Reconnaissance des digits

Erwan Tanguy

17 mars 2022

Sommaire

Acquisition des données

Le travail s'effectue par trinôme.

Les chiffres de 0 à 9 sont enregistrés avec différents micros. Les micros ne sont pas calibrés, les sons ne sont pas compressés et retravaillés.

Les enregistrements sont effectués par le module SoundDevice de Python selon une durée de 2 secondes, modifiée par la suite à une seconde afin d'avoir d'avantage de voix et moins de bruits parasites du micro.

La fréquence d'échantillonnage est définie à 48000 frames par secondes, avec un seul canal. L'enregistrement de chaque chiffre est dans un format numpy, un calcul assure le format qui permet le traitement (16 bytes).

Le module python_speech_features :

- Calcule la transformée de Fourier
- Pondère le spectre d'amplitude
- Calcule la transformée en cosinus discrète
- Retourne un tableau Numpy

Le module Pandas exporte les tableaux numpy dans un fichier csv.

Analyse des données

Le jeu de données est complet, et les données sont bien dans un format numérique.

Il y a 44 (x 10 digits de 0 à 9) variables disponibles pour l'apprentissage et le test pour le csv complet, 24 pour la sélection seule de mes enregistrements (sélectionnée par pandas.iloc).

L'analyse des minimas et maximas démontre la nécessité de mettre les variables sur une même échelle. La stratégie choisie est la standardisation avec StandardScaler.

Le jeu de données est divisé en 80% de données d'entraînement, 20% de données pour le test.

Les répartitions des cibles est équilibrée dans ces deux jeux.

Différentes sélection de variables :

1. Tout le dataframe
2. Selon la corrélation de Pearson :

	Fe1	Fe2	Fe3	Fe6	Fe7	Target
0	14.274666	0.306604	6.509358	8.236848	4.603504	0.0
1	14.075767	4.393329	6.786652	8.307470	0.646492	1.0
2	13.385394	2.412716	4.905458	11.484525	0.222476	2.0
3	14.335635	3.923458	8.288954	2.935059	5.941685	3.0
4	12.976791	7.668485	9.849166	7.288091	0.330541	4.0

Création des modèles

Voici les différents classifieurs initialisés :

- DecisionTreeClassifier
- RandomForestClassifier
- GradientBoostingClassifier (écarté, résultats non probants et trop de temps d'exécution)
- MLPClassifier
- KNeighborsClassifier
- XGBClassifier
- SVM

Des pipelines sont créés pour chaque classifieur, avec une stratégie de standardisation des données.

Les évaluations se font ensuite avec GridSearchCV, afin d'évaluer plusieurs paramètres pour chaque classifieur.

Pour chaque modèle une matrice de confusion permet d'apprécier les bonnes prédictions et les erreurs de classifications.

Les différents pipelines sont testés avec les deux sélections de variables énoncés précédemment.

Les résultats avec les variables sélectionnées selon la corrélation de Pearson ne sont pas satisfaisants, ce jeu de données est écarté des futurs tests.

Essais

1 – avec données personnelles recopiées plusieurs fois pour avoir plus de données :

- les matrices de confusions donnent des résultats fiables (aucune erreur de classification)

- mauvais résultats dans l'application

Le problème est que les données de test ont forcément été entrainées puisque le jeu de données est répété plusieurs fois.

Cette stratégie est naturellement écartée.

2 – avec données personnelles :

```
Decision Tree
--Résultat du Train : 1.0
--Résultat du Test : 0.4583333333333333
Random Forest
--Résultat du Train : 1.0
--Résultat du Test : 0.6875
MLP classifier
--Résultat du Train : 1.0
--Résultat du Test : 0.75
KNN
--Résultat du Train : 1.0
--Résultat du Test : 0.6458333333333334
XGB classifier
--Résultat du Train : 0.890625
--Résultat du Test : 0.7083333333333334
SVC
--Résultat du Train : 0.875
--Résultat du Test : 0.5833333333333334
```

3 – Avec données provenant du trinôme :

```
Decision Tree
--Résultat du Train : 1.0
--Résultat du Test : 0.5113636363636364
Random Forest
--Résultat du Train : 1.0
--Résultat du Test : 0.6363636363636364
MLP classifier
--Résultat du Train : 1.0
--Résultat du Test : 0.7840909090909091
KNN
--Résultat du Train : 1.0
--Résultat du Test : 0.6931818181818182
XGB classifier
--Résultat du Train : 0.7670454545454546
--Résultat du Test : 0.6477272727272727
SVC
--Résultat du Train : 0.7471590909090909
--Résultat du Test : 0.6590909090909091
```

Analyse des solutions

Les classifieurs qui fonctionnent le mieux pour mes données sont :

1 - le MPLClassifier

2 - le XGBClassifier.

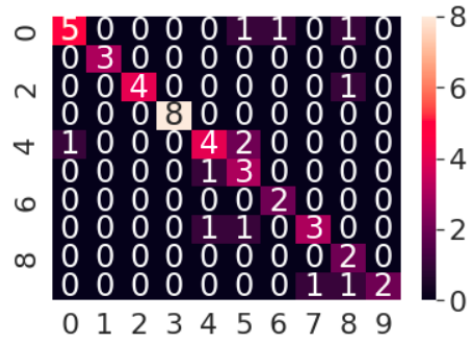
Le classifieur qui fonctionne le mieux pour le dataframe du trinôme est le MLP Classifier.

3 - le MPLClassifier

Dans l'ordre des classifieurs ci-dessus, voici les matrices de confusions et la liste des meilleurs hyper-paramètres:

```
meilleur score : 1.0
meilleur score : 0.75
meilleurs paramètres : {'mlpclassifier__learning_rate_init': 0.007, 'mlpclassifier__max_iter': 5000, 'mlpclassifie
r__random_state': 50, 'mlpclassifier__warm_start': False}

<AxesSubplot:>
```




```
meilleur score : 0.8854166666666666
meilleur score : 0.7083333333333334
meilleurs paramètres : {'xgbclassifier__booster': 'gblinear'}

<AxesSubplot:>
```


```
meilleur score : 1.0
meilleur score : 0.7840909090909091
meilleurs paramètres : {'mlpclassifier__learning_rate_init': 0.002, 'mlpclassifier__max_iter': 5000, 'mlpclassifie
r__random_state': 50, 'mlpclassifier__warm_start': False}

<AxesSubplot:>
```

Conclusion

Les MLP Classifieurs ont de meilleurs résultats.

Néanmoins, les essais dans l'application ne sont pas concluants ; environ 60% de bons résultats (avec un micro externe).

Des données plus importantes, plus diversifiées, un matériel de meilleure qualité, ainsi qu'un traitement sonore plus poussé permettrait sans aucun doute d'obtenir de bien meilleurs scores !