

Projet N°12

Mars 2022



RAPPORT

Meilleur Classifieur pour La reconnaissance des Digits audios

***Développer un modèle IA pour la reconnaissance des
Digits provenant d'un enregistrement audio***

Auteur : Maïna LE DEM

Indice: A

Date d'émission: 22/03/2022



SOMMAIRE

Partie 1 : Base de données, Analyse, Prétraitement et Préparation....	Erreur ! Signet non défini.
1.1 - Fusionner les fichiers réponses.....	Erreur ! Signet non défini.
1.2 - Analyse, Prétraitement et Préparation.....	Erreur ! Signet non défini.
Partie 2 : Développement et entraînement d'un modèle KNN	Erreur ! Signet non défini.
2.1 - KNN From Scratch.....	Erreur ! Signet non défini.
2.2 - KNN Sklearn	Erreur ! Signet non défini.
Partie 3 : Mettre en place la solution dans l'application de test de personnalité ..	Erreur ! Signet non défini.
3.1 - Enregistrement du meilleur modèle avec « joblib » :	5
3.2 - Intégration de la solution à l'Application	5
3.3 - Comparaison du résultat avec et sans IA.....	Erreur ! Signet non défini.
Partie 4 : Conclusion	6



Partie 1 : Base de données, Analyse, Prétraitement et Préparation

Après réalisation des enregistrements.

1.1 - Analyse :

- La data obtenu contient 80 lignes non-null de float : Normal puisque nous avons enregistré le jeu de données
- 1 numéro est associé à 12 entées (Fe1 à Fe12)
- Nous avons donc un jeu de données équilibré par rapport à la target.

1.2 - Prétraitement et Préparation

Sans objet, puisque que le jeu de données est homogène, de types quantitatifs et sans valeurs nulles.

Partie 2 : Pipeline Apprentissage/Classification Supervisée

2.1 - Recherche du meilleur modèle

1) Création d'une liste de modèle

```
model_name = ['SVM', 'KNN', 'DecisionTreeRegressor', 'DecisionTreeClass',  
              'RandomForest', 'XGBoost']
```

2) Création de la liste des pipeline

```
pipelines = [Pipeline([('scaler', StandardScaler()), ('svc', SVC()),]),  
             Pipeline([('scale', StandardScaler()), ('KNN', KNN()),]),  
             Pipeline([('scale', StandardScaler()),  
                       ('DecisionTreeRegressor', DTR()),]),  
             Pipeline([('scale', StandardScaler()), ('DecisionTreeClass', DTC()),]),  
             Pipeline([('scale', StandardScaler()), ('RandomForest', RFC()),]),  
             Pipeline([('scaler', StandardScaler()), ('XGB', xgb())],)]
```

3) Création de la liste de dictionnaire avec les hyperparamètres par classifieur

```
parametres = [{ 'svc__kernel': ['linear', 'rbf'], 'svc__C':[1, 10]},  
               {'KNN__n_neighbors' : np.arange(1,15),  
                'KNN__metric' : ['minkowski','euclidean','manhattan'],  
                'KNN__algorithm' : ['auto', 'ball_tree', 'kd_tree', 'brute']}],  
               {'DecisionTreeRegressor__criterion':['mae'],  
                'DecisionTreeRegressor__max_depth': np.arange(2,8),  
                'DecisionTreeRegressor__max_leaf_nodes': np.arange(1,10),  
                'DecisionTreeRegressor__ccp_alpha': [0,0.1,0.2,0.3,0.4]}],
```



```

{'DecisionTreeClass__max_depth': np.arange(2,8),
 'DecisionTreeClass__max_leaf_nodes': np.arange(1,10),
 'DecisionTreeClass__ccp_alpha':[0,0.1,0.2,0.3,0.4],},
{'RandomForest__n_estimators': np.arange(5, 50, 5),
 '#RandomForest__min_samples_split':[2, 3, 10]},
 'RandomForest__max_depth': np.arange(2,8),
 'RandomForest__max_leaf_nodes': np.arange(2,10)},
{'XGB__eta' : [0.1, 0.2, 0.3, 0.4],
 'XGB__max_depth' : [2, 4, 6, 8, 10],
 'XGB__eval_metric' : ['mlogloss']},
]

```

4) Entraînement des modèle avec le GridSearch pour calculer le modèle obtenant le meilleur résultat

```

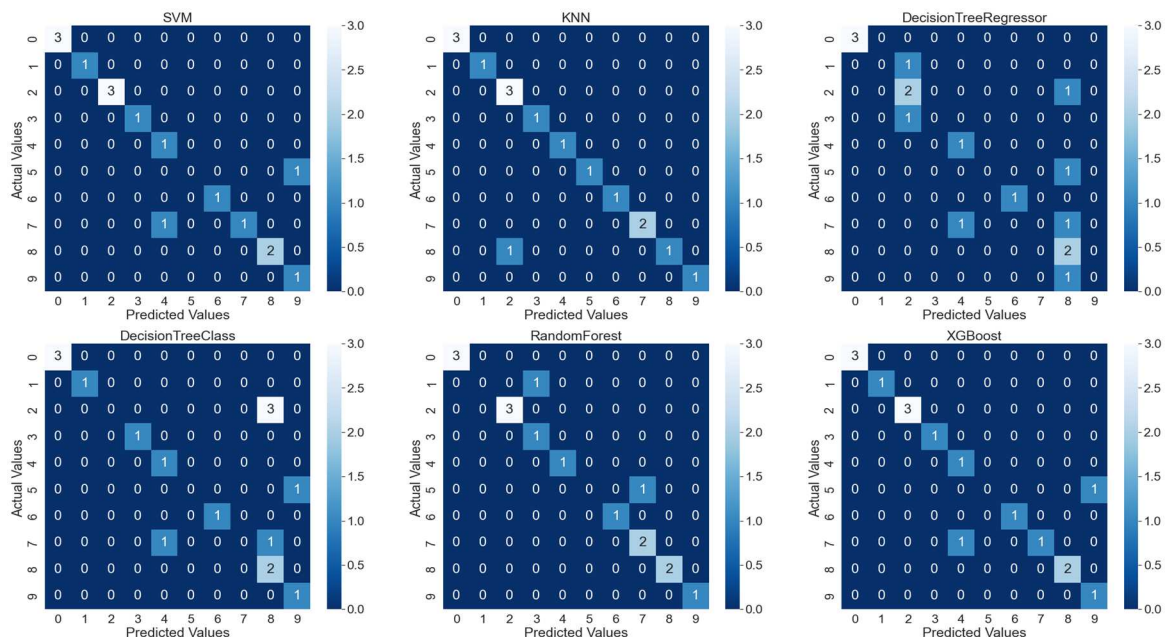
cm_list = []
score = []
best_model = []
best_grid = []

for model, p, name in zip(pipelines, parametres, model_name):
    exp = GridSearchCV(model, param_grid = p,
                        cv = 3, scoring='accuracy', return_train_score=False,
                        verbose=1)
    grid_search = exp.fit(X_train, y_train)
    y_pred = grid_search.predict(X_test)
    #best_model = grid_search.best_estimator_

    # Récupérer le modèle avec les meilleurs paramètres
    score.append(grid_search.score(X_test, y_test))
    cm_list.append(confusion_matrix(y_test, y_pred))
    best_model.append(grid_search.best_estimator_)
    best_grid.append(grid_search.best_params_)

```

5) Affichage des matrices de confusion par modèle avec les meilleurs paramètres





6) Affichage des meilleurs résultats et définition du meilleur modèle

	Model	Score	best_model	best_grid
1	KNN	0.9375	(StandardScaler(), KNeighborsClassifier(metric...	{'KNN__algorithm': 'auto', 'KNN__metric': 'man...
0	SVM	0.8750	(StandardScaler(), SVC(C=1, kernel='linear'))	{'svc__C': 1, 'svc__kernel': 'linear'}
4	RandomForest	0.8750	(StandardScaler(), (DecisionTreeClassifier(max...	{'RandomForest__max_depth': 4, 'RandomForest__...
5	XGBoost	0.8750	(StandardScaler(), XGBClassifier(base_score=0....	{'XGB__eta': 0.3, 'XGB__eval_metric': 'mloglos...
3	DecisionTreeClass	0.6250	(StandardScaler(), DecisionTreeClassifier(ccp_...	{'DecisionTreeClass__ccp_alpha': 0, 'DecisionT...
2	DecisionTreeRegressor	0.5625	(StandardScaler(), DecisionTreeRegressor(ccp_a...	{'DecisionTreeRegressor__ccp_alpha': 0, 'Decis...

```
2 best_grid_final
{'KNN__algorithm': 'auto', 'KNN__metric': 'manhattan', 'KNN__n_neighbors': 5}
```

```
2 best_model_final
Pipeline(steps=[('scale', StandardScaler()),
                 ('KNN', KNeighborsClassifier(metric='manhattan'))])
```

Partie 3 : Mettre en place la solution dans une application Test Temps Réel

3.1 - Enregistrement du meilleur modèle avec « joblib » :

```
import joblib

filename = './Tools/model_pred'
joblib.dump(best_model_final, filename)
nom_du_modele = joblib.load(filename)
```

3.2 - Intégration de la solution à l'Application

Mise en place de la solution dans le fichier test, renommé en **tool_pred.py** :

```
import joblib

# Digit Recognition
def rec2():
    print("Attention, l'enregistrement commence dans :")
    (rate, sig) = wav.read("./Tools/beep-04.wav")
    sd.play(sig, rate)
    for i in range(0, 6):
        time.sleep(1)
        print(5-i)
    time.sleep(1)
    rate = 48000
    duration = 2
    print("Prononcer votre Digit : ")
    data = sd.rec(int(duration * rate), samplerate=rate, channels=1)
```



```
sd.wait()
data = data / data.max() * np.iinfo(np.int16).max
data = data.astype(np.int16)
mfcc_feat = np.mean(mfcc(data, rate, numcep=12), axis=0)
mfcc_feat = np.expand_dims(mfcc_feat, axis=0)

classifier = joblib.load('model_pred')
pred = classifier.predict(mfcc_feat)
print('-----')
print('Digit : ', pred[0])
print('-----')
```

3.3 - Essais

J'exécute la fonction rec2 sur le notebook « Résultat en temps réel ».

Exemple de résultat :

```
: 1 from Tools.tools import rec2
  2 rec2()

Attention, l'enregistrement commence dans :
5
4
3
2
1
0
Prononcer votre Digit :
-----
Digit : 3.0
-----
```

Dans ce cas le résultat est conforme.

Il a été 2 fois sur 5

Partie 4 : Conclusion

Le jeu de données a été enregistré avec un micro.

Il est uniforme, mais ne comporte pas beaucoup d'exemple (8 par nombre).

J'obtiens un modèle moins performant que prévu.

Il serait intéressant d'essayer avec seulement 2 chiffres de la target

Et/ou de sélectionner et réduire le jeu de données selon leur corrélation.