COMPTE RENDU

Ceci est un compte rendu de notre projet d'IA numérique.

Nous allons donc voir les étapes dans la construction de notre IA et aussi les difficultés que nous avions pu rencontrer.

Première étape :

Données : collecte, pré-traitement et choix de la prédiction

A force de recherche et ne pouvant pas trouver un dataset qui convenait parfaitement à notre IA symbolique, nous avons un peu dérivé du principe de base mais nous sommes restés dans le même thème.

Voici donc la database que nous avions trouvé :

https://github.com/hesther/reactiondatabase/blob/main/data/e2sn2.csv

Elle contenait au total 3626 samples, afin de pouvoir la rendre plus lisible et pour que les informations soient mieux réparties nous avons séparé les réactifs des produits afin de les classer en quatres catégories : prod1AAM, prod2AAM, react1AAM, react2AAM puis la catégorie ea pour l'électronégativité. Afin de rendre les informations plus lisibles, nous avons utilisé la librairie pysmiles qui permets de lire les informations que la database contient et nous avons décrit tous les atomes en fonction de leur numéro atomique.

Voici la partie pré-traitement des données

```
fields = ["prod1AAM", "prod2AAM", "react1AAM", "react2AAM", "ea"]
dict_trad = []
with open('dataset2.csv', 'r') as f:
    reader = csv.DictReader(f)
    for row in reader:
        ea = str(round(float(row['ea'])))
        mols = [pysmiles.read_smiles(row["prod1AAM"]),pysmiles.read_smiles(row["prod2AAM"]),
        pysmiles.read_smiles(row["react1AAM"]),pysmiles.read_smiles(row["react2AAM"])]
        readed = []
        for mol in mols:
            readed.append(mol.nodes(data='element'))
        atom_trad = ""
        row_trad = []
        for _ in readed:
                for atom in _:
                    atom_trad += str(periodic_table[atom[1]])
                row_trad.append(atom_trad)
                atom_trad = ""
        row_trad.append(ea)
        dict_trad.append(row_trad)
        row_trad = []
print(dict_trad)

with open("dataset.csv" , "w") as f1:
    writer = csv.writer(f1)
    writer.writerow(fields)
    writer.writerows(dict_trad)"""
```

```
print(lists)
dict = []
import csv
with open('dataset.csv', "r+") as f:
    reader = csv.DictReader(f)
    for row in reader:
        el = []
        for _ in row.values():
            el.append(_)
        if int(row["ea"]) <= 15:
            el.append(1)
        elif int(row["ea"]) <= 30:
            el.append(2)
        elif int(row["ea"]) <= 45:
            el.append(3)
        elif int(row["ea"]) <= 60:
            el.append(4)
        dict.append(el)


with open("enfinnnn.csv", "w") as f2:
    writer = csv.writer(f2)
    writer.writerow(["prod1AAM", "prod2AAM", "react1AAM", "react2AAM", "ea", "category"])
    writer.writerows(dict)
```

```
    for mleaf in [1,2,3,4,5,6,7,8,9,10,11,12,13,14]:
        model = DecisionTreeClassifier(min_samples_leaf=mleaf)
        model = model.fit(X_train, y_train)
        val.append(model.score(X_test, y_test)))
    ev.append(val)
print(ev)
vals=[0,0,0,0,0,0,0,0,0,0,0,0,0,0]
for it in ev:
    for i in range(len(it)):
        vals[i] = (vals[i] + it[i])/2
print(vals)
print(max(vals))
```

Deuxième étape :

Entrainement, paramétrage et test + Prédiction scénario d'usage

Voici la partie entrainement des hyperparamètres :

```
ev = []
print("-"*20)
for i in range(100):
    val =[]
    for mdepth in [1,2,3,4,5,6,7,8,9,10,11,12,13,14]:
        model = DecisionTreeClassifier(max_depth=mdepth)
        model = model.fit(X_train, y_train)
        val.append(model.score(X_test, y_test)))
    ev.append(val)

moy = []
for i in range(100):

    val =[]
    for criter in ["entropy", "gini", "log_loss"]:
        model = DecisionTreeClassifier(criterion=criter)
        model = model.fit(X_train, y_train)
        val.append(model.score(X_test, y_test))
    moy.append(val)

print(ev)
vals=[0,0,0,0,0,0,0,0,0,0,0,0,0,0]
for it in ev:
    for i in range(len(it)):
        vals[i] = (vals[i] + it[i])/2
print(vals)
print(max(vals))

print(moy)
valmoy = [0,0,0]
for it in moy:
    for i in range(len(it)):
        valmoy[i] = (valmoy[i] + it[i])/2

ev = []
print("-"*20)
for i in range(100):
    val =[]
```

Pour nous le KFold n'a pas très bien fonctionné, en effet les métriques nous donnait des résultats pires que si nous n'utilisions pas le KFold :

Résultats de base

```
Accuracy : 0.654320987654321
Precision : 0.6911680911680912
Recall : 0.654320987654321
F1 Score: : 0.6407064471879287
```

Résultats avec KFold :

```
Accuracy : 0.475
Precision : 0.3404829545454545
Recall : 0.475
F1 Score: : 0.36924324324324326
```

Pour la phase de test, nous avions de multiples tests :

Voici des exemples :

```
print(model.predict([[35,3566167161116111,35,3566167161116111]])) # 3
print(model.predict([[9,"3566161116111611161116111",35,"9661611161116111611116111"]])) # 4
print(model.predict([[35,"666111111",9,"356611161111"]])) # 1
print(model.predict([[17,"1766171171111",17,"1766171171111"]])) # 3
print(model.predict([[17,"9661611161116111611116111",9,"1766161116111611161116111"]])) # 2
print(model.predict([[17,"1661788161116111",1,"17661788161116111"]])) # 2
```

```
[3]
[4]
[1]
[2]
[2]
[2]
```

Nous avions donc décider que l'IA était donc prête à l'utilisation et que les paramètres qui la constitue sont assez optimisés.

Puis nous avons fait la comparaison avec CLIPS (fichier disponible dans l'archive)

L'ensemble du code utilisé et des étapes se trouvent à la suite de code source python dans le fichier main.py