Intelligent Systems - Assignment 1

Imports

```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from pyfume.Clustering import Clusterer
from pyfume.EstimateAntecendentSet import AntecedentEstimator
from pyfume.EstimateConsequentParameters import ConsequentEstimator
from pyfume.SimpfulModelBuilder import SugenoFISBuilder
from pyfume.Tester import SugenoFISTester
from sklearn.metrics import mean_squared_error,
mean_absolute_percentage_error, explained_variance_score
from numpy import clip, column_stack, argmax
from sklearn.metrics import accuracy_score, recall_score,
precision_score, f1_score, cohen_kappa_score
```

Hair Dryer Dataset

Loading dataset and Creating train-test sets

```
data = pd.read_csv('hairdryer.csv', header = None)
data.columns = ["Voltage","Air_Temperature"]
var_names = []
var_names.append(data.columns[0])
data = data.to_numpy()
scaler = MinMaxScaler()
data = scaler.fit_transform(data)
X = data[:,0:1]
y = data[:,1]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
0.2, random_state=42)
```

Cluster the input-output space

```
cl = Clusterer(x_train = X_train, y_train = y_train, nr_clus=10)
clust_centers, part_matrix, _ = cl.cluster(method='fcm')
```

Estimate membership functions parameters

```
ae = AntecedentEstimator(X_train, part_matrix)
antecedent_params = ae.determineMF()
```

Estimate consequent parameters

```
ce = ConsequentEstimator(X_train, y_train, part_matrix)
conseq_params = ce.suglms()
```

Build first-order Takagi-Sugeno model

```
modbuilder = SugenoFISBuilder(antecedent_params, conseq_params,
var_names, save_simpful_code=False)
model = modbuilder.get_model()
```

```
 * Detected 10 rules / clusters
 * Detected Sugeno model type
```

Get model predictions

```
modtester = SugenoFISTester(model, X_test, var_names)
y_pred = modtester.predict()[0]
```

Compute regression metrics

```
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error: {:.3f}".format(mse))
mape = mean_absolute_percentage_error(y_test, y_pred)
print("Mean Absolute Percentage Error: {:.1f}%".format(mape*100))
exp_var = explained_variance_score(y_test, y_pred)
print("Explained Variance Score: {:.3f}".format(exp_var))
```

```
Mean Squared Error: 0.072
Mean Absolute Percentage Error: 75.4%
Explained Variance Score: 0.008
```

Wisconsin Breast Cancer Original Dataset

Loading dataset and Creating train-test sets

```
data1 = pd.read_csv('wbco.csv')
data1.replace('?', None, inplace = True)
data1.dropna(inplace = True)
var_names1 = ["ClumpThickness", "UniformityOfCellSize",
"UniformityOfCellShape", "MarginalAdhesion",
"SingleEpithelialCellSize", "BlandChromatin", "NormalNucleoli",
"Mitoses"]
data1 = data1.to_numpy()
scaler1 = MinMaxScaler()
data1 = scaler1.fit_transform(data1)
X1 = data1[:,:9]
y1 = data1[:,9]
```

```
X_train1, X_test1, y_train1, y_test1 = train_test_split(X1, y1,
test_size = 0.2, random_state = 42)
```

Cluster the input-output space

```
cl1 = Clusterer(x_train = X_train1, y_train = y_train1, nr_clus=10)
clust_centers1, part_matrix1, _ = cl1.cluster(method='fcm')
```

Estimate membership functions parameters

```
ae1 = AntecedentEstimator(X_train1, part_matrix1)
antecedent_params1 = ae1.determineMF()
```

Estimate consequent parameters

```
ce1 = ConsequentEstimator(X_train1, y_train1, part_matrix1)
conseq_params1 = ce1.suglms()
```

Build first-order Takagi-Sugeno model

```
modbuilder1 = SugenoFISBuilder(antecedent_params1, conseq_params1,
var_names1, save_simpful_code=False)
model1 = modbuilder1.get_model()
```

```
 * Detected 10 rules / clusters
 * Detected Sugeno model type
```

Get model predictions

```
modtester1 = SugenoFISTester(model1, X_test1, var_names1)
y_pred_probs1 = clip(modtester1.predict()[0], 0, 1)
y_pred_probs1 = column_stack((1 - y_pred_probs1, y_pred_probs1))
y_pred1 = argmax(y_pred_probs1,axis=1)
```

Compute classification metrics

```
acc_score = accuracy_score(y_test1, y_pred1)
print("Accuracy: {:.3f}".format(acc_score))
rec_score = recall_score(y_test1, y_pred1)
print("Recall: {:.3f}".format(rec_score))
prec_score = precision_score(y_test1, y_pred1)
print("Precision Score: {:.3f}".format(prec_score))
F1_score = f1_score(y_test1, y_pred1)
print("F1-Score: {:.3f}".format(F1_score))
kappa = cohen_kappa_score(y_test1, y_pred1)
print("Kappa Score: {:.3f}".format(kappa))
```

```
Accuracy: 0.978
Recall: 0.980
Precision Score: 0.961
```

```
F1-Score: 0.970
Kappa Score: 0.953
```

GITHUB

https://github.com/FranciscoCarvalho26/SI