Data Mining Project

IMAGE RECOMMENDER SYSTEM

Etienne ENGEL | Thibault GILG | 04/04/2020

Objectives

The goal of this project was to create a python script that could recommend images based on the user's preferences thanks to all the practical and theoretical courses we had in data mining. This project involved notions of data mining such as the use of dataframes, the Minibatch k-means algorithm and data prediction.

Implementation of the system

DATA ACQUISITION

In this system, we used a database of 809 unlabelled pokemon images downloaded from the website *kaggle.com*. The images are associated to a CSV file structuring the database. The CSV file informs about the name of the pokemon, its primary type and its secondary type.

The use of a notebook is practical as we could separate the whole code into multiple cells to have a clear organization. After having imported the CSV file and all the images, we had to download the colormath module in order to have all the colour-related analysing tools. Then, we decided to convert the CSV file into a JSON file as we learned how to parse JSON files in a previous practical lesson. In addition, to extract the data and store it in a dataframe, it must be a JSON file. We subsequently created the dataframe that would hold each pokemon's attributes: name, type 1 and type 2. This dataframe will be enlarged with further data afterwards.

DATA EXTRACTION & TRANSFORMATION

Then, we run the initialization cell thanks to the GetColors() function in order to extract data from all the pokemon images and store it in the dataframe.

First, we started by identifying the three predominant colours of each image. For that, we used the Mini Batch k-means clustering method. The Mini Batch K-means algorithm is an alternative to the K-means algorithm for clustering massive datasets. The advantage lies in dividing the entire data into multiple subsets of data, and using a subset of a fixed size for each iteration. This strategy is useful to gain computation time and avoid having the whole dataset into memory for each iteration as for the usual k-means method. However, this version has some side effects in terms of precision which are insignificant comparing to the execution time we gain. K-means clustering aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean (also cluster centres), serving as a prototype of the cluster. Basically, we first set k initial means corresponding to k clusters. Then, we assign each observation to the cluster with the nearest mean. Finally, we recalculate the means for observations assigned to each cluster which will be re-centered according to their new mean. We reiterate the last two steps until the assignments no longer change. By the end, we will have k clusters, each one representing one colour.

Furthermore, we created a list containing all the pokemon images, PNG and JPEG files. We also added some columns to the previous dataframe concerning the three predominant colours and the number of pixels for each. For each pokemon, we found the three predominant colours and the number of pixels for each thanks to the *GetColors()* function. It was important to convert the colour values into the correct colour space in order to parse them. The sRGB space is the colour space commonly used on computer screens that we utilized to collect the data in order to convert it to the CIELAB colour space. This colour space is essential for the comparison of values according to our method. Every value was stored into the dataframe afterwards.

The next primordial data for our system was the size. For this criterion, we considered the size of the pokemon, not the size of the image, because they all had the same dimensions. The size (not the height) of each pokemon was calculated per the number of coloured pixels (neither absolute black, nor absolute white). The method used was not exactly the same between JPEG and PNG images because of two factors. First, the *getpixel()* function returns the RGB values of each pixel for the JPEG images and the RGBA values of each pixel for the PNG images. The second factor was because PNG has an absolute white background (0,0,0) and the JPEG has an absolute black background (255,255,255). This method consisted in incrementing a variable whenever a pixel was different than absolute white (or absolute black), which would represent the size of a pokemon. We added a *Size* column in our dataframe to store the size of each pokemon.

DATA ANALYSIS

The final part is the most important one as it enables us to recommend images the user may like. After having stored all the data in a dataframe, it was time to analyse this data. So, we created the *GetDataFromDataframe()* function, returning for each pokemon, its index in the dataframe, its primary type, its secondary type (if there is one), its size, its three predominant colours and the number of pixels for each colour. Then, we added a column *PredictRate* in the dataframe, a coefficient whose value would change according to each pokemon's resemblance with the chosen one.

For the interaction, the user will choose how many images (between 1 and 809) he wants to see. In response, the system will display this number of random images. Subsequently, the user will have to enter the number of the pokemon he prefers. For the chosen pokemon, all of his data will be retrieved. In order to have a fair prediction of the similar pokemons, we decided to equally distribute the influence of each feature on the variable *PredictRate*:

- 35% by the colour
- 35% by the size
- 20% by the primary type
- 10% by the secondary type

Concerning the colour, we had to set the colour acceptance limit to 10 for the Delta E (CIE1976) between two colours. The Delta E characterizes the contrast between two colours represented in the CIELAB colour space. Delta E equals 0 if both colours are the same and

100 if we compare black and white. If the Delta E respects the colour acceptance limit, we increment *PredictRate*. For the size, its acceptance limit is set to 200 pixels. In this limit range, the variable *PredictRate* is multiplied by a certain value. Whenever, one of the pokemon's type is the same as the preferred pokemon's type, *PredictRate* is also increased. The values are approximate, to match the influence distribution as much as possible. Furthermore, the dataframe is sorted by the values of *PredictRate* in a descending order and the index are reset according to the new order. Finally, the three most resembling pokemons we display are the three first ones in the dataframe. As a verification of the efficacity of our system, it almost always displays the chosen pokemon among the three.

Self-evaluation

Overall, our system works and recommends similar pokemons. We have reused notions mastered during practical work. However, our system saves the preferences of the user just for one run so, every run of the program is independent from the others. Eventually, some functionalities could be improved or added:

- Store the preferences of the user in an array and reuse them for the next executions of the system;
- Find a more precise method to equally distribute the influence of each feature on the prediction;
- Compare all the colours together rather than only comparing colour 1 with colour 1, colour 2 with colour 2, … (our method);
- Use classification and supervised learning algorithms such as Perceptron.

Conclusion

This project was a very concrete way to apply all the notions learnt during this Data Mining course. At the end, our system manages to analyse the size of a pokemon, its colours and its types to finally suggest similar pokemons. Building an image recommender system was very interesting but not easy. In fact, data mining is completely new to most of us and quite complex. Therefore, it would have been useful to sum up the main notions and algorithms at the end of every practical lesson, because most of the exercises consist in copying some lines of code, analysing them and their result and slightly changing them when needed.