Data Mining Techniques

Spring Semester 2022-2023

2nd Assignment – Work individually or in groups of 2

Assignment purpose

The purpose of this assignment is to learn the basic stages of the procedure followed to apply data mining techniques, namely: data preprocessing / cleaning, conversion, application of data mining techniques and evaluation. The implementation will be done in Python using tools / libraries: jupyter notebook, pandas and SciKit Learn.

Description

Goodreads is a social platform that is combined with a book database and looks very similar to IMDb for movies: the users can search books, label them, share their thoughts and discuss. Most importantly, they can rate books that they have read on a scale of 1 to 5 and discover new books to read. Our goal is to study a subset of data from this platform and create a program that recommends books. Another goal of this project is to categorize books into the genre they belong to, automatically, based on their descriptions.
Finally, you can optionally utilize the book covers and create a simple image retrieval system based on its content ***(bonus)***.

The file you are going to use is the Best Books Ever Dataset. You can find the file in the zenodo repository using the following link (you will also find a way of downloading it) https://zenodo.org/record/4265096#.Y-N2DnbP1jE.

Questions

At the beginning, you will import the entire file into a dataframe and check for any NaN values in the columns to remove these rows from the dataframe. Then you will write the appropriate commands in Python in order to answer the following questions. Most of the times, the answers will be given with a graph and you can use any python library you like for this purpose.

Preprocessing (10%)

If you examine **ratingsByStars** column, you will notice that it contains 5 values separated by comma. Split these values and add the ratings to the dataframe separately, i.e. **ratingStar5**, **ratingStar4**, **ratingStar3** etc.

Moreover, genres column contains more than one genres for each book. Create a new column (name if **genreSingle**) and keep only the first genre from all genres given in each row (e.g. ['Fantasy', 'Young Adult', 'Fiction', 'Magic', 'Childrens', 'Adventure', 'Audiobook', 'Middle Grade', 'Classics', 'Science Fiction', 'Fantasy'] → the new column is going to have 'Fantasy'). Delete books without any genre information.

Use **publishDate** column to create a new column with the publication year of each book (you can use to_datetime() method included in pandas or any other method).

Questions to examine data – answer to 5 of the following (20%)

1. Draw the histogram of the ratings in the dataset (use column **rating**).
2. What are the 10 books with the most pages?
3. What are the 10 books with the most 5-star ratings (use only books that have r received over 10,000 5-star ratings in **ratingStar5** column)?
4. What are the most frequent words used in book titles (after stop word removal)?
5. Who are the 10 authors with the most books written?
6. Who are the 10 most reviewed authors (use **numRatings** column)?
7. Rank authors based on the number of books written per year.
8. What are the most frequent languages in which the books in your dataset have been written?
9. Who are the 10 publishers with the most books published?
10. Do books with the most pages (e.g. more than 1000 pages) higher ratings?
11. Collect all unique book genres in a graph or table. What are the most common book genres? Do the same for the awards.
12. Create a wordcloud for **description** column. In this question, remove stop words, experiment with different wordcloud parameters and find the most characteristic words used in books in your dataset.
13. How many books are published per year?

Recommendation System implementation (35%)

The goal of a recommendation system is (1) to predict a user's ratings for books he hasn't read yet and (2) to display a sorted list with the top **N** books we think they would like to learn more about. Another goal of a recommendation system is (3) to help users discover new books they would not have found in another way.

In this question you are going to need
BookId
Description
columns
and only the rows with "English" language.

Create the **TF-IDF** (Term Frequency – Inverse Document Frequency) table of unigrams and bigrams for description column (use the stop_word parameter of TfidfVectorizer).

Cosine Similarity: this metric calculates the similarity of two vectors x, y, using the angle between them (when the angle is 0, x and y are equal, excluding their length). Go through the TF-IDF table and calculate the similarity of each book with the rest of the books. Store the 100 most similar books in a python dictionary.
Prediction: write a function that takes an id and an integer N as input and returns the N most similar books.

recommend(item_id = 4085439, num = 5)

The output of the function should have the following form

```
Recommending 5 books similar to: The Hunger Games
-------------------------------------------------
Recommended: NAME
Description: DESCRIPTION
(score:0.12235188993161432)

Recommended: NAME
Description: DESCRIPTION
(score:0.12235188993161432)
...
```

Classification (25%)

Using **genreSingle** column, find the 10 most frequent genres and keep books that belong to these 10 categories in a new dataframe. You are going to need **bookId**, **description** and **genreSingle**. Then clean **description** column using the methods shown in the tutorials (e.g. punctuation mark removal, conversion of all characters to lowercase etc.). Apply word2vec method to the descriptions and then calculate a vector with 200-300 values (features) for each description using embeddings – that will be the mean of the embeddings of words from which the description is created.

*Use Python pickle library to store features in *.pkl files. In that way, there's no need to calculate the features from scratch every time you run your program, but you can only load them to memory using the respective **load** method.*

Divide the dataset into train (80%) and test (20%) using **train_test_split()** method from sklearn library.

In this question, your program should be able to find the categories (genres) of the test set using the following classification methods:

- Naive Bayes
- Support Vector Machines (SVM, experiment with kernel parameters (rbf, linear), c and gamma. The choice of parameters may also be done with **GridSearchCV**)
- Random Forests

All models written above must be trained **ONLY** on the train set and be evaluated on the test set. Moreover, you should evaluate and write down the performance of each method using 10-fold Cross Validation using the following metrics:

- Precision / Recall / F-Measure
- Accuracy

Finally, create a table with your experiment results for each metric / parameter that you used.

BONUS – Judging a book by its cover…!

Content Based Image Retrieval is a technique that uses visual characteristics (such as color, texture, shape) for image search. Color features are considered to be among the most used low-level characteristics for searching images in large image databases. A color histogram is just a histogram that shows the color level for each individual RGB color channel (where pixel values are in the range 0-225).

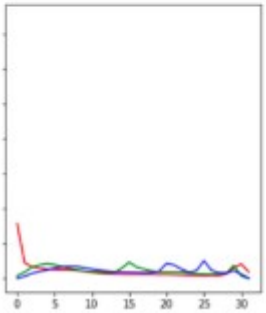
You will find a python notebook in eclass, which you can use to download all covers of Best Books Ever Dataset in .jpg form. Instructions for downloading the images will also be given in the tutorials.

Step one: use the given code to download a set of images of the dataset locally (you can download e.g. 400-500. The more images you use, the better the results of the search in the last step will be).
Keep some images for testing later (e.g. 5%).

Step two: histogram calculation for each image. For this purpose, you are going to use **OpenCV** library and specifically the **calcHist** method.

cv2.calcHist([images], [channels], [mask], [bins], [hist_range])

images is the image in BGR form. This argument expects a list of images and thus, we have placed an image between square braces [] if it's going to be just one image.
channels is the color channel (BGR) for which we want to create a histogram – we do this for a single channel at a time.
mask is another matrix of images that consists of values 0 and 1, which allow us to cover (e.g. to hide) some part of the images. We are not going to use this, so we set it to **None**.
bins is the number of histogram bars on which we place our values.
We can set it to 256 if we want to keep all initial values, but we suggest to use 32 or 8 bins for performance related reasons.
hist_range is the range of expected color values. Because we use RGB / BGR, we expect a minimum of 0 and a maximum of 255, so we write [0, 256].



You can create a histogram for each color channel (BGR) – do this for a single channel at a time and then combine the results.

After converting our images to three vectors representing the three color channels, you have to compose a vector out of these three. Apply this procedure to every image and what the result will be is a dataframe of the histograms for each image with an id, so that you can distinguish them.

Step three: choose an image from the test set and calculate its histogram.

Step four: search for the most similar images in the image dataset comparing the histograms either with the euclidean distance or the cosine similarity. For each query, display the 4 most similar images. Use **matplotlib** to display the results. What you need to know about this library is that it loads images in **B**lue **G**reed **R**ed (BGR) form instead of the classic RGB that was referenced earlier. You will have to convert each image (mainly flipping – reversal with **flip()** command of numpy library) to depict it correctly in the output.