Intelligent Systems – Assignment 1

Alexandre Baptista ist1 100514

This assignment addresses fuzzy modelling of two machine learning tasks: regression and classification. The chosen approach is based on Takagi–Sugeno–Kang (TSK) fuzzy systems, which combine fuzzy rules with local linear models to provide interpretable and flexible predictive models.

Two datasets were considered:

The Diabetes dataset (from scikit-learn) for regression, where the objective is to predict a quantitative measure of disease progression based on 10 baseline medical variables.

The Pima Indians Diabetes dataset (from OpenML) for classification, where the task is to predict the presence of diabetes from 8 clinical features.

The models were developed in Python using fuzzy c-means clustering to identify rule antecedents and least-squares estimation for rule consequents, following the teacher's template. Performance was evaluated using Mean Squared Error (MSE) for regression and Accuracy (ACC) for classification. Additionally, I experimented with alternative consequents (logistic regression per rule) to test whether performance could be improved.

Dataset 1: Diabetes Dataset (Regression)

In [128…

```
import os
import numpy as np
import pandas as pd
from dataclasses import dataclass
from typing import Tuple
import torch
import torch.nn as nn

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import (
    mean_squared_error,
    accuracy_score,
    f1_score,
    roc_auc_score,
)
from sklearn.datasets import load_diabetes, fetch_openml
import skfuzzy as fuzz

# =========================
# CONFIG
# =========================
TASK = "regression"   # "regression" ou "classification"
DATASET = "sklearn_diabetes"   # "sklearn_diabetes" | "pima_openml" | "excel"
EXCEL_PATH = "data.xlsx"  # se DATASET="excel", apontar para o ficheiro
TARGET_COL = "target"     # nome da coluna target para o Excel

N_CLUSTERS = 6            # nº de regras / clusters
M_FCM = 1.6               # fuzzifier
TEST_SIZE = 0.2
RANDOM_STATE = 42
# =========================

# -------- utils ----------
def _to_numpy(x):
    return x.detach().cpu().numpy() if isinstance(x, torch.Tensor) else x

def _weighted_mean_std(Xz: np.ndarray, U: np.ndarray) -> Tuple[np.ndarray, np.ndarray]:
    """
    Estima média (centro) e sigma por regra/feature com pesos U^m (consistentes com FCM).
    Retorna centers (R,D) e sigmas (R,D).
    """
    R, N = U.shape
    D = Xz.shape[1]
    Um = U ** M_FCM
    centers = np.zeros((R, D))
    sigmas = np.zeros((R, D))
    for r in range(R):
        w = Um[r][:, None]  # (N,1)
        mu = (w * Xz).sum(axis=0) / (w.sum(axis=0) + 1e-12)
        centers[r] = mu
```

```python
        # var ponderada
        var = (w * (Xz - mu) ** 2).sum(axis=0) / (w.sum(axis=0) + 1e-12)
        sigmas[r] = np.sqrt(var + 1e-6)  # evitar sigma=0
    return centers, sigmas

def _design_matrix(Xz: np.ndarray, centers: np.ndarray, sigmas: np.ndarray) -> np.ndarray:
    """
    Constroi Phi para TSK (ordem 1): para cada amostra i,
    concatena para cada regra r:  [w_r_normalizada(i), w_r_normalizada(i)*x(i)]
    (i.e., b_r e W_r partilham a mesma ponderação normalizada).
    Resultado: Phi shape (N, R*(1+D))
    """
    R, D = centers.shape
    N = Xz.shape[0]
    # Gaussian MFs por feature
    # w_r(i) = prod_d exp(-0.5 * ((x_id - c_rd)/sigma_rd)^2)
    # Para estabilidade, somar logs:
    exps = []
    for r in range(R):
        z = (Xz - centers[r]) / (sigmas[r] + 1e-12)  # (N,D)
        log_phi = -0.5 * (z ** 2).sum(axis=1)        # (N,)
        exps.append(log_phi)
    log_w = np.stack(exps, axis=1)  # (N,R)
    # normalizar por regra para cada amostra
    # w_norm = softmax(log_w) sem “temperatura”
    maxlog = np.max(log_w, axis=1, keepdims=True)
    w = np.exp(log_w - maxlog)
    w = w / (w.sum(axis=1, keepdims=True) + 1e-12)  # (N,R) normalizado

    # Construir Phi
    Phi_parts = []
    ones = np.ones((N, 1))
    for r in range(R):
        wr = w[:, [r]]  # (N,1)
        Phi_r = np.hstack([wr * ones, wr * Xz])  # (N, 1+D)
        Phi_parts.append(Phi_r)
    Phi = np.hstack(Phi_parts)  # (N, R*(1+D))
    return Phi, w

# ------- Modelo TSK -------
@dataclass
class TSKModel(nn.Module):
    centers: np.ndarray  # (R,D)
    sigmas: np.ndarray   # (R,D)
    D: int
    R: int

    def __post_init__(self):
        super().__init__()
        self.D = self.centers.shape[1]
        self.R = self.centers.shape[0]
        # Parâmetros consequentes (empilhados): para cada regra r: [b_r, w_r1, ..., w_rD]
        # Inicializa zeros; serão aprendidos por LS
        self.theta = nn.Parameter(torch.zeros(self.R, self.D + 1), requires_grad=False)

    def forward(self, Xz: torch.Tensor):
        """
        Xz: (N,D) padronizado
        Retorna:
          y_pred: (N,1)
          w_norm: (N,R) firing strengths normalizados
          Phi:    (N,R*(1+D)) design matrix usada no LS
        """
        X = Xz  # (N,D)
        N = X.shape[0]
        # computar w_norm e Phi em numpy (mais simples) e converter
        Phi_np, w_norm_np = _design_matrix(_to_numpy(X), self.centers, self.sigmas)
        Phi = torch.from_numpy(Phi_np).to(dtype=torch.float32, device=X.device)       # (N, R*(1+D))
        w_norm = torch.from_numpy(w_norm_np).to(dtype=torch.float32, device=X.device) # (N,R)

        # y = sum_r ( w_norm_r * (b_r + w_r^T x) )
        # Podemos obter y via Phi @ vec(theta)
        theta_vec = self.theta.reshape(-1)  # (R*(1+D),)
        y = Phi @ theta_vec  # (N,)
        return y.view(-1, 1), w_norm, Phi

# ------ Least Squares ------
def train_ls(model: TSKModel, Xz: np.ndarray, y: np.ndarray, task: str):
    """
    Ajusta theta por LS:
      theta = (Phi^T Phi)^(-1) Phi^T y
    Para classificação, ajusta LS no espaço do 'logit' (aproximação):
      y_tilde = log(p/(1-p))  com clipping p∈[1e-3, 1-1e-3]
```

```
    """
    device = torch.device("cpu")
    model.to(device)
    Xz_t = torch.from_numpy(Xz.astype(np.float32))
    y_t = torch.from_numpy(y.astype(np.float32)).view(-1, 1)

    # Para classificação: transformar rótulos (0/1) em valores-alvo contínuos via logit
    if task == "classification":
        p = y_t.clamp(1e-3, 1 - 1e-3)  # evita inf
        y_ls = torch.log(p / (1 - p))
    else:
        y_ls = y_t

    # Obter Phi
    with torch.no_grad():
        _, _, Phi = model(Xz_t)  # (N, R*(1+D))

    # Resolver LS: theta = (Phi^T Phi + λI)^(-1) Phi^T y
    lam = 1e-6
    A = Phi.T @ Phi + lam * torch.eye(Phi.shape[1])
    b = Phi.T @ y_ls
    # >>> FIX AQUI: usar torch.linalg.solve <<<
    theta_vec = torch.linalg.solve(A, b)
    theta = theta_vec.view(model.R, model.D + 1)
    with torch.no_grad():
        model.theta.copy_(theta)


# --------- Dados ----------
def load_data():
    if DATASET == "sklearn_diabetes":
        # REGRESSÃO
        ds = load_diabetes()
        X = ds.data.astype(float)
        y = ds.target.astype(float)
        names = list(ds.feature_names)
        return X, y, names

    elif DATASET == "pima_openml":
        # CLASSIFICAÇÃO
        df = fetch_openml(name="diabetes", version=1, as_frame=True)
        X_df = df.data.copy()
        # corrigir zeros impossíveis e imputar
        zero_bad = ["Glucose", "BloodPressure", "SkinThickness", "Insulin", "BMI"]
        for c in zero_bad:
            if c in X_df.columns:
                X_df[c] = X_df[c].replace(0, np.nan)
        X_df = X_df.fillna(X_df.median(numeric_only=True))
        # target string -> binário
        y_ser = df.target.astype(str).str.strip().str.lower()
        y = y_ser.isin(["tested_positive", "positive", "pos", "1", "true", "yes"]).astype(int).to_numpy()
        X = X_df.to_numpy().astype(float)
        names = list(X_df.columns)
        return X, y, names

    elif DATASET == "excel":
        # Lê de Excel (última coluna = target, a não ser que TARGET_COL esteja definido)
        X_df = pd.read_excel(EXCEL_PATH)
        if TARGET_COL in X_df.columns:
            y = X_df[TARGET_COL].to_numpy()
            X_df = X_df.drop(columns=[TARGET_COL])
        else:
            # assume última coluna é o target
            y = X_df.iloc[:, -1].to_numpy()
            X_df = X_df.iloc[:, :-1]
        X = X_df.to_numpy().astype(float)
        names = list(X_df.columns)
        return X, y, names

    else:
        raise ValueError("DATASET inválido. Use 'sklearn_diabetes', 'pima_openml' ou 'excel'.")

# --------- Main -----------
def main():
    X, y, feat_names = load_data()

    # Força coerência com TASK
    if TASK == "regression":
        y = y.astype(float)
    else:
        # garante 0/1
        y = (y > 0).astype(int)
```

```python
    # Escalonamento
    scaler = StandardScaler().fit(X)
    Xz = scaler.transform(X)

    # Split train/test
    Xtr, Xte, ytr, yte = train_test_split(
        Xz, y, test_size=TEST_SIZE, random_state=RANDOM_STATE, stratify=(y if TASK=="classification" else None)
    )

    # FCM sobre treino (em espaço escalonado)
    centers, U, *_ = fuzz.cluster.cmeans(
        data=Xtr.T, c=N_CLUSTERS, m=M_FCM, error=1e-5, maxiter=300, init=None, seed=RANDOM_STATE
    )  # centers: (R,D), U: (R,Ntr)

    # Estimar sigmas ponderados
    centers_w, sigmas_w = _weighted_mean_std(Xtr, U)    # (R,D), (R,D)
    # Usa centers do FCM + sigmas ponderadas
    centers_use = centers
    sigmas_use = sigmas_w

    # Construir modelo TSK
    R, D = centers_use.shape
    model = TSKModel(centers=centers_use, sigmas=sigmas_use, D=D, R=R)

    # Treino por LS (fecho analítico)
    train_ls(model, Xtr, ytr, TASK)

    # Avaliação
    y_pred_tr, _, _ = model(torch.from_numpy(Xtr.astype(np.float32)))
    y_pred_te, _, _ = model(torch.from_numpy(Xte.astype(np.float32)))

    if TASK == "regression":
        yhat_te = _to_numpy(y_pred_te).ravel()
        mse = mean_squared_error(yte, yhat_te)
        print(f"[REG] Test MSE: {mse:.3f}")
    else:
        # para classificação, aplicar sigmoid ao output TSK (logit aproximado)
        yhat_proba_te = 1 / (1 + np.exp(-_to_numpy(y_pred_te).ravel()))
        yhat_te = (yhat_proba_te >= 0.5).astype(int)
        acc = accuracy_score(yte, yhat_te)
        f1 = f1_score(yte, yhat_te)
        try:
            auc = roc_auc_score(yte, yhat_proba_te)
        except Exception:
            auc = float("nan")
        print(f"[CLS] Test Acc: {acc:.3f} | F1: {f1:.3f} | ROC-AUC: {auc:.3f}")

    # Info de regras (centros/sigmas em z-score)
    print("\nRegras (centros/sigmas em espaço padronizado):")
    for r in range(R):
        c_txt = ", ".join([f"{feat_names[d]}≈{centers_use[r,d]:+.2f}σ" for d in range(D)])
        s_txt = ", ".join([f"σ_{feat_names[d]}={sigmas_use[r,d]:.2f}" for d in range(D)])
        print(f"- Regra {r+1}: centro[{c_txt}] | {s_txt}")

if __name__ == "__main__":
    main()
```

```
[REG] Test MSE: 2443.032

Regras (centros/sigmas em espaço padronizado):
- Regra 1: centro[age≈+0.31σ, sex≈+0.44σ, bmi≈+0.49σ, bp≈+0.33σ, s1≈+0.94σ, s2≈+0.97σ, s3≈-0.62σ, s4≈+1.13σ, s5≈
+0.81σ, s6≈+0.62σ] | σ_age=0.85, σ_sex=0.93, σ_bmi=0.86, σ_bp=0.94, σ_s1=0.92, σ_s2=0.98, σ_s3=0.72, σ_s4=0.93,
σ_s5=0.84, σ_s6=0.91
- Regra 2: centro[age≈+0.22σ, sex≈+0.50σ, bmi≈-0.22σ, bp≈-0.06σ, s1≈-0.37σ, s2≈-0.27σ, s3≈-0.03σ, s4≈-0.25σ, s5≈
-0.28σ, s6≈-0.11σ] | σ_age=0.87, σ_sex=0.90, σ_bmi=0.79, σ_bp=0.87, σ_s1=0.78, σ_s2=0.75, σ_s3=0.85, σ_s4=0.72,
σ_s5=0.81, σ_s6=0.81
- Regra 3: centro[age≈-0.92σ, sex≈-0.44σ, bmi≈-0.84σ, bp≈-0.80σ, s1≈-1.00σ, s2≈-0.96σ, s3≈+0.35σ, s4≈-0.87σ, s5≈
-0.90σ, s6≈-0.83σ] | σ_age=0.89, σ_sex=0.86, σ_bmi=0.71, σ_bp=0.72, σ_s1=0.69, σ_s2=0.68, σ_s3=0.78, σ_s4=0.53,
σ_s5=0.64, σ_s6=0.83
- Regra 4: centro[age≈+0.28σ, sex≈-0.27σ, bmi≈+0.54σ, bp≈+0.48σ, s1≈+0.26σ, s2≈+0.18σ, s3≈-0.17σ, s4≈+0.22σ, s5≈
+0.48σ, s6≈+0.46σ] | σ_age=0.83, σ_sex=0.95, σ_bmi=0.94, σ_bp=0.99, σ_s1=0.85, σ_s2=0.86, σ_s3=0.86, σ_s4=0.82,
σ_s5=0.87, σ_s6=0.89
- Regra 5: centro[age≈+0.38σ, sex≈+0.47σ, bmi≈+0.49σ, bp≈+0.49σ, s1≈+0.22σ, s2≈+0.25σ, s3≈-0.55σ, s4≈+0.59σ, s5≈
+0.57σ, s6≈+0.53σ] | σ_age=0.83, σ_sex=0.92, σ_bmi=0.91, σ_bp=0.94, σ_s1=0.86, σ_s2=0.87, σ_s3=0.77, σ_s4=0.84,
σ_s5=0.86, σ_s6=0.90
- Regra 6: centro[age≈-0.03σ, sex≈-0.58σ, bmi≈-0.25σ, bp≈-0.24σ, s1≈-0.03σ, s2≈-0.17σ, s3≈+0.75σ, s4≈-0.60σ, s5≈
-0.42σ, s6≈-0.35σ] | σ_age=0.87, σ_sex=0.77, σ_bmi=0.80, σ_bp=0.85, σ_s1=0.77, σ_s2=0.74, σ_s3=0.98, σ_s4=0.63,
σ_s5=0.73, σ_s6=0.83
```

The regression task was conducted on the Diabetes dataset from scikit-learn, with 442 samples and 10 clinical features. A TSK fuzzy model was constructed using Fuzzy C-Means clustering to define rule antecedents and least-squares estimation for the consequents.

For the regression task, the number of fuzzy rules (N_clusters) and the fuzziness parameter (M_fCM) were chosen through grid search with cross-validation. Different combinations were tested, and the configuration that minimized the mean squared error (MSE) on validation folds was selected. This process led to the use of N_clusters=6 and M_fCM=1.6, which achieved the best generalization performance.

The model achieved a Test MSE of 2443.032. The Mean Squared Error (MSE) represents the average of the squared differences between predicted and true values. Lower MSE values indicate better predictive accuracy, and the obtained result confirms that the fuzzy TSK model can capture relevant patterns in the data. Also, the rules obtained illustrate how the fuzzy system partitions the feature space into interpretable regions, each associated with a local regression model.

Dataset 2: Pima Indians Diabetes Dataset (Classification)

For the classification task, we used the Pima Indians Diabetes dataset from OpenML, which contains 768 samples with 8 clinical features. A TSK fuzzy model was again applied, following the same approach as in the regression case: Fuzzy C-Means clustering for antecedents and least-squares estimation for consequents.

A similar grid search was applied, evaluating combinations of clusters and fuzzyness using cross-validation and ROC-AUC as the selection criterion. Logistic regression was used as the consequent model to better suit the binary classification setting.

Using this method, we obtained the following performance on the test set:

[CLS] Test Acc: 0.734 | F1: 0.631 | ROC-AUC: 0.794

Regras (centros/sigmas em espaço padronizado):

- Regra 1: centro[preg≈+0.01σ, plas≈-0.06σ, pres≈+0.04σ, skin≈-0.08σ, insu≈-0.13σ, mass≈-0.03σ, pedi≈+0.01σ, age≈-0.02σ] | σ_preg=0.91, σ_plas=0.94, σ_pres=0.96, σ_skin=0.96, σ_insu=0.84, σ_mass=0.94, σ_pedi=0.99, σ_age=0.89
- Regra 2: centro[preg≈-0.37σ, plas≈+0.30σ, pres≈+0.12σ, skin≈+0.67σ, insu≈+0.59σ, mass≈+0.49σ, pedi≈+0.34σ, age≈-0.27σ] | σ_preg=0.80, σ_plas=0.92, σ_pres=0.83, σ_skin=0.83, σ_insu=1.02, σ_mass=0.90, σ_pedi=1.05, σ_age=0.77
- Regra 3: centro[preg≈-0.48σ, plas≈-0.56σ, pres≈-0.30σ, skin≈-0.38σ, insu≈-0.34σ, mass≈-0.72σ, pedi≈-0.26σ, age≈-0.61σ] | σ_preg=0.63, σ_plas=0.72, σ_pres=0.85, σ_skin=0.70, σ_insu=0.55, σ_mass=0.83, σ_pedi=0.74, σ_age=0.59
- Regra 4: centro[preg≈+0.76σ, plas≈+0.71σ, pres≈+0.29σ, skin≈+0.49σ, insu≈+0.53σ, mass≈+0.24σ, pedi≈+0.36σ, age≈+0.80σ] | σ_preg=1.00, σ_plas=0.95, σ_pres=0.82, σ_skin=0.88, σ_insu=1.11, σ_mass=0.87, σ_pedi=1.05, σ_age=0.94
- Regra 5: centro[preg≈+0.78σ, plas≈+0.23σ, pres≈+0.26σ, skin≈-0.90σ, insu≈-0.53σ, mass≈-0.09σ, pedi≈-0.32σ, age≈+0.99σ] | σ_preg=0.92, σ_plas=0.89, σ_pres=0.86, σ_skin=0.79, σ_insu=0.59, σ_mass=0.88, σ_pedi=0.80, σ_age=0.98
- Regra 6: centro[preg≈-0.44σ, plas≈-0.34σ, pres≈-0.07σ, skin≈+0.40σ, insu≈-0.01σ, mass≈+0.21σ, pedi≈-0.06σ, age≈-0.48σ] | σ_preg=0.71, σ_plas=0.82, σ_pres=0.81, σ_skin=0.79, σ_insu=0.75, σ_mass=0.84, σ_pedi=0.87, σ_age=0.68

To further explore possible improvements, I experimented with an alternative version of the model in which the consequents are estimated using logistic regression per rule rather than least squares. This adjustment increased accuracy and ROC-AUC, as presented in the following section.

In [131…

```
# classification_pima_tsk.py
# TSK fuzzy classification on Pima Indians dataset via OpenML
# Usage: python classification_pima_tsk.py
# Dependencies: numpy, pandas, scikit-learn, scikit-fuzzy, matplotlib

import os, json, numpy as np, pandas as pd, matplotlib.pyplot as plt
from dataclasses import dataclass
from typing import List
from sklearn.datasets import fetch_openml
from sklearn.model_selection import KFold
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, f1_score, roc_auc_score, roc_curve
import skfuzzy as fuzz

ARTIFACT_DIR = r"C:\Users\alexa\Desktop\Ist100514\si\assignement1\artifacts"
os.makedirs(ARTIFACT_DIR, exist_ok=True)
rng = np.random.default_rng(42)

def fcm_train(X, n_rules, m=2.0, max_iter=300, error=1e-5, seed=42):
    cntr, U, _, _, _, _, _ = fuzz.cluster.cmeans(
        data=X.T, c=n_rules, m=m, error=error, maxiter=max_iter, init=None, seed=seed
    )
    return cntr, U

def fcm_membership_for_new(X, centers, m=2.0, eps=1e-12):
    n_rules = centers.shape[0]; n_samples = X.shape[0]
    d = np.zeros((n_rules, n_samples))
    for r in range(n_rules):
        diff = X - centers[r]
        d[r] = np.linalg.norm(diff, axis=1) + eps
    power = 2.0/(m-1.0)
```

```python
    denom = np.zeros_like(d)
    for r in range(n_rules):
        denom[r] = np.sum((d[r][:,None] / d.T)**power, axis=1)
    return 1.0/denom

@dataclass
class TSKClassifier:
    n_rules: int
    m: float = 2.0
    centers_: np.ndarray = None
    clfs_: List[LogisticRegression] = None
    scaler_: StandardScaler = None
    feature_names_: List[str] = None

    def fit(self, X: np.ndarray, y: np.ndarray, feature_names=None):
        self.feature_names_ = feature_names or [f"x{i}" for i in range(X.shape[1])]
        self.scaler_ = StandardScaler().fit(X)
        Xs = self.scaler_.transform(X)
        centers, U = fcm_train(Xs, self.n_rules, self.m, seed=42)
        self.centers_ = centers
        self.clfs_ = []
        for r in range(self.n_rules):
            w = (U[r]**self.m)
            clf = LogisticRegression(max_iter=400, solver="lbfgs")
            clf.fit(Xs, y, sample_weight=w)
            self.clfs_.append(clf)
        return self

    def predict_proba(self, X: np.ndarray) -> np.ndarray:
        Xs = self.scaler_.transform(X)
        U = fcm_membership_for_new(Xs, self.centers_, self.m)
        w = (U**self.m)
        pr = np.stack([clf.predict_proba(Xs)[:,1] for clf in self.clfs_], axis=0)
        p = np.sum(w * pr, axis=0) / np.sum(w, axis=0)
        return np.vstack([1-p, p]).T

    def predict(self, X: np.ndarray) -> np.ndarray:
        return (self.predict_proba(X)[:,1] >= 0.5).astype(int)

    def pretty_rules(self) -> List[str]:
        rules = []
        for r, c in enumerate(self.centers_):
            center = ", ".join([f"{name}≈{c[i]:.2f}σ" for i, name in enumerate(self.feature_names_)])
            rules.append(f"Rule {r+1}: IF x near center[{center}] THEN output via logistic model (see weights).
        return rules

def grid_search_tsk_clf(X, y, n_rules_grid=(2,3,4,5,6), m_grid=(1.6,2.0,2.4), cv=5, random_state=42):
    kf = KFold(n_splits=cv, shuffle=True, random_state=random_state)
    best = {"auc": -np.inf}
    for R in n_rules_grid:
        for m in m_grid:
            aucs = []
            for tr, va in kf.split(X):
                model = TSKClassifier(n_rules=R, m=m).fit(X[tr], y[tr])
                proba = model.predict_proba(X[va])[:,1]
                try:
                    aucs.append(roc_auc_score(y[va], proba))
                except ValueError:
                    aucs.append(0.5)
            auc = float(np.mean(aucs))
            if auc > best["auc"]:
                best = {"auc": auc, "n_rules": R, "m": m}
    return best

def main():
    # Load Pima from OpenML (id=37, name='diabetes')
    ds = fetch_openml(name="diabetes", version=1, as_frame=True)

    # --- Robust label mapping (strings -> 0/1) ---
    y_series = ds.target.astype(str).str.strip().str.lower()
    # True/positive bucket covers common variants
    y = y_series.isin(["tested_positive", "positive", "pos", "1", "true", "yes"]).astype(int).to_numpy()

    # --- Optional: clean physiologically invalid zeros and impute ---
    X_df = ds.data.copy()
    zero_bad = ["Glucose", "BloodPressure", "SkinThickness", "Insulin", "BMI"]
    for c in zero_bad:
        if c in X_df.columns:
            X_df[c] = X_df[c].replace(0, np.nan)
    X_df = X_df.fillna(X_df.median(numeric_only=True))

    names = list(X_df.columns)
    X = X_df.to_numpy().astype(float)
```

```python
    # Hyperparam search for TSK
    best = grid_search_tsk_clf(X, y)
    print(f"Best CV (ROC-AUC): {best['auc']:.3f} with R={best['n_rules']} and m={best['m']}")

    # 80/20 split (reproducible)
    n = X.shape[0]
    idx = np.random.default_rng(42).permutation(n)
    split = int(0.8*n)
    tr, te = idx[:split], idx[split:]

    model = TSKClassifier(n_rules=best['n_rules'], m=best['m']).fit(X[tr], y[tr], feature_names=names)
    proba = model.predict_proba(X[te])[:,1]
    pred = (proba >= 0.5).astype(int)

    acc = float(accuracy_score(y[te], pred))
    f1  = float(f1_score(y[te], pred))
    try:
        auc = float(roc_auc_score(y[te], proba))
    except ValueError:
        auc = float('nan')
    print(f"Test Acc: {acc:.3f} | F1: {f1:.3f} | ROC-AUC: {auc:.3f}\n")
    print("Rules (standardized space):")
    for s in model.pretty_rules(): print(" -", s)

    # ROC curve
    fpr, tpr, _ = roc_curve(y[te], proba)
    plt.figure()
    plt.plot(fpr, tpr, label=f"TSK (AUC={auc:.3f})")
    plt.plot([0,1],[0,1], linestyle="--")
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.title("ROC Curve — TSK Classifier (Pima)")
    plt.legend()
    plt.tight_layout()
    plt.savefig(os.path.join(ARTIFACT_DIR, "pima_roc_curve.png"))
    plt.close()

    # Save artifacts
    art = {
        "task": "classification_pima",
        "best_cv": best,
        "test_accuracy": acc,
        "test_f1": f1,
        "test_auc": auc,
        "feature_names": names,
        "centers": model.centers_.tolist(),
        "per_rule_logreg": [
            {"coef": clf.coef_.ravel().tolist(), "intercept": float(clf.intercept_[0])}
            for clf in model.clfs_
        ],
    }
    with open(os.path.join(ARTIFACT_DIR, "classification_artifacts.json"), "w") as f:
        json.dump(art, f, indent=2)
    print(f"\nArtifacts saved to: {os.path.abspath(ARTIFACT_DIR)}")

if __name__ == "__main__":
    main()
```

```
Best CV (ROC-AUC): 0.834 with R=6 and m=1.6
Test Acc: 0.799 | F1: 0.674 | ROC-AUC: 0.865

Rules (standardized space):
 - Rule 1: IF x near center[preg≈-0.23σ, plas≈0.65σ, pres≈0.15σ, skin≈0.62σ, insu≈0.91σ, mass≈0.49σ, pedi≈0.25σ,
age≈-0.19σ] THEN output via logistic model (see weights).
 - Rule 2: IF x near center[preg≈0.95σ, plas≈0.53σ, pres≈0.34σ, skin≈0.40σ, insu≈0.25σ, mass≈0.25σ, pedi≈0.26σ,
age≈0.79σ] THEN output via logistic model (see weights).
 - Rule 3: IF x near center[preg≈0.72σ, plas≈0.26σ, pres≈0.30σ, skin≈-0.94σ, insu≈-0.53σ, mass≈-0.11σ, pedi≈-0.2
6σ, age≈1.04σ] THEN output via logistic model (see weights).
 - Rule 4: IF x near center[preg≈-0.19σ, plas≈-0.17σ, pres≈-0.09σ, skin≈-0.16σ, insu≈-0.14σ, mass≈-0.14σ, pedi≈0
.02σ, age≈-0.23σ] THEN output via logistic model (see weights).
 - Rule 5: IF x near center[preg≈-0.47σ, plas≈-0.35σ, pres≈-0.06σ, skin≈0.55σ, insu≈0.03σ, mass≈0.35σ, pedi≈-0.0
6σ, age≈-0.48σ] THEN output via logistic model (see weights).
 - Rule 6: IF x near center[preg≈-0.47σ, plas≈-0.62σ, pres≈-0.35σ, skin≈-0.35σ, insu≈-0.35σ, mass≈-0.70σ, pedi≈-
0.30σ, age≈-0.61σ] THEN output via logistic model (see weights).

Artifacts saved to: C:\Users\alexa\Desktop\Ist100514\si\assignement1\artifacts
```

In addition to the teacher-style TSK model, I implemented a variant where the rule consequents are trained as weighted logistic regressions instead of least squares. Each fuzzy cluster defines a local region, and within that region a logistic regression is fitted using the cluster memberships as sample weights. At inference time, the per-rule probabilities are combined according to the normalized membership degrees.

Comparison with Template derived Model

Template approach (LS + sigmoid): Accuracy ≈ 0.734, ROC-AUC ≈ 0.794.

Logistic TSK: Accuracy ≈ 0.799, ROC-AUC ≈ 0.834.

The improvement arises because least squares regression is not optimal for classification: it minimizes squared error on 0/1 labels and only approximates the probability via a sigmoid. In contrast, logistic regression directly optimizes log-likelihood, leading to better separation between classes and more calibrated probabilities.

Thus, while both methods retain the interpretability of fuzzy rules, the logistic-based TSK achieves higher predictive performance, particularly in terms of ROC-AUC and overall accuracy.

Discussion and Conclusion

This work applied TSK fuzzy systems to regression and classification tasks. On the Diabetes regression dataset, the model achieved a test MSE of 2443, showing that fuzzy rules can capture relevant relationships while remaining interpretable.

For the Pima Indians classification dataset, the template-style model with least-squares consequents reached an accuracy of 0.734. An alternative variant with logistic regression consequents improved performance to accuracy 0.799 and ROC-AUC 0.834, demonstrating that adapting the consequent type to the task can enhance results.

Overall, the experiments confirm that TSK fuzzy systems are effective and interpretable, and that model performance can be further improved with task-appropriate consequents.

All artifacts and versions of code can be found in the github repo: https://github.com/Al3c2/assignment1_fuzzy-/tree/main

In [ ]:

```
```