Subanalysis: a visualization of dialogues in a TV series

Carlo Cabras[1] and Francesco Pisu[1]

[1]Department of Mathematics and Computer Science, University of Cagliari
carlocabras21@gmail.com, fra.pisu1@gmail.com

***Abstract*— The goal of this project was to develop an application for gathering data about dialogues in TV series and to display them through an interactive bar chart. The first step consisted of gathering a quantitative value, that we call *words per hour* (w/h), about how much dialogue is there in a TV series. After all these estimates had been gathered, a bar chart displaying TV series, seasons or single episodes along with the words per hour values on the y-axis has been created. Through a simple mechanism of switching between displayed data it is possible to appreciate w/h values even of single episodes and to easily perform compare and lookup tasks.**

I. Introduction

Most of us watch several TV series of different genres and it happens that we get so involved that we like wearing some cool t-shirts about them. While watching *Samurai Jack* you may notice they barely speak and most of the story development is made not from the dialogues but rather from the action. We asked ourselves "What about the other series?". Therefore, we thought of design an interactive tool to show how many words are uttered in each series.

A. The Project

The repository of this project can be found at the following Github link `https://github.com/francescopisu/subanalysis`.

II. Data extraction

A. The Words per Hour quantity

First of all, we needed something to visualize. We could have just counted how many words the characters say (and how this is achieved is explained in the next section) for each episode, of each season, of each series, and then printing it. However, the episodes have different duration, so one can expect that in a 60-minutes episode there will be more talking than in a 20-minutes episode. So we needed to normalize the word counting like every episode was a 60-minute one:

$$Words\ per\ Hour = \frac{words\ count}{episode\ duration} \times 60 \qquad (1)$$

The quantity above represents how many words are being told in 60 minutes. You can see from that simple formula that we could just have used the "Words per Minute" quantity, obtained by dividing the words by the episode duration. However, we decided the hour would be a more natural choice, since we use something like that in every-day life (e.g. car speed, km/h). Also, most of the episodes are closer to an hour duration than a minute duration.

B. The series subtitles

Using some software that listens the dialogues from an episode, then extracting the words so we can count them, would have been nearly impossible. Luckily, most of the series comes with their subtitles. Words can then be easily counted via subtitles files, which come in different standardized forms; we used the `.srt`. It looks like this:

```
52
00:11:39,343 --> 00:11:41,110
He won't be a boy forever.

53
00:11:43,013 --> 00:11:45,347
And winter is coming.
```

We can clearly see the file structure and how it can be parsed to obtain the number of words:

- `sub number`: discard it;
- `start --> end`: discard it;
- `dialogue`: count these words.

There are some cases where the `dialogue` is not that easy: it's common use to put some credits to the translators/syncers and the affiliation they work to. So at the end of the episode we can find subtitles containing something like:

```
<i><font color="cyan"> sync
& correction by f1nc0
Addic7ed.com   </font></i>
```

that needs to be removed since those words are not said in the show. Other types of strings unrelated to the w/h are, for example, closed caption like `[wind whistling]` used for deaf people, so we removed any text enclosed by square brackets. We also needed to remove any HTML tag used for formatting (e.g. the subtitle `<i>Can you hear me?</i>` will be showed as *Can you hear me?* where the italics means that the person talking is not showed on screen) so, like before, we removed any text enclosed by angular brackets; be careful that we're removing *only* the HTML tags, not the dialogue inside them (e.g. `<i>Can you hear me?</i>` will become `Can you hear me?`). See the appendix VI-B for an example of text cleaning.

For the Words per Hour formula (1) we also needed the episode duration. We could just have used the standard duration of each series episode (e.g 50min) but we preferred to get it from the subtitle file taking as reference the timestamp of the very last entry of the file. In this way we are quite sure that the duration is not too different from the real one. Also, we don't have to worry when some episodes are

longer/shorter than the standard duration. Finally all these values have been averaged to get an average w/h for each series and its seasons.

C. The properties of a TV Series

Since the Words/Hour quantity alone isn't enough to get a full overview of a TV series, several other information such as genre, year of airing and a brief description next to an easily recognizable poster are shown (Fig. 1).

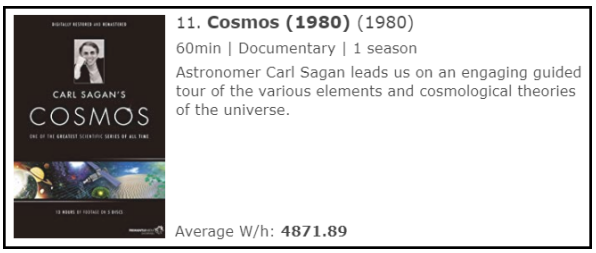


Fig. 1: How the user sees the series data in the chart.

All of this data is stored in the *specs.csv* and *description.txt* files.

D. The data-extractor-scripts

We wrote various Python scripts to extract what we needed:

- `srt_handler.py`: this is the "main" script, the one that actually extracts and calculates the Words/Hour of each series. Basically it runs on all the subtitle files and, while reading their lines, it filters the ones that are not needed and counts the words in those that represent actual dialogue;
- `wrong_encoding_finder.py`: this script finds the files that aren't encoded in UTF-8 that could potentially give an error;
- `utils.py`: contains several methods used in different parts of the scripts.

For a more detailed view of what each function does, see the code and its comments.

E. The project structure

The project has the following structure:

- main folder: it contains the web page and the data.json file;
- /assets: the page icon;
- /css: the css files;
- /data_extraction_scripts: the Python files used for creating the `data.json` file;
- /jquery: Jquery files;
- /js: Javascript files;
- /series: the series data.

1) The Javascript files: `Chart.js` draws the graph. `Controller.js` handles the user interactions and sends the data to the chart. `main.js` is the initialization file. `Tooltip.js` handles the tooltip.

2) The series data: Every series is organized in a folder that contains:

- the subtitles with titles for each episode;
- the black-and-white logo and the original logo;
- the poster;
- `specs.csv` containing its name, episode duration, genre, first and last year it was aired;
- `description.txt` with a brief description.

With the data organized in this way, we can easily add or remove a series.

F. The JSON file

The final goal of the data extraction process is to have a file named `data.json` which contains all the data needed to create a visualization. It consists of an array of objects containing the information regarding a TV series, such as title, description and for example the Words/Hour of each episode. This file is generated by the `srt_handler.py` script and it is available at the following link .

A more detailed description can be found in the appendix VI-A.

G. The TV Series analysed

We analysed the first 50 series of Top 250 IMDb series[1] with some changes; in fact, we added Samurai Jack (we had to have it, everything started from it) and removed other series which didn't have subtitles. Also, it was important to have a well balanced set of TV series from the genres point of view.

III. Implementation

A. The D3 library

D3 (Fig. 2) is a JavaScript library for visualizing data with HTML, SVG, and CSS. We chose it because it has everything we needed.

Fig. 2: Screenshot of the D3 site's homepage.

B. The bar graph

For what concerns the *visual encoding*, we decided a bar chart was the best fit due to its ability of conveying a lot of information while remaining quite basic in its composition. A key aspect regards **marks** and visual **channels** used in the representation. Marks can be thought as geometric primitives like dots, lines or areas. Visual channels as the way

[1] `https://www.imdb.com/chart/toptv`

the information is "channelled" through marks, e.g colour, position (horizontal, vertical) or size.

In our case we needed to encode two attributes: words per hour quantity (quantitative) and the series genre (categorical). A bar chart uses lines as marks and length as visual channels to convey quantitative values. Therefore, in its simplest form it only allows to encode one categorical attribute and one quantitative attribute. Our quantitative attribute (words per hour) has been encoded through bars' height: we are dealing with an "how much" channel formally related to the area (2d size), which is the best choice for quantitative/ordered data. In fact, bars with lower height have a smaller area. The other attribute has been encoded using colour hue as a "what" channel, which instead is optimal for categorical data. Moreover, a bar chart has one spatial region for each mark (i.e a bar for each series, season or episode depending on which data is displayed); these regions are separated horizontally and aligned vertically. This setting allows for a simplification of compare and lookup tasks. Despite its simplicity this combination is very effective for our purposes. The *effectiveness* and *expressiveness* of this combination can be properly demonstrated by taking into account that the visual encoding expresses all of, and only, the information in the data set attributes[3]. The most important attribute (words per hour) has been encoded with the most effective channel, which is length (i.e bars' height).

Furthermore, the two channels are *separable* meaning that differences between bars in terms of height and colour hue are easily detectable. For example, given two bars of different heights and different or same colour, a judgement on the quantitative value of words per hour can be done easily, regardless of the colour. Again, if we were to make a judgement on series genres (colour hue), we could do it easily without interference between the two visual channels.

Our graph is a single view application meaning that all information is visible within a single main view. While all data related to series and seasons can be checked out easily, the same is not true for episodes data which are in far greater number. Therefore, to allow for a proper navigation, a constrained navigation system based on geometric zooming and panning has been implemented, complying with the "eyes over memory" principle even though an overview window is not provided. In fact, for a lesser use of user's cognitive resources, spatial position and zoom level remain the same when data to display is changed.

Context between data set configurations following change of data to display (or filter/order operations) has been maintained with a conscientious use of animations, helping users to track objects when they come into view or they disappear. If a new element enters the chart, it grows from the x-axis. When it exits, it shrinks down. During an update, the bar moves to its new position. When sorting, there is a small delay in order to have a smooth movement of the bars.

C. The data shown

The data is simple: we have vertical bars each one with its title, so we needed their x and y coordinates. D3 retrieves them from an unique identificator and the w/h value. However, since we have a tooltip that shows the series data and appears when hovering over a bar (the tooltip is different when visualizing series, season or episode. See fig. 5), we must keep the data in memory storing it in an array. This array is extracted from the JSON object that D3 loads from `data.json`: we proceed to sort and filter it (according to the sorting/filtering parameters set by the user, see sec. IV-B) extracting the `dataForBars` array. D3's drawing functions are used for visualizing the bars while we take care of the tooltip.

After every change in the sorting/filter parameters, we just extract from scratch the `dataForBars` array. While this might appear as a rude and heavy operation, it is the easiest option: we have only 50 series and since the operation takes less than 50 milliseconds to be done, we don't have to worry about more difficult tasks.

IV. The web page

A. The main chart

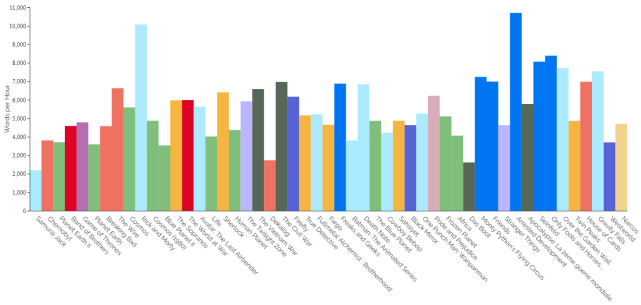


Fig. 3: The main chart.

As we can see in figure 3, the x-axis displays each of the fifty series taken into consideration, while the y-axis displays the words per hour quantity in a range of [0, 11000] w/h. Depending on which data is currently being displayed, the bars stand for series, seasons or episodes. As already mentioned above, series genres are encoded with colour hue (for example blue represents Comedy whereas green represents Documentary). The user can see the colour mapping in the filtering menu.

In the seasons/episodes visualization, a black horizontal line is displayed over the bars' series: it shows the w/h value for the series, so that the user can see how much the season/episode differs from the average w/h of the series (fig. 4). Also when zoomed in, the x-axis shows the season or episode number.

By clicking the "How it works?" button, a tour will guide the user by showing him how to interact with the page.

B. The interactions

Two types of interactions are available for use: filtering and sorting. Through filtering only a subset of elements are displayed in the view (series, seasons or episodes) whereas sorting allows to spatial ordering of elements in the display.

For the former we have planned a filtering based on genres, range of airing years and range of words per hour. The sorting instead is based on different parameters such as w/h, year or genre and it can be accomplished in a descending or ascending way.

The fig. 6 in the appendix shows how the interactions are presented to the user. In particular, fig. 6d shows a little button in the bottom-left of the page: if pressed half of the series will be deleted. Now the universe is *perfectly balanced, as all the things should be*.

V. Conclusions

Leveraging web technologies such as JavaScript and the D3 visualization framework we were able to develop an interactive web application with which users can have an overview about dialogue quantity in the most rated TV series.

Users can see first-hand the relationships between a novel quantitative value, the words-per-hour, and several attributes of a TV series such as genre, year of airing and average duration of episodes.

Future tasks include allowing a user to textually search for a specific TV series and automate the process of extracting its core values, allowing a new TV series to be added to the chart through a simple textual query.

VI. Appendix

A. The JSON structure

We use some examples of objects to show how the data is encoded in `data.json`. It contains an array of objects where each object is a series:

```
{
  "id_": 0,
  "episode_length": 22,
  "genre": "Animation Action ...",
  "start_year": 2001,
  "description": "A samurai sent ...",
  "seasons": [],
  "end_year": 2017,
  "name": "Samurai Jack",
  "wh": 2268.046612903226
}
```

- `id_`: unique id for the series;
- `episode_length`: standard duration of the episode;
- `genre`: different genres for the series, we use only the first one;
- `start_year`: the year the series started to be aired;
- `end_year`: the last year the series is aired, if it's 9999 it means that it's not finished yet;
- `name`: the series name;
- `wh`: the average Words/Hour;
- `seasons`: the array containing all the seasons information.

A season array looks like like this:

```
"seasons": [
  {
    "id_": 1,
    "wh": 2609.059230769231,
    "episodes": [ ... ]
  },
  {
    "id_": 2,
    "wh": 1951.466923076923,
    "episodes": [ ... ]
  },
  ...
]
```

- `id_`: unique id for the season;
- `wh`: the average Words/Hour;
- `episodes`: the array containing all the episodes information.

Finally the episode array:

```
"episodes": [
  {
    "id_": 1,
    "title": "Episode I: ...",
    "length": 22,
    "wh": 2539.09
  },
  {
    "id_": 2,
    "title": "Episode II: ...",
    "length": 22,
    "wh": 2121.82
  },
  ...
]
```

- `id_`: unique id for the episode;
- `title`: the episode title;
- `length`: the duration;
- `wh`: the Words/Hour.

B. The text cleaning

This is a text sample used to show how we cleaned the text from the strings that would alter the w/h value.

Before:

```
1
00:01:57,963 --> 00:02:01,899
Jack: Once again, I am free to
smite the world...

2
00:02:02,067 --> 00:02:05,559
...as I did in days long past.

3
00:02:05,871 --> 00:02:10,205
<i>When the evil shape-shifting
wizard, Aku,</i>
<i>arose from the bowels of
hate...</i>

                    ...

79
00:10:50,040 --> 00:10:51,029
[wind whistling]

                    ...

160
00:22:11,773 --> 00:22:18,256
<i><font color="cyan"> sync &
correction by f1nc0
Addic7ed.com </font></i>
```

After:

Once again, I am free to smite the worId...
When the evil shape-shifting wizard, Aku,
arose from the bowels of hate...

`words count: 22`, which is correct.

C. Images

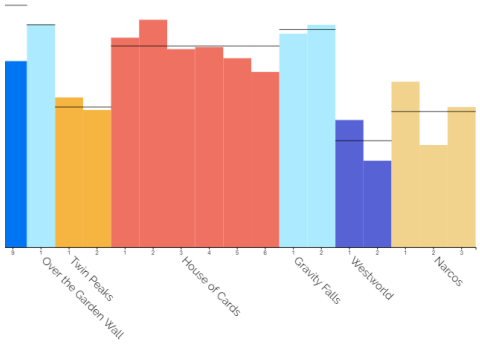
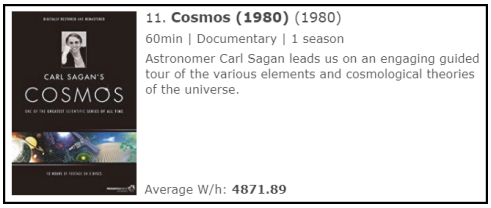
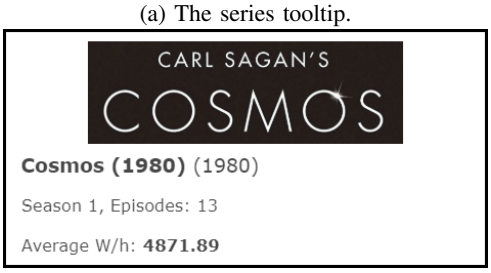


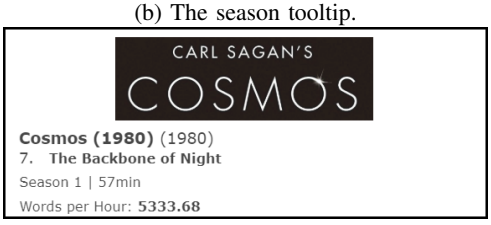
Fig. 4: In the season view, there is a black horizontal line displaying the words/hour of the series.



(a) The series tooltip.



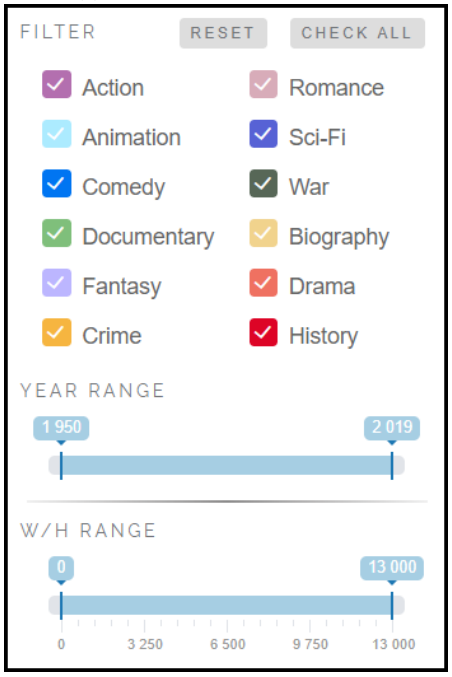
(b) The season tooltip.



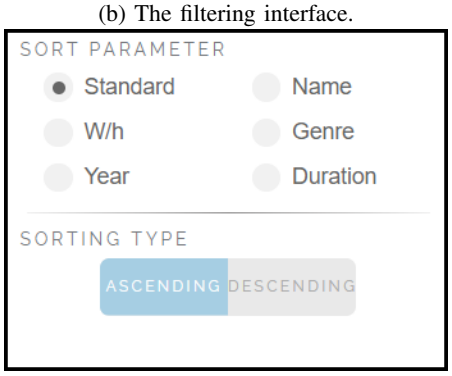
(c) The episode tooltip.

Fig. 5: The tooltip is different when visualizing series, seasons or episodes.

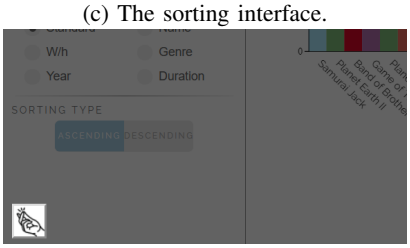
(a) The data selector interface.



(b) The filtering interface.



(c) The sorting interface.



(d) The *Thanos* button.

Fig. 6: The interactions interface.

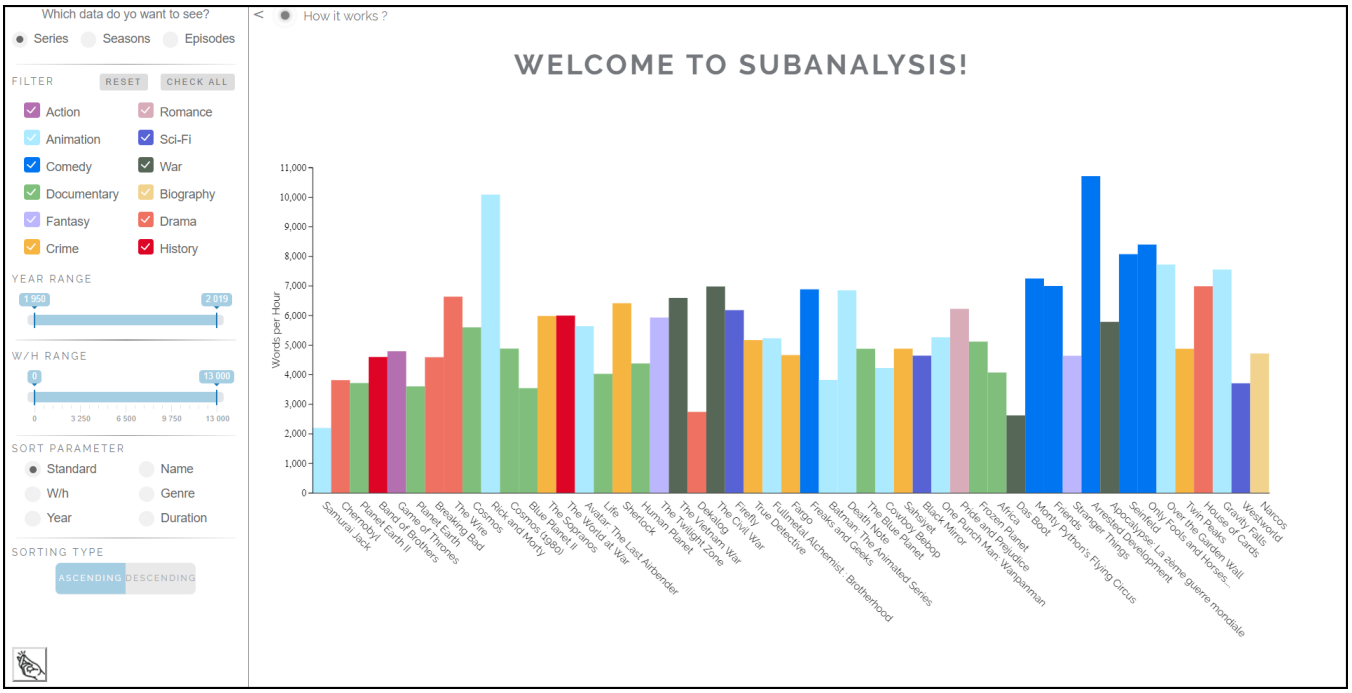


Fig. 7: The full page.

REFERENCES

[1] Link of the Github project: https://github.com/francescopisu/subanalysis.

[2] Link of the D3 Library: https://d3js.org/.

[3] Tamara Munzner, Visualization Analysis and Design, A K Peters, 2014.

[4] Bay-Wei Chang, David Ungar. Animation: From Cartoons to the User Interface. Proc. ACM UIST 1993

[5] Kit Oliynyk: Jedi Principles of UI Animation, link. (last seen on May 27th, 2019)