

UK Amazon Bestsellers Dataset

December 17, 2023

Code :

- ▶ Repository: https://github.com/girofleeorientale/amazon_bestellers

BSR: caractéristiques essentielles

- ▶ "The Amazon Best Sellers Rank (BSR) is an important indicator of how well the listed product is sold and it is a common interest to all sellers." (Sharma, Liu 2020)
- ▶ Catégorie "volatile", change toutes les heures / jours
- ▶ Catégorie absente pour certains produits (à vérifier)

Dataset

- ▶ Source: Kaggle
 - ▶ Très déséquilibré (chiffres approximatifs : 2 200 000 articles dont 6 018 marqués comme bestsellers)
 - ▶ Hypothèse : dataset obtenu à partir de la version mobile du site (marquage "bestseller" beaucoup plus rare que dans la version desktop + catégorie "boughtInLastMonth" évidente)
 - ▶ Toutes les colonnes : "asin", "title", " imgUrl", "productURL", "stars", "reviews", "price", "isBestSeller", "boughtInLastMonth", "categoryName"
 - ▶ Colonnes finalement utilisées : "title", "reviews", "price", "boughtInLastMonth", "categoryName"
 - ▶ La colonne de "y" : "isBestSeller"

Une mise à jour du dataset s'impose [FAILED]

- Le dataset ne prend pas en compte les bestsellers dans les "petites" catégories, donc il faudrait une māj à partir de la version desktop
- En essayant de le faire, j'apprends qu'un serveur virtuel avec des IP qui alternent est nécessaire...

```
def is_bestseller_tr(soup):
    try:
        tr_elements = soup.find_all('tr')
        print('RES', tr_elements)

        res = []

        for i, tr in enumerate(tr_elements):
            if 'Best Sellers Rank' in str(tr):
                res.append(str(tr))

        string_split = res[0].split()
        res = ''

        for i, elt in enumerate(string_split):
            if (elt == 'in') and ("span" in string_split[i-1]):
                res = ''
            for x in string_split[i-1]:
                if x.isdigit() == True:
                    res+=x

        res = int(res)
    except AttributeError:
        res = 100

    return (res <= 10)
```

```
</div>

<noscript>
    <img height="1" width="1"
</noscript>

<script>window.ue && ue.count &
<!-- sp:end-feature:csm:body-cl
</div></body></html>
<!--
    |   |
    .___.(.)< (MEOW)
    \_____
----->
<!-- sp:eh:Uel04ntMDNzsMH0atHw+I
```

EMS OUTPUT DEBUG CONSOLE TERMINAL

Train / validation

- ▶ Séparé en train / test 8 : 2
- ▶ Le traitement de test se fait en itérations; à chaque itération, un ensemble de validation est choisi (0.25)
- ▶ Pour le test final, la première itération est prise en compte
- ▶ Pour faciliter certains calculs, le fichier "bestsellers.csv" est créé

Methodologie

- ▶ But : classification binaire
- ▶ Est-ce qu'on veut avoir moins de faux positifs ou de faux négatifs ?
- ▶ Peut-être moins de faux négatifs... (et donc le recall est plus important)
- ▶ En plus, si on vérifie finalement les faux positifs obtenus, il y aura un certain nombre de bestsellers (d'après la définition moins stricte). (Pas de vrais statistiques sur cette hypothèse)

Traitement

- ▶ La colonne "boughtInLastMonth" a l'aspect trop évident. Est-ce qu'elle permet d'avoir "isBestSeller" automatiquement ?
- ▶ Réponse : non (voir corrélation). Mais elle y contribue (si on utilise cette colonne, les quantités des articles prédis comme bestsellers augmentent, mais restent dans les mêmes proportions)
- ▶ (Plus de corrélations entre les colonnes dans l'Annexe; en résumé - les valeurs sont proches de 0 et en dessous de 0.3, donc pas pertinentes)

```
with: df_bought = chunk[chunk['boughtInLastMonth'] > x]:  
  
    df_bought['boughtInLastMonth'] <-> df_bought['isBestSeller']  
if x > θ:  
    moy = 0.2438  
  
if x > 50:  
    moy = 0.2471  
  
if x > 100:  
    moy = 0.2542
```

Traitement

- ▶ Les colonnes "title" et "categoryName" (plus de 150 catégories) sont importantes. Comment encoder leurs valeurs textuels ?
- ▶ Première idée pour "title" : TF-IDF pour toute la colonne, mettre les vecteurs avec les autres valeurs numériques (numpy hstack). Problème de mémoire
- ▶ Question : est-ce qu'il y a des mots qui "vendent" mieux que d'autres ? Avis:

1 Answer

Sorted by: Highest score (default)

- 1 The main question is whether the product name helps predicting whether an item will be returned.... I don't see any clear case where the product name can give this kind of clue, so I simply wouldn't include the product name in the features. But I would probably still include this variable as a one-hot encoded variable, because it's likely that some products are returned more often than others.
- 2

Share Improve this answer Follow

answered Oct 30, 2022 at 16:45

 Erwan
25k 3 14 35

Traitement: "title"

- ▶ On prend le fichier qui contient seulement des bestsellers de l'ensemble d'apprentissage
- ▶ On y applique TF-IDF; on prend les n terms les plus importants
- ▶ Résultats un peu différents pour des n différents

```
anna@anna-IdeaPad-3-15ADA05:~/study/projetML$ python3 dict_from_bestsellers.py
['pack' 'black' 'kids' 'set' 'women' 'white' 'men' 'bag' 'baby' 'girls'
 '10' 'free' 'steel' '12' 'kit' '100' 'outdoor' 'home' 'boys' 'gift'
 'waterproof' 'adjustable' 'car' 'storage' 'water' 'blue' 'unisex' 'soft'
 'holder' 'cm' 'shoes' 'size' 'cover' 'high' 'box' 'large' 'garden' 'uk'
 'inch' 'light' 'hair' 'portable' 'dog' 'cotton' 'metal' 'mini' 'cable'
 'plastic' 'gifts' 'kitchen']
```

Traitement: "categoryName"

- ▶ Première approche : tout encoder avec label_encoder.fit_transform. Pas réussi
- ▶ Deuxième : calculer le prix médian pour chaque catégorie et diviser en 7 groupes
- ▶ Hypothèse : le prix d'un bestseller des articles moins chers a un comportement particulier
- ▶ Cette dernière hypothèse est plus efficace que la séparation en 7 groupes
- ▶ Pas réussi : approche "isFMCG"

Traitement: "reviews", "boughtInLastMonth", "stars"

- ▶ Normalisation de "reviews" ((*"valeur initiale"* - mean)/stdev). Efficace
- ▶ Normalisation similaire de "bouhgtnLastMonth" : pas efficace, j'ai laissé les valeurs telles quelles
- ▶ Stars : colonne finalement mise à part

Algorithme (les 3 dans sklearn)

- ▶ Essentiel : Régression Logistique
- ▶ Essayé, presque aussi efficace : Multinomial Naive Bayes. (Valeurs négatives non traitées)
- ▶ Essayé, beaucoup moins efficace : SVM. Très erroné même sur l'ensemble de validation

Premier résultat (régression logistique, mode "standard")

- On va noter $a = \text{accuracy}$, $p = \text{precision}$, $r = \text{recall}$.

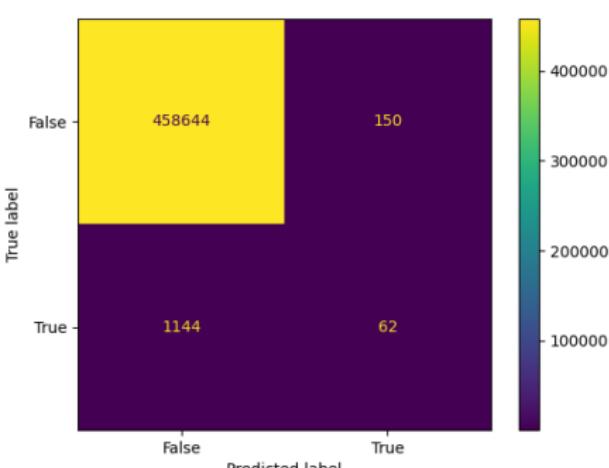
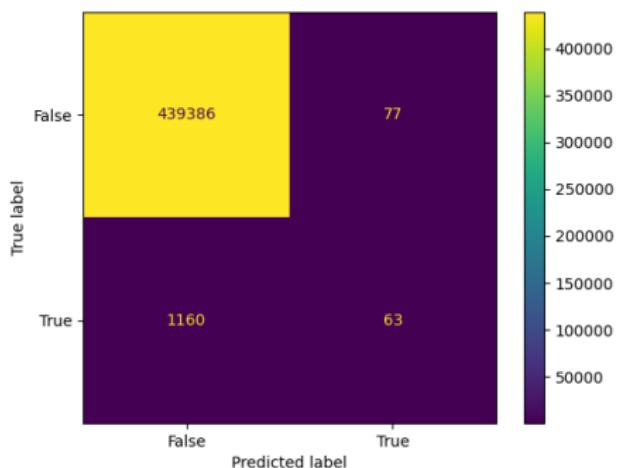


Figure: Left (validation dans train): $a = 0.99$, $p = 0.45$, $r = 0.05$. Right (test): $a = 0.99$, $p = 0.29$, $r = 0.05$

Algorithme (suite)

- ▶ Une des idées "how to deal with highly imbalanced data" :
- ▶ Pour l'entraînement, un dataset "équilibré" est créé (dans mon cas: tous les bestsellers (4812) + autant de "non-bestsellers")
- ▶ Idée initiale : centroïdes; mon implémentation : aléatoire

Deuxième résultat (régression logistique, "sampled") 1/2

- Ici on tient compte de la catégorie "les moins chers", mais on n'utilise pas "title".

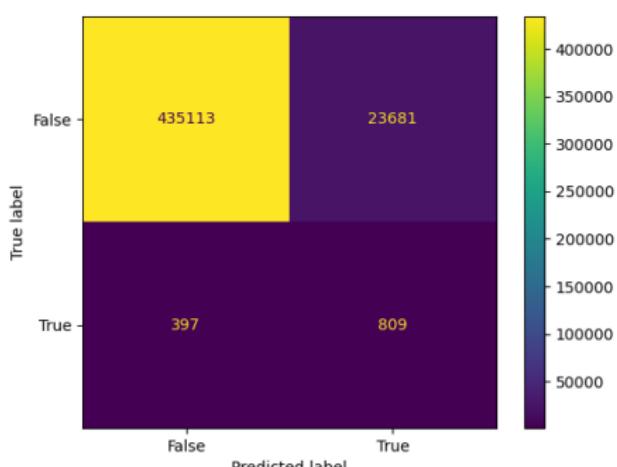
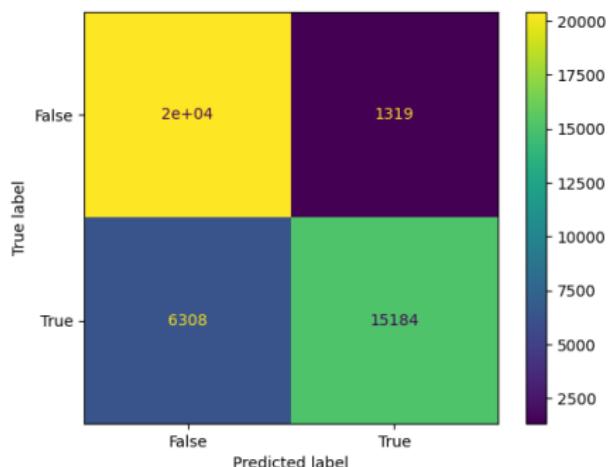


Figure: Left (validation dans train): $a = 0.82$, $p = 0.92$, $r = 0.7$. Right (test): $a = 0.94$, $p = 0.03$, $r = 0.67$

Deuxième résultat (régression logistique, "sampled") 2/2

- Ici on tient compte de la catégorie "les moins chers" et on utilise "title".

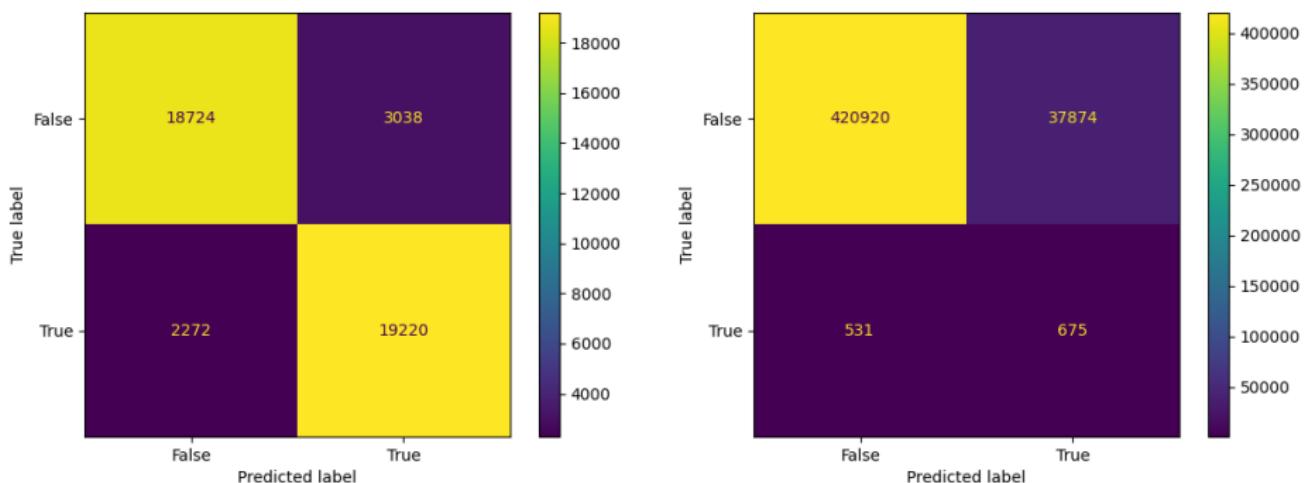


Figure: Left (validation dans train): $a = 0.87$, $p = 0.86$, $r = 0.89$. Right (test): $a = 0.91$, $p = 0.017$, $r = 0.55$

Discussion

- ▶ La catégorie à prédire est très instable
- ▶ L'influence de "title" est peut-être surestimée
- ▶ Dans le modèle, on aimerait bien diminuer le nombre de false positives (centroïdes restent à essayer peut-être...)

Code :

- ▶ TF-IDF pour construire un vocabulaire des termes plus importants

```
import pandas as pd
import numpy as np

from sklearn.feature_extraction.text import TfidfVectorizer


def dictionary_from_bestsellers (path, n):
    """
    Gives n most frequent terms in bestsellers titles
    """

    colnames = ['asin','title','imgUrl','productURL','stars','reviews','price',\
    'isBestSeller','boughtInLastMonth','categoryName']

    df = pd.read_csv(path, names=colnames)

    df_titles = df['title'].copy()

    tfidf_vectorizer = TfidfVectorizer(max_df=0.95, min_df=2, stop_words='english')
    tfidf = tfidf_vectorizer.fit_transform(df_titles)
    importance = np.argsort(np.asarray(tfidf.sum(axis=0)).ravel())[::-1]
    tfidf_feature_names = np.array(tfidf_vectorizer.get_feature_names_out())

    return tfidf_feature_names[importance[:n]]
```

Code :

► Début d'une régression logistique

```
17 logreg = LogisticRegression(random_state=16)
18 logreg_1 = LogisticRegression(random_state=16)
19
20 first = True
21 bestsellers_array = []
22
23 df_y_test_total = []
24 data_y_pred = []
25
26 mean_ = 385
27 rev_stdev = 3827
28
29 for chunk in pd.read_csv('train.csv', chunksize = 500000):
30     bestsellers_current= chunk.loc[chunk['isBestSeller'] == True]
31     bestsellers_array.append(bestsellers_current)
32
33 # back to df, 4812 items
34 bestsellers = pd.concat(bestsellers_array)
35
36 for chunk in pd.read_csv('train.csv', chunksize = 100000):
37     df = chunk[['reviews', 'boughtInLastMonth', \
38                 'price', 'categoryName', 'isBestSeller']].copy()
39
40     # get a df without bestsellers
41     df_intermediate = df.loc[df['isBestSeller'] == False]
42
43     df_no_bs = df_intermediate[['reviews', \
44                                 'boughtInLastMonth', 'price', 'categoryName', 'isBestSeller']]
45     df_with_bs = bestsellers[[['reviews', \
46                               'boughtInLastMonth', 'price', 'categoryName', 'isBestSeller']]
47
48     df_no_bs['reviewNormalized'] = (df_no_bs['reviews'] - mean_)/rev_stdev
49     df_with_bs['reviewNormalized'] = (df_with_bs['reviews'] - mean_)/rev_stdev
```

Code :

► Régression logistique (suite)

```
237     X = (df_united[feature_cols])
238
239     X_train, X_test, y_train, y_test = train_test_split\
240         (X, y, test_size=0.25, random_state=16)
241
242     df_y_test_total.append(y_test)
243
244     logreg.fit(X_train, y_train)
245     y_pred = logreg.predict(X_test)
246
247     if first:
248         logreg_1.fit(X_train, y_train)
249         first = False
250
251     data_y_pred.append(y_pred)
252
253
254     final_y_test = pd.concat(df_y_test_total)
255     arr = np.concatenate(data_y_pred).ravel()
256
257     cm = confusion_matrix(final_y_test, arr, labels=logreg.classes_)
258     disp = ConfusionMatrixDisplay(confusion_matrix=cm,
259                                     display_labels=logreg.classes_)
260     disp.plot()
261     plt.show()
262
263
264
265     kmeans = KMeans(n_clusters=100).fit(df)
266     centroids = kmeans.cluster_centers_
267     closest, _ = pairwise_distances_argmin_min(kmeans.cluster_centers_, df)
268
269     df_test = pd.read_csv('test.csv')
```

Sources

- ▶ Best Seller Rank (BSR) to Sales: An empirical look at Amazon.com.
Sharma, Liu et al.
- ▶ ce premier lien
- ▶ ce deuxième lien
- ▶ ce troisième lien

Annexe 1/4 : Ce que j'ai essayé et ça n'a pas marché

- ▶ SVM
- ▶ TF-IDF "global"
- ▶ Normaliser la catégorie "stars"
- ▶ Même pas essayé, mais reste une idée : novelty + outliers detection

Annexe 2/4 Minor insights from data

- ▶ Caractère particulier de la catégorie "stars"
- ▶ Distribution intéressante du prix médian

Annexe 3/4 Corrélations

- ▶ reviews et price: moyenne -0.018
- ▶ reviews et isBestSeller: 0.095
- ▶ isBestSeller et price: -0.01

Annexe 4/4 MNB

- Ici "reviews" n'est pas normalisé.

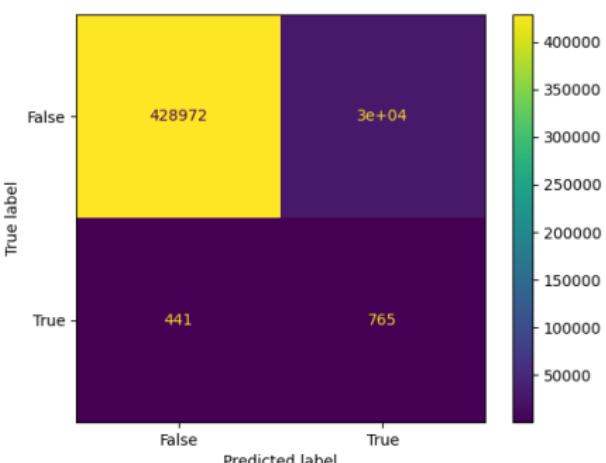
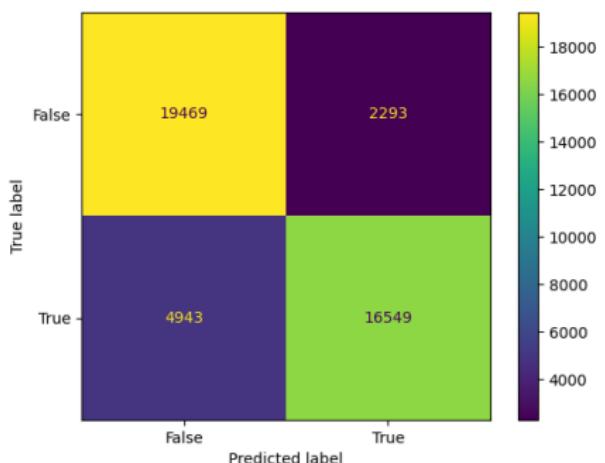


Figure: Left (validation dans train): $a = 0.83$, $p = 0.87$, $r = 0.77$. Right (test): $a = 0.93$, $p = 0.025$, $r = 0.63$

Merci!