Monjoux Hugo
01/09/2020

Report for the technical test

Exercise 1A

The context for this exercise regards the usage of the ExpiryApp that helps teams in stores to manage expiration dates of products. When a product is added to the App, the expiration date needs to be set and the app will then keep track of all the products. The problem that is addressed here is that the products need to be manually added to the ExpiryApp product list. Sometimes products can be forgotten.

A script is therefore needed to support the identification of such products in the store product list and to recommend them for the ExpiryApp.

The approach is in 3 steps:

- Analysis of available data.
- Identification of an algorithm and implementation
- Recommendation for improvements

1. Data Analysis

Two datasets are available:

- A fake set of products currently tracked by our system in a specific shop (3114 products)
- A fake extraction of the shop internal system which contains its vision of the existing references and the current shop assortment (39136 products)

Each product tracked in the ExpiryApp system as the following fields associated with it: shop name, aisle, bar code reference (EAN), the name of the product and the last time it was checked. The Shop internal system tracks many more fields for each products. For example it tracks the Label (a common name used to refer to a category of products) and the quantity still in stock.
The two datasets can be joined on the column EAN present in the two.

First, it is interesting to see how many Labels are defined:

Label	**Count**
Famille	66
Sous-Famille	313
Unité de Besoin	1489
Code Interne Enseigne	33524

As you can see the more the Label is precise, the more the count of different Labels is high. Products defined in a specific category can be found based on the hierarchy of labels: Famille > Sous-Famille > Unité de Besoin > Code Interne Enseigne > Product Name.
The following table illustrates the repartition of labels for the tracked products:

Label	**Count in Shop dataset**	**Count in App dataset**
Famille	66	40
Sous-Famille	313	177
Unité de Besoin	1489	504
Code Interne Enseigne	33524	2931

Some labels are not represented in the App, the user may have decided to track them by other means. It is also interesting to take a look at the spread of products by Label in the App. In fact it may be possible that some types of products appear only once or twice. The following table presents a summary of this study:

	Famille	**Sous-Famiille**	**Unité de Besoin**	**Code Interne Enseigne**
Mean	74	16	5	1
Standard Deviation	73	21	7	0
25%	12	3	1	1
Min	1	1	1	1

For each label the number of products tracked,the mean, the standard deviation, the minimum value and the first quartile are provided. It can be seen that some families have only one product defined on the App. What’s more the standard deviations are really large, which means that there is an important difference between the number of products of the different classes. It also shows that Unité de besoin and Code Interne Enseigne are too specific to give us any information on the product repartition.

Another interesting field defined for each product is the ‘Quantity in Stock’ field. It is a numerical value that tracks the number of products in the shop. The value can be positive or negative. When negative it indicates that the product was a pack and only an item of the pack was taken. When the value is null it means that even if the product is defined in the system there is no such product in the shop. An indication on the stock is given by the mean with a value of 4.

The Etat assortiment fields keep track of the displayed products in the shop. If a product is not displayed it is defined as Hors assortiment. Such a product may expire but is not at risk to be sold to a shopper.

The last_ control field corresponds to the last time the user updated the values of the product in the app (he checked if the product was still good). In our dataset the timerange goes from 01/2020 to 08/2020.

The date of the last reception corresponds to the last time a product was received by the shop, it goes from 06/2015 to 08/2020. Products received too long ago may introduce a bias as they may have been left out on purpose.

2. Algorithm and implementation

The environment in which the script runs is a python 3.7 environments with the pandas and xlrd libraries installed. Pandas DataFrames were used as the base structure to work with. The csv and xlrd, once imported, are therefore stored on the same format.

Script objective: “find and recommend the most pertinent products that would need to be inserted in the ExpiryApp.”
The problem is subdivided in two steps:

- Is it possible to find all the products that should be on the Expiry App?
- Is it possible to select only the most pertinent ones?

To answer the first question it is necessary to filter the products listed in the shop system without missing any. The filters used are based on the result of the data analysis done before, they are (in order of usage in the script):

- Filtering the product by dividing the products already initialized on the app and those that are not. From the initial 39136 only 36139 remain.
- Filtering the products by keeping only the Sous-Famille Labels seen in the ExpiryApp. They are general enough so as to not impede new products to be added to the app. Only 23911 products remain.
- Filtering the products by only keeping the products that are in the shop (the quantity in stock needs to be different from 0). Only 2023 products remain.

The number of products is too important considering the work that needs to be done by the user. It is also interesting to note that only 30 products were received before 2020 and none before 2019.
As these products are not older than one year, the assumption that they are still used by the store is made.

We now need to answer the second question and to find the most pertinent products to initialize. The steps used to do this are the following:

- Filtering the products based on their Etat d’assortiment. The products not displayed will not be a high priority for the app user when checking expiration dates. We can therefore remove them from our list. From the 2023 products 1806 remain.
- The remaining products are sorted by the labels Famille and Sous-Famille which have the most representation in the Expiry app.

The remaining list is recommended to the user to be initialized in the App.

For the optional objective the aisle name was taken from the field aisle defined in the ExpiryApp and associated with the Sous-Famille label so every product has an associated aisle.

3. Conclusion and Improvements

Error management:
Some errors in the datasets can be present and not addressed with this version of the script. For example the quantity in stock could be negative while the product is not a pack. Another type of error is present when doing the join between the in app data and the shop dataset. Some products defined in the app are not present in the shop.
This type of error is not yet handled by the script.

User preferences:
The resulting script provides a recommended list taking some assumptions on the user preferences. An improvement could be based on getting a user feedback by asking the user to choose a preferred filter.

Exercise 1B

Context:
When a product is near it's expiration date, ExpiryApp will notify the user. There are different ways to handle the close product expiry date. The user could put a discount sticker on it, or may give it to a charity or a association that could use some products.
Would it be possible to make the ExpiryApp suggest a solution to optimize this choice?

The approach is defined in 3 steps:

- Data collection and definition of clients needs
- Prototyping a predictive model
- Transfer the model in operations

1. Data Collection and definition of needs by the clients

The first step would be to collect more contextual data. Some additional data like the price of the product, the seasonality of the product, where the shop is situated, what time of the year it is, what type of clientele frequents the shop usually, etc… would be needed.
The needs of the clients also need to be clearly defined during this step of the work.
What's more, there may be a need to restructure the data as the raw data may not be exploited easily.

Implementation would require the development of an environment running with python to leverage the multiple libraries of data visualization, data mining and exploratory data analysis such as pandas, numpy, matplotlib, seaborn….
Additional database would be needed to contain the data to be used. Depending on how the data is structured, a NoSql or Sql might need to be put in place (examples of those are Elasticsearch,

MongoDb, PostgreSql, MySql).

2. Prototyping a predictive model

Several models are strategies can be defined. It will be important to introduce a way to compare the respective performances. A metric that could show the respective performances of the models could be introduced. The metric should be based on the evaluation of cost and benefits between the different solutions. A large part of this step lays on how the metric function is defined as it will serve as the reference between all the models tested and make sure the right one is chosen. The cost may not be the unique criteria as the participation to charity can be also considered as very much beneficial for the image of the store.

Depending on how the data is gathered there are multiple solutions in regard to the architecture of the predictive model.

The first one would be divided in two different models. A prediction of the sales for the products and a choice between the outcomes when the expiration date is considered. The prediction of the products sales is a prediction of a continuous value and as such should be treated with algorithms like Random Forests, SVM (support vector machines), linear regressions and if interpreted as a time series problem RNN (recurrent neural network) could be used. Then the choice of action is a classification problem. The algorithms to test are SVMs, Random Forests, logistic regressions and CNN (convolutional neural networks)

The second solution would be to consider that the problem is a classification problem from the beginning and leverage the same algorithms to create a model.

In both cases the training dataset and test dataset should be big enough to give plausible results. Once a fitting model is found, it can be improved by checking it's parameters. A grid search will make it possible to test each parameters against each other to find the best possible combination for the model.

In the models above, the participation to charity can be modeled as a benefit for a certain amount of products or directly at product level. In such a case, the users would need to be careful that the model is not predicting the income but rather a general benefit for the store including on its public image.

The tools used for this step are: a python environment with the scikit-learn, tensorflow, keras, pandas libraries

3. Transfer the model in operations

Once models are defined, it needs to be put into operations. To choose the best way to implement it depends on the size of it. If it is too computationally heavy it may not run on the user's application. In that case it may be better to run it on a server or to run the model with common products so as to create a database of results that can be easily accessed by the user. This step depends heavily on the two previous steps.

This project depends on how the data is gathered and the user interaction with the models. Implementation and validation are not too difficult and a period of 5 to 6 months of development can be estimated once the data collection part is confirmed.