An application of Naive Bayes on Amazon's products

Arthur Bonetti
Master in Finance - Financial Entrepreneurship and Data Science
University of Lausanne
Lausanne, Switzerland
arthur.bonetti@unil.ch

***Abstract*—Amazon is a vast marketplace where it can be difficult to find the perfect product that meets our need. This project proposes the implementation of a Naive Bayes classification algorithm on a small subset of Amazon's product to advise better products to an user. The process is optimized by looping on small sections of the database and performing pre-processing algorithm.**

***Index Terms*—Naive Bayes, Amazon, database, scraper, optimization, classification**

I. Introduction

One of the yearly struggle that everyone has already experienced once is the trouble of finding a good gift for a relative or friend. With a lot of marketplaces available such as Amazon, Aliexpress, Zalando or Digitech and million of products to choose from, one can easily spend hours searching and lose himself in the vast expanse of products available.

Algorithms used by these giants re-sellers are already at the cutting edge of the technology and companies like Amazon spend 42 million a year on technology and content expenses [1]. However, the drawback for the user, with these algorithms, is the constant monitoring of the activity on the website. Every second an user spend searching for the perfect gift is tracked, from how long he looked at a product to his search activities history. The aim of the company is clear, it is to make the user spend more time and money on their website by advising complimentary products among others. The kind act of looking for a gift for a relatives can quickly become a psychological battle between algorithms and self control.

Furthermore, the shopping activity does not end once the user left the company's website. The activity is tracked and is used to advertise products on other websites. Someone who just wanted to buy baby socks for his friend's newborn can easily be advised baby products for a few weeks.

II. Research Question

A. Problem

A current problem for some people when searching for a product online is over information. According to a paper published in 2017 [2], with the internet providing massive amount of heterogeneous information, people may perceive this medium as challenging. Information overload can be a consequence of the difficulty to select and evaluate relevant information. The paper was conducted in the context of online news exposure. In a safe way, this can be extended to the selection of a gift while online shopping.

B. Objective

The objective of this paper is to create a program for people looking into buying a gift in an easy way, without overwhelming them with information. The aim is to propose products based on a like or dislike test and to recommend new gifts based on a Naive Bayes classification algorithm. Naive Bayes is a classification method that is often used for text classification, but variants for arbitrary data exist. Naive Bayes relies on the Bayes theorem and the assumption that the data is independent. The end result of our program is to redirect the user directly on the page of the product he has chosen so that he can buy the gift he has selected without being tracked by the company's website.

C. Scope

Considering the size of the product's database and the difference of information available among online shopping companies, this project's scope lies on a self made database from a subset of products from Amazon. The database contains 950 products from eight different categories. The scope can easily be extended to larger database and different online shopping website.

III. Methodology

A. Database

The first step into the project is to create an adequate database to serve as an example and a basis for testing the program capabilities. The initial idea was to use the Application User Interface (API) from Aliexpress to retrieve data from products. However, this process required to own a seller's account and was not adapted for the scope of this project. Therefore, we decided to use a tool available on amzscout [3]. The tool amzscout Product Database allow to fetch product from different categories and obtain useful information on the product features. The product database is limited to a few searches on the free account, but this can easily be overridden by the use of temporary emails and the creation of a few free accounts. We then collected the different fetched data and combined them into a single .csv file.

B. Scraping

The second step is to scrape images for each product from Amazon's website. Indeed, having a picture of the product is an essential element to help the user decide on the likeness of a gift. Pictures usually are larger in term of memory size than common data, therefore they are not stored in the database. Because we need to show the product's picture to the user, we decided to scrap the main picture from the product's URL. This is done using the imgscrape library from Pypi.

C. Visual Interface

The project needs a way to show the information to the user. To this end we use a convenient library named Tkinter that allows us to create windows and add pictures, label and buttons with Python. The window is updated every time the user interacts with the interface. To keep a trace of the user's likeness of a product, we update a subset of our database every time the user interacts with the program.

D. Naive Bayes Classification

To advise better products to the user, we use a Naive Bayes classification algorithm. At first, we don't have information about the user's preference. Therefore, we select ten products at random from the categories that are selected. Then, the user decides for each product whether he likes it or not. If the user likes the product, it is classified as 1 and if he dislikes it, it is classified as 0. After ten iterations, we compute a probability score for each product on the whole data set using the classification algorithm and fetch the ten products obtaining the score closest or equal to one. The next step is to show the new products to the user and continue the process. Every ten iterations, we compute again the classification probability with the new observations and thus provide a better classification every time.

IV. Implementation

The program for this project was programmed using Python 3.8.3 and requires chromedriver and the following packages to run:

- Pandas - version 1.2.3
- Numpy - version 1.19.2
- Tkinter - version 8.6
- Selenium - version 4.0
- Pillow - version 8.2.0
- Requests - version 2.25.1
- Mkl-random - version 1.1.1
- Sklearn - version 0.0
- Matplotlib - version 3.3.4
- Imgscrape - version 0.1.1

A. Pre-processing algorithm

Pre-processing the data is a necessary step for the correct working of the program. We implement the cleaning, scraping and saving of the data in the scrape_n_save.py file. First, we import the raw data from the data_prog.csv file to a pandas dataframe. We then drop the column that are not useful for our program to reduce the amount of data. We also drop the columns containing empty values that might causes the classification algorithm to complain later on. After that we create two functions. The first function scrape_img(lst_var) calls the scrape function from the imgscrape package, gives it a list of URLs and return the result. The second function iter_image(df, df3) iterates over the database, fetch the URL for each product from the database by bulk of ten, save them in a smaller dataframe and call the scrape_img(lst_var) function. When this function is called the computer will use chromedriver to open a chrome tab, go to each product's URL and save the product's main image URL in a variable. When the program has fetched the pictures' URL from the ten products, it adds them to the dataframe under a new column named Image and save the dataframe as a csv.

At first, we wanted to scrape the image of the product during the execution of the program so that we were not stocking the pictures' URL for each product. However, it appeared that scraping the pictures was a time consuming process as the computer has to open a chrome tab for every request. We considered the use of parallelism for this process, but it seems that it is not quite a good idea to open ten google chrome tabs at the same time and the computer won't allow it anyway.

We fetch the picture by bulk of tens to offer a guarantee in case the scraper encounters an error. Indeed, a product can be deleted from Amazon or the computer can go to sleep during the process. Saving the results every ten iteration gives the possibility to take back the process for where it crashed when an unexpected error occurred instead of having to run it from the beginning again. For information, during the first run of this file and with raw data, the program encountered two URLs leading to missing product, went to sleep twice and was once stopped by an accidental touch of the keyboard by the programmer.

Once the pictures' URL are collected into the respective .csv files, we import the csv to dataframes and concatenate them together to have the final database that will be used for the program. The database is saved in the datav_prog_final.csv file.

B. Keeping data of interest

For this project, we put some effort to keep the size of the data used during the program to the minimum. We have eight product's categories in our database and we know that some categories will interest some user and not others. Therefore, at the beginning of the main program, we ask the user for the categories of interest. Because it is time consuming and not efficient to write the name of the categories, we simply ask for numbers. The user can input the categories he wants to keep. Each number is attributed to a category. For example one particular user could write "12378". We then use a loop and a str() function to see if the input contains numbers.

The numbers are then converted to create a new dataframe that keeps only the categories of interest that will be used for the program. The new dataframe created is named df2. The algorithm used to analyse the input will keep only the numbers. Therefore, if an user gives as an input ”aiàéèç2kl45”, the algorithm will detect 2, 4 and 5 and keeps the affiliated category. If an user does not give any input, we consider that he wants to keep all categories.

After that step, we select ten rows at random from df2 and save them in a new dataframe named df3. This new dataset will serve as a basis to display the information in the Tkinter window and compute the classification score.

	Name	Price	Min Price	Net	FBA Fees	LQS	Category	Sellers
827	AstroAI Air Comp	25.49	25.49	16.15	9.34	78	Tools & Home Improvement	1
842	Filtrete 10x20x1,	80.06	57.25	59.08	20.98	80	Tools & Home Improvement	4
817	Rust-Oleum 249	4.79	4.79	-0.85	5.64	84	Tools & Home Improvement	10
570	PowerA Play & C	14.88	11.99	9.48	5.4	87	Video Games	4
556	Nintendo Blue/ I	66.99	66.99	57.42	9.57	84	Video Games	23
392	Avidlove Lingerie	15.99	15.99	9.06	6.93	90	Clothing, Shoes & Jewelry	3
144	Insect Lore Butte	36.99	36.99	26.34	10.65	84	Toys & Games	2
659	WORX WG896 1	93.53	86.05	64.12	29.41	74	Patio, Lawn & Garden	12
32	Zmodo SD-H108(	30.6	30.6	21.08	9.52	68	Electronics | Computers | Acces	2
851	Filtrete UR10, fil	101.94	101.94	76.12	25.82	86	Tools & Home Improvement	3

Rank	Est. Sales	Est. Revenu	Reviews Co	Rating	Weight	Colors	Sizes	URL	Image	Score	MI_score
20	94560	25.49	4396	4.4	2.2	2	1	https://ww	https://ima	1	0
82231	73880	5412640	30	4.9	0.9	1	1	https://ww	https://ima	0	0
51	34228	441061.1	34214	4.7	1	127	1	https://ww	https://ima	0	0
496	2243	33366.09	5912	4.7	0.48	6	1	https://ww	https://ima	0	0
45	9751	653219.49	63026	4.8	0.37	6	1	https://ww	https://ima	0	0
434	8501	135930.99	24268	4.5	0.13	26	7	https://ww	https://ima	0	0
10	26939	996473.61	13214	4.7	1.3	1	1	https://ww	https://ima	0	0
405	8332	779291.96	5518	4.4	16	1	1	https://ww	https://ima	0	0
9	44292	1355335.2	1859	4.3	0.8	1	1	https://ww	https://ima	0	0
33	32914	3355253.2	17384	4.8	4.34	1	26	https://ww	https://ima	0	0

Fig. 1. Dataframe df3 example

C. Tkinter interface

We use Tkinter to show the information about a product to the user. Our dataframe contains eighteen columns and most of the information is not necessary for the user to decide of his likeness to a product. The information that we consider essential for the user is :

- The picture of the product
- The description of the product
- The price of the product

We consider these three items as an object, therefore we create a class named Model(object) that keeps the above mentioned information in a single object. In the same class, we also initialize a counter that allows us to keep track of the number of products that has been shown to the user. We get the values from the first row of the dataframe df3 and create a Model(object) named count_object. During the whole process of the program, we will use this count_object and update it with the values of the current product.

When the program is launched, we initialize the Tkinter window for the first time. We get the picture's URL from our object and use the request function to save the picture in a variable. After that, the picture is resized to a 400x400 format and is added to the window. We also get the description and the price of the product and create two labels under the picture.

We then create four buttons.

- Button ”Yes” which allows the user to say that he likes a product and save the value 1 in the dataframe
- Button ”No” which allows the user to say that he does not like a product
- Button ”Stop” which allows the user to stop the program
- Button ”Buy” which redirects the user to the page of the product on the merchant website

Once these elements are placed in the Tkinter window, we show them to the user.

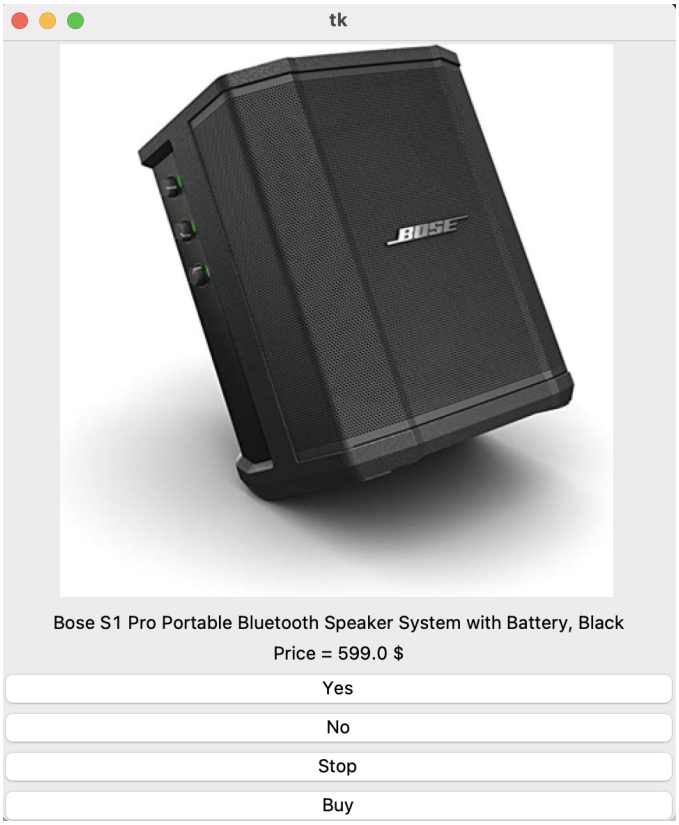

Fig. 2. Tkinter window example

Now, we create a function update_tk() that updates the window to show new information every time the user click on Yes or No. The function updates the counter of the current object and updates the values of the object to be the values of the next row in the dataframe df3. Once this is done, we update the picture and the labels accordingly and we keep the current layout. This process is repeated until the user reaches the last (tenth) product of the dataframe df3. As a reminder, every time that the user clicks on ”Yes”, the value 1 is saved in a column named ”Score”. The default value of the ”Score” column is 0 so that when the user clicks on ”No”, the program does not have to update the dataframe.

D. Naive Bayes Classification

Once the user has gone over the ten first product, we predict the probabilities of a product from the dataframe

df2 to be classified as 1. To this end, we create a function named ml_score(df3, df2). In this function, we first define which columns we wish to consider for the X values and we collect these values from the dataframe df3 and save them to a variable X. We then define our regressor y which is the values of the column "Score". The score's value is 1 if the user liked the product and 0 otherwise. Therefore, we are trying to predict which other product could have a score of 1 based on the X observations. The values we are trying to predict are in the dataframe df2, which as a reminder is the dataframe that contains the information on all products from categories that the user decided to keep. From this datafrmame, we select the same columns as in X and create our variable X_test. We then use a preprocessing.MinMaxScaler() to normalize the data and fit our data in a Naive Bayes Classification algorithm from the sklearn package. The last step is to predict the probabilities of the product to belong to the class 1. The values are stored in the dataframe to the "Ml_score" column.

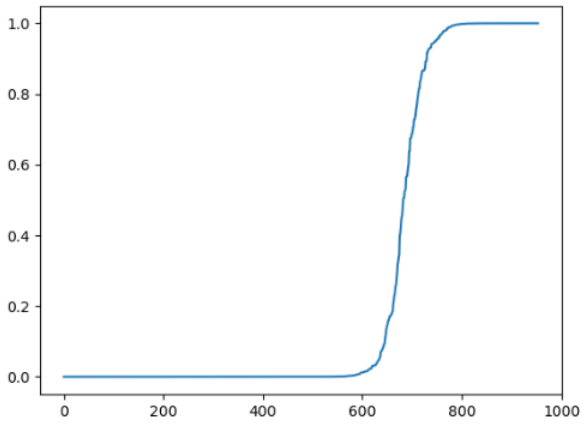

Fig. 3. Naive Bayes Classification example with 3 Yes out of 10

E. Updating the dataframe df3

Now that we know the predicted score for our whole dataframe, we select the ten products that got the biggest predicted score with the Naive Bayes Classification algorithm and add them to the small dataframe df3. Because we use this dataframe a lot throughout the program, we decided that it was best to keep track of it by creating a class named Model_df(object). We store the dataframe as a Model_object and update it every time we get the ten new best results with the classification algorithm.

Because we do not want to fetch products that we already showed to our user, we drop the product's row from our dataframe df2 that we add to the dataframe df3. Another advantage of that process is that we get rid of data duplicates and therefore save time on computation.

F. Finding the dream gift

For every iteration, we collect more user's preference and can compute the Naive Bayes Classification algorithm with more observations. The result of this is a more precise classification every time the user's gives his opinion on a product by clicking on "Yes". The program shows products that are more likely to interest the user and he will, more often than not, find the perfect gift that he came looking for. Once he has found that gift, the user simply has to click on the buy button to be redirected to the web page of the product. This is simply done by using chromedriver and giving the product's URL as an argument.

G. Performance

The program for this project does not require a lot of processing power and does not need a lot of time to process the computation. Indeed, the program is written in a way that each loop iterates for a maximum of ten times. The computation in the loop are quite basics and does not require a lot of resources from the computer. The overall improvement in performance from the first version of the project was done by pre-processing the data and scrape the product's picture URL in advance so that the user has a lag free experience. The project can easily be replicated on a bigger dataset without causing trouble. However, we have to keep in mind that the Naive Bayes Classification algorithm predicts values for the dataframe df2. It can cause some slow-down in the execution of the program if the size of this dataframe is big enough. In that case, we would advise to perform the classification algorithm on a smaller dataframe. Because we only need to fetch ten new products every time, an enormous dataframe is not needed.

V. Maintenance and update

At the moment, the program requires the manual input of the category's names and the manual selection of the column's names of the dataset for the Naive Bayes Classification algorithm. Therefore, an interested third party should update these variables to meet his needs before launching the program.

The codebase for this program is located on my personal git repository: LaPetiteBiche/prog_final_project . Anyone interested can create a pull request to update the code. The last git pull is a working version of the program with a lot of comments added to help any third parties understand the code. The README is also updated and gives the necessary information for every person interested in this program.

VI. Results

The result of this project is a working program that helps the user find the perfect gift. To illustrate the result of the project, we will follow the steps of an hypothetical user. The user launches the program and has first to decide of the categories he wants to keep. Our user wants to find a product for his nephew who likes video games and sport, therefore he selects the categories 1. Toys and Games, 3. Sport, Outdoor and

Fitness, 4. Video Games, 8. Electronics. He puts as an input for the program ”1348”, then he accidentally pushes ”jbkn” before submitting. The program recognizes that the user wants the categories 1, 3, 4 and 8 and creates a dataframe df2 containing only the product from these categories.

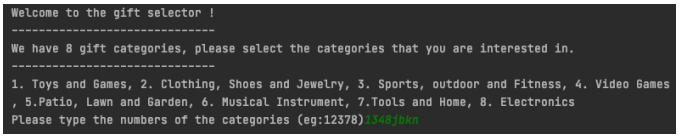

Fig. 4. Welcome Message and Input example

The resulting df2 is a 18 columns and 463 rows dataframe. At that stage, the user is shown a product and has to decide if he likes it or not. The first product he gets is a T-shirt. He dislikes it, clicks on ”No” and continues the process. On the 10 first products, he only likes one, which is a Minecraft Lego set.

Fig. 5. Tkinter window, Minecraft Lego set

After 10 products, the program computes the probability score for the 453 remaining products to have a score of 1 with the Naive Bayes Classification algorithm. Because the user only liked one product so far, the algorithm classifies most of the product to belong to the group 0.

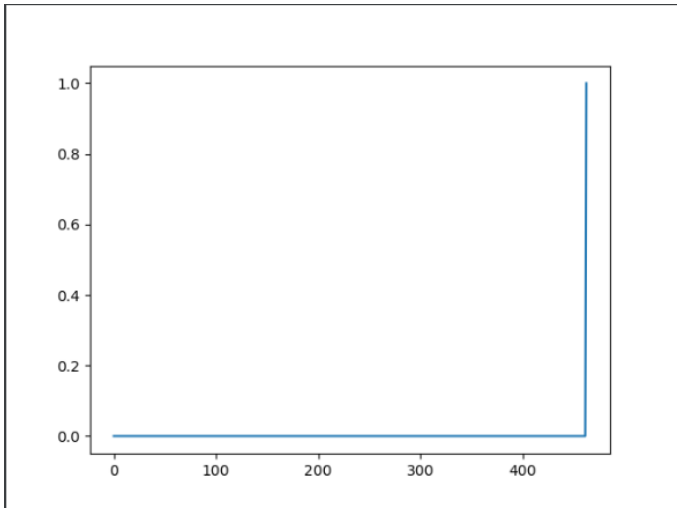

Fig. 6. Predicted Classification Score after 10 products

The program fetches the higher score and adds them to the products dataframe df3 to show them to the user. The user continues to likes and dislikes products and this time found 5 products that he liked. The program executes again the classification algorithm and chooses the 10 next best score to add to the df3.

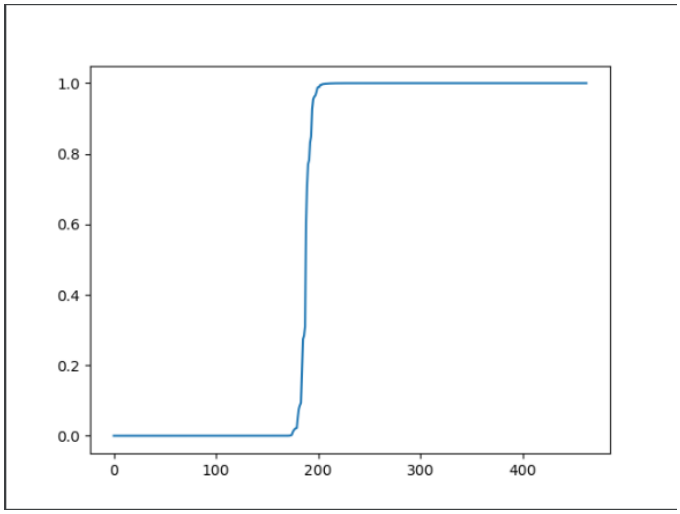

Fig. 7. Predicted Classification Score after 20 products

This time, with more observations, the program can create a new estimation for the predicted scores. Because the user liked more products, the program can with more confidence classifies more products to belong to the category with a score of 1.

For the 21th product, the program seems to have understood that the user likes products similar to the Minecraft Lego dataset from the beginning and proposes another Minecraft Lego set.

Fig. 8. Tkinter Window, another Minecraft Lego set

The user continues to use the program and after a few products, a new classification score is computed. The classification score is computed in the background and it is impossible for the user to notice when this occurs. For him, the experience is lag free.

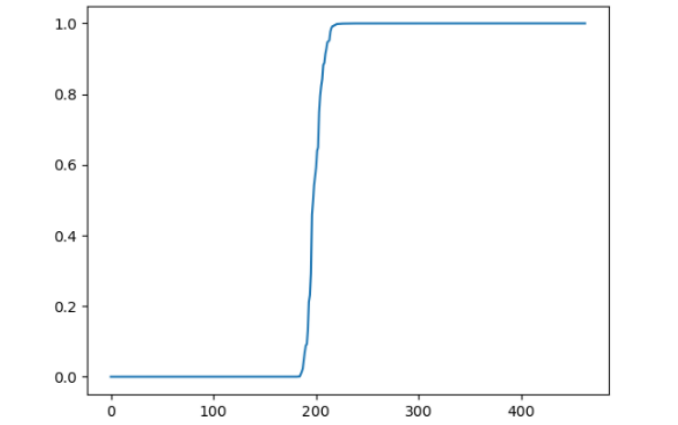

Fig. 9. Predicted Classification Score after 30 products

The Classification algorithm takes into account the new observations and computes new predicted values. Again, it fetches the predicted values with the highest score and proposes new products to the user. This time, the user is shown another Lego set with dinosaur Fossils. He fell in love with the product and decides to offer it as a gift to his nephew.

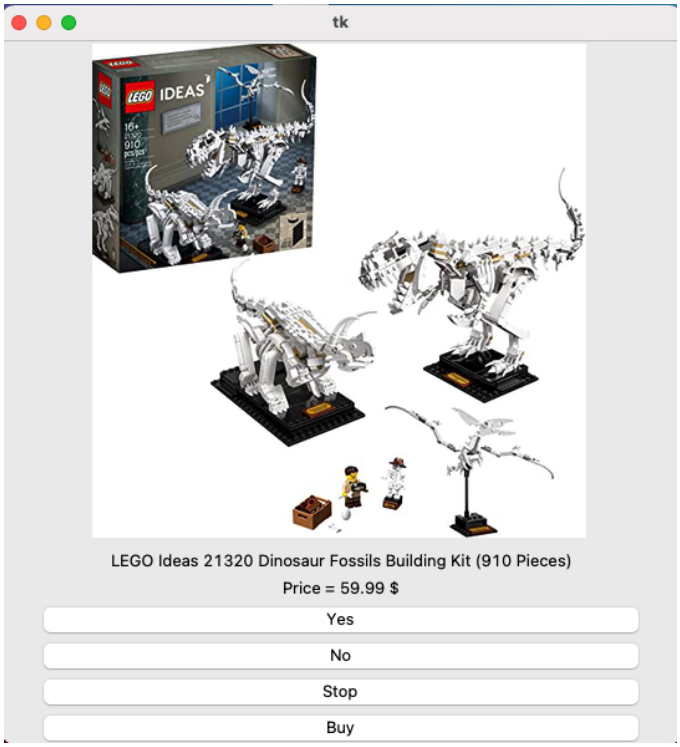

Fig. 10. Tkinter Window, Dinosaur Fossils Lego set

He clicks on ”Buy” and he is directly redirected to the product’s page on Amazon. He thanks the creator of the program for this experience and go on with his life. The full process only took 2 minutes, far less than what he would have taken using directly the website and a lot easier too. The user is impressed by the lag free experience and the accuracy of the products proposed.

VII. Conclusion

This project resulted in a working version of the wanted result. The process is easier than using a regular medium to look for a gift. Therefore, we can affirm that this project has accomplished the goal of overcoming information overload. Furthermore, the program allows the user to bypass the websites and the tracking resulting from their use. On top of that, the Naive Bayes Classification algorithm seems to be greatly adapted to this project and offers meaningful results.

The program can certainly be improved. One way to improve the program can be to implement more general functions so that it can work with multiple databases without having to input manually the column names for the classification algorithm or the categories’ names. It can also be interesting to implement other classification algorithms such as Gaussian Process classification and compare the results between the two algorithms.

The improvement in performance was done using the brain rather than parallel implementation. Indeed, the project was thought in a way that every loop is optimized so that only

relevant information is taken into account. The loops only execute ten times at a time with easy computation. The biggest optimization on performance was done by pre-processing the data and scraping the pictures' URL in advance so that this process had not to be done by the final user.

Overall, this project has attained his objectives and can still be improved visually and be generalize to more databases. At this stage, this is more a prototype than an end product that could be commercialized to the public. Nonetheless, it showed that it was possible with a database and a Naive Bayes Classification algorithm to show meaningful products based on an user's preferences.

VIII. Appendix: Code examples and Algorithms

A. *Welcome_message.py*

```

# Welcome message function that return the input
def welcome():
    print("Welcome to the gift selector !")
    print("------------------------------")
    print("We have 8 gift categories, please select the categories that you are interested in.")
    print("------------------------------")
    print("1. Toys and Games, 2. Clothing, Shoes and Jewelry, 3. Sports, outdoor and Fitness, 4. Video Games")
    print(", 5.Patio, Lawn and Garden, 6. Musical Instrument, 7.Tools and Home, 8. Electronics")
    category = input("Please type the numbers of the categories (eg:12378)")
    return(category)
```

B. *Scrape_n_save.py*

```
import pandas as pd
import numpy as np
from imgscrape import imgscrape

# Create empty dataframe and give path to chromedriver
df3 = pd.DataFrame()
any_var = "/Users/arthur/Desktop/chromedriver"

# Import raw data
df = pd.read_csv('data/data_prog.csv')

# Cleaning - Deleting info we don't need
df = df.drop("Brand",axis=1)
df = df.drop("Available From",axis=1)
df = df.drop("Seller",axis=1)
df = df.drop("Shipping Weight",axis=1)
df = df.drop("Size",axis=1)
df = df.drop("Oversize",axis=1)
df = df.drop("ASIN",axis=1)
df = df.dropna()

# Scrap function
def scrape_img(lst_var):
    var = imgscrape(*lst_var, path=any_var)
    return(var)

# Iterating over dataframe 10 rows at a time and save image into df
def iter_image(df, df3) :
    x = 10
    while x < df.shape[0] :
        for i in range(x-10,x):
            df3 = df3.append((df.iloc[[i]]))
        lst_var_df = df3['URL']
        lst_var = lst_var_df.values.tolist()
        y = scrape_img(lst_var)
        m = np.asarray(y)
        df3["Image"] = m
        df3.to_csv(f'data/data_prog_{x}.csv')
        x+=10
        df3 = pd.DataFrame()

# Call function
iter_image(df, df3)

# Create variables
x=10
liste = []

# Create a list with all the csv
while x < 890 :
    df = pd.read_csv(f'data/data_prog_{x}.csv')
    liste.append(df)
    x+=10

# Concat the csv and do some additional cleaning -> save the end result
result = pd.concat(liste)
result.to_csv(f'data/data_prog_final.csv')
df = pd.read_csv(f'data/data_prog_final.csv')
df = df.drop(df.columns[0],axis=1)
df = df.drop(df.columns[0],axis=1)
df.to_csv(f'data/data_prog_final.csv')
```

C. *plot.py*

```
from matplotlib import pyplot as plt

# plot function
def plot_mlscore(ml_score) :
    ml_score.sort()
    plt.plot(ml_score)
    plt.show()
```

D. *main.py*

```
import pandas as pd
import numpy as np
import welcome_message
import random
import tkinter
from selenium import webdriver
from PIL import Image
import PIL.ImageTk
import requests
from sklearn import preprocessing
from sklearn.naive_bayes import *
from plot import *

# Create variables and create categories with categories name from dataset
categories = [
    "Toys & Games", "Clothing, Shoes & Jewelry", "Sports | Outdoors & Fitness",
    "Video Games", "Patio, Lawn & Garden", "Musical Instruments",
    "Tools & Home Improvement", "Electronics | Computers | Accessories"]
categories_final = []
df2 = pd.DataFrame()
df3 = pd.DataFrame()

# Path to chromedriver
any_var = "/Users/arthurbonetti/Desktop/chromedriver"

```

```
# Import pre processed data from scv
df = pd.read_csv('data/data_prog_final.csv')

# Welcome message + input categories
categories_wanted = str(welcome_message.welcome())

# Save wanted categories in a list
for i in range(0, 9):
    if str(i) in categories_wanted:
        categories_final.append(categories[i - 1])
    else:
        continue

if categories_final == [] :
    for i in range(0, 9):
        categories_final.append(categories[i-1])

# Create new dataframe df2 to keep only categories of interest
for i in categories_final:
    temp_df = df[df['Category'] == i]
    df2 = df2.append(temp_df)

# Reset index
df2 = df2.reset_index(drop=True)

# Select 10 random elements from df2 and save into df3
for i in range(10):
    x = random.randint(0, len(df2)-1)
    df3 = df3.append((df2.iloc[[x]]))

# Drop element in df2 so that we don't choose them again
for i in df3[[]]:
    df2.drop(i)

# New column df3 for ML
df3['Score'] = 0
df3['Ml_score'] = 0

# Keep info we have to show
var = df3['Image'].values.tolist()
var2 = df3['Name'].values.tolist()
var3 = df3['Price'].values.tolist()

# Create Model with count, image, price, description
class Model(object):
    def __init__(self, img, descr, price):
        self.currentObject = 0
        self.img = img
        self.descr = descr
        self.price = price

# Initialise object with first element
count_object = Model(var[0], var2[0], var3[0])

# Object to stock dataframe
class Model_df(object):
    def __init__(self, data_f, data_f2):
        self.data_f = data_f
        self.data_f2 = data_f2

# Create object with current df3
my_df = Model_df(df3, df2)

# Function to compute Naive Bayes Classification algorithm
def ml_score(df3, df2) :
    # Columns name we use as X variables
    dfml = df3[['Price','Net','FBA Fees','LQS','Sellers', 'Rank','Est. Sales","Est. Revenue","Reviews Count","Rating","Weight","Colors","Sizes"]]
    X = dfml.values
    # y is the score value
    y = df3['Score'].values
    # Same variable as before for the test dataset
    X_test = df2[['Price','Net','FBA Fees','LQS','Sellers', 'Rank','Est. Sales","Est. Revenue","Reviews Count","Rating","Weight","Colors","Sizes"]].values
    # Normalize the data
    min_max_scaler = preprocessing.MinMaxScaler()
    min_max_scaler.fit(X_test)
    X = min_max_scaler.transform(X)
    X_test = min_max_scaler.transform(X_test)
    # initialize the Naive Bayes and fit
    clf = GaussianNB()
    model = clf.fit(X=X, y=y)
    # Compute the probabilities for score
    y_proba = model.predict_proba(X_test)[:,1]
    print(y_proba)
    # Save probabilities in new column and plot
    m = np.asarray(y_proba)
    df2["Ml_score"] = m
    plot_mlscore(m)

# Function to update df3
def update_df3(df3, df2, var, var2, var3) :
    print(df2)
    # Add the 10 elements with biggest predicted values from df2 and remove from df2
    for i in range(10) :
        x = df2['Ml_score'].idxmax()
        df3 = df3.append(df2.loc[x])
        df2 = df2.drop(x)

    # Add the new elements to the list used to show the product
    temp = df3['Image'].tail(10).values.tolist()
    temp2 = df3['Name'].tail(10).values.tolist()
    temp3 = df3['Price'].tail(10).values.tolist()

    for i in range(10) :
        var.append(temp[i])
        var2.append(temp2[i])
        var3.append(temp3[i])

    # Reset index and fill empty score values with 0
    df3 = df3.reset_index(drop=True)
    df3['Score'] = df3['Score'].fillna(0)
    # Update object with new df3
    my_df.data_f = df3
    my_df.data_f2 = df2

# Tkinter
# Update window with new infos
def update_tk(var_img, var2, var3, count):
    # Update the product count
    count.currentObject += 1
    # Update infos
    count.img = var_img[count.currentObject]
    count.descr = var2[count.currentObject]
    count.price = var3[count.currentObject]
    # Update window
    im = Image.open(requests.get(count.img, stream=True).raw)
    im = im.resize((400, 400), Image.ANTIALIAS)
    photo = PIL.ImageTk.PhotoImage(image=im)
    canvas.create_image(0, 0, image=photo, anchor=tkinter.NW)
    name.config(text=count.descr)
    price.config(text='Price = ' + str(count.price) + ' $')
    window.mainloop()

# Function button Yes
```

```
def button_yes():
    # Select Score column and update to 1
    my_df.data_f.iloc[count_object.currentObject, -2] = 1
    # Condition for 10 products
    if (count_object.currentObject + 1) % 10 == 0 :
        # Compute classification score and update dataframe
        ml_score(my_df.data_f,my_df.data_f2)
        update_df3(my_df.data_f,my_df.data_f2,var, var2,var3)

    print(my_df.data_f)
    # Update tkinter window
    update_tk(var, var2, var3, count_object)

# Function button No
def button_no():
    # Condition for 10 products
    if (count_object.currentObject + 1) % 10 == 0:
        ml_score(my_df.data_f, my_df.data_f2)
        update_df3(my_df.data_f, my_df.data_f2, var, var2, var3)

    print(my_df.data_f)
    # Update tkinter window
    update_tk(var, var2, var3, count_object)

# Function button stop -> close window
def button_stop():
    window.destroy()

# Function button buy -> open chrome tav to the product URL
def button_buy():
    driver = webdriver.Chrome(executable_path=any_var)
    driver.get(my_df.data_f.iloc[count_object.currentObject, -4])


print(df2)
print(df3)
print(df2.size)
print(df3.size)

# 1st initialization tkinter window
window = tkinter.Tk()
# Get product picture from URL
im = Image.open(requests.get(count_object.img, stream=True).raw)
im = im.resize((400, 400), Image.ANTIALIAS)
# Create a PIL object and add to the canvas
photo = PIL.ImageTk.PhotoImage(image=im)
canvas = tkinter.Canvas(window, width=400, height=400)
canvas.pack()
canvas.create_image(0, 0, image=photo, anchor=tkinter.NW)

# Create labels
name = tkinter.Label(window, text =count_object.descr)
name.pack(anchor=tkinter.CENTER, expand=True)
price = tkinter.Label(window, text ='Price = ' + str(count_object.price) + ' $')
price.pack(anchor=tkinter.CENTER, expand=True)

# Create buttons
btn_yes = tkinter.Button(window, text="Yes", width=50, command=button_yes)
btn_yes.pack(anchor=tkinter.CENTER, expand=True)
btn_no = tkinter.Button(window, text="No", width=50, command=button_no)
btn_no.pack(anchor=tkinter.CENTER, expand=True)
btn_stop = tkinter.Button(window, text="Stop", width=50, command=button_stop)
btn_stop.pack(anchor=tkinter.CENTER, expand=True)
btn_buy = tkinter.Button(window, text="Buy", width=50, command=button_buy)
btn_buy.pack(anchor=tkinter.CENTER, expand=True)
window.mainloop()
```

REFERENCES

[1] https://www.statista.com/statistics/991947/amazons-annual-technology-and-content-expenses/ , last consulted the 22 may 2021

[2] Schmitt, Josephine & Debbelt, Christina Schneider, Frank. (2017). Too much Information? Predictors of Information Overload in the Context of Online-News Exposure. Information Communication and Society. 21. 1151–1167. 10.1080/1369118X.2017.1305427.

[3] https://amzscout.net/product-database/ , last consulted the 22 may 2021