Proietti

Stefano Rosini

December 2023

Contents

1 Introduction

In this report we're gonna see the implementation of the second option given in the `homework12024.pdf`. The project goal was implementing a machine learning model by giving to it as input a dataset that contains N symbolic variables one-hot encoded into M binary variables. The same machine learning problem is then trained with a dataset that had missing variables.

2 Datasets

The datasets used to train our MLP were `LetterRecognition.data` and `car.data`. The latter was taken from the UCI Machine Learning Repository and consists of a series of columns and rows that describe cars. Here we'll show the list of columns:

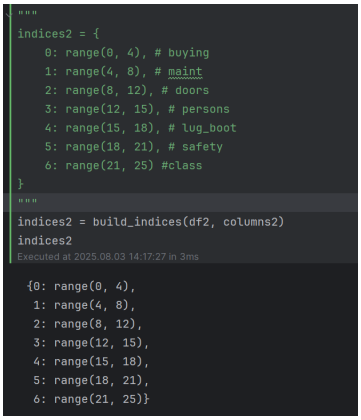
- buying: Price at the selling;
- maint: Maintenance costs;
- doors: Number of doors;
- persons: Number of people that can enter in the car;
- lug_boot: Capacity of the trunk;
- safety: How is the safety of the car;
- class: Condition of the car;

3 Preprocessing Steps

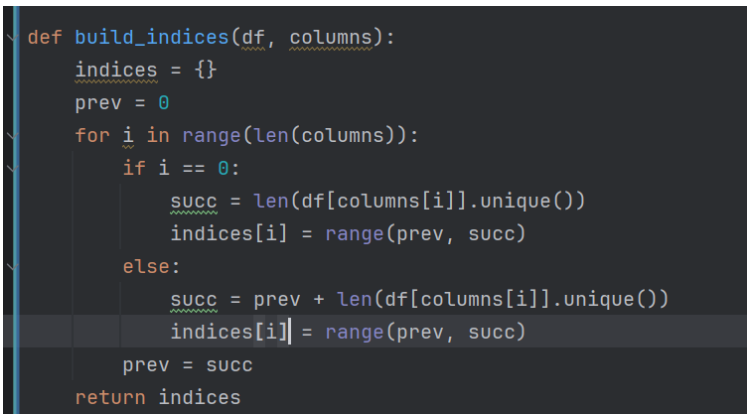
Before discussing the MLP, we need to make a brief digression on dataset preprocessing.

Since the datasets were composed of symbolic variables, they could not be used directly as input for our machine learning model. Therefore, we applied one-hot encoding, a process that transforms each symbolic value into a unique binary vector.

After this procedure, we identified the indexes corresponding to each encoded feature.



(a) Prima immagine



(b) Seconda immagine

Figure 1: Indexing Data Classes

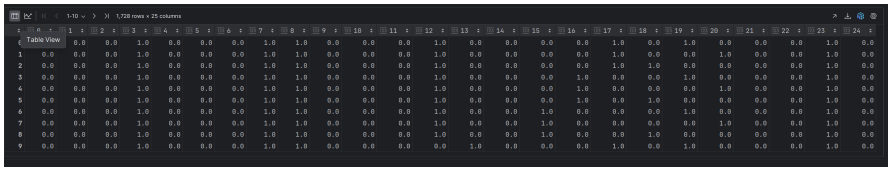


Figure 2: A part of the whole one-hot encoded dataset

Finally, after this process, a function that simulates missing values in a dataset was created.

Given the original dataset, we apply the `adding_missing_values` algorithm, which simulates missing values by zeroing out one-hot encoded columns corresponding to symbolic features.

Specifically, it generates all possible combinations of k symbolic variables, and for each combination, it randomly selects m rows. In those rows, it sets to zero the one-hot encoded columns corresponding to the selected symbolic features.

The target used during training remains the original dataset. In this way, the model learns to reconstruct the missing values based on the available information. We conducted two tests: one where $k = 1$ and the other when $k = 4$.

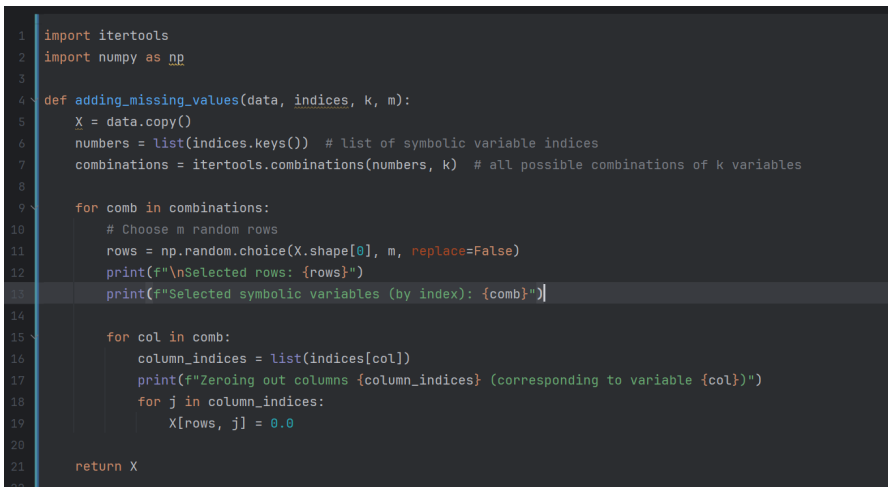


Figure 3: Enter Caption

These operations were done for both datasets.

4 Creating the Model

After preparing the data for training, we split the dataset into training and testing sets. We use the original dataset as the target in order to train the model to reconstruct the missing values. We built the model using the Keras library from TensorFlow.

```
input_dim = X_train.shape[1]

model = keras.Sequential([
    layers.Input(shape=(input_dim,)),
    layers.Dense(128, activation='relu'),
    layers.Dense(128, activation='relu'),
    layers.Dense(input_dim, activation='sigmoid')  # output
     binario
])
```

Figure 4: Architecture of the model: a fully connected feedforward neural network with one hidden layer of 128 neurons (ReLU) and an output layer with sigmoid activation

We used the Adam optimizer and binary cross-entropy as the loss function. The performance metric selected for evaluation was Binary Accuracy.

The model was trained for 50 epochs with a batch size of 32, and 20% of the training data was used for validation.

At the end of the training, the model achieved the following results:

- bit_accuracy = 0.99;
- loss = 2.0413e-04;
- val_loss = 0.0041;

Since both loss and validation loss were very low, the MLP has well generalized the problem. Testing results gave us a missing accuracy of 99%, meaning that the model could correctly reconstruct the dataset from unseen data.

With the same parameters, the same test was applied to the other dataset and it gave us the following results:

- bit_accuracy = 0.99;
- loss = 0.0035;
- val_loss = 0.0237;

The accuracy on missing values during testing gave us a result of 98%.

4.1 Second Experiment

The second experiment was done by adding more missing values to both datasets, by setting $k = 4$. However, even though the model was complex, it still made good results for the Letter dataset:

- bit_accuracy: 0.9399;
- loss: 0.1885;
- val_bit_accuracy: 0.9397;
- val_loss: 0.1886;

However, for the Car dataset, performance drastically decreased:

- bit_accuracy: 0.7494;
- loss: 0.5304;
- val_bit_accuracy: 0.7316;
- val_loss: 0.5730;

Qui invece mostriamo il grafico delle accuracy bit-wise di tutti i modelli.

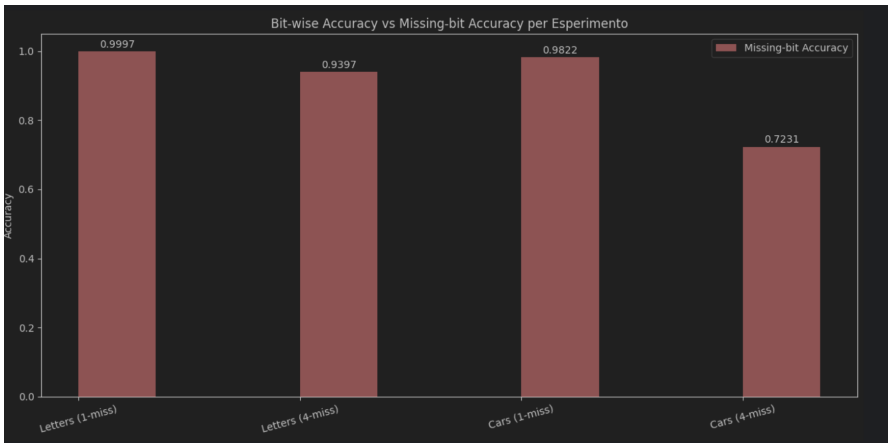


Figure 5: Bit-wise accuracy of all experiments. This plot shows how well the model has placed correct values in missing fields. For $k = 4$, the model didn't respond well using `car.data`.