Relazione Assignment 2

Gianmaria Balducci

19 November 2020

Data Preprocessing

Goal: The assignment consists in the prediction of default payments using a neural network. The dataset contains information on default payments, demographic factors, credit data, history of payment, and bill statements of credit card clients in Taiwan from April 2005 to September 2005.
The dataset provided is already divided in trainset, testset and the target. Train and test are complete, there is no *null* value in any variable, and there aren't duplicate rows. There is an high correlation between BILL_AMTX variables that are Amount of bill statement from September, 2005 to April, 2005 (NT dollar) with X from 1 to 6. Howerver this correlation does not affect the results so i've decided to keep this variables for the training.
Types of the variable's dataset are all float/integer, there isn't any categorical variable, therefore it no need to factorize any variable. Variable called *ID* is probably an error in phase of reading the csv file, it's a kind of index and it is deletd from the train, test and the target. In the *EDUCATION* feature there are 2 value (*5 and 6* that are related to the same type of education :*unknown* , so i've decided to change all the values equal to 6 to values equal to 5, because are the same status. Target variable *default.payment.next.month'* is boolean: default payment (1=yes, 0=no), so the task is a binary classification. Target variable is unbalanced: the values equals to 0 are 18677 and those equals to 1 are 5323. Indeed the performance of my model on the classification of the 0 values are very good while the performance over the classification of the value 1 is too poor. For this reason i have implemented KMeansSMOTE to oversampling the trainset on the minority class. This apply a KMeans clustering before to over-sample using SMOTE. For the train and test preprocessing i used the *StandardScaler*

Model

I split 20% of the trainset and the target in order to create a validation set.
The structure of the model is:

```
model = Sequential()
model.add(Dense(256, input_shape=(dims,), activation = "relu", kernel_initializer=initializer))
model.add(Dense(128, activation = "relu", kernel_initializer=initializer))
model.add(Dense(32, activation = "relu", kernel_initializer=initializer))
model.add(Dense(16, activation = "relu", kernel_initializer=initializer))
model.add(Dense(1, activation = "sigmoid", kernel_initializer=initializer))


model.compile(optimizer=SGD(lr=0.001), loss='binary_crossentropy',metrics=['accuracy', metrics.Precision(name='precision'), metrics.Recall(name='recall')]
```

Figura 1: Neural Network

It has 4 hidden layers consisting of 256, 128, 32 and 16 units, the input layer contain units as many as the labels of the trainset and the output layer contain only one units as our target. I've tried to increase the number of the hidden layers but the result was worse. I've tried also to increase or decrease the number of units for each layer but this change does not affect affect the reslults.
Activaction function: I choose *ReLU* function that is good for this task and in general is a good choice for the hidden layers.
Optimizer: I choose *SGD* because it reaches better performance than *adam*.
Loss: I've tried different loss function such as *mae*, *mse*... but the *binary_ cross entropy*

is optimized for binary classification task and it reached better perfomance so i choose this one.
Metrics: in this model i keep track of *accuracy* of the model, *precision* , and *recall* which give an estimate of how much the model is a good classifier.
Output function: Our target is boolean [0,1], so i try *softmax*, that give me very poor performance and *sigmoid* that is suitable for this training.

```
n_epochs =90
network_history = model.fit(X_train, y_train, batch_size=128,
                            epochs=n_epochs, verbose=2, validation_data=(X_validation, y_validation))
```

Figura 2: Epochs and batchsize

I tried to balance the number of epochs with the batchsize and i see that over 90 epochs the accuracy of the model doesn't increase and the *loss function* doesn't decrease too much. For the batch i've tried different size (64,128,256,512) in order to exploit the GPU and better results with this number of epochs are reached by the size of *128*. This are the results of the model whitout regularization:
Accuracy = 0.83
Precision = 0.81
Recall = 0.82
F1-score = 0.83

Regularization

As it was done in practicals, i tried different types of regularization : **L1**, **L2**, **Dropout**. In L1 and L2 there is need to set lambda parameter that wieghts the contribute of regularization. With *L1* and *L2* i set lambda parameter equal to *0.01*. Increasing this parameter to *0.1*, accuracy of the model doesn't increase too much and after some epochs it decrease. So i keep the lambda parameter equal to 0.01.
I have analyzed the changes of the weights by setting a seed to compare the penalization on the model without regularization with the penalization on the others three models with *L1* and *L2* regularization and with dropout. The results indicate that **L1** regularization lead the weights to zero more faster. However all the weights tend to zero. *Dropout* in term of the leading weights to zero is the slower.
Moreover **Loss Function** in both cases of *L1* and *L2* is bigger than the others two models but deacrease faster.
When the model is trained with Dropout i choose to dropout in all the hidden layers 20% of the units. Training results with dropout, are very similar to the model trained whitout regularization, but distance beetween validation loss and training loss is larger as between accuracies of train and validation.

Results

In order to see and understand which model obtain the best result on the training and validation set i've plotted the results and the classification report. Taking in to account the considerations above my best model is the one to which **L2 Regularization** has been applied. With this kind of regularization the weights tend to zero faster than the model without regularization, while the accuracy and F1-score in both value are slightly lower. In the model regularized by *L2* the weights tend to zero faster than the model with *Dropout* but slower than the model with *L1* regularization, However, accuracy of the L2 model is higher than the model which is applied *L1*. So, to predict the testset data i used *model_l2* in the prediction function.

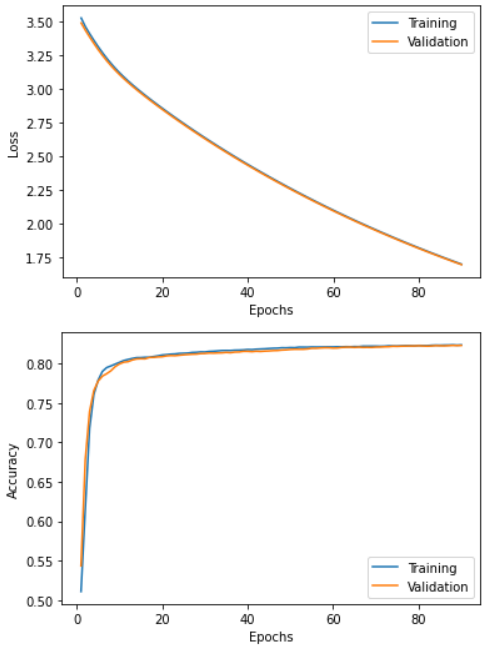

Figura 3: Training and Validation Loss and Accuracy

	precision	recall	f1-score	support
0	0.82	0.82	0.82	3722
1	0.82	0.82	0.82	3750
accuracy			0.82	7472
macro avg	0.82	0.82	0.82	7472
weighted avg	0.82	0.82	0.82	7472

Figura 4: Classification Report