id'])

df.columns

train=df[0:80000]
predict=df[80001:100000]

from sklearn.ensemble import RandomForestRegressor
features_train=train[['content_media_count','pnorm','forward_min', 'forward_max', 'forward_median'
, 'forward_mean',
       'comment_min', 'comment_max', 'comment_median', 'comment_mean',
```

```
       'like_min', 'like_max', 'like_median', 'like_mean']]
features_test=predict[['content_media_count','pnorm','forward_min', 'forward_max',
'forward_median', 'forward_mean',
       'comment_min', 'comment_max', 'comment_median', 'comment_mean',
       'like_min', 'like_max', 'like_median', 'like_mean']]
labels_train=train[['forward_count', 'comment_count', 'like_count']]
labels_test=predict[['forward_count', 'comment_count', 'like_count']]

x = features_train
y = labels_train
x1 = features_test
y1 = labels_test

regr = RandomForestRegressor(max_depth=50, random_state=0,n_estimators=100)
regr.fit(x, y)
pred2 = regr.predict(x1)
temp = pd.DataFrame.from_records(pred2)
temp=temp.round()
temp=(np.maximum(temp,0))
temp=temp.abs()
temp=temp.astype(int)

train_real_pred=y1
train_real_pred['fp']=temp[0].values
train_real_pred['cp']=temp[1].values
train_real_pred['lp']=temp[2].values
print("Score:{0:.2f}%".format(precision(train_real_pred.values)*100))

lm=linear_model.LinearRegression()
model=lm.fit(x,y)
pred1=lm.predict(x1)

temp = pd.DataFrame.from_records(pred1)
temp=temp.round()
temp=(np.maximum(temp,0))
train_real_pred=y1
train_real_pred['fp']=temp[0].values
train_real_pred['cp']=temp[1].values
train_real_pred['lp']=temp[2].values
print("Score:{0:.2f}%".format(precision(train_real_pred.values)*100))
```

Ensemble - Averaging

Contribution: Apoorva Malemath

In [ ]:

```
df1=pd.read_csv("E:\\5th-Sem\\DMA Project\\Project\\weibo_train1_cpts.csv")
df2=pd.read_csv("E:\\5th-Sem\\DMA Project\\Project\\weibo_train2_cpts.csv")
frames=[df1,df2]
train_all=pd.concat(frames)
train=train_all[0:983694]
predict=train_all[983695:1229618]
train.columns
```

Model 1 - Linear Regression

In [ ]:

```
X_train=train[["content_media_count","content_length","content_emoji_count","hour","min","sec","for
ward_median","comment_median","like_median"]]
Y_train=train[["forward_count","comment_count","like_count"]]
X_test=predict[["content_media_count","content_length","content_emoji_count","hour","min","sec","fo
rward_median","comment_median","like_median"]]
Y_test=predict[["forward_count","comment_count","like_count"]]
print(X_train.shape,Y_train.shape)
print(X_test.shape,Y_test.shape)

pd.options.mode.use_inf_as_na = True
X_train.fillna(X_train.max(),inplace=True)
X_test.fillna(X_test.max(),inplace=True)
```

```
lm=linear_model.LinearRegression()
model=lm.fit(X_train,Y_train)
pred1=lm.predict(X_test)
temp = pd.DataFrame.from_records(pred1)
temp=temp.round()
temp=(np.maximum(temp,0))
train_real_pred=Y_test
train_real_pred['fp']=temp[0].values
train_real_pred['cp']=temp[1].values
train_real_pred['lp']=temp[2].values
print("Score:{0:.2f}%".format(precision(train_real_pred.values)*100))
```

Model 2 - Random Forest

In [ ]:

```
## Spliting of training dataset into 70% training data and 30% testing data randomly
features_train=train[["content_media_count","content_#_count","content_length","content_emoji_count
","forward_median","comment_median","like_median"]]
features_test=predict[["content_media_count","content_#_count","content_length","content_emoji_cour
t","forward_median","comment_median","like_median"]]
labels_train=train[['forward_count', 'comment_count', 'like_count']]
labels_test=predict[['forward_count', 'comment_count', 'like_count']]

x = features_train
y = labels_train
x1 = features_test
y1 = labels_test



regr = RandomForestRegressor(max_depth=50, random_state=0,n_estimators=100)
regr.fit(x, y)
pred2 = regr.predict(x1)
temp = pd.DataFrame.from_records(pred2)
temp=temp.round()
temp=(np.maximum(temp,0))
temp=temp.abs()
temp=temp.astype(int)
train_real_pred=Y_test
train_real_pred['fp']=temp[0].values
train_real_pred['cp']=temp[1].values
train_real_pred['lp']=temp[2].values
print("Score:{0:.2f}%".format(precision(train_real_pred.values)*100))
```

Model 3 - OLS

In [ ]:

```
model3=sm.OLS(Y_train,X_train).fit()
pred3=model3.predict(X_test)
```

Model 4- Ridge

In [ ]:

```
lm1=linear_model.Ridge(alpha=3)
model4=lm1.fit(X_train,Y_train)
pred4=lm1.predict(X_test)
```

Model 5- Lasso

In [ ]:

```
lm1=Lasso(alpha=0.01)
model5=lm1.fit(X_train,Y_train)
```

```
pred5=lm1.predict(X_test)
```

Ensemble - Averaging

In [ ]:

```
pred=(pred1+pred2+pred3+pred4+pred5)/5
pred=pred.round()
pred=(np.maximum(pred,0))
pred=pred.abs()
pred1=pred.astype(int)
train_real_pred=Y_test
train_real_pred['fp']=pred1[0]
train_real_pred['cp']=pred1[1]
train_real_pred['lp']=pred1[2]
print("Score:{0:.2f}%".format(precision(train_real_pred.values)*100))
```

Mapping UID

Coontribution: Apoorva Malemath

In [ ]:

```
unique_id=train_all['u_id'].unique().tolist()
uid_df = pd.DataFrame({'u_id':unique_id})
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
le.fit(unique_id)
```

In [ ]:

```
l=[]
l=le.transform(unique_id)
df = pd.DataFrame({'u_id':l})
uid_df['id']=df['u_id']
```

In [ ]:

```
train_all=train_all.set_index('u_id').join(uid_df.set_index('u_id'))
train=train_all[0:983694]
predict=train_all[983695:1229618]
```

In [ ]:

```
X_train=train[["id","content_media_count","content_length","content_emoji_count","hour","min","sec"
,"forward_median","comment_median","like_median"]]
Y_train=train[["forward_count","comment_count","like_count"]]
X_test=predict[["id","content_media_count","content_length","content_emoji_count","hour","min","sec
","fo
                rward_median","comment_median","like_median"]]
Y_test=predict[["forward_count","comment_count","like_count"]]
print(X_train.shape,Y_train.shape)
print(X_test.shape,Y_test.shape)

pd.options.mode.use_inf_as_na = True
X_train.fillna(X_train.max(),inplace=True)
X_test.fillna(X_test.max(),inplace=True)

lm=linear_model.LinearRegression()
model=lm.fit(X_train,Y_train)
pred1=lm.predict(X_test)
temp = pd.DataFrame.from_records(pred1)
temp=temp.round()
temp=(np.maximum(temp,0))
temp=temp.abs()
temp=temp.astype(int)
train_real_pred=Y_test
train_real_pred['fp']=temp[0].values
train_real_pred['cp']=temp[1].values
```

```
train_real_pred['lp']=temp[2].values
print("Score:{0:.2f}%".format(precision(train_real_pred.values)*100))
```

In [ ]:

```
## Spliting of training dataset into 70% training data and 30% testing data randomly
features_train=train[["id","content_media_count","content_#_count","content_length","content_emoji_
count","forward_median","comment_median","like_median"]]
features_test=predict[["id","content_media_count","content_#_count","content_length","content_emoji
_count","forward_median","comment_median","like_median"]]
labels_train=train[['forward_count', 'comment_count', 'like_count']]
labels_test=predict[['forward_count', 'comment_count', 'like_count']]

x = features_train
y = labels_train
x1 = features_test
y1 = labels_test



regr = RandomForestRegressor(max_depth=50, random_state=0,n_estimators=100)
regr.fit(x, y)
pred2 = regr.predict(x1)
temp = pd.DataFrame.from_records(pred2)
temp=temp.round()
temp=(np.maximum(temp,0))
temp=temp.abs()
temp=temp.astype(int)
train_real_pred=Y_test
train_real_pred['fp']=temp[0].values
train_real_pred['cp']=temp[1].values
train_real_pred['lp']=temp[2].values
print("Score:{0:.2f}%".format(precision(train_real_pred.values)*100))
```

Best F C L Match Model

Contribution: Ashish Kar

In [ ]:

```
import _pickle as cPickle
import import_ipynb
import pandas as pd
import numpy as np
from genUidStat import loadData,genUidStat
from evaluation import precision
from runTime import runTime
from pathos.pools import _ProcessPool
from multiprocess.pool import Pool

df1=pd.read_csv("weibo_train1.csv")
df2=pd.read_csv("weibo_train2.csv")
frames=[df1,df2]
traindata=pd.concat(frames)

def splitDataFrameIntoSmaller(df, chunkSize = 10000):
    listOfDf = list()
    numberChunks = len(df) // chunkSize + 1
    for i in range(numberChunks):
        listOfDf.append(df[i*chunkSize:(i+1)*chunkSize])
    return listOfDf

uid_stat=pd.read_csv("train_uid_stat.csv")

uid = splitDataFrameIntoSmaller(uid_stat, chunkSize = 500)

uid[0].shape[0]

# Generation of Best Scoring F C L for each UID

def search_all_uid(stat_dic,file):
 import pandas as pd
 import numpy as np
```

```
 def _deviation(predict, real, kind):
  t = 5.0 if kind=='f' else 3.0
  return abs(predict - real) / (real + t)
 def _precision_i(fp, fr, cp, cr, lp, lr):
  return 1 - 0.5 * _deviation(fp, fr, 'f') - 0.25 * _deviation(cp, cr, 'c') - 0.25 * _deviation(lp,
lr, 'l')
 def _sgn(x):
  return 1 if x>0 else 0
 def _count_i(fr, cr, lr):
  x = fr + cr + lr
  return 101 if x>100 else (x+1)
 def precision(real_and_predict):
  numerator,denominator = 0.0,0.0
  for  fr, cr, lr,fp, cp, lp in real_and_predict:
   numerator += _count_i(fr, cr, lr) * _sgn(_precision_i(fp, fr, cp, cr, lp, lr) - 0.8)
   denominator += _count_i(fr, cr, lr)
  return (numerator / denominator)
 def score(uid_data,pred):
  """
  uid_data:
   pd.DataFrame
  pred:
   list, [fp,cp,lp]
  """
  uid_real_pred = uid_data[['forward_count','comment_count','like_count']]
  uid_real_pred['fp'] = pred[0]
  uid_real_pred['cp'] = pred[1]
  uid_real_pred['lp'] = pred[2]
  return precision(uid_real_pred.values)
 def search(uid_data,target,args):
  args = list(args)
  target_index = ['forward_count','comment_count','like_count'].index(target)
  target_min,target_median,target_max = args[3*target_index:3*target_index+3]
  del args[3*target_index:3*target_index+3]
  pred = (args[1],args[4])

  best_num = [target_median]
  best_pred = list(pred)
  best_pred.insert(target_index,target_median)
  best_score = score(uid_data,best_pred)
  for num in range(target_min,target_max+1):
   this_pred = list(pred)
   this_pred.insert(target_index,num)
   this_score = score(uid_data,this_pred)
   if this_score >= best_score:
    if this_score > best_score:
     best_num = [num]
     best_score = this_score
    else:
     best_num.append(num)

  return best_num[np.array([abs(i - target_median) for i in best_num]).argmin()]
 uid_best_pred = {}
 pool = _ProcessPool()
 uids,f,c,l = [],[],[],[]
 m=1
 for uid in stat_dic:
  print ("search uid:{}".format(uid),m)
  m=m+1
  uid_data = traindata[traindata.u_id == uid]
  arguments = stat_dic[uid][['forward_min','forward_median','forward_max','comment_min',\
     'comment_median','comment_max','like_min','like_median','like_max']]
  arguments = tuple([int(i) for i in arguments])
  f.append(pool.apply_async(search,args=(uid_data,'forward_count',arguments)))
  c.append(pool.apply_async(search,args=(uid_data,'comment_count',arguments)))
  l.append(pool.apply_async(search,args=(uid_data,'like_count',arguments)))
  uids.append(uid)
 pool.close()
 pool.join()
 f = [i.get() for i in f]
 c = [i.get() for i in c]
 l = [i.get() for i in l]
 for i in range(len(uids)):
  uid_best_pred[uids[i]] = [f[i],c[i],l[i]]
 #cPickle.dump(uid_best_pred,open('uid_best_pred'+str(file)+'.pkl','ab'))
 label = ['forward_count','comment_count','like_count']
 pd.DataFrame.from_dict(data=uid_best_pred,orient='index').to_csv("G:\\Anconda_Prog\\BestPred\\wei
```

```
bo_uidbest"+str(file)+".csv",header=label)
 print("Written to file")
```

In [ ]:

```
uid_stat=pd.read_csv("train_uid_stat.csv")
uid_stat=uid_stat.set_index('u_id')
uid = splitDataFrameIntoSmaller(uid_stat, chunkSize = 100)
n=368
while n<373:
    df=uid[n].T
    stat=df.to_dict('series')
    n=n+1
    search_all_uid(stat,n)
```

In [ ]:

```
i=0
for i in range(0,373):
 st[i]=pd.read_csv("weibo_uidbest"+str(i)+".csv")
for i in range(0,373):
 string=string+"st["+str(i)+"],"
string=string[:-1]
frames=[string]
uid_best_pred=pd.concat(frames)
uid_best_pred.rename(columns={uid_best_pred[0]:"u_id"})
uid_best_pred.to_csv("train_best_pred.csv",sep=',',index=False,encoding='utf-8')

# FILES GENERATED..... NOW RESULT GENERATION

@runTime
def predict_by_search(submission=True):
 traindata,testdata = loadData()
 ub=pd.read_csv("train_best_pred.csv")
 ub=ub.set_index('u_id')
 df=ub.T
 uid_best_pred=df.to_dict('series')
 #uid_best_pred = search_all_uid()
 #print ("search done,now predict on traindata and testdata...")

 #predict traindata with uid's best fp,cp,lp
 forward,comment,like = [],[],[]
 for uid in traindata['u_id']:
  if uid in uid_best_pred:
   forward.append(int(uid_best_pred[uid][0]))
   comment.append(int(uid_best_pred[uid][1]))
   like.append(int(uid_best_pred[uid][2]))
  else:
   forward.append(0)
   comment.append(0)
   like.append(0)
 #score on the traindata
 train_real_pred = traindata[['forward_count','comment_count','like_count']]
 train_real_pred['fp'],train_real_pred['cp'],train_real_pred['lp'] = forward,comment,like
 print ("Score on the training set:{0:.2f}%".format(precision(train_real_pred.values)*100))
 if submission:
  test_pred = testdata[['u_id','m_id']]
  forward,comment,like = [],[],[]
  for uid in testdata['u_id']:
   if uid in uid_best_pred:
    forward.append(int(uid_best_pred[uid][0]))
    comment.append(int(uid_best_pred[uid][1]))
    like.append(int(uid_best_pred[uid][2]))
   else:
    forward.append(0)
    comment.append(0)
    like.append(0)
  test_pred['fp'],test_pred['cp'],test_pred['lp'] = forward,comment,like

  #generate submission file
  result = []
  filename = "weibo_predict_search.txt"
  for _,row in test_pred.iterrows():
   result.append("{0}\t{1}\t{2},{3},{4}\n".format(row[0],row[1],row[2],row[3],row[4]))
  f = open(filename,'w')
```

```
  f.writelines(result)
  f.close()
  print ('generate submission file "{}"'.format(filename))

predict_by_search()
```

----------Performance Analysis and Lederboard Ranking----------

Leaderboard Ranked Model Analysis

Contribution: All

In [ ]:

```
import pandas as pd
import numpy as np
from sklearn import linear_model
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import mean_squared_error
from matplotlib import pyplot as plt
import statsmodels.api as sm

# Best Model Evaluation

## Model 1: UID FCL Median Model
## Train Score: 32.73 Official Score:29.38989214 Rank:241

df3=pd.read_csv("weibo_train1_cpts.csv")
df4=pd.read_csv("weibo_train2_cpts.csv")
frames=[df3,df4]
train_dataset=pd.concat(frames)

print(mean_squared_error(train_dataset['forward_count'],train_dataset['forward_median']))
print(mean_squared_error(train_dataset['comment_count'],train_dataset['comment_median']))
print(mean_squared_error(train_dataset['like_count'],train_dataset['like_median']))

print(accuracy_score(train_dataset['forward_count'],train_dataset['forward_median']))
print(accuracy_score(train_dataset['comment_count'],train_dataset['comment_median']))
print(accuracy_score(train_dataset['like_count'],train_dataset['like_median']))

## Model 2: UID FCL Best Match Model
## Train Score: 34.76 Official Score:29.67695154 (29.68) Rank:218

df1=pd.read_csv("weibo_train1.csv")
df2=pd.read_csv("weibo_train2.csv")
frames=[df1,df2]
train_dataset=pd.concat(frames)

stat=pd.read_csv("train_best_pred.csv")

trainstat=pd.merge(train_dataset,stat,on=['u_id'],how='left')

trainstat.shape

trainstat.head(614809).to_csv("weibo_train1_final.csv",sep=',',index=False,encoding='utf-8')
trainstat.tail(614809).to_csv("weibo_train2_final.csv",sep=',',index=False,encoding='utf-8')

df3=pd.read_csv("weibo_train1_final.csv")
df4=pd.read_csv("weibo_train2_final.csv")
frames=[df3,df4]
train_dataset=pd.concat(frames)


print(mean_squared_error(train_dataset['forward_count_x'],train_dataset['forward_count_y']))
print(mean_squared_error(train_dataset['comment_count_x'],train_dataset['comment_count_y']))
print(mean_squared_error(train_dataset['like_count_x'],train_dataset['like_count_y']))

print(accuracy_score(train_dataset['forward_count_x'],train_dataset['forward_count_y']))
print(accuracy_score(train_dataset['comment_count_x'],train_dataset['comment_count_y']))
print(accuracy_score(train_dataset['like_count_x'],train_dataset['like_count_y']))
```

```
print(accuracy_score(train_dataset['like_count_x'],train_dataset['like_count_y']))

## Summary

![](fcomp.png)
```

----------Conclusions----------

We built models to predict the number of forwards, comments and likes for a weibo. We followed an iterative KDD process to refine the pre-processing of data and improve the performance of model. As the crucial part of our data is text, we apply text preprocessing techniques namely removal of noise, stop words, stemming and lemmatization.

We found that using the algorithmic approach where we defined the best possible value for each user based the UID yielded good result. The model used statistical factors and yielded a score of 29.82% and got a rank of 218 on leaderboard.

Also computing polarity of the text and performing a min max normalization on it yielded a score of 78.92% on the cross validation.