Votre prénom et nom : Hermann Ky

Votre numéro d’étudiant : 03216182

**Sujet** : Home Depot Product Search Relevance (Pertinence de la recherche de produits Home Depot)

Lien Kaggle pour les données : <https://www.kaggle.com/competitions/home-depot-product-search-relevance/data>

Lien de Travail GitHub : <https://github.com/lowpien/HomeDepotProductSearchRelevance>

**Produit Finale Fonctionnel Projet : Home Depot Product Search Relevance**

Pour toute requête de recherche saisie par le client, je dois trouver les produits les plus pertinents et les montrer à l'utilisateur par ordre de pertinence. D’un point de vue commercial, quelques points doivent être pris en compte. Premièrement, les produits doivent être classés. Ainsi, même parmi les produits les plus pertinents, nous devons être capables de déterminer lequel est le plus pertinent. Deuxièmement, il existe une contrainte de temps, c'est-à-dire que les résultats doivent être affichés en quelques secondes.

Formulation par apprentissage automatique du problème commercial :

La tâche peut être formulée comme suit : étant donné une recherche et un produit, trouvez le score de pertinence entre eux, c'est-à-dire la pertinence de ce produit par rapport à la requête de recherche en question. Supposons donc que ma machine ait appris à prédire le score de pertinence pour une paire (requête de recherche, produit). Désormais, pour toute recherche effectuée par l'utilisateur, je peux calculer le score de pertinence pour cette même recherche associée à tous les produits de ma base de données et montrer les (disons) les 10 meilleurs résultats au client.

Ainsi, si j'ai beaucoup de données étiquetées, c'est-à-dire beaucoup de paires (requête-recherche, produit) avec leurs scores de pertinence, je peux alors poser cela comme un problème de (Machine Learning) ML supervisé. Et c'est exactement ce que j'ai fait dans cette étude de cas. Les données que j'ai utilisées sont fournies par Home Depot pour le concours Kaggle home-depot-product-search-relevance

Dans un moteur de recherche de commerce électronique réel, calculer le score de pertinence de chaque produit pour une recherche donnée n'est pas possible, car sur tout site Web de commerce électronique classique, le nombre de produits est très important et est donc coûteux en termes de calcul et prendrais beaucoup de temps.

Dans un premier temps, nous récupérons quelques produits candidats en utilisant un modèle de récupération simple qui permet une évaluation rapide des requêtes. Et dans la deuxième phase, un modèle d’apprentissage automatique plus précis est utilisé pour reclasser ces produits.

Pour expliquer cela, considérons que nous disposons d’un ensemble de 1 00 000 produits. Et la requête de recherche est « lampe solaire ». Le premier modèle de récupération plus simple récupérera quelques produits candidats. Ce modèle peut être aussi simple qu'un opérateur AND entre les mots de recherche et le texte du produit. Donc ici, un modèle régi par l'opérateur AND obtiendra tous les produits avec les mots « solaire » et « lampe ». Disons qu'il récupère quelque 500 produits. Désormais, en plus de ces 500 produits, nous pouvons exécuter notre algorithme complexe d'apprentissage automatique et calculer le score de pertinence pour chaque paire (recherche, produit) où notre terme de recherche « lampe solaire » reste constant et les produits varient. Et maintenant, nous les reclassons en fonction de leur score de pertinence et montrons les meilleurs produits à l'utilisateur. C'est ce qu'on appelle Apprendre à classer (LTOR).

Aperçu des données

Les données se présentent sous la forme de quatre fichiers CSV : train.csv, test.csv, attributes.csv et product\_description.csv. A télécharger à partir d'ici :

<https://www.kaggle.com/competitions/home-depot-product-search-relevance/data>

Données de formation (train.csv) : nous avons un total de 74 067 lignes pour la formation dans le fichier train.csv. Une partie des données se présente comme indiqué ci-dessous, chaque ligne correspondant à une paire de (search\_term, product\_title) et le score de pertinence indiquant dans quelle mesure ce produit est pertinent pour la recherche. Un score de pertinence est un nombre réel compris entre 1 et 3 (1 pour non pertinent et 3 pour une correspondance parfaite). Pour chaque produit, nous avons généralement plusieurs lignes, c'est-à-dire plusieurs requêtes de recherche avec leur score de pertinence. Le product\_uid est un identifiant unique pour chaque produit.

Une image contenant texte, capture d’écran, Police, noir

Description générée automatiquement

Le product\_description.csv contient une description textuelle de chaque produit. Et le fichier attributes.csv contient des informations supplémentaires sur un sous-ensemble de produits.

La requête de recherche se présente sous la forme d'une chaîne et le produit est représenté par des données textuelles dont nous disposons sous la forme de son titre, de sa description et de certains attributs. Dans le monde réel, nous pouvons également avoir d'autres fonctionnalités pour le produit, comme son image, ses notes, etc., mais ici, nous n'avons que du texte avec lequel travailler.

Les scores de pertinence des données ont été marqués manuellement. Chaque paire (recherche, produit) a été évaluée par au moins trois évaluateurs humains, puis le score de pertinence final a été pris comme moyenne.

Données de test (test.csv) : un fichier test.csv contenant environ 166 000 lignes pour lesquelles nous devons prédire les scores de pertinence. Environ la moitié des produits de train se trouvent dans l’ensemble de test. Le diagramme de Venn pour les deux ensembles est présenté ci-dessous.

****

Analyse exploratoire des données : partie 1

Dans cette section, je présente l’analyse des données disponibles. J'ai enregistré le fichier train.csv en tant que dataframe dans la variable dt\_train.

def show\_data\_info(dt):

print(f"dt shape: \n {dt.shape} \n")

print("bd columns: \n", dt.columns, "\n")

dt.info()

return dt.head(2)

dt\_train = pd.read\_csv("f:/train.csv", encoding='latin-1')

show\_data\_info(dt\_train)

print(dt\_train)

print(dt\_train.head())

print(dt\_train.describe())

print("Il ya au Total {} produit ".format(len(dt\_train.product\_title.unique())))

print("Il ya au Total {} requête de recherche ".format(len(dt\_train.search\_term.unique())))

print("Il ya au Total {} product\_uid".format(len(dt\_train.product\_uid.unique())))

Une image contenant texte, capture d’écran

Description générée automatiquement

Statistiques sur le nombre de notes dans l'ensemble dt\_train

def count\_SR():

high\_SR = [i for i in dt\_train['relevance'] if i == 3.00]

midle\_SR = [i for i in dt\_train['relevance'] if i > 1.00 and i < 3.00]

low\_SR = [i for i in dt\_train['relevance'] if i == 1.00]

return [len(high\_SR), len(midle\_SR), len(low\_SR)]

plt.bar(['haut\_sr', 'moyen\_sr', 'bas\_sr'], count\_SR(), label='Score de pertinence', color='blue')

plt.xlabel('Notes')

plt.ylabel('Nombres')

plt.title('Nombres de notes')

plt.legend()

plt.show()

Une image contenant texte, capture d’écran, diagramme, ligne

Description générée automatiquement

La colonne id ne sert à rien. Le product\_uid est un identifiant unique de chaque produit et est important lors de la fusion des données d'autres fichiers texte. Il n'y a pas de valeurs nulles dans les données du train.

Scores de pertinence

Les scores de pertinence se sont avérés de nature catégorique. En effet, la plupart des scores correspondent à la note moyenne de 3 évaluateurs et chaque évaluateur ne pouvait donner qu'une note intégrale de 1, 2 ou 3.

sns.countplot(x="relevance", data=dt\_train)

plt.title('Répartition des scores de pertinence')

plt.show()

Une image contenant texte, capture d’écran, Police, diagramme

Description générée automatiquement

La majorité des produits avaient un score de pertinence élevé, c'est-à-dire que la plupart des produits étaient pertinents par rapport à la requête. En raison de la nature catégorique des scores de pertinence, un modèle de classification peut également être construit sur cette base si cela facilite le classement final. Les scores {1,25, 1,5, 1,75, 2,25, 2,5, 2,75} sont apparus à des fréquences très basses.

Fichier Test

dt\_test = pd.read\_csv("f:/test.csv", encoding='latin-1')

show\_data\_info(dt\_test)

print("Il ya au Total {} produit ".format(len(dt\_test.product\_title.unique())))

print("Il ya au Total {} requête de recherche ".format(len(dt\_test.search\_term.unique())))

print("Il ya au Total {} product\_uid".format(len(dt\_test.product\_uid.unique())))

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Les attributs

Pour les produits, nous disposons d'informations textuelles sous la forme d'un titre du produit, d'une description du produit et d'attributs. Le titre du produit se trouve dans le fichier train.csv. La description et les attributs sont extraits respectivement des fichiers product\_description.csv et attributs.csv.

Une instance du fichier attributs.csv est présentée ci-dessous. Le champ « nom » représente le nom de l'attribut et « valeur », le texte de l'attribut.

dt\_attributes = pd.read\_csv("f:/attributes.csv", encoding='latin-1')

show\_data\_info(dt\_attributes)

print(dt\_attributes.head ())

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

graphique montrant le nombre de product\_uids apparaissant dans les deux ensembles dt\_test et dt\_train

venn2([set(dt\_train["product\_uid"]), set(dt\_test["product\_uid"])], set\_labels=('train', 'test'), set\_colors=('green', 'blue'))

plt.show()

Une image contenant cercle, capture d’écran, diagramme, texte

Description générée automatiquement

le graphique représente le nombre de search\_terms apparaissant dans les deux ensembles dt\_test et dt\_train

venn2([set(dt\_train["search\_term"]), set(dt\_test["search\_term"])], set\_labels=('train', 'test'), set\_colors=('green', 'blue'))

plt.show()

Une image contenant cercle, capture d’écran, diagramme, texte

Description générée automatiquement

Description :

dt\_descriptions = pd.read\_csv("f:/product\_descriptions.csv", encoding='latin-1')

show\_data\_info(dt\_descriptions)

print("Il ya au Total {} product\_uid ".format(len(dt\_descriptions.product\_uid.unique())))

print("Il ya au Total {} product\_descriptions ".format(len(dt\_descriptions.product\_description.unique())))

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Le product\_description.csv contient la description des produits. Contrairement aux attributs, nous avons des descriptions pour tous les produits dans le fichier train.csv.

Attributes data

Contient les champs product\_uid, name et value

show\_data\_info(dt\_attributes)

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Nettoyage des données

C’était l’une des parties les plus importantes de toute cette étude de cas. Une grande quantité de temps a été consacrée à l'observation du texte et à la réalisation des étapes de prétraitement nécessaires pour le rendre adapté à la modélisation. De nombreuses techniques issues du traitement du langage naturel ont été utilisées.

Une motivation très simple expliquant pourquoi le prétraitement du texte est si important dans ce projet est que nous essayons en quelque sorte de trouver la similitude entre la requête de recherche et le texte du produit. Considérez la recherche comme « seau de 5 gal » et le titre du produit comme « seau noir Leaklite de 5 gal ». Si nous utilisions une mesure telle que le nombre de mots communs, alors la similarité, dans ce cas, serait nulle car il n'y a aucun mot en commun. Maintenant, avec un peu de prétraitement, notre recherche et notre titre deviennent respectivement « seau de 5 gal » et « seau noir Leaklite de 5 gal ». Maintenant, notre score de similarité est de 3 car nous avons maintenant 3 mots en commun. Ainsi, peu de prétraitement peut faire une énorme différence.

Prétraitement de base

Les étapes de prétraitement de base impliquaient la mise en minuscules, la suppression des caractères spéciaux, la suppression des mots vides et la radicalisation.

De nombreux titres de produits contiennent les dimensions du produit. Un mode de représentation très courant est « 8 pi x 5 pi x 8 pi ». La même chose est observée dans la description du produit.

Dans product\_title, les unités sont principalement représentées par {in. lb. carré lb. oz. fille. mph}, mais dans product\_description et search\_term, les unités sont représentées de plusieurs manières. Par ex. pouce comme {in. en pouces} et en gallons en {gal. gallon de filles}. Et par conséquent, ces unités doivent être standardisées.

En termes de recherche, il existe des erreurs courantes lors de la spécification des mesures. Quelques exemples sont « 4shelves », « 9x12 », « g135 », « 5gal », « 1/2 in by 12 ». Il fallait donc séparer les chiffres des mots.

Étapes de prétraitement :

Fonction pour supprimer les doublons, Récupération de la plus grande partie commune dans le champ du nom sous forme de puce au lieu de puce0 x y z, Creation du champ product\_attributes à partir du nom et de la valeur

Fusion des attributs qui partagent le même identifiant en un seul et concaténer les product\_attributes correspondants ensemble, Recuperation de la partie commune du nom et supprime les jetons en double. Joindre la table dt\_attributes à la table dt\_descriptions en utilisant l'attribut commun product\_id

def remove\_duplicates(string):

lits\_tokens = []

[lits\_tokens.append(str(\_)) for \_ in string.split() if \_ not in lits\_tokens]

return ' '.join(lits\_tokens)

dt\_attributes['name'] = [\_[:6] if 'bullet' in str(\_).lower() else \_ for \_ in dt\_attributes['name'].tolist()]

dt\_attributes['product\_attributes'] = dt\_attributes['name'] + ' ' + dt\_attributes['value']

dt\_attributes = dt\_attributes.drop(['name', 'value'], axis=1)

dt\_attributes = dt\_attributes.groupby('product\_uid').aggregate({'product\_attributes': lambda \_ : ' '.join(\_.astype(str))})

dt\_attributes['product\_attributes'] = [remove\_duplicates(\_) for \_ in dt\_attributes.product\_attributes.tolist()]

dt\_des\_attr = pd.merge(dt\_descriptions, dt\_attributes, on='product\_uid', how='left')

dt\_des\_attr.info()

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Remplacer toutes les valeurs Null par ''

dt\_des\_attr['product\_attributes'].fillna('', inplace = True)

Suppression des phrases mal copiées dans le champ des descriptions de produits, y compris les descriptions en HTML et les liens.

strings = ['br',

'src',

'href',

'alt',

'please visit'

'Click here to review our return policy for additional information regarding returns',

'Click here to see Home Depot',

'Click here for our Project Guide',

'Click here for our Buying Guide',

'Click on the More Info tab to download',

'CLICK HERE to create your own collection',

'Click Here for details on the services',

'Click Here for Ideas and Designs',

'Click Here for a Demo of the Design',

'Click Here to learn more about',

'CLICK HERE to view our',

'Click below to visit our',

'Click here to purchase a sample of this',

'click on the link to get started',

'Click image to enlarge',

'https://www.ryobitools.com/nation',

'http://www.homedepot.com/ApplianceDeliveryandInstallation',

'http://itemvideo-dev.microsite.homedepot.com/111414/26P/online\_BB\_banner\_111114.jpg',

'http://www.homedepot.com/p/Rev-A-Shelf-Door-Mounting-Kit-5WB-DMKIT/202855698']

for string in strings:

dt\_des\_attr['product\_description'] = [\_.lower().replace(string.lower(), '') for \_ in dt\_des\_attr.product\_description]

dt\_des\_attr.head(5)

Une image contenant capture d’écran, texte, Police

Description générée automatiquement

Creation d’un nouveau champ contenant à la fois la description du produit et ses valeurs d'attribut, suppression de deux anciens champs

dt\_des\_attr['product\_description\_attributes'] = dt\_des\_attr['product\_description'] + ' ' + dt\_des\_attr['product\_attributes']

dt\_des\_attr = dt\_des\_attr.drop(['product\_description', 'product\_attributes'], axis=1)

dt\_des\_attr.head(5)

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

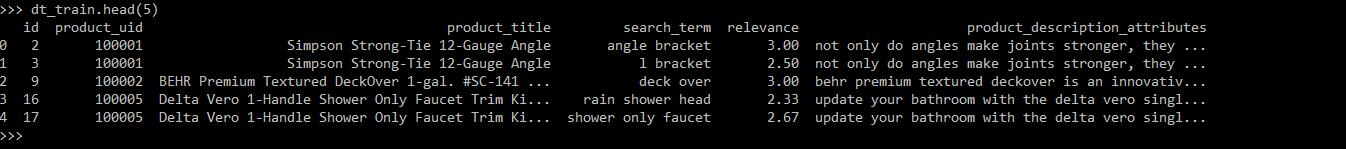
concaténer l'ensemble dt\_train avec dt\_des\_attr

dt\_train = pd.merge(dt\_train, dt\_des\_attr, on='product\_uid', how='left')

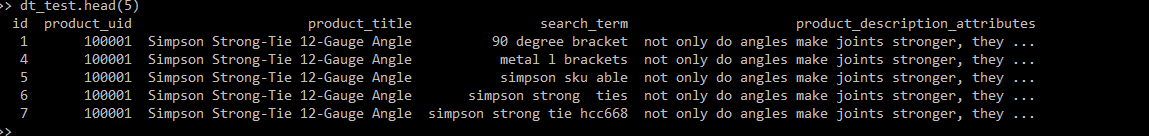
joindre le fichier dt\_test avec le fichier dt\_des\_attr

dt\_test = pd.merge(dt\_test, dt\_des\_attr, on='product\_uid', how='left')

dt\_train.head(5)



dt\_test.head(5)



Fonctions de données propres

obtenir les données du fichier .json pour corriger les fautes d'orthographe

spell\_check = json.load(open('C:/Users/User/spell\_check.json', 'r'))

def spell\_fix(string):

for (k,v) in spell\_check.items():

string = string.replace(k, v)

return string

Séparez les nombres des mots : « 9x12 » → « 9 x 12 », « 5gal » → « 5 gal »…

Amenez toutes les unités de mesure à une représentation cohérente. Par ex. {fille. gals gallon} tous sont convertis en « gallon ». De même, {ft fts foots foot foots} sont convertis en « pieds ».

#Nous n'incluons pas les unités (ft. lb. sq. ...) car le point final sera pris en charge par

# re.split(r'\W+', sent) fonction dans la fonction de prétraitement

# Dans la description et le titre, 'inch' se trouve généralement comme 'in.' où comme dans search\_term,

# on le trouve à la fois comme « in » et « in ». nous traitons donc les deux cas séparément.

#fonction de prétraitement pour les termes de recherche

#Nous n'utilisons plus de stemming ni de mots vides pour le moment. Nous utiliserons le stimming après correction des termes de recherche

#'in.','in' en inch est pris en charge lors de l'étape de prétraitement

#'inches' en inch sera pris charge dans le stemming.

# Prétraitement des données

def standardize\_units(text):

if text is None:

return ""

text = " " + text + " "

text = re.sub('( gal | gals | galon )', ' gallon ', text)

text = re.sub('( ft | fts | feets | foot | foots )', ' feet ', text)

text = re.sub('( squares | sq )', ' square ', text)

text = re.sub('( lb | lbs | pounds )', ' pound ', text)

text = re.sub('( oz | ozs | ounces | ounc )', ' ounce ', text)

text = re.sub('( yds | yd | yards )', ' yard ', text)

return text

def preprocessing(sent):

if pd.isna(sent):

return ""

sent = sent.replace('in.', ' inch ')

words = re.split(r'\W+', sent)

words = [word.lower() for word in words]

res = re.sub("[A-Za-z]+", lambda ele: " " + ele[0] + " ", ' '.join(words))

cleaned = standardize\_units(res)

cleaned = ' '.join(cleaned.split()) # Supprime les espaces supplémentaires

return cleaned

def preprocessing\_search(sent):

if pd.isna(sent):

return ""

sent = sent.replace('in.', ' inch ')

words = re.split(r'\W+', sent)

words = [word.lower() for word in words]

res = re.sub("[A-Za-z]+", lambda ele: " " + ele[0] + " ", ' '.join(words))

res = standardize\_units(res)

res = res.replace(' in ', ' inch ')

cleaned = ' '.join(res.split()) # Supprime les espaces supplémentaires

return cleaned

Suppression des mots vides

Cette fonction ne fonctionne que sur les champs de description et d'attribut

def remove\_stopwords(string):

return ' '.join([w for w in string.split() if not w in stop\_words])

Correction de fautes de frappe pour les champs search\_term, product\_title et product\_description\_attributes

En parcourant les termes de recherche, il a été constaté qu'ils contenaient de nombreuses fautes d'orthographe, comme « tondeuse » était orthographié comme « mowe », « toilettes » comme « toiled », etc. Cela est compréhensible, car ces recherches sont saisies par les utilisateurs. et dans la recherche en ligne, de telles fautes d’orthographe sont courantes. Mais c'était un gros problème, car des mots mal orthographiés pouvaient rendre des fonctionnalités telles que le nombre de mots communs très inefficaces. Par conséquent, pour corriger cela, j’ai utilisé un correcteur orthographique.

correcteur orthographique

def correct\_spell(s, spell\_check\_dict):

for tyto in spell\_check\_dict:

s = s.replace(tyto, spell\_check\_dict[tyto])

return s

dt\_train['search\_term'] = dt\_train['search\_term'].map(lambda x:spell\_fix(x))

dt\_train['product\_title'] = dt\_train['product\_title'].map(lambda x:spell\_fix(x))

dt\_train['product\_description\_attributes'] = dt\_train['product\_description\_attributes'].map(lambda x:spell\_fix(x))

show\_data\_info(dt\_train)

dt\_train.head(5)

Une image contenant texte, capture d’écran

Description générée automatiquement

Normalisation du champ search\_term

dt\_train['search\_term'] = [standardize\_units(\_) for \_ in dt\_train.search\_term]

Standardisation des champs product\_title et product\_description\_attributes

dt\_train['product\_title'] = [standardize\_units(\_) for \_ in dt\_train.product\_title]

dt\_train['product\_description\_attributes'] = [standardize\_units(\_) for \_ in dt\_train.product\_description\_attributes]

Suppression des mots vides dans le champ product\_description\_attributes

dt\_train['product\_description\_attributes'] = [remove\_stopwords(\_) for \_ in dt\_train.product\_description\_attributes]

show\_data\_info(dt\_train)

dt\_train.head(5)

Une image contenant texte, capture d’écran

Description générée automatiquement

Test de fichier propre

Correction des fautes de frappe dans les termes de recherche

dt\_test['search\_term'] = dt\_test['search\_term'].map(lambda x:spell\_fix(x))

dt\_test['product\_title'] = dt\_test['product\_title'].map(lambda x:spell\_fix(x))

dt\_test['product\_description\_attributes'] = dt\_test['product\_description\_attributes'].map(lambda x:spell\_fix(x))

show\_data\_info(dt\_test)

dt\_test.head(5)

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Normalisation du champ search\_term

dt\_test['search\_term'] = [standardize\_units(\_) for \_ in dt\_test.search\_term]

standardization champs product\_title et product\_description\_attributes

dt\_test['product\_title'] = [standardize\_units(\_) for \_ in dt\_test.product\_title]

dt\_test['product\_description\_attributes'] = [standardize\_units(\_) for \_ in dt\_test.product\_description\_attributes]

Suppression des mots vides dans le champ product\_description\_attributes

dt\_test['product\_description\_attributes'] = [remove\_stopwords(\_) for \_ in dt\_test.product\_description\_attributes]

show\_data\_info(dt\_test)

dt\_test.head(5)

Une image contenant texte, capture d’écran

Description générée automatiquement

Prétraitement supplémentaire :

Fonction qui permet d'observer l'influence de la longueur du champ sur la pertinence

def correlation(dt\_sample, dt\_field, transform=True):

x\_ar = np.array(dt\_sample[dt\_field].map(lambda x:len(str(x).split())).astype(np.int64)) if transform else dt\_sample[dt\_field]

y\_ar = np.array(dt\_sample['relevance'])

plt.plot(x\_ar, y\_ar, 'bo')

Droite de régression

m, b = np.polyfit(x\_ar, y\_ar, 1)

plt.plot(x\_ar, m \* x\_ar + b,'r')

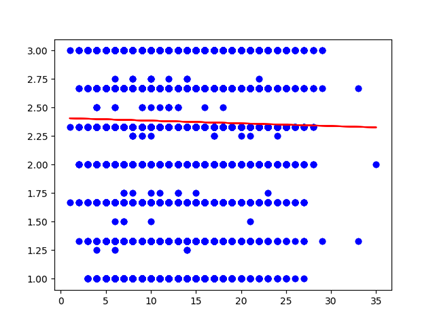
plt.show()

Fichier dt\_train brut n’ont transformé

dt\_raw\_train = pd.read\_csv('f:/train.csv', encoding='latin-1')

Representation du champ product\_title

correlation(dt\_raw\_train, 'product\_title')



Representation du champ search\_term

correlation(dt\_raw\_train, 'search\_term')

Une image contenant ligne, capture d’écran, Tracé, diagramme

Description générée automatiquement

Champ des attributs de description du produit

correlation(dt\_train, 'product\_description\_attributes')

Une image contenant capture d’écran, texte, ligne, nombre

Description générée automatiquement

Fonction pour créer des fonctionnalités à partir de l'occurrence de mots dans search\_term

def str\_common\_tokens(sentence\_1, sentence\_2):

return sum(1 for word in str(sentence\_2).split() if word in set(str(sentence\_1).split()))

def str\_common\_word(sentence\_1, sentence\_2):

return sum(1 for word in str(sentence\_2) if word in set(sentence\_1))

def set\_shared\_words\_whole(row\_data):

return str\_common\_tokens(row\_data[0], row\_data[1])

def set\_shared\_words\_part(row\_data):

return str\_common\_word(row\_data[0], row\_data[1])

dt\_train['len\_of\_querry'] = [len(\_.split()) for \_ in dt\_train['search\_term'].values]

dt\_train['shared\_words\_whole\_st\_pt'] = [set\_shared\_words\_whole(\_) for \_ in dt\_train[['search\_term','product\_title']].values]

dt\_train['shared\_words\_whole\_st\_pdat'] = [set\_shared\_words\_whole(\_) for \_ in dt\_train[['search\_term','product\_description\_attributes']].values]

dt\_train['shared\_words\_part\_st\_pt'] = [set\_shared\_words\_part(\_) for \_ in dt\_train[['search\_term', 'product\_title']].values]

dt\_train['shared\_words\_part\_st\_pdat'] = [set\_shared\_words\_part(\_) for \_ in dt\_train[['search\_term', 'product\_description\_attributes']].values]

Similarité de search\_term et product\_title

dt\_train['similarity'] = [Levenshtein.ratio(\_[0], \_[1]) for \_ in dt\_train[['search\_term', 'product\_title']].values]

show\_data\_info(dt\_train)

dt\_train.head(5)

Une image contenant capture d’écran, texte

Description générée automatiquement

Fonctionnalités de l'ensemble de test

dt\_test['len\_of\_querry'] = [len(\_.split()) for \_ in dt\_test['search\_term'].values]

dt\_test['shared\_words\_whole\_st\_pt'] = [set\_shared\_words\_whole(\_) for \_ in dt\_test[['search\_term', 'product\_title']].values]

dt\_test['shared\_words\_whole\_st\_pdat'] = [set\_shared\_words\_whole(\_) for \_ in dt\_test[['search\_term', 'product\_description\_attributes']].values]

dt\_test['shared\_words\_part\_st\_pt'] = [set\_shared\_words\_part(\_) for \_ in dt\_test[['search\_term', 'product\_title']].values]

dt\_test['shared\_words\_part\_st\_pdat'] = [set\_shared\_words\_part(\_) for \_ in dt\_test[['search\_term', 'product\_description\_attributes']].values]

similarité de search\_term et product\_title

dt\_test['similarity'] = [Levenshtein.ratio(\_[0], \_[1]) for \_ in dt\_test[['search\_term', 'product\_title']].values]

show\_data\_info(dt\_test)

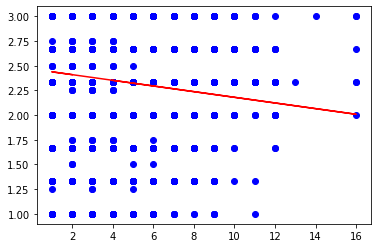
dt\_test.head(5)

Une image contenant capture d’écran, texte

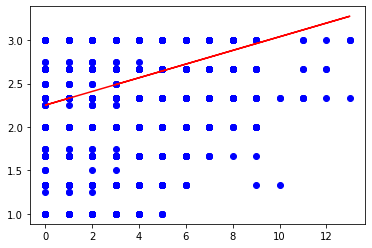
Description générée automatiquement

Reevaluation de la qualité des fonctionnalités avec la fonction de corrélation en utilisant l'ensemble dt\_train

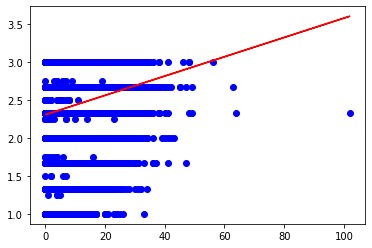
correlation(dt\_train, 'len\_of\_querry', transform=False)



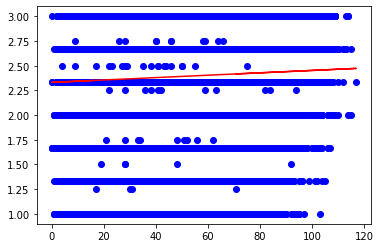
correlation(dt\_train, 'shared\_words\_whole\_st\_pt', transform=False)



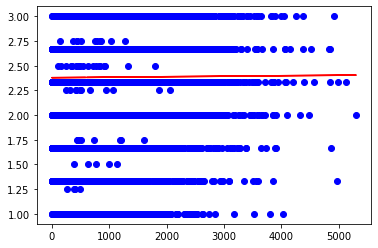
correlation(dt\_train, 'shared\_words\_whole\_st\_pdat', transform=False)



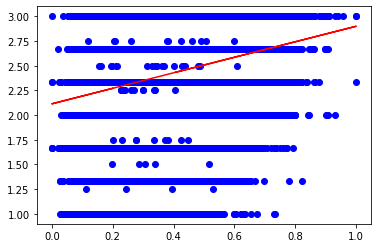
correlation(dt\_train, 'shared\_words\_part\_st\_pt', transform=False)



correlation(dt\_train, 'shared\_words\_part\_st\_pdat', transform=False)



correlation(dt\_train, 'similarity', transform=False)



Suppression de la fonctionnalité shared\_words\_part\_st\_pdat dans les ensembles de train et de test

sdt\_train = dt\_train.drop(['shared\_words\_part\_st\_pdat'],axis=1)

sdt\_test = dt\_test.drop(['shared\_words\_part\_st\_pdat'],axis=1)

Fractionner les données

supprimer les colonnes de texte en ne laissant que les colonnes contenant des fonctionnalités

sdt\_train = sdt\_train.drop(['product\_title','search\_term','product\_description\_attributes'],axis=1)

définir x\_train et y\_train

y\_train = sdt\_train['relevance'].values

X\_train = sdt\_train.drop(['id','relevance'], axis=1).valuesdt\_train.head(5)

show\_data\_info(sdt\_train)

sdt\_train.head(5)

Une image contenant texte, capture d’écran

Description générée automatiquement

Dans l'ensemble de test, nous supprimons également tous les autres champs, ne laissant que les champs avec des fonctionnalités

X\_test = sdt\_test.drop(['id','product\_title','search\_term','product\_description\_attributes'],axis=1).values

L'identifiant du test ici est l'identifiant des paires search\_term et titre du produit

id\_test = sdt\_test['id']

show\_data\_info(sdt\_test)

sdt\_test.head(5)

Une image contenant capture d’écran, texte

Description générée automatiquement

Régresseur de forêt aléatoire (Random forest regressor)

Réaliser une formation sur l'ensemble de données avec le modèle "Random forest" fourni dans la bibliothèque sklearn

from sklearn.ensemble import RandomForestRegressor

ensemble de paramètres est référencé à partir d'une autre source

rfr = RandomForestRegressor(n\_estimators=30, n\_jobs=-1, random\_state=17, max\_depth=10)

rfr.fit(X\_train, y\_train)

y\_pred = rfr.predict(X\_test)

Les données Train et Test

from sklearn.preprocessing import StandardScaler

from sklearn.linear\_model import Ridge

from sklearn.model\_selection import RandomizedSearchCV

from sklearn.metrics import mean\_squared\_error

from scipy.stats import loguniform

print('Shape of final train data:', X\_train.shape, y\_train.shape)

print('Shape of final test data:', X\_test.shape, id\_test.shape)

Standardisation des données

scaler\_final = StandardScaler()

X\_train\_std = scaler\_final.fit\_transform(X\_train)

Ajustement des Hyperparametres

random\_grid = {

'alpha': loguniform(1e-5, 1e4)

}

model = Ridge()

random\_search = RandomizedSearchCV(estimator=model,

param\_distributions=random\_grid, n\_iter=100,

scoring='neg\_root\_mean\_squared\_error',

cv=5, verbose=False, return\_train\_score=True,

random\_state=42, n\_jobs=-1)

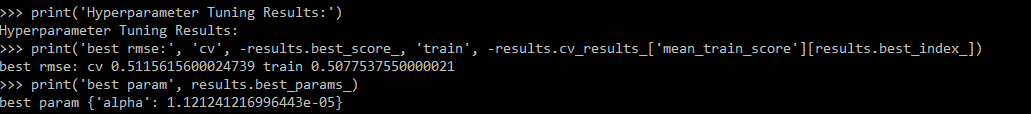
results = random\_search.fit(X\_train\_std, y\_train)

print('-'\*50)

print('Hyperparameter Tuning Results:')

print('best rmse:', 'cv', -results.best\_score\_, 'train', -results.cv\_results\_['mean\_train\_score'][results.best\_index\_])

print('best param', results.best\_params\_)



Training Finale

best\_params = results.best\_params\_

ridge\_model = Ridge(\*\*best\_params)

ridge\_model.fit(X\_train\_std, y\_train)

Testing Finale

X\_test\_std = scaler\_final.transform(X\_test)

id\_test\_pred = ridge\_model.predict(X\_test\_std)

rmse = mean\_squared\_error(id\_test, id\_test\_pred, squared=False)

print('-'\*50)

print('Final performance on test set:')

print('Final rmse on test data is:', rmse)

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Importer le stemmer de NLTK

from nltk.stem import PorterStemmer

Fonction pour mettre le texte dans la forme racine

def set\_root\_form(text):

stemmer = PorterStemmer()

words = text.split()

root\_form\_words = [stemmer.stem(word) for word in words]

return ' '.join(root\_form\_words)

def contains\_query(string, query):

return query in string

Trouver les produits les plus pertinents en fonction de la requête utilisateur

def find\_most\_relevant\_products(query, model, scaler, dt\_train):

preprocessed\_query = preprocessing\_search(query)

preprocessed\_query = standardize\_units(preprocessed\_query)

preprocessed\_query = set\_root\_form(preprocessed\_query)

contains\_query\_feature = int(contains\_query(dt\_train['search\_term'][0], preprocessed\_query))

query\_features = pd.DataFrame({

'len\_of\_querry': [len(preprocessed\_query.split())],

'shared\_words\_whole\_st\_pt': [set\_shared\_words\_whole((preprocessed\_query, dt\_train['product\_title'][0]))],

'shared\_words\_whole\_st\_pdat': [set\_shared\_words\_whole((preprocessed\_query, dt\_train['search\_term'][0]))],

'shared\_words\_part\_st\_pt': [set\_shared\_words\_part((preprocessed\_query, dt\_train['product\_title'][0]))],

'similarity': [Levenshtein.ratio(preprocessed\_query, dt\_train['product\_title'][0])],

'contains\_query': [contains\_query\_feature]

})

query\_features\_std = scaler.transform(query\_features.values)

relevance\_prediction = model.predict(query\_features\_std)[0]

results = dt\_train[['product\_title', 'search\_term', 'relevance']].copy()

results['relevance\_prediction'] = relevance\_prediction

results['contains\_query'] = (results['product\_title'].str.contains(preprocessed\_query, case=False) |

results['search\_term'].str.contains(preprocessed\_query, case=False))

results = results[results['contains\_query']]

results = results.sort\_values(by=['relevance\_prediction'], ascending=[False])

return results[['product\_title', 'search\_term', 'relevance\_prediction']]

Exemple d'utilisation de la fonction pour trouver les produits les plus pertinents pour une requête utilisateur

user\_query = "air conditioner"

relevant\_products = find\_most\_relevant\_products(user\_query, ridge\_model, scaler\_final, dt\_train)

show\_data\_info(relevant\_products)

print("Most relevant products for the query:", user\_query)

print(relevant\_products)

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Exemple d'utilisation de la fonction pour trouver les produits les plus pertinents pour une requête utilisateur

user\_query = "pan"

relevant\_products = find\_most\_relevant\_products(user\_query, ridge\_model, scaler\_final, dt\_train)

show\_data\_info(relevant\_products)

print("Most relevant products for the query:", user\_query)

print(relevant\_products)

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Exportation du fichier csv

pd.DataFrame({"id": id\_test, "relevance": y\_pred}).to\_csv(r'f:\submission\_HK.csv', index=False)

submission\_db = pd.read\_csv('f:/submission\_HK.csv', encoding='latin-1')

print(submission\_db)

print(submission\_db.head())

print(submission\_db.describe())

Une image contenant texte, capture d’écran

Description générée automatiquement

Durée

La durée d'exécution variait selon les différentes recherches.

L'objectif principal était de modéliser la recherche de produits, en se concentrant principalement sur la création de fonctionnalités utiles sur lesquelles appliquer les algorithmes ML.

Conclusion

J'ai exploré une variété de techniques du domaine de la recherche d'informations pour extraire des caractéristiques des données fournies. Les techniques allaient des simples opérateurs d'ensembles à l'indexation sémantique latente.

References

https://www.kaggle.com/c/home-depot-product-search-relevance

https://en.wikipedia.org/wiki/Information\_retrieval

Boolean Model in Infomation Retrieval for Search Engines

Fuzzy information retrieval model revisited by SławomirZadrożny and KatarzynaNowacka

Integrating the Probabilistic Model BM25/BM25F into Lucene

https://en.wikipedia.org/wiki/Learning\_to\_rank by Joaquín Pérez-Iglesias, José R. Pérez-Agüera, et al.

Indri: A language-model-based search engine for complex queries by Trevor Strohman, Donald Metzler, Howard Turtle, and W. Bruce Croft.

Using Linear Algebra for Information Retrieval by Michael W. Berry, et al.

Query Expansion Using Word Embeddings by Saar Kuzi, et al.

Feature Selection and Model Comparison on Microsoft Learning-to-Rank Data Sets by Sen Lei, et al.

The Vector Space Model in Information Retrieval: Term Weighting Problem

https://opensourceconnections.com/blog/2016/03/29/semantic-search-with-latent-semantic-analysis/

http://www.ccs.neu.edu/home/jaa/CSG339.06F/Lectures/vector.pdf

Video Lecture on Language Models for IR by Victor Lavrenko

http://datasciencebar.github.io/blog/home-depot-produce-relevance-review/

https://medium.com/kaggle-blog/home-depot-product-search-relevance-winners-interview-2nd-place-thomas-sean-qingchen-nima-68068f9f9ffd

https://arxiv.org/pdf/1803.05127.pdf

http://billy-inn.github.io/papers/cmput690.pdf

https://arxiv.org/pdf/2001.04980.pdf

<https://www.appliedaicourse.com>