Exercise 2 – Processing of music data

Universidad San Jorge: DATA EN**GENIEERING**

3ero de IA + INF

Made by: Javier Liarte

Index

Issues to solve:

• (scrapper) Modify the get_songs in ‘songs.py’ file to use the catalog instead of scrapping again. (2 points)

```
# Override this
 catalog = files.load_from_json(Path(f"{output_directory}catalogs/catalog.json"))
 print(catalog)
 for artist in catalog:
     for song in artist["songs"]:
         get_song_lyrics(song["song_title"], song["song_url"], song["lyrics_path"])
         time.sleep(0.5)
```

Instead of reading from the web, we extract the songs from our already downloaded catalog.

• (scrapper) Check the logs for something strange. Try to know where those messages come from. No need to solve it, only get the origin. (0.5 points).

After reading the catalog instead of scrapping the web from the internet, the logs are working correctly, so I do not think I have any issues.

• (cleaner) The cleaner is getting the catalogs also as inputs. Although it is not a problem for now, we need to avoid it. Implement a solution. (0.5 points).

```
#this function is used to list every file using recursive algorithm
def list_files_recursive(path=INPUT_DIRECTORY):
    print(INPUT_DIRECTORY)
    if not os.path.exists(path):
        raise FileNotFoundError(f"La carpeta no existe: {path}")

    #prints are for debugging
    for entry in os.listdir(path):
        print(entry)
        full_path = os.path.join(path, entry)
        print(full_path)

        # avoid avoiding the catalog we improve the execution time because we dont process the catalog, we do not need the catalog after extracting
        # avoiding cleaned is used to avoid saving cleaned songs inside de cleaned folder more than once.
        if entry in ("catalogs", "cleaned"):
            print("IGNORA ESTAS CARPETAS")
            print(full_path)
            continue
```

Implementing this condition allows you to not read the catalog nor the already cleaned files.

• (Validator) There is an issue when reading/saving the files as it is creating more directories than needed. (0.5 points).

```python
def list_files_recursive(path: str = "."):
    """Lists all files in a directory recursively."""
    for entry in os.listdir(path):

        #IGNORE VALIDATIONS FOLDER TO AVOID THE RECURSIVE PROBLEM
        if entry in ("validations"):
            print("IGNORA ESTAS CARPETAS")
            continue
        full_path = os.path.join(path, entry)
        if os.path.isdir(full_path):
            list_files_recursive(full_path)
        else:
            #the problem in windows
            dir_list.append(full_path.replace("\\", "/"))
```

Ignoring the validations folder allows us to resolve that problem. I know that is not the best way to implement it, but it is what worked for my code.

• (Validator) Implement an additional validation rule (0.5 points)

• Propose any change to make the code cleaner, clearer and better. (0.5 points).

```python
def validate_song_format(song):
    """Validates if the song follows a basic expected format."""

    # # Regex pattern for song format
    # pattern = r"((?:[A-Z]+\s+)*\n.+)+"

    #my new pattern to get just songs with lyrics, not guitar tabs or whatever:
    pattern = r"^[A-Za-z\s]+$"


    # another pattern to avoid spam
    forbidden = "espero les guste" in song.lower()

    # Check if the song matches the pattern
    match = re.fullmatch(pattern, song, flags=re.DOTALL)


    # If there is a match, the song is in the correct format
    if match and not forbidden:
        return True
    else:
        return False
```

The changes made here are to improve the code before (using a new Regex function, modifying it to get more songs that look correctly, and the forbidden line is to avoid those tabs that are with that type of text with normally are some message we do not want.

Functionalities to add:

• Add a new Python module called ‘results’ that checks the number of files we have for each output. (0.5 points).

```python
def count_files(path):
    #recursive counting
    total = 0
    #we dont need the root or the dir, because we are only counting files
    for _, _, files in os.walk(path):
        total += len(files)
    return total
```

This is the important function to make it work.

• Add a new python module called ‘lyrics’ that removes all the chords from the successfully validated files and stores it in the file’s directory. (2 points).

```python
def remove_chords(text: str) -> str:

    # Removes chord lines using a very simple heuristic:
    # - If a line contains at least one lowercase letter → treat it as lyrics.
    # - If a line has NO lowercase letters → assume it's a chord line and remove it.

    lyric_lines = []

    for line in text.splitlines():
        # Keep only lines that contain lowercase letters (lyrics)
        if re.search(r"[a-záéíóúñü]", line):
            lyric_lines.append(line)

    return "\n".join(lyric_lines) + "\n"
```

• Add a new python module called ‘insights’ that merges all OK lyrics into a single text f ile for each artist. Count, for each artist, the top 10 words (nouns, verbs, adjectives) that his lyrics have. Do it also globally, but with the top 20 (2 points).

To be honest, I do not even know how to implement this code, and with the reduced catalog the results are way too poor to see it.
What I would do is:
Recoursively go through the OK lyrics, write them in the same text file, and then inside that folder count the top 10. After that I would make the same recursive function search between all the files we got and count the top 20.

• Create a Python file that executes all modules in order. It has to have his own log file, and make sure that if any of the processes fail, is registered there (1 point).

```python
def run_script(script):
    try:
        log.info(f"Running {script}")
        subprocess.run([sys.executable, script], check=True)
        log.info(f"SUCCESS: {script}")
    except Exception as e:
        log.error(f"FAILED: {script} | Error: {e}")
        print(f"Pipeline failed executing {script}. Check pipeline.log")
        sys.exit(1)

def main():

    run_script("scrapper/main.py")
    run_script("tab_cleaner/main.py")
    run_script("tab_validator/main.py")
    run_script("results.py")
    run_script("lyrics.py")

    print("Pipeline finished successfully!")
    log.info("Pipeline finished successfully")
```

The first important function is how it writes it in the console.