**BIG DATA LAB REPORT**

YUTIKA KULWE

CED15I017

28TH MARCH 2019

Dataset: <https://www.kaggle.com/petersunga/google-amazon-facebook-employee-reviews>

1. Establish association between attributes and stars rating.
   1. How to run the program:
      1. python arm.py
   2. Input: All the attributes of the dataset.
   3. Implementation: Apyori library was used to implement Apriori, which helped in establishing associations between the attributes and stars rating.
   4. Code:

from sklearn import metrics

from collections import Counter

from scipy.spatial.distance import pdist,squareform

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

from apyori import apriori

from tqdm import tqdm

store\_data = pd.read\_csv('yutika.csv')

records = []

for i in range(98):

records.append([str(store\_data.values[i,j]) for j in range(9)])

association\_rules = apriori(records, min\_support=0.0002, min\_lift=3, min\_length=2)

association\_results = list(association\_rules)

for item in association\_results:

# first index of the inner list

# Contains base item and add item

pair = item[0]

items = [x for x in pair]

confidence = item[2][0][2]

lift = item[2][0][3]

print("Rule: " + items[0] + " -> " + items[1])

#second index of the inner list

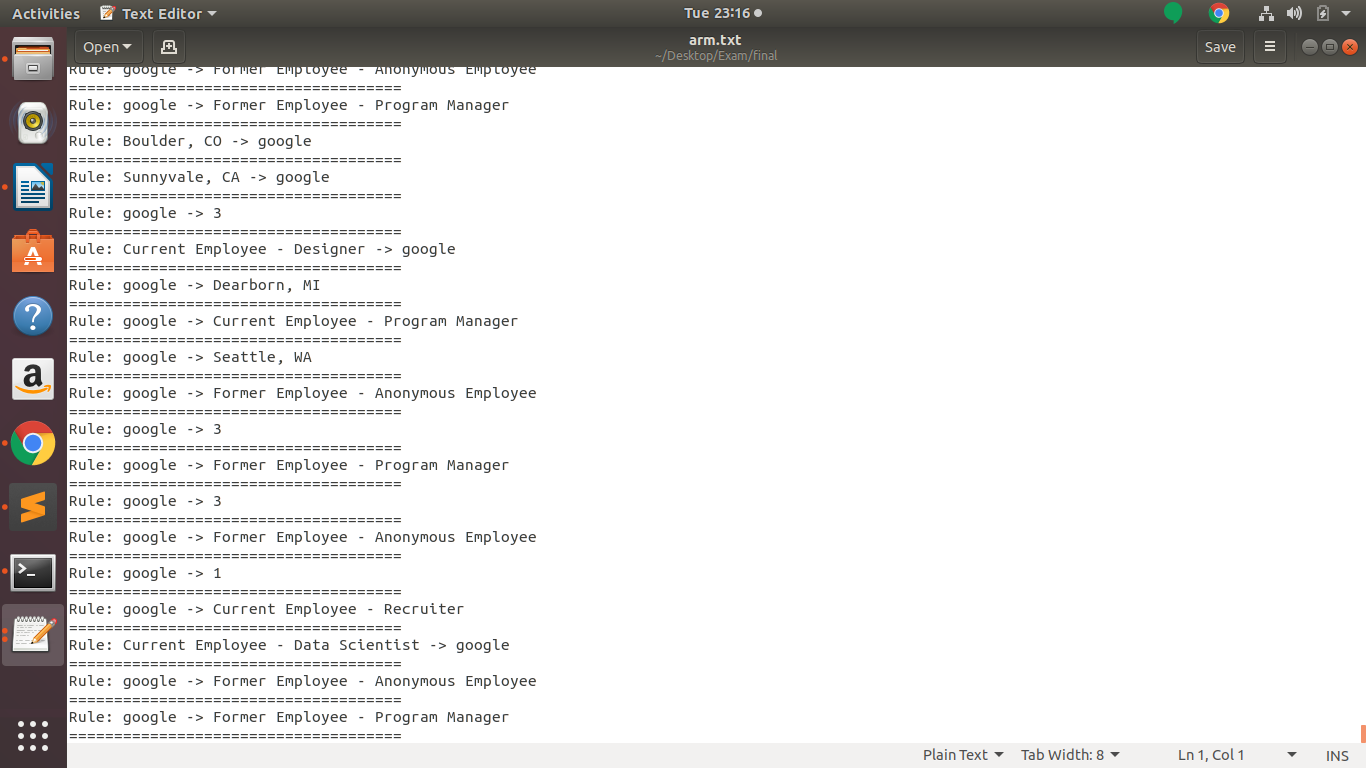
support\_AC = item[1]

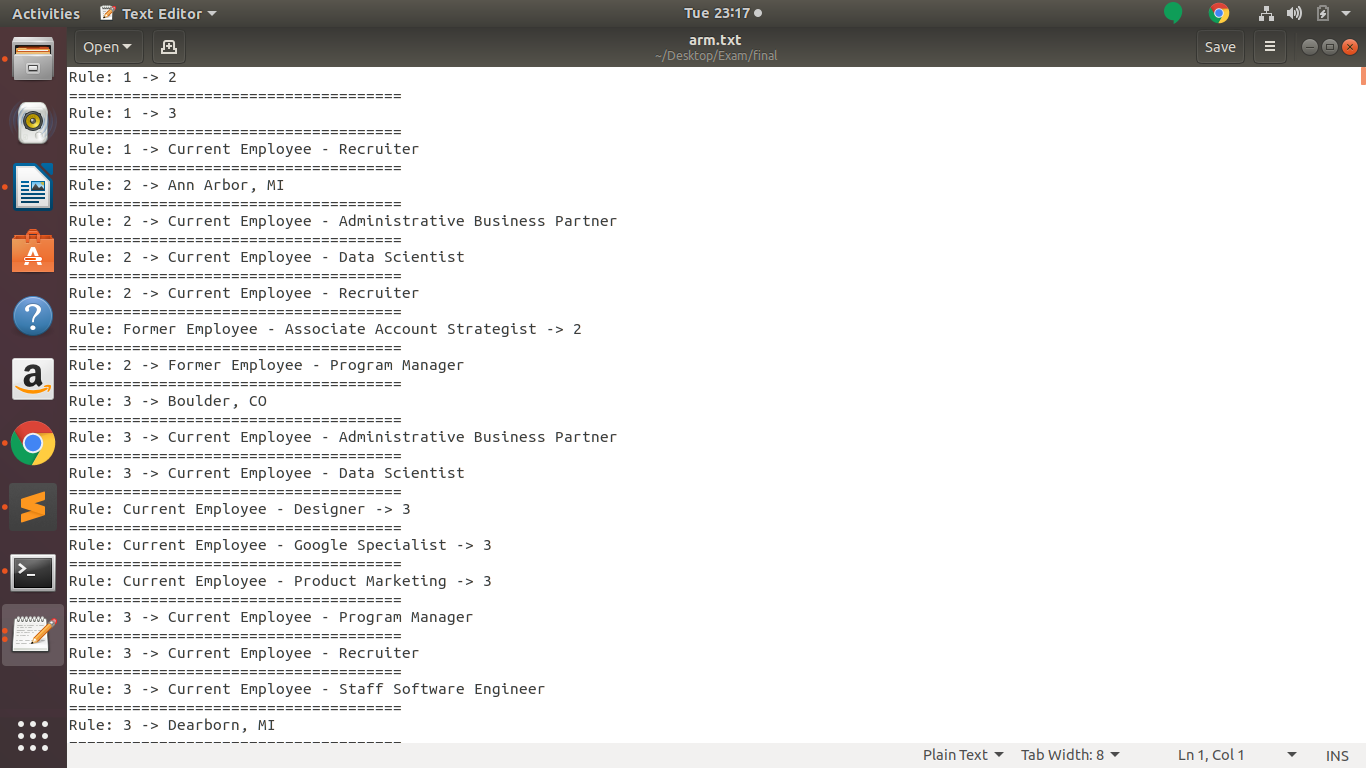
support\_A = support\_AC / confidence

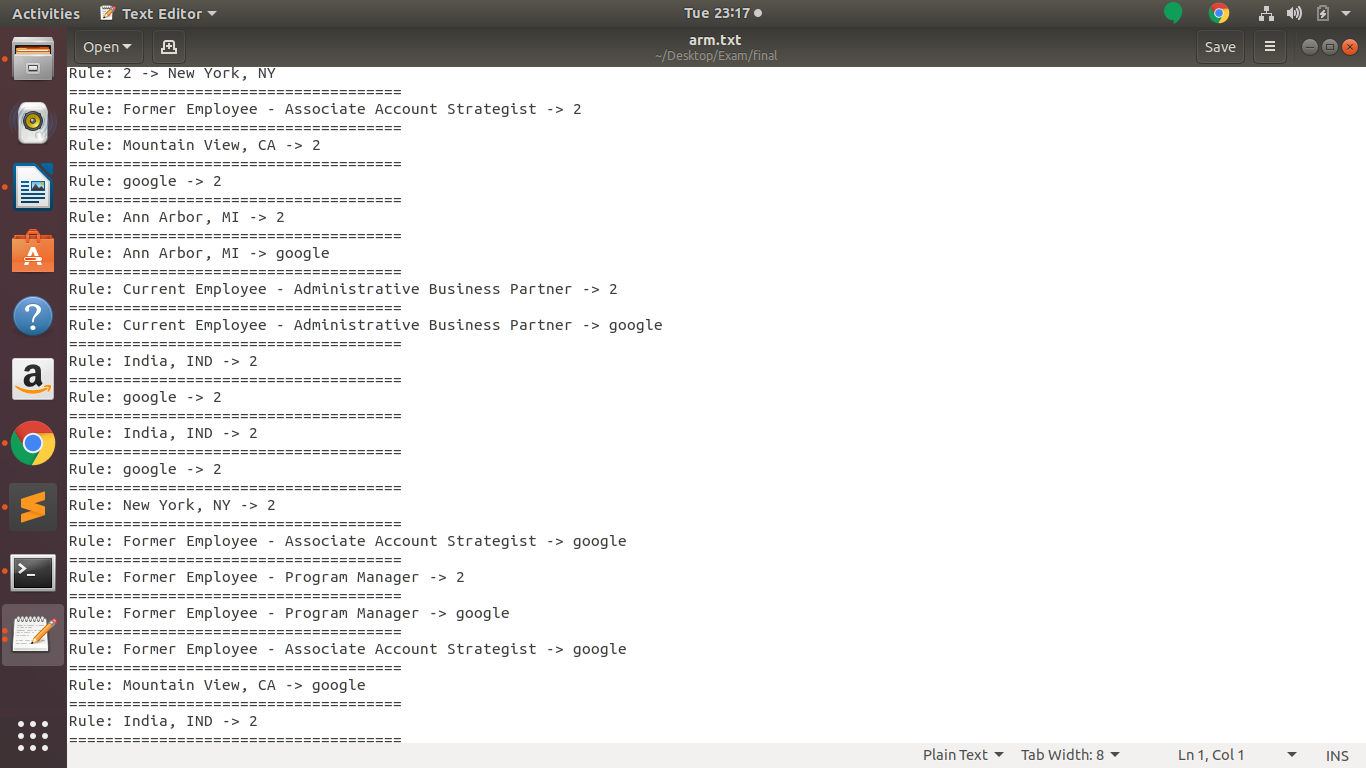
support\_C = confidence / lift leverage = support\_AC - support\_A\*support\_C

print("=====================================")

* 1. Output:





****

1. Fit any two classifiers for each stars rating.
   1. How to run the program:
      1. python 1RF.py
      2. python 1GB.py
   2. Input: Stars rating column
   3. Implementation: RandomForest classifier and GradientBoost classifier are used in this. You need to change to specific stars rating column in the code itself.
   4. Code:

*RANDOMFOREST CLASSIFIER:*

import matplotlib.pyplot as plt

import seaborn as sns

#import the data, specify data types

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

#%matplotlib inline

import seaborn as sns

import re

import pickle

#import mglearn

import time

from nltk.tokenize import TweetTokenizer

import nltk

from nltk import Text

from nltk.tokenize import regexp\_tokenize

from nltk.tokenize import word\_tokenize

from nltk.tokenize import sent\_tokenize

from nltk.corpus import stopwords

from nltk.stem import WordNetLemmatizer

from nltk.stem import PorterStemmer

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.linear\_model import LogisticRegression

from sklearn.naive\_bayes import MultinomialNB

from sklearn.multiclass import OneVsRestClassifier

from sklearn.model\_selection import cross\_val\_score

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

from sklearn import metrics

from sklearn.metrics import classification\_report

from sklearn.metrics import confusion\_matrix

from sklearn.model\_selection import GridSearchCV

from sklearn.pipeline import make\_pipeline

df=pd.read\_csv('employee\_reviews.csv')

df.dropna(inplace=True)

#df.drop(df.columns[0],axis=1,inplace=True)

non\_cat = [f for f in df.columns if df.dtypes[f] != 'object']

cat = [f for f in df.columns if df.dtypes[f] == 'object']

def treat\_missing\_numeric(df,columns,how = 'mean'):

if how == 'mean':

for i in columns:

print("Filling missing values with mean for columns - {0}".format(i))

df.ix[:,i] = df.ix[:,i].fillna(df.ix[:,i].mean())

elif how == 'mode':

for i in columns:

print("Filling missing values with mode for columns - {0}".format(i))

df.ix[:,i] = df.ix[:,i].fillna(df.ix[:,i].mode())

elif how == 'median':

for i in columns:

print("Filling missing values with median for columns - {0}".format(i))

df.ix[:,i] = df.ix[:,i].fillna(df.ix[:,i].median())

elif how == 'ffill':

for i in columns:

print("Filling missing values with forward fill for columns - {0}".format(i))

df.ix[:,i] = df.ix[:,i].fillna(method ='ffill')

elif type(how) == int or type(how) == float:

for i in columns:

print("Filling missing values with {0} for columns - {1}".format(how,i))

df.ix[:,i] = df.ix[:,i].fillna(how)

else:

print("Missing value fill cannot be completed")

return df

def treat\_missing\_categorical(df,columns,how = 'mode'):

if how == 'mode':

for i in columns:

print("Filling missing values with mode for columns - {0}".format(i))

df.ix[:,i] = df.ix[:,i].fillna(df.ix[:,i].mode()[0])

elif type(how) == str:

for i in columns:

print("Filling missing values with {0} for columns - {1}".format(how,i))

df.ix[:,i] = df.ix[:,i].fillna(how)

elif type(how) == int or type(how) == float:

for i in columns:

print("Filling missing values with {0} for columns - {1}".format(how,i))

df.ix[:,i] = df.ix[:,i].fillna(str(how))

else:

print("Missing value fill cannot be completed")

return df

treat\_missing\_numeric(df,non\_cat,how = 'mean')

treat\_missing\_categorical(df,cat,how = 'mode')

#print(df)

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

X = df.copy()

X = df['company']

y = df['senior-mangemnet-stars']

def holdout\_cv(X,y,size = 0.3, seed = 1):

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = size, random\_state = seed)

return X\_train, X\_test, y\_train, y\_test

X\_train, X\_test, y\_train, y\_test = holdout\_cv(X, y, size = 0.3, seed = 1)

from sklearn.ensemble import RandomForestClassifier

from sklearn.datasets import make\_classification

X, y = make\_classification(n\_samples=67528, n\_features=4,

n\_informative=2, n\_redundant=0,

random\_state=0, shuffle=False)

clf = RandomForestClassifier(n\_estimators=100, max\_depth=2,

random\_state=0)

clf.fit(X, y)

print(clf.feature\_importances\_)

#print(clf.fit(X, y))

n=clf.predict(list(X))

#print(n)

for i in range(1,len(n)+1):

print("the",i,"went to class",n[i-1])

#print(clf.predict(list(X)))

*GRADIENTBOOST CLASSIFIER:*

import matplotlib.pyplot as plt

import seaborn as sns

#import the data, specify data types

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

#%matplotlib inline

import seaborn as sns

import re

import pickle

#import mglearn

import time

from nltk.tokenize import TweetTokenizer # doesn't split at apostrophes

import nltk

from nltk import Text

from nltk.tokenize import regexp\_tokenize

from nltk.tokenize import word\_tokenize

from nltk.tokenize import sent\_tokenize

from nltk.corpus import stopwords

from nltk.stem import WordNetLemmatizer

from nltk.stem import PorterStemmer

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.linear\_model import LogisticRegression

from sklearn.naive\_bayes import MultinomialNB

from sklearn.multiclass import OneVsRestClassifier

from sklearn.model\_selection import cross\_val\_score

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

from sklearn import metrics

from sklearn.metrics import classification\_report

from sklearn.metrics import confusion\_matrix

from sklearn.model\_selection import GridSearchCV

from sklearn.pipeline import make\_pipeline

df=pd.read\_csv('employee\_reviews.csv')

df.dropna(inplace=True)

#df.drop(df.columns[0],axis=1,inplace=True)

non\_cat = [f for f in df.columns if df.dtypes[f] != 'object']

cat = [f for f in df.columns if df.dtypes[f] == 'object']

def treat\_missing\_numeric(df,columns,how = 'mean'):

'''

Function to treat missing values in numeric columns

Required Input -

- df = Pandas DataFrame

- columns = List input of all the columns need to be imputed

- how = valid values are 'mean', 'mode', 'median','ffill', numeric value

Expected Output -

- Pandas dataframe with imputed missing value in mentioned columns

'''

if how == 'mean':

for i in columns:

print("Filling missing values with mean for columns - {0}".format(i))

df.ix[:,i] = df.ix[:,i].fillna(df.ix[:,i].mean())

elif how == 'mode':

for i in columns:

print("Filling missing values with mode for columns - {0}".format(i))

df.ix[:,i] = df.ix[:,i].fillna(df.ix[:,i].mode())

elif how == 'median':

for i in columns:

print("Filling missing values with median for columns - {0}".format(i))

df.ix[:,i] = df.ix[:,i].fillna(df.ix[:,i].median())

elif how == 'ffill':

for i in columns:

print("Filling missing values with forward fill for columns - {0}".format(i))

df.ix[:,i] = df.ix[:,i].fillna(method ='ffill')

elif type(how) == int or type(how) == float:

for i in columns:

print("Filling missing values with {0} for columns - {1}".format(how,i))

df.ix[:,i] = df.ix[:,i].fillna(how)

else:

print("Missing value fill cannot be completed")

return df

def treat\_missing\_categorical(df,columns,how = 'mode'):

'''

Function to treat missing values in numeric columns

Required Input -

- df = Pandas DataFrame

- columns = List input of all the columns need to be imputed

- how = valid values are 'mode', any string or numeric value

Expected Output -

- Pandas dataframe with imputed missing value in mentioned columns

'''

if how == 'mode':

for i in columns:

print("Filling missing values with mode for columns - {0}".format(i))

df.ix[:,i] = df.ix[:,i].fillna(df.ix[:,i].mode()[0])

elif type(how) == str:

for i in columns:

print("Filling missing values with {0} for columns - {1}".format(how,i))

df.ix[:,i] = df.ix[:,i].fillna(how)

elif type(how) == int or type(how) == float:

for i in columns:

print("Filling missing values with {0} for columns - {1}".format(how,i))

df.ix[:,i] = df.ix[:,i].fillna(str(how))

else:

print("Missing value fill cannot be completed")

return df

treat\_missing\_numeric(df,non\_cat,how = 'mean')

treat\_missing\_categorical(df,cat,how = 'mode')

#print(df)

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

X = df.copy()

X = df['company']

y = df['senior-mangemnet-stars']

def holdout\_cv(X,y,size = 0.3, seed = 1):

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = size, random\_state = seed)

return X\_train, X\_test, y\_train, y\_test

X\_train, X\_test, y\_train, y\_test = holdout\_cv(X, y, size = 0.3, seed = 1)

from sklearn.ensemble import GradientBoostingClassifier

from sklearn.datasets import make\_classification

import pandas as pd

X, y = make\_classification(n\_samples=67528, n\_features=4,

n\_informative=2, n\_redundant=0,

random\_state=0, shuffle=False)

clf = GradientBoostingClassifier(n\_estimators=100, max\_depth=2,

random\_state=0)

clf.fit(X, y)

print(clf.feature\_importances\_)

#print(clf.fit(X, y))

n=clf.predict(list(X))

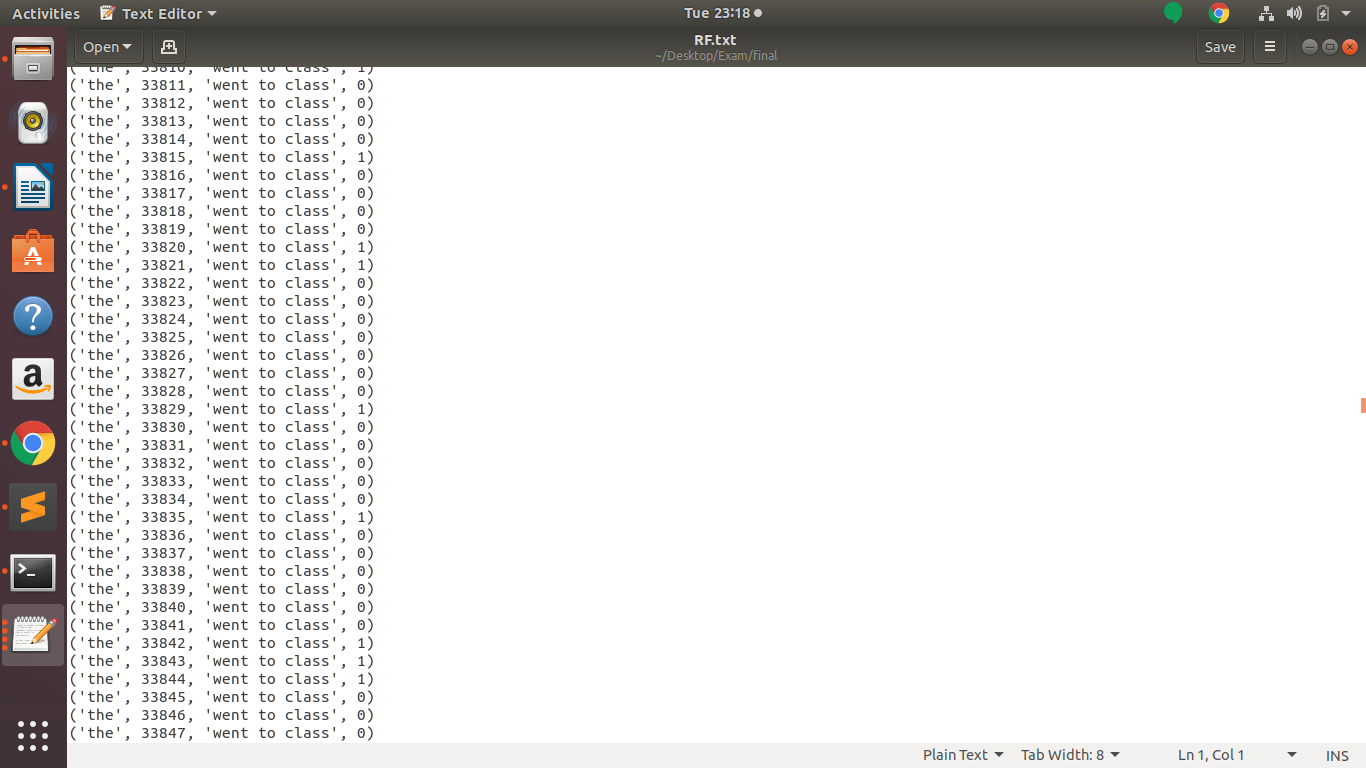
#print(n)

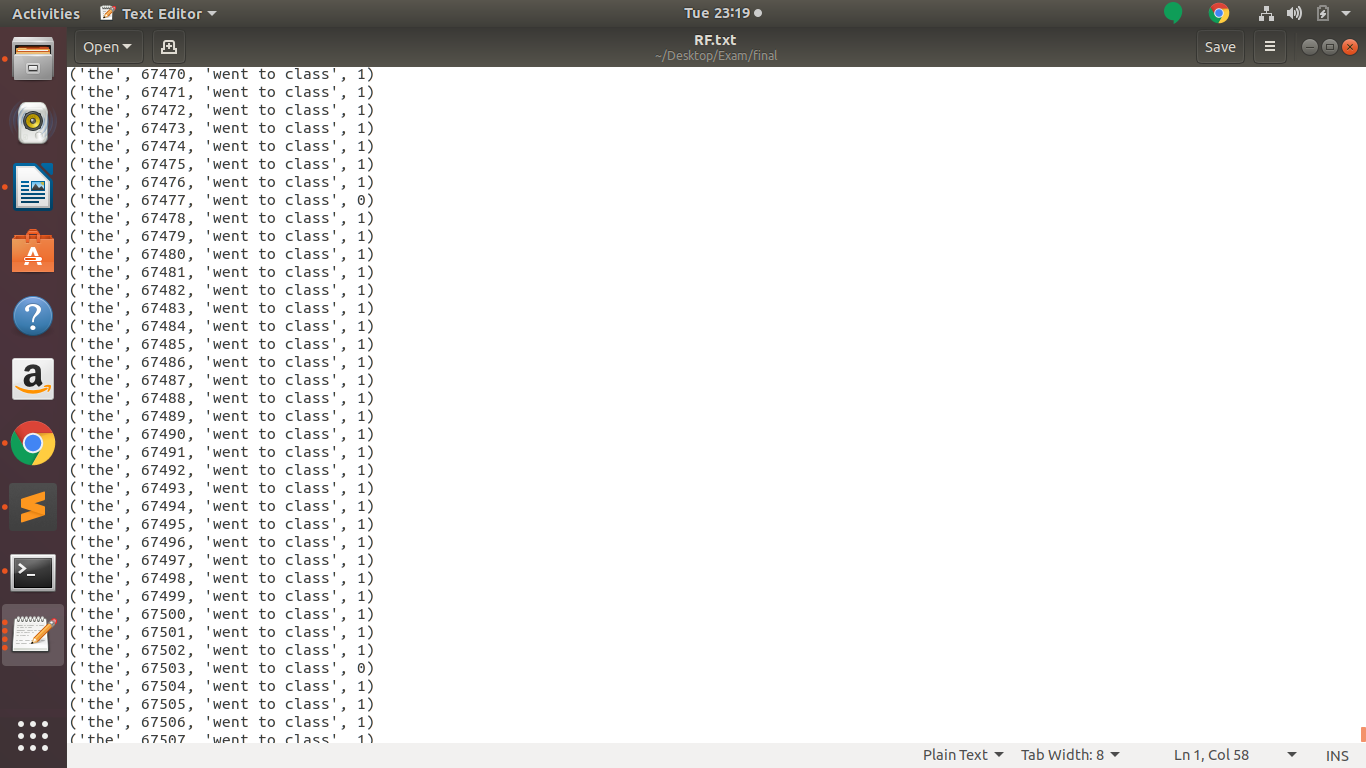
for i in range(1,len(n)+1):

print("the",i,"went to class",n[i-1])

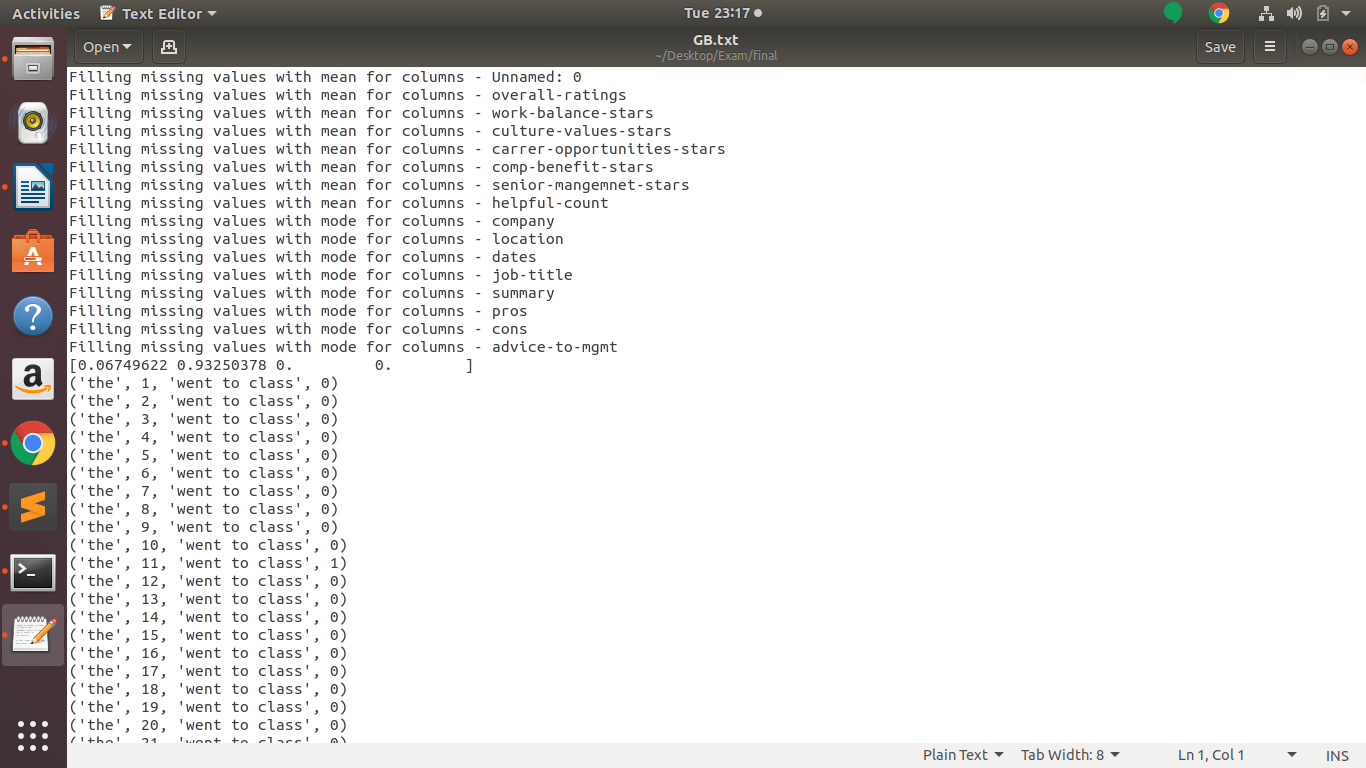
#print(clf.predict(list(X)))

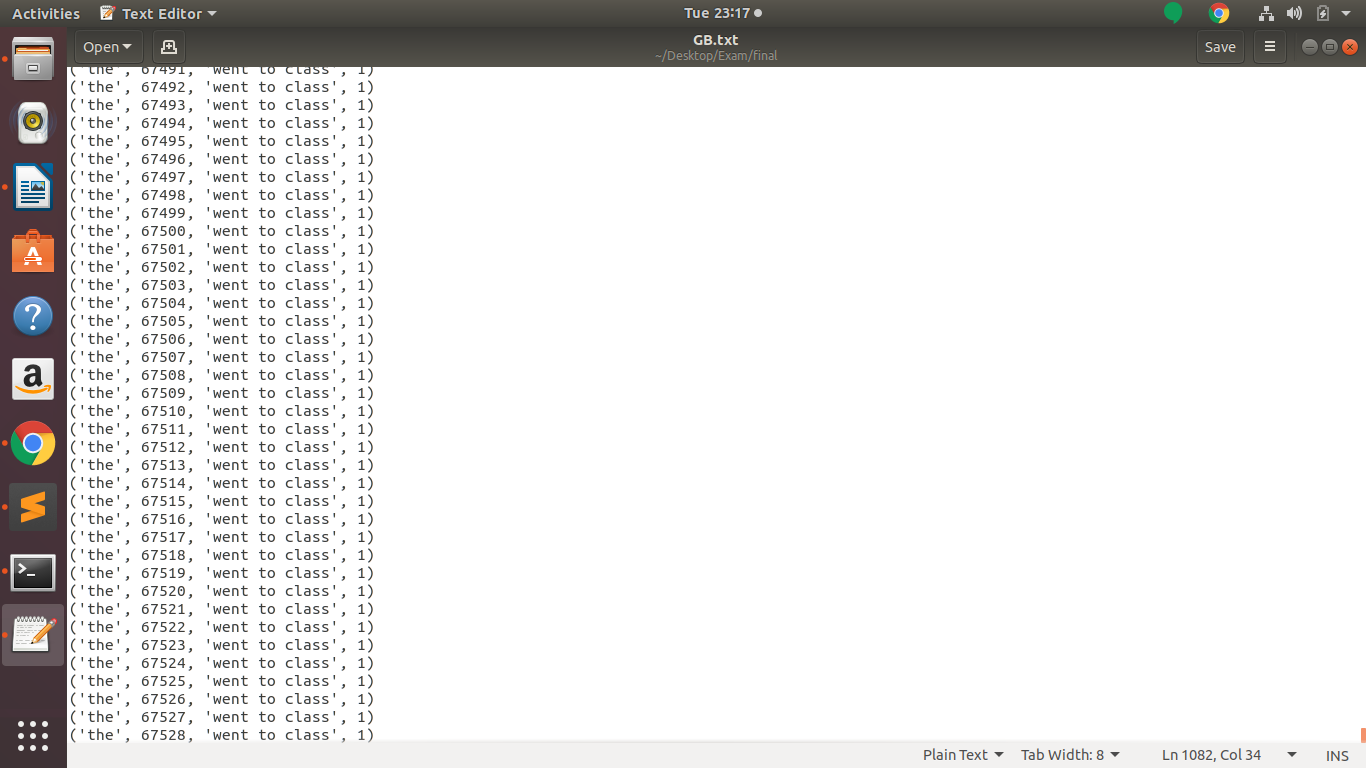
e. Output:

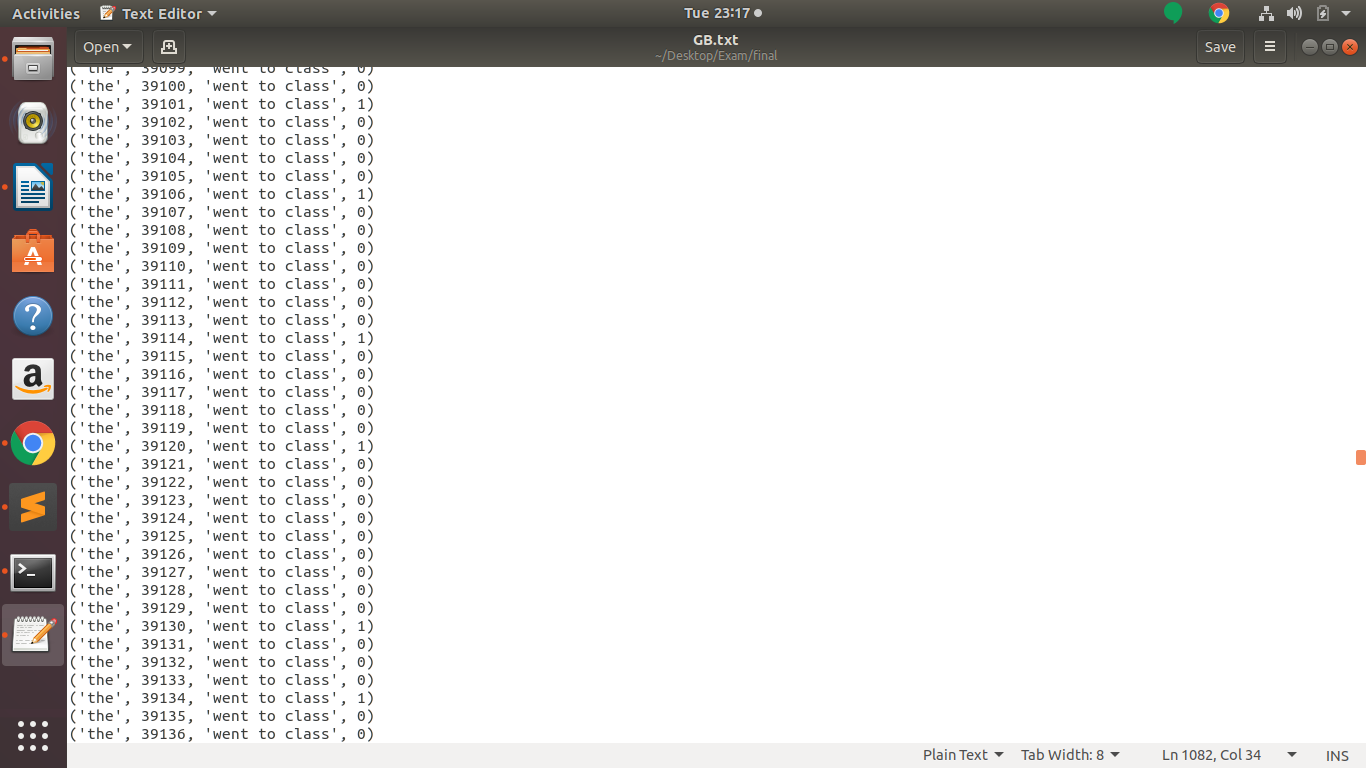


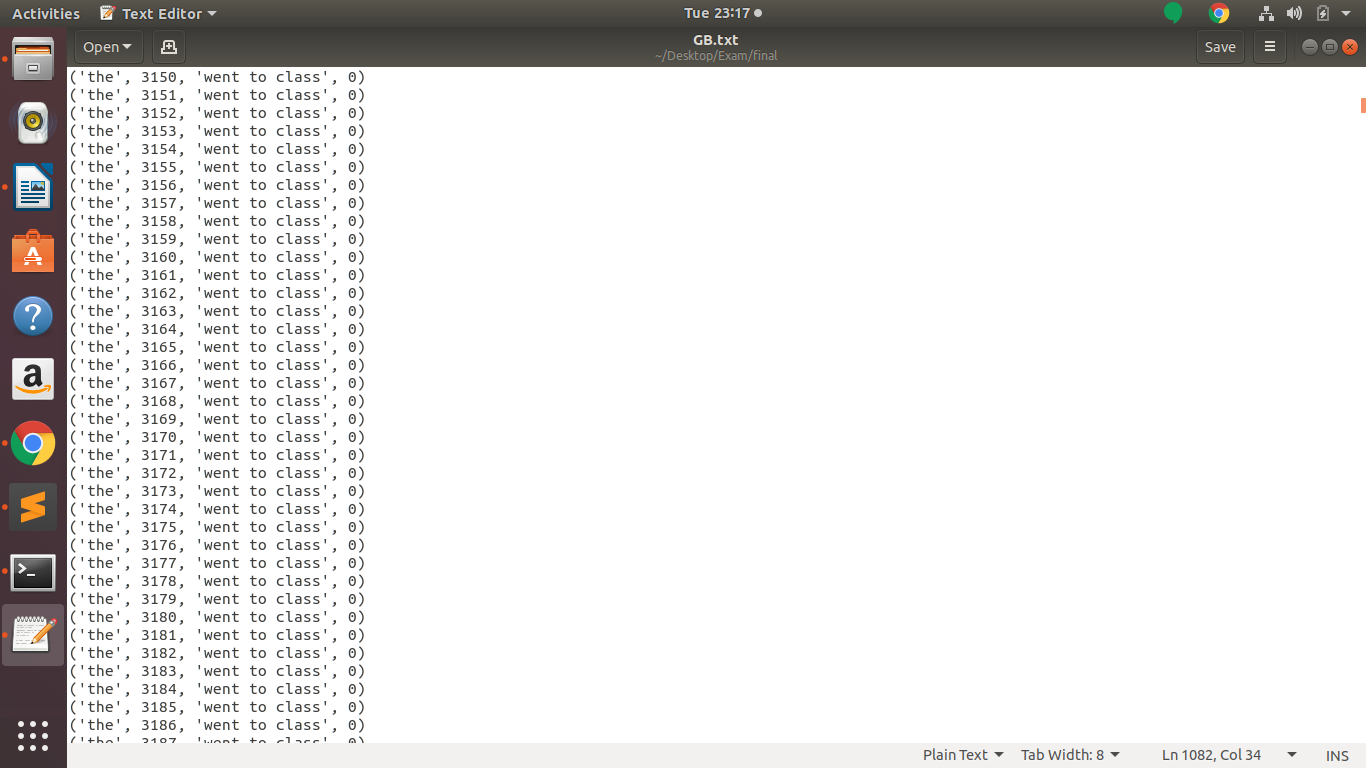


**RandomForest Classifier**









**Gradient Boost**

1. Cluster customer reviews using similarity metric.
   1. Kmeans clustering is used
   2. How to run the program:
      1. python kmeans.py
   3. Input: Customer reviews is factorized to group similar reviews together which is the summary column of the datatset. Here, “k” is 3. You can change the “k” value in the code.
   4. Implementation: It is implemented using sklearn,nltk library. Eucleadean distance is used to find the difference between two points and then cluster it. “pdist” is the inbuilt function in sklearn library that computes Euclidean distance.
   5. Code:

import matplotlib.pyplot as plt

import seaborn as sns

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import re

import pickle

import time

from nltk.tokenize import TweetTokenizer # doesn't split at apostrophes

import nltk

from nltk import Text

from nltk.tokenize import regexp\_tokenize

from nltk.tokenize import word\_tokenize

from nltk.tokenize import sent\_tokenize

from nltk.corpus import stopwords

from nltk.stem import WordNetLemmatizer

from nltk.stem import PorterStemmer

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.linear\_model import LogisticRegression

from sklearn.naive\_bayes import MultinomialNB

from sklearn.multiclass import OneVsRestClassifier

from sklearn.model\_selection import cross\_val\_score

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

from sklearn import metrics

from sklearn.metrics import classification\_report

from sklearn.metrics import confusion\_matrix

from sklearn.model\_selection import GridSearchCV

from sklearn.pipeline import make\_pipeline

df=pd.read\_csv('employee\_reviews.csv')

df.dropna(inplace=True)

from pandas.api.types import CategoricalDtype

labels, uniques = pd.factorize(df['summary'])

print("Numeric Representation : \n", labels)

print("Unique Values : \n", uniques)

X = labels

print(X)

y, z = pd.factorize(df['company'])

X, y = X.tolist(), y.tolist()

print(X, y)

x = [[0 for i in range(2)] for j in range(len(X))]

for i in range(len(X)):

x[i][0] = X[i]

x[i][1] = y[i]

print(x)

from sklearn.cluster import KMeans

import numpy as np

#X = np.array([[1, 2], [1, 4], [1, 0],[10, 2], [10, 4], [10, 0]])

estimator=KMeans(n\_clusters=3)

#kmeans = KMeans(n\_clusters=2, random\_state=0).fit(X)

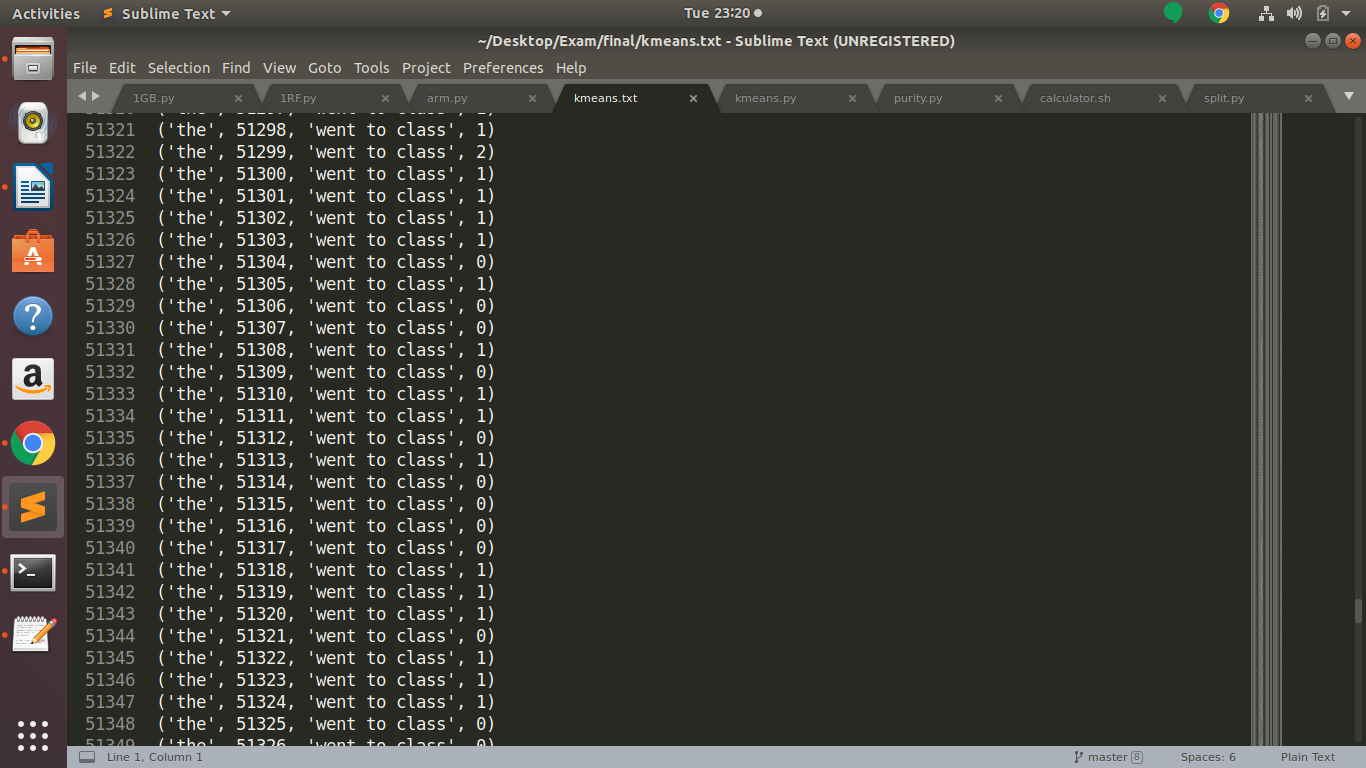
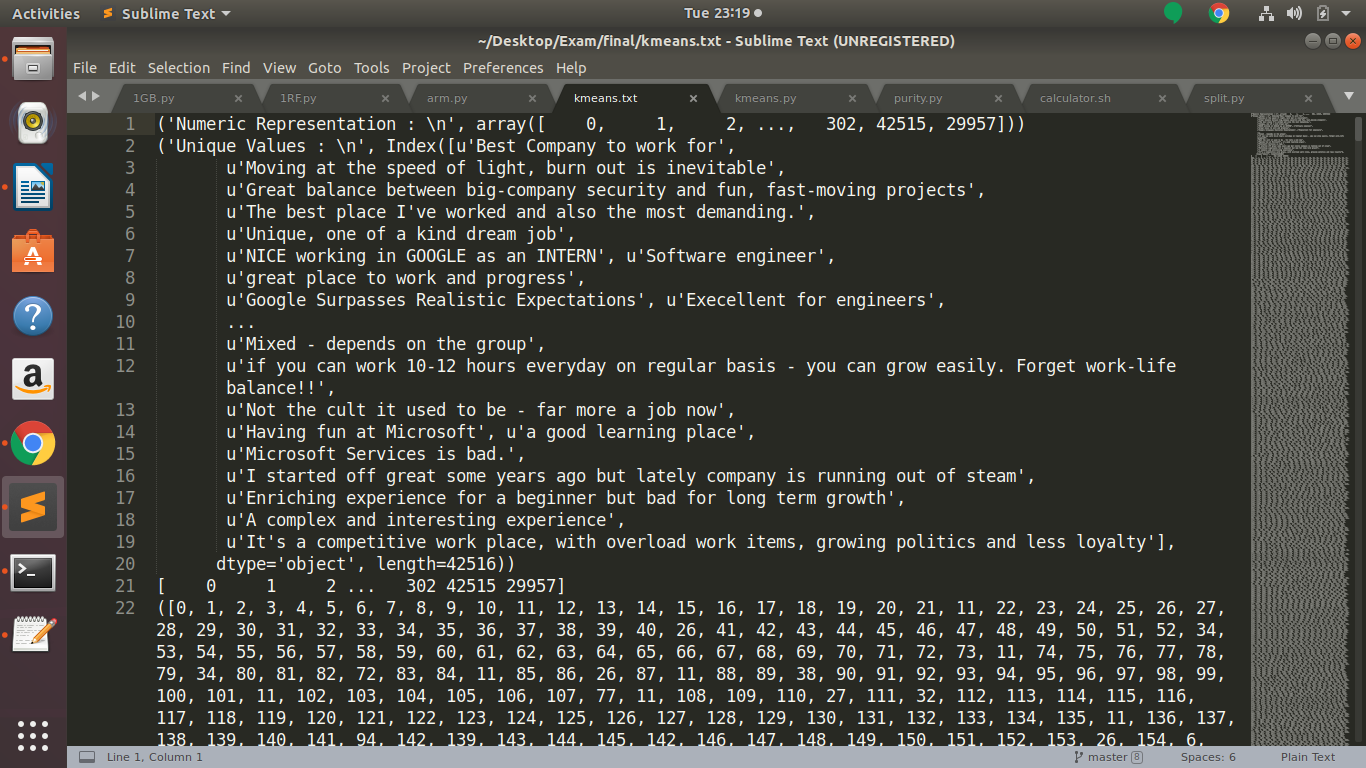
#kmeans.labels\_

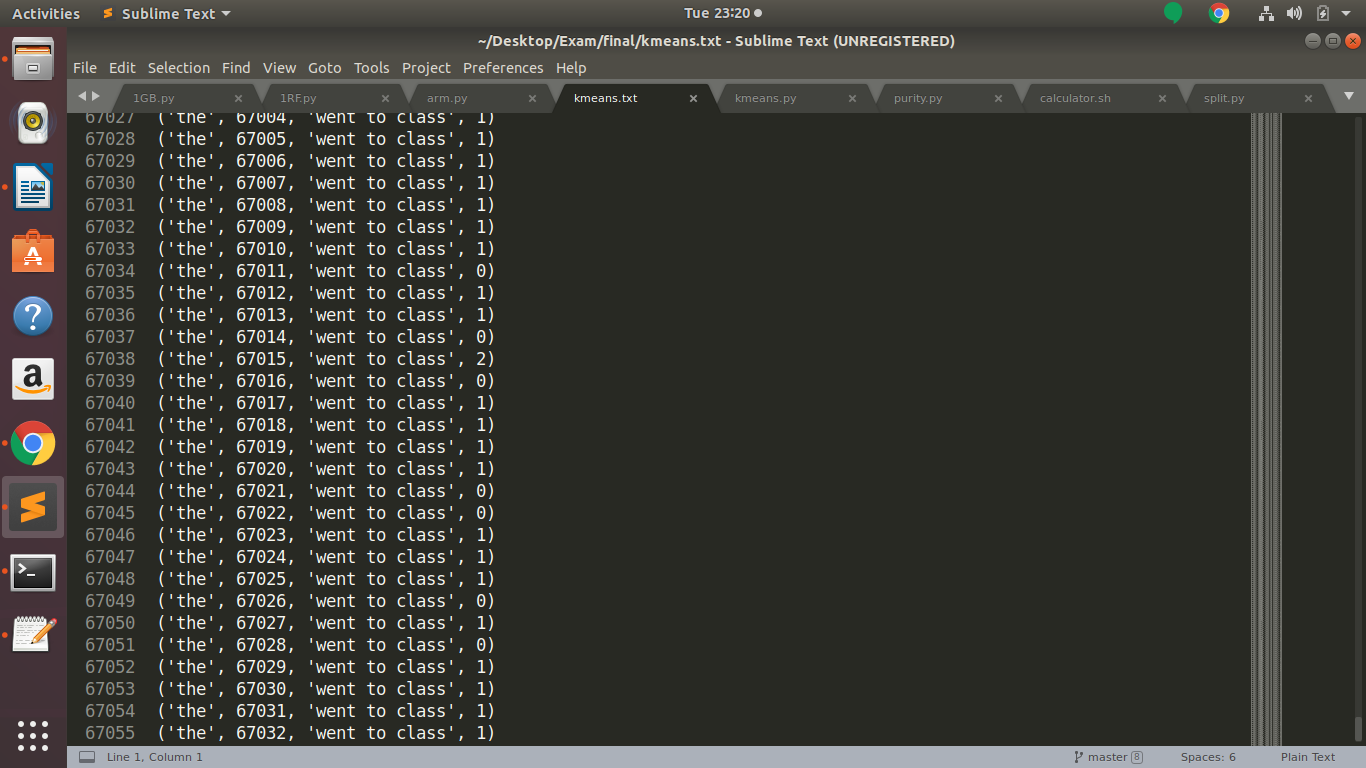
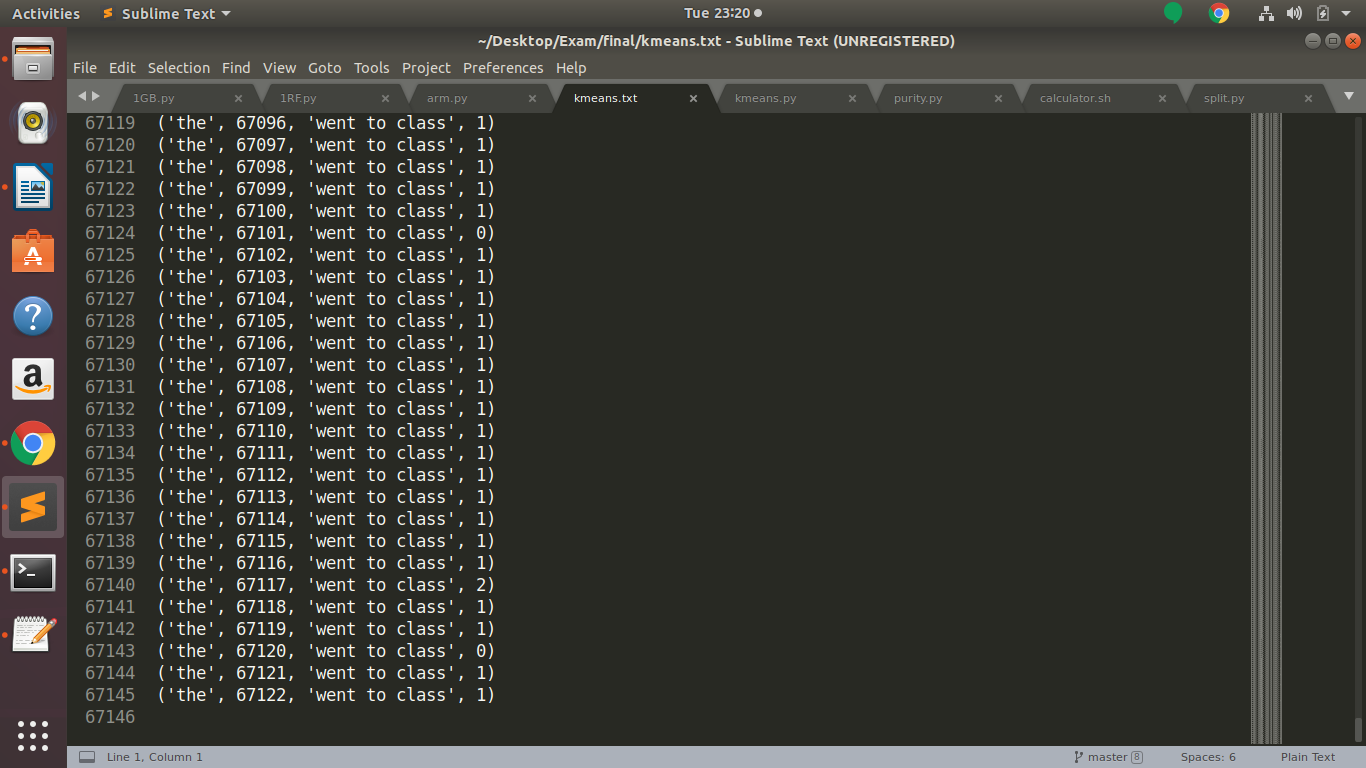
y\_kmeans=estimator.fit\_predict(x)

for i in range(1,len(y\_kmeans)+1):

print("the",i,"went to class",y\_kmeans[i-1])

* 1. Output:





1. Evaluate the quality of clustering in Q3.
   1. Purity metric is used to evaluate the quality.
   2. How to run the program:
      1. Python purity.py
   3. Input: I’m randomly generating 10 datatpoints everytime and checking the purity of it.
   4. Implementation: We are computing Distances betweens two points, Cluster centres, Labels that tells which point is part of which cluster.
   5. Code:

import matplotlib.pyplot as plt

import numpy as np

from sklearn.cluster import KMeans

from sklearn import metrics

from collections import Counter

from scipy.spatial.distance import pdist,squareform

import random

X = np.array([[random.randint(0,67530),random.randint(0,67530)],

[random.randint(0,67530),random.randint(0,67530)],

[random.randint(0,67530),random.randint(0,67530)],

[random.randint(0,67530),random.randint(0,67530)],

[random.randint(0,67530),random.randint(0,67530)],

[random.randint(0,67530),random.randint(0,67530)],

[random.randint(0,67530),random.randint(0,67530)],

[random.randint(0,67530),random.randint(0,67530)],

[random.randint(0,67530),random.randint(0,67530)],

[random.randint(0,67530),random.randint(0,67530)],])

true\_clusters\_labels = [0,0,0,0,0,1,1,1,1,1]

plt.scatter(X[:,0],X[:,1], label='True Position')

kmeans = KMeans(n\_clusters=3)

distances = kmeans.fit\_transform(X)

print("=================Distances==================")

print(distances)

plt.scatter(X[:,0],X[:,1], c=kmeans.labels\_, cmap='rainbow')

print("=================Cluster centres==================")

print(kmeans.cluster\_centers\_)

print("=================Labels==================")

print(kmeans.labels\_)

predicted\_labels = kmeans.labels\_

num\_of\_members = Counter(predicted\_labels)

cluster\_radius = {x:0 for x in predicted\_labels}

for distance,label in zip(distances,predicted\_labels):

cluster\_radius[label] += distance[label]

for label in num\_of\_members:

cluster\_radius[label] /= num\_of\_members[label]

print('Cluster Radius : ',cluster\_radius)

def purity\_score(y\_true, y\_pred):

# compute contingency matrix (also called confusion matrix)

contingency\_matrix = metrics.cluster.contingency\_matrix(y\_true, y\_pred)

print("=================Contingency Matrix==================")

print(contingency\_matrix)

# return purity

return np.sum(np.amax(contingency\_matrix, axis=0)) / np.sum(contingency\_matrix)

purity\_score(true\_clusters\_labels,predicted\_labels)

* 1. Output:

