# ProvaIris

%% Testing with IRIS DATASET

clc;

clear;

close all;

addpath Codes\; **Añadir la capeta “Codes” al path de MATLAB (acceder a funciones o clases de la carpeta)**

%% Initialization of hyperparameters

pol\_deg = 1; **Orden polinomio en el procesamiento de características**

testratio = 30; **% Datos de prueba**

lambda = 0; **Parametro de regularización (evitar overfitting)**

learningRate = 0.1; **Tasa aprendizaje para el optimizador**

hiddenLayers = [3,2,1,2,3]; **Arquitectura red neuronal (número de nodos en cada capa oculta = 5 capas con tamaños 3,2,1,2 y 3 nodos)**

%% INITIALIZATION

% try different feature combination, pairs of features enable the possibility of plotting boundaries

s.fileName = '/Iris.csv' **Nombre del archivo del dataset**

s.polynomialOrder = 1; **Grado polinomio aplicado a las características (control linealidad)**

s.testRatio = 30; **% Datos para la prueba = valor de testratio (definifo antes)**

data = Data(s); **Clase Data**

s.networkParams.hiddenLayers = hiddenLayers; **Configuración capas ocultas**

s.data = data; **Información procesada del dataset**

s.optimizerParams.learningRate = learningRate; **Tasa aprendizaje para el optimizador**

s.costParams.lambda = lambda; **Parametro regularización**

opt = OptimizationProblem(s); **Clase OptimizationProblem**

opt.solve(); **Realizar el entrenamiento del modelo**

opt.plotConfusionMatrix(); **Generar matriz de confusión para evaluar el desempeño**

%opt.plotSurface(); **Representación fronteras de decisión del modelo de forma espacial**

if data.nFeatures == 2 %If you want to be asked for Features change it in "Data" Class

opt.plotBoundary('contour');

end

# Data

classdef Data < handle

properties (Access = public) **Propiedades publicas de la clase**

nFeatures **Numero columnas de los datos de entrenamiento**

nLabels **Numero clases de los datos de entrenamiento**

Xtrain **Conjunto de características de entrenamiento**

Ytrain **Conjunto de etiquetas de entrenamiento**

Xtest **Conjunto de características de prueba**

Ytest **Conjunto de etiquetas de prueba**

Ntest **Numeor de muestras en el conjunto de prueba**

end

properties (Access = private) **Propiedades privadas de la clase**

polynomialOrder **Grado del polinomio para transformar las características**

X **Características normalizadas pera el modelo**

Y **Etiquetas de X**

Data **Conjunto de datos completo cargado desde el archivo**

fileName **Nombre del archivo desde el que se cargan los datos**

testRatio **% de datos utilizados para la prueba**

end

methods (Access = public) **Metodos publicos de la clase Data**

function obj = Data(cParams)

obj.init(cParams) **Inicializar las popiedades de la clase utilizando los paramteros pasados a traves de cParms**

obj.loadData(); **Cargar datos y normalización de las caracteristicas**

obj.splitdata() **Dividir los datos en conjuntos de entrenamiento y prueba según testRatio**

obj.nLabels = size(obj.Ytrain,2); **Asignar numero etiquestas**

obj.nFeatures = size(obj.Xtrain,2); **Asignar caracteristicas**

end

function plotdata(self,i,j)  **Este método genera un gráfico de dispersión para dos características seleccionadas (por ejemplo, las columnas i y j de Xtrain) y colorea los puntos según las etiquetas de Ytrain.**

gscatter(self.Xtrain(:,i),self.Xtrain(:,j),self.Ytrain,'rgbcmyk','\*')

xlabel(['X',num2str(i)]);

ylabel(['X',num2str(j)]);

end

function plotCorrRow(obj,k) **Este método genera una fila de gráficos correlacionados entre las características del conjunto de datos. Si la característica es la misma que la seleccionada (k), se crea un histograma de esa característica, de lo contrario, se genera un gráfico de dispersión entre las dos características correspondientes**

x = obj.data(:,1:end-1);

nf = size(x,2);

for i = 1:nf

nexttile((i-1)\*nf+k)

if i == k

histogram(x(:,i))

else

gscatter(obj.data(:,k),obj.data(:,i),obj.Y,'rgbcmyk','\*')

end

end

end

function plotCorrMatrix(obj) **Este método genera una matriz de correlación entre todas las características del conjunto de datos. Para cada par de características, se muestra un gráfico de dispersión o histograma.**

x = obj.data(:,1:end-1);

nf = size(x,2);

figure

t = tiledlayout(nf,nf,'TileSpacing','Compact');

title(t,'Features correlation matrix');

for i = 1:nf

obj.plotCorrRow(i);

end

end

function updateHyperparameter(obj,h) **Este método permite actualizar los hiperparámetros de la clase, como el porcentaje de datos para prueba (testRatio) o el grado del polinomio (polyGrade). Dependiendo del tipo de hiperparámetro que se actualice, el método llama a diferentes funciones, como splitdata() o buildModel().**

switch h.type

case 'testRatio'

obj.testRatio = h.value;

obj.splitdata()

case 'polyGrade'

obj.polynomialOrder = h.value;

obj.buildModel(obj.X,obj.polynomialOrder);

end

end

end

methods (Access = private) **Métodos utilizados internamente dentro de la clase y no son accesibles desde fuera**

function init(obj,cParams)

obj.fileName = cParams.fileName;

obj.testRatio = cParams.testRatio;

obj.polynomialOrder = cParams.polynomialOrder;

end

function loadData(obj) **Este método carga el conjunto de datos desde un archivo (definido por fileName). Luego permite al usuario seleccionar las características que se utilizarán en el modelo, normaliza esas características entre 0 y 1, y las guarda en X. Las etiquetas (Y) se codifican en formato one-hot encoding.**

f = fullfile('C:\Users\desir\Desktop\Machine\_Learning\_Desi\',obj.fileName);

obj.data = load(f);

fprintf('Features to be used (1:%d):',(size(obj.data,2)-1))

feat = input(' ');

x = obj.data(:, feat);

% IDENTIFIER

% ydata = obj.data(:, end);

% y = zeros(length(ydata),max(ydata));

ydata = obj.data(:, feat);

y = zeros(length(ydata),width(ydata));

u = unique(ydata);

for i=1:length(ydata)

for j = 1:length(u)

if ydata(i) == u(j)

y(i,j) = 1;

end

end

end

obj.X = (x-min(x,[],1))./(max(x,[],1)-min(x,[],1)+10^(-10));

% obj.Y = y;

obj.Y = obj.X;

end

function Xful = buildModel(obj) **Este método crea un modelo polinómico a partir de las características X (usando el grado del polinomio especificado por polynomialOrder). Transforma las características en una combinación polinómica de las características originales.**

x = obj.X;

d = obj.polynomialOrder;

x1 = x(:,1);

x2 = x(:,2);

cont = 1;

for g = 1:d

for a = 0:g

Xful(:,cont) = x2.^(a).\*x1.^(g-a);

cont = cont+1;

end

end

obj.X = Xful;

end

function splitdata(obj) **Este método divide los datos en conjuntos de entrenamiento y prueba de acuerdo al porcentaje testRatio definido. La división es aleatoria (randperm).**

nD = size(obj.data,1);

TP = obj.testRatio;

r = randperm(nD);

ntest = round(TP/100\*nD);

ntrain = nD - ntest;

obj.Xtrain = obj.X(r(1:ntrain),:);

obj.Xtest = obj.X(r((ntrain + 1):end),:);

obj.Ytrain = obj.Y(r(1:ntrain),:);

obj.Ytest = obj.Y(r((ntrain + 1):end),:);

obj.Ntest = ntest;

end

end

end