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

Table of Contents

[1. Introduction 3](#_Toc149689682)

[2. Crawler 3](#_Toc149689683)

[2.1 Analyze the page structure of the Rotten Tomatoes’ website. 3](#_Toc149689684)

[2.1.1 Movie Title and Stream Time 4](#_Toc149689685)

[2.1.2 Scores and Sentiment from Audience and Critics 4](#_Toc149689686)

[2.2 Store the analyzed data into an Excel file. 5](#_Toc149689687)

[2.3 Code Preview 6](#_Toc149689688)

[3. Classification 8](#_Toc149689689)

[3.1.1 get\_entropy() Function 9](#_Toc149689690)

[3.1.2 Binary analysis 10](#_Toc149689691)

# 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

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) |

Moreover, 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 coming.

|  |
| --- |
| ***\_\_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

## 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!!! otherwise you'd create duplicate rows in the excel file  '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. Classification

Due to the large amount of data, we would implement classification using Python instead of 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).

## 3.1.1 get\_entropy() Function

Since this process involves abundant calculations about entropy, a generic *get\_entropy()* function is 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 children *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.

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

|  |
| --- |
| ***\_\_get\_movies\_\_.py*** |
| def get\_entropy(a, b):  total = a + b  # Deal with divide by 0 problem  a\_ratio = a / total if total != 0 else 0  b\_ratio = b / total if total != 0 else 0   # 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 Binary analysis

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*** |
| id\_cond\_neg = [] # negative conditions id\_cond\_pos = [] # positive conditions num\_targ\_neg = 0 # num of negative targets num\_targ\_pos = 0 # num of positive targets  # go through every row in the condition for index, row in enumerate(condition):  if row == 0:  id\_cond\_neg.append(index)  if row == 1:  id\_cond\_pos.append(index) |

*Second,* 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. At the mean time, accumulate number of 0 and 1s in the target for further calculation of parent entropy.

|  |
| --- |
| ***\_\_classify\_movies\_\_.py*** |
| # Focus on condition negative neg\_neg = 0 # target negative neg\_pos = 0 # target positive for index in id\_cond\_neg:  if target[index] == 0:  neg\_neg += 1  num\_targ\_neg += 1  if target[index] == 1:  neg\_pos += 1  num\_targ\_pos += 1   # Focus on condition positive pos\_neg = 0 # target negative pos\_pos = 0 # target positive for index in id\_cond\_pos:  if target[index] == 0:  neg\_neg += 1  num\_targ\_neg += 1  if target[index] == 1:  neg\_pos += 1  num\_targ\_pos += 1 |

*Third,* calculate parent’s entropy and all the children’s entropy. For parent’s entropy, just record number of condition 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 # Parent entropy num\_total = num\_targ\_neg + num\_targ\_pos # Total num of records neg\_ratio = num\_targ\_neg / num\_total if num\_total != 0 else 0 # Target negative pos\_ratio = num\_targ\_pos / num\_total if num\_total != 0 else 0 # Target positive e\_parent = get\_entropy(num\_targ\_neg, num\_targ\_pos)  # Child entropy e\_neg = get\_entropy(neg\_neg, neg\_pos) # Condition negative e\_pos = get\_entropy(pos\_neg, pos\_pos) # Condition positive  # Information gain info\_gain = e\_parent - (neg\_ratio \* e\_neg + 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"):  def get\_entropy(a, b):  total = a + b  # Deal with divide by 0 problem  a\_ratio = a / total if total != 0 else 0  b\_ratio = b / total if total != 0 else 0   # 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   id\_cond\_neg = [] # negative conditions  id\_cond\_pos = [] # positive conditions  num\_targ\_neg = 0 # num of negative targets  num\_targ\_pos = 0 # num of positive targets   # go through every row in the condition  for index, row in enumerate(condition):  if row == 0:  id\_cond\_neg.append(index)  if row == 1:  id\_cond\_pos.append(index)   # Focus on condition negative  neg\_neg = 0 # target negative  neg\_pos = 0 # target positive  for index in id\_cond\_neg:  if target[index] == 0:  neg\_neg += 1  num\_targ\_neg += 1  if target[index] == 1:  neg\_pos += 1  num\_targ\_pos += 1   # Focus on condition positive  pos\_neg = 0 # target negative  pos\_pos = 0 # target positive  for index in id\_cond\_pos:  if target[index] == 0:  neg\_neg += 1  num\_targ\_neg += 1  if target[index] == 1:  neg\_pos += 1  num\_targ\_pos += 1   # Compute entropy  # Parent entropy  num\_total = num\_targ\_neg + num\_targ\_pos # Total num of records  neg\_ratio = num\_targ\_neg / num\_total if num\_total != 0 else 0 # Target negative  pos\_ratio = num\_targ\_pos / num\_total if num\_total != 0 else 0 # Target positive  e\_parent = get\_entropy(num\_targ\_neg, num\_targ\_pos)   # Child entropy  e\_neg = get\_entropy(neg\_neg, neg\_pos) # Condition negative  e\_pos = get\_entropy(pos\_neg, pos\_pos) # Condition positive   # Information gain  info\_gain = e\_parent - (neg\_ratio \* e\_neg + pos\_ratio \* e\_pos) # Total information gain   # Print in console  print(  '\n\n' +  ">>>>>>>>>> " + cond\_name + " <<<<<<<<<<" + '\n' +  "|---------- Records ----------|"  )  print(  "Total num of condition '0's: " + str(num\_targ\_neg) + '\n' +  "Total num of condition '1's: " + str(num\_targ\_pos) + '\n' +  "Negative ratio: " + str(neg\_ratio) + '\n' +  "Positive ratio: " + str(pos\_ratio) + '\n' +  "Total num: " + str(num\_total)  )  print("|---------- Entropy ----------|")  wow\_eneg = " (wow!)" if e\_neg == 0 else ""  wow\_epos = " (wow!)" if e\_pos == 0 else ""  print(  "Entropy of parent: " + str(e\_parent) + '\n' +  "Entropy of class '0': " + str(e\_neg) + wow\_eneg + '\n' +  "Entropy of class '1': " + str(e\_pos) + wow\_epos  )  print("|--------- Info Gain ---------|")  wow = " (wow!)" if info\_gain == 0 else ""  print("Information gain: " + str(info\_gain) + wow)   return info\_gain |

## 3.1.3 Numeric Analysis

Numeric analysis seperates condition into three categories using two thresholds: T1 and T2 (T1<T2). The three categories in the condition are:

*Low,* whose score value is less than T1,

*Medium,* whose score is between T1 and T2, and

*Large,* whose score is larger than T2.

To make it simpler, we make T1 be the lower average (i.e. average of mean and minimum value among the condition scores) and T2 be the higher average (likewise).

|  |
| --- |
| ***\_\_classify\_movies\_\_.py*** |
| # Defines left mean and right mean mean\_val = condition.mean() left\_mean = (condition.min() + mean\_val) / 2 right\_mean = (condition.max() + mean\_val) / 2 |

The rest part is essentially the same. One difference is, since there are three conditional values,