UNIVERSITÉ DE NANTES

IAE NANTES
ÉCONOMIE & MANAGEMENT

Riiid ! Prédiction de l'exactitude de la réponse

M2 EKAP 2020-2021

BLAISE Maxime et HENNEBOIS Salomé

Sommaire

I. Introduction

« Sans éducation, l'enfant est orphelin », ce proverbe français est on ne peut plus vrai, l'accès à l'éducation est un droit que n'importe quel être humain devrait avoir, sa privation ne peut qu'empêcher le développement. Beaucoup trop d'enfants en sont encore privés et ne parviennent pas à s'instruire correctement. Aujourd'hui ce problème est encore plus d'actualité à l'heure de la Covid-19, nombreuses sont les écoles qui ferment et rare sont les solutions pour les écoliers. De même, trop d'élèves ont encore des difficultés avec l'enseignement de certaines matières et auraient besoin d'aide supplémentaires qu'ils n'arrivent pas à trouver. L'idée de la compétition Kaggle que nous avons sélectionnée est de permettre, grâce au Machine Learning et à de nombreux algorithmes, à tous ces enfants de pouvoir étudier avec comme seule nécessité une connexion internet. Pour cela les organisateurs de la compétition, Riiid, nous proposent une base avec plus de 100 000 000 d'observations correspondant à des interactions de millions d'élèves, le but étant de pouvoir prédire la qualité de leurs réponses.

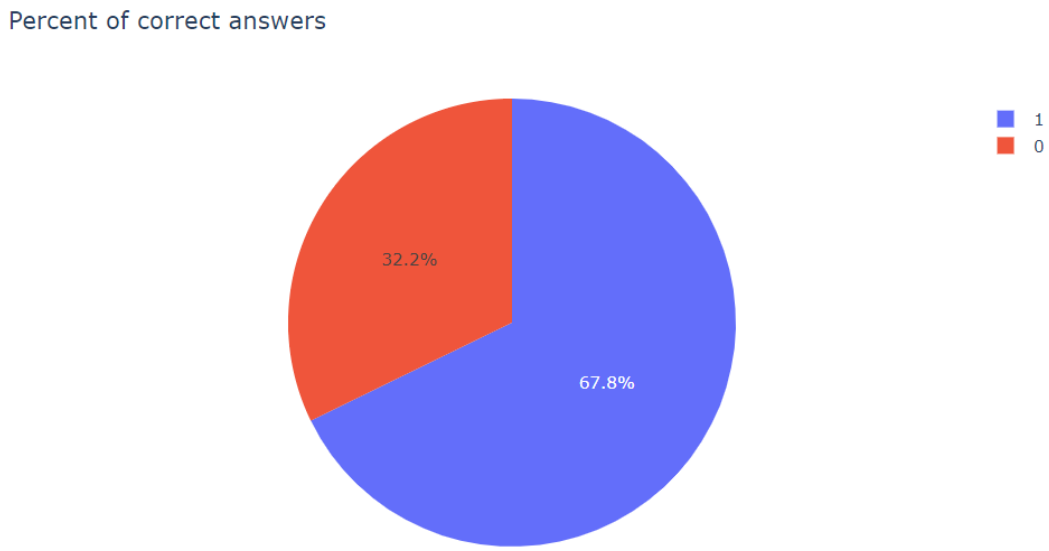
II. Présentation des variables

On retrouve une base de 101 millions d'observations décrites en 10 variables :

- La première « **row_id** » correspond tout simplement au numéro de la ligne
- La seconde « **timestamp** » nous indique le temps en millisecondes entre l'interaction de l'utilisateur et l'achèvement du premier événement de cet utilisateur (c'est-à-dire sa réponse à la question ou la fin de sa lecture)
- « **user_id** » correspond à l'identifiant de l'utilisateur, elle nous permet de différencier chaque élève.
- « **content_id** » Il s'agit de l'identification de l'interaction, elle nous permet de différencier chaque interaction
- « **content_type_id** » : 0 = question posée, 1 = lecture, cela nous indique si l'interaction était une question ou un moment de lecture
- « **task_container_id** » : code du lot de questions/conférences, les interactions sont regroupées en groupe, cet identifiant nous permet de reconnaitre chaque groupe
- « **user_answer** » : réponse à la question (-1 = nul, pour les moments de lectures)
- « **answered_correctly** » : 1 si l'utilisateur a répondu correctement, 0 s'il s'est trompé et -1 s'il s'agissait d'un moment de lecture.

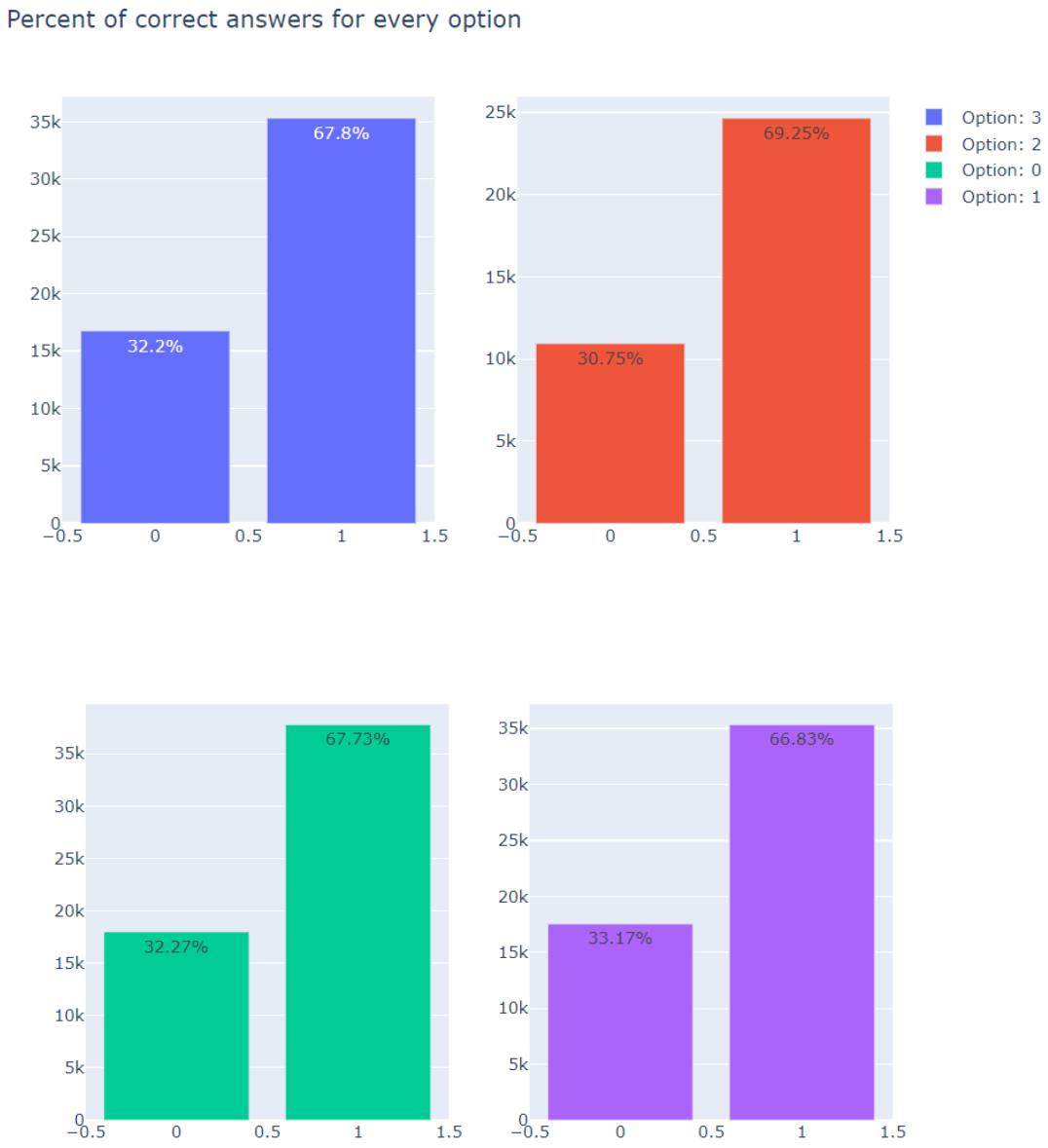
- « **prior_question_elapsed_time** » : temps moyen en millisecondes nécessaire à un utilisateur pour répondre à chaque question du groupe d'interactions précédentes, en ignorant tous les instants de lectures. Par défaut, on indique 0 pour la première série de questions ou la première conférence d'un utilisateur
- « **prior_question_had_explanation** » : Indique si l'utilisateur a vu ou non une explication et la ou les bonnes réponses après avoir répondu au groupe de questions précédent. La valeur est la même pour tous le groupe de questions (qu'on peut reconnaitre grâce à « task_container_id)». Par défaut, elle est égale à 0 pour le premier groupe de question. En règle générale, les premières questions qu'un utilisateur voit faisaient partie d'un test de diagnostic d'intégration.

<u>Représentation de la variable à expliquer answered_correctly :</u>



Source : Graphique obtenu grâce à la plateforme Jupyter Notebook

Représentation de « answered correctly » en fonction de la réponse attendue :



Source : Graphique obtenu grâce à la plateforme Jupyter Notebook

III. Présentation des modèles

Une observation de la base nous permet de constater que le taux de réponses correctes est d'environ 60% (sur notre échantillon étudié, nous obtenons 67.8% de bonnes réponses). Nous allons tenter divers modèles de Machine Learning afin de prédire au mieux l'exactitude des réponses données.

Le premier modèle tenté est un classifier SVM (Support Vector Machine). Le taux de prédictions correctes de ce modèle est de 66.85%.

Modèle SVM sur l'échantillon initial :

```
from sklearn import svm
from sklearn.model_selection import cross_val_score

svm_classifier = svm.SVC(kernel = 'rbf')
svm_fit = svm_classifier.fit(train, train.answered_correctly)
svm_pred = svm_classifier.predict(train)

cv_results = cross_val_score(svm_classifier, train, train.answered_correctly, cv=2)
mean_cv_results = cv_results.mean()
print("Taux de prédictions correctes : {}" .format(cv_results))
print("Taux moyen de prédictions correctes : {}" .format(mean_cv_results))
```

```
Taux de prédictions correctes : [0.66847405 0.66846053]
Taux moyen de prédictions correctes : 0.6684672930110257
```

Source : Résultats obtenus grâce à la plateforme Jupyter Notebook

Le SVM effectuant une classification linéaire, nous avons souhaité essayer de prédire correctement notre variable à expliquer via un modèle de régression logistique. Ce fut un échec, avec certes 33.15% de prédictions correctes, simplement car il prédisait toutes les réponses comme fausses.

Modèle de régression logistique sur l'échantillon initial :

```
ypred = result.predict(X)
logreg_pred = list(map(round, ypred))
from sklearn.metrics import (accuracy_score)

print("Taux de prédictions correctes : {}", accuracy_score(train.answered_correctly, logreg_pred))
```

```
Taux de prédictions correctes : {} 0.3315327068512139
```

Source : Résultats obtenus grâce à la plateforme Jupyter Notebook

Nous avons aussi tenté un réseau de neurones à propagation directe : le classifier MLP (MultiLayer Perceptron). Moins efficace pour prédire les réponses fausses, il affichait 66.45% de perceptions correctes en moyenne.

Modèle MLP sur l'échantillon initial :

```
#MLP
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import confusion_matrix
MLP_classifier = MLPClassifier()
MLP_classifier.fit(train, train.answered_correctly)
MLP_pred = MLP_classifier.predict(train)

cv_results2 = cross_val_score(MLP_classifier, train, train.answered_correctly, cv=2)
mean_cv_results2 = cv_results2.mean()
print("Taux de prédictions correctes : {}" .format(cv_results2))
print("Taux moyen de prédictions correctes : {}" .format(mean_cv_results2))
```

```
Taux de prédictions correctes : [0.67063476 0.6582681 ]
Taux moyen de prédictions correctes : 0.6644514303090578
```

Source : Résultats obtenus grâce à la plateforme Jupyter Notebook

Après avoir découpé notre base en deux échantillons (train et test) de manière aléatoire, nous avons tenté de modéliser notre variable à expliquer par un arbre de décision puis par une forêt aléatoire.

Séparation aléatoire de l'échantillon :

```
from sklearn.model_selection import train_test_split

entrain , valid = train_test_split(train, test_size=0.30, random_state=123)

print(entrain.shape)
print(valid.shape)
```

```
(34339, 10)
(14718, 10)
```

Source : Résultats obtenus grâce à la plateforme Jupyter Notebook

Nous cherchions à maximiser le score F1, c'est-à-dire la moyenne pondérée de la précision et la sensibilité du modèle. Le modèle au score F1 le plus haut pour la base d'entraînement (55.01%) maintenait le même ordre de précision pour la base de test (55.49%).

Premier modèle Decision tree :

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import f1_score
clf = DecisionTreeClassifier(max_depth = 20,
                             min_samples_split = 0.3,
                             class_weight = "balanced",
                             random_state=0)

clf.fit(Xtrain, Ytrain)

print("Train F1-score : {}".format(
    f1_score(Ytrain, clf.predict(Xtrain), average='macro')
))

print("Valid F1-score : {}".format(
    f1_score(Ytest, clf.predict(Xtest), average='macro')
))
```

```
Train F1-score : 0.5501286830297364
Valid F1-score : 0.5548623201062225
```

Source : Résultats obtenus grâce à la plateforme Jupyter Notebook

Deuxième modèle Decision tree :

```
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier(max_depth = 30,
                             min_samples_split = 0.3,
                             random_state=0)

clf.fit(Xtrain, Ytrain)

print("Train F1-score : {}".format(
    f1_score(Ytrain, clf.predict(Xtrain), average='macro')
))

print("Valid F1-score : {}".format(
    f1_score(Ytest, clf.predict(Xtest), average='macro')
))
```

```
Train F1-score : 0.5177856033919201
Valid F1-score : 0.516452883846174
```

Source : Résultats obtenus grâce à la plateforme Jupyter Notebook

Random Forest :

```
from sklearn.metrics import f1_score
from sklearn.ensemble import RandomForestClassifier

clf = RandomForestClassifier(random_state=0)
clf.fit(Xtrain, Ytrain)

print("Train F1-score : {}".format(
    f1_score(Ytrain, clf.predict(Xtrain), average='macro')
))

print("Valid F1-score : {}".format(
    f1_score(Ytest, clf.predict(Xtest), average='macro')
))
```

```
Train F1-score : 0.9693493872608543
Valid F1-score : 0.5558739510059382
```

Source : Résultats obtenus grâce à la plateforme Jupyter Notebook

Sur ces mêmes échantillons nous avons retenté notre classifier SVM : le taux de prédictions correctes reste dans le même ordre de grandeur que sur l'échantillon initial : environ 67%.

Modèle SVM sur l'échantillon train :

```
from sklearn import svm
from sklearn.model_selection import cross_val_score

svm_classifier = svm.SVC(kernel = 'rbf')
svm_fit = svm_classifier.fit(Xtrain, Ytrain)
svm_pred = svm_classifier.predict(Xtrain)

cv_results = cross_val_score(svm_classifier, Xtrain, Ytrain, cv=2)
mean_cv_results = cv_results.mean()
print("Taux de prédictions correctes : {}" .format(cv_results))
print("Taux moyen de prédictions correctes : {}" .format(mean_cv_results))
```

```
Taux de prédictions correctes : [0.67047175 0.6705108 ]
Taux moyen de prédictions correctes : 0.6704912787071741
```

Source : Résultats obtenus grâce à la plateforme Jupyter Notebook

Modèle SVM sur l'échantillon test :

```
svm_classifier = svm.SVC(kernel = 'rbf')
svm_fit = svm_classifier.fit(Xtest, Ytest)
svm_pred = svm_classifier.predict(Xtest)

cv_results = cross_val_score(svm_classifier, Xtest, Ytest, cv=2)
mean_cv_results = cv_results.mean()
print("Taux de prédictions correctes : {}" .format(cv_results))
print("Taux moyen de prédictions correctes : {}" .format(mean_cv_results))
```

```
Taux de prédictions correctes : [0.66381302 0.66367713]
Taux moyen de prédictions correctes : 0.6637450740589754
```

Source : Résultats obtenus grâce à la plateforme Jupyter Notebook

De tous nos modèles, il en résulte que la méthode de classification SVM est la plus précise pour notre échantillon.

Prédiction des modèles sur l'échantillon initial :

Modèle	SVM	MLP	Logistic Regression
Taux moyen de prédictions correctes sur l'échantillon initial	66.85%	66.45%	33.15%

Source : Résultats obtenus grâce à la plateforme Jupyter Notebook

Prédiction des modèles sur les échantillons train et test :

Modèle	Decision tree Max depth = 20	Decision tree Max depth = 30	Random forest	SVM
Taux moyen de prédictions correctes sur l'échantillon d'entraînement	55.01%	51.78%	96.93%	67.05%
Taux moyen de prédictions correctes sur l'échantillon test	55.49%	51.65%	55.59%	66.38%

Source : Résultats obtenus grâce à la plateforme Jupyter Notebook

Table des matières