Rapport projet :

Classifieur pour la reconnaissance de digits audios

Objectif du projet :

Obtenir le meilleur classifieur pour reconnaître des digits audios à partir d'un jeu de données créé pour l'occasion.

Création du jeu de données :

Il a fallu enregistrer les nombres de 0 à 9 plusieurs fois pour obtenir un jeu de données exploitable. C'était une partie cruciale du projet puisque la qualité des données était primordiale pour la suite. Le problème était d'obtenir une bonne qualité d'enregistrement à cause du bruit ambiant (pollution sonore élevée) et de la qualité des micros. Le ton et la diction se sont également révélés décisifs : un enregistrement mal prononcé ou brouillé pouvait fausser l'ensemble.

Visualisation du dataframe :

	Fe1	Fe2	Fe3	Fe4	Fe5	Fe6	Fe7	Fe8	Fe9	Fe10	Fe11	Fe12	Target
0	8.769831	-28.132754	-7.217933	-2.926102	2.479673	-9.801402	-0.709304	-2.712402	-3.349622	-0.302495	-1.934686	-0.776286	0.0
1	9.101204	-27.712503	-7.402189	-7.684519	-3.144193	-11.474646	0.909369	3.870697	6.033722	4.852083	0.816946	-0.086821	1.0
2	8.269127	-28.096719	-4.344011	-5.690030	-0.695199	-6.964167	1.608848	-0.066286	1.533915	0.957556	-3.404431	-2.244226	2.0
3	8.952217	-25.828383	-3.867474	-2.178898	-0.195159	-13.190862	-4.091815	-2.099132	0.560830	3.301503	1.377358	-0.666695	3.0
4	9.360507	-25.953896	-9.181735	-5.169506	-0.276754	-12.081353	1.247758	1.873248	1.063382	3.797565	1.554781	-1.630786	4.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
148	15.102664	-28.235252	-4.492319	-4.023210	1.178125	-7.598198	0.878794	-2.118058	-2.574859	-0.634327	-1.241931	-1.183252	8.0
149	10.129180	-22.451345	1.847628	-3.316869	-0.030938	-5.764195	1.284451	1.734123	2.107894	2.278862	-0.228384	-0.600783	9.0
150	15.034194	-34.597553	-7.789384	-9.458021	-2.478272	-8.380656	0.804191	0.064931	0.899076	0.601875	1.008121	0.102041	0.0
151	12.063015	-33.331564	-7.149928	-7.889918	-1.927351	-9.324026	0.304137	-1.215918	-0.214500	-0.193237	-0.701778	-0.598745	1.0
152	13.018290	-37.300417	-4.942152	-10.585364	-4.900305	-8.113873	-0.961918	0.488839	1.450137	1.604482	2.980293	1.826622	2.0

153 rows × 13 columns

Informations sur le dataframe :

```
: mydata.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 153 entries, 0 to 152
Data columns (total 13 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Fe1     153 non-null    float64
 1   Fe2     153 non-null    float64
 2   Fe3     153 non-null    float64
 3   Fe4     153 non-null    float64
 4   Fe5     153 non-null    float64
 5   Fe6     153 non-null    float64
 6   Fe7     153 non-null    float64
 7   Fe8     153 non-null    float64
 8   Fe9     153 non-null    float64
 9   Fe10    153 non-null    float64
 10  Fe11    153 non-null    float64
 11  Fe12    153 non-null    float64
 12  Target  153 non-null    float64
dtypes: float64(13)
memory usage: 15.7 KB
```

Il n'y a pas de valeurs nulles ou manquantes dans notre jeu de données.

Partage data entre X et y :

```
X = mydata.iloc[:,:-1]
y = mydata['Target']

print(X.shape)
print(y.shape)
```

```
(153, 12)
(153,)
```

Split data (train/test) et standardisation :

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
print(scaler.fit(X_train))
```

```
StandardScaler()
```

Grid search :

Création d'un grid search sous la forme :

gs = GridSearchCV(cls, param, cv=3).fit(X_train, y_train)

où 'cls' est le classifieur et 'param' les paramètres à varier selon la technique demandée.

On peut ensuite :

- Afficher les meilleurs paramètres : gs.best_params_
- Afficher le meilleur score (de la validation croisée) : gs.best_score_
- Afficher le meilleur score sur le Test : gs.score(X_test, y_test)

Pipeline :

Création d'un pipeline pour tester plusieurs modèles.

```
param_range = [1, 2, 3, 4, 5, 6]
n_estimators = [50,100,150]
learning_rates = [.1,.2,.3]



pipeline_lr=Pipeline([('scalar1',StandardScaler()),

                      ('lr_classifier',LogisticRegression())])

pipeline_dt=Pipeline([('scalar2',StandardScaler()),

                      ('dt_classifier',DecisionTreeClassifier())])

pipeline_svm = Pipeline([('scalar3', StandardScaler()),

                       ('clf', svm.SVC())])

pipeline_knn=Pipeline([('scalar4',StandardScaler()),

                      ('knn_classifier',KNeighborsClassifier())])

pipeline_randomforest = Pipeline([('pca4', PCA(n_components=2)),
                      ('randomforest_classifier',RandomForestClassifier())])

pipeline_xgboost = Pipeline([('scalar5', StandardScaler()),
                      ('xgboost',XGBClassifier())])

xgb_param_grid = [{'XGBlearning_rate': learning_rates,
                   'XGBmax_depth': param_range,
                   'XGBmin_child_weight': param_range[:2],
                   'XGBsubsample': param_range_fl,
                   'XGB__n_estimators': n_estimators}]
```

Le pipeline n'est pas ici unique : il aurait fallu rassembler tous les modèles de classification dans un seul pipeline.

```
pipelines = [pipeline_lr, pipeline_dt, pipeline_svm, pipeline_knn, pipeline_randomforest, pipeline_xgboost]
pipe_dict = {0: 'Logistic Regression', 1: 'Decision Tree', 2: 'Support Vector Machine', 3:'K Nearest Neighbor', 4: 'Randomforest

for pipe in pipelines:
  pipe.fit(X_train, y_train)
for i,model in enumerate(pipelines):
    print("{} Test Accuracy:{}".format(pipe_dict[i],model.score(X_test,y_test)))
```

```
[14:44:28] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115: Starting in XGBoost 1.3.
0, the default evaluation metric used with the objective 'multi:softprob' was changed from 'merror' to 'mlogloss'. Explicitly s
et eval_metric if you'd like to restore the old behavior.
Logistic Regression Test Accuracy:0.6129032258064516
Decision Tree Test Accuracy:0.25806451612903225
Support Vector Machine Test Accuracy:0.6451612903225806
K Nearest Neighbor Test Accuracy:0.6451612903225806
Randomforest Test Accuracy:0.22580645161290322
Xgboost Test Accuracy:0.41935483870967744
```

La boucle for parcourt chaque pipelines créés et l'applique à X_train et y_train. Puis une deuxième boucle affiche le nom du modèle et son résultat.

Application :

Le but est de tester notre modèle pour voir s'il parvient à reconnaître correctement les digits.

```
for i in range(0,6):
    time.sleep(1)
    print(5-i)


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)
pred = pipeline_lr.predict(mfcc_feat)
print('------------------')
print('Digit : ', pred[0])
print('------------------')
```

Conclusion du projet :

Nous n'avons pas réussi à obtenir de résultats satisfaisants sur notre jeu de données. La plupart du temps, le système se trompait en se focalisant sur un seul chiffre.