**SOURCE CODE**

from tkinter import messagebox

from tkinter import \*

from tkinter import simpledialog

import tkinter

from tkinter import filedialog

import matplotlib.pyplot as plt

from tkinter.filedialog import askopenfilename

from CustomButton import TkinterCustomButton

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import precision\_score

from sklearn.metrics import recall\_score

from sklearn.metrics import f1\_score

from sklearn.metrics import accuracy\_score

import pandas as pd

import numpy as np

import pickle

import os

from string import punctuation

from nltk.corpus import stopwords

import nltk

from nltk.stem import WordNetLemmatizer

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.svm import LinearSVC

from sklearn.linear\_model import LogisticRegression

from sklearn.naive\_bayes import MultinomialNB

from sklearn.neural\_network import MLPClassifier

from sklearn.linear\_model import SGDClassifier

from sklearn.linear\_model import RidgeClassifier

from numpy import dot

from numpy.linalg import norm

main = Tk()

main.title("Integrated ML with NLP framework for Drug Recommendation System")

main.geometry("1300x1200")

global filename

global dataset

global X, Y

global X\_train, X\_test, y\_train, y\_test

accuracy = []

precision = []

recall = []

fscore = []

stop\_words = set(stopwords.words('english'))

lemmatizer = WordNetLemmatizer()

global drug\_name, condition, review, rating

global tfidf\_vectorizer

global classifier

def cleanPost(doc):

tokens = doc.split()

table = str.maketrans('', '', punctuation)

tokens = [w.translate(table) for w in tokens]

tokens = [word for word in tokens if word.isalpha()]

tokens = [w for w in tokens if not w in stop\_words]

tokens = [word for word in tokens if len(word) > 1]

tokens = [lemmatizer.lemmatize(token) for token in tokens]

tokens = ' '.join(tokens)

return tokens

def uploadDataset():

global filename

global dataset

text.delete('1.0', END)

filename = askopenfilename(initialdir = "Dataset")

tf1.insert(END,str(filename))

text.insert(END,"Dataset Loaded\n\n")

dataset = pd.read\_csv(filename,sep="\t",nrows=5000)

text.insert(END,str(dataset.head()))

label = dataset.groupby('rating').size()

label.plot(kind="bar")

plt.title("Ratings Graph")

plt.show()

def preprocessDataset():

global X, Y

global X\_train, X\_test, y\_train, y\_test

global drug\_name, condition, review, rating

global dataset

text.delete('1.0', END)

if os.path.exists('model/data.npy'):

data = np.load("model/data.npy")

drug\_name = data[0]

condition = data[1]

review = data[2]

rating = data[3]

else:

for i in range(len(dataset)):

dname = dataset.get\_value(i,"drugName")

cond = dataset.get\_value(i,"condition")

reviewText = dataset.get\_value(i,"review")

ratings = dataset.get\_value(i,"rating")

reviewText = str(reviewText)

reviewText = reviewText.strip().lower()

reviewText = cleanPost(reviewText)

drug\_name.append(dname)

condition.append(cond)

review.append(reviewText)

rating.append(ratings-1)

print(i)

data = [drug\_name,condition,review,rating]

data = np.asarray(data)

np.save("model/data",data)

text.insert(END,"Reviews after cleaning and preprocessing\n\n")

text.insert(END,str(review))

label = dataset.groupby('drugName').size().head(20)

label.plot(kind="bar")

plt.title("Top 20 Drug Name Graph")

plt.show()

def TFIDFExtraction():

global drug\_name, condition, review, rating

global tfidf\_vectorizer

text.delete('1.0', END)

global X, Y

global X\_train, X\_test, y\_train, y\_test

tfidf\_vectorizer = TfidfVectorizer(stop\_words=stop\_words, use\_idf=True, smooth\_idf=False, norm=None, decode\_error='replace', max\_features=700)

tfidf = tfidf\_vectorizer.fit\_transform(review).toarray()

df = pd.DataFrame(tfidf, columns=tfidf\_vectorizer.get\_feature\_names())

text.insert(END,str(df)+"\n\n")

text.insert(END,str(df.values[0]))

df = df.values

X = df[:, 0:df.shape[1]]

Y = rating

indices = np.arange(X.shape[0])

np.random.shuffle(indices)

X = X[indices]

Y = Y[indices]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=0.2)

def test(cls,name):

predict = cls.predict(X\_test)

acc = accuracy\_score(y\_test,predict)\*100

p = precision\_score(y\_test,predict,average='macro') \* 100

r = recall\_score(y\_test,predict,average='macro') \* 100

f = f1\_score(y\_test,predict,average='macro') \* 100

text.insert(END,name+" Precision : "+str(p)+"\n")

text.insert(END,name+" Recall : "+str(r)+"\n")

text.insert(END,name+" F1-Score : "+str(f)+"\n")

text.insert(END,name+" Accuracy : "+str(acc)+"\n\n")

precision.append(p)

accuracy.append(acc)

recall.append(r)

fscore.append(f)

def TrainML():

global X, Y

global X\_train, X\_test, y\_train, y\_test

global classifier

text.delete('1.0', END)

if os.path.exists('model/lr.txt'):

with open('model/lr.txt', 'rb') as file:

lr\_cls = pickle.load(file)

file.close()

test(lr\_cls,"Logistic Regression")

else:

lr\_cls = LogisticRegression(max\_iter=500)

lr\_cls.fit(X,Y)

test(lr\_cls,"Logistic Regression")

with open('model/lr.txt', 'wb') as file:

pickle.dump(lr\_cls, file)

file.close()

if os.path.exists('model/svc.txt'):

with open('model/svc.txt', 'rb') as file:

svc\_cls = pickle.load(file)

file.close()

test(svc\_cls,"Linear SVC")

else:

svc\_cls = LinearSVC()

svc\_cls.fit(X,Y)

test(svc\_cls,"Linear SVC")

with open('model/svc.txt', 'wb') as file:

pickle.dump(svc\_cls, file)

file.close()

if os.path.exists('model/ridge.txt'):

with open('model/ridge.txt', 'rb') as file:

ridge\_cls = pickle.load(file)

file.close()

test(ridge\_cls,"Ridge Classifier")

else:

ridge\_cls = RidgeClassifier()

ridge\_cls.fit(X,Y)

test(ridge\_cls,"Ridge Classifier")

with open('model/ridge.txt', 'wb') as file:

pickle.dump(ridge\_cls, file)

file.close()

if os.path.exists('model/nb.txt'):

with open('model/nb.txt', 'rb') as file:

nb\_cls = pickle.load(file)

file.close()

test(nb\_cls,"Multinomial Naive Bayes")

else:

nb\_cls = MultinomialNB()

nb\_cls.fit(X,Y)

test(nb\_cls,"Multinomial Naive Bayes")

with open('model/nb.txt', 'wb') as file:

pickle.dump(nb\_cls, file)

file.close()

if os.path.exists('model/sgd.txt'):

with open('model/sgd.txt', 'rb') as file:

sgd\_cls = pickle.load(file)

file.close()

test(sgd\_cls,"SGDClassifier")

else:

sgd\_cls = MultinomialNB()

sgd\_cls.fit(X,Y)

test(sgd\_cls,"SGDClassifier")

with open('model/sgd.txt', 'wb') as file:

pickle.dump(sgd\_cls, file)

file.close()

if os.path.exists('model/mlp.txt'):

with open('model/mlp.txt', 'rb') as file:

mlp\_cls = pickle.load(file)

file.close()

test(mlp\_cls,"Multilayer Perceptron Classifier")

classifier = mlp\_cls

else:

mlp\_cls = MLPClassifier()

mlp\_cls.fit(X,Y)

test(mlp\_cls,"Multilayer Perceptron Classifier")

with open('model/mlp.txt', 'wb') as file:

pickle.dump(mlp\_cls, file)

file.close()

classifier = mlp\_cls

def graph():

df = pd.DataFrame([['Logistic Regression','Accuracy',accuracy[0]],['Logistic Regression','Precision',precision[0]],['Logistic Regression','Recall',recall[0]],['Logistic Regression','FScore',fscore[0]],

['Linear SVC','Accuracy',accuracy[1]],['Linear SVC','Precision',precision[1]],['Linear SVC','Recall',recall[1]],['Linear SVC','FScore',fscore[1]],

['Ridge Classifier','Accuracy',accuracy[2]],['Ridge Classifier','Precision',precision[2]],['Ridge Classifier','Recall',recall[2]],['Ridge Classifier','FScore',fscore[2]],

['MultinomialNB','Accuracy',accuracy[3]],['MultinomialNB','Precision',precision[3]],['MultinomialNB','Recall',recall[3]],['MultinomialNB','FScore',fscore[3]],

['SGDClassifier','Accuracy',accuracy[4]],['SGDClassifier','Precision',precision[4]],['SGDClassifier','Recall',recall[4]],['SGDClassifier','FScore',fscore[4]],

['MLP Classifier','Accuracy',accuracy[5]],['MLP Classifier','Precision',precision[5]],['MLP Classifier','Recall',recall[5]],['MLP Classifier','FScore',fscore[5]],

],columns=['Parameters','Algorithms','Value'])

df.pivot("Parameters", "Algorithms", "Value").plot(kind='bar')

plt.show()

def recommendDrug():

text.delete('1.0', END)

global X

global drug\_name, condition, review, rating

global classifier

global tfidf\_vectorizer

filename = askopenfilename(initialdir = "Dataset")

testData = pd.read\_csv(filename)

testData = testData.values

for i in range(len(testData)):

review = cleanPost(testData[i,0].strip().lower())

array = tfidf\_vectorizer.transform([review]).toarray()

predict = classifier.predict(array)[0]

maxValue = 0

dname = "none"

print(str(array[0].shape)+" "+str(X.shape))

for j in range(len(X)):

score = dot(X[j], array[0])/(norm(X[j])\*norm(array[0]))

if score > maxValue:

maxValue = score

dname = drug\_name[j]

text.insert(END,"Disease Name: "+str(testData[i,0])+"\n")

text.insert(END,"Recommended Drug: "+str(dname)+"\n")

text.insert(END,"Predicted Ratings: "+str(predict)+"\n\n")

font = ('times', 15, 'bold')

title = Label(main, text='Integrated ML with NLP framework for Drug Recommendation System')

title.config(bg='HotPink4', fg='yellow2')

title.config(font=font)

title.config(height=3, width=120)

title.place(x=0,y=5)

font1 = ('times', 13, 'bold')

ff = ('times', 12, 'bold')

l1 = Label(main, text='Dataset Location:')

l1.config(font=font1)

l1.place(x=50,y=100)

tf1 = Entry(main,width=60)

tf1.config(font=font1)

tf1.place(x=230,y=100)

uploadButton = TkinterCustomButton(text="Upload Drug Review Dataset", width=300, corner\_radius=5, command=uploadDataset)

uploadButton.place(x=50,y=150)

preprocessButton = TkinterCustomButton(text="Read & Preprocess", width=300, corner\_radius=5, command=preprocessDataset)

preprocessButton.place(x=400,y=150)

tfidfButton = TkinterCustomButton(text="TF-IDF Features Extraction", width=300, corner\_radius=5, command=TFIDFExtraction)

tfidfButton.place(x=790,y=150)

trainMLButton = TkinterCustomButton(text="Train ML models", width=300, corner\_radius=5, command=TrainML)

trainMLButton.place(x=50,y=200)

graphButton = TkinterCustomButton(text="Performance Comparison", width=300, corner\_radius=5, command=graph)

graphButton.place(x=400,y=200)

predictButton = TkinterCustomButton(text="Recommend Drug from Test Data", width=300, corner\_radius=5, command=recommendDrug)

predictButton.place(x=790,y=200)

font1 = ('times', 13, 'bold')

text=Text(main,height=20,width=130)

scroll=Scrollbar(text)

text.configure(yscrollcommand=scroll.set)

text.place(x=10,y=250)

text.config(font=font1)

main.config(bg='plum2')

main.mainloop()

A diagram of a drug procedure

Description automatically generated

Fig. 1: Block Diagram of Proposed System.

Diagram, schematic

Description automatically generated

Architecture of MLP.

**RESULTS AND DISCUSSION**

**DRUGREVIEW Dataset**

**Data Set Information**

The dataset provides patient reviews on specific drugs along with related conditions and a 10-star patient rating reflecting overall patient satisfaction. The data was obtained by crawling online pharmaceutical review sites. The intention was to study.

1. Sentiment analysis of drug experience over multiple facets, i.e., sentiments learned on specific aspects such as effectiveness and side effects,
2. The transferability of models among domains, i.e., conditions, and
3. The transferability of models among different data sources (see 'Drug Review Dataset (Druglib.com)').

The data is split into a train (75%) and a test (25%) partition (see publication) and stored in two.tsv (tab-separated-values) files, respectively.

**Attribute Information**

* drugName (categorical): name of drug.
* condition (categorical): name of condition.
* review (text): patient review.
* rating (numerical): 10-star patient rating.
* date (date): date of review entry.
* usefulCount (numerical): number of users who found review useful.

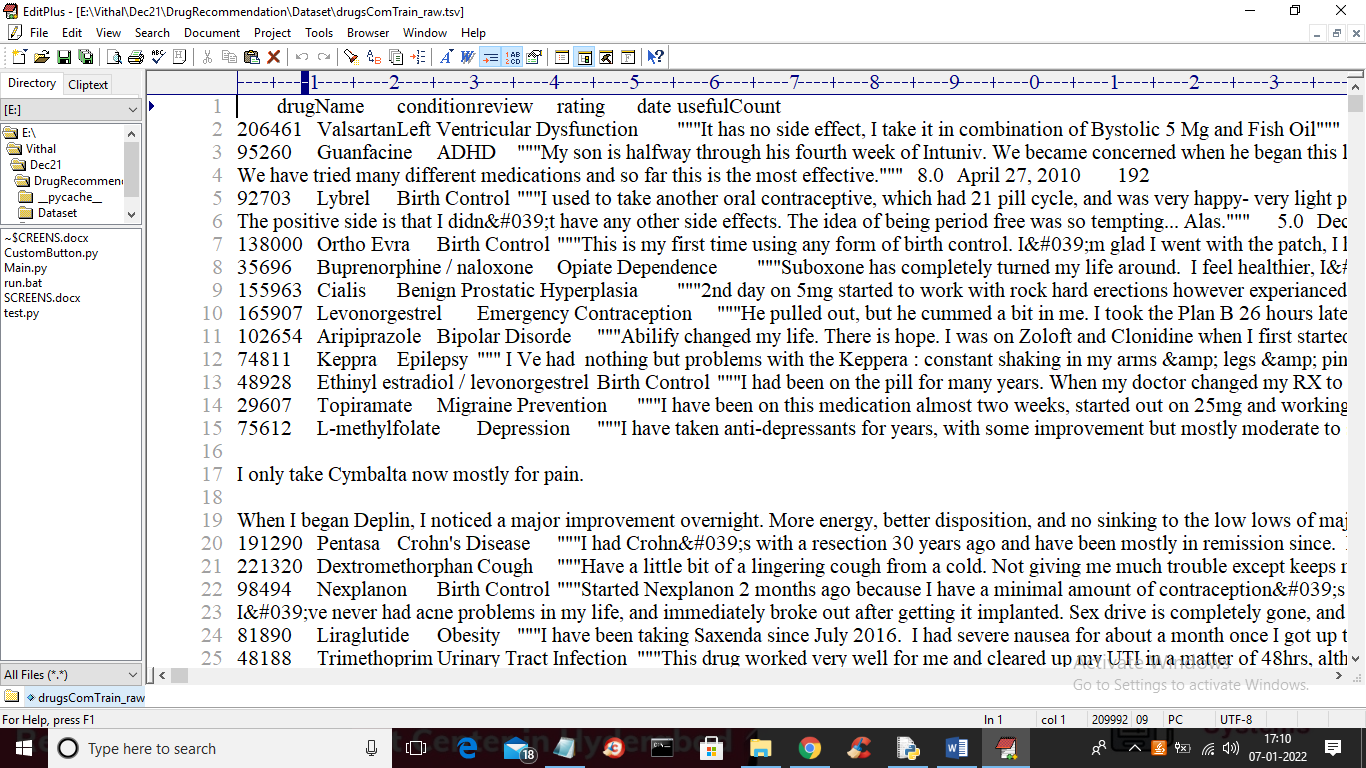


Figure 1. Sample dataset

In above screen first row represents dataset column names such as drug name, condition, review and rating and remaining rows contains dataset values and we will used above REVIEWS and RATINGS to trained machine learning models. Below is the test data which contains only disease name and machine learning will predict Drug name and ratings.

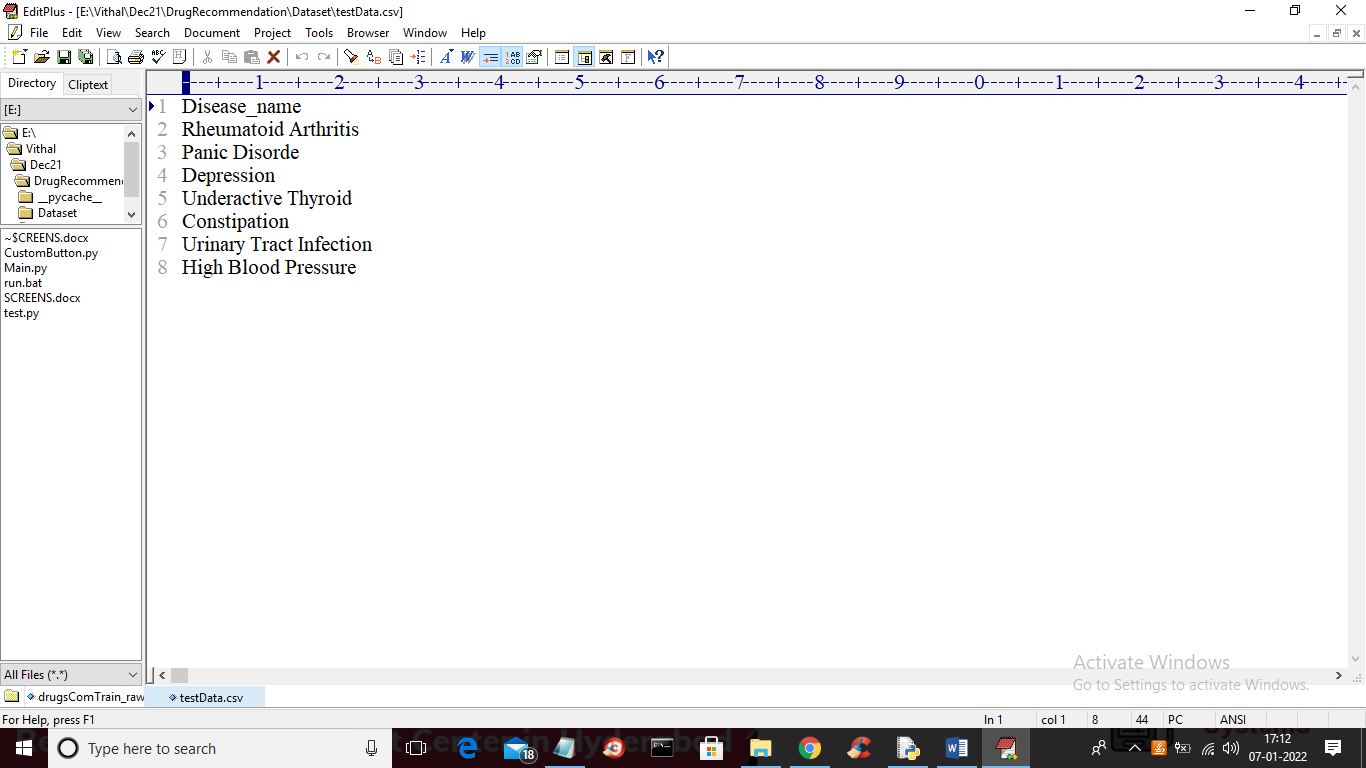


Figure 2. Dataset properties

In above test data we have only disease name and machine learning will predict ratings and drug names.

A picture containing chart

Description automatically generated

Figure 3. Drugs ratings graph

In above graph we can see dataset loaded and in graph x-axis represents ratings and y-axis represents total number of records which got that rating. Now close above graph and then click on ‘Read & Pre-process Dataset’ button to read all dataset values and then pre-process to remove stop words and special symbols and then form a features array.

A picture containing graphical user interface

Description automatically generated

Figure 5. Drug names dataset

In above screen we can see from all reviews stop words and special symbols are removed and in graph I am displaying TOP 20 medicines exist in dataset. In above graph x-axis represents drug name and y-axis represents its count.

Table 1. Performance comparison

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Method | Precision | Recall | F1-Score | Accuracy |
| Existing Logistic regression | 80.54 | 79.30 | 79.27 | 76 |
| Existing SVC | 70.51 | 71.18 | 70,46 | 67.80 |
| Existing Ridge classifier | 66.786 | 37.72 | 42.78 | 55.1 |
| Existing Multimodal navie bayes | 41.32 | 47.98 | 43.14 | 47.19 |
| Existing SGDC | 41.324 | 47.18 | 43.44 | 47.49 |
| Proposed MLP | 99.96 | 99.72 | 99.84 | 99.9 |

In above table for each algorithm we calculate accuracy, precision, recall and FSCORE and in all algorithms MLP has got high performance.

Chart, bar chart

Description automatically generated

Figure 6. Performance comparison graph

In above graph x-axis represents algorithm name and y-axis represents accuracy, precision recall and FSCORE where each different colour bar will represent one metric and in above graph we can see MLP got high performance.

Graphical user interface

Description automatically generated

Figure 7. Drug recommendations from test data.

In above screen for each disease name application has predicted recommended drug name and ratings.