Compte-rendu du brief : Meilleur Classifieur Pour La Reconnaissance Des Digits Audios

Réalisé par : Sanchez Pauline

Mars 2022

Fichiers à utiliser :

- Digits_Recognition_complet.ipynb
- Tools2.py (fourni dans le brief puis modifié par la suite)
- DataSet_lo__.csv
- modele_rfm.pkl

Le fichier DataSet__lo__.csv est un dataset crée à partir des cinq enregistrements audios des chiffres de 0 à 9.

Le fichier Digits_Recognition_complet.ipynb est un Jupyter Notebook qui va de l'importation des données et des différentes bibliothèques à la sauvegarde du modèle KNN et à son utilisation.

Description du fichier Digits_Recognition_complet.ipynb :

- Importation des bibliothèques
- Collection du data grâce à la fonction collection() qui utilise le fichier Tools2.py
- Importation du Dataset
- Analyse du dataset (aperçu, taille)
- Analyse des données manquantes, valeurs quantitatives, valeurs qualitatives
- Visualisation de la colonne Target
- Séparation des données en features : X et target : y
- Séparation des données en fonction des données d'entraînement et de test (80% pour l'entraînement, 20% pour le test)
- Début de la recherche du meilleur modèle. On va créer un pipeline pour chaque modèle (KNN, SVM, arbre de décision, forêt aléatoire, XGBoost) et ce pipeline va permettre de tester différents scalers et différents hyperparamètres (propres à chaque modèle) pour chaque modèle. Ainsi, à la fin des entraînements des différents modèles, on obtient, pour chaque modèle, le meilleur taux d'accuracy avec quels hyperparamètres.
- Pour le **modèle KNN** on peut obtenir une accuracy optimale de **0.84** avec comme hyperparamètres : {'scalers': 'standard', 'dim_red': None, 'knn_n_neighbors': 3, 'knn_metric': 'minkowski', 'knn_weights': 'uniform', 'leaf_size': 20, 'p': 2}
- Pour le **modèle arbre de décision** on peut obtenir une accuracy optimale de **0.67** avec comme hyperparamètres : {'scalers': 'minmax', 'dim_red': None, 'criterion': 'entropy', 'max depth': 160}
- Pour le **modèle SVM** on peut obtenir une accuracy optimale de **0.66** avec comme hyperparamètres : {'scalers': 'standard', 'dim_red': None, 'kernel': 'rbf', 'gamma': 69, 'C': 247}
- Pour le **modèle de forêt aléatoire** on peut obtenir une accuracy optimale de **0.82** avec comme hyperparamètres : {'scalers': 'standard', 'dim_red': None, 'max_depth': 70, 'min_samples_leaf': 2, 'min_samples_split': 4, 'n_estimators': 108}
- Pour le modèle **XGBoost** on peut obtenir une accuracy optimale de **0.55** avec comme hyperparamètres : {'dim_red': None, 'booster': 'dart', 'eval_metric': 'mlogloss', 'max_depth': 7}
- Le meilleur modèle est donc le KNN avec une accuracy de 0.84.

- On applique le modèle KNN à notre dataset (avec le scaler et les hyperparamètres trouvés précédemment)
- On réalise une matrice de confusion du résultat
- Enregistrement du modèle KNN grâce à joblib sous le nom : « modele_rfm.pkl »
- On utilise notre modèle pour la reconnaissance. La fonction rec() permet de lancer l'enregistrement et de donner la prédiction grâce à notre modèle.
- On réalise différents graphiques afin d'observer la répartition de données. Tout d'abord sur l'ensemble du dataset puis sur deux chiffres qui semblent se distinguer au premier coup d'œil.

Résultats :

Notre modèle fonctionne de manière aléatoire malgré son accuracy de 0.84. Il serait nécessaire d'augmenter la taille du dataset et de réaliser des enregistrements de meilleure qualité.

```
FrozenTrial(number=19, values=[0.8400000000000001], datetime_start=datetime.datetime(2022, 3, 22, 12, 58, 31, 484464),
datetime_complete=datetime.datetime(2022, 3, 22, 12, 58, 31, 558461), params={'scalers': 'standard', 'dim_red': None, 'knn_n_neighbors': 3,
'knn_metric': 'minkowski', 'knn_weights': 'uniform', 'leaf_size': 20, 'p': 2}, distributions={'scalers': CategoricalDistribution(choices=('minmax',
'standard')), 'dim_red': CategoricalDistribution(choices=('PCA', None)), 'knn_n_neighbors': IntUniformDistribution(high=19, low=1, step=2),
'knn_metric': CategoricalDistribution(choices=('euclidean', 'manhattan', 'minkowski')), 'knn_weights': CategoricalDistribution(choices=('uniform',
'distance')), 'leaf_size': IntUniformDistribution(high=40, low=20, step=1), 'p': IntUniformDistribution(high=2, low=1, step=1)}, user_attrs={},
system_attrs={}, intermediate_values={}, trial_id=19, state=TrialState.COMPLETE, value=None)
```

Les données récoltées à la fin de notre pipeline spécial KNN afin de configurer notre meilleur modèle.

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform (X_test)

classifier = KNeighborsClassifier(n_neighbors= 3, metric= 'minkowski', weights= 'uniform', leaf_size=20, p=2)
classifier.fit(X_train,y_train)
y_predict = classifier.predict(X_test)
from sklearn.metrics import accuracy_score,classification_report,confusion_matrix
accuracy_score(y_test,y_predict)
✓ 0.1s                                                                                                    Python
0.8
```

Notre meilleur modèle avec ses hyperparamètres et la standardisation des données.