COMP3420 Assignment 3

Due: 27th October 2023

Introduction

One of the goals of setting this assignment is to give you a program that you can be proud of which you can use in job interviews and put in a portfolio.

What you will be building:

1. A system that takes a description of a television show and reports the genres of the show. Details are on page 4.

2. A system that takes a description of a television show and finds the most likely television show in its database. Details are on page 5.

For both tasks it is easy to get something simple working – this will be enough for a pass. But it is also very hard to create something excellent that handles everything — which you would need to do to get top marks.

The sample data for both sections will be a SQLite database, which is available on ilearn. It is not the same data that your programs will be evaluated on, but the schema will be the same.

Remember Macquarie's academic integrity policy. You will find that if you submit work that you didn't create yourself for this assignment, you will find it much harder to use it for future job interviews.

Your Submission

You will submit a zip file *studentnumber.zip* which will contain everything needed to run your application. Here is a checklist for you:

- ☐ `requirements.txt`
  The marking software will create a `virtualenv` and will install the packages listed in `requirements.txt`.
- ☐ `train.py`
  A program to train your classifier.
- ☐ `classify.py`
  The classifier program
- ☐ `classifier-design.md`
  A short document where you explain how your classifier works.
- ☐ `index.py`
  A program to set up an index for your search engine
- ☐ `search.py`
  Your search engine program
- ☐ `search-design.md`
  A short document where you explain how your search engine works.
- ☐ A pre-trained model so that your classifier works out-of-the-box. What you call this is up to you.

You will probably have other files as well.

How we will run your programs

Your zip file will be expanded into a folder `/app`. When we run `train.py`, `classify.py` etc, the current working directory will be `/app` too. Your program will have write-access to this directory.

Most questions will be marked automatically. We will use an Ubuntu 23.04 Docker image installed with Python3.11 to run your software.

Some other environment variables will be set as well. Don't feel you need to use PostgreSQL or OpenAI, but they will be available if you want them. SQLite is available as well, obviously (it's part of the Python3.11 distribution), and you have the `/app` filesystem as well.

PGHOSTNAME, PGPORT, PGDATABASE, PGUSER, PGPASSWORD These will point to an empty PostgreSQL 15 database, hosted at Amazon. It will have the `pg_trgm` and `pgvector` extensions enabled. Again, don't feel you have to use them; they are available if it suits your design.

OPENAI_API_KEY If your solution uses OpenAI's APIs for some processing step, this environment variable holds the API key to use. It's the key itself, not the filename to the key.

Specification for the classification system

train.py

`train.py --training-data` *sqlite-file.sqlite*

This program takes the data in the SQLite database and does whatever training process you perform to make a model. The `tvmaze_genre` table lists the genres to predict. It should write that model out somewhere. `/app` is a sensible location, but you could also use the provided postgresql database too if you wanted to. If you want to write back into the SQLite database, remember to copy it somewhere consistent so that `classify.py` can read it.

The databases are not very large, so we aren't expecting training to take very long. If your training takes more than about 10 minutes we will assume it has failed, unless there is some good reason otherwise documented in `classifier-design.md`. (In which case, we'll use judgement about what is reasonable.)

classify.py

`classify.py --input-file` *filename* `--output-json-file` *jsonfilename* `[--encoding` *textencoding* `] [--explanation-output-dir]` *output-directory*

Make sure that your `classify.py` program reads from the same file, directory or database that `train.py` writes to.

This program will always be called with `--input-file` as an argument. This will be a filename, and the file will contain a description of one television show. Television shows have multiple genres: your program will identify what genres this television show is likely to be, based on the description. It will write these genres out to the file listed in `--output-json-file` in JSON format as a list. For example:

```
["Mystery", "Comedy", "Drama"]
```

If `--encoding` is not specified, you can assume that the input file is UTF-8 encoded. If it is specified, you can assume that it is a valid Python encoding.

The output JSON file will always be written in UTF-8.

If `--explanation-output-dir` is present, it will give the name of a directory that your program can write to. You can put anything you like into this directory: an image, an HTML file, a text file. **This is going to be marked by a human being, so use sensible file names and make sure it's obvious what we should be looking at when we mark it.**

classifier-design.md

This document should be about two paragraphs long. First explain how your classifier works. Then explain why you chose to write it that way.

The people reading this will be people like me (Greg), Abid and Bilal, so we are familiar with terminology for machine learning and natural language processing.

Specification for the search system

index.py

`index.py --raw-data` *sqlite-file.sqlite*

The speed of the search performance isn't very important, but at the very least your `index.py` program will need to copy the data somewhere so that your `search.py` program can use it. This is a good time to calculate embeddings, stem words, or any other preprocessing task you need to perform.

search.py

`search.py --input-file` *filename* `--output-json-file` *jsonfilename* `[--encoding` *textencoding*

The arguments are similar to `classify.py`. Read the text in the `--input-file` file (with the textencoding specified by `--encoding`, or UTF-8 if not specified); write to `--output-json-file` the shows that match the search. The JSON should contain the `tvmaze_id` and the `showname` of up to three shows that match the search. Your program's best guess should appear first.

```
[{"tvmaze_id": 24, "showname": "Hawaii Five-0"},
 {"tvmaze_id": 67729, "showname": "Invisible"},
 {"tvmaze_id": 66311, "showname": "Mrs Sidhu Investigates"}
]
```

Note that the search text could be any of:

- An exact match phrase.
- A glob pattern like `super*`.
- Some words which have a different tense or number to the description.
- A sentence that has a lot of words in common with the show's description.
- A sentence that describes something about the show.

search-design.md

Again, this document should be about two paragraphs long. First explain how your search engine works. Then explain why you chose to write it like that.

Marking rubric

Environment setup We will create a `virtualenv` using your `requirements.txt` file. This completes successfully (runs without an error). **1 mark**.

Original inference We will run a few genre-predicting tasks based on the genres in the `tvmaze.sqlite` database provided on iLearn. Full marks if you get all genres correct, Half marks if there's a 50% overlap between your genres and the correct ones. **2 marks**.

Training We will run your `train.py` on some data. It should complete and then we should be able to run a classification using that new model. **1 mark**.

Trained inference We will ask your program to make genre inferences based on that new training data. Full marks if you get all genres correct, Half marks if there's a 50% overlap between your genres and the correct ones. **6 marks**.

Explain We will ask your program to give an explanation of one or more of its genre classifications. Marking on this one is a little bit subjective: we will decide whether the explanation that your program gave seems reasonable and appropriate. Does it help us understand how your program reached its decision? **3 marks**.

Classifier design We're looking to see that your description matches your code and that there is some thought behind your decision process. **2 marks**.

Indexing `index.py` runs successfully. **1 mark**.

Searching We have a bank of searches that we will run. Full marks for each time you get the right television show in position one, half marks if it is in the top three. **7 marks**.

Search design Just like with the classifier design, we're looking to see that your description matches your code and that there is some thought behind your decision process. **2 marks**.

Suggested order

We want to leave it open for you to approach this project however you want. If you feel you need some guidance, consider writing the programs in this order:

- `train.py`
- `classify.py` and `classifier-design.md`
- `index.py`
- `search.py` and `search-design.md`

You will find yourself adding and removing from `requirements.txt` as you write your programs.