Rapport

Projet Machine Learning : APST2

Challenge : Prediction of transaction claims status

by Rakuten PriceMinister

Mouktar ABDILLAHI - Pierre AUBRET

Supervisé par
M. Bertrand Michel

9 mars 2019

Table des matières

1 Introduction

PriceMinister-Rakuten, en tant qu'une des plus grandes plateformes de commerce électronique en France, proposant des produits neufs et d'occasions, participant à de nombreuses transactions par jour. En agissant en tant que tiers de confiance entre l'acheteur et le vendeur, Priceminister-Rakuten doit traiter les réclamations qui pourraient provenir d' articles cassés, de produits non reçus, etc. Ces réclamations ont un impact énorme sur l'expérience utilisateur de ces acheteurs et l'équipe Relations Clientèle de PriceMinister-Rakuten travaille chaque jour pour améliorer la qualité du service fourni.

Être capable de prédire si une transaction risque de conduire à une réclamation pourrait être une avancée énorme pour PriceMinister-Rakuten qui pourrait ainsi se concentrer sur ces transactions risquées. Cette prédiction est difficile car elle pourrait être basée sur un mélange de multiples caractéristiques hétérogènes.

Ainsi, dans le cadre de ce challenge, dont l'objectif est de prédire si une transaction réalisée sur le site de PriceMinister sera sujette à une réclamation de la part de l'acheteur. Nous avons donc testé de nombreux algorithmes de Machine Learning mais aussi de Deep Learning pour essayer d'obtenir la meilleure note possible en terme de ROC AUC Score et nous vous présenterons nos résultats dans la suite du rapport.

2 Présentation des données

2.1 Data description

Le but est d'évaluer si une transaction va entraîner une réclamation et si oui, laquelle parmi 7 types de réclamation possibles, que voici : "WITHDRAWAL", "SEL-

LER_CANCEL_POSTERIORI, "NOT_RECEIVED", "DIFFERENT", "UNDEFINED", "DAMAGED", "FAKE"

Les données sont dans 3 fichiers différents.

Tout d'abord le fichier train_X.csv : les données de training avec 22 colonnes et 100000 transactions spécifiées ici :

- ID : identifiant de la transaction
- SHIPPING_MODE : mode d'envoi
- SHIPPING_PRICE : coût de l'envoi
- WARRANTIES_FLG : Vrai si l'utilisateur a pris une garantie
- WARRANTIES_PRICE : Prix de la garantie, si elle existe
- CARD_PAYEMENT : Transaction payée par CB
- COUPON_PAYEMENT : Transaction payée avec un coupon de réduction
- RSP_PAYEMENT : Transaction payée avec les Rakuten Super Points
- WALLET_PAYMENT : Transaction payée avec le porte-monnaie PriceMinister-Rakuten
- PRICECLUB_STATUS : Statut de l'acheteur
- REGISTRATION_DATE : Année d'inscription de l'acheteur
- PURCHASE_COUNT : Montant des anciens achats de l'acheteur
- BUYER_BIRTHDAY_DATE : année de naissance de l'acheteur
- BUYER_DEPARTMENT : Son département
- BUYING_DATE : Date(mois/année) de l'achat
- SELLER_SCORE_COUNT : Montant des anciennes ventes du vendeur
- SELLER_SCORE_AVERAGE : Son score moyen sur PriceMinister-Rakuten
- SELLER_COUNTRY : Pays de provenance du vendeur
- SELLER_DEPARTMENT : Département du vendeur (-1 si hors de france)
- PRODUCT_TYPE : type du produit acheté
- PRODUCT_FAMILY : famille du produit acheté
- ITEM_PRICE : prix du produit acheté

Ensuite, nous avons test_X.csv qui contient les données de prédiction (95000 transactions) et qui est agencé exactement de la même façon que le précédent fichier.

Enfin, train_Y.csv qui contient la target data, donc le statut de chaque transaction en ce qui concerne les réclamations avec 2 colonnes :

- ID : identifiant de la transaction
- CLAIM_TYPE : type de réclamations

2.2 Problématiques soulevées par le jeu de données

Dans ce jeu de données de 22 variables, on retrouve de nombreuses variables catégorielles et comme de nombreux algorithmes de Machine Learning n'arrivent pas à traiter ce genre d'informations, il va falloir les numériser et les différentes méthodes que l'on peut employer pour réaliser cela peuvent mener à des scores complètement différents pour un même algorithme finalement. Certaines des variables catégorielles prennent aussi plus d'une centaine de catégories ce qui paraît être aussi un problème par la suite.

Ensuite, nous avons aussi des données manquantes en quantité plus ou moins importante selon la variable considérée. Il faut donc aussi régler ce problème pour les mêmes raisons que précédemment.

3 Transformations des données

3.1 Traitement des données manquantes

Nous avons identifié 2 types de variables avec des données manquantes :

Variables	Nombre de données	Type des données
SHIPPING_MODE	99685	object
SHIPPING_PRICE	32390	object
WARRANTIES_FLG	100000	bool
WARRANTIES_PRICE	3397	object
CARD_PAYEMENT	100000	int64
COUPON_PAYEMENT	100000	int64
RSP_PAYEMENT	100000	int64
WALLET_PAYMENT	100000	int64
PRICECLUB_STATUS	99943	object
REGISTRATION_DATE	100000	int64
PURCHASE_COUNT	100000	object
BUYER_BIRTHDAY_DATE	94164	float64
BUYER_DEPARTMENT	100000	int64
BUYING_DATE	100000	object
SELLER_SCORE_COUNT	99994	object
SELLER_SCORE_AVERAGE	99994	float64
SELLER_COUNTRY	100000	object
SELLER_DEPARTMENT	100000	int64
PRODUCT_TYPE	100000	object
PRODUCT_FAMILY	100000	object
ITEM_PRICE	100000	object

TABLE 1 – Etat de X_train initialement.

— ceux à qui ils manquent quelques centaines de données sur 100000 - ex : SHIPPING_MODE
— ceux à qui ils manquent plus de la moitié des données - ex : WARRANTIES_PRICE

Nous ne pouvons pas traiter ces deux groupes de la même manière et c'est ce que nous avons fait.

1. Groupe 1 : on a remplacé les données manquantes par l'attribut le plus fréquent pour les variables catégorielles et par la valeur la plus fréquente ou la moyenne des valeurs pour les variables numériques
2. Groupe 2 : Nous avons supprimé WARRANTIES_PRICE car elle avait 97% de valeurs manquantes et pour SHIPPING_PRICE, nous avons ajouté un nouvel attribut "missing" pour toutes les données manquantes.

3.2 Rendre les données numériques

3.2.1 Label Encoding

Une première approche classique de traitement des variables catégorielles est de simplement convertir chaque valeur de notre colonne en un nombre. Ainsi, si elle contient n valeurs différentes, ces n valeurs seront remplacés par des nombres entre 0 et n-1.

3.2.2 OneHot Encoding

La première approche a pour avantage d'être intuitive mais elle a ses limites puisque l'algorithme peut mal interpréter ces valeurs qui ne sont que représentatives en réalité et qui n'ont donc pas leur vraie valeur.

Nous avons donc une seconde approche que l'on appelle le OneHot Encoding. La stratégie de base est de convertir chaque valeur catégorielle en une nouvelle colonne et

d'assigner un 1 ou 0 dans cette colonne pour chaque transaction selon que la transaction prenne cette valeur catégorielle ou non. Ceci a l'avantage de ne pas mettre du poids à certaines valeurs de façon involontaire mais à l'inconvénient d'ajouter un nombre important de colonnes à notre jeu de données.

Prenons l'exemple de SHIPPING_MODE :

SHIPPING_MODE	0	1	2	3	4	5	6	7	8	9	10
Normal	1	0	0	0	0	0	0	0	0	0	0
Recommande	0	1	0	0	0	0	0	0	0	0	0
Suivi	0	0	1	0	0	0	0	0	0	0	0
Mondial_Relay	0	0	0	1	0	0	0	0	0	0	0
So_Point_Relay	0	0	0	0	1	0	0	0	0	0	0
Mondial_Relay_Prepaye	0	0	0	0	0	1	0	0	0	0	0
So_Recommande	0	0	0	0	0	0	1	0	0	0	0
Chronopost	0	0	0	0	0	0	0	1	0	0	0
Express_Delivery	0	0	0	0	0	0	0	0	1	0	0
Pick-Up	0	0	0	0	0	0	0	0	0	1	0
Kiala	0	0	0	0	0	0	0	0	0	0	1

TABLE 2 – OneHot Encoding de SHIPPING_MODE.

3.2.3 Binary Encoding

La troisième approche que nous testerons est celle que l'on appelle la Binary Encoding. Dans cette approche, on numérote les valeurs catégorielles comme dans le Label Encoding puis l'on traduit chaque nombre en binaire avant de représenter chaque variable catégorielle par un nombre fini de 0 et de 1 (nombre qui est cependant moins important que les n colonnes nécéssaires dans le OneHot Encoding)

Nous allons voir un exemple de cet encodage avec SHIPPING_MODE :

SHIPPING_MODE	0	1	2	3
Normal	0	0	0	0
Recommande	0	0	0	1
Suivi	0	0	1	0
Mondial_Relay	0	0	1	1
So_Point_Relay	0	1	0	0
Mondial_Relay_Prepaye	0	1	0	1
So_Recommande	0	1	1	0
Chronopost	0	1	1	1
Express_Delivery	1	0	0	0
Pick-Up	1	0	0	1
Kiala	1	0	1	0

TABLE 3 – Binary Encoding de SHIPPING_MODE.

Nous avons aussi testé d'autres méthodes de traitement des variables catégorielles comme la Sum Encoding, la Helmert Encoding ou la Backward Difference Encoding. Ce genre de méthodes sont moins intuitives que les précédentes mais elles peuvent se révéler intéressantes pour certains jeux de données.

3.3 Obtenir de l'information

Après avoir transformé toutes les données avec les différentes méthodes nous avons remarqué que nous ne pouvions pas dépasser un certain score et ce malgré des classifieurs

pertinents et optimisés. C'est dans cette mesure que le mot feature engineering prend tout son sens. Dans un premier temps nous avons transformé les dates comme affiché ci-dessous étant donné que le mois et l'année peuvent avoir un impact sur la livraison des colis (parole d'ancien facteur).

BUYING_MONTH	BUYING_YEAR
3	2017
8	2017

Nous avons aussi voulu appliquer la même idée pour séparer d'autres variables catégorielles de la même façon pour en faire des variables continues d'une certaine façon. Ainsi nous avons appliquer cette séparation aux variables suivantes : SELLER_SCORE_COUNT, PURCHASE_COUNT et ITEM_PRICE. Nous avons ainsi obtenu ceci par exemple :

SELLER_SCORE_MIN	SELLER_SCORE_MAX
10000	100000
1000	10000
PURCHASE_COUNT_MIN	PURCHASE_COUNT_MAX
0	5
50	100
ITEM_PRICE_MIN	ITEM_PRICE_MAX
500	1000
0	10

Ensuite, voyant que nos résultats ne s'amélioraient pas de manière signicative, on a remarqué qu'un autre élément rentrait en ligne de compte : On peut transformer nos données en faisant apparaître des distances entre l'expéditeur et le receveur. En effet, intuitivement, plus un colis parcourt une longue distance et franchis des frontières et plus il est probable qu'il lui arrive quelques péripéties. Nous avons démarré par les envois venant de vendeur à l'étranger. Nous avons donc réalisé une première matrice recensant les distances entre les capitales des pays étrangers et Paris pour avoir une première estimation de la distance parcourue par ce genre de colis. Un exemple de la matrice obtenue est fourni en annexe.

Ensuite, nous avons traité le cas des vendeurs et acheteurs situés en France. Nous avons ainsi créer une première matrice recensant les distances qui séparent les préféctures des différents départements de France Métropolitaine. Nous avons pu réaliser ceci grâce à l'API de Google appelée DistanceMatrix qui est utilisable gratuitement et directement sur Python en chargeant une certaine librairie. La seule contrainte que nous imposait ce service est le nombre de calculs qu'il pouvait réaliser, nous avons donc du relancer ce programme à plusieurs reprise pour à la fin obtenir ce que nous voulions.

Le jeu de données était aussi problèmatique pour certaines transactions puisqu'on pouvait retrouver des départements (vendeurs ou acheteurs) qui prenaient la valeur de 0 ou 99 donc qui ne correspondent à aucun départemennt connu en France ou encore les valeurs 96 à 98 qui elles peuvent correspondre à différents départements d'Outre-Mer. Nous avons ainsi pris le parti de mettre une certaine valeur à ces exceptions qui sont en nombre minime étant donnée l'étendue de notre jeu de données. Vous retrouverez en annexe un aperçu de la matrice des distances que nous avons ainsi réalisée.

En fin de compte, une fois que les distances entre les acheteurs et les vendeurs furent calculés, nous avons pu nous séparer des colonnes SELLER_DEPARTMENT, SELLER_COUNTRY et BUYER_DEPARTMENT pour éviter la redondance des informations et se préserver d'un éventuel overfitting .

4 Analyses réalisées

4.1 EDA : Exploration des features

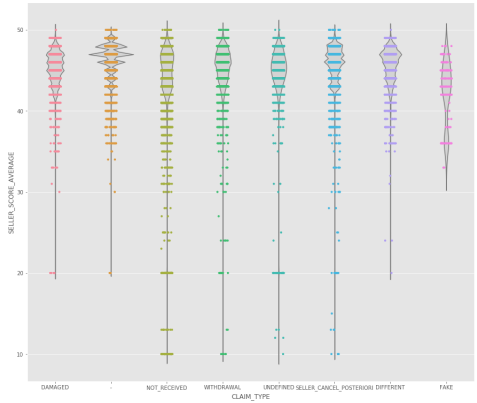

FIGURE 1 – Stripplot : SellerScoreAverage et Claimtype

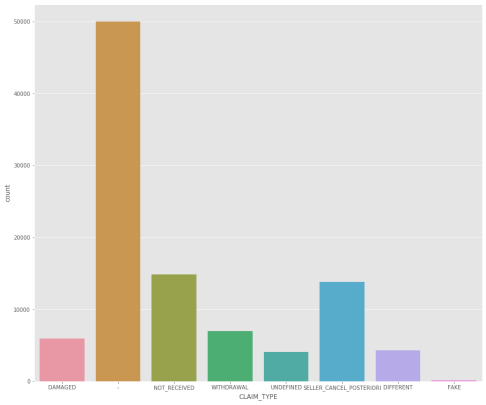

FIGURE 2 – Countplot : Claimtype

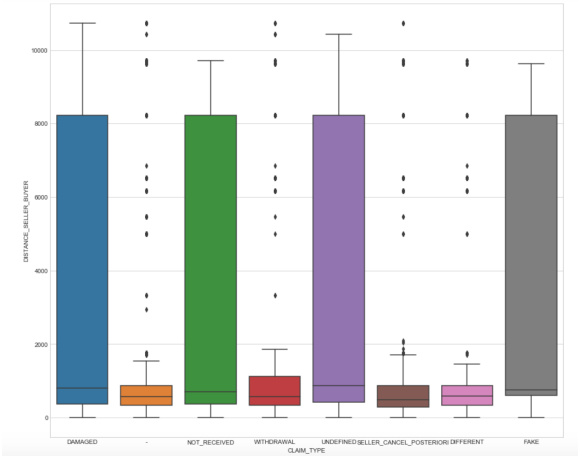

FIGURE 3 – Boxplot : DistanceSeller-Buyer et Claimtype

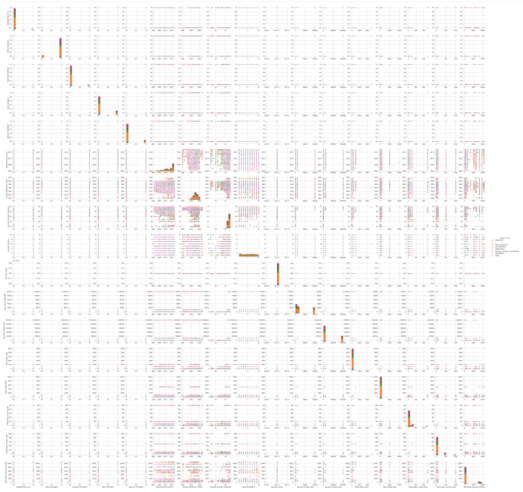

FIGURE 4 – PairsPlot

Vous retrouverez ces graphes dans le notebook pour que vous puissiez les voir en entier et zoomer.

4.2 1er test de plusieurs algorithmes de Machine Learning

Nous avons voulu réaliser un premier test de plusieurs algorithmes pour comparer leur efficacité. Nous nous sommes heurtés à un problème en ce qui concerne le scoring de ces algorithmes. en effet, la méthode préconisée par PriceMinister était l'AUC Score sauf que nous avons un problème de classification multi-classe et la façon de définir ce type de score est compliquée dans ce cas. Nous avons tout de même essayé de le faire, en définissant un ROC AUC SCORE pour chaque classe (parmi les 8 que contient CLAIM_TYPE) sous la forme d'un OneVsRest. C'est à dire, que nous avons fait comme si pour chaque classe nous avions une classification binaire entre la dite classe et toutes les autres classes. Vous avez dans la suite les résultats obtenus pour les différents algorithmes testés que ce soit l'accuracy ou les 8 Score AUC.

Model	Runtime Training	Score
GradientBoosting	135.873651	0.52925
XGBoost	76.305909	0.52530
Random Forest	1.438294	0.52165
Bagging Classifier	5.296585	0.51575
AdaBoost	8.171891	0.51130
ExtraTrees	1.987083	0.51115
LDA	0.416764	0.50590
KNN	3.766384	0.47980
Naive Bayes	0.185176	0.45070
QDA	0.306224	0.21390
Logistic Regression	37.836369	0.11870

FIGURE 5 – Accuracy score et timing pour différents modèles

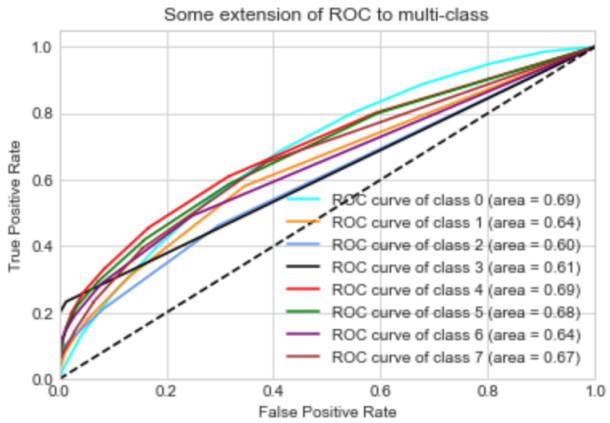

FIGURE 6 – RandomForest

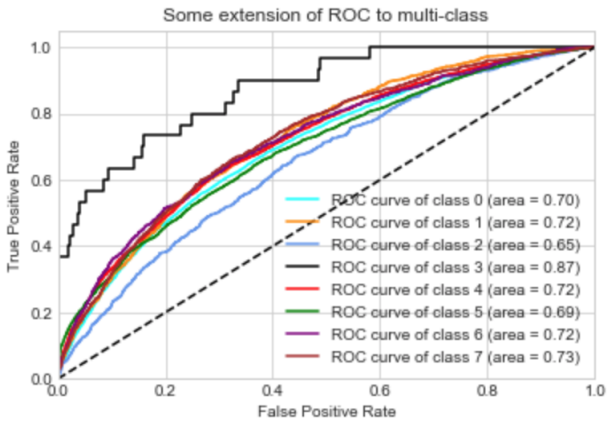

FIGURE 7 – XGBoost

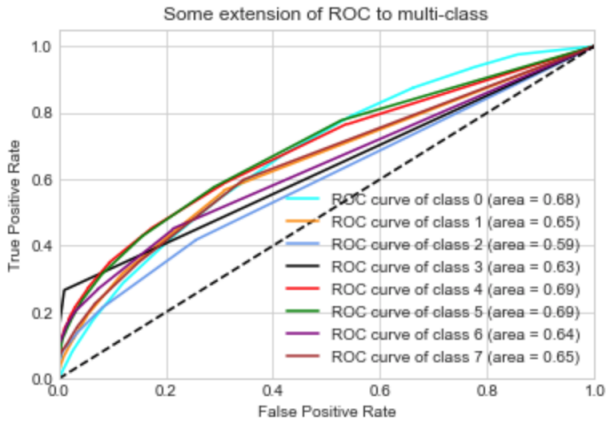

FIGURE 8 – ExtraTrees

4.3 Choix des paramètres pour améliorer notre score

Pour essayer d'améliorer nos résultats, nous avons voulu choisir les valeurs pour les paramètres de nos classifieurs les plus performants initialement. Comme notre jeu de données est assez conséquent et que le nombre de paramètres que nous devons étudier est important, nous avons choisi de partir tout d'abord sur un RandomizedSearchCV au lieu d'un GridSearchCV classique. Voici les paramètres obtenus par cette première approche :

	Paramètres	Score
RandomForest	max_depth = 200 min_samples_leaf = 1 max_features = auto min_samples_split = 5 n_estimators = 636	0.54885
ExtraTreesClassifier	max_depth = None min_samples_leaf = 2 max_features = auto min_samples_split = 5 n_estimators = 806	0.547675
XGBClassifier	max_depth = 10 learning_rate = 0.1 booster = gbtree	0.5404875
BaggingClassifier	n_estimators = 927 max_samples = 0.6 max_features = 0.6	
AdaBoostClassifier	n_estimators = 927 learning_rate = 0.3	0.5127

Cette première étude nous montre que les trois premiers classifieurs sont les plus intéressants. Nous allons tenter de les exploiter le plus possible pour en tirer le meilleur score. Pour cela nous avons pensé à un GridSearchCV sur n_estimators pour les deux premiers et sur le learning_rate pour le dernier.

Nous avons donc utiliser GridSearchCV pour encore mieux optimiser nos paramètres pour nos 3 algorithmes les plus performants. nous avons ainsi pu obtenir les résultats suivants :

	Paramètres	Score
RandomForest	max_depth = 200 min_samples_leaf = 1 max_features = auto min_samples_split = 5 n_estimators = 685	0.549225
ExtraTreesClassifier	max_depth = None min_samples_leaf = 2 max_features = auto min_samples_split = 5 n_estimators = 806	0.54867
XGBClassifier	max_depth = 10 learning_rate = 0.14 booster = gbtree	0.5432375

Finalement, le meilleur score obtenu est fait avec RandomForest avec un score final de 0.549225.

Commentaire : Lorsque nous avons posté la prédiction faite avec notre RandomForest optimisé pour notre jeu de données transformé, nous n'avons pas pu améliorer notre score puisque l'AUC Score selon la plateforme ChallengeData est de 0.60 pour ce cas-ci au lieu de 0.61 avec un XGBoost optimisé que nous avions réalisé précédemment avec un jeu de données pour lequel nous avions seulement labellisé nos variables catégorielles. Ce fut donc une déception que notre feature engineering n'améliore pas sensiblement nos résultats.

4.4 Deep Learning : Neural Networks

Nous avons aussi pensé à tester le Deep Learning pour notre problème de classification multi-classe. Pour cela, nous avons créé une fonction create_model qui prend pour argument l'optimiseur, le nombre de couches, le nombre de neurones par couche ainsi que la fonction d'activation que l'on souhaite utiliser. Cette fonction va nous permettre de créer de nombreux modèles différents assez rapidement pour pouvoir tester leur efficacité.

Nous avons donc voulu construire notre meilleur modèle en faisant une sorte de GridSearchCV manuel pour les différents paramètres d'un neural network. En effet, il est très difficile d'optimiser un Neural Network, nous avons du donc tatonner et y aller petit à petit pour améliorer au mieux notre score de départ qui était aux alentours de 0.51. Voici les étapes par lesquels nous sommes passés :

- Nous avons choisi le meilleur optimiseur parmi un éventail d'optimiseurs pour un jeu de paramètres donnés. Notre choix s'est porté sur **Adamax** car nous avons obtenu le meilleur validation score avec celui-ci.

- Nous avons ensuite choisi le nombre de couches intermédiaires à avoir dans notre réseau de neurones. Pour cela, nous avons cherché le meilleur validation score parmi plusieurs nombres de couches en mettant comme optimiseur **Adamax**. Nous avons obtenu la valeur de 2.

Nous avons donc fait pareil pour le nombre de neurones par couche et pour la fonction d'activation. Au bout du compte, nous avons donc pu améliorer notre score sensiblement puisque le validation score de 0.5304 a été atteint.

Toutefois, le score baisse un peu, une fois que nous évaluons notre modèle sur la partie Test de notre jeu de données puisque le validation score était lui calculé sur une part de la partie Train de nos données. On se rend donc compte que, malgré notre tentative d'optimisation de notre neural network, le score est faible par rapport à notre Random Forest. Nous ne retiendrons donc pas cette méthode au bout du compte. Cela était prévisible étant donné que le deep learning est réservé aux problèmes en plus grandes dimensions où des convolutions interviennent.

4.5 Autres méthodes

Comme nous l'avons dit dans les parties précédentes, nous avions aussi d'autres idées en ce qui concerne le traitement des variables catégorielles. Nous allons vous présenter dans la suite les difficultés rencontrées avec ces méthodes ainsi que les résultats obtenus pour certaines :

- **Binary Encoding** : Une fois que ce traitement est effectué, notre nombre de variables augmente et double presque. Par conséquent les calculs prennent plus de temps à s'exécuter ce qui est gênant. De plus, les résultats obtenus pour les différents classifieurs sont moins intéressants que pour le LabelEncoding. Comme nous n'avons pas d'amélioration au niveau du score en utilisant cette méthode, nous avons du la laisser tomber.

Final Model

```
In [74]: model = create_model('Adamax', 2, 300, 'relu')
         model.fit(X_Train_nor, y_Train1, validation_split = 0.2, epochs = 20, callbacks = [early_stopping_monitor])
```

```
Train on 64000 samples, validate on 16000 samples
Epoch 1/20
64000/64000 [==============================] - 22s 343us/step - loss: 1.4463 - acc: 0.5154 - val_loss: 1.4106 - val_
acc: 0.5237
Epoch 2/20
64000/64000 [==============================] - 21s 324us/step - loss: 1.4135 - acc: 0.5208 - val_loss: 1.3970 - val_
acc: 0.5250
Epoch 3/20
64000/64000 [==============================] - 20s 319us/step - loss: 1.3967 - acc: 0.5250 - val_loss: 1.3914 - val_
acc: 0.5229
Epoch 4/20
64000/64000 [==============================] - 21s 328us/step - loss: 1.3815 - acc: 0.5280 - val_loss: 1.3837 - val_
acc: 0.5275
Epoch 5/20
64000/64000 [==============================] - 21s 323us/step - loss: 1.3699 - acc: 0.5302 - val_loss: 1.3810 - val_
acc: 0.5271
Epoch 6/20
64000/64000 [==============================] - 21s 327us/step - loss: 1.3562 - acc: 0.5324 - val_loss: 1.3879 - val_
acc: 0.5282
Epoch 7/20
64000/64000 [==============================] - 21s 333us/step - loss: 1.3421 - acc: 0.5346 - val_loss: 1.3872 - val_
acc: 0.5268
Epoch 8/20
64000/64000 [==============================] - 21s 332us/step - loss: 1.3277 - acc: 0.5383 - val_loss: 1.3953 - val_
acc: 0.5242
```

```
Out[74]: <keras.callbacks.History at 0x287cf1f98>
```

```
In [76]: # evaluate the model
         score = model.evaluate(X_Test_nor, y_Test1, verbose=0)
         score
```

```
Out[76]: [1.4061680550575257, 0.52490000000000003]
```

- **OneHot Encoding** : Ce traitement a pour inconvénient d'ajouter un trop grand nombre de variables à notre jeu de données qui est déjà très chargé. Par conséquent, les temps d'exécution se rallongent et on perd ainsi tout intérêt pour ce genre de méthodes.

Voici les scores obtenus pour ces deux types d'encodage des variables catégorielles.

Model	Runtime Training	Score
GradientBoosting	626.573208	0.52950
XGBoost	522.500675	0.52795
Bagging Classifier	14.914231	0.52250
Random Forest	2.210990	0.52110
AdaBoost	18.438199	0.51325
ExtraTrees	3.484277	0.50220
LDA	3.104260	0.50040
KNN	4.347884	0.47905
Naive Bayes	0.482434	0.45835
Logistic Regression	142.055664	0.05655
QDA	2.573178	0.02585

FIGURE 9 – OneHot Encoding

Model	Runtime Training	Score
GradientBoosting	170.640833	0.52600
XGBoost	119.152187	0.52400
Random Forest	1.346469	0.51970
Bagging Classifier	6.057472	0.51740
AdaBoost	9.160534	0.51030
LDA	0.566003	0.50610
ExtraTrees	1.675228	0.50360
KNN	2.756633	0.47865
Naive Bayes	0.182805	0.45830
Logistic Regression	46.996095	0.08740
QDA	0.379882	0.02885

FIGURE 10 – Binary Encoding

- **Helmert, Sum and BackwardDifference Encodings :** Pour ces trois dernières méthodes, nous avons le même inconvénient que pour le OneHot Encoding à savoir le nombre trop important de nouvelles variables créées et ainsi le temps d'éxécution. Toutefois, nous avons aussi le fait que les manipulations faites sur nos variables initiales sont compliquées et qu'il n'est donc pas facile de remonter aisément à ces dernières, une fois le traitement effectué.

5 Scoring et Conclusion

Nous dressons un tableau récapitulatif qui va permettre de retracer tout ce qui a été essayé. Le score affiché est celui obtenu sur le site en utilisant donc toutes nos données train. Nous avons retracé uniquement les tests les plus pertinents et non redondants en privilégiant ceux qui nous ont permis d'obtenir les meilleurs scores. .

Fonction	Feature Engineering	Commentaire	Score
Random Forest	Label Encoding	50 estimateurs, valeurs manquantes mises à la moyenne	0.6015
Random Forest	Label Encoding	85 estimateurs, GridSearch,	0.6028
Random Forest	Label Encoding, Mise à niveau des dates, Recherche de distance (expéditeur/réceptionneur)	Paramètres tunés avec RandomizedSearchCV	0.6047
ExtraTrees	Label Encoding	85 estimateurs, GridSearch	0.6040
XGBoost	Label Encoding	Paramètres tunés avec RandomizedSearchCV	**0.6164**

Notre notebook contiendra l'ensemble de nos résultats et des méthodes que nous avons testé. Toutefois en ce qui concerne le traitement des données, nous avons choisi de présenter notre étape finale, c'est à dire l'étape pour laquelle notre feature Engineering est le plus perfectionné. Les étapes précédentes se retrouvent dans cette étape, c'est pourquoi nous l'avons choisie.

Difficulté la plus importante rencontrée : La taille de notre jeu de données a rendu certains classifieurs non pertinents puisqu'ils prenaient trop de temps pour s'éxécuter, nous avons pour cela l'exemple des méthodes à noyaux SVM ou encore du Gradient-Boosting.

Ce Data Challenge nous a permis d'appliquer les connaissances acquises pendant les TPs d'APST2 à un cas réel de bout en bout. Nous avons ainsi pu tester les classifieurs vus en cours mais aussi d'en découvrir de nouveaux. Il nous a aussi permis de comprendre que le Data Cleaning et le Feature Engineering étaient les étapes les plus importantes lors du traitement de ce genre de challenge. En effet, nous nous sommes très vite rendus compte de l'impact que pouvait avoir le traitement préalable des données sur les scores que nous faisions au bout de compte. La plateforme Kaggle ou les tutorials de DataCamps furent ainsi très intéressants pour nous permettre d'acquérir des compétences complémentaires à celles du cours et de perfectionner notre étude en nous inspirant de ce que les gagnants d'autres challenges (Kaggle) avaient fait pour arriver à de tels résultats.

Ainsi, ce challenge nous a permis de nous investir sur un projet de Data Science pendant un temps assez long pour pouvoir nous perfectionner et apprendre aussi par nous-mêmes. C'est une démarche qui nous a vraiment plu et nous espérons pouvoir en refaire prochainement.

6 Annexe

	0	1	2	3	4	5	6	7	8	9	...	86	87	88	89	90	91	92	93	94	95
0	0.0	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	9.0	...	86.0	87.0	88.0	89.0	90.0	91.0	92.0	93.0	94.0	95.0
1	1.0	0.0	437.0	165.0	307.0	241.0	421.0	195.0	446.0	548.0	...	413.0	356.0	266.0	237.0	235.0	375.0	415.0	406.0	393.0	433.0
2	2.0	437.0	0.0	373.0	746.0	671.0	860.0	620.0	92.5	853.0	...	450.0	495.0	289.0	221.0	380.0	155.0	143.0	126.0	136.0	139.0
3	3.0	165.0	373.0	0.0	437.0	388.0	551.0	272.0	404.0	507.0	...	249.0	200.0	333.0	152.0	332.0	262.0	302.0	294.0	280.0	321.0
4	4.0	307.0	746.0	437.0	0.0	132.0	115.0	193.0	752.0	448.0	...	647.0	542.0	566.0	544.0	508.0	682.0	721.0	712.0	699.0	740.0
5	5.0	241.0	671.0	388.0	132.0	0.0	230.0	239.0	668.0	547.0	...	623.0	538.0	451.0	479.0	386.0	617.0	656.0	648.0	634.0	675.0
6	6.0	421.0	860.0	551.0	115.0	230.0	0.0	304.0	866.0	511.0	...	753.0	641.0	681.0	658.0	623.0	796.0	836.0	827.0	813.0	854.0
7	7.0	195.0	620.0	272.0	193.0	239.0	304.0	0.0	631.0	375.0	...	459.0	355.0	461.0	409.0	429.0	529.0	569.0	561.0	547.0	588.0
8	8.0	446.0	92.5	404.0	752.0	668.0	866.0	631.0	0.0	902.0	...	525.0	561.0	253.0	253.0	355.0	237.0	233.0	215.0	223.0	230.0
9	9.0	548.0	853.0	507.0	448.0	547.0	511.0	375.0	902.0	0.0	...	469.0	366.0	809.0	649.0	783.0	714.0	747.0	748.0	734.0	764.0
10	10.0	273.0	164.0	223.0	579.0	512.0	694.0	457.0	181.0	721.0	...	372.0	383.0	199.0	72.1	268.0	138.0	165.0	150.0	143.0	181.0
11	11.0	488.0	830.0	464.0	368.0	467.0	431.0	295.0	868.0	79.6	...	468.0	361.0	754.0	615.0	718.0	691.0	728.0	725.0	710.0	745.0
12	12.0	339.0	671.0	299.0	365.0	442.0	440.0	212.0	703.0	209.0	...	349.0	230.0	600.0	451.0	574.0	547.0	584.0	581.0	566.0	601.0
13	13.0	372.0	797.0	464.0	133.0	256.0	159.0	197.0	808.0	376.0	...	637.0	520.0	636.0	592.0	601.0	722.0	762.0	755.0	741.0	781.0
14	14.0	592.0	322.0	440.0	878.0	833.0	992.0	712.0	412.0	790.0	...	322.0	426.0	569.0	360.0	639.0	238.0	207.0	226.0	229.0	198.0
15	15.0	306.0	602.0	232.0	412.0	437.0	495.0	229.0	636.0	275.0	...	279.0	162.0	543.0	384.0	533.0	470.0	507.0	504.0	489.0	524.0

FIGURE 11 – Aperçu Matrice Distance France

CHINA	GERMANY	HONG KONG	UNITED KINGDOM	ANDORRA	SWITZERLAND	SPAIN	UNITED STATES	ITALY	BELGIUM	...	SLOVENIA	SWEDEN	MALTA	VATICAN CITY STATE (HOLY SEE)	ESTONIA	C
8217	868	9628	341	709	488	1051	6162	1119	259	...	965	1545	1741	1103	1859	

FIGURE 12 – Aperçu Matrice Distance Etranger