Université Côte d'Azur
Enrico Formenti

Année 2020/21
M1 Info
Exercises NNL

Pre-traitement (Preprocessing)

1 Le problème des données manquantes

Il est assez normal que dans une BD il y ait des données manquantes. Cela peut arriver pour plusieurs raisons. Les causes peuvent être les plus disparates : de l'erreur humain (l'opérateur qui a inséré les données a oublié de les rentrer) ou un senseur qui temporairement n'a pas envoyé ses données, etc. On repère une donnée manquant par un espace vide dans la BD ou par une valeur 'NA'.

Pour palier à ce problème on pourrait s'imagine de mettre 0 (ou une autre valeur) à la place de chaque donnée manquante. Malheureusement, ce n'est pas toujours aussi simple. Dans d'autre cas, il faut mettre en place des procédés spécifiques comme ceux qui font l'objet de ce TP.

Avant de procéder cherchons de mesurer l'impact que les données manquantes ont sur la BD qu'on est en train d'analyser, cela nous guidera sur les choix à faire par rapport à ces données.

Exercice 1

1. Reprenez la base 'Wine reviews' de la dernière feuille ; cette fois-ci faites attention de bien marquer les 'NA' (utilisez le paramètre `na_values` de `read_csv`) ;
2. pour chaque colonne calculez le pourcentage de données absentes ou marquées comme `NA` ;
3. calculez le pourcentage de lignes contenantes une valeur vide ou `NA`.

On considère que ce n'est pas acceptable de effacer les lignes contenantes des données manquantes quand leur impact sur le total dépasse le 4/5%. Si l'on est au dessus de ce pourcentage il faudra choisir des stratégies plus avancées.

Quand la colonne contient exclusivement des valeurs numériques, une première possibilité, facile à mettre en place, de traiter notre problème consiste à remplacer les données manquantes par la valeur moyenne de la colonne.

Exercice 2

1. Pour toute colonne numérique, remplacez les données manquantes par la valeur moyenne de la colonne.
2. Une autre possibilité, consiste à remplacer les valeurs manquantes par la valeur la plus fréquente dans la colonne.

3. Cherchez à comparer les deux solutions précédentes. Que remarquez-vous ?

Le problème est que parfois les deux procédés précédents donnent des résultats trop grossiers en rendant *de facto* les données inutilisables pour des fins de prédiction, par exemple. L'idée serait alors de remplacer une valeur manquante par une similaire que l'on peut trouver dans une entrée qui est similaire à celle qui contient la donnée manquante.

Le problème revient alors à comment définir une notion de *similarité*. Une possibilité consiste à définir une distance entre deux entrées. Supposons d'avoir les entrées suivantes :

$$A = (a_1, a_2, a_3, a_4)$$
$$B = (b_1, b_2, b_3, b_4)$$

alors la distance entre A et B est définie comme suit :

$$d(A, B) = \sqrt{\sum_{i=1}^{4} (a_i - b_i)^2}$$

Cette distance donne une bonne idée de à quel point les deux entrées de notre BD se ressemble (contiennent des valeurs proches). Par contre, on s'aperçoit tout de suite que l'on est confrontés à une problème de mise en application de cette distance. En effet, notre BD contient aussi des données alphanumériques. Comment faire alors ?

Il conviendra de procéder au cas par cas. Tout d'abord on commencera par classer les colonnes de notre BD en fonction des données qu'elles contiennent. On distinguera donc des colonnes *numériques*, *textuelles*, *ensemblistes*, *ordinales*.

Pour les numériques, il n'y a rien à faire comme nous avons déjà vu. Les colonnes textuelles sont celles qui contiennent des textes plus ou moins longs (par exemple la description d'un objet). C'est très difficile d'attribuer à ces colonnes une valeur numérique qui ait une sémantique significative. On les exclura tout simplement de notre calcul de distance.

Une colonne ensembliste contient de valeurs pris dans un ensemble donné qui n'est pas un ensemble de valeurs numériques. Par exemple, une colonne `couleurs` qui contient les valeurs jaune, rouge, vert et bleu est une colonne ensembliste. On pourra alors attribuer des valeurs entières uniques à chaque valeur distinct de la colonne. Pour revenir à l'exemple de la colonne `couleurs` on pourra attribuer à jaune la valeur 1, à rouge la valeur 2, 3 pour le vert et 4 pour le bleu mais toute autre jeu de valeurs entier ou réels conviendra aussi bien (d'au moins à ce stade).

On dit avoir une colonne ordinale quand elle contient des valeurs qui forment un ensemble ordonné. Imaginons par exemple une colonne qui contient la taille d'une robe dans une BD d'un magasin de vêtements, on trouvera par exemple L, XS, XL, S, etc. Dans ce cas on sait qu'il y a un ordre entre ces valeurs et que, par exemple, XS est plus petite que L et que L, à son tour, est plus petite que XL, etc. On va donc attribuer des valeurs numériques à ces données, un peu comme on a fait pour les colonnes ensemblistes mais cette fois-ci on veillera à

choisir des valeurs qui respectent l'ordre des valeurs d'origine. Par exemple, si l'on donne à XL la valeur 3, on fera en sorte que la valeur numérique choisie pour L soit inférieure à 3.

Exercice 3

1. Classez les colonnes de votre BD en *numériques*, *textuelles*, *ensemblistes*, *ordinales*.
2. Autant que possible re-encodez chaque colonne non-numérique avec les procédés donnés dans les paragraphes précédents.
3. Pour toute colonne re-encodée calculez moyenne et variance et stockez les résultats dans un nouveau dataframe qui contiendra le même noms de colonne et uniquement deux lignes : l'une pour les valeurs de la moyenne et l'autre pour la variance.

L'introduction des encodage pour les colonnes ensemblistes risque de biaiser les statistiques que l'on peut faire dessus. En effet, imaginez la situation suivante par rapport à notre exemple avec la colonne `couleur`. Supposons d'avoir vraiment beaucoup de couleurs distincts, 100, par exemple. Supposons aussi d'avoir encodé 'rouge' avec 1 et 'bleu' avec 100. Même si la couleur rouge plus fréquente que la couleur bleu, par biais de leur valeur d'encodage on risque que le bleu soit sur-représenté dans la moyenne par rapport au rouge. Cela peut donc complètement fausser les conclusions qu'on ira tirer des données. Pour réduire le biais d'encodage on pourra utiliser la méthode *one-hot encoding*. L'idée est que l'on va introduire dans notre base autant de nouvelle colonnes que de valeurs distincts dans la colonne d'origine. Chaque nouvelle colonne portera le nom de l'un des valeurs de la colonne d'origine. En reprenant l'exemple précédent on introduira 100 nouvelles colonnes, une portera le nom 'rouge', une autre 'blue', une autre encore 'vert', etc. Chaque nouvelle colonne contiendra une valeur 1 à une certaine ligne si la ligne avait cette couleur à l'origine, 0 sinon.

Exercice 4

1. Implémentez la technique one-hot encoding pour les colonnes ensemblistes de votre base.
2. Quelles sont les limites de la méthode que l'on vient d'introduire ?

Pour terminer, remarquez que quand bien même on avait transformé toutes les colonnes de notre BD en colonnes numériques, quand on va calculer la distance entre deux lignes les valeurs contenus dans certaines colonnes pourraient influencer beaucoup plus la distance que d'autres. Pour éviter cela nous disposons essentiellement de deux techniques de mise à l'échelle pour des valeurs numériques : la *standardisation* et la *normalisation*.

La standardisation prends le max y^M et le min y^m d'une colonne y donnée et on remplacer chaque valeur y_i par

$$\frac{y_i - y^m}{y^M - y^m}$$

Par contre la normalisation s'appuie sur la moyenne μ_y de la colonne y et sur son écart type σ_y. Chaque valeur y_i est remplacé par sa version normalisée comme suit :

$$\frac{y_i - \mu_y}{\sigma_y}$$

Exercice 5

1. Appliquez la normalisation à toutes les colonnes que vous avez re-encodées ainsi que à toute colonne numérique d'origine.
2. Recalculez pour chaque colonne moyenne et variance. Comparez vos résultats avec le dataframe que vous aviez crée au point 3 de l'exercice 3. Que constatez-vous ?
3. Refaites les deux points précédents en utilisant la standardisation au lieu de la normalisation.