|  |  |  |
| --- | --- | --- |
| A logo on a black background  Description automatically generated | | |
| **Faculty of Science and Technology**  **CISC3014 – Information Retrieval and Web Search** | | |
| **Project Title: Evaluation on Movie Classification of Popular list from Rotten Tomatoes** | | |
| **Group Members:** | | |
| Huang Yanzhen | | DC126732 |
|  | Chen Zirui | DC127901 |

Table of Contents

[1. Introduction 3](#_Toc150014606)

[2. Crawler Part 3](#_Toc150014607)

[2.1 Analyze the page structure of the Rotten Tomatoes’ website. 3](#_Toc150014608)

[2.1.1 Movie Title and Stream Time 4](#_Toc150014609)

[2.1.2 Scores and Sentiment from Audience and Critics 4](#_Toc150014610)

[2.2 Store the analyzed data into an Excel file. 5](#_Toc150014611)

[2.3 Code Preview 7](#_Toc150014612)

[3. Evaluation Algorithms by Information Gain (Code part) 8](#_Toc150014613)

[3.1 Define Utilities Functions 9](#_Toc150014614)

[3.1.1 Define *count\_within\_condition()* Function 9](#_Toc150014615)

[3.1.2 Define *get\_entropy()* Function 10](#_Toc150014616)

[3.1.2 Define *console\_log ()* Function 10](#_Toc150014617)

[3.2 Define binary\_analysis() Function 11](#_Toc150014618)

[3.3 Define numeric\_analysis() Function 14](#_Toc150014619)

[3.4 Rank Information Gain 17](#_Toc150014620)

[3.4.1 Hybrid Analysis 17](#_Toc150014621)

[3.4.2 Extract data 17](#_Toc150014622)

[4. Results and Conclusion 19](#_Toc150014623)

[4.1 Critics Score 19](#_Toc150014624)

[4.2 Audience Score 19](#_Toc150014625)

[4.3 Audience Sentiment 19](#_Toc150014626)

# 1. Introduction

Rotten tomatoes is a review-aggregation website for film and television in the U.S. The site is influential among U.S. moviegoers, of whom a third states that they’d consult it before going to the cinema.

Being an individual movie-ranking website, Rotten tomatoes has its own special ranking system called “Tomato meter Rankings”, where evaluation of a movie varies from three tiers: “Certified Fresh”, “Fresh” and “Rotten”. The feedback-giving individuals are divided into two kinds: Audience and Critics. The ratings of the two kinds are separated.

This project is going to evaluate the relationships between the score (numerically from 0 to 100) and sentiment (binary, being positive or negative) given the Audience and Critics. To be specific, we are attempting to evaluate the classification of movies in Rotten Tomatoes’ top 100+ list into critics-positive and critics-negative by testing the impurity of classification on these three classifiers: audience sentiment (binary), audience score (numeric) and critics score (numeric).

This project includes: Crawler, which helps us get tons of scoring data; and Classification, which gives great evaluation of the classification on given conditions.

# 2. Crawler Part

To begin with, we wrote a Python Scrapy crawler to crawl information from Rotten Tomatoes’ top list.

## 2.1 Analyze the page structure of the Rotten Tomatoes’ website.

The collection of movie items is represented by a list of *<div>* container.

A screenshot of a computer

Description automatically generated

Hence, to get all movies in the page, we just simply finds every *<div>* item whose class is “flex-container”, storing it in the variable *movie\_list*. For each *<div>* item, there are a few essential data we need to obtain.

A screenshot of a computer program

Description automatically generated

### 2.1.1 Movie Title and Stream Time

Movie Title: Title is stored in a *<span>* with class *“p-small”*. Since this *<span>* was under the *<div>* movie item, we could easily extract it by xpath.

The title we obtain infact has some spaces and enters, so we want to omit them using Python’s regular expression tools.

Mean while, we may get some false elements, whose extracted title (actually along with every single attribute we retrieves) is “None”. Hence we added a guardian method to filter out these useless information.

|  |
| --- |
| ***\_\_get\_movies\_\_.py*** |
| movie\_title = score\_container.xpath(".//span[@class='p--small']/text()").get() # Guardian method. Don't get useless message if movie\_title is None:  continue movie\_title = re.sub(r"\n|\s","", movie\_title) |

Stream Time: Stream time is at the same level as movie title. It is contained in another span with class “smaller”. To make it clear, besides enters and spaces, we omitted the “Streaming” string in the streaming section to make it more clear.

|  |
| --- |
| ***\_\_get\_movies\_\_.py*** |
| # Stream time stream\_time = score\_container.xpath(".//span[@class='smaller']/text()").get() stream\_time = re.sub(r"\n|\s|Streaming", "", stream\_time) |

Although movie title and stream time is not as essential as the four scores to be discussed below, it really helps to identify different movie sections, which helps us to make sure that there is no false information.

### 2.1.2 Scores and Sentiment from Audience and Critics

It’s obvious that the four scores lies in the *<score-pairs-deprecated>* tag item. To be clear, this tag represents the score pair bar UI on the card view of a movie.

*A screenshot of a movie magazine

Description automatically generated*

The tag has four attributes: *audiencesentiment, audiencescore, criticssentiment, criticsscore,* which are exactly what we want for our analysis. We first extract the *<score-pairs-deprecated>* item as a root, and then get its attribute one-by-one:

|  |
| --- |
| ***\_\_get\_movies\_\_.py*** |
| # Overall score container evaluation\_scores = score\_container.xpath(".//score-pairs-deprecated") # Four important scores aud\_sentiment = evaluation\_scores.xpath("@audiencesentiment").get() aud\_score = evaluation\_scores.xpath("@audiencescore").get() critics\_sentiment = evaluation\_scores.xpath("@criticssentiment").get() critics\_score = evaluation\_scores.xpath("@criticsscore").get() |

## 2.2 Store the analyzed data into an Excel file.

Remark that, in the parse function, after extracting the list of movie items *movie\_list*, we use a for-loop to extract data sets. Besides extracting data, we do extra work at each iteration: The crawler process adds a new dataframe into the excel file. At the end of the loop, the target excel file is the full dataset to be analyzed. The storing function is a function we wrote in *\_\_save\_data\_\_.py*, whose explaining would be put in the attachment, since it’s not in the scope of discussion of this project.

|  |
| --- |
| ***\_\_get\_movies\_\_.py*** |
| # Processing data # Change negative/positive to binary expressions, which is a lot easier to cope with bin\_aud\_sentiment = 0 if aud\_sentiment == 'negative' else 1 bin\_critics\_sentiment = 0 if critics\_sentiment == 'negative' else 1 data = {  "title": movie\_title,  "stream\_time": stream\_time,  'link': score\_link,  "audience score": aud\_score,  "critics score": critics\_score,  "audience sentiment": bin\_aud\_sentiment,  "critics sentiment": bin\_critics\_sentiment, }  # Store the data in a new Excel file self.start\_urls.append("https://www.rottentomatoes.com/" + data['link']) if self.custom\_settings['SAVE\_DATA']:  \_\_save\_data\_\_.save\_data\_to\_excel(data) |

Additionally, we don’t want to save any data while testing the crawler. So we added a switch in the *custom\_settings* data set at the head of the *parse()* function. When *“SAVE\_DATA”* option is set *False*, we don’t save any data, otherwise there’d be thousands of copies in the data sheets.

|  |
| --- |
| ***\_\_get\_movies\_\_.py*** |
| custom\_settings = {  # Terminal settings: Ignore some warnings and logs  'LOG\_LEVEL': logging.CRITICAL if is\_terminal\_beautiful is True else logging.ERROR,  'REQUEST\_FINGERPRINTER\_IMPLEMENTATION': '2.7',   # Local settings  'LANG': 'EN',  'USE\_EXACT\_VALUES': True,  **'SAVE\_DATA': False,**   'ROUND\_LEVEL': 3, } |

Below is the resulting excel file (externally formatted for further processing, data captured at Oct 31, 2023, 14:09).

A screenshot of a computer

Description automatically generated

There are two rows of bad data, losing some of the attributes. We removed them.

## 2.3 Code Preview

Below is the full code of implementing the crawler.

|  |
| --- |
| ***\_\_get\_movies\_\_.py*** |
| import scrapy import re import logging import pandas as pd from scrapy.crawler import CrawlerProcess from scrapy.signalmanager import dispatcher from scrapy import signals  # local imports import \_\_save\_data\_\_ import \_\_urls\_\_ import \_\_get\_movie\_articles\_\_  class RTMovieCrawler(scrapy.Spider):  name = 'Rotten Tomatoes'  allowed\_domains = ['rottentomatoes.com']  start\_urls = ["https://www.rottentomatoes.com/browse/movies\_at\_home/sort:popular?page=21"]  is\_terminal\_beautiful = True  custom\_settings = {  # Terminal settings: Ignore some warnings and logs  'LOG\_LEVEL': logging.CRITICAL if is\_terminal\_beautiful is True else logging.ERROR,  'REQUEST\_FINGERPRINTER\_IMPLEMENTATION': '2.7',   # Local settings  'LANG': 'EN',  'USE\_EXACT\_VALUES': True,  'SAVE\_DATA': False, # Don't alter this!!!   'ROUND\_LEVEL': 3,  }   # These aren't really useful yet...  movie\_data = []  article\_start\_urls = []  index = 1   def parse(self, response):  # Entire movie grid  movie\_list = response.xpath("//div[@class='flex-container']")  for movie in movie\_list:  # Father element: Overall container  score\_container = movie.xpath(".//a[@data-track='scores']")  score\_link = score\_container.xpath("@href").get()   # Title  movie\_title = score\_container.xpath(".//span[@class='p--small']/text()").get()  # Guardian method. Don't get useless message  if movie\_title is None:  continue  movie\_title = re.sub(r"\n|\s","", movie\_title)    # Stream time  stream\_time = score\_container.xpath(".//span[@class='smaller']/text()").get()  stream\_time = re.sub(r"\n|\s|Streaming", "", stream\_time)   # Overall score container  evaluation\_scores = score\_container.xpath(".//score-pairs-deprecated")  # Four important scores  aud\_sentiment = evaluation\_scores.xpath("@audiencesentiment").get()  aud\_score = evaluation\_scores.xpath("@audiencescore").get()  critics\_sentiment = evaluation\_scores.xpath("@criticssentiment").get()  critics\_score = evaluation\_scores.xpath("@criticsscore").get()   # Processing data  # Change negative/positive to binary expressions   bin\_aud\_sentiment = 0 if aud\_sentiment == 'negative' else 1  bin\_critics\_sentiment = 0 if critics\_sentiment == 'negative' else 1  data = {  "title": movie\_title,  "stream\_time": stream\_time,  'link': score\_link,  "audience score": aud\_score,  "critics score": critics\_score,  "audience sentiment": bin\_aud\_sentiment,  "critics sentiment": bin\_critics\_sentiment,  }   # Store the data in a new Excel file  self.start\_urls.append("https://www.rottentomatoes.com/" + data['link'])  if self.custom\_settings['SAVE\_DATA']:  \_\_save\_data\_\_.save\_data\_to\_excel(data)   # Print beautiful data in console :)  print("--------------------")  print(  str(self.index) + '\n' +  "title: " + movie\_title + '\n' +  "stream time: " + stream\_time + '\n' +  "link:" + score\_link + '\n' +  "audience sentiment: " + aud\_sentiment + '\n' +  "audience score: " + aud\_score + '\n' +  "critics sentiment: " + critics\_sentiment + '\n' +  "critics score: " + critics\_score  )   # This thread is async so I have to do this to omit race conditions  self.index += 1   def run\_process():  process = CrawlerProcess()  process.crawl(RTMovieCrawler)  process.start()  run\_process() |

What’s more, we further formatted the output in the console window to inspect for false inforamtions. It may vary at different time:

|  |
| --- |
| Sample output of***\_\_get\_movies\_\_.py***  (Console Window, screenshot taken at Oct 31, 2023, 23:08) |
|  |

# 3. Evaluation Algorithms by Information Gain (Code part)

Due to the large amount of data, we would implement classification using Python rather than manual calculation. This process is done in \_*\_classify\_movies\_\_.py.* We decide to measure impurity of classification by Information Gain, which requires the calculation of Entropy at each classification condition.

First 10 rows of the extracted data:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **title** | **audience score** | **critics score** | **audience sentiment** | **critics sentiment** |
| FiveNightsatFreddy's | 88 | 26 | 1 | 0 |
| PainHustlers | 70 | 23 | 1 | 0 |
| NoHardFeelings | 87 | 71 | 1 | 1 |
| TheExorcist:Believer | 59 | 22 | 0 | 0 |
| TalktoMe | 82 | 94 | 1 | 1 |
| SawX | 89 | 79 | 1 | 1 |
| TheBurial | 83 | 91 | 1 | 1 |
| MilliVanilli | 85 | 100 | 1 | 1 |
| ThePigeonTunnel | 75 | 96 | 1 | 1 |
| WhenEvilLurks | 57 | 99 | 0 | 1 |
| … | … | … | … | … |

The classification target is *critics\_sentiment*. We’re using the first three attributes to evaluate the classification. This involves two types of analysis: Binary analysis (by audience sentiment) and Numeric analysis (audience score and critic score).

## 3.1 Define Utilities Functions

These utilities funcitons doesn’t influence the results of the information gain computing. Their role is just to make the entire algorithm looks more cleaner, improving readability and decreasing the chance of false calculations.

### 3.1.1 Define *count\_within\_condition()* Function

This function is used to count number of positive and negative targets within a settled conditional situation. A conditional situation, is an array of indexes which point to rows of that condition, having either negative or positive target (i.e. a “children” of classification using that condition).

For instance, for *audience\_sentiment = 0*, there are some records of *critical\_sentiment = 0* and *critical\_sentiment = 1*. The conditional situation array of *audience\_sentiment = 0* is the array of indexes of rows where *audience\_sentiment = 0*.

Since counting for different target within a conditional situation is quite generic, we wrote a specific function to perform it.

|  |
| --- |
| ***\_\_classify\_movies\_\_.py*** |
| # Count positive and negative samples within a conditional class def count\_within\_condition(cond\_class, target):  sample\_neg = 0  sample\_pos = 0  for target\_index in cond\_class:  if target[target\_index] == 0:  sample\_neg += 1  elif target[target\_index] == 1:  sample\_pos += 1  return sample\_neg, sample\_pos |

### 3.1.2 Define *get\_entropy()* Function

Since the evaluation process involves abundant calculations about entropy, a generic *get\_entropy()* function is also required. An entropy of a classification of a node is:

where *class1, class2, … , classn* are number of records classified into the corresponding target classes in a specific conditional situation *t*. Our case is simpler: The target class set is binary, having values only 0 and 1 (i.e. two classes). So there are only two input for this function: The numbers of class “0” and class “1” in the node’s record.

There is still one unexpected problem. Since ratio is required and there is a chance where one of the two input is zero (which denotes very low impurity), the division may yield infinity. So, we added a guard condition, letting result yield zero whenever a 0-division occurs.[[1]](#footnote-1)

Here is the *get\_entropy()* function. For a node with *a* class1 and *b* class2:

|  |
| --- |
| ***\_\_classify\_movies\_\_.py*** |
| def get\_entropy(a, b):  total = a + b  # Deal with divide by 0 problem  a\_ratio = a / total   b\_ratio = b / total  # Deal with divide by 0 problem  a\_log\_a = a\_ratio \* math.log10(a\_ratio) if a\_ratio != 0 else 0  b\_log\_b = b\_ratio \* math.log10(b\_ratio) if b\_ratio != 0 else 0   # Finally, calculate the entropy.....  entropy = (-1) \* (a\_log\_a + b\_log\_b)  return entropy |

### 3.1.2 Define *console\_log ()* Function

After computing entropy, a set of data (in the form of key-value pairs) is generated for analysis. For example, this data contains the *records* of this condition, i.e. number of records in each condition situations and their ratios, etc; and the *entropy* of each conditional situations, and so on.

|  |
| --- |
| *(data)* |
| data = {  "Name": cond\_name,  "Records": {  "Num of negative condition": num\_cond\_neg,  "Num of positive condition": num\_cond\_pos,  "Num of negative target": num\_targ\_neg,  "Num of positive target": num\_targ\_pos,  "Negative ratio": cond\_neg\_ratio,  "Positive ratio": cond\_pos\_ratio,  "Total record num": num\_total,  },  "Entropy": {  "Parent Entropy": e\_parent,  "Condition negative entropy": e\_neg,  "Condition positive entropy": e\_pos,  },  "Information Gain": {  "Info gain": info\_gain,  } } |

Therefore, due to abundant amount of data, we wrote a function to print data in a neat manner.

|  |
| --- |
| ***\_\_classify\_movies\_\_.py*** |
| def console\_log(data, indent=0):  # Treat "Name" tage specially  condition\_name = data.get("Name")  if condition\_name is not None:  print("\n\n>>>>>>>>>>>> " + condition\_name + " <<<<<<<<<<<<")  del data['Name']   # Traverse all the data items.  for key, val in data.items():  # Dictionary type, parse section  if isinstance(val, dict):  # Trick: make header constant length  # As data go deeper, the length of title is shorter  hyphen\_num = 45 - 10 \* indent  left\_hyphens = (hyphen\_num - len(key)) // 2  right\_hyphens = hyphen\_num - len(key) - left\_hyphens  title = "|-" + "-" \* left\_hyphens + key + "-" \* right\_hyphens + "-|"   # Section title  print(title)  console\_log(val, indent+1)  else:  # Section contents  print(key + ": " + str(val)) |

## 3.2 Define binary\_analysis() Function

Binary analysis helps to analyze the classification using binary conditions. Here, the condition *audience\_sentiment* is binary. Using python, we implement the classification process by:

*First,* seperate every “0”s and “1”s in the **classification** **condition** **(Not target! The values are from the *audience sentiment* column.)** by storing indexes pointing to them respectively into arrays *negative* and *positive.*

|  |
| --- |
| ***\_\_classify\_movies\_\_.py*** |
| # Categorize into 2 conditions: negative and positive id\_cond\_neg = [] # index of negative conditions id\_cond\_pos = [] # index of positive conditions  for index, row in enumerate(condition):  if row == 0:  id\_cond\_neg.append(index)  elif row == 1:  id\_cond\_pos.append(index)  else:  raise ValueError("Invalid binary value!") |

*Second,* using *count\_within\_condition()*, accumulate the rows yielding 0 or 1 in the **classification target** in each group. To be specific, in group *negative* (whose *audience sentiment* value is 0), there are some records having 0 in the *critical sentiment*, others having 1; and vise versa.

|  |
| --- |
| ***\_\_classify\_movies\_\_.py*** |
| # Negative and positive part for each conditions [num\_neg\_neg, num\_neg\_pos] = count\_within\_condition(id\_cond\_neg, target) [num\_pos\_neg, num\_pos\_pos] = count\_within\_condition(id\_cond\_pos, target) |

*Third,* calculate parent’s entropy and all the children’s entropy. For parent’s entropy, just record number of target negatives and positives, following the formula. The following process would be easy using the *get\_entropy()* function.

*Finally,* calculate information gain by:

simply subtract parent entropy by weighted sum of child entropies.

|  |
| --- |
| ***\_\_classify\_movies\_\_.py*** |
| # Compute entropy # Compute number of target neg & pos num\_targ\_neg = 0 num\_targ\_pos = 0 for target\_value in target:  if target\_value == 0:  num\_targ\_neg += 1  else:  num\_targ\_pos += 1  # Parent entropy num\_total = num\_targ\_neg + num\_targ\_pos # Total num of records e\_parent = get\_entropy(num\_targ\_neg, num\_targ\_pos)  # Child entropy # Number of instances in each condition num\_cond\_neg = len(id\_cond\_neg) num\_cond\_pos = len(id\_cond\_pos)  # Weights of each condition cond\_neg\_ratio = num\_cond\_neg / num\_total if num\_total != 0 else 0 # Target negative cond\_pos\_ratio = num\_cond\_pos / num\_total if num\_total != 0 else 0 # Target positive  # Entropy of each condition e\_neg = get\_entropy(num\_neg\_neg, num\_neg\_pos) # Condition negative e\_pos = get\_entropy(num\_pos\_neg, num\_pos\_pos) # Condition positive  # Information gain info\_gain = e\_parent - (  cond\_neg\_ratio \* e\_neg +  cond\_pos\_ratio \* e\_pos ) |

The larger the information gain, the purer the classification is.

Here is the full code of *binary\_analysis():*

|  |
| --- |
| ***\_\_classify\_movies\_\_.py*** |
| # Input binary condition and binary target, output information gain def binary\_analysis(condition, target, cond\_name="Binary Analysis"):   # Categorize into 2 conditions: negative and positive  id\_cond\_neg = [] # index of negative conditions  id\_cond\_pos = [] # index of positive conditions   for index, row in enumerate(condition):  if row == 0:  id\_cond\_neg.append(index)  elif row == 1:  id\_cond\_pos.append(index)  else:  raise ValueError("Invalid binary value!")   # Negative and positive part for each conditions  [num\_neg\_neg, num\_neg\_pos] = count\_within\_condition(id\_cond\_neg, target)  [num\_pos\_neg, num\_pos\_pos] = count\_within\_condition(id\_cond\_pos, target)   # Compute entropy  # Compute number of target neg & pos  num\_targ\_neg = 0  num\_targ\_pos = 0  for target\_value in target:  if target\_value == 0:  num\_targ\_neg += 1  else:  num\_targ\_pos += 1   # Parent entropy  num\_total = num\_targ\_neg + num\_targ\_pos # Total num of records  e\_parent = get\_entropy(num\_targ\_neg, num\_targ\_pos)   # Child entropy  # Number of instances in each condition  num\_cond\_neg = len(id\_cond\_neg)  num\_cond\_pos = len(id\_cond\_pos)   # Weights of each condition  cond\_neg\_ratio = num\_cond\_neg / num\_total if num\_total != 0 else 0   cond\_pos\_ratio = num\_cond\_pos / num\_total if num\_total != 0 else 0    # Entropy of each condition  e\_neg = get\_entropy(num\_neg\_neg, num\_neg\_pos) # Condition negative  e\_pos = get\_entropy(num\_pos\_neg, num\_pos\_pos) # Condition positive   # Information gain  info\_gain = e\_parent - (  cond\_neg\_ratio \* e\_neg +  cond\_pos\_ratio \* e\_pos  )   # Print in console  data = {  "Name": cond\_name,  "Records": {  "Num of negative condition": num\_cond\_neg,  "Num of positive condition": num\_cond\_pos,  "Num of negative target": num\_targ\_neg,  "Num of positive target": num\_targ\_pos,  "Negative ratio": cond\_neg\_ratio,  "Positive ratio": cond\_pos\_ratio,  "Total record num": num\_total,  },  "Entropy": {  "Parent Entropy": e\_parent,  "Condition negative entropy": e\_neg,  "Condition positive entropy": e\_pos,  },  "Information Gain": {  "Info gain": info\_gain,  }  }  console\_log(data)  return info\_gain |

## 3.3 Define numeric\_analysis() Function

Categorizing conditions in numeric analysis is more complicated than in binary analysis. We give out three methods to categorize numeric condition variables into different nodes, and use the method that gives the best information gain. The methods are:

*Binary means:* Divide data into two chunks using the mean value;

*Three splits:* Split data into three chunks using left and right thresholds T1 & T2:

*Small,* data that’s smaller than T1;

*Medium,* data that’s within T1 and T2;

*Large,* data that’s larger thatn T2;

*Combination:* Merge *small & medium* or *medium & large* from three splits into a single node, yielding a binary division.

The rest part is essentially the same. The only difference is that there are three conditional values rather than two. Similar to binary analysis, count negative and positive part of each conditions.

According to the Central Limit Theorem, the distribugtion of sample means approximates a normal distribution as the sample size gets larger (usually greater than 30), regardless of the population’s distribution. Sicne our sample size is 117, i.e. there are totally 117 records of condition values, it is safe to use a normal distribution approach to find the two thresholds.

|  |
| --- |
| ***\_\_classify\_movies\_\_.py*** |
| # Defines left mean and right mean mean\_val = condition.mean()  if method == "normal\_dist":  # Normal distribution  left\_thresh = mean\_val - degree \* condition.std()  right\_thresh = mean\_val + degree \* condition.std() else:  left\_thresh = (condition.min() + mean\_val) / 2  right\_thresh = (condition.max() + mean\_val) / 2 |

The rest part is essentially the same. Count target samples in each splits:

|  |
| --- |
| ***\_\_classify\_movies\_\_.py*** |
| # Categorizing # Categorize into two conditions: small and large id\_cond\_left = [] id\_cond\_right = [] # Categorize into 3 conditions: small, medium, and large id\_cond\_small = [] # condition small id\_cond\_medium = [] # condition medium id\_cond\_large = [] # condition large  for index, row in enumerate(condition):  # Divide by mean  if row < mean\_val:  id\_cond\_left.append(index)  id\_cond\_right.append(index)   # Divide by normal distribution  if row < left\_thresh:  id\_cond\_small.append(index)  elif row > right\_thresh:  id\_cond\_large.append(index)  else:  id\_cond\_medium.append(index) |

Finally, the entropy part is also roughly the same. Here is the full code of *numeric\_analysis():*

|  |
| --- |
| ***\_\_classify\_movies\_\_.py*** |
| def numeric\_analysis(condition, target, cond\_name="Numeric Analysis", method="normal\_dist", degree=0.4):  # Defines left mean and right mean  mean\_val = condition.mean()   if method == "normal\_dist":  # Normal distribution  left\_thresh = mean\_val - degree \* condition.std()  right\_thresh = mean\_val + degree \* condition.std()  else:  left\_thresh = (condition.min() + mean\_val) / 2  right\_thresh = (condition.max() + mean\_val) / 2   # Categorizing  # Categorize into two conditions: small and large  id\_cond\_left = []  id\_cond\_right = []  # Categorize into 3 conditions: small, medium, and large  id\_cond\_small = [] # condition small  id\_cond\_medium = [] # condition medium  id\_cond\_large = [] # condition large   for index, row in enumerate(condition):  # Divide by mean  if row < mean\_val:  id\_cond\_left.append(index)  id\_cond\_right.append(index)   # Divide by normal distribution  if row < left\_thresh:  id\_cond\_small.append(index)  elif row > right\_thresh:  id\_cond\_large.append(index)  else:  id\_cond\_medium.append(index)   # Statistics within categorizing  # Negative and positive part of each conditions  # Single node: Two divisions  [lf\_neg, lf\_pos] = count\_within\_condition(id\_cond\_left, target)  [rt\_neg, rt\_pos] = count\_within\_condition(id\_cond\_right, target)  # Single node: Three divisions  [s\_neg, s\_pos] = count\_within\_condition(id\_cond\_small, target)  [m\_neg, m\_pos] = count\_within\_condition(id\_cond\_medium, target)  [l\_neg, l\_pos] = count\_within\_condition(id\_cond\_large, target)  # Combined node  [sm\_neg, sm\_pos] = [s\_neg + m\_neg, s\_pos + m\_pos]  [ml\_neg, ml\_pos] = [m\_neg + l\_neg, m\_pos + l\_pos]   # Compute Entropy  # Compute number of target neg & pos  num\_targ\_neg = 0  num\_targ\_pos = 0  for target\_value in target:  if target\_value == 0:  num\_targ\_neg += 1  else:  num\_targ\_pos += 1   # Parent entropy  num\_total = num\_targ\_neg + num\_targ\_pos  e\_parent = get\_entropy(num\_targ\_neg, num\_targ\_pos)   # Child entropy  # Number of instances in each condition  # Single Nodes: Two divisions  num\_cond\_left = len(id\_cond\_left)  num\_cond\_right = len(id\_cond\_right)  # Single Nodes: Three divisions  num\_cond\_small = len(id\_cond\_small)  num\_cond\_medium = len(id\_cond\_medium)  num\_cond\_large = len(id\_cond\_large)  # Combined Nodes  num\_cond\_small\_medium = num\_cond\_small + num\_cond\_medium  num\_cond\_medium\_large = num\_cond\_medium + num\_cond\_large   # Entropy of each condition  # Single Nodes: Two divisions  e\_left = get\_entropy(lf\_neg, lf\_pos)  e\_right = get\_entropy(rt\_neg, rt\_pos)  # Single Nodes: Three divisions  e\_small = get\_entropy(s\_neg, s\_pos)  e\_medium = get\_entropy(m\_neg, m\_pos)  e\_large = get\_entropy(l\_neg, l\_pos)  # Combined Nodes  e\_small\_medium = get\_entropy(sm\_neg, sm\_pos)  e\_medium\_large = get\_entropy(ml\_neg, ml\_pos)   # Weights of each condition  # Single Nodes: Two divisions  cond\_left\_ratio = num\_cond\_left / num\_total if num\_total != 0 else 0  cond\_right\_ratio = num\_cond\_right / num\_total if num\_total != 0 else 0  # Single Nodes: Three divisions  cond\_small\_ratio = num\_cond\_small / num\_total if num\_total != 0 else 0  cond\_medium\_ratio = num\_cond\_medium / num\_total if num\_total != 0 else 0  cond\_large\_ratio = num\_cond\_large / num\_total if num\_total != 0 else 0  # Combined Nodes  cond\_small\_medium\_ratio = num\_cond\_small\_medium / num\_total if num\_total != 0 else 0  cond\_medium\_large\_ratio = num\_cond\_medium\_large / num\_total if num\_total != 0 else 0   # Information gain  # Binary mean  info\_gain\_use\_mean = e\_parent - (  cond\_left\_ratio \* e\_left +  cond\_right\_ratio \* e\_right  )  # Three splits  info\_gain\_three\_splits = e\_parent - (  cond\_small\_ratio \* e\_small +  cond\_medium\_ratio \* e\_medium +  cond\_large\_ratio \* e\_large  )  # Combined  info\_gain\_use\_small = e\_parent - (  cond\_small\_ratio \* e\_small +  cond\_medium\_large\_ratio \* e\_medium\_large  )  info\_gain\_use\_large = e\_parent - (  cond\_small\_medium\_ratio \* e\_small\_medium +  cond\_large\_ratio \* e\_large  )   # Print result  # Get the best info gain!  info\_gain\_candidates = {  'Mean': info\_gain\_use\_mean,  'Three split': info\_gain\_three\_splits,  'Left thresh': info\_gain\_use\_small,  'Right thresh': info\_gain\_use\_large  }   info\_gain = max(info\_gain\_candidates.values())  partition\_method = [variable for variable, value in info\_gain\_candidates.items() if value == info\_gain]  partition\_method = partition\_method[0]    # yield data  data = {  "Name": cond\_name,  "Records": {  "Num of condition small": num\_cond\_small,  "Num of condition medium": num\_cond\_medium,  "Num of condition large": num\_cond\_large,  "Num of negative target": num\_targ\_neg,  "Num of positive target": num\_targ\_pos,  "Num of small ratio": cond\_small\_ratio,  "Num of medium ratio": cond\_medium\_ratio,  "Num of large ratio": cond\_large\_ratio,  "Total record num": num\_total,  },  "Entropy": {  "Parent entropy": e\_parent,  "Binary Mean": {  "Entropy left": e\_left,  "Entropy right": e\_right,  },  "Three Split": {  "Entropy small": e\_small,  "Entropy medium": e\_medium,  "Entropy large": e\_large,  },  "Binary Norm Dist": {  "Entropy small & medium": e\_small\_medium,  "Entropy medium & large": e\_medium\_large,  }  },  "Information Gain": {  "Info gain": info\_gain,  "Uses partition": partition\_method,  }  }  console\_log(data)   return info\_gain |

## 3.4 Rank Information Gain

By ranking the information gain of all conditions, we could obtain the optimized classifying sequence.

### 3.4.1 Hybrid Analysis

We combine two methods into one using a binary value *is\_binary:*

|  |
| --- |
| ***\_\_classify\_movies\_\_.py*** |
| # Hybrid analysis, combining numeric and binary analysis. def hybrid\_analysis(condition, target, is\_binary=True, cond\_name="Analysis Name", method="normal\_dist", degree=0.4):  if is\_binary:  info\_gain = binary\_analysis(condition, target, cond\_name)  else:  info\_gain = numeric\_analysis(condition, target, cond\_name, method, degree)  return info\_gain |

### 3.4.2 Extract data

We first read the excel file generated by the crawler.

|  |
| --- |
| ***\_\_classify\_movies\_\_.py*** |
| # Read Excel file and do analysis filepath = './movie\_list/movie\_data.xlsx' movie\_data = pd.read\_excel(filepath) |

Then, by using the *pandas* library, we extract condition columns and target column into corresponding *Series* objects.

|  |
| --- |
| ***\_\_classify\_movies\_\_.py*** |
| # Conditions aud\_score = movie\_data['audience score'] critics\_score = movie\_data['critics score'] aud\_sentiment = movie\_data['audience sentiment'] # Target critics\_sentiment = movie\_data['critics sentiment'] |

After that, by calling the *hybrid\_analaysis()* function, taking the conditions and target as inputs, we get a list of information gain.

|  |
| --- |
| ***\_\_classify\_movies\_\_.py*** |
| # Get info gain from the three conditions ig\_aud\_sentiment = hybrid\_analysis(  aud\_sentiment,  critics\_sentiment,  True,  "Audience Sentiment" ) # Audience sentiment  ig\_aud\_score = hybrid\_analysis(  aud\_score,  critics\_sentiment,  False,  "Audience Score",  degree=0.4 ) # Audience score  ig\_critics\_score = hybrid\_analysis(  critics\_score,  critics\_sentiment,  False,  "Critics Score",  degree=0.5 ) # Critics score |

Lastly, we sort the three values and print the result. Notice that we also print detailed data in *binary\_analysis()* and *numeric\_analysis()* by the *console\_log()* function. Hence, we’ll get detailed results in the console window.

|  |
| --- |
| ***\_\_classify\_movies\_\_.py*** |
| # Sort the three conditions from high to low variables = [  ('Audience Sentiment', ig\_aud\_sentiment),  ('Audience Score', ig\_aud\_score),  ('Critics Score', ig\_critics\_score) ]  variables\_sorted = sorted(variables, key=lambda x: x[1], reverse=True)  # Print the three values: print("~~~~~~~~~~~ Info Gain Ranking ~~~~~~~~~~~") for i, (name, value) in enumerate(variables\_sorted):  print(f"{i+1}. Info Gain from {name}: {value}") |

# 4. Results and Conclusion

Here are the results printed in the console window:

|  |  |  |
| --- | --- | --- |
| Audience Sentiment | Audience Score | Critics Score |
|  |  |  |
| Ranking of Information Gain | | |
|  | | |

### 4.1 Critics Score

Critic Score is the purest, and the best partition method is three split. By inspecting the detailed data, we can see that the entropy of *small* and *large* part is zero, meaning that it is well-classified in the two parts. In deed, the information gain of entropy is the highest.

### 4.2 Audience Score

Audience Score is the second purest. However, it is still significantly smaller than the information gain from the *Critics Score.* We can see that the dividing condition of *small* and *medium* is roughly the same, while the entropy of *large* section is significantly lower (around 0.16 compared to around 0.29). This means that the *large* section has a better classification than the other two.

### 4.3 Audience Sentiment

Audience Sentiment has the lowest purity, yet it’s at the same metering level of *Audience Score*. The two condition situations (negative and positive) both has quite high an entropy, meaning the dividing is indeed not very ideal. This indicates that the audience sentiment doesn’t quite match with the critics one.

1. This may cause unexpected calculation errors. [↑](#footnote-ref-1)