52002 Final Assignment Spring 2021

Shira Esudri

A general description of the assignment ¶

In this assignment, we will read and analyze a large dataset from the Amazon Web Service (AWS) Simple Storage Service (S3).
The data we will work with is the *Amazon Customer Reviews Dataset*, that contains textual data of customer reviews, in addition to the rating of each product (on a scale of 1 to 5) and additional features.

The four parts of the assignments are:

- **Part 1:** Connecting to the service and reading data
- **Part 2:** Data preprocessing and feature engineering
- **Part 3:** Fitting a classification model to a large dataset
- **Part 4:** Fitting streaming data using Stochastic Gradient Descent

About the Data: Amazon Customer Reviews Dataset

The dataset contains over 130 million customer reviews vailable to researchers as part of this release, collected from from 1995 until 2015. The data is available in tab-delimited compressed (zipped) `tsv` files in the `amazon-reviews-pds S3 bucket` (see later).
Each line in the data files corresponds to an individual review (tab delimited).Bucket: `amazon-reviews-pds` .
Tab Separated Values Data pre-fix: `tsv` . The dataset is divided into different product categories, identified by `keys` . For example, a category for `cameras` is represented by the key:
`amazon-reviews-pds/tsv/amazon_reviews_us_Camera_v1_00.tsv.gz`

What is Amazon Web Services S3?

Amazon Simple Storage Service (Amazon S3, similar to Google Cloud Storage, Azure Blob Storage, ...) is storage for the Internet. It is designed to make web-scale data transfer, reading, writing and computing easier. Amazon S3 provides a simple interface that gives any developer access to the same highly scalable, reliable, fast, and inexpensive data storage infrastructure using code (for example in `python`).
However, the S3 file system is not typical, and is built around a key-value/object mapping (key is the location, object is the content and meta data of the file).

These key-values are stored in "folders" called `buckets` . Buckets are the fundamental containers in Amazon S3 for data storage, and contain `objects` (files). We can store an unlimited amount of data in a bucket, where each object can contain up to 5 TB of data. Each object is stored and retrieved using a unique developer-assigned key.

In [1]:
```
%%capture
# Installing libraries
%pip install boto3
!pip3 install flair
```

In [2]:
```
%%capture
import warnings
warnings.filterwarnings('ignore')

# Importing the AWS module. Allows also to work with and read from zipped file
s.
import boto3
import numpy as np
import pandas as pd
import gzip
import time
import random

# Useful text-preprocessing commands
from flair.models import TextClassifier
from flair.data import Sentence
classifier = TextClassifier.load('sentiment-fast') # building sentiment featur
es

import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')

# Useful machine-learning commands
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from sklearn.linear_model import LogisticRegression,SGDClassifier
from sklearn import metrics
from sklearn_pandas import DataFrameMapper
from sklearn.preprocessing import Normalizer
from sklearn.metrics import mean_squared_error
from sklearn.metrics import accuracy_score

import matplotlib.pyplot as plt  # for plotting

import math
from math import e
from math import log
# tic and toc functions for measuring time
def tic():
    import time
    global startTime_for_tictoc
    startTime_for_tictoc = time.time()
```

Part 1: Connecting to Amazon Web Server and reading data using boto3

In this section we will get familiar with the AWS S3 cloud storage using API modules. We will create a connection to the cloud storage and access the data.
Specifically, we will use the popular `boto3` AWS library for python to connect to the `amazon-reviews-pds` bucket.

1.(a) [3 pt] Using `boto3` 's method called `resource` , run the code below create a connection to AWS S3 named `s3conn` , with your `aws_access_key_id` and `aws_secret_access_key` .

Add a line defining a variable called `reviews` that points the `s3conn` conncetion to Amazon's `amazon-reviews-pds` data using the `Bucket` method of `boto3` . Print the `reviews` variable to verify that it represents the `Bucket` with the `amazon-reviews-pds` dataset.

Solution:

In [3]:
```
# If you have an AWS account, replace these with your key-id and access key:
akid = 'AKIA5SWU2IND3QS4UA5S'
sak = '9CDABiddu52jA6ROVwUMZSJV8ydpGJuNrLQJ4wdz'

# Using boto3's resource method, create a connection to AWS S3
s3conn = boto3.resource(
    's3',
    aws_access_key_id = akid,
    aws_secret_access_key = sak
)

#define a variable called reviews
reviews = s3conn.Bucket("amazon-reviews-pds")
print(reviews)
```

```
s3.Bucket(name='amazon-reviews-pds')
```

I define and print a variable called reviews, The variable points the s3conn conncetion to Amazon's amazon-reviews-pds data using the Bucket method of boto3.

1.(b) [6 pt] We are only interested in the data within the `tsv` (tab separated values) parent-key ("folder"):

- Run the code-cell below to get all the `keys` of the files within the `tsv` parent-key and their respected file size into a dedicated list.
  Each key should contain the name of the `object` and the size of the `object` in bytes.
- Filter the keys to include only `tsv` objects containing reviews, and only from the `us` .
- Print the first 15 elements of the filtered keys list. In addition, print the total size in GB (rounded to 3 dec. place) of all the objects of type `tsv` (zipped).

Solution:

In [4]:
```
keys_list = []
for my_bucket_object in reviews.objects.all():
    keys_list.append([my_bucket_object.key,my_bucket_object.size])

#Filter the keys to include only tsv objects containing reviews, and only from
the us

filter_keys = [] #create an empty list
for obj in keys_list: #for loop on the given list
  if obj[0].endswith('tsv.gz'):
    if 'us' in obj[0]: #fillter according the Q:
      filter_keys.append(obj) #append to the results list

#Print the first 15 elements of the filtered keys list
display(filter_keys[0:15])

#print the total size in GB
sum_ = 0 #counter to sum the total size in GB
for i in range(len(filter_keys)):
  sum_ = sum_ + filter_keys[i][1]
sum_GB = sum_/10**9 #in terms of GB
print("The total size in GB:", round(sum_GB,3))
```

```
[['tsv/amazon_reviews_us_Apparel_v1_00.tsv.gz', 648641286],
 ['tsv/amazon_reviews_us_Automotive_v1_00.tsv.gz', 582145299],
 ['tsv/amazon_reviews_us_Baby_v1_00.tsv.gz', 357392893],
 ['tsv/amazon_reviews_us_Beauty_v1_00.tsv.gz', 914070021],
 ['tsv/amazon_reviews_us_Books_v1_00.tsv.gz', 2740337188],
 ['tsv/amazon_reviews_us_Books_v1_01.tsv.gz', 2692708591],
 ['tsv/amazon_reviews_us_Books_v1_02.tsv.gz', 1329539135],
 ['tsv/amazon_reviews_us_Camera_v1_00.tsv.gz', 442653086],
 ['tsv/amazon_reviews_us_Digital_Ebook_Purchase_v1_00.tsv.gz', 2689739299],
 ['tsv/amazon_reviews_us_Digital_Ebook_Purchase_v1_01.tsv.gz', 1294879074],
 ['tsv/amazon_reviews_us_Digital_Music_Purchase_v1_00.tsv.gz', 253570168],
 ['tsv/amazon_reviews_us_Digital_Software_v1_00.tsv.gz', 18997559],
 ['tsv/amazon_reviews_us_Digital_Video_Download_v1_00.tsv.gz', 506979922],
 ['tsv/amazon_reviews_us_Digital_Video_Games_v1_00.tsv.gz', 27442648],
 ['tsv/amazon_reviews_us_Electronics_v1_00.tsv.gz', 698828243]]

The total size in GB: 32.377
```

First i created an empty results list, run a loop and fillter the keys to include only tsv objects, and only from the us containing reviews. To filter I use the "endwith" function and check if the string "us" is in each string. Then i Print the first 15 elements of the filtered keys list.

Next i calculated the total size in GB by running in for loop over, sum the size in the filter list and then divide the sum by 10^9 to make the results in term of GB.

1.(c) [6 pt] Using the output of **1.(b)**, create a `pandas` dataframe named `file_categories_df` with the following columns:

- `category` : the file's (object's) category, parsed from the key string (without unnecessary characters). If there are multiple categories with the same name, use `_00, _01` ... suffixes for different versions
- `size` : the size in bytes of the file
- `sizeGB` : the size in GB of the file
- `estSizeGB` : the estimated size in GB of the `uncompressed` file, assuming the gzip compresses a file to size of around `30%` of the original size.

For example, one row of the table should be:
`Digital_Software, 18997559, 0.017693, 0.05898`

Show the top-10 rows of the created data-frame using the `head` method.

Solution:

In [5]:
```
#creat a list for the category names
categor_list = []
for i in range(len(filter_keys)):
  categor_list.append(filter_keys[i][0][22:-13]) #remove unnecessary character
s

#add the 00, 01..
for i in range(len(categor_list)-1):
  if categor_list[i] == categor_list[i+1] and categor_list[i+1] == categor_lis
t[i+2]: #check if  identical
      categor_list[i] = categor_list[i]+"_00"
      categor_list[i+1] = categor_list[i][:-3]+"_01"
      categor_list[i+2] = categor_list[i+1][:-3]+"_02"
  if categor_list[i] == categor_list[i+1]:
    categor_list[i] = categor_list[i]+"_00"
    categor_list[i+1] = categor_list[i][:-3]+"_01"

#creat pandas dataframe
file_categories_df = pd.DataFrame(categor_list,columns=["category"])
file_categories_df["size"] = pd.DataFrame(filter_keys)[1] #the size in bytes
file_categories_df["sizeGB"] = pd.DataFrame(filter_keys)[1]/10**9 #the size in
GB
file_categories_df["estSizeGB"] = (pd.DataFrame(filter_keys)[1]/10**9)*(100/30
) #the estimated size in GB

file_categories_df.head(10) #show results
```

Out[5]:

	category	size	sizeGB	estSizeGB
0	Apparel	648641286	0.648641	2.162138
1	Automotive	582145299	0.582145	1.940484
2	Baby	357392893	0.357393	1.191310
3	Beauty	914070021	0.914070	3.046900
4	Books_00	2740337188	2.740337	9.134457
5	Books_01	2692708591	2.692709	8.975695
6	Books_02	1329539135	1.329539	4.431797
7	Camera	442653086	0.442653	1.475510
8	Digital_Ebook_Purchase_00	2689739299	2.689739	8.965798
9	Digital_Ebook_Purchase_01	1294879074	1.294879	4.316264

To create the list of categories, I created an empty category list. I run in for loop over the categories list from the previous section and removed the unnecessary characters of the strings (by stirngs operations).

After that I checked if there are identical categoriess. If so I added numbers 00, 01, 02 at the end as required. next I created a new data frame which has four columns: Category, size, sizeGB and estimated size GB. Finally I showed the first 10 categories as requiredby using Panda's head function.

1.(d) [6 pt] Reading the Data:

- Run the code cell below using the `download_file` method for the `bucket` of `s3conn`, to download the file of one of the six *smallest* categories (size < `30MB`) into the colab/local file-system, and read the entire file into a dataframe.
- Read the entire data from the category into a pandas dataframe called `df` and print the number of rows (data points) and columns (features).
- Compute and print the average size in bytes of each data point (an amazon product review). Use this number and the total size computed in **1.(b)** to estimate the total number of reviews in the entire dataset over all categories. How close is it to 130 million? explain what can cause the difference in numbers.
- Apply `df.head(5)` to view the start of the data-frame.

Note: during code development, it is allowed here and in other sub-questions (and even recommended) to limit the size of what we load into memory. However, for submission, the full files/requested data sizes should be used.

Solution:

In [6]:
```
%%capture
print('file to read/stream: ', filter_keys[13][0]) # downloading: Digital_Vide
o_Games. Modify to download a different category
fileToStream = filter_keys[13][0]
# Reading the file (may take time?)
s3conn.Bucket('amazon-reviews-pds').download_file(fileToStream, 'tmp.gz')
% ls /content/ -lah

with gzip.open('/content/tmp.gz', 'rb') as f_in:
    tmp = f_in.readlines() # Reading lines into a python object
```

In [7]:
```
#1.Read the entire data and Preparations
lst = [i.decode('utf-8') for i in tmp] #use decode to remove part of a string
df = pd.DataFrame([i.split('\t') for i in lst],columns= lst[0].split('\t')) #s
plit the tmp to create data frame
df = df.drop(labels=[0], axis=0) #drop the first row
df["review_date\n"] = [df["review_date\n"].iloc[i][:-2] for i in range(len(df[
"review_date\n"])) if df["review_date\n"].iloc[i].endswith('\n') ] # remove \n
df = df.rename(columns = {'review_date\n': 'review_date'})

#print the number of rows and columns
shape = df.shape
print("Df Shape ={}\nData Points = {}\nFeatures = {}".format(shape, shape[0],
shape[1]))

#2.Compute and print the average size in bytes of each data point
avg_dat_point_size = file_categories_df["size"].iloc[13]/len(df["marketplace"
])
print("The average size in bytes of each data point: ", round(avg_dat_point_si
ze,3))

#Use this number and the total size computed in 1.(b) to estimate the total nu
mber of reviews in the entire dataset over all categories
#compute the estimation
esti_tot_rev = file_categories_df["size"].sum()/avg_dat_point_size
print("The estimate total number of reviews in the entire dataset over all cat
egories: ", round(esti_tot_rev,3))

#3.Apply df.head(5) to view the start of the data-fram
df.head(5)
```

```
Df Shape =(145431, 15)
Data Points = 145431
Features = 15
The average size in bytes of each data point:  188.699
The estimate total number of reviews in the entire dataset over all categorie
s:  171581922.309
```

Out[7]:

	marketplace	customer_id	review_id	product_id	product_parent	product_title	
1	US	21269168	RSH1OZ87OYK92	B013PURRZW	603406193	Madden NFL 16 - Xbox One Digital Code	Di
2	US	133437	R1WFOQ3N9BO65I	B00F4CEHNK	341969535	Xbox Live Gift Card	Di
3	US	45765011	R3YOOS71KM5M9	B00DNHLFQA	951665344	Command & Conquer The Ultimate Collection [Ins...	Di
4	US	113118	R3R14UATT3OUFU	B004RMK5QG	395682204	Playstation Plus Subscription	Di
5	US	22151364	RV2W9SGDNQA2C	B00G9BNLQE	640460561	Saints Row IV - Enter The Dominatrix [Online G...	Di

First I did prepreparation for the data. I used the decode function to download part of a string, used the split function when the separation parameter was \t, and removed unnecessary passages in strings like \n. Next I turned the list into dataframe as requested and printed out the number of rows and columns in it by using the shape method.

Then I Computed and print the average size in bytes of each data point and estimate the total number of reviews in the entire dataset over all categories.To calculate the estimate the total number of reviews, I compute the sum of the size column in file_categories_df and divide it by the average size in bytes of each data point. In the end, i showed the first five rows of the data as requested.

How close is it to 130 million? explain what can cause the difference in numbers

As you can see the estimate total number of reviews in the entire dataset over all categories is over 130 million. A possible explanation for this could be that the category we are working on (filter_keys [13]: Digital Video Games) is more popular than others and therefore on average there are more reviews about it than about less popular categories.

Part 2: Data preprocessing and feature engineering

In this part we use the textual data to create predictive features, and preprocess additional features, to prepare the data-frame to be used in training a classifier.

2.(a) [7 pt] Apply the following common text normalization and standardization steps to the `review_body` textual column. It is recommended to use the `nltk` package:

1. Create a new column named `reviews_processed` , and assign it the `review_body` strings, just with all characteres as lower-case letters. All the following transformations should be done on the `reviews_processed` column.
2. Split to words using the `word_tokenize` method.
3. Keep only alpha-numerical values (you can use `str.isalpha()`).
4. Unite together words with similar meanining using the `WordNetLemmatizer` command.
5. Remove non-informative words (stop-words) using the `stopwords` command
6. Finally, join back all the tokens (words) to a single string for each row.
7. Remove rows with empty strings of `review_body` .

Write first a function that recieves as input a data-frame and modifies it according to the steps below. Then, apply the function to the `df` dataframe.

Show the top-10 records using the command `df.reviews_processed.head(10)` once you are done.

Note: Running the commands for this sub-questions may take a few minutes.

In [8]:
```
orig_col_names = df.columns #save of later

#create the function
def df_function(df,col):

  #for section 4+5
  lemmatizer =WordNetLemmatizer()
  stop_words = set(stopwords.words('english'))

#1.Create a new column named reviews_processed
  df[col] = df['review_body'].str.lower()

#2.Split to words using the word_tokenize method.
  df[col] = df[col].apply(word_tokenize)

#3.Keep only alpha-numerical values using isalpha method
  res = []
  for list in df[col]:
    res.append([i for i in list if i.isalpha()])
  df[col] = res

#4.Unite together words with similar meanining using the WordNetLemmatizer com
mand.
  res2 = []
  for i in range(len(df[col])):
    res2.append(([lemmatizer.lemmatize(word) for word in df[col].iloc[i]]))
  df[col] = res2

#5.Remove non-informative words using the stopwords command
  res3 = []
  for i in range(len(df[col])):
    res3.append([w for w in df[col].iloc[i] if w not in stop_words])
  df[col] = res3

#6.Join back all the tokens to a single string for each row by ubsing join fun
ction
  res4 = []
  for i in range(len(df[col])):
    res4.append(' '.join(df[col].iloc[i]))
  df[col] = res4

#7.Remove rows with empty strings
  df[col].replace('', np.nan, inplace=True)
  df.dropna(subset=[col], inplace=True)

  return df

#apply the function to the df dataframe.
df = df_function(df,"reviews_processed")

#Show the top-10 records using the command df.reviews_processed.head(10)
df.reviews_processed.head(10)
```

```
Out[8]: 1     keep buying madden every year hoping get back ...
        2                                               awesome
        3     prepping end world one thing installed pc hail...
        4                                               perfect
        5                                               awesome
        6                                               awesome
        7     like new skill like herbalism camping fun also...
        8                                                 super
        9                                 excellent fast secure
        10                                                   ok
        Name: reviews_processed, dtype: object
```

In this section I built a function that gets a data frame and a name of a column and makes changes to it as we were asked to do. I tried at every step to work efficiently and use as few two loops as possible. The purpose of the function is to make changes to the texts in it so that we can get information from the data of column.

First created a new column named review_processed. Next I separated the string that is in each cell in the word column by word_tokenize, went through all the lists in the column in each list and saved only alpha-numerical values by using the isalpha function. I Unite together words with similar meanining words with similar meanings by the WordNetLemmatizer command, When here too I had to go through every cell in the column for all the words. Then, I downloaded meaningless words by stop-words in the same way, combined in each cell all the words it has into one string using join function, downloaded rows in the data where the NA value appears. Finally showed the first 10 rows of the new column I created

2.(b) [6 pt] Sentiment analysis using the `flair` library: We next want to add *sentiment* features to each review, that use a pre-trained model to predict if the review text is positive or negative. Note that we use this only as a way for defining new predictive features from the text. We will train our actual classifier in **Part 3**.

1. *Execute* the cells below with the command:
   `classifier = TextClassifier.load('sentiment-fast')`
   to load the a pre-trained text classifier object
2. Next, Loop over the first 10 reviews and for each one apply `flair` 's `Sentence` method on the processed review text, an input the resulting sentence to the `classifier.predict()` method. Print for the first 10 reviews out both the reviews and their sentiment (`NEGATIVE/POSITIVE` and the sentiment score). Do the generated sentiments represent the text?
3. How long does it take to compute the sentiments of `100, 1,000` , and `10,000` datapoints in this manner? (do not print them, but make sure that the sentiments are actually computed). Extrapolate and estimate how long would it take to compute the sentiments over all data points in your category.

Solution:

In [9]:
```
classifier = TextClassifier.load('sentiment-fast')

#Loop over the first  10  reviews and for each one apply flair's Sentence meth
od on the processed review text
for i in range(10):
  sen_list = Sentence(df["reviews_processed"].iloc[i]) #use Sentence method  o
n each cell
  classifier.predict(sen_list) #classifier.predict on the Sentence[cell]
  print(df["reviews_processed"].iloc[i]) #print the reviews
  print(sen_list.labels) #print the lables

#compute the sentiments of 100 and check time
start1 = time.time() #compute the time
for i in range(100):
  sen_list1 = Sentence(df["reviews_processed"].iloc[i])
  classifier.predict(sen_list1)
end1 = time.time()
print(f"The Runtime of compute the sentiments of 100 datapoints: {round((end1
 - start1),3)}")

#compute the sentiments of 1000 and check time
start2 = time.time() #compute the time
for i in range(1000):
  sen_list2 = Sentence(df["reviews_processed"].iloc[i])
  classifier.predict(sen_list2)
end2 = time.time()
print(f"The Runtime of compute the sentiments of 1000 datapoints: {round((end2
- start2),3)}")

#compute the sentiments of 10000 and check time
start3 = time.time() #compute the time
for i in range(10000):
  sen_list3 = (Sentence(df["reviews_processed"].iloc[i]))
  classifier.predict(sen_list3)
end3 = time.time()
print(f"The Runtime of compute the sentiments of 10000 datapoints: {round((end
3 - start3),3)}")
```

```
keep buying madden every year hoping get back football year version little be
tter last year saying game look great thing wrong animation way player always
tripping br br gameplay still slowed bloated control used take two button gia
nt pita get done opponent snap ball play clock run br br turbo button back pl
ayer movement still slow awkward liked last year version guessing like chance
play anything training online game crossing finger hoping rest br br one thin
g recommend buy madden bundle game come download hate trading gamestop
[NEGATIVE (0.9999)]
awesome
[POSITIVE (0.9971)]
prepping end world one thing installed pc hail great yuri
[POSITIVE (0.8517)]
perfect
[POSITIVE (0.9941)]
awesome
[POSITIVE (0.9971)]
awesome
[POSITIVE (0.9971)]
like new skill like herbalism camping fun also like new build mode item
[POSITIVE (0.8617)]
super
[POSITIVE (0.9509)]
excellent fast secure
[POSITIVE (0.9975)]
ok
[POSITIVE (0.564)]
The Runtime of compute the sentiments of 100 datapoints: 0.248
The Runtime of compute the sentiments of 1000 datapoints: 2.524
The Runtime of compute the sentiments of 10000 datapoints: 26.442
```

I first looped through the first 10 reviews in the column I created in the previous section and in each cell I ran the Sentence function and then the classifier.predict. The classifier.predict function tries to understand whether the reviews are positive or negative in nature. I printed the ten reviews and their labels. It can be seen that for each review there are two parameters: positive / negative and score. Then I made three loops when each ran on a different range of numbers. For each loop I use the sentence function and the prediction function. In addition I measured the run time of each loop. As we can see the time rises as the range is bigger.

Do the generated sentiments represent the text?

It can be seen that there are positive and negative reactions and that the classifier manages very well to classify the reactions to positive and negative in a high rating. That is, It can be concluded that generated sentiments represent the text very well.

Extrapolate and estimate how long would it take to compute the sentiments over all data points in your category

The runtime seems to jump in multiples of 10 (+-), so for data like ours (a little less then 100,000 lines) the estimataion will be that the runtime will be around 300-400.

2.(c) [6 pt] To speed up the sentiment retrieval, we would like to implement a `mini-batch` approach: We first create a list of sentences `Sentence(text)``s for all input texts, and then go over them in groups of size` mini_batch_size `, and apply the` .predict `function to the entire` mini-batch` instead of 1 data-point at a time:

1. Define a `batch` function that has an iterable and a `mini-batch` size `int` as inputs, and yields out the mini-batches of the iterable.
2. Define a `get_sentiment` function that takes an `np.array` of text datapoints as input, and outputs a `np.array` of the label objects from the sentiments. Make sure you use the `mini_batch_size=128` and `verbose=True` parameters within the `.predict` method. Also, make sure you call the `.predict` method only once, on a list of `Sentence` objects.
3. Run the above functions on all the processed text datapoints of `df` using a `mini_batch_size` of `128` , and a `batch size` of `10,000` . Print out the total time it takes and compare it to the predicted time from **2.(b)**.
4. Add both the sentiments' scores (`sent_score`) and the sentiments' values (`sent_value`) as new separate colomns (new features) in `df` .

Execute a `df.head()` command to view the top values when done.

Note: Running the functions on all data points may take a few minutes.

Solution:

In [10]:
```
#According to the clarification given by the lecturer about this part there wa
s no need to build the batch function.
%%capture
#Define a get_sentiment function that takes an np.array of text datapoints as
 input, and outputs a np.array of the label objects from the sentiments
def get_sentiment(array):
  sentence_list = [] #empty list
  for i in range(len(array)):
    sentence_list.append(Sentence(array.iloc[i])) #create Sentence and add it
 to the list
  classifier.predict(sentence_list,mini_batch_size=128, verbose=True)  #predic
t all the  Sentence in the list together
  return [sentence_list[i].labels for i in range(len(sentence_list))] #returnt
the labels
```

In [11]:
```
#3.Run the above functions on all the processed text datapoints of df using a
 mini_batch_size of 128, and a batch size of 10,000.
#Print out the total time it takes and compare it to the predicted time from
 2.(b).

#use my function on all the processed text datapoints of df[reviews_processed]
labs = get_sentiment(df["reviews_processed"])

#calculate the time of get_sentiment on 10000 rows
start4 = time.time()
get_sentiment(df["reviews_processed"][:10000])
end4 = time.time()
print(f"Runtime of compute the sentiments of 10000 datapoints by get_sentiment
function: {round((end4 - start4),3)}") #print the time

#4.Add both the sentiments' scores (sent_score) and the sentiments' values (se
nt_value) as new separate coloms (new features) in df.
#add two list ot values and scors from the lables pbjects
scores =[]
values =[]
for i in range(len(df["reviews_processed"])):
  scores.append(round(labs[i][0].score,3))
  values.append(labs[i][0].value)

#add the new columns to the data
df["sent_score"] = scores
df["sent_value"] = values

#Execute a df.head() command to view the top values when done.
df.head()
```

```
Inferencing on batch 1133: 100%|██████████| 1133/1133 [03:01<00:00,  6.25it/
s]
Inferencing on batch 79: 100%|██████████| 79/79 [00:07<00:00, 11.06it/s]

Runtime of compute the sentiments of 10000 datapoints by get_sentiment functi
on: 20.799
```

Out[11]:

	marketplace	customer_id	review_id	product_id	product_parent	product_title	
1	US	21269168	RSH1OZ87OYK92	B013PURRZW	603406193	Madden NFL 16 - Xbox One Digital Code	Di
2	US	133437	R1WFOQ3N9BO65I	B00F4CEHNK	341969535	Xbox Live Gift Card	Di
3	US	45765011	R3YOOS71KM5M9	B00DNHLFQA	951665344	Command & Conquer The Ultimate Collection [Ins...	Di
4	US	113118	R3R14UATT3OUFU	B004RMK5QG	395682204	Playstation Plus Subscription	Di
5	US	22151364	RV2W9SGDNQA2C	B00G9BNLQE	640460561	Saints Row IV - Enter The Dominatrix [Online G...	Di

I built a function: get_sentiment - The function creates an empty list and then runs in a loop on the length of the array it was given. For each member in the array it runs the Sentence function and then adds it to the empty list. After this run classifier.predict on the list of sentences we received by 128 mini_batch. The function returns the labels of each sentence from the list of sentences it has created.

Next I created two lists. One that goes through the entire reviews_processed column and activates the Sentence function for each cell. The second only goes over the first 10000 rows in the same column and does the same thing. I ran the get_sentiment function I built on the list of 10000 "sentences" and checked how long it takes to run. As we see in this way the run is much faster than in the previous section.

Then I extracted the values and score of the first list I made (the one that contains "sentences" of the whole reviews_processed column). I added two new columns to the data: scores and values.

compare it to the predicted time from 2.(b).

As we can see the predicted time is faster then the predicted time from 2.(b)

(According to the clarification given by the lecturer in this part I did not build the batch function but I built it and used it in part 4.)

2.(d) [8 pt] Setting binary variables:

Write a function that modifies a `pandas` data-frame as follows:

- To simplify the target variable and get a binary classification problem, map star ratings of `1,2` to `0`, and `4,5` to `1` (corresponding to `negative` and `positive` sentiments respectively). Filter neutral star ratings of `3`.

In addition:

- Make sure the types of the data is compatible with modeling
- Set binary variables values to zero/one where applicable
- Get rid of `np.nan, np.inf, -np.inf`

Run the function on the data-frame `df` and then run `df.head()` to view the top values when done.

Solution:

In [12]:
```
#creat a modifing function
def modifies_func(data):
#map star ratings of 1,2 to 0, and 4,5 to 1
  data["binstar"] = data["star_rating"] #create new column
  for i in range(len(data["star_rating"])):
    if data["star_rating"].iloc[i] == "1":  #according the conditions
      data["binstar"].iloc[i] = 0
    if data["star_rating"].iloc[i] == "2":
      data["binstar"].iloc[i] = 0
    if data["star_rating"].iloc[i] == "4" :
      data["binstar"].iloc[i] = 1
    if data["star_rating"].iloc[i] == "5":
      data["binstar"].iloc[i] = 1

  #Filter neutral star ratings of 3.
  data = data[data["binstar"] != "3"]

  #change the value according POSITIVE\NEGATIVE
  for i in range(len(data["sent_value"])):
    if data["sent_value"].iloc[i] == "POSITIVE":
      data["sent_value"].iloc[i] = 1
    if data["sent_value"].iloc[i] == "NEGATIVE":
      data["sent_value"].iloc[i] = 0

  #Set binary variables values to zero/one where applicable
  data["star_rating"] = data["star_rating"].astype(float)
  data["sent_score"] = data["sent_score"].astype(float)
  data["total_votes"] = data["total_votes"].astype(int)
  data["helpful_votes"] = data["helpful_votes"].astype(float)
  data["product_parent"] = data["product_parent"].astype(float)
  data["customer_id"] = data["customer_id"].astype(float)

#Get rid of np.nan, np.inf, -np.inf
  data.replace('', np.nan, inplace=True)
  data.replace([np.inf, -np.inf], np.nan, inplace=True)
  data.dropna(inplace=True)

  return data
#Run the function on the data-frame df
df = modifies_func(df)
#df.head() to view the top values
df.head()
```

Out[12]:

	marketplace	customer_id	review_id	product_id	product_parent	product_title	
1	US	21269168.0	RSH1OZ87OYK92	B013PURRZW	603406193.0	Madden NFL 16 - Xbox One Digital Code	Di
2	US	133437.0	R1WFOQ3N9BO65I	B00F4CEHNK	341969535.0	Xbox Live Gift Card	Di
3	US	45765011.0	R3YOOS71KM5M9	B00DNHLFQA	951665344.0	Command & Conquer The Ultimate Collection [Ins...	Di
4	US	113118.0	R3R14UATT3OUFU	B004RMK5QG	395682204.0	Playstation Plus Subscription	Di
5	US	22151364.0	RV2W9SGDNQA2C	B00G9BNLQE	640460561.0	Saints Row IV - Enter The Dominatrix [Online G...	Di

In this section I have built a function whose purpose is to arrange the data in a way that will fit for the following sections. The function prepares the data for us to run a classification model on it later.

First I added a column called binstar (the target column) and convert it to binari. It has values 0,1,3. 0 if the star_rating column has a score of 1 or 2 and 1 if it has a score of 4 or 5. I filtered the data from binstr = 3 so the data contain only 0 or 1 value. Then I turned the variable "values" to 1 if positive and to 0 if negative, removed missing or (+-)infinite values and changed all the columns that have numeric values to float or int objects. Finally I Run the function on the data-frame df and show the results.

2.(e) [1 pt] In addition to the `flair` sentiment features we would like to add also a `TfidfVectorizer` feature.

Run the code cell below to create a numpy array named `final_df` that will be used for modeling in **Part 3**. Add a line printing the shape (number of samples and features) of the resulting array.

Solution:

In [13]:
```
# Add the features of the dataframe that you want to transform and/or combine
mapper = DataFrameMapper([
     ('reviews_processed', TfidfVectorizer(max_features=100)),
     ('helpful_votes', None),
     ('total_votes', None),
     ('sent_score', None),
     ('sent_value', None)
 ], df_out=False)

"""
Use the fit_transform method to transform the old dataframe into a new one
that can be fed to the machine learning algorithm.
"""
mapper_fit = mapper.fit(df)
final_df = mapper.transform(df) # a numpy array

#print the shape (number of samples and features) of the resulting array.
fin_df_shape = final_df.shape
print("DF Shape ={}\nData points = {}\nFeatures = {}".format(fin_df_shape, fin
_df_shape[0], fin_df_shape[1]))
```

```
DF Shape =(133389, 104)
Data points = 133389
Features = 104
```

Part 3: Fitting a classification model to a large dataset

Our goal is to predict the binarized star rating (`binstar`) variable using the other features for each review.

3.(a) [6 pt] Create a `x_train,x_test,y_train,y_test` random split of the `final_df` and the target `binstar` , with the test set containing 20% of the data and the training set containing 80% of the data.

In addition, normalize/standardize the data as you wish.

In [14]:
```
#set the results
random.seed(100)

#create a x_train,x_test,y_train,y_test random split of the final_df and the t
arget binstar using train_test_split from sklearn packege
X_train, X_test, y_train, y_test = train_test_split(pd.DataFrame(final_df) , d
f["binstar"], test_size = 0.20)

#I kept a copy of data train, data test and the target variable without normal
ization because I will need you after this in question 4.
X_train_not_norm, X_test_not_norm,  = X_train, X_test
y_train_not_norm, y_test_not_norm = y_train, y_test

#normalize the data by Normalizer from sklearn
X_test = Normalizer().fit_transform(X_test)
X_train = Normalizer().fit_transform(X_train)
```

In this section we were asked to fit a classification model to a large data field when our variable variable is binstar. To do this we first apply our data to the train and test for the explanatory variables (X) and for the objective variable (Y). I performed this operation on me by the built-in train_test_split function of the sklearn package

3.(b) [5 pt] Fit a logistic regression model to the training set. You may use the `sklearn` package. Print the train and test model accuracy

Solution:

In [15]:
```
#Fit a logistic regression model to the training set
reg_train = LogisticRegression().fit(X_train, y_train)
accuracy_train = reg_train.score(X_train, y_train)
accuracy_test = reg_train.score(X_test, y_test)

#Print the train  model accuracy
print("Logistic Regression model - train accuracy:", round(accuracy_train,3))

#Print the test  model accuracy
print("Logistic Regression model - test accuracy:", round(accuracy_test,3))
```

```
Logistic Regression model - train accuracy: 0.883
Logistic Regression model - test accuracy: 0.882
```

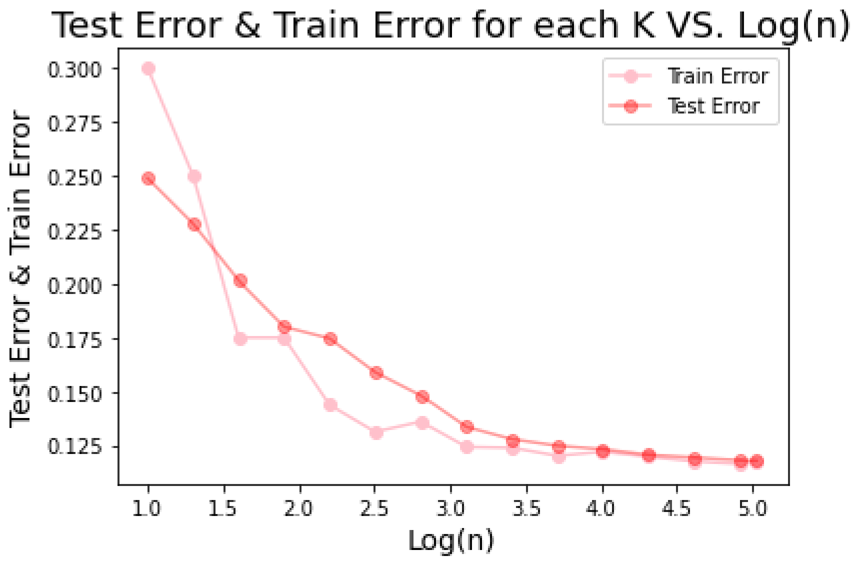
Now, I applied a logistic regression model (by a built-in function, LogisticRegression from a sklearn package) once to the Tain data and once to the test data. After this I ran the score function to calculate the accuracy index of each of the regressions. It can be seen that the accuracy is very high both in regression on the train data and also in regression on the test data.

3.(c) [6 pt] For different lengths of $n = 10 \times 2^k$, for $k = 0, 1, . . , 14$ extract only the first n values in the train set (i.e. first rows of `x_train` and first values of `y_train`) and use them to fit the logistic regression model. (If n is larger than the total number of rows, set n to the actua number).

Plot the training error (for each k on the appropriate train set) and test error (on the entire test set) vs. the sample size n shown on a log-scale. Do we see an improvement when increasing n?

Solution:

In [16]:
```
#For different n=10×2k , for  k=0,1,..,14  extract only the first  n  values i
n the train set
n = []
for k in range(15):
  if 10*(2**k) < len(X_train):
    n.append(10*(2**k))
  if 10*(2**k) > len(X_train): #If the number is greater than the length of th
e data then we will add the length of the data to the list.
    n.append(len(X_train))

#use the numers to take parts of the data and fit the logistic regression mode
l
train_error = []
test_error = []
for i in range(len(n)):
  regg_train = LogisticRegression().fit(X_train[:n[i]], y_train[:n[i]])
  train_error.append(1-regg_train.score(X_train[:n[i]], y_train[:n[i]])) #comp
ut error for test
  test_error.append(1-regg_train.score(X_test, y_test)) #comput error for test

#create the data frame for the plot
data_plot = pd.DataFrame(train_error,columns = ["train_error"])
data_plot["test_error"] = test_error
data_plot["log(n)"] = np.log10(n)

#plot
fig,ax = plt.subplots()
ax.plot(data_plot["log(n)"],data_plot["train_error"], color="pink", marker="o"
)
ax.plot(data_plot["log(n)"], data_plot["test_error"],color="red",alpha = 0.4,m
arker="o")

# set labels
ax.set_xlabel("Log(n)",fontsize=14)
ax.set_ylabel("Test Error & Train Error",fontsize=14)
ax.set_title('Test Error & Train Error for each K VS. Log(n)',fontsize=18)
plt.legend(["Train Error",'Test Error'])
plt.show()
```



In this section I examined how the accuracy changes for different data sizes (or the opposite way: how the error decreases as the data size increases). For this purpose I built a "n" list containing 15 values equal to 102^*k when K runs up to 14 (including 0). (If the 10*2^k > data length I add the len(data) into n[i]). Next I performed a logistic regression, each time for a larger portion of the data (according to the n size) and calculated the test and train error for each such regression. I created a new data and presented in a plot the errors of the train and the test for each size of the data (according to n) when the axis is X is a log (n).

Do we see an improvement when increasing n ?

It can be clearly seen that the error decreases as the n is greater i.e. the larger the data. We can see it for the both test for the train. In addition it can be seen that for small part of the data there are little jumps but in almost all the values the test error is larger than the train error (as expected) and the more data is used the error decreases and at the end when reaching all the data the distance between the errors becomes minimal. That is, as there are more observations the accuracy increases and the error decreases and the distance between the accuracy (or the error) between the test and the train decreases, as expected.

3.(d) [9 pt] Combining models from similar data:

- Find the maximal power k such that the test error for $n = 10 \times 2^k$ is at least 0.02 lower than the test error for the maximal n (denoted n_{max} you have used in **3.(c)**.
- Split the maximal training set you have used into n_{max}/n random blocks of equal size.
- Run a logistic regression model on each block separately
- Finally, average the fitted models coefficients to get a combined model.
- Report the train and test accuracy for the combined model on the entire (unified) training and test set.
- How does it compare to the results from the **3.(c)** where fitting the model using the entire dataset? is the accuracy diminished/comparable/improved?
- At what circumstances would you recommend this approach of splitting the data to blocks and combining the models?

Solution:

In [17]:
```
#set the results
random.seed(100)

#comput the max k
max_k_test = np.round(test_error - test_error[-1],3)
inx = 0
for i in range(len(max_k_test)):
  if max_k_test[i]> 0.02:
    inx = i

#Split the maximal training set you have used into  nmax/n  random blocks of e
qual size.

#compute the the blocks
maxn = n[-1]
n_mechane = 10*2**inx
blocks = int(round(maxn/n_mechane,0))
lines = list(range(0,len(X_train)))
coeff = [] #list for the coeff

#run over the block and preforme LogisticRegression
for i in range(blocks):
  subset = list(random.sample(lines,round(len(X_train)/blocks-1))) #randomize
 the train
  lines = [i for i in lines if i not in subset]
  y_sub = np.array(y_train)[subset] #create the y (for this block)
  X_sub = X_train[subset,:] #remove the y from the data
  log_reg = LogisticRegression().fit(X_sub,y_sub) #fit LogisticRegression
  coeff.append(log_reg.coef_) #take coef

#average the fitted models coefficients to get a combined model.
#calculate the mean of each list of coeff in the big doeff list
coeff_sum = coeff[0]
for i in range(1,len(coeff)):
  coeff_sum += coeff[i]
coeff_mean = coeff_sum/len(coeff)

#build the model
logit_model_train = e**(X_train@coeff_mean[0])/ (1+e**(X_train@coeff_mean[0]))
log_model_test = e**(X_test@coeff_mean[0])/(1+e**(X_test@coeff_mean[0]))

#the y should be binary (Logistic Regression) so i built trashhold = 0.5, if l
ogit_model >=  0.5 so 1 and  if logit_model >=  0.5 so 0
for i in range(len(logit_model_train)):
  if logit_model_train[i] == 0.5:
    logit_model_train[i] = 1
  if logit_model_train[i] > 0.5:
    logit_model_train[i] = 1
  if logit_model_train[i] < 0.5:
    logit_model_train[i] = 0

for i in range(len(log_model_test)):
  if log_model_test[i] == 0.5:
    log_model_test[i] = 1
  if log_model_test[i] > 0.5:
    log_model_test[i] = 1
  if log_model_test[i] < 0.5:
    log_model_test[i] = 0

#Report the train and test accuracy for the combined model on the entire (unif
ied) training and test set.
mse_logit_model_train = ((y_train-logit_model_train)**2).sum() /len(logit_mode
l_train) #the formula of mse
mse_log_model_test = ((y_test-log_model_test)**2).sum() /len(log_model_test) #
the formula of mse
acuur_logit_train = round(1-mse_logit_model_train,3)
acuur_logit_test = round(1-mse_log_model_test,3)

print("The train accuracy for the combined model on the entire data: ", acuur_
logit_train)
print("The test accuracy for the combined model on the entire data: ", acuur_l
ogit_test)

#compare to the accuracy of  3.c (print the accuracy estimates to make the com
parison easier)
print("The train accuracy for the model on the entire data from Q 3.c: ", roun
d(1-data_plot["train_error"][14],3))
print("The test accuracy for the model on the entire data from Q 3.c: ", round
(1-data_plot["test_error"][14],3))
```

```
The train accuracy for the combined model on the entire data:  0.791
The test accuracy for the combined model on the entire data:  0.788
The train accuracy for the model on the entire data from Q 3.c:  0.883
The test accuracy for the model on the entire data from Q 3.c:  0.882
```

The purpose of this section was to run a logistic regression in parts and use the coefficient's averages to manually construct a logistic regression model and then check how the accuracy index changes in this case to compare the accuracy index of the model in the previous section.

In this section I randomly divided the data into blocks of the same size (n_max/n). next I ran on each data block logistic regression and saved its coefficients. After I got a list of coefficient lists, I did an average for each sub-list so in the end I got a list of coefficient averages. Following this I manually built a logistic regression model with the train data and the list of the coefficient averages by the familiar formula of logistic regression. I also added a condition that wiii assures that the Y obtained from the model will be binary (Properly in logistical regression) Finally I calculated the MSE of the model (accuracy) and the error (1-MSE) and presented the results.

How does it compare to the results from the 3.(c) where fitting the model using the entire dataset? is the accuracy diminished/comparable/improved?

It can be seen that there is differences in the accuracy index compared to the previous part. The combine model i built here has lower accuuracy then the model on the entire data on 3.c. This may be due to the fact that regression on the blocks retains features that do not represent all the data, so the accuracy index decreases when we compute it on all the data together.

At what circumstances would you recommend this approach of splitting the data to blocks and combining the models?

In this method we actually divide the data into blocks when for each block logistic regression is run. One of the advantages of the method is the ability to preserve interesting or unexpected results in each block specifically that we cannot see when we do logistic regression on all the data together. Therefore, perhaps use this method in a situation where the data is arranged in a certain way i.e. the data is composed of parts and each part has similar properties. So in this case if we apply logistic regression on each part we may get better accuracy.

3.(e) [10 pt] Combining models from different datasets:

- Pick 10 different categories from the Amazon reviews dataset.
- For each dataset, read a batch of $100,000$ examples and split to train/test randoly with an `80%|20%` ratio as before.
- **Note:** You may pick large categories. **Do not** download the whole file as in **1.(d)**. Instead, use `readlines` to download only the first $100,000$.
- Run a logistic regression model on each category separately and report the train/test accuracies in a table.
- Finally, average the fitted models coefficients from all categories to get a `combined model` .
- Report the train and test accuracy for the `combined model` on the training and test set of each category separately.
- Did adding $\times 10$ more examples help in improving the accuracy compared to fitting each category separately? why do you think this happened?

Solution:

In [18]:
```
%%capture
#1.take 10 categoris
ten_cat = filter_keys[9:19]

#Pick  10  different categories from the Amazon reviews dataset.
big_list = []
for i in range(len(ten_cat)):
  fileToStream2 = ten_cat[i][0]
# Reading the file (may take time?)
  s3conn.Bucket('amazon-reviews-pds').download_file(fileToStream2, 'tmp.gz')
  % ls /content/ -lah
  with gzip.open('/content/tmp.gz', 'rb') as f_in:
      big_list.append(f_in.readlines(10000000)) #Because each category has a n
umber of different bits I chose to read 10000000 and then cut the data to 1000
0 lines as required.

#2.For each dataset, read a batch of  10000  examples and  process them by the
functions constructed in Part 2.
lists= []
for j in range(len(big_list)):
  lists.append([i.decode('utf-8') for i in big_list[j]]) #use decode to remove
part of a string

#create the 10 datas
dfs = ["df0","df1","df2","df3","df4","df5","df6","df7","df8","df9"] #list of t
he datas names
d = {} #dict that will contain all 10 datas
for name,l in zip(dfs,lists):
    d[name] = pd.DataFrame([i.split('\t') for i in l],columns= l[0].split('\t'
)) #seperate lines
    d[name] = d[name][:10000] #take only the 10000 first rows
    d[name] = d[name].drop(labels=[0], axis=0) #remove first row (the col name
s)
    d[name] = df_function(d[name],"reviews_processed") #use the df_function
    lab = get_sentiment(d[name]["reviews_processed"]) #use the get_sentiment

    scores2 =[]
    values2 =[]
    for i in range(len(d[name]["reviews_processed"])): #add the two columns sc
ore and values
      scores2.append(round(lab[i][0].score,3))
      values2.append(lab[i][0].value)

    #add the to the data
    d[name]["sent_score"] = scores2
    d[name]["sent_value"] = values2

    #use the modifies_func
    d[name] = modifies_func(d[name])
```

In [19]:
```
#split to train/test randoly with an 80%|20% ratio as before.

#set the results
random.seed(100)

#create lists to contain the acuuracy and coef
accur_list_tra = []
accur_list_tes = []
coefficients_lists = []

#create dicts to contain all the train and test datas
trainin_dict_y = {}
trainin_dict_x = {}
testin_dict_y = {}
testin_dict_x = {}

#use the provided mapper function on the datas
for name in dfs:
  y = d[name]["binstar"]
  mapper_fit = mapper.fit(d[name])
  x = pd.DataFrame(mapper.transform(d[name]))# a numpy array

  #3.Run a logistic regression model on each category separately and report th
e train/test accuracies in a table.

  #create a x_train,x_test,y_train,y_test random split of the final_df and the
target binstar
  X_tra, X_tes, y_tra, y_tes = train_test_split(x , y, test_size = 0.20)

  #normalize the datas
  X_tes = Normalizer().fit_transform(X_tes)
  X_tra = Normalizer().fit_transform(X_tra)
  trainin_dict_x[name] = X_tra
  trainin_dict_y[name] = y_tra
  testin_dict_x[name] = X_tes
  testin_dict_y[name] = y_tes

  #Run a logistic regression and compute accuracies and coefficients
  reg_tra = LogisticRegression().fit(X_tra, y_tra)
  accur_list_tra.append(reg_tra.score(X_tra, y_tra))
  coefficients_lists.append(reg_tra.coef_)
  accur_list_tes.append(reg_tra.score(X_tes, y_tes))

#4.create data frame of accuracies
resi = pd.DataFrame(accur_list_tra, columns = ["Accuracy train"] )
resi["Accuracy test"] = accur_list_tes
resi["Data"] =["df_0","df_1","df_2","df_3","df_4","df_5","df_6","df_7","df_8",
"df_9"]
display(resi)

#5.average the fitted models coefficients from all categories to get a combine
d model.

#calculate the mean of each list of coeff in the big coeff list
coeff_sum2 = coefficients_lists[0]
for i in range(1,len(coefficients_lists)):
  coeff_sum2 += coefficients_lists[i]
coeff_mean2 = coeff_sum2/len(coefficients_lists)

#create accuracies lists for combine model
combined_acuur_train = []
combined_acuur_test = []

#build the model
for name in dfs:
  combined_model = e**(trainin_dict_x[name]@coeff_mean2[0])/ (1+e**(trainin_di
ct_x[name]@coeff_mean2[0]))
  #the y should be binary (LogisticRegression) so i built trashhold = 0.5, if
 logit_model>=  0.5 so 1 and  if logit_model>=  0.5 so 0
  for i in range(len(combined_model)):
    if combined_model[i] == 0.5:
      combined_model[i] = 1
    if combined_model[i] > 0.5:
      combined_model[i] = 1
    if combined_model[i] < 0.5:
      combined_model[i] = 0
  mse_combine = ((trainin_dict_y[name]-combined_model)**2).sum() /len(trainin_
dict_x[name]) #compute mse like the formola
  combined_acuur_train.append(round(1-mse_combine,3)) #compute acurr

  #same for test
  combined_model_test = e**(testin_dict_x[name]@coeff_mean2[0])/ (1+e**(testin
_dict_x[name]@coeff_mean2[0]))
  #the y should be binary (LogisticRegression) so i built trashhold = 0.5, if
 logit_model>=  0.5 so 1 and  if logit_model>=  0.5 so 0
  for i in range(len(combined_model_test)):
    if combined_model_test[i] == 0.5:
      combined_model_test[i] = 1
    if combined_model_test[i] > 0.5:
      combined_model_test[i] = 1
    if combined_model_test[i] < 0.5:
      combined_model_test[i] = 0
  mse_combine_test = ((testin_dict_y[name]-combined_model_test)**2).sum() /len
(testin_dict_x[name])
  combined_acuur_test.append(round(1-mse_combine_test,3))

#7.create data frame of the train and test accuracy for the combined model on
 the training and test set of each category separately.
resi_combine = pd.DataFrame(combined_acuur_train, columns = ["Accuracy combine
d train"] )
resi_combine["Accuracy combined test"] = combined_acuur_test
resi_combine["Data"] =["df_0","df_1","df_2","df_3","df_4","df_5","df_6","df_7"
,"df_8","df_9"]
display(resi_combine)
```

	Accuracy train	Accuracy test	Data
0	0.934795	0.921720	df_0
1	0.973990	0.968000	df_1
2	0.862329	0.861446	df_2
3	0.937569	0.933738	df_3
4	0.902727	0.901666	df_4
5	0.907556	0.894483	df_5
6	0.907891	0.886201	df_6
7	0.961360	0.968573	df_7
8	0.913072	0.920913	df_8
9	0.887932	0.884346	df_9

	Accuracy combined train	Accuracy combined test	Data
0	0.863	0.851	df_0
1	0.934	0.942	df_1
2	0.831	0.840	df_2
3	0.910	0.916	df_3
4	0.869	0.869	df_4
5	0.878	0.867	df_5
6	0.862	0.866	df_6
7	0.930	0.939	df_7
8	0.882	0.900	df_8
9	0.830	0.819	df_9

In this question we ran a logistic regression model on 10 categories in the Amazon reviews dataset instead of just one category as in the previous sections. After reading the data I built a dictionary which contain all the datas. I activated all the functions I built in the second part on the 10 datas to adjust the data to run a logistic regression.

After the early preparation I performed a logistical regression on each of the 10 datas and calculated the accuracy of the train and the test (shown below in the first table).

Next I built a combined model of all 10 train datas. I did this by extracting the coefficients from all the regressions of all the data and built an overall model of all 10 data train by the familiar formula of a logistic regression. I calculated f(x) manually so I created a condition that would allow y to have values of only 1 or 0. For the combined model I calculated the average MSE risk and accuracy. I did the same process for the 10 test datas. Then I calculated the accuracy of the combine train and combine test (shown below in the second table).

Did adding ×10 more examples help in improving the accuracy compared to fitting each category separately? why do you think this happened?

It can be seen that in both tables the accuracy ranges between 0.8-0.9 for the train datas and for the test datas. Therefore I concluded that this method does not actually improve\reduce the quality of the model. I think this is happening because it is possible that each category is different from the other in terms of its characterization and therefore it is more difficult to predict ten different categories in the same model instead of making a model for each category (Or if the accuracy does not decrease, at least it does not improve). When we average accuracy on all categories we actually lose important mains that are in each of the datas so the accuracy probably does not improve.

Part 4: Fitting steaming data using Stochastic Gradient Descent

While we were able to create a predictive model, we used a small subset of the entire data. Using Colab's available resources, it is impossible to run the model on the entire data. To get around loading the entire data in memory at once, we can use `stochastic gradient descent` to train the model a `(mini) batch` at a time.

4.(a) [3 pt] Create an `SGDClassifier` object with the logisitc regression loss. You may choose parameters for learning rate (the step-size at each iteration), penalties etc.
See here (https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html#sklearn.linear_model.SGDClassifier.) for more details. If you don't understand a parameter, just keep the default value.

Solution:

In [20]:
```
GSDlogred = SGDClassifier(loss='log', random_state= 4000, shuffle=False) # add additional parameters to object
```

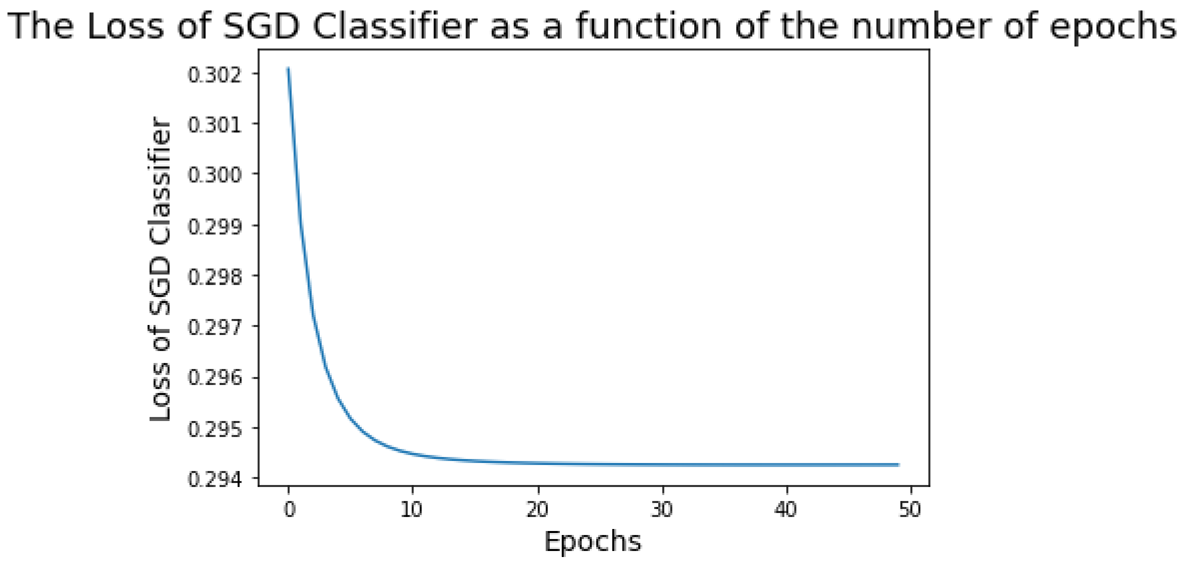
After reading about the SGDClassifier algorithm I chose to add the parameter shuffle and the parameter random_state= 4000 becuase this parameter actually causes the algorithm to save its results in each run and therefor learn, deepen and improve its accuracy in each run.

4.(b) [6 pt]

- For the same `train_set` fitted in **3.(a), 3.(b)**, apply the SGD classifier for 50 epochs (passes over the entire data) using the `partial_fit` method of the object `GSDlogred` you have created in **4.(a)**.
- Plot the loss of the classifier as a function of the number of epochs. Does it seem to converge?
- Compute the test error of the final output classifier. How does it compare to the error in **3.(b)** ?

Solution:

In [21]:
```
#same train and test as 3a
x_train_GSD = X_train
x_test_GSD = X_test
y_train_GSD = y_train
y_test_GSD = y_test

#According to what I read in the SGDClassifier algorithm the target variable i
s 1 or -1 so I changed the 0 to -1.
y_modi_GSD = y_train_GSD.replace(0,-1) #for y train
y_test_GSD_modi = y_test_GSD.replace(0,-1) #for y test

loss = []
sgd_loss = [] #the mean loss list
#apply the SGD classifier for  50  epochs  using the partial_fit method
for n in range(50):
  GSD_model = GSDlogred.partial_fit(x_train_GSD,y_train_GSD,classes=np.unique(
y_train_GSD)) #apply the partial_fit
  w = GSD_model.coef_ #compute the weight
  f_x = np.dot(w,x_train_GSD.T) + GSD_model.intercept_[0] #compute the f(x)
  vec = np.multiply(f_x[0],y_modi_GSD) #multiply the matrixs
  loss = [] #cumpute the loss
  for i in vec:
    loss.append(math.log(1+math.exp(-i)))  #the loss according the formula
  mean_loss = np.mean(loss) #mean on all the losses on one run
  sgd_loss.append(mean_loss) #append to the average list of losses

# plot loss of the classifier as a function of the number of epochs
fig,ax = plt.subplots()
#  a plot
ax.plot(sgd_loss)
# set labels
ax.set_xlabel("Epochs",fontsize=14)
ax.set_ylabel("Loss of SGD Classifier",fontsize=14)
ax.set_title('The Loss of SGD Classifier as a function of the number of epoch
s',fontsize=18)
plt.show()

#Compute the test error of the final output classifier.
error_GSD = (1-GSD_model.score(x_test_GSD,y_test_GSD))
print("The test error for the  final output SGD Classifier: ", round(error_GSD
,3))
print("The accuracy for the  final output SGD Classifier: ", round(GSD_model.s
core(x_test_GSD,y_test_GSD),3))
```



```
The test error for the  final output SGD Classifier:  0.122
The accuracy for the  final output SGD Classifier:  0.878
```

In this section I ran on the same train and test from 3a Q the SGD classifier for 50 epochs using the partial_fit method. In each run I calculated the weights by subtracting the coefficients from the model, the f(x) using the defined formula and the y[i]f(x[i]). Next I calculated the loss of SGD classifier model using the defined formula. The purpose of this algorithm is to study and deepen the model so in each iteration it gets better and its loss is smaller. Indeed we can see in the graph that the loss is converging in each iteration.

Does it seem to converge?

It can be clearly seen that the loss is converging. This makes sense because the idea of the model is that it learns with each re-run so after 50 iteration it is expected that the loss will be greatly reduced.

How the model accuracy compare to the error in 3.(b) ?

Compared to the accuracy index obtained in question 3B, it seems that there is a minimal difference in favor of the result of 3B when the accuracy index of the SGD classifier model is slightly lower.

4.(c) [6 pt] Pick one of the six *largest* categories in the AWS dataset.

Modify and use the `obj` defined in the cell below, to `stream` (readlines) and loop through the gzip file of this category `100000` bytes at a time, with at least `100` batches (all done inside a `with` connection to the gzip file):

- Make sure each `batch` is parsed appropriately (as have been done already)
- Create a `pandas` dataframe for each `batch` (within each loop) and print the number of rows (data points) in the `batch`
- Update the model parameters by executing the `GSDlogred.partial_fit` method on each processed batch (apply all the preprocessing steps we did in **Part 2** to get a processed numpy array, and the splitting to train and test sets before training the model as in **Part 3**).
  For each batch report the test accuracy for this batch, and also the *average* test accuracy over all batches so far. Do you see an improvement as you use more and more batches?

Note: Make sure that your fitting algorithm uses each data point in the batch only once, as is appropriate for the streaming model. Also make sure that when updating the parameters for each batch, the classifier is initialized with the parameters fitted already using previous batches. Modify the `GSDlogred` object if needed, before looping over batches, to accomodate these changes and other changes needed to deal with the data stream.

Solution:

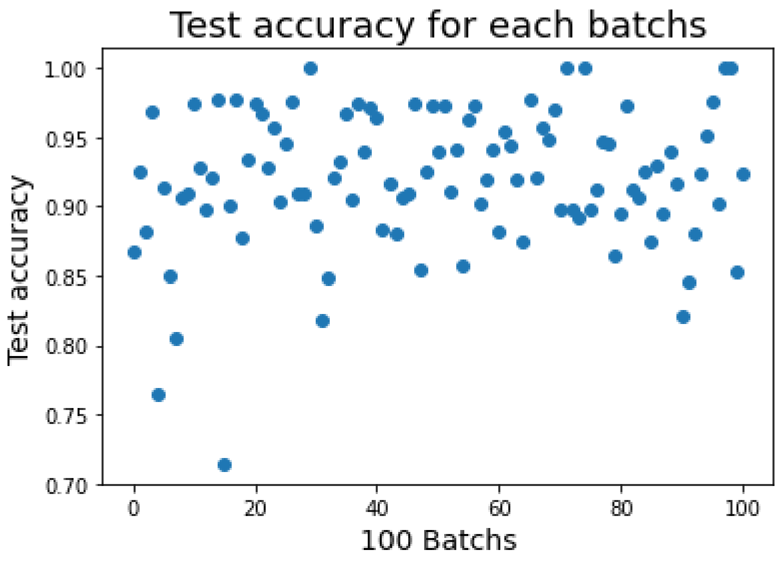
In [22]:
```
%%capture
maxCount = 100
batch_size = 100000 #bytes

#Modify the obj
obj = s3conn.Object('amazon-reviews-pds', filter_keys[4][0])
d2= {} #dict for all the batch
counter = 0 #create counter
with gzip.GzipFile(fileobj=obj.get()["Body"] ) as gzipfile:
    while counter <= maxCount:
        batch = [i.decode().replace('""','"').strip().split('\t') for i in gzi
pfile.readlines(batch_size)]

        #define the name  and col names of each data batch
        name = "df" + str(counter)
        d2[name] = pd.DataFrame(batch,columns = ['marketplace', 'customer_id',
'review_id', 'product_id',
       'product_parent', 'product_title', 'product_category', 'star_rating',
       'helpful_votes', 'total_votes', 'vine', 'verified_purchase',
       'review_headline', 'review_body', 'review_date'])
        d2[name] = d2[name].drop(labels=[0], axis=0)

        #print the number of rows
        print(d2[name].shape[0])

        #activate all the three function i created on section 2 for preprossec
ing
        d2[name] = df_function(d2[name],"reviews_processed")
        lab2 = get_sentiment(d2[name]["reviews_processed"])
        scores3 =[]
        values3 =[]
        for i in range(len(d2[name]["reviews_processed"])):
          scores3.append(round(lab2[i][0].score,3))
          values3.append(lab2[i][0].value)
        d2[name]["sent_score"] = scores3
        d2[name]["sent_value"] = values3
        d2[name] = modifies_func(d2[name])

        #update the counter
        counter += 1
```

In this section I selected the 4th category in the AWS dataset that its size is 2740337188 [Books_01]. I Modify and use the obj to stream and loop through the gzip file of this category 100000 bytes at a time, with 100 batches. I got 100 batches and for each of them, I split the rows again, turned them into data frames, changed their column names and so on. I have created a dictionary that will contain all the datas so that it will be easy to access. In addition I ran on each batch the three functions I built in section 2 to fit them to the classification model and printed the number of lines in each batch as required. I added a counter variable that will update the loop so that in each iteration it will move to the next batch.

In [23]:
```
#executing the GSDlogred.partial_fit method on each processed batch

#set the results
random.seed(100)

#I restart the SGDClassifier from the previous section.
GSDlogred2 = SGDClassifier(loss='log', random_state= 4000, shuffle=False)

#create empty lists of accur and average accur
avg_accu = []
accur = []

#run on all batchs
for name in d2.keys():
  y2 = d2[name]["binstar"]  #define the y
  mapper_fit = mapper.fit(d2[name]) #use the provided mapper function
  x2 = pd.DataFrame(mapper.transform(d2[name]))

  #create a x_train,x_test,y_train,y_test random split of the final_df and the
target binstar for each batch
  X_tra2, X_tes2, y_tra2, y_tes2 = train_test_split(x2 , y2, test_size = 0.20)

  #normalize the data  for each batch
  X_tes2 = Normalizer().fit_transform(X_tes2)
  X_tra2 = Normalizer().fit_transform(X_tra2)

  #preforme the GSDlogred.partial_fit methon on each batch
  GSD_model2 = GSDlogred2.partial_fit(X_tra2,y_tra2,classes=np.unique(y_tra2))
  accur.append(GSD_model2.score(X_tes2,y_tes2)) #compute the accur
  avg_accu.append(sum(accur)/len(accur)) #compute the average accure of all ac
cure until now

accur_data = pd.DataFrame(accur, index= d2.keys(),columns = ["Average test acc
uracy"] )
display(round(accur_data.transpose(),3)) #show results

#show the results Average test accuracy in plot
fig,ax = plt.subplots()
plt.scatter(range(101),accur)
ax.set_xlabel("100 Batchs",fontsize=14)
ax.set_ylabel("Test accuracy",fontsize=14)
ax.set_title('Test accuracy for each batchs',fontsize=18)
plt.show()
```

	df0	df1	df2	df3	df4	df5	df6	df7	df8	df9	df10	df11	df1
Average test accuracy	0.867	0.925	0.882	0.969	0.765	0.914	0.85	0.805	0.907	0.909	0.975	0.929	0.89

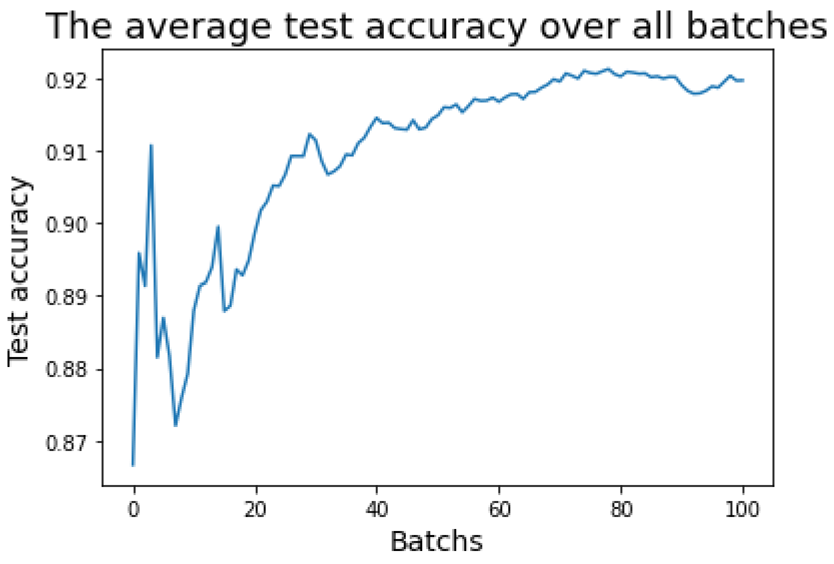
1 rows × 101 columns



I ran the SGDClassifier classification model on all batches. I first initialized the container (so that it would not remember what it learned in the previous section). I went through a loop on all the batches (data) and ran the mapping function provided to us. I divided the batchs into train and test, normalized and ran the partial_fit method. Its accuracy and I also calculated in each iteration the average accuracy index of all the models that have been so far.

I presented the accuracy indices of all the models in the list and also in the graph. It can be seen that the values of the accuracy indices are relatively uniformly distributed throughout the batchs And concentrate mainly on values between 0.8 -1 with no visible trend.

In [24]:
```
#I choose to show the average accuracy  over all batches so far in plot
fig,ax = plt.subplots()
plt.plot(avg_accu)
ax.set_xlabel("Batchs",fontsize=14)
ax.set_ylabel("Test accuracy",fontsize=14)
ax.set_title('The average test accuracy over all batches',fontsize=18)
plt.show()
```



Do you see an improvement as you use more and more batches?

I chose to report the average test accuracy over all batches **so far** Using a graph because I think that way we can see the results most clearly. When we see the graph showing the average test accuracy over all batches **so far** we can clearly see that the accuracy improves on average as the amount of batchs increases. At the beginning there are some jumps but an increase as trend but after 40 batch we can se that the average index converges to 0.92 area. That is, we can learn that although we can not see improvement when looking at the accuracy index of each batch individually, it is possible to conclude that the model improves on average its accuracy the more times it is operated (meaning the more batchs we run it on).

4.(d) [6 pt] Open Question How can you improve the classifier? Suggest and implement a way to train a classifier such that the test error on the category you picked in question **1.(d)** will improve compared to all previous results for this test set.

You can be creative: use additional data, fit additional models (e.g. nonlinear), change the training algorithm, etc.

Report the running time and the test accuracy of your algorithm

Solution:

In [27]:
```
#first implement
from sklearn.ensemble import RandomForestClassifier

s = time.time() #start time

#Create a RandomForest Classifier unnormelized train and test
clf=RandomForestClassifier(n_estimators=400).fit(X_train_not_norm, y_train_not
_norm)

#compute accuracy of train and test
accuracy_forest_train = clf.score(X_train_not_norm, y_train_not_norm)
accuracy_forest_test = clf.score(X_test_not_norm, y_test_not_norm)

e = time.time() #end time
t = e-s

print("Random Forest Classifier model - train accuracy:", round(accuracy_fores
t_train,3))
print("Random Forest Classifier model - test accuracy:", round(accuracy_forest
_test,3))
print("The running time is: ", round(t,3))
```

```
Random Forest Classifier model - train accuracy: 0.997
Random Forest Classifier model - test accuracy: 0.908
The running time is:  181.709
```

In [28]:
```
#second implement
from sklearn.ensemble import GradientBoostingClassifier

s2 = time.time()  #start time

#Create a Gradient Boosting Classifier use unnormelized train and test
clf2=GradientBoostingClassifier(n_estimators=100).fit(X_train_not_norm, y_trai
n_not_norm)

#compute accuracy of train and test
accuracy_forest_Boosting_train = clf2.score(X_train_not_norm, y_train_not_norm
)
accuracy_forest_Boosting_test = clf2.score(X_test_not_norm, y_test_not_norm)

e2 = time.time()  #end time
t2 = e2 - s2

print("Gradient Boosting Classifier model - train accuracy:", round(accuracy_f
orest_Boosting_train,3))
print("Gradient Boosting Classifier model - test accuracy:", round(accuracy_fo
rest_Boosting_test,3))
print("The running time is: ", round(t2,3))
```

```
Gradient Boosting Classifier model - train accuracy: 0.903
Gradient Boosting Classifier model - test accuracy: 0.901
The running time is:  88.802
```

In order to improve the Classifier I have tried several types of classifications to find the best one. For example, regression RIDGE, LASSO, KNN and etc. I decided to chose two algorithms: Random Forest Classifier and Gradient Boosting Classifier. The reason I chose these two algorithms is that it seems that despite all the changes we made during the various sections on the data, we did not seem to be able to improve the accuracy of the model significantly (and even in some cases actually lower the accuracy index). So I decided to choose another family of classification algorithms to try to improve the model.

Random forest is an ensemble learning method for classification or regression. The algorithem construct a multitude of decision trees at training time. For classification tasks, the output of the random forest is the class selected by most trees.

Gradient boosting is a machine learning technique for regression or classification, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. It builds the model in a stage-wise fashion, and it generalizes them by allowing optimization of an arbitrary differentiable loss function.

Both algorithms are based on division into decision trees where each has its advantages and disadvantages. I chose to compare these two algorithms because they have similar characteristics and because both yielded the best Classification results for the data. I worked with both algorithms in a similar way. Since the Random Forest and the Gradient boosting algorithms does not based on distances I thought there was no need to normalize the data (to check myself, I tried to normalize it and the result did not improve).

At first I chose to create 100 trees in Random forest algorithms but I got too big difference between the train accuracy index and the test accuracy index But his running time was very short (about 20-30). Following these results I realized that it is necessary to build more trees to evoid over-fitting and to improve the result. I decided to make some attempts of choosing more trees (to try to reduce the overfiting) and finally chose 400 trees for Random forest (This number of trees improved the result and reduced a bit the over-fitting). But when I added more trees it can be seen that the runtime increased much, even more than the runtime of the second algorithm. Conversely when I selected 100 trees of the Gradient boosting algorithm, there did not appear to be over-fitting according to the results at all (as you can see in the train results). And also run time is shorter than in the Random Forest algorithm.

In conclusion, each of the methods has advantages and disadvantages and both of them manages to raise the accuracy index not bad.
But in this case it can be seen that Gradient boosting algorithm Gives better results, both in terms of runtime and for over-fitting and still manages to improve the accuracy. Conversely the Random forest algorithms still does over fitting and its runing time is higher, therefor I choose Gradient boosting classifier to improve the classifier.