Configuration Manual

Classification of Online Patient Reviews Based on Effectiveness Using Machine Learning Algorithms

1 Introduction

This document is details how the project is implemented step by step. This project aims to achieve three objectives. One is to extract useful and critical information from online drug reviews by applying natural language technique. The second is to critically assess and classify drugs based on effectiveness from online drug reviews submitted by patients in online public forums.Third one is to find the correlation between sentiment and effectiveness of the reviews. To achieve this result different software and machine learning techniques are used, the details of following are specified below.

2 PC Requirements

This section details the minimum hardware requirements needed for this project. It also gives details about the different software applications that are needed to implement and run the project.

2.1 Hardware Requirements

- OS: Windows 7/10
- Processor: Intel Core i3 4340/AMD FX-6300
- Memory: 8GB
- HDD Space: 50GB
- GPU: Intel HD Graphics 500

This project was run on Windows 10 OS, with 8 GB RAM, 2TB of HDD space, AMD graphics card and Intel i7 8th Gen PC.

2.2 Software Requirements

2.2.1 Software needed to be installed

- Anaconda Navigator – Version 1.9.6. Copyright 2016 Anaconda, Inc. This software must have Spyder IDE installed in order to run the python code.
- Microsoft Office 2003
- Python – Version 3.6. (compatible to both x86 and x64 processors)

2.2.2 Packages needed to install and run the code

- Pandas – for data manipulation and analysis
- Numpy – for array related operations
- Matplotlib – for plotting graphs

- Seaborn – for plotting data
- Itertools – for plotting
- Nltk – for NLP related operations
- sklearn (upon installation of sklearn it must include the following - model_selection, svm, ensemble, feature extraction, linear model, metrics, neighbors) – for machine learning algorithms.
- imblearn (upon installation of imblearn it must include the following - metrics, over_sampling) – for handling imbalance class data
- textblob – for finding the sentiment of a review
- xgboost – for xgboost classifier
- scrapy[1] - for web crawler installation

How to install a package :

- To install a package first open Anaconda Prompt.
- Run the command – pip install <package name>
  E.g.: pip install sklearn

3 Data Sources

The data for implementation will be scraped from online public forums. For this research study, two public forums have been taken into consideration namely Drugs.com and WebMD.com.

The links for the same are as follows:

1) Drugs.com[2]

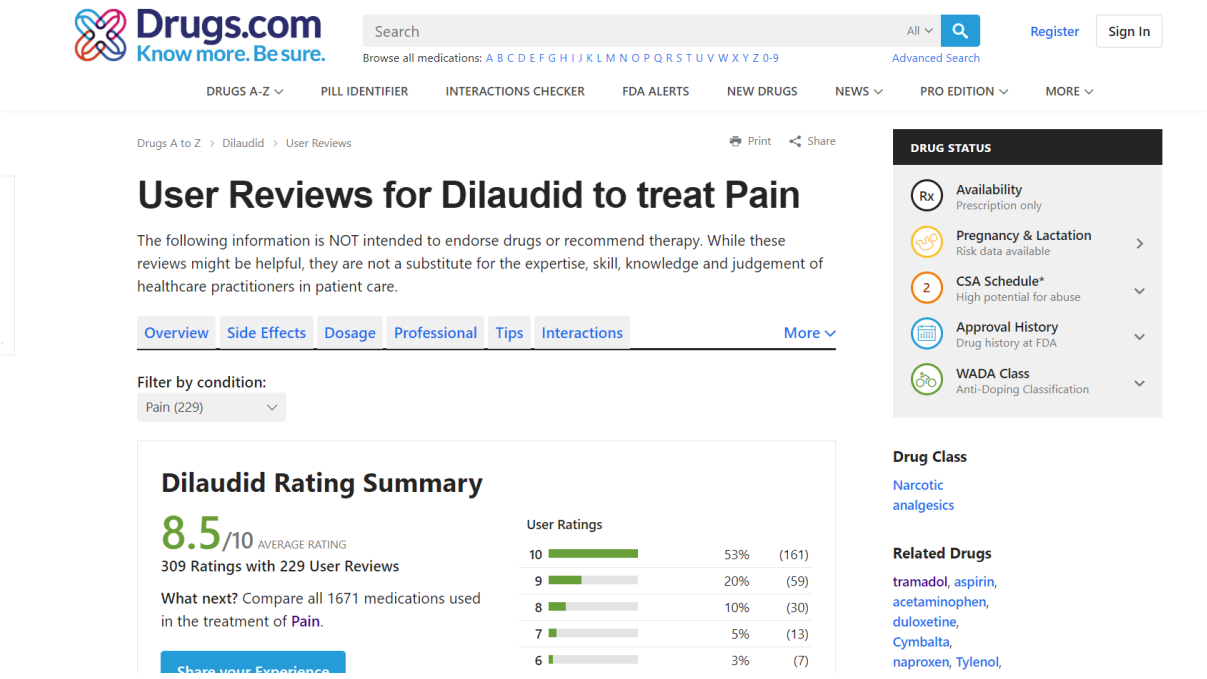
Various drugs have been scraped from Drugs.com, one such link is - https://www.drugs.com/comments/hydromorphone/dilaudid-for-pain.html

https://www.drugs.com/comments/hydromorphone/dilaudid-for-pain.html
https://www.drugs.com/comments/tramadol/for-pain.html
https://www.drugs.com/comments/acetaminophen-hydrocodone/for-pain.html
https://www.drugs.com/comments/oxycodone/for-pain.html
https://www.drugs.com/comments/tapentadol/for-pain.html
https://www.drugs.com/comments/gabapentin/for-pain.html
https://www.drugs.com/comments/ketorolac/for-pain.html
https://www.drugs.com/comments/hydromorphone/for-pain.html
https://www.drugs.com/comments/tapentadol/nucynta-for-pain.html
https://www.drugs.com/comments/diclofenac-topical/for-pain.html
https://www.drugs.com/comments/acetaminophen-oxycodone/for-pain.html
https://www.drugs.com/comments/amitriptyline/for-pain.html
https://www.drugs.com/comments/amitriptyline/for-pain.html
https://www.drugs.com/comments/oxycodone/oxycontin-for-pain.html
https://www.drugs.com/comments/acetaminophen-oxycodone/percocet-for-pain.html
https://www.drugs.com/comments/ketorolac/toradol-for-pain.html

[1] https://scrapy.org/download/
[2] https://www.drugs.com/

https://www.drugs.com/comments/fentanyl/for-pain.html
https://www.drugs.com/comments/acetaminophen-hydrocodone/norco-for-pain.html
https://www.drugs.com/comments/morphine/for-pain.html
https://www.drugs.com/comments/hydrocodone/for-pain.html
https://www.drugs.com/comments/methadone/for-pain.html
https://www.drugs.com/comments/diclofenac/for-pain.html
https://www.drugs.com/comments/acetaminophen-codeine/for-pain.html
https://www.drugs.com/comments/diclofenac-topical/voltaren-gel-for-pain.html
https://www.drugs.com/comments/pregabalin/for-pain.html
https://www.drugs.com/comments/lidocaine-topical/for-pain.html
https://www.drugs.com/comments/tramadol/ultram-for-pain.html
https://www.drugs.com/comments/buprenorphine/for-pain.html
https://www.drugs.com/comments/acetaminophen-hydrocodone/vicodin-for-pain.html
https://www.drugs.com/comments/acetaminophen-codeine/tylenol-with-codeine-3-for-pain.html
https://www.drugs.com/comments/nortriptyline/for-pain.html

Screenshot of website:



2) WebMD.com[3]

Similarly various drugs have been scraped from this website, one such link is - https://www.webmd.com/drugs/drugreview-9130-dilaudid.aspx?drugid=9130&drugname=dilaudid

List of drugs that have been taken from WebMD.com for text processing:

[3] https://www.webmd.com/

- https://www.webmd.com/drugs/drugreview-9130-dilaudid.aspx?drugid=9130&drugname=dilaudid
- https://www.webmd.com/drugs/drugreview-15915-Vicodin-ES-oral.aspx?drugid=15915&drugname=Vicodin-ES-oral
- https://www.webmd.com/drugs/drugreview-7277-percocet.aspx?drugid=7277&drugname=percocet
- https://www.webmd.com/drugs/drugreview-2798-OxyContin+oral.aspx?drugid=2798&drugname=OxyContin+oral&sortby=3
- https://www.webmd.com/drugs/drugreview-2671-methadone+oral.aspx?drugid=2671&drugname=methadone+oral&sortby=3
- https://www.webmd.com/drugs/drugreview-4398-tramadol-hcl-er.aspx?drugid=4398&drugname=tramadol-hcl-er
- https://www.webmd.com/drugs/drugreview-10710-nortriptyline-hcl.aspx?drugid=10710&drugname=nortriptyline-hcl
- https://www.webmd.com/drugs/drugreview-63-Norco-oral.aspx?drugid=63&drugname=Norco-oral
- https://www.webmd.com/drugs/drugreview-5173-naproxen-oral.aspx?drugid=5173&drugname=naproxen-oral
- https://www.webmd.com/drugs/drugreview-11276-Ultram-oral.aspx?drugid=11276&drugname=Ultram-oral
- https://www.webmd.com/drugs/drugreview-7292-Lortab-oral.aspx?drugid=7292&drugname=Lortab-oral
- https://www.webmd.com/drugs/drugreview-152563-Nucynta-oral.aspx?drugid=152563&drugname=Nucynta-oral
- https://www.webmd.com/drugs/drugreview-3499-roxicodone.aspx?drugid=3499&drugname=roxicodone
- https://www.webmd.com/drugs/drugreview-214-etodolac.aspx?drugid=214&drugname=etodolac
- https://www.webmd.com/drugs/drugreview-15178-endocet.aspx?drugid=15178&drugname=endocet
- https://www.webmd.com/drugs/drugreview-1098-aleve.aspx?drugid=1098&drugname=aleve
- https://www.webmd.com/drugs/drugreview-21825-ultracet.aspx?drugid=21825&drugname=ultracet
- https://www.webmd.com/drugs/drugreview-362-acetaminophen.aspx?drugid=362&drugname=acetaminophen
- https://www.webmd.com/drugs/drugreview-11255-nubain-solution.aspx?drugid=11255&drugname=nubain-solution
- https://www.webmd.com/drugs/drugreview-16080-roxicet-solution.aspx?drugid=16080&drugname=roxicet-solution
- https://www.webmd.com/drugs/drugreview-9845-neurontin.aspx?drugid=9845&drugname=neurontin
- https://www.webmd.com/drugs/drugreview-251-hydrocodone-acetaminophen.aspx?drugid=251&drugname=hydrocodone-acetaminophen
- https://www.webmd.com/drugs/drugreview-5166-ibuprofen-tablet-chewable.aspx?drugid=5166&drugname=ibuprofen-tablet-chewable
- https://www.webmd.com/drugs/drugreview-327-morphine-sulfate-er-capsule-multiphase-24-hr.aspx?drugid=327&drugname=morphine-sulfate-er-capsule-multiphase-24-hr

Screenshot of website :

4 Installation and Scrapy (web-crawler) setup

4.1 Creating a Project

- Command to create Project:

  - scrapy startproject Thesis

- New Scrapy project ‘Thesis’, using template directory '//anaconda/lib/python2.7/site-packages/scrapy/templates/project', created in:
  /Development/PetProjects/ScrapyCrawlers/olx

- Command to start first spider with:
  - cd Thesis (this is the path where the project has been created)
  - scrapy genspider example example.com (Here example is the Name of the spider and example.com is the website on which the spider will be created)

- The spider would be generated inside a folder named Spider. Traverse to the folder spider and open the file containing the name of the spider, in this case example.py

- Now make the required changes in the file to scrape the data from the website.

4.2 Code for scraping data from Drugs.com

```
# -*- coding: utf-8 -*-
#import scrapy
```

```
from scrapy.spiders import  CrawlSpider, Rule
from scrapy.linkextractors import LinkExtractor


class DrugreviewsSpider(CrawlSpider):
    name = 'drugreviews'
    allowed_domains = ['drugs.com']
    start_urls = ['https://www.drugs.com/comments/hydromorphone/dilaudid-for-pain.html',
                  'https://www.drugs.com/comments/tramadol/for-pain.html',
                  'https://www.drugs.com/comments/acetaminophen-hydrocodone/for-pain.html',
                  'https://www.drugs.com/comments/oxycodone/for-pain.html',
                  'https://www.drugs.com/comments/tapentadol/for-pain.html',
                  'https://www.drugs.com/comments/gabapentin/for-pain.html',
                  'https://www.drugs.com/comments/ketorolac/for-pain.html',
                  'https://www.drugs.com/comments/hydromorphone/for-pain.html',
                  'https://www.drugs.com/comments/tapentadol/nucynta-for-pain.html',
                  'https://www.drugs.com/comments/diclofenac-topical/for-pain.html',
                  'https://www.drugs.com/comments/acetaminophen-oxycodone/for-pain.html',
                  'https://www.drugs.com/comments/amitriptyline/for-pain.html',
                  'https://www.drugs.com/comments/amitriptyline/for-pain.html',
                  'https://www.drugs.com/comments/oxycodone/oxycontin-for-pain.html',
                  'https://www.drugs.com/comments/acetaminophen-oxycodone/percocet-for-pain.html',
                  'drugs.com/comments/ketorolac/toradol-for-pain.html',
                  'https://www.drugs.com/comments/fentanyl/for-pain.html',
                  'https://www.drugs.com/comments/acetaminophen-hydrocodone/norco-for-pain.html',
                  'https://www.drugs.com/comments/morphine/for-pain.html',
                  'https://www.drugs.com/comments/hydrocodone/for-pain.html',
                  'https://www.drugs.com/comments/methadone/for-pain.html',
                  'https://www.drugs.com/comments/diclofenac/for-pain.html',
                  'https://www.drugs.com/comments/acetaminophen-codeine/for-pain.html',
                  'https://www.drugs.com/comments/diclofenac-topical/voltaren-gel-for-pain.html',
                  'https://www.drugs.com/comments/pregabalin/for-pain.html',
                  'https://www.drugs.com/comments/lidocaine-topical/for-pain.html',
                  'https://www.drugs.com/comments/tramadol/ultram-for-pain.html',
                  'https://www.drugs.com/comments/buprenorphine/for-pain.html',
                  'https://www.drugs.com/comments/acetaminophen-hydrocodone/vicodin-for-
pain.html',
                  'https://www.drugs.com/comments/acetaminophen-codeine/tylenol-with-codeine-3-for-
pain.html',
                  'https://www.drugs.com/comments/nortriptyline/for-pain.html',
                  ]

    rules = (
        Rule(LinkExtractor(allow=(), restrict_css=('.noVisit')),
             callback="parse_item",
             follow=True),)

    first_response = True

    def parse_item(self, response):
```

```
        print('Processing url: '+response.url)

        review_comment = []
        review_rating = []
        drug_name =
(response.xpath('//*[@id="content"]/div[2]/div[2]/h2/text()').extract())[0].split()[0]
        for i in range(0,len(response.css('#content > div.contentBox > div > div.ddc-comment-
rating.ddc-comment-section > div.comment-rating-score > div::text').extract())):
            rating = response.css('#content > div.contentBox > div > div.ddc-comment-rating.ddc-
comment-section > div.comment-rating-score > div::text').extract()[i]
            comment = response.css('#content > div.contentBox > div > p.ddc-comment-content >
span::text').extract()[i]
            #drug = response.css('#content > div.contentBox > h2::text').extract()[0].split()[2]
            rating = rating.replace("\"","")
            rating = int(rating)/2
            #drug_name.append(drug)
            review_rating.append(rating)
            review_comment.append(comment)

        row_data=zip(review_comment, review_rating)

        print(row_data)

        #Making extracted data row wise
        for item in row_data:
            #create a dictionary to store the scraped info
            scraped_info = {
                #key:value, #item[0] means product in the list and so on, index tells what value to assign
                'review_comment' : item[0],
                'effectiveness_rating' : item[1],
                'drug_name' : drug_name
            }

            yield scraped_info
```

4.3 Code for scraping data from WebMD.com

```
# -*- coding: utf-8 -*-
#import scrapy
from scrapy.spiders import  CrawlSpider, Rule
from scrapy.linkextractors import LinkExtractor


class DrugreviewsSpider(CrawlSpider):
    name = 'drugreviews_webmd'
    allowed_domains = ['webmd.com']
    start_urls = ['https://www.webmd.com/drugs/drugreview-9130-
dilaudid.aspx?drugid=9130&drugname=dilaudid',
                  'https://www.webmd.com/drugs/drugreview-15915-Vicodin-ES-
oral.aspx?drugid=15915&drugname=Vicodin-ES-oral',
```

```
            'https://www.webmd.com/drugs/drugreview-7277-
percocet.aspx?drugid=7277&drugname=percocet',
            'https://www.webmd.com/drugs/drugreview-2798-
OxyContin+oral.aspx?drugid=2798&drugname=OxyContin+oral&sortby=3',
            'https://www.webmd.com/drugs/drugreview-2671-
methadone+oral.aspx?drugid=2671&drugname=methadone+oral&sortby=3',
            'https://www.webmd.com/drugs/drugreview-4398-tramadol-hcl-
er.aspx?drugid=4398&drugname=tramadol-hcl-er',
            'https://www.webmd.com/drugs/drugreview-10710-nortriptyline-
hcl.aspx?drugid=10710&drugname=nortriptyline-hcl',
            'https://www.webmd.com/drugs/drugreview-63-Norco-
oral.aspx?drugid=63&drugname=Norco-oral',
            'https://www.webmd.com/drugs/drugreview-5173-naproxen-
oral.aspx?drugid=5173&drugname=naproxen-oral',
            'https://www.webmd.com/drugs/drugreview-11276-Ultram-
oral.aspx?drugid=11276&drugname=Ultram-oral',
            'https://www.webmd.com/drugs/drugreview-7292-Lortab-
oral.aspx?drugid=7292&drugname=Lortab-oral',
            'https://www.webmd.com/drugs/drugreview-152563-Nucynta-
oral.aspx?drugid=152563&drugname=Nucynta-oral',
            'https://www.webmd.com/drugs/drugreview-3499-
roxicodone.aspx?drugid=3499&drugname=roxicodone',
            'https://www.webmd.com/drugs/drugreview-214-
etodolac.aspx?drugid=214&drugname=etodolac',
            'https://www.webmd.com/drugs/drugreview-15178-
endocet.aspx?drugid=15178&drugname=endocet',
            'https://www.webmd.com/drugs/drugreview-1098-
aleve.aspx?drugid=1098&drugname=aleve',
            'https://www.webmd.com/drugs/drugreview-21825-
ultracet.aspx?drugid=21825&drugname=ultracet',
            'https://www.webmd.com/drugs/drugreview-362-
acetaminophen.aspx?drugid=362&drugname=acetaminophen',
            'https://www.webmd.com/drugs/drugreview-11255-nubain-
solution.aspx?drugid=11255&drugname=nubain-solution',
            'https://www.webmd.com/drugs/drugreview-16080-roxicet-
solution.aspx?drugid=16080&drugname=roxicet-solution',
            'https://www.webmd.com/drugs/drugreview-9845-
neurontin.aspx?drugid=9845&drugname=neurontin',
            'https://www.webmd.com/drugs/drugreview-251-hydrocodone-
acetaminophen.aspx?drugid=251&drugname=hydrocodone-acetaminophen',
            'https://www.webmd.com/drugs/drugreview-5166-ibuprofen-tablet-
chewable.aspx?drugid=5166&drugname=ibuprofen-tablet-chewable',
            'https://www.webmd.com/drugs/drugreview-327-morphine-sulfate-er-capsule-
multiphase-24-hr.aspx?drugid=327&drugname=morphine-sulfate-er-capsule-multiphase-24-hr',
            ]

    rules = (
        Rule(LinkExtractor(allow=(),
restrict_xpaths=("//*[@id=\"ratings_fmt\"]/div[3]/div[2]/a[contains(text(),'Next')]",)),
             callback="parse_item",
```

```
            follow=True),)

    first_response = True

    def parse_item(self, response):
        print('Processing url: '+response.url)

        review_comment = []
        review_rating = response.xpath('//*[@id=\"ctnStars\"]/div[1]/p[2]/span/text()').re(r"Rating:
(\d+)")[1:]
        for i in range(1,6):
            review = response.xpath('//*[@id=\"comFull'+str(i)+'\"]/text()').extract()
            review_comment.append(review)
            drug = str(response.xpath('//*[@id="header"]/div/h1/text()').extract()).split()[-2]

        row_data=zip(review_comment, review_rating)
        print(row_data)

        #Making extracted data row wise
        for item in row_data:
            #create a dictionary to store the scraped info
            scraped_info = {
                #key:value, #item[0] means product in the list and so on, index tells what value to assign
                'review_comment' : item[0],
                'effectiveness_rating' : item[1],
                'drug_name' : drug
            }

            yield scraped_info

    def parse(self, response):
        if self.first_response == True:
            # use it or pass it to some other function
            print('Processing url: '+response.url)

        review_comment = []
        review_rating = response.xpath('//*[@id=\"ctnStars\"]/div[1]/p[2]/span/text()').re(r"Rating:
(\d+)")[1:]
        for i in range(1,6):
            review = response.xpath('//*[@id=\"comFull'+str(i)+'\"]/text()').extract()
            review_comment.append(review)

        drug = str(response.xpath('//*[@id="header"]/div/h1/text()').extract()).split()[-2]

        row_data=zip(review_comment, review_rating)

        print(row_data)

        #Making extracted data row wise
        for item in row_data:
```

```
        #create a dictionary to store the scraped info
        scraped_info = {
            #key:value, #item[0] means product in the list and so on, index tells what value to assign
            'review_comment' : item[0],
            'effectiveness_rating' : item[1],
            'drug_name' : drug
        }

        yield scraped_info
    self.first_response = False

    # Pass the response to crawlspider
    for r in super(DrugreviewsSpider, self).parse(response):
        yield r
```

4.4 Command to crawl the data from website

- scrapy crawl <spider name>

E.g. scrapy crawl example

5 Implementation

For the implementation KDD process has been followed, various stages has to be performed in order to arrive at the results. The initial step of extracting the data from website has been detailed in section 4 of this document. The next step in the process is data cleaning, data transformation, data mining models and results.

Important Point : Please make sure to store and read the file from the correct path. Three different files are used, one csv file for all the review, one csv file for reviews from webmd.com and one csv file for reviews from drugs.com. Csv files containing reviews from drugs.com and webmd.com is used for case study 2 and csv file with all the reviews is used for case study 1.

The code for the following is as below:

Importing Libraries necessary for implementation:

```
import pandas as pd
import numpy as np
import re
import string
import matplotlib.pyplot as plt
import seaborn as sns
import itertools
import sys
from sklearn import preprocessing
from nltk.corpus import stopwords
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from textblob import TextBlob
from xgboost import XGBClassifier
from imblearn.over_sampling import SMOTE
```

Pre-processing code:

Pre-processing code consists of grouping the reviews into two groups namely ‘not completely effective’ and ‘very effective’, expanding the contracted words, removal of numbers, lower casing

the reviews, replacing the punctuations with null characters. The reviews with length less than 10 words are also removed from final dataset.

```
'''print option for printing the complete array'''
np.set_printoptions(threshold=sys.maxsize)

ratings = pd.read_csv("D:\Thesis\drugs\drugreviews_asasd.csv")

ratings = ratings.dropna()
ratings = ratings.reset_index(drop=True)


'''manually classifying the data based on the rating'''
ratings.loc[ratings['effectiveness_rating'] <= 3, 'effectiveness'] = 'not completely effective'
ratings.loc[ratings['effectiveness_rating'] > 3, 'effectiveness'] = 'very effective'

'''ploting the data based on different options'''
sns.countplot(x='effectiveness', data=ratings)

'''function to replace decontracted words with actual words,
   remove numbers and unknown characters from the string'''
def expandwords_del_numbers(phrase):
    # general contractions
    phrase = re.sub(r"\â\?\?", "'", phrase)
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    phrase = re.sub(r"\r", " ", phrase)
    phrase = re.sub(r"\n", " ", phrase)
    phrase = re.sub(r"[0-9]", "", phrase)
    return phrase

for i in range(0,len(ratings)):
    phrase = str(ratings.loc[i,'review_comment']).lower()
    ratings.loc[i,'review_comment'] = expandwords_del_numbers(phrase)

'''replace punctuation in the string with null character'''
table = str.maketrans({key: None for key in string.punctuation})
for i in range(0,len(ratings)):
    ratings.loc[i,'review_comment'] = ratings.loc[i,'review_comment'].translate(table)

'''lower case all the letters in the string'''
for i in range(0,len(ratings)):
    print(ratings.iloc[i]['review_comment'].lower())

'''saving to a file for further investigation'''
ratings.to_csv("D:\Thesis\drugs\effectiveness.csv", index=False)

'''reading from the data from saved file'''
ratings_parsed = pd.read_csv("D:\Thesis\drugs\effectiveness.csv")

'''remove reviews that are shorter'''
for i in range(0, len(ratings_parsed)):
    length = len(str(ratings_parsed.loc[i,'review_comment']).split())
    if length < 10:
        ratings_parsed.drop(i, axis=0, inplace=True)

'''reset index of the data frame'''
ratings_parsed = ratings_parsed.reset_index(drop=True)
```

Code for obtaining metrics information:
Code to obtain metrics namely accuracy, precision, recall, f1-score, geometric mean and also plotting the confusion matrix.

```
'''function to plot the confusion matrix'''
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.winter):
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title, fontsize=30)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, fontsize=10)
    plt.yticks(tick_marks, classes, fontsize=10)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.

    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt), horizontalalignment="center",
                 color="black" if cm[i, j] < thresh else "white", fontsize=10)

    plt.tight_layout()
    plt.ylabel('True label', fontsize=10)
    plt.xlabel('Predicted label', fontsize=10)

    return plt


'''function to get metric scores'''
def get_metrics(y_test, y_predicted):
    # true positives / (true positives+false positives)
    precision = precision_score(y_test, y_predicted, pos_label=None,
                                average='weighted')
    # true positives / (true positives + false negatives)
    recall = recall_score(y_test, y_predicted, pos_label=None,
                          average='weighted')

    # harmonic mean of precision and recall
    f1 = f1_score(y_test, y_predicted, pos_label=None,
                  average='weighted')

    # true positives + true negatives/ total
    accuracy = accuracy_score(y_test, y_predicted)

    return accuracy, precision, recall, f1
```

Code for SVM:

Code for SVM algorithm with respective parameters set.

```
'''function for SVM algorithm'''
def SVM_algorithm(X,Y):
    X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.30,
                                                        random_state = 40)

    svclassifier = SVC(kernel='linear', C=1, gamma=10, degree=1,
                       random_state=40)
    svclassifier.fit(list(X_train), y_train)

    y_pred = svclassifier.predict(list(X_test))

    cm = confusion_matrix(y_test, y_pred)
    fig = plt.figure(figsize=(10, 10))
    plot = plot_confusion_matrix(cm, classes=['not completely effective','very effective'],
                                 normalize=False, title='Confusion matrix')
    plt.show()

    accuracy, precision, recall, f1 = get_metrics(y_test, y_pred)
    print("accuracy = %.3f, precision = %.3f, "
          "recall = %.3f, f1 = %.3f" %
          (accuracy*100, precision*100, recall*100, f1*100))
```

Code for Random Forest Grid Search:

```
'''Grid Search for Random Forest'''
def Rf_GridSearch():
    # Number of trees in random forest
    n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
    # Number of features to consider at every split
    max_features = ['auto', 'sqrt']
    # Maximum number of levels in tree
    max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
    max_depth.append(None)
    # Minimum number of samples required to split a node
    min_samples_split = [2, 5, 10]
    # Minimum number of samples required at each leaf node
    min_samples_leaf = [1, 2, 4]
    # Method of selecting samples for training each tree
    bootstrap = [True, False]
    # Create the random grid
    random_grid = {'n_estimators': n_estimators,
                   'max_features': max_features,
                   'max_depth': max_depth,
                   'min_samples_split': min_samples_split,
                   'min_samples_leaf': min_samples_leaf,
                   'bootstrap': bootstrap}

    rf = RandomForestClassifier()
    # Random search of parameters, using 3 fold cross validation,
    # search across 100 different combinations, and use all available cores
    rf_random = RandomizedSearchCV(estimator = rf, param_distributions = random_grid,
                                   n_iter = 100, cv = 3, verbose=2, random_state=42,
                                   n_jobs = -1)
    ngram_vectorizer = CountVectorizer(max_features = 1500, binary=True,
                                       ngram_range=(1, 2),
                                       max_df = 0.5, min_df = 1,
                                       stop_words=stopwords.words('english'))
    ngram_vectorizer.fit(ratings_parsed['review_comment'])
    X_bigram = ngram_vectorizer.transform(ratings_parsed['review_comment']).toarray()
    Y = ratings_parsed['effectiveness']
    X_train, y_train = smt.fit_resample(X_bigram, Y)
    # Fit the random search model
    rf_random.fit(X_train, y_train)

    return rf_random.best_params_
```

Code for Random Forest:

Code for Random Forest algorithm with respective parameters set from grid search.

```
'''function for RandomForest algorithm'''
def Rf_algorithm(X, Y):

    parameters = Rf_GridSearch()

    clf=RandomForestClassifier(n_estimators=parameters['n_estimators'],
                               min_samples_split = parameters['min_samples_split'],
                               min_samples_leaf = parameters['min_samples_leaf'],
                               max_depth = parameters['min_samples_leaf'],
                               max_features = parameters['max_features'],
                               random_state = 40,
                               bootstrap = parameters['bootstrap'])

    X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.40,
                                                        random_state = 40)

    clf.fit(list(X_train),y_train)

    y_pred=clf.predict(list(X_test))

    cm = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(10, 10))
    plot_confusion_matrix(cm, classes=['not completely effective',
                                       'very effective'],
                          normalize=False, title='Confusion matrix')
    plt.show()

    accuracy, precision, recall, f1 = get_metrics(y_test, y_pred)
    print("accuracy = %.3f, precision = %.3f, "
          "recall = %.3f, f1 = %.3f" %
          (accuracy*100, precision*100, recall*100, f1*100))
```

Code for KNN grid search:

```
'''Grid Search for KNN'''
def GridSearch_knn():
    parameters = {'n_neighbors':[3,4,5,6,7,11,19],
              'leaf_size':[1,3,5],
              'algorithm':['auto', 'kd_tree'],
              'n_jobs':[-1],
              'metric' : ['euclidean','manhattan']}
    gs = GridSearchCV(estimator=KNeighborsClassifier(),
                      param_grid = parameters,
                      verbose = 1,
                      cv = 5,
                      n_jobs = -1)

    gs_res = gs.fit(X_train, y_train)

    return gs_res.best_params_
```

Code for KNN:

Code for KNN algorithm with respective parameters set from grid search.

```
'''function for KNN algorithm'''
def kNN_algorithm(X_para, Y):

    gs_res = GridSearch_knn()

    model = KNeighborsClassifier(algorithm=gs_res['algorithm'],
                                 leaf_size=gs_res['leaf_size'],
                                 metric=gs_res['metric'],
                                 n_jobs=gs_res['n_jobs'],
                                 n_neighbors=gs_res['n_neighbors'])

    encoder = preprocessing.LabelEncoder()
    Y = encoder.fit_transform(Y)

    X_train, X_test, y_train, y_test = train_test_split(X_para, Y, test_size = 0.30)

    model.fit(list(X_train), y_train)

    y_pred = model.predict(list(X_test))

    cm = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(10, 10))
    plot_confusion_matrix(cm, classes=['not completely effective',
                                       'very effective'],
                          normalize=False, title='Confusion matrix')
    plt.show()

    accuracy, precision, recall, f1 = get_metrics(y_test, y_pred)
    print("accuracy = %.3f, precision = %.3f, "
          "recall = %.3f, f1 = %.3f" %
          (accuracy*100, precision*100, recall*100, f1*100))
```

Code for Logistic Regression Grid Search:

```
'''Log Reg Grid Search'''
def LogReg_GS():

    clf = LogisticRegression()

    '''performing grid search'''
    grid_values = {'C':[0.001,.009,0.01,.09,1,5,10,25],
                   'solver' : ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga']}
    grid_clf_acc = GridSearchCV(clf, param_grid = grid_values)
    grid_clf_acc.fit(X_train, y_train)

    return grid_clf_acc.best_params_
```

Code for Logistic Regression:

Code for Logistic Regression algorithm with respective parameters set from grid search.

```
'''function for Logistic Regression algorithm'''
def LogReg(X, Y):
    X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3,
                                                        random_state=40)

    best_param = LogReg_GS()

    clf = LogisticRegression(C=best_param['C'],
                             penalty = 'l2',
                             class_weight='balanced', solver=best_param['solver'],
                             multi_class='multinomial', random_state=40)

    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)

    cm = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(10, 10))
    plot_confusion_matrix(cm, classes=['not completely effective','very effective'],
                                  normalize=False, title='Confusion matrix')
    plt.show()

    accuracy, precision, recall, f1 = get_metrics(y_test, y_pred)
    print("accuracy = %.3f, precision = %.3f, "
          "recall = %.3f, f1 = %.3f" %
          (accuracy*100, precision*100, recall*100, f1*100))
```

Code for XGBoost Classifier:

Code for XGBoost algorithm with respective parameters set.

```
def XGB(X, Y):
    X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.30)

    classifier = XGBClassifier(learning_rate =0.1,
                               n_estimators=1000,
                               max_depth=5,
                               min_child_weight=1,
                               gamma=0,
                               subsample=0.8,
                               colsample_bytree=0.8,
                               objective= 'binary:logistic',
                               nthread=4,
                               scale_pos_weight=1,
                               seed=27,
                               random_state = 40)
    classifier.fit(X_train,y_train)
    y_pred=classifier.predict(list(X_test))

    cm = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(10, 10))
    plot_confusion_matrix(cm, classes=['not completely effective','very effective'],
                                normalize=False, title='Confusion matrix')
    plt.show()

    accuracy, precision, recall, f1 = get_metrics(y_test, y_pred)
    print("accuracy = %.3f, precision = %.3f, "
          "recall = %.3f, f1 = %.3f" %
          (accuracy*100, precision*100, recall*100, f1*100))
```

Code for Oversampling using SMOTE:

```
'''OverSampling Technique'''
smt = SMOTE(random_state = 40)
```

Case 1 – In-domain Data Analysis:

Analysis is done using the entire dataset with feature set obtained from all the reviews from both the websites. The code for the same is as shown below.

Code:

```
'''Uni-gram model'''
ngram_vectorizer = CountVectorizer(max_features = 2500, binary=True,
                                   ngram_range=(1,1),
                                   max_df = 0.5, min_df = 1,
                                   stop_words=stopwords.words('english'))
ngram_vectorizer.fit(ratings_parsed['review_comment'])
X_Unigram = ngram_vectorizer.transform(ratings_parsed['review_comment']).toarray()
Y = ratings_parsed['effectiveness']
X_train, y_train = smt.fit_resample(X_Unigram, Y)
Rf_algorithm(X_train, y_train)
LogReg(X_train, y_train)
SVM_algorithm(X_train, y_train)
kNN_algorithm(X_train, y_train)
XGB(X_train, y_train)

'''Bi-gram model'''
ngram_vectorizer = CountVectorizer(max_features = 2500, binary=True,
                                   ngram_range=(1, 2),
                                   max_df = 0.5, min_df = 1,
                                   stop_words=stopwords.words('english'))
ngram_vectorizer.fit(ratings_parsed['review_comment'])
X_Bigram = ngram_vectorizer.transform(ratings_parsed['review_comment']).toarray()
Y = (ratings_parsed['effectiveness'])
X_train, y_train = smt.fit_resample(X_Bigram, Y)
Rf_algorithm(X_train, y_train)
LogReg(X_train, y_train)
SVM_algorithm(X_train, y_train)
kNN_algorithm(X_train, y_train)
XGB(X_train, y_train)

'''tf-idf Unigram-model'''
tfidfconverter = TfidfVectorizer(max_features = 2500, max_df=.5, min_df=1,
                                 ngram_range=(1,1),
                                 stop_words=stopwords.words('english'),
                                 use_idf=True, norm=None)
X_tfidf = tfidfconverter.fit_transform(ratings_parsed['review_comment']).toarray()
Y = ratings_parsed['effectiveness']
X_train, y_train = smt.fit_sample(X_tfidf, Y)
Rf_algorithm(X_train, y_train)
LogReg(X_train, y_train)
SVM_algorithm(X_train, y_train)
kNN_algorithm(X_train, y_train)
XGB(X_train, y_train)

'''tf-idf Bigram-model'''
tfidfconverter = TfidfVectorizer(max_features = 2500, max_df=.5, min_df=1,
                                 ngram_range=(1,2),
                                 stop_words=stopwords.words('english'),
                                 use_idf=True, norm=None)
X_tfidf = tfidfconverter.fit_transform(ratings_parsed['review_comment']).toarray()
Y = ratings_parsed['effectiveness']
X_train, y_train = smt.fit_sample(X_tfidf, Y)
Rf_algorithm(X_train, y_train)
LogReg(X_train, y_train)
SVM_algorithm(X_train, y_train)
kNN_algorithm(X_train, y_train)
XGB(X_train, y_train)
```

Case 2 – Cross-domain Data Analysis

Here the analysis is done across websites. Firstly the feature set is obtained from one website and it is used to form the vector representation of reviews from the other website. The code for this as shown below.

Code:

```
'''Uni-gram model'''
ngram_vectorizer = CountVectorizer(max_features = 2500, binary=True,
                                   ngram_range=(1,1),
                                   max_df = 0.5, min_df = 1,
                                   stop_words=stopwords.words('english'))
ngram_vectorizer.fit(ratings_drugs_parsed['review_comment'])
X_Unigram = ngram_vectorizer.transform(ratings_webmd_parsed['review_comment']).toarray()
Y = ratings_webmd_parsed['effectiveness']
X_train, y_train = smt.fit_resample(X_Unigram, Y)
Rf_algorithm(X_train, y_train)
LogReg(X_train, y_train)
SVM_algorithm(X_train, y_train)
kNN_algorithm(X_train, y_train)
XGB(X_train, y_train)

ngram_vectorizer = CountVectorizer(max_features = 2500, binary=True,
                                   ngram_range=(1,1),
                                   max_df = 0.5, min_df = 1,
                                   stop_words=stopwords.words('english'))
ngram_vectorizer.fit(ratings_webmd_parsed['review_comment'])

X_Unigram = ngram_vectorizer.transform(ratings_drugs_parsed['review_comment']).toarray()
Y = ratings_drugs_parsed['effectiveness']
X_train, y_train = smt.fit_resample(X_Unigram, Y)
Rf_algorithm(X_train, y_train)
LogReg(X_train, y_train)
SVM_algorithm(X_train, y_train)
kNN_algorithm(X_train, y_train)
XGB(X_train, y_train)

'''Bi-gram model'''
ngram_vectorizer_bigram_drugs = CountVectorizer(max_features = 2500, binary=True,
                                   ngram_range=(1, 2),
                                   max_df = 0.5, min_df = 1,
                                   stop_words=stopwords.words('english'))
ngram_vectorizer_bigram_drugs.fit(ratings_drugs_parsed['review_comment'])
X_Bigram = ngram_vectorizer_bigram_drugs.transform(ratings_webmd_parsed['review_comment']).toarray()
Y = (ratings_webmd_parsed['effectiveness'])
X_train, y_train = smt.fit_resample(X_Bigram, Y)
Rf_algorithm(X_train, y_train)
LogReg(X_train, y_train)
SVM_algorithm(X_train, y_train)
kNN_algorithm(X_train, y_train)
XGB(X_train, y_train)

ngram_vectorizer_bigram_webmd = CountVectorizer(max_features = 2500, binary=True,
                                   ngram_range=(1,2),
                                   max_df = 0.5, min_df = 1,
                                   stop_words=stopwords.words('english'))
ngram_vectorizer_bigram_webmd.fit(ratings_webmd_parsed['review_comment'])
X_Unigram = ngram_vectorizer_bigram_webmd.transform(ratings_drugs_parsed['review_comment']).toarray()
Y = ratings_drugs_parsed['effectiveness']
X_train, y_train = smt.fit_resample(X_Unigram, Y)
Rf_algorithm(X_train, y_train)
LogReg(X_train, y_train)
SVM_algorithm(X_train, y_train)
kNN_algorithm(X_train, y_train)
XGB(X_train, y_train)
```

```
'''tf-idf Unigram-model'''
tfidfconverter_drugs_uni = TfidfVectorizer(max_features = 2500, max_df=.5, min_df=1,
                                   ngram_range=(1,1),
                                   stop_words=stopwords.words('english'),
                                   use_idf=True, norm=None)
tfidfconverter_drugs_uni.fit(ratings_drugs_parsed['review_comment'])
X_tfidf = tfidfconverter_drugs_uni.transform(ratings_webmd_parsed['review_comment']).toarray()
Y = ratings_webmd_parsed['effectiveness']
X_train, y_train = smt.fit_sample(X_tfidf, Y)
Rf_algorithm(X_train, y_train)
LogReg(X_train, y_train)
SVM_algorithm(X_train, y_train)
kNN_algorithm(X_train, y_train)
XGB(X_train, y_train)

tfidfconverter_webmd_uni = TfidfVectorizer(max_features = 2500, max_df=.5, min_df=1,
                                   ngram_range=(1,1),
                                   stop_words=stopwords.words('english'),
                                   use_idf=True, norm=None)
tfidfconverter_webmd_uni.fit(ratings_webmd_parsed['review_comment'])
X_tfidf = tfidfconverter_webmd_uni.transform(ratings_drugs_parsed['review_comment']).toarray()
Y = ratings_drugs_parsed['effectiveness']
X_train, y_train = smt.fit_sample(X_tfidf, Y)
Rf_algorithm(X_train, y_train)
LogReg(X_train, y_train)
SVM_algorithm(X_train, y_train)
kNN_algorithm(X_train, y_train)
XGB(X_train, y_train)

'''tf-idf Bigram-model'''
tfidfconverter_drugs_bi = TfidfVectorizer(max_features = 2500, max_df=.5, min_df=1,
                                   ngram_range=(1,2),
                                   stop_words=stopwords.words('english'),
                                   use_idf=True, norm=None)
tfidfconverter_drugs_bi.fit(ratings_drugs_parsed['review_comment'])
X_tfidf = tfidfconverter_drugs_bi.transform(ratings_webmd_parsed['review_comment']).toarray()
Y = ratings_webmd_parsed['effectiveness']
X_train, y_train = smt.fit_sample(X_tfidf, Y)
Rf_algorithm(X_train, y_train)
LogReg(X_train, y_train)
SVM_algorithm(X_train, y_train)
kNN_algorithm(X_train, y_train)
XGB(X_train, y_train)

tfidfconverter_webmd_bi = TfidfVectorizer(max_features = 2500, max_df=.5, min_df=1,
                                   ngram_range=(1,2),
                                   stop_words=stopwords.words('english'),
                                   use_idf=True, norm=None)
tfidfconverter_webmd_bi.fit(ratings_webmd_parsed['review_comment'])
X_tfidf = tfidfconverter_webmd_bi.transform(ratings_drugs_parsed['review_comment']).toarray()
Y = ratings_drugs_parsed['effectiveness']
X_train, y_train = smt.fit_sample(X_tfidf, Y)
Rf_algorithm(X_train, y_train)
LogReg(X_train, y_train)
SVM_algorithm(X_train, y_train)
kNN_algorithm(X_train, y_train)
XGB(X_train, y_train)
```

Case 3 – Correlation between Sentiment and Effectiveness of Review

Here the correlation between the sentiment and effectiveness of the review is calculated using Pearson correlation coefficient. The code for this is as shown below, it includes distribution of reviews across sentiment and effectiveness and the code to obtain the Pearson correlation.

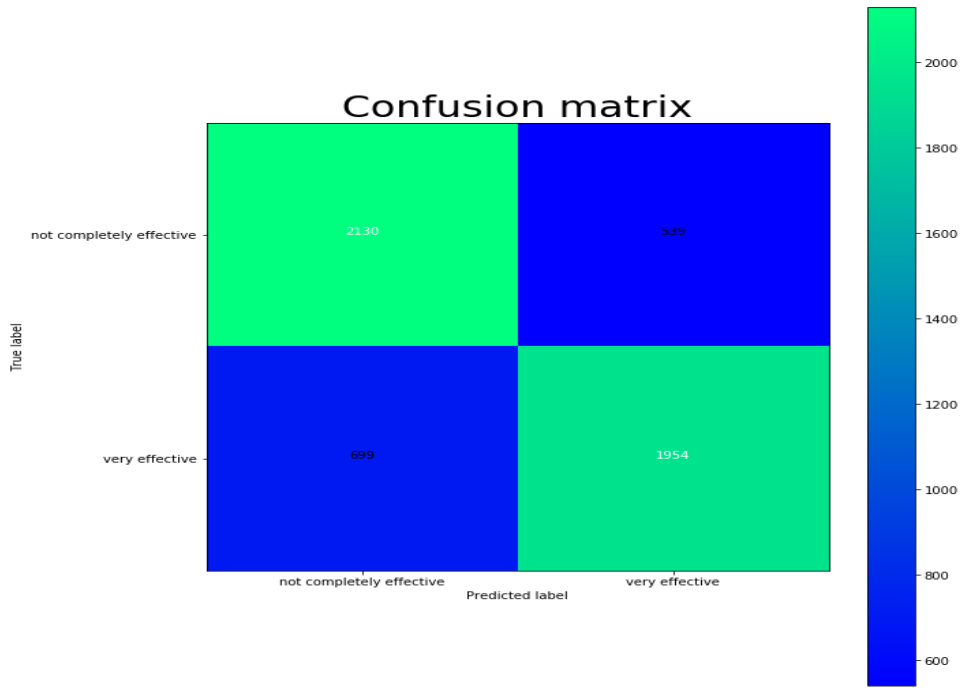
Code:

```
'''labeling the sentiment of each review using TextBlob'''
for i in range(0,len(ratings_parsed['review_comment'])):
    score = TextBlob(ratings_parsed.loc[i,'review_comment'])
    if (float(score.sentiment[0]) < 0):
        ratings_parsed.loc[i,'sentiment'] = 'negative'
    else:
        ratings_parsed.loc[i,'sentiment'] = 'positive'

'''to find correlation between sentiment and effectiveness'''
encoder = preprocessing.LabelEncoder()
sentiment_encoded = encoder.fit_transform(ratings_parsed['sentiment'])
sentiment_encoded = sentiment_encoded.reshape(-1, 1)
effectiveness_encoded = encoder.fit_transform(ratings_parsed['effectiveness'])
effectiveness_encoded = effectiveness_encoded.reshape(-1, 1)

from scipy.stats import pearsonr
# calculate Pearson's correlation
corr, _ = pearsonr(sentiment_encoded, effectiveness_encoded)
print('Pearsons correlation: %.3f' % corr)

sns.violinplot(y='effectiveness_rating', x='sentiment', data=ratings_parsed)
sns.violinplot(y='effectiveness_rating', x='effectiveness', data=ratings_parsed)
```

The code provided above has to be saved to a python file. The python file can be run using the Spyder IDE provided by the Anaconda Framework.

The results are obtained as shown in the figure below for every algorithm that is run.

```
In [2]: XGB(X_train, y_train)
```




```
accuracy = 76.738, precision = 76.831, recall = 76.738, f1 = 76.715
```