ChAllenge By MathA Data

Rapport projet initiation aux Data sciences

Product data classification into *product type codes*

Rakuten France Multimodal Product Data Classification

Participants :

E. Hamza

F. Quentin

Sommaire :

1. **Contexte**

Ce challenge porte sur la classification des produits. L'objectif est de prédire le code type (prdtypecode) de chaque produit en utilisant des données textuelles (désignation et description) et des images issues du catalogue de Rakuten France.

Pour notre projet nous ne tiendrons pas compte des images à des fins de simplification.

2. **Objectif du projet**

- Prétraiter le texte
- Convertir les données dans une matrice TF-IDF
- Appliquer différentes approches sur ces données pour prédire la variable
- Comparer les performances
- Conclure sur l'approche la plus appropriée

3. **Prétraitement du texte**

Pour ce défi, Rakuten France propose environ 99 000 listes de produits au format CSV, incluant à la fois l'ensemble d'entraînement (84 916) et l'ensemble de test (13 812) pour lequel on ne possède pas les labels. Le jeu de données comprend 4 features:

- Désignations de produits,
- Descriptions de produits,
- Images de produits,
- Codes de produits correspondants.

1. "X_train_update.csv" : un tableau contenant des échantillons d'entraînement avec la description textuelle et la référence du fichier image associé.
2. "y_train_CVw08PX.csv" : un tableau contenant les 84 916 codes des produits.
3. "X_test_update.csv" : un tableau contenant des échantillons de test.
4. "images.zip" : un fichier qui regroupe toutes les images, il ne sera pas utilisé pour cette partie

Les fichiers "X" CSV sont organisés de la manière suivante :

- **Id** : un identifiant entier pour le produit, utilisé pour associer le produit à son code de produit.
- **designation** : le titre du produit, un texte court résumant le produit.
- **description** : un texte plus détaillé décrivant le produit. Ce champ peut contenir des valeurs manquantes car tous les utilisateurs ne le renseignent pas.
- **productid** : un identifiant unique pour le produit.
- **imageid** : un identifiant unique pour l'image associée au produit.

De plus, on remap les valeurs des classes pour simplifier la lecture et le développement, tout en diminuant les valeurs qui sont calculées (26<<2905) ce qui peut accélerer le calcul.

Obtention de la matrice TF-IDF

La conversion des données textuelles en une matrice TF-IDF (Term Frequency-Inverse Document Frequency) est une étape cruciale pour transformer les textes en une forme numérique utilisable par les algorithmes de machine learning.

Le TF-IDF est une technique qui permet de mesurer l'importance d'un terme dans un document par rapport à un corpus de documents. Cette importance est calculée en multipliant deux métriques : la fréquence d'un terme dans un document (TF) et l'inverse de la fréquence du terme dans le corpus (IDF).

Pour ce projet, nous avons utilisé la bibliothèque **scikit-learn** pour générer la matrice TF-IDF. Les étapes sont les suivantes :

- **Concaténation des champs textuels** : Pour chaque produit, nous avons combiné les champs "designation" et "description" en un seul champ textuel.
- **Nettoyage des textes** : Nous avons converti les textes en minuscules, supprimé les caractères spéciaux, les stop words et effectué une lemmatisation pour réduire les mots à leur forme racine.
- **Calcul du TF-IDF** : Utilisation de **TfidfVectorizer** de **scikit-learn** pour transformer les textes en une matrice TF-IDF.

```
from sklearn.feature_extraction.text import TfidfVectorizer
# Concaténation des champs textuels (fonction create_text utilisée dans le git)
for i in range(xtrain.shape[0]):
    xtrain['text'][i] = create_text(xtrain['designation'][i],
xtrain['description'][i])

# Nettoyage des textes (exemple)
xtrain['text'] = xtrain['text'].apply(lambda text : lower_case(text))  # Conversion
en minuscules
***.apply(lambda text : remove_accent(text)) # Suppression des accents
***.apply(lambda x:' '.join([word for word in x if word not in stopwords]))
#stopwords
***.apply(lambda text : remove_htmltags(text))  # Suppression des encodages htmls
***.apply(lambda text : keeping_essentiel(text))

# Calcul du TF-IDF
tfidf_vectorizer = TfidfVectorizer(max_features=5000)
X_tfidf = tfidf_vectorizer.fit_transform(X_train['text'])
```

5. **Modélisation**

Pour la modélisation, nous avons expérimenté avec plusieurs algorithmes de classification pour prédire les codes de produit (prdtypecode). Les algorithmes choisis incluent :

- **Support Vector Machines (SVM)**
- **Régression Logistique (Logistic Regression)**
- **k-Nearest Neighbors (KNN)**
- **Random Forest**
- **Voting Classifier**

Support Vector Machines (SVM)

Le SVM est un classifieur linéaire puissant, particulièrement efficace dans les espaces de grande dimension. Il fonctionne bien pour la classification de textes grâce à sa capacité à trouver un hyperplan optimal qui sépare les classes.

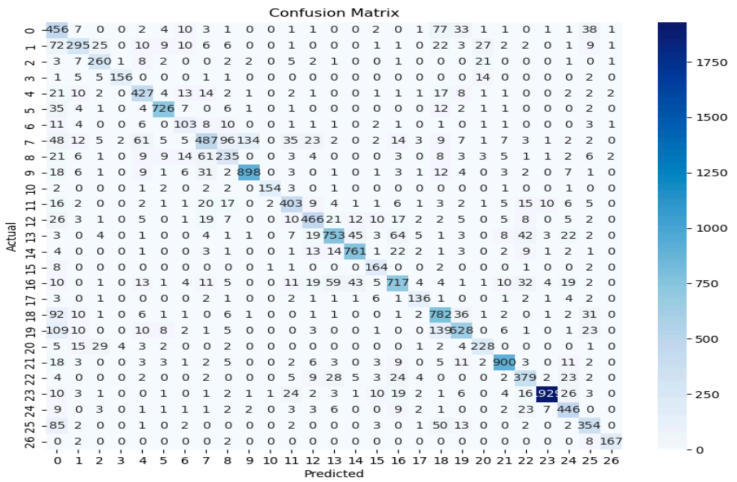
```
from sklearn.svm import SVC from sklearn.metrics import accuracy_score
svm = SVC(kernel='linear', C=1) svm.fit(X_tfidf, y_train) y_pred_svm =
svm.predict(X_tfidf_test) accuracy_svm = accuracy_score(y_test,
y_pred_svm)
```

Régression Logistique

La régression logistique est un modèle de classification linéaire simple qui peut être très efficace pour la classification de texte en raison de sa capacité à gérer les problèmes de classification binaire et multicatégorie.

```
from sklearn.linear_model import LogisticRegression logreg =
LogisticRegression(max_iter=1000) logreg.fit(X_tfidf, y_train)
y_pred_logreg = logreg.predict(X_tfidf_test) accuracy_logreg =
accuracy_score(y_test, y_pred_logreg)
```

Matrice de confusion



On remarque sur la matrice de confusion que les classes qui présentent plus de données pour l'apprentissage ont plus de réussite à être détectée que les autres (classe 23~2585)

k-Nearest Neighbors (KNN)

Le KNN est un algorithme de classification non paramétrique. Il classifie les nouveaux échantillons en fonction des k plus proches voisins dans l'espace des caractéristiques.

```
from sklearn.neighbors import KNeighborsClassifier knn =
KNeighborsClassifier(n_neighbors=5) knn.fit(X_tfidf, y_train) y_pred_knn
= knn.predict(X_tfidf_test) accuracy_knn = accuracy_score(y_test,
y_pred_knn)
```

Random Forest

Le Random Forest est un ensemble de méthodes qui construit plusieurs arbres de décision et combine leurs prédictions pour améliorer la performance et réduire le surapprentissage.

```
from sklearn.ensemble import RandomForestClassifier rf =
RandomForestClassifier(n_estimators=100) rf.fit(X_tfidf, y_train)
y_pred_rf = rf.predict(X_tfidf_test) accuracy_rf =
accuracy_score(y_test, y_pred_rf)
```

Utilisation de GridSearchCV avec RandomForest

Afin d'optimiser les performances de notre modèle RandomForest, nous avons utilisé GridSearchCV pour rechercher les meilleurs hyperparamètres. GridSearchCV effectue une recherche exhaustive sur une spécification de grille de paramètres, en utilisant la validation croisée pour évaluer chaque combinaison de paramètres.

Meilleurs Paramètres et Performance du Modèle

Après l'optimisation, les meilleurs paramètres obtenus sont les suivants :

- `n_estimators`: 200
- `max_depth`: None
- `min_samples_split`: 5
- `min_samples_leaf`: 1

Grâce à ces paramètres optimisés, la précision du modèle RandomForest a été améliorée à 8 points environ (de 0.71 à 0.79).
Cette méthode a aussi été utilisée sur KNN

Voting Classifier

Le Voting Classifier combine les prédictions de plusieurs modèles de base pour améliorer les performances globales. Nous avons utilisé un vote de type "hard" où chaque modèle contribue de manière égale à la prédiction finale.

```
from sklearn.ensemble import VotingClassifier
voting_clf = VotingClassifier(estimators=[('logreg', logreg), ('knn',
knn), ('rf', rf) ], voting='hard') voting_clf.fit(X_tfidf, y_train)
y_pred_voting = voting_clf.predict(X_tfidf_test) accuracy_voting =
accuracy_score(y_test, y_pred_voting)
```

6. **Comparatif**

Les performances des différents modèles ont été comparées en termes de précision (accuracy) sur l'ensemble de test. Les résultats sont présentés dans le tableau ci-dessous :

Model	Precision
SVM	0.22
Logistic Regression	0.78
KNN	0.78
Random Forest	0.79
Voting Classifier	0.80

Conclusion

À partir des résultats obtenus, nous observons que le Voting Classifier offre les meilleures performances avec une précision de 0.80. Ce modèle combine les forces de plusieurs algorithmes de base, ce qui lui permet de mieux généraliser sur les données non vues.