Notation d'un objet informatique

28.12.2021

Kouyoumdjian Pierre

Marynowicz Michael

Vue d'ensemble

I. Le contexte du projet et la description des tâches.
II. Les données utilisées
III. Détails sur la conception
IV. Evaluation
V. Analyse des erreurs
VI. Conclusion et amélioration possibles

Le contexte du projet et la description des tâches

Le but de ce projet est de classifier l'avis d'un consommateur concernant un objet informatique.

Pour ce faire nous traitons les reviews de cette manière :

1. Extraction des données à partir du csv
2. Transformation des notes en catégories
3. Vectorisation des reviews
4. Délimitation des zones de test et d'entraînements pour la svm
5. Applications des 3 SVM
    a. création des 3 SVM
    b. entraînement des SVM avec leurs labels correspondants
    c. test des SVM
    d. test sur un avis extérieur au fichier
    e. génération des statistiques sur chaque SVM puis sur le fichier global

Dans ce projet nous nous sommes beaucoup aidé des librairies sklearn, nltk, pandas que ce soit pour les classifications ou pour les statistiques.

Les données utilisées

Dans un premier temps, nous avons passé beaucoup de temps à chercher une base de données conséquente et aussi facile à traiter. Car ici le principal but n'est pas la récupération des données d'une base de données mais bien le traitement de ces informations.

Nous avons tout d'abord commencé à essayer de récupérer les avis directement sur un site tel que backmarket ou amazon mais nous nous sommes vite heurté à des problèmes peu compréhensibles qui nous faisaient perdre du temps.

En continuant nos recherches nous avons récupéré une base de données sur kaggle. (ref : https://www.kaggle.com/PromptCloudHQ/amazon-reviews-unlocked-mobile-phones?select=Amazon_Unlocked_Mobile.csv)

Cette dernière contient plus de 400 000 avis sur différents produits informatiques. Cette large base de données nous a permis de bien entraîner notre algorithme comme nous le détaillerons dans une future partie.

De plus son traitement a été très simple car nous l'avons téléchargée au format csv, nous n'avions donc plus cas récupérer les colonnes que nous voulions et le nombre de ligne grace a la commande : data[nomDeLaColonne].head(nbrDeLignes)

Ce qui nous donne une liste que nous traiterons ensuite.

Représentation de notre base de données :

Nom du produit	Note	Avis
, MP3/MP4 Player and microSD Slot - White International Version/Warranty	5	I love it!
, MP3/MP4 Player and microSD Slot - White International Version/Warranty	5	Very practical and user-friendly phone. I am very satisfied with it.
, MP3/MP4 Player and microSD Slot - White International Version/Warranty	5	Sharp and classy phone
, MP3/MP4 Player and microSD Slot - White International Version/Warranty	5	Shipped quickly and was exactly what I expected!
, MP3/MP4 Player and microSD Slot - White International Version/Warranty	4	Works wonderful for the price!

De plus, cette base de données est très intéressante car elle regroupe plusieurs objets informatiques ce qui permet d'élargir le vocabulaire de notre algorithme. Dans certaines bases de données ce n'est pas le cas et nous aurions pu avoir un langage spécifique pour un produit en particulier.

Détails sur la conception

I. Analyse des adjectifs

Dans un premier temps, nous avons eu l'idée d'identifier les adjectifs dans chaque reviews. Les adjectifs sont stockés et nous extrayons ceux qui sont évalués comme neutre.

Ces derniers sont réévalués avec leurs contextes c'est-à-dire que nous effectuons une sorte de trigram afin d'évaluer l'adjectif avec le mot avant et le mot après.

Une fois les adjectifs caractérisés nous effectuons un calcul de polarité. Nous souhaitions donner une note entre 0 et 5. Pour ce faire nous faisions pour chaque adjectif : (sentence.sentiment.polarity+1)*2.5

En effet, sentence.sentiment.polarity nous donne une note entre -1 et 1, en ajoutant 1 nous avons une note entre 0 et 2 puis en multipliant par 2.5 nous avons une note entre 0 et 5

En faisant la moyenne des notes obtenues par tous les adjectifs nous retournons la note finale du produit.

Cette méthode est longue et ne nous donne pas un résultat aussi précis que nous le souhaitons.

Le code de cet essais est present sur le github et le fichier se nomme "essais2.py"

De plus ayant trouvé une base de données de plus de 400 000 avis notés nous avons trouvé ça dommage de ne pas profiter pleinement de toutes ces données avec un autre algo.

II. Utilisation d'une SVM

Comme nous l'avons dit précédemment, nous avons cherché un modèle qui exploite pleinement les atouts de notre base de données.

Nous avons donc décidé d'utiliser une SVM qui est facile d'utilisation grâce à la bibliothèque sklearn.

Tout d'abord nous avons changé la représentation de la note dans notre base de données. Désormais ce ne sont plus des notes de 1 à 5 mais 3 catégories :

“very good” = 5, “neutre” = [2-4], “very bad” = 1

Nous avons choisi cette répartition car il y a en moyenne plus de notes égales à 5 et 1 que d’autres notes. Donc pour avoir une bonne répartition nous avons jugé que cela était le mieux.

Puis nous avons changé la représentation des reviews par des vecteurs. Pour cela nous avons utilisé CountVectorizer, disponible directement dans la librairie sklearn, afin de pouvoir les passées comme arguments a la fonction train_test_split.

Finalement, nous appelons la fonction train_test_split, issue de sklearn, qui va diviser notre tableau en sous-ensemble d'entraînement et de test. Nous lui affectons 70% d'entraînement car tout simplement nous nous somme dit que plus il s'entraîne plus il sera performant.

Une fois notre SVM créé nous n’avions plus qu'à l'entraîner et à appliquer l’algo grâce à la fonction “predict”.

Pour voir les performances de notre algo nous avons affiché la matrice de confusion, qui n'était pas vraiment satisfaisante.

Nous avons cherché un moyen d'être plus précis et nous avons essayé, comme dans le premier essai, de prendre seulement les adjectifs et non pas la reviews entière. Or les statistiques étaient moins bonnes, nous avons donc abandonné cette idée.

En analysant la base de données plus précisément nous avons observé certaines lignes qui n'étaient pas en anglais, or des lignes dans une autre langue que le français faussait totalement les statistiques de notre algo.

Pour palier à ce problème nous avons décidé de traduire les phrases en anglais dynamiquement, nous avons vite constaté que cette méthode prenait beaucoup trop de temps. Nous avons donc fait un programme qui va filtrer les reviews et qui va stocker les reviews anglaise dans un nouveau fichier. Une fois ce programme fini (ce qui a bien duré 20 minutes) nous avons notre nouvelle base de données entièrement en langue anglaise.

En analysant à nouveau la matrice nous voyons une nette amélioration, mais nous n'étions pas encore complètement satisfait de nos résultats.

Nous nous sommes donc une fois de plus remis en question et nous avons observé que de nombreux avis contenaient des fautes d’orthographes importantes qui pouvait entraver le bon fonctionnement de l’algorithme.

Grâce à la fonction TextBlob issue de la librairie textblob nous avons pu corriger les fautes dynamiquement or nous nous sommes vite rendu compte que c'était beaucoup trop lent et qu’au final les résultats n'étaient pas meilleurs nous avons donc abandonné cette idée de corriger les fautes.

A la suite de cette étape a commencé une longue phase de recherche afin de vraiment bien cerner les algorithmes en profondeur afin de pouvoir améliorer notre algorithme.

III. One vs One

Nous avons trouvé le One vs One et le One vs Rest car nous voulions éviter la prédiction multiclasse.

En effet, une SVM est plus performante lorsqu'elle est binaire.

Nous avons décidé de prendre One vs One car la classification était plus précise que pour One vs Rest. Elle demande plus de modèle mais vu qu'on classe en 3 catégories nous n'avons pas besoin de plus de 3 modèles.

Il consiste à prendre le modèle (ici notre svm) et à l'entraîner sur 2 classes, puis à recréer un modèle vierge et l'entraîner sur 2 autres classes.

Dans notre cas nous avons entraîner 3 modèles :

- Un qui compare les catégories "very good" avec les "neutre"
- Un qui compare les "neutre" et les "very bad"
- Un qui compare les "very good" et les "very bad"

La dernière étape consiste à vérifier dans quelle classe se trouve l'entrée en pratiquant une prédiction avec chacun des 3 modèles.

Si les 3 modèles retournent : neutre, neutre, very bad

On place l'entrée dans la catégorie neutre.

Nous avons dû trier la zone d'entraînement pour que chaque modèle prenne les bonnes entrées. Si le modèle vérifie "very good" et "neutre" alors on lui donne la zone d'entraînement qui ne possède pas "very bad", cela nous permet d'avoir une classification binaire. Puis comme pour la zone d'entraînement on a dû trier la zone de test de la même manière.

En regardant la documentation, nous avons remarqué que la SVM prend en paramètre decision_function_shape qui permet avec l'argument "ovo" de faire un One vs One.

Après quelques tests, ce paramètre nous donne de moins bons résultats que notre implémentation à la main. Nous avons donc décidé de maintenir notre One vs One maison.

Evaluation

Les statistiques sans la mise en place du one vs one montre bien l'efficacité de celui-ci :

```
matrice de confusion without one vs one =
 [[ 550  105  133]
 [ 108  534   14]
 [ 234   36 1286]]
classification report without one vs one =
              precision    recall  f1-score   support

      neutre       0.70      0.62      0.65       892
    very bad       0.81      0.79      0.80       675
   very good       0.83      0.90      0.86      1433

    accuracy                           0.79      3000
   macro avg       0.78      0.77      0.77      3000
weighted avg       0.79      0.79      0.79      3000
```

Certes ces résultats sont bons, mais pour nous, nous avions une trop grande disproportion entre les différentes catégories. Exemple 0.65 de f1-score pour les neutres et 0.86 pour les very good. C'est donc cette raison qui nous a poussé à continuer nos recherches pour améliorer notre algo

Statistiques de nos 3 SVM pour le One vs One :

- ➔ Ces statistiques sont parfois très bons ce qui peut sembler bizarre comme nous en avions parlé en cours, or après de nombreuses vérifications la fonction train_test_split va bien séparer la zone de test de la zone d'application donc ces statistiques ne sont pas erroné par le fait que nous appliquons notre algo sur les mêmes lignes sur lesquelles nous nous sommes entraîné.

Very good and neutre :

- Matrice de confusion :

```
very good and neutre =
 [[ 656  131]
 [ 271 1267]]
```

- Classification report :

```
very good and neutre =
              precision    recall  f1-score   support

      neutre       0.83      0.71      0.77       927
   very good       0.82      0.91      0.86      1398

    accuracy                           0.83      2325
   macro avg       0.83      0.81      0.81      2325
weighted avg       0.83      0.83      0.82      2325
```

Analyse :

Nous pouvons constater ici que les statistiques de ces 2 catégories ont nettement augmentées. De plus si nous avons des disproportions ici ce n'est pas grave car grâce au One vs One nous allons comparer ce résultats avec les autres SVM.

Avoir de meilleurs résultats entre ces 2 catégories est très compliqué car elles sont relativement proches. Pour avoir de meilleures statistiques il faudrait se référer à un vocabulaire très complet pour étudier les mots dans leurs contextes à partir de ce vocabulaire. Dans cette circonstances word2vec aurait sans doute été utile malgrés qu'il ne contienne qu'un vocabulaire assez restreint (~44 000 mots)

Neutre and very bad :

- Matrice de confusion :

```
neutre and very bad =
 [[831 126]
 [ 96 549]]
```

- Classification report :

```
neutre and very bad =
              precision    recall  f1-score   support

      neutre       0.87      0.90      0.88       927
    very bad       0.85      0.81      0.83       675

    accuracy                           0.86      1602
   macro avg       0.86      0.85      0.86      1602
weighted avg       0.86      0.86      0.86      1602
```

Analyse : Ici nous pouvons voir de très bons résultats aussi avec une faible disproportion.

Very good and very bad :

- Matrice de confusion :

```
very good and very bad =
 [[ 623   41]
 [  52 1357]]
```

- Classification report :

```
very good and very bad =
              precision    recall  f1-score   support

    very bad       0.94      0.92      0.93       675
   very good       0.96      0.97      0.97      1398

    accuracy                           0.96      2073
   macro avg       0.95      0.95      0.95      2073
weighted avg       0.96      0.96      0.96      2073
```

Analyse : Dans ce cas la différenciation entre une reviews very bad et une reviews very good est nette contrairement aux deux exemples précédents.

C'est pourquoi nous pouvons constater d'excellents résultats.

Global :

- Matrice de confusion :

```
[[ 898  176  196]
 [ 283 1225   45]
 [ 501  105 2571]]
```

- Classification report :

```
statistiique totaux =
              precision    recall  f1-score   support

     neutre       0.71      0.53      0.61      1682
    very bad       0.79      0.81      0.80      1506
   very good       0.81      0.91      0.86      2812

    accuracy                           0.78      6000
   macro avg       0.77      0.75      0.76      6000
weighted avg       0.78      0.78      0.77      6000

la precision sur l'ensemble des reviews traitées est de : 78.23333333333333%
```

Analyse : On voit un taux de précision de 78% ce qui est bien. Or nous pouvons voir que cette disproportion persiste même avec le one vs one et que la précision a baissé. Ceci est une limite de notre algorithme que nous détaillerons dans la partie suivante

Analyse des erreurs

1) Une source de conflit dans notre code pourrait être engendrée par un commentaire trop court. En effet nous avons mis en place un one vs one pour de nouveaux commentaires qui seront insérés afin de noter directement le produit grâce au commentaire laissé par l'utilisateur. Or si le commentaire est trop court notre algo pourra générer un conflit ou se tromper de catégorie car il a été entraîné avec des commentaires avec des contextes.
2) De plus si le commentaire n'est pas en anglais la classification risque de poser problème.
3) Comme précisé dans la partie précédente nos statistiques ne sont pas vraiment meilleures qu'avant la mise en place du one vs one, ce problème a été le problème majeur de notre projet. Et ça a été aussi le plus frustrant car malgré que nous ayons bien compris le réel problème nous n'avons pas trouvé de solution pour le résoudre.

   En effet ce problème provient du fait que pour "regrouper" nos 3 svm afin de tester le fichier sans filtrer les catégories préalablement, nous passons aux SVM spécialisées X_test entièrement. Exemple la première SVM est spécialisé dans les catégories "neutre" et "very good" or nous lui passons X_test qui peut contenir des reviews caractérisées de "very bad" or ne connaissant pas cette catégorie, la première SVM va dire que cette reviews est "neutre". Dans la matrice de confusion de la deuxième SVM nous pouvons constater qu'elle fait des petites erreurs de classification (la SVM classifie la reviews de "neutre" alors qu'elle est "very bad") ces deux erreurs accumulées et affectées à chaque catégorie biaise nos statistiques.

Conclusion et améliorations possibles

Nous pouvons en conclure que ce projet nous a permis de mieux comprendre les algorithmes de classification en particulier l'algorithme de SVM.

Les améliorations que nous pouvons apporter à notre projet serait d'utiliser Word2Vec pour vectoriser nos reviews comme étudié dans le cours afin de prendre plus précisément les mots dans leurs contextes, malheureusement nous n'avons pas réussi à passer d'une système qui utilise countVectorizer à word2vec à cause de problème lié à la taille des listes.

De plus pour améliorer l'analyse de notre métrique, la visualisation de la courbe ROC et precision-recall auraient été un plus pour voir où se trouve le problème de notre modèle ou de notre implémentation.

La principale amélioration que nous aurions aimé apporter à temps pour ce projet est la correction de la dernière erreur citée précédemment.

En effet nous avons pensé à ne pas passer X_test entierement et passer des X_test triés au SVM spécialisés or cette solution nous a posé de nombreux bugs. Nous pensons que cette solution est la bonne car le fait de passer X_test entierement est sans doute la source de nos soucis il faut donc modifier ce paramètre.