Smartphone-Based Recognition of Human Activities and Postural Transitions Data Set

Zoé Horta

Le « problème »

Il s'agit ici de prédire l'activité d'une personne portant un Samsung Galaxy S2 à la taille.

Les 30 volontaires qui ont participé à la récolte de données permettant une telle prédiction on réalisé 3 activités statiques (position debout, position assise et position allongée) et 3 activités dynamiques (marche, marche en montant, marche en descendant). Ces activités ont aussi permis de relever des données concernant les transitions entre deux activités (position debout à assise, assise à debout, assise à allongée, allongée à assise, debout à allongée et allongée à debout). Ce qui porte à 12 le nombre de classes.

Le dataset

Divisé en trois grandes parties :

- RawData :
  - 61 fichiers avec les données du gyroscope (3 colonnes : axes X, Y et Z et entre 17000 et 20000 lignes) ;
  - 61 fichiers avec les données de l'accéléromètre (3 colonnes : axes X, Y et Z et entre 17000 et 20000 lignes) ;
  - 1 fichier qui pour chaque groupe de données, (parfois 1000 lignes, parfois 100) indique la posture/l'activité de l'utilisateur, son identifiant, l'identifiant de l'expérience et l'intervalle de valeurs provenant de l'un des 122 fichiers évoqués précédemment utilisées (5 colonnes, 1214 lignes).
- Train :
  - X_train : 561 colonnes obtenues à l'issue de feature engineering (pas par moi) et 7767 lignes. Les colonnes « fonctionnent », la plupart du temps, par groupe de trois (pour les axes X, Y et Z) ;
  - y_train : 1 colonne indiquant la posture de l'utilisateur et 7767 lignes.
- Test :
  - X_test : 561 colonnes et 3162 lignes ;
  - y_test : 1 colonnes et 3162 lignes.

On a donc un séparation 71%/29% pour train et test. Proche du classique 80%/20%.

Le preprocessing

Le dataset comprenait déjà une partie « train » (X_train et y_train) et une partie « test » (X_test et y_test).

Comme ces 4 fichiers étaient dans 4 fichier .txt distincts, sans le nom des colonnes, qui étaient dans un 5e fichier, j'ai à chaque fois rassemblé les noms de colonnes et les valeurs numériques dans un dataframe.

Mais au moment de d'ajouter les noms de colonnes aux dataframes X_train et X_test, un problème est survenu : certains noms apparaissaient plus d'une fois. J'ai donc écrit une boucle `for` qui m'a permis de remplir un tableau avec le nom de colonne redondants. J'ai ensuite cherché ces nom dans le fichier texte, pour voir où ils se trouvaient et quelle stratégie je devrai adopter pour ma fonction de renommage. J'ai donc écrit deux fonctions qui s'adaptent aux différents cas rencontrés.

Après ça, j'ai pu importer convenablement les données dans les 4 dataframes.

La visualisation des données

J'ai décidé de « comparer » le trio (tBodyAcc-STD-1, tBodyAcc-STD-2, tBodyAcc-STD-3) avec Postures (la colonne des classes). Je me suis dit que l'écart-type était un bon moyen de déterminer l'activité de l'individu. Si l'écart-type est proche de 0, on devine que l'individu est en position statique (assis, debout, allongé) et au contraire, s'il est plus grand ou plus petit que 0, on devine que l'individu est en mouvement.

Les modèles

Comme il s'agit d'un problème de classification, avec des valeurs numériques, j'ai utilisé les modèles suivants :

- `BaggingClassifier`
- `SVC`
- `KNeighborsClassifier`
- `DecisionTreeClassifier`

Pour `SVC`, `KNeighborsClassifier` et `DecisionTreeClassifier`, j'ai utilisé une `SearchGrid` pour « jouer » avec les hyperparamètres.

J'ai ensuite fait une k-fold cross-validation avec ces 4 modèles, afin de les comparer.

Pour une représentation plus visuelle des résultats, j'ai aussi affiché les matrices de confusions des modèles et des boxplots, pour avoir, cette fois-ci, les 4 modèle sur un même graphe.

Pour chacun des modèles, j'ai aussi calculé R^2 . Le nombre de colonnes étant important, je voulais m'assurer que ça ne faussait pas les résultats.

J'ai calculé l'erreur quadratique des 4 modèles. Les modèles se sont révélés assez précis puisque les résultats sont tous très proches de 0.

Enfin, j'ai sauvegardé le modèle le plus performant, en utilisant `joblib`, pour pouvoir l'importer dans mon API Django.

Par curiosité…

Après avoir effectué les différentes opérations expliquées précédemment sur les subsets déjà existants, je me suis intéressée à la corrélation entre les features et la cible.

J'ai fixé α à 0.05 et je n'ai gardé que les colonnes dont la valeur du coefficient de corrélation de Pearson y était inférieure.

J'ai donc gardé 457 colonne sur 561.

J'ai refait tourner les 4 modèle sur ces nouveaux jeux de données, les résultats étaient quasiment les mêmes qu'avec les données de base.

L'API

Pour l'API, j'ai choisi le modèle avec les meilleurs résultats : SVC (C-Support Vector Classification).

J'ai testé l'API avec des commandes CURL et aussi sur Postman.


```
    "fBodyGyroJerkMagIQR1": -0.8980968545120932,
    "fBodyGyroJerkMagropy1": -0.2348152901855691,
    "fBodyGyroJerkMagMaxInds1": -1.0,
    "fBodyGyroJerkMagMeanFreq1": 0.1228301368024483,
    "fBodyGyroJerkMagSkewness1": -0.3456843742823689,
    "fBodyGyroJerkMagKurtosis1": -0.7090872022899664,
    "tBodyAccAngleWRTGravity1": 0.006462402864424899,
    "tBodyAccJerkAngleWRTGravity1": 0.162919820165899,
    "tBodyGyroAngleWRTGravity1": -0.8258856226927339,
    "tBodyGyroJerkAngleWRTGravity1": 0.2711514521489811,
    "tXAxisAccAngleWRTGravity1": -0.7205591038773121,
    "tYAxisAccAngleWRTGravity1": 0.2767794147286349,
    "tZAxisAccAngleWRTGravity1": -0.0510740294455464,
    "posture": 5.0
}
```


Fin de la réponse renvoyée par l'API, avec la valeur prédite de "posture"

Conclusion

Ce devoir maison était mon premier projet de machine learning et c'était aussi la première fois que je faisais ma propre API. Malgré mes connaissances assez limitées en machine learning, je suis contente d'avoir réussi à faire ce projet.