TP Gamification adaptative

BRUNEAU Richard - VASLIN Pierre

18 Janvier 2021

Première partie : Recommandations à partir de profils

Etape 2 : Analyses PLS (PLS Path modeling)

Décrivez/Commentez deux des matrices de résultats de l'analyse PLS

Nous allons considérer les matrices ./Hexad/avatarPathCoefs (ci-dessous *Matrice 1*) et ./Hexad/avatarpVals (ci-dessous *Matrice 2*). La matrice pVals permet de relativiser les informations issues de la première. Nous sommes donc dans le cadre de joueurs qui ont eu à leur disposition l'élément avatar. On constate que pour la variation de la motivation intrinsèque, il n'est pas possible de déterminer quelque chose de significatif puisque dans la matrice 2, les valeurs de la ligne MIVar sont toutes supérieurs à 0,1. En revanche, pour la motivation extrinsèque, nous constatons que des données sont exploitables. On constate par exemple, que la caractéristique "player" a eu une influence de près de 50% avec une fiabilité en la valeur assez haute (0,003). Avec une précision plus faible on peut constater que pour le profil "socialiser" cela a eu une influence négative sur la ME. Les profils "achiever", "freeSpirit", "disruptor" et "philantropist" ont une fiabilité trop faible pour pouvoir garder des informations. Pour la variation de l'amotivation, on ne peut se concentrer uniquement sur le socialiser qui provoque une variation de près de 30%.

On constate que la fiabilité augmente pour les valeurs très franches, en revanche quand la variation est proche de 0, la fiabilité est faible.

Etape 3 : Recommendations à partir des matrices PLS

Notre code est disponible dans le fichier code/vecteursAffinites.py. A l'intérieur du fichier nous avons défini une variable globale qui permet de choisir l'étudiant souhaité. Pour calculer les motivations initiales nous avons pris le partie de simplement sommer les composantes caractéristiques de chacune des motivations. *Exemple: Motivation Intrinsèque Initiale = micoI + miacI + mistI*

Avant de chercher à déterminer le profil, nous avons pris la décision de réaliser une étape de pré-traitement afin de remplacer par 0 les valeurs avec une incertitude trop élevée. Au cours de notre exécution, nous avons fixé ce seuil à 0.05. A nouveau, nous avons déclaré une variable globale `EPSILON` afin de pouvoir paramétrer cela aisément.

Pour déterminer les vecteurs, nous avons décidé de ne pas accorder plus d'importance à l'une des deux motivations. Dans la fonction `strategy`, nous sommons la motivation intrinsèque et extrinsèque en déduisant l'amotivation. De cette façon, on s'assure que la stratégie se concentrera sur la motivation la plus élevée.

Il est possible que le vecteur soit composé uniquement de 0 et de variable inférieur à 0. Cela signifie que nous savons que certains éléments ont un effet négatif et d'autre un effet inconnu. C'est au programmeur de déterminer si il prend des risques en donnant une valeur égale à 0 car elle avait beaucoup d'incertitude ou de prendre la valeur négative la plus grande afin de limiter les pertes. Pour la suite de ce TP, nous avons fait le choix de prendre des risques.

Deuxième partie : Algorithme d'adaptation

Etape 1 : Réflexion à partir d'exemples

Lors de notre étape de réflexion nous nous sommes concentrés sur quatre individus, elevebf12, eleveig14, elevekg10 et elevelf04. Nous avons veillé à choisir deux individus de sexe masculin et deux individus de sexe féminin. Ainsi nous espérons nous protéger d'un éventuel biais qui aurait pu être apporter par le sexe des individus.

Nous avons isolé les cas suivants :

- Cas 1 : Les vecteurs sont les mêmes. Nous choisissons donc le premier élément.
- Cas 2 : Le premier élément des deux vecteurs est le même. Nous choisissons donc le premier élément.
- Cas 3 : Les vecteurs sont inversés. (Le premier élément de l'un est le dernier élément de l'autre et ainsi de suite). La solution n'est pas évidente.
- Cas 4 : Les vecteurs semblent mélangé aléatoirement et aucun élément ne sautent aux yeux. La solution n'est pas évidente.
- Cas 5 : Toutes les valeurs sont égales à 0 ou négative. La solution n'est pas évidente.

Nous avons pris la décision de normaliser les valeurs à l'intérieur des vecteurs. Nous avons affiché les vecteurs de chacun des individus et nous avons constaté que le vecteur de motivation contenait des valeurs positives très élevées en comparaisons au valeur du vecteur du profil Hexad pour la même position au sein du vecteur. Il nous semble donc incongru de comparer ses valeurs entre elles et la normalisation nous semble la meilleure option afin de permettre une comparaison plus neutre.

Notre stratégie consiste à sommer la valeur associé à un label dans chacun des vecteurs. Ainsi nous obtenons la valeurs cumulé pour le label. De ce fait notre algorithme garde l'élément de jeu ayant obtenu le score le plus haut. Cette méthode comporte l'avantage de ne pas donner un poids plus important à l'un des deux vecteurs d'affinités.

Etape 2 : Explication de la stratégie en pseudo-code

```
Titre: Recommendation d'un élément de jeu pour un élève donné
Pré-condition: Les vecteurs sont triés par ordre décroissant,
    mesure la même taille et contiennent les labels de chacune des valeurs.
Entrée: Vecteurs d'affinités vecteurMotiv (pour la motivation)
    et vecteurHexad (pour le profil Hexad).
Sortie: Élément recommandé
Traitement:
    vecteurMotiv ← Normalisation(vecteurMotiv)
    vecteurHexad ← Normalisation(vecteurHexad
    Si PremierLabel(vecteurMotiv) = PremierLabel(vecteurHexad) alors
        Retourner PremierLabel(vecteurMotiv)
    Fin du Si
    max ← 0
    maxLabel ← Null
    Pour chaque label contenu dans vecteurMotiv faire:
        temporaire ← vecteurMotiv[label] + vecteurHexad[label]
        Si (temporaire > max) ou
           (temporaire = max et Aléatoire([0,1]) < 0.5)
        Alors
            max = temporaire
            maxLabel = label
        Fin du Si
    Fin du Pour
    Retourner maxLabel
Fin du traitement
```

Etape 3 : Ecriture de l'algorithme en Python

Notre code est disponible dans le fichier code/recommendation.py. Il s'occupe d'appeler le fichier de la première partie afin de générer les vecteurs d'affinités et de déterminer quel est l'élément de jeu le plus intéressant pour l'étudiant. L'étudiant et la précision sont des variables globales fixées à l'intérieur du fichier.

Nous avons constaté sur l'ensemble des élèves, notre algorithme de recommandation sélectionne fréquemment l'élément "avatar" (214 fois sur 258 étudiants).

Etape 4 : Evaluation de la pertinence

Pour l'évaluation, notre code est disponible dans le fichier code/evaluation.py. Pour chaque élève, nous avons effectué une recommandation. Nous avons avons ensuite séparé les élèves en fonction de l'élément de jeu qu'ils ont obtenu.

Les résultats:

```
Nombre de recommendations correspondantes 40
Nombre de recommendations différente 218
T-test variable : Time
Ttest_indResult(statistic=0.3120572187667718, pvalue=0.7552510515342136)
T-test variable : CorrectCount
Ttest_indResult(statistic=0.020785480495290384, pvalue=0.9834329713880039)
T-test variable : FullyCompletedLessonCount
Ttest_indResult(statistic=-0.43550528439202024, pvalue=0.6635626683054437)
T-test variable : MiVar
Ttest_indResult(statistic=1.8077289108220995, pvalue=0.07182236698157281)
T-test variable : MeVar
Ttest_indResult(statistic=-0.3439330758033224, pvalue=0.7311789409559865)
T-test variable :  amotVar
Ttest_indResult(statistic=0.19182458434294583, pvalue=0.8480316779143454)
```

Le groupe avec l'élément de jeu recommandé contient 40 élèves, tant dit que l'autre groupe en contient 218. Nous constatons que pour l'ensemble des statistiques la pvalue est supérieure à 0,1 et que nous ne pouvons pas conclure sur la réussite ou non de notre recommandation.

Suite a cette absence de conclusion, nous avons essayé diverses approches afin d'obtenir des résultats exploitables. Nous avons changé notre façon de normalisé les valeurs du vecteur hexads et du vecteur de motivation en passant d'une normalisation entre 0 et 1 à une normalisation entre -1 et 1. La pvalue a légèrement diminuée et c'est cela qui explique que nous avons conservé cette normalisation, mais elle reste trop élevée pour pouvoir établir une quelconque conclusion.

Nous avons essayé plusieurs stratégies, à l'origine, nous sommions la motivation intrinsèque à la motivation extrinsèque et nous soustrayons l'amotivation. Nous avons essayé de soustraire deux fois l'amotivation, sans succès. Nous avons donc essayé d'ajouter l'amotivation, mais à nouveau, les résultats n'étaient pas concluants. Nous sommes donc revenu sur notre idée de départ.

Ensuite, nous avons revu notre façon de sélectionner l'élément. L'implémentation est disponible dans le fichier code/recommendation.py avec la fonction `recommendation2`. Ce programme attribue 6 points au premier élément de jeu de chaque vecteur, 5 au second, 4 au troisième et ainsi de suite. Nous sommons pour chaque élément de jeu le nombre de points obtenu avec le vecteur d'affinité Hexad et avec le vecteur d'affinité de Motivation. Ensuite, nous choisissons l'élément de jeu qui a obtenu le plus de points. Malheureusement, cette seconde méthode de sélection n'a pas permis d'obtenir de meilleurs résultats. Comme vous pouvez le voir en exécutant le fichier code/recommendation.py, cette méthode offre un résultat parfois différent.

Nous avons ensuite changé la façon dont nous déterminions les vecteurs d'affinités. Nous réalisons une multiplication de matrice entre la matrice de coefficient de l'élément et la transposé des statistiques de l'étudiant. Nous avons essayé une autre méthode en multipliant d'abord chaque colonne par l'attribut qui lui correspond(présent dans les statistiques de l'étudiant), on réalise ensuite une fusion des colonnes de statistiques

à l'aide d'une moyenne (code disponible dans code/vecteursAffinites.py), mais à nouveau sans réussite. Nous avons donc repris notre méthode de multiplication matricielle.

Enfin, pour conclure sur nos expérimentations, nous avons également joué avec la variable epsilon afin d'être plus ou moins exigeant sur les pValue, mais nous n'avons pas constaté de changements significatifs.