Prévisions de vente de produits dans les relais gérés par Argedis
Challenge Mines Télécom

Rapport de projet, par Antonin VILLEMIN,

Elève à l'IMT Atlantique Bretagne Pays de la Loire, campus de Brest

SOMMAIRE

I. Introduction

Le but de ce challenge est d'essayer de prédire les ventes de différents articles dans les magasins d'autoroutes du groupe Total. En effet, pour mieux gérer les stocks, il peut être intéressant de savoir le nombre de ventes prévues pour la semaine qui arrive.

II. Les données

Pour pouvoir effectuer ces prédictions de quantités vendues, Total nous a mis à disposition une base de données contenant les quantités vendues de 204 articles différents, répartis sur deux magasins. Ces données sont disponibles du 16/04/2017 au 30/04/2017 pour la phase d'entrainement. Il faudra finalement être capable de prédire le plus précisément possible les quantités vendues du 01/05/2017 au 15/05/2017. Entrons maintenant dans le détail des données disponibles.

La base de données contient les informations liées à un article, à un jour donné et à un magasin donné. Ainsi, à chaque estimation, on va prédire la quantité vendue d'un article par jour et par lieu. Pour se faire, la base de données met à notre disposition 4 types de données :

- Le jour et le lieu de vente
- La description de l'article (son nom et à quelles catégories il appartient)
- La météo à Paris et à Rouen
- La circulation d'après Bison Futé dans les différentes zones de France

On peut réfléchir en quoi ces données sont corrélées avec les quantités vendues pour mieux comprendre notre problème. Pour ce qui est du jour et du lieu, il semble correct de penser que les ventes au mois de juillet peuvent être différentes du mois de décembre. De même, il est possible qu'un des deux magasins soit plus fréquenté que l'autre, par rapport à son accessibilité par exemple. Pour ce qui est de l'article, on peut facilement dire que l'on ne vend pas un sandwich comme on vend un chargeur de téléphone. Il est donc bien nécessaire de différencier chaque article. La météo peut également être corrélée avec nos ventes à venir. On peut par exemple penser que s'il fait très froid, les ventes de chocolats chauds ou de cafés seront importants. Et finalement, pour ce qui est de la circulation, il semble évident que plus il y a de monde sur les routes et plus il y aura de personnes dans les magasins d'autoroutes. Toutes ces données semblent donc légitimes à la résolution de notre problème.

III. Pré choix du modèle mathématique

Maintenant que nous avons compris l'utilité des différentes variables, il va falloir rendre celles-ci compréhensibles par notre programme de machine learning. Avant de se lancer, il faut savoir quel modèle nous avons l'intention d'utiliser, pour ainsi traiter les données en conséquences. En effet, certains modèles ne nécessitent pas de prétraitement des variables catégoriques par exemple. Pour répondre au problème posé, qui est un problème de régression, j'ai fait le choix d'utiliser un algorithme de forêt aléatoire, basé sur une multitude d'arbres décisionnels. Pour ce type de modèle, il va être nécessaire d'encoder les variables catégoriques mais il sera inutile d'appliquer une mise à l'échelle des données. Nous pourrons aussi appliquer un autre algorithme, XGBoost, qui nécessite le même prétraitement.

IV. Importation et prétraitement des données

J'ai choisi de traiter ce problème avec le langage de programmation Python et de ses librairies, pandas pour importer et manipuler les données et scikit-learn pour créer le modèle de prédiction. Après avoir importé correctement les données, il est nécessaire de regarder s'il n'y a pas de données manquantes dans notre dataset. Dans notre cas, il n'y en a pas. On peut donc passer directement au prétraitement.

Entrée [2]:
```
import pandas as pd
dataset = pd.read_csv('train.csv',';')
X_eval = pd.read_csv('test.csv',';')
```

Entrée [3]:
```
X_eval.head()
```

Out[3]:

id	implant	date	article_nom	id_categorie_6	id_categorie_5	id_categorie_4	cat6_nom	cat5_nom	cat4_nom	t_9h_rouen	n_9h_rouen	rr3_9h_roue
'325d4544	NF059473	2017-04-16	Cookie Cara Noix Pecan 70G Michel Augustin	1001672	1001664	1001639	Patiss PréEmballé	Patisserie	Boulan Vienn Patiss	283,85	100,0	0
73c9fb5d5	NF059473	2017-04-16	Cookie Choc Blc 70G Michel Augustin	1001672	1001664	1001639	Patiss PréEmballé	Patisserie	Boulan Vienn Patiss	283,85	100,0	0
l4d247208	NF059473	2017-04-16	Cookie Choc Nois 70G Michel Augustin	1001672	1001664	1001639	Patiss PréEmballé	Patisserie	Boulan Vienn Patiss	283,85	100,0	0
68f5fe7f6c	NF059473	2017-04-16	Cookie Choco 70G Michel Augustin	1001672	1001664	1001639	Patiss PréEmballé	Patisserie	Boulan Vienn Patiss	283,85	100,0	0
:71ad7069	NF059473	2017-04-16	Croissant 60G Cuit/Place Neuhauser	1001674	1001665	1001639	Vienn cuit/place	Viennoiserie	Boulan Vienn Patiss	283,85	100,0	0

Les variables catégoriques sont celles qui prennent des valeurs discrètes. Dans notre cas il va falloir encoder le nom de l'article, le magasin dans lequel est cet article ainsi que les 3 sous-catégories définissants notre article. Pour ce qui est des prévisions de bison futé, j'ai choisi de les prendre en tant que variables continues, telles qu'elles sont présentes à l'origine. J'ai également essayé des les prendre comme variables discrètes mais cela n'améliorait pas mes résultats.

Comme nous l'avons vu précédemment, il faut encoder ces variables. Pour se faire, il faut créer une nouvelle variable par catégorie. On va mettre 1 dans cette colonne si l'article appartient à cette catégorie, 0 sinon. Voici un exemple et la partie de mon code qui encode l'identifiant du magasin :

Entrée [7]:
```
pd.get_dummies(X.implant,prefix='implant').head(3)
```

Out[7]:

	implant_NF059473	implant_NF078544
0	1	0
1	1	0
2	1	0

Entrée [8]:
```
X = pd.concat([X,pd.get_dummies(X.implant,prefix='implant')],axis = 1)
X = X.drop(columns='implant')
```

Le pré traitement des variables catégoriques étant terminé, il reste à parler des variables continues. Dans notre fichier .csv, les virgules dans les nombres sont représentées avec une virgule, or pour que Python puisse comprendre ces nombres, il va falloir transformer les virgules en point, comme le fait le code suivant.

Entrée [26]:
```
for i in range(5,X.shape[1]):
    X.iloc[:,i] = X.iloc[:,i].apply(lambda x:float(x.replace(',','.')) if isinstance(x, str) else x)
```

Pour finir, nous allons travailler sur la date. J'ai décidé d'extraire deux informations de la date : le jour de la semaine (0=lundi, 1=mardi…) ainsi que le jour de l'année (de 1 à 365). Le jour de la semaine sera pris comme une variable discrète, on va donc créer 7 nouvelles colonnes. Le jour de l'année sera encodé comme une variable cyclique, c'est-à-dire que nous allons ajouter le cosinus et le sinus du jour de l'année.

Encode day of the week and day of year

Entrée [18]:
```
dayofweek = []
dayofyear = []
for date in X.date:
    day = datetime.strptime(date, "%Y-%m-%d").weekday()
    dayyear = datetime.strptime(date, "%Y-%m-%d").timetuple().tm_yday
    dayofweek.append(day)
    dayofyear.append(dayyear)
encoded_dayofweek = np.zeros((X.shape[0],7))
for i in range(X.shape[0]):
    encoded_dayofweek[i,dayofweek[i]] = 1

cos_dayofyear = np.cos(2*np.pi*(np.array(dayofyear)-1)/365,dtype = float)
sin_dayofyear = np.sin(2*np.pi*(np.array(dayofyear)-1)/365,dtype = float)
```

J'ai également fait le choix de modifier notre variable dépendante, les quantités vendues connues, afin d'obtenir une distribution centrée réduite, pour chaque article. J'ai donc calculé la moyenne et la variance de chaque quantité vendue, afin de réaliser des prédictions sur une même échelle de quantité. Puis, pour une quantité donné, j'ai soustrait cette valeur par la moyenne et j'ai divisé par la variance.

Entrée [26]:
```
std = {}
for i in group:
    std[i] = np.std(group[i])
```

Entrée [27]:
```
for i in range(len(y)):
    article = meta.article_nom[i]
    y[i] = (float(y[i])-means[article])/std[article]
```

Le prétraitement étant terminé, on peut maintenant créer notre modèle.

V. Construction du modèle

Comme vu précédemment, j'ai choisi d'utiliser une forêt aléatoire pour résoudre notre problème. Toutefois, il reste encore à choisir les paramètres de ce modèle. Pour cela j'ai choisi d'appliquer une méthode de Grid Search, couplé à de la Cross Validation, afin d'obtenir les meilleurs paramètres. La première méthode va créer plusieurs modèles avec des paramètres différents, la deuxième va permettre d'évaluer chacun de ces modèles de manière robuste afin d'extraire celui avec les meilleurs résultats. Voici la partie de code qui réalise cette tâche :

Entrée [38]:
```
from sklearn.model_selection import GridSearchCV

param_grid = {
            "n_estimators"      : [10,20,30],
            "max_features"      : ["auto", "sqrt", "log2"],
            "min_samples_split" : [2,4,8],
            "bootstrap": [True, False],
            }

grid_search = GridSearchCV(estimator = regressor,
                           param_grid = param_grid,
                           scoring = 'neg_mean_squared_error',
                           cv = 5,
                           n_jobs = 4,
                           verbose = 4)
grid_search = grid_search.fit(X, y)
```

Voici les 5 meilleurs choix de paramètres avec leur score associé :

```
('[CV]  bootstrap=False, max_depth=50, max_features=sqrt, min_samples_leaf=
2, min_samples_split=10, n_estimators=30', -0.5738941)
('[CV]  bootstrap=True, max_depth=90, max_features=auto, min_samples_leaf=4
, min_samples_split=10, n_estimators=30', -0.57312772)
('[CV]  bootstrap=False, max_depth=90, max_features=sqrt, min_samples_leaf=
2, min_samples_split=10, n_estimators=30', -0.5720478)
('[CV]  bootstrap=True, max_depth=50, max_features=sqrt, min_samples_leaf=1
, min_samples_split=10, n_estimators=30', -0.57139966)
('[CV]  bootstrap=True, max_depth=None, max_features=sqrt, min_samples_leaf
=1, min_samples_split=10, n_estimators=30', -0.5663198400000001)
```

On peut maintenant construire et entrainer notre modèle final :

Entrée [41]:
```
# Fitting Random Forest Regression to the dataset
from sklearn.ensemble import RandomForestRegressor
regressor = RandomForestRegressor(bootstrap=True, max_depth=None, max_features='sqrt',
                                  min_samples_leaf=1, min_samples_split=10, n_estimators=30,verbose = 2,n_jobs = 5)
```

Entrée [42]:
```
regressor.fit(X, y)
```

VI. Prédiction des quantités vendues

Entrée [22]:
```
y_pred = regressor.predict(X_eval)
```
```
[Parallel(n_jobs=5)]: Done  30 out of  30 | elapsed:    0.0s finished
```

Il faut ensuite décoder notre prédiction afin d'obtenir les quantités réelles :

Entrée [31]:
```
for i in range(len(y_pred)):
    article = meta_eval.article_nom[i]
    y_pred[i] = y_pred[i]*std[article]+means[article]
```

J'ai remarqué que les valeurs prises par nos quantités vendues dans notre base de données d'entrainement sont discrètes, comme le montre l'image suivante qui compte chaque valeur prise pour un article donné :

```
{'Cookie Cara Noix Pecan 70G Michel Augustin': {'0.56': 79,
  '1.11': 19,
  '0.19': 106,
  '0.37': 108,
  '0.0': 551,
  '0.74': 50,
  '1.3': 11,
  '0.93': 32,
  '1.48': 7,
  '1.67': 5,
  '1.85': 1,
  '2.04': 1,
  '2.6': 1,
  '2.41': 1},
```

J'ai donc choisi de transformer chaque valeur prédite en sa valeur discrète la plus proche.

From continuous to discrete

Entrée [32]:
```
for i in range(len(y_pred)):
    dictio = prices_dist[meta_eval.article_nom[i]]
    mini = 100
    min_dist = 1000
    for j in dictio:
        if abs(y_pred[i]-float(j))<min_dist:
            min_dist = abs(y_pred[i]-float(j))
            mini = j
    y_pred[i] = float(mini)
```

Entrée [33]:
```
y_pred
```

Out[33]:
```
array([0.93, 0.56, 0.74, ..., 0.46, 0.  , 0.  ])
```

On peut maintenant écrire notre fichier .csv final, sans oublier de retransformer les points en virgules pour les nombres flottants.

Write the final csv file

Entrée [34]:
```
y_pred_final = []
for i in range(len(y_pred)):
    y_pred_final.append(','.join(str(y_pred[i]).split('.')))
```

Entrée [35]:
```
y_final = pd.concat([meta_eval.id,pd.DataFrame(y_pred_final,columns=['quantite_vendue'])],axis=1)
```

Entrée [36]:
```
y_final.head()
```

Out[36]:

	id	quantite_vendue
0	d6b7325d4544	0,93
1	7e973c9fb5d5	0,56
2	98134d247208	0,74
3	0d68f5fe7f6c	1,85
4	f2d271ad7069	1,3

Entrée [37]:
```
y_final.to_csv('pred.csv',';',index=False)
```

Le fichier .csv a finalement été créé et contient toutes les prédictions demandées.

VII. Autres méthodes testées

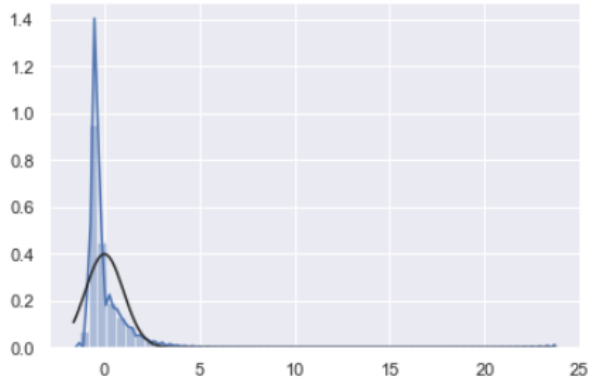
J'ai essayé d'appliquer deux autres modèles à notre problème : XGBoost et CatBoost, tous deux utilisant un méthode de gradient boosting sur des arbres décisionnels. Malheureusement, cela n'a pas amélioré mes résultats.

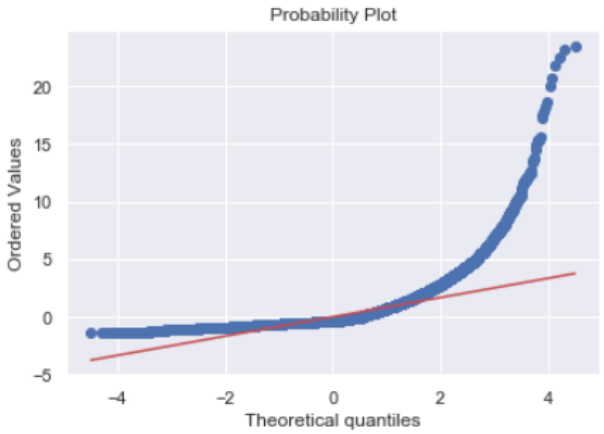
De plus, j'ai également regardé si les quantités vendues définissaient des gaussiennes :

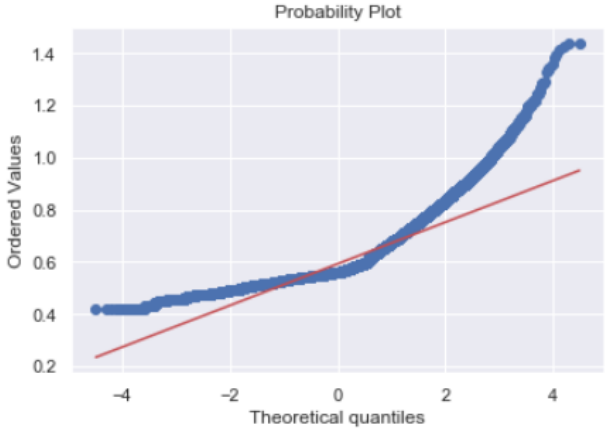
Entrée [30]:

```
from scipy.stats import norm
from scipy import stats
sns.distplot(list(y), fit=norm);

fig = plt.figure()
res = stats.probplot(list(y), plot=plt)
fig = plt.figure()
res = stats.probplot(list(np.log10(np.array(list(y))+4)), plot=plt)
```







Dans ce genre de problème, il peut arriver que les données définissent une gaussienne mais que celle-ci soit asymétrique. On peut notamment appliquer une fonction logarithme pour résoudre ce problème et ainsi améliorer nos prédictions. Toutefois, dans notre cas, beaucoup d'articles étaient très loin de représenter une gaussienne et cela n'améliorait donc pas ces résultats.

VIII. Possibilité d'amélioration

Une piste d'amélioration peut être de combiner linéairement plusieurs modèles, car il arrive que plusieurs mauvais modèles soient meilleurs qu'un seul bon modèle.

IX. Conclusion

Finalement j'ai fait le choix de répondre à ce problème à l'aide d'un modèle utilisant une forêt aléatoire, donnant les meilleurs résultats, pour à la fin prédire des quantités vendues discrètes.