Department of Computer Science
Technical University of Cluj-Napoca

Intelligent Systems

Laboratory activity 2016-2017

Adrian Groza, Anca Marginean and Radu Razvan Slavescu
Tool:

Name: Dreghiciu Anca
Group: 30235
Email: anca_dreghiciu_1997@yahoo.com

Assoc. Prof. dr. eng. Adrian Groza
Adrian.Groza@cs.utcluj.ro

Contents

Chapter 1

Descrierea proiectului (W_5)

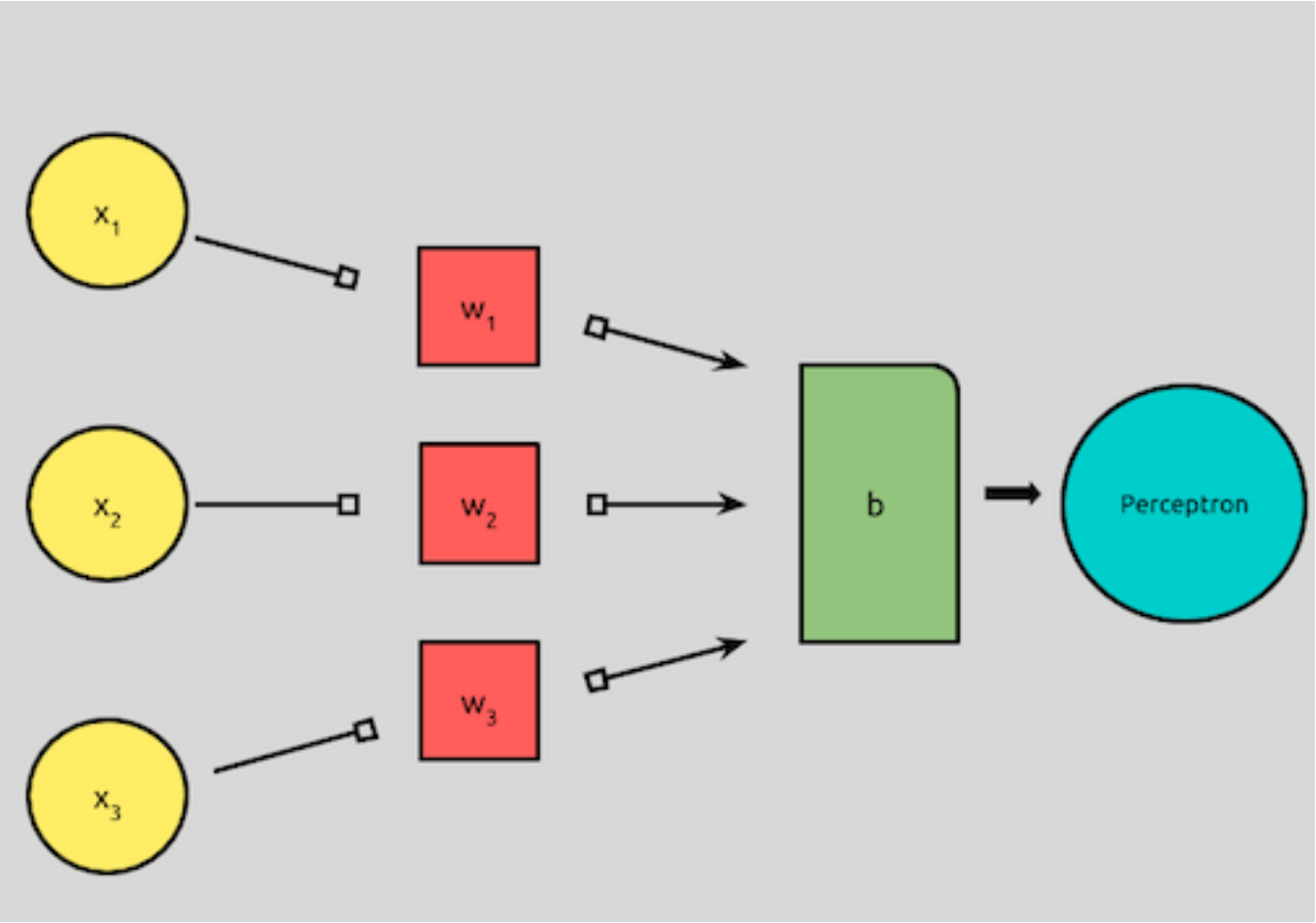
Pentru acest proiect am pus in aplicare diferite tipuri de probleme din machine learning. Avand mai multe categorii, am parcurs pentru fiecare cateva exemple. Aceste categorii de probleme sunt: probleme bazate pe invatare supervizata, invatare nesupervizata si reinforcement learning.
Proiectul se bazeaza pe dezvoltarea capacitatii de a intelege si a putea prelucra date, sub diferite forme, si de a alege modalitatea cea mai eficienta de a rezolva anumite probleme.

1.1 Descrierea MLP

Multi Layer Perceptron utilizeaza tehnica invatarii supervizate numite si backpropagation. Acesta are o functie de activare a tuturor neuronilor. Acesta nu este format dintr-un perceptron cu mai multe layere, ci este mai degraba mai multi perceptroni organizati in layere.

1.2 Descrierea Problemei Perceptron

Pentru o prima abordare a retelelor neurale, am implementat un Perceptron. Un perceptron este un algoritm ce implementeaza o functie liniara. Acesta poate avea oricate "inputuri", dar produce o singura data de iesire, care se mai numeste si activator. Fiecare input are o greutate/prioritate. Pentru a activa perceptronul se va face suma greutatiilor si se va verifica daca este peste un prag.

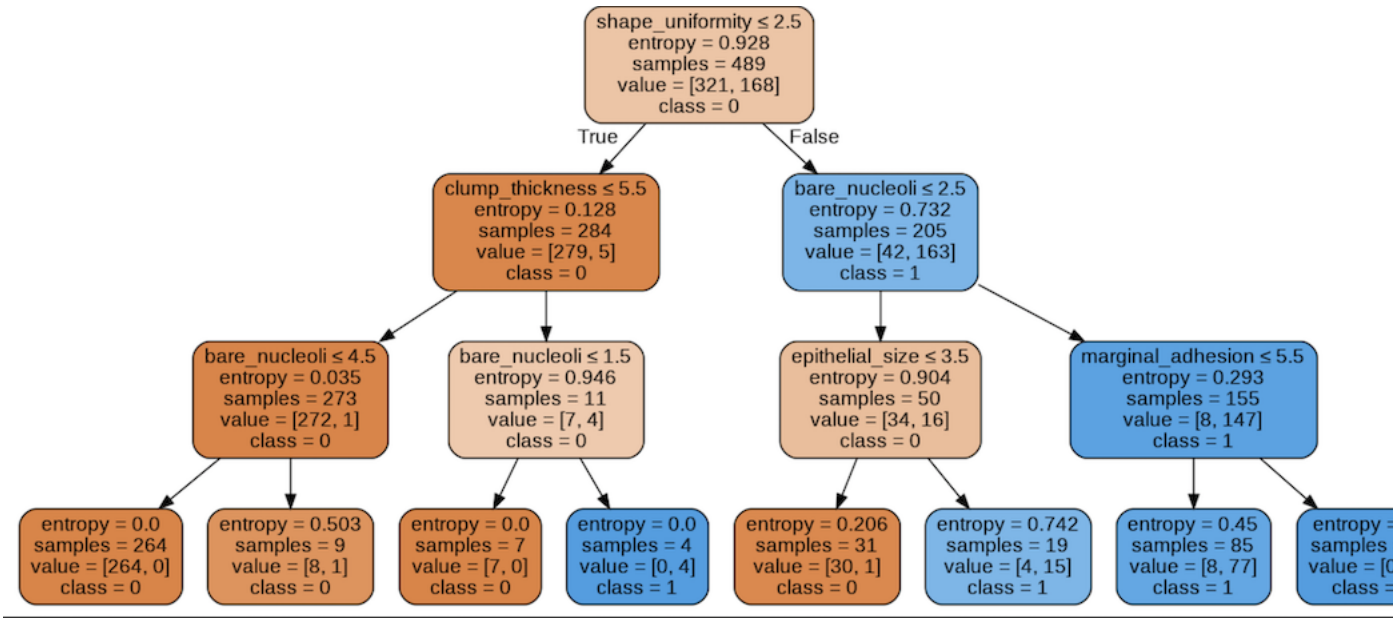


1.3 Descrierea Retelelor Neuronale

O retea neuronala este o combinatie de mai multi neuroni conectati intre ei. O retea poate avea mai multe nivele, unul de input si unul de output, ce sunt necesare, iar intre acestea se mai pot construii mai multe nivele numite nivele ascunse. Aceasta se poate invata din datele pe care le primeste, astfel se pot calcula pierderile pe care le are aceasta retea, adica cat de bine se descurca reteaua. Prin invatare se incearca obtinerea unei pierderi cat mai mici. Noi am folosit MSE (Mean Squared Error) care este diferenta dintre data asteptata si data prezisa la patrat. Se poate itera de mai multe ori prin date. Aceste iteratii complete se numesc epoci. In cazul acestui context intra si problemele de regresie liniara si predictia diabetului.

1.4 Descrierea Arborilor Decizionali

Un arbore decizional este sub forma unui arbore flow-chart, unde fiecare nod reprezinta un feature si o ramura reprezinta o decizie, iar frunzele reprezinta rezultatul tuturor deciziilor. Prin aceasta se poate realiza invatarea mai rapid decat printr-o retea neuronala. Acest algoritm selecteaza datele relevante din setul de date, si compara datele. Selectarea datelor se poate face dupa Information Gain, Gain Ratio, si Gini Index.



1.5 Descrierea K-Means

Algoritmul K-Means Clustering este un algoritm de invatare nesupervizata, acesta incearca gasirea de similitudin intre date, nu are Output ci doar input. Clusterizarea este gruparea unui set de obiecte in diferite clustere, care au mai multe similitudini cu obiectele din acelasi cluster decat cu cele din alt cluster. Similaritatea este o metrica ce refleclta puterea unei relatii intre 2 obiecte. In principal este un algoritm bazat pe clusterizarea centroizilior, functionand iterativ, in care similaritatea este data de cat de aproape este data cautata, de centroidul unui cluster.

1.6 KNN

K Nearest Neighbor este un algoritm de clasificare, non-parametric. Strictura modelului este determinata de datele sale. Algoritmul ia un numar k pentru un punct si cauta cele mai aproape puncte faza de acel punct initial, iar k este numarul vecinilor. Daca k este un numar mic, va avea un zgomot mare ce se va reflecta in rezultat, iar un k mai mare va face ca costul calculelor sa creasca.

1.7 SVM

Suport Vector Machine este un algoritm de clasificare, de invatare supervizata. El separa datele folosind un plan cu cea mai larga margine, astfel el imparte datele in cel putin 2 categorii. Vectorii suport sunt cei mai apropiati de plan, acestia definesc linia, pentru a calcula marginile planului. Planul este un spatiu de decizie care imparte datele in clasele sale.

Appendix A

Your original code

A.1 Mini Batch Breast Cancer

matplotlib este folosit pentru a realiza grafice import matplotlib.pyplot as plt plt.style.use("ggplot") df = pd.read_csv("/content/breastCancer.csv") prof=ProfileReport(df) prof.to_file(output_file='/content/

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

X_train = sc.fit_transform(X_train) X_test = sc.transform(X_test) class Dataset(Dataset): """ Diabetes dataset.""" // Initialize your data, download, etc. def __init__(self, x, y): //Citim setul de date self.len = len(x)

self.x=torch.tensor(x).float() self.y=torch.tensor(y.values).long()

def __getitem__(self, index): return self.x[index], self.y[index]

def __len__(self): return self.len

validationLoader=DataLoader(dataset=validationDataset, batch_size=32, shuffle=True, num_workers=

class BreastCancerNN(nn.Module): def __init__(self): super(BreastCancerNN, self).__init__()

Sequential oferă o alternativă mai estetică a codului Rețeaua noastră are 2 neuroni pentru output. Unul va prezice probabilitatea pentru cazul afirmativ al bolii, iar celălalt va prezice probabilitatea cazului negativ al bolii. self.sequential= nn.Sequential(nn.Linear(13,100), nn.ReLU(), nn.Linear(100, 60), nn.ReLU(), nn.Linear(60, 2))

def forward(self, x): return self.sequential(x)

optimizer = optim.SGD(net.parameters(), lr=0.01) // CrossEntropyLoss este folosit adeseori in problemele de clasificare //Acesta este compus din functia SoftMax și NLLLoss // Softmax - Mapează elementele din Tensor in intervalul [0,1] și face ca suma lor să fie 1. O functie foarte utilă atunci cand vrem sa calculam probabilitati dintr-un Tensor. // NLLLoss - negative log likelihood loss, functie folosită adeseori in problemele de clasificare criterion = nn.CrossEntropyLoss() scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, patience=5, verbose=True)

A.2 Perceptron

training_inputs = [] training_inputs.append(np.array([1, 1])) training_inputs.append(np.array([1, 0])) training_inputs.append(np.array([0, 1])) training_inputs.append(np.array([0, 0]))

labels = np.array([0, 1, 1, 0])

perceptron = Perceptron(2) perceptron.train(training_inputs, labels)

inputs = np.array([1, 1]) print(perceptron.predict(inputs))

inputs = np.array([0, 1]) print(perceptron.predict(inputs))

A.3 Retele Neuronale

Pentru retele neuronale am schimbat functia sigmoid cu cea Relu si functia sa derivativa. Apoi am adaugat inca un nivel ascuns in retea si avand astfel 5 neuroni ascunsi.

def relu(x): ReLu function f(x) = 0 if x ¡ 0, x otherwise var = x if x ¡ 0: var = 0 return var def deriv_relu(x): fx = relu(x) return fx * (1 - fx)

Functia de antrenare: for epoch in range(epochs): for x, y_true in zip(data, all_y_trues): — Do a feedforward (we'll need these values later) sum_h1 = self.w1 * x[0] + self.w2 * x[1] + self.b1 h1 = relu(sum_h1)

sum_h2 = self.w3 * x[0] + self.w4 * x[1] + self.b2 h2 = relu(sum_h2)

sum_h3 = self.w7 * x[0] + self.w8 * x[1] +self.b4 h3 = relu(sum_h3)

sum_h4 = self.w5 * h1 + self.w6 * h2 + self.w9 * h3 + self.b5 h4 = relu(sum_h4)

sum_h5 = self.w10 * h1 + self.w11 * h2 + self.w12 * h3 + self.b6 h5 = relu(sum_h5)

sum_o1 = self.w13 * h4 + self.w14 * h5 o1 = relu(sum_o1) y_pred = o1

crearea a 2 neuroni:

//Neuron h4 d_h4_d_w5 = deriv_relu(sum_h1) * deriv_relu(sum_h4) d_h4_d_w6 = deriv_relu(sum_h2) * deriv_relu(sum_h4) d_h4_d_w9 = deriv_relu(sum_h3) * deriv_relu(sum_h4) d_h4_d_b5 = deriv_relu(sum_h4)

//Neuron h5 d_h5_d_w10 = deriv_relu(sum_h1) * deriv_relu(sum_h5) d_h5_d_w11 = deriv_relu(sum_h2) * deriv_relu(sum_h5) d_h5_d_w12 = deriv_relu(sum_h3) * deriv_relu(sum_h5) d_h5_d_b6 = deriv_relu(sum_h5)

updatarea datelor din neuroni:

//Neuron h4 self.w5 -= learn_rate * d_L_d_ypred * d_ypred_d_h4 * d_h4_d_w5 self.w6 -= learn_rate * d_L_d_ypred * d_ypred_d_h4 * d_h4_d_w6 self.w9 -= learn_rate * d_L_d_ypred * d_ypred_d_h4 * d_h4_d_w9 self.b5 -= learn_rate * d_L_d_ypred * d_ypred_d_h4 * d_h4_d_b5

//Neuron h5 self.w10 -= learn_rate * d_L_d_ypred * d_ypred_d_h5 * d_h5_d_w10 self.w11 -= learn_rate * d_L_d_ypred * d_ypred_d_h5 * d_h5_d_w11 self.w12 -= learn_rate * d_L_d_ypred * d_ypred_d_h5 * d_h5_d_w12 self.b6 -= learn_rate * d_L_d_ypred * d_ypred_d_h5 * d_h5_d_b6

A.4 Decision Tree

Pentru aceasta functie am schimbat setul de date cu cel pentru Cancer Mamar.

col_names = ['id','clump_thickness', 'size_uniformity', 'shape_uniformity', 'marginal_adhesion', 'epithelial_size', 'bare_nucleoli', 'bland_chromatin', 'normal_nucleoli', 'mitoses','label'] //load dataset pima = pd.read_csv("/content/breastCancer.csv")

pima.columns=col_names
del pima['id']
pima.head()

//split dataset in features and target variable feature_cols = ['clump_thickness', 'shape_uniformity', 'marginal_adhesion', 'epithelial_size','bare_nucleoli','bland_chromatin','normal_nucleoli'] X = pima[feature_c // Features y = pima.label // Target variable

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1) // 70

// Create Decision Tree classifer object clf = DecisionTreeClassifier() // Train Decision Tree Classifer clf = clf.fit(X_train,y_train)

A.5 KNN

weather=['Sunny','Sunny','Overcast','Rainy','Rainy','Rainy','Overcast','Sunny','Sunny', 'Rainy','Sunny','O
// Second Feature temp=['Hot','Hot','Hot','Mild','Cool','Cool','Cool','Mild','Cool','Mild','Mild','Mild','Ho
// Label or target variable play=['No','No','Yes','Yes','Yes','No','Yes','No','Yes','Yes','Yes','Yes','Yes','N
// Import LabelEncoder from sklearn import preprocessing //creating labelEncoder le = preprocessing.LabelEncoder() // Converting string labels into numbers. weather_encoded=le.fit_transform(print(weather_encoded)
// converting string labels into numbers temp_encoded=le.fit_transform(temp) label=le.fit_transform(pl
//combinig weather and temp into single listof tuples features=list(zip(weather_encoded,temp_encoded)
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier(n_neighbors=3)
// Train the model using the training sets model.fit(features,label)
//Predict Output predicted= model.predict([[0,2]]) // 0:Overcast, 2:Mild print(predicted)
// Split dataset into training set and test set X_train, X_test, y_train, y_test = train_test_split(wine.data, wine.target, test_size=0.3) // 70
//Import knearest neighbors Classifier model from sklearn.neighbors import KNeighborsClassifier
//Create KNN Classifier knn = KNeighborsClassifier(n_neighbors=5)
//Train the model using the training sets knn.fit(X_train, y_train)
//Predict the response for test dataset y_pred = knn.predict(X_test)
//Import scikit-learn metrics module for accuracy calculation from sklearn import metrics // Model Accuracy, how often is the classifier correct? print("Accuracy:",metrics.accuracy_score(y_test, y_pred))

Bibliography