Open in Colab

Practical Machine Learning

Masters in Green Data Science, ISA/ULisboa, 2022-2023

Instructor: Manuel Campagnolo mlc@isa.ulisboa.pt

Assignment due March 24th, 2023

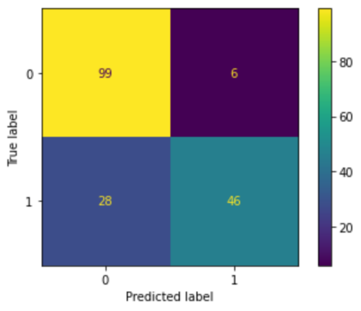
Estimated time to complete the assignment: 2-3 hours.

The script below implements the perceptron model for the Titanic tabular data set as discussed in class.

The two parts of the assignment are:

1. Try changing hyperparameters like the batch size, number of epochs, learning rate, or pre-processing of the numerical data, to try to get the least possible error rate over the validation set. For instance, one possible result with the percepton model is the following, with some set of meta-parameters:

```
coefficients GD: tensor([ 0.4393, -0.3373, -2.8524,  0.5435,  3.2030, -5.3135,  5.1787,  1.9074,
         1.7415, -3.2398,  1.4545,  0.6536, -1.1690])
```



2. Adapt the script below to implement a multiple layer *feed-forward neural network*, as described at the end of ML_overview_with_examples.ipynb. In order to do this easily, use and adapt if needed the functions defined in the last part of the notebook Lesson5_edited_for_colab_linear_model_and_neural_net_from_scratch.ip
It is recommend to use a GPU if there are several hidden layers or the number of epochs is large.

Then, do as in part 1 of the assigment and try changing the architecture of the network as well as the other hyperparameters to obtain a model that performs well over the validation data set. You can describe the architecture by the number of neurons in each hiden layer (from the first to the last).

You should write a short report (one single `pdf` file for both parts), where you indicate, for part 1 and part 2, the hyperparameters and the confusion matrix (by row: true negatives, false positives, false negatives, true positives) as in the following example. You only need to report three or four results for each part, enough to illustrate some pattern that indicate how you can improve your results. Sort the rows in each table by overall performance.

You can add some short comments for each part.

If you wish, you can use a data set other than the Titanic data set. In that case, please add a brief description of the variables and labels.

If you can, **please print** a hard copy of your report and bring it to the class. If you're not able to print, then you may send the report by email (subject: assignment PML).

(This is an example of a report: you should choose your own meta-parameters)

Name: ______________________

Date: _____________________________

PART 1 (perceptron)

Batch size	Learning rate	Epochs	Pre-processing	TN, FP, FN, TP	Error_rate
20	0.1	20	Added 1s + standardized	102, 3, 66, 8	0.3855
10	0.1	20	Added 1s + standardized	105, 0, 70, 4	...
...					

Comments: ____

PART 2 (feed-forward NN)

NN architecture	Batch size	Learning rate	Epochs	Pre-processing	TN, FP, FN, TP	Error_rate
10,5,10	20	0.1	300	standardized	88,17,21,53	0.212
10,10	20	0.1	20	standardized	102, 3, 66, 8	...
...						

Comments: ____

Code to adapt and execute (please *do not* include the code in your report)

In [ ]:
```
import os
from pathlib import Path
import matplotlib.pyplot as plt
import torch
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, ConfusionMatrixDis
torch.manual_seed(42)

B=20 # batch size
lr = 0.1 # learning rate
iter=20 # number epochs

############################################## Reading Titanic num
var_names=['Age', 'SibSp', 'Parch', 'LogFare', 'Sex_male', 'Sex_
path=Path('/content/drive/MyDrive/AAA/Lesson_5/titanic_data') #
X,y=torch.load(path/'titanic_tensor_data_set.ts') # these values
y=y[:,None] # to turn it into a column vector

##################################### Create train and validatic
X_train, X_valid, y_train, y_valid = train_test_split(X, y, test

############################################### Ordinary least sq
# variables to keep to avoid linear dependencies
var_keep=['Age', 'SibSp', 'Parch', 'LogFare', 'Sex_male',  'Pcla
keep=np.isin(var_names,var_keep) # boolean list
#
from sklearn.linear_model import LinearRegression
reg = LinearRegression().fit(X_train[:,keep], y_train)
print('coefficients MLR:',reg.intercept_,reg.coef_)
y_pred=reg.predict(X_valid[:,keep])
disp = ConfusionMatrixDisplay(confusion_matrix(y_valid,(y_pred>0
disp.plot()
plt.show()

####################################################### Gradient
# if you want to standardize X and include an additional additiv
if False:
  means = X.mean(dim=1, keepdim=True)
  stds = X.std(dim=1, keepdim=True)
  X=normalized_data = (X - means) / stds
  # add column
  ones=torch.ones(X.shape[0]).reshape(X.shape[0],1)
  X=torch.cat((ones,X),1)
  X_train, X_valid, y_train, y_valid = train_test_split(X, y, te

# initial weights
def init_coeffs(n_coeff): return (torch.rand(n_coeff,1)-0.5).rec

# defining the function for prediction: the output is a vector c
def calc_preds(coeffs,X): return  torch.sigmoid(X@coeffs) # usir

# Computing MSE loss for one batch of exemples: the output is a
def calc_loss_from_labels(y_pred, y): return torch.mean((y_pred
```

```python
# update coeffs
def update_coeffs(coeffs, lr):
    coeffs.sub_(coeffs.grad * lr)
    # zerofy gradients (because they add up)
    coeffs.grad.zero_()

# compute initial weights as a column matrix
n_coeff = X_train.shape[1] # number of columns of X, or X_train,
coeffs = init_coeffs(n_coeff)

# create lists to store losses for each epoch
training_losses=[]; validation_losses=[]

# epochs
for i in range(iter):
  # calculating loss as in the beginning of an epoch and storing
    y_pred = calc_preds(coeffs,X_train)
    training_losses.append(calc_loss_from_labels(y_pred, y_train
    y_pred = calc_preds(coeffs,X_valid)
    validation_losses.append(calc_loss_from_labels(y_pred, y_val
    # mini-batch gradient descent: weight are updated after each
    for idx_start in np.arange(0,X_train.shape[0],B):
        # create batch
        batch_X=X_train[idx_start:(idx_start+B),:]
        batch_y=y_train[idx_start:(idx_start+B):]
        # making a prediction in forward pass
        y_pred = calc_preds(coeffs,batch_X)
        # calculating the loss between predicted and actual valu
        loss = calc_loss_from_labels(y_pred, batch_y)
        # compute gradient
        loss.backward()
        with torch.no_grad():
            # update coeffs
            update_coeffs(coeffs, lr)

# predictions and confusion matrix
print('coefficients GD:',torch.flatten(coeffs.requires_grad_(Fal
y_pred=calc_preds(coeffs,X_valid)
disp = ConfusionMatrixDisplay(confusion_matrix(y_valid,(y_pred>0
disp.plot()
plt.show()

# plot losses along epochs
plt.plot(training_losses, '-g',  validation_losses, '-r')
plt.gca().legend(('train','validation'))
plt.ylim(0, 1)
plt.xlabel('epoch')
plt.ylabel('loss (MSE)')
#plt.title("Train (green) and validation (red) losses")
plt.show()
```