Predictive Models for BPIC'11 Dataset

Final Project

Course: Process Mining 2020-21
Lectures: De Leoni Massimiliano, Di Francescomarino Chiara

Ma Jiawei

April 24, 2021

1 Introduction

In this project we are asked to build some Machine Learning (ML) models to predict a binary label contained in each sample of the dataset taken form BIP'11, so this is a *binary classification problem.*
Firstly, we preprocessed the data using the required encoding functions and then, we tried many classic ML models and Deep Learning (DL) models, the outcomes are evaluated according to the model accuracy on a separated dataset (test set).
The results indicate that all models utilized performed similarly, in particular using Recurrent Neural Networks, which are the most preferable and popular models in DL field for dealing with temporal sequences, we didn't gained higher accuracy. Taking into account computational cost we concluded that ML techniques are preferable for solving this problem. We also verified that embedding timestamp information contained in the datasets didn't provide more useful information for this prediction task.

2 Dataset

The dataset is a real-life event log taken from a Dutch Academic Hospital. The log contains more than 150.000 events in over 1100 cases came from the data recorded in a Gynaecology department, each event represent an activity took place within the department and for privacy issues, the patients' name are not visible. The data are provided in *XES format*, a standard specifically defined for process mining dataset, and they are divided in two files, one is for model training and other one will be used for model evaluation. Below we reported the first lines of the training dataset:

```
<trace>
<date key="Start date" value="2007-01-01T00:14:24.000+01:00"/>
<string key="concept:name" value="00000912"/>
<string key="Diagnosis code" value="M16"/>
<date key="End date" value="2008-01-28T23:45:36.000+01:00"/>
<string key="Specialism code" value="SC7"/>
<string key="Treatment code" value="TC803"/>
<string key="Diagnosis" value="Sereus adenoca:  ovarium st IIIc"/>
<string key="Diagnosis Treatment Combination ID" value="DTC674183"/>
<string key="label" value="true"/>
```

```
<int key="Age" value="51"/>
<event>
<string key="Activity code" value="AC370000"/>
<string key="concept:name" value="assumption laboratory"/>
<string key="Specialism code" value="SC86"/>
<string key="Producer code" value="CRPO"/>
<string key="lifecycle:transition" value="complete"/>
<string key="Section" value="Section 4"/>
<int key="Number of executions" value="1"/>
<date key="time:timestamp" value="2007-03-01T00:00:00.000+01:00"/>
<string key="group" value="General Lab Clinical Chemistry"/>
</event>
```

In particular, training set consists of 80% of the total dataset.
The log contains many informations about the events, but what we needed were only event sequences with timestamp information and binary labels for each trace which are carried out by this formula in *Linear Temporal Logic*:

$$F('tumormarkerCA - 19.9') \vee F('tumormarkerCA - 125usingMEIA').$$

Some details about these two cancer markers can be found in [2] and [1], whereas the reference website of the BIPC'11 is [3].

3 Data preprocessing

For this work we leveraged the Python library *PM4Py*, it's a quite well documented/user-friendly Python library with event log manipulation and filtering operations, in addition it provides also some common process discovery algorithm. It's a very nice tool for dealing with XES format files and it supports also other data storage format such as CSV.
The main goal of this first part of the project was to create the datasets; training and test set. They were used in the next step for model training. We had several ways to do that by designing proper *encoding methods*, due to different trace length we had to take only *fixed length prefixes* and shorter trace will be completed with some padding methods. Here the length of each encoded traces is fixed to 20, this number is denoted by *PL*, and we are required to create the following functions:

- **encode trace simple index**: it takes as input a trace of the log and returns a list of integers of length PL. Since it was not specified which padding method we should apply, we decided to use *zero-padding* method when needed;

- **encode timestamp**: given an event of a trace it returns a list containing different time format for the event;

- **encode event simple index with timestamp**: we input an event and the function should return a list composed of the event name and the list computed by *encode timestamp* function with the same event object;

- **encode trace simple index with timestamp**: this last function utilizes *encode event simple index with timestamp* function to encode an entire trace, so the result is a list of length PL and each element is computed by the latter function with as inputs events of the trace.

In order to encode the data we created a *dictionary* for the events' name which simply contains *name-integer pairs* so that each name is associate with an integer ranges from 1 to number of different events' name and this is done randomly. The number zero is used for padding the short traces.
We also checked how trace lengths are distributed and since the dataset size is relatively small we computed the exact number of traces shorter than PL. It turned out that almost 35% of cases have to be zero-padded. This is not a problem because we believed that the models are able to distinguish padding part and data information.
As just said the training dataset is small, then we didn't further divided it into training and validation set. The hyperparameter search was carried out using *Cross-Validation* method.
At the end of data preprocessing we obtained two pairs of dataset, one pair created without timestamp information and other one contains both event name and one of timestamp format. To be more clear, the first training dataset was of the shape (number of samples, PL) and the second one had the shape (number of samples, PL, 2).

4 Train predictive models

In this section we describe ML and DL models we experimented with and we report the outcomes come from two different evaluations, first on the dataset built by simple encoding function and the second on the dataset built by simple encoding with timestamp function, both previously mentioned.
As the first operation we verified if the datasets were balanced, since with unbalanced datasets any metric used to evaluate the models would be not reliable. Fortunately, we discovered that training dataset was quite balanced with about 60% of elements labelled as 0 and the test dataset was almost perfectly balanced meaning that random guess on the this dataset will be, on average, 50%, so we need to do better than that.
The codes were executed in *jupyter notebooks* by using popular Python libraries which are *Sklearn* for classic ML models and *TensorFlow* for neural networks based models.

Dataset	**Logistic** (L_2)	**SVM (sigmoid)**	**Tree (depth=3)**
Training set	0.78	0.74	0.78
Validation set	0.61	0.54	0.6
Forest (d=3,n=24)	**GB (lr=**0.1**,d=3)**	**MLP** ($p = 3651$)	**GRU** ($P = 421$)
0.77	0.77	0.79	0.77
0.59	0.61	0.6	0.6
LSTM ($p = 2041$)	**Bidir.** ($p = 5941$)		
0.76	0.77		
0.6	0.62		

Table 1: Model training results with **simple encoding**. We included also the best hyperparameters.

4.1 Models training using simple encoding

After loaded the datasets created in the first part, the only data preparation we performed was the feature standardization, this is a typical data processing operation in ML because the algorithms can achieve better performance and it's true especially in DL, where backpropagation algorithm is theoretically more likely to converge [4].

The table 1 shows the results of the models and below we briefly discuss each model with best hyperparameters tuned by using *Cross-Validation*:

Logistic Regression Model
It's a very first model to try when one has to solve a classification problem, it belongs to the large family of *Generalize Linear Models* and the model outputs probabilities if given inputs are labelled as 0 or 1 in this binary classification problem.
The model is trained by minimizing *binary cross-entropy* loss over the training data and it is robust to outliers in the data with respect to the simple multiple linear models, but it's not suitable for complex scenario (high dimensional problems). Nevertheless, since our dataset is not a very high dimensional problem it performed quite well, it's even better than other more sophisticate models in terms of accuracy on the test set such as SVM.
The hyperparameter search was done by varying the penalization term, the possibilities were L_1, L_2 and elasticnet ratio, the results indicated that the best penalization was L_2, this is something expected since L_1 penalization is good in high dimensional problem (sparsity) and to define the elasticnet ratio requires prior knowledge about the data.

Support Vector Machine (SVM)
It's one of the most popular ML models nowadays, because it can solve non-linear classification problems using different *kernels* and it has efficient training algorithms. However, in our case it didn't performed well with a poor test accuracy. The CV analysis suggested to use a sigmoid kernel.

Decision Tree and Random Forest
These are also very fashionable models due to the fact that, more for Decision Tree, they are highly interpretable. Indeed, even if they had no high accuracy on the test set, drawing the best tree model we can observe that *if a trace is short (we have to complete it with zero-padding)*, then we can be almost sure that it belongs to class 0. This is quite intuitive since who were suspected to be ill they were subjected to multiple examinations. To understand more about the results we need to know more about each events, this is very complicate because we are not experts in the field. We show the plots in fig. 1 and fig. 2
Concerning Random Forest model, we didn't obtain better accuracy and in this case we can compute *importance* of each variable. From fig. 5 and fig. 2 which are related to the sixth tree in the model we see that, once again, the most discriminant feature was the length of the event traces.

Gradient Boosting
This is another very outstanding model in ML and Statistics. Internally, it relies on regression trees to fit the data and so, generally, it has low training loss (high accuracy), on the other hand it's quite interpretable. Here the hyperparameters to test were learning rate and maximum depth of the trees and again, CV was used and it turned out that the best maximum depth number was only 1, this is a reasonable result due to the simplicity of our dataset. Moreover, the test accuracy obtained was quite good with respect to other models.
Fig. 3 is the first regression tree plot and not really surprisingly, it looks like very similar to what we drew for the decision tree model.

Multi Layer Perceptron (MLP)
After classic ML models we also experimented with some DL models and the start point for this problem was a MLP model. Usually MLPs are very powerful to gain high accuracy (low loss) on training dataset but they have not the same performance on a unseen dataset, this is what we

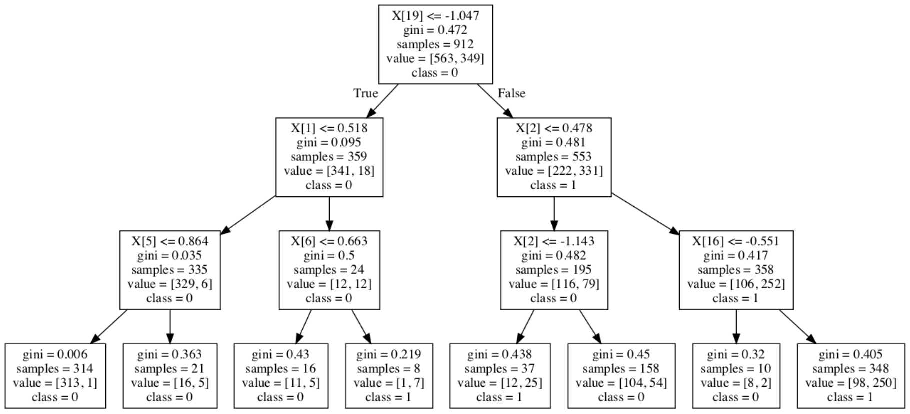

Figure 1: Tree model decision graph: **simple encoding**. The value inside boxes are standardized values, see the attached notebooks for details.

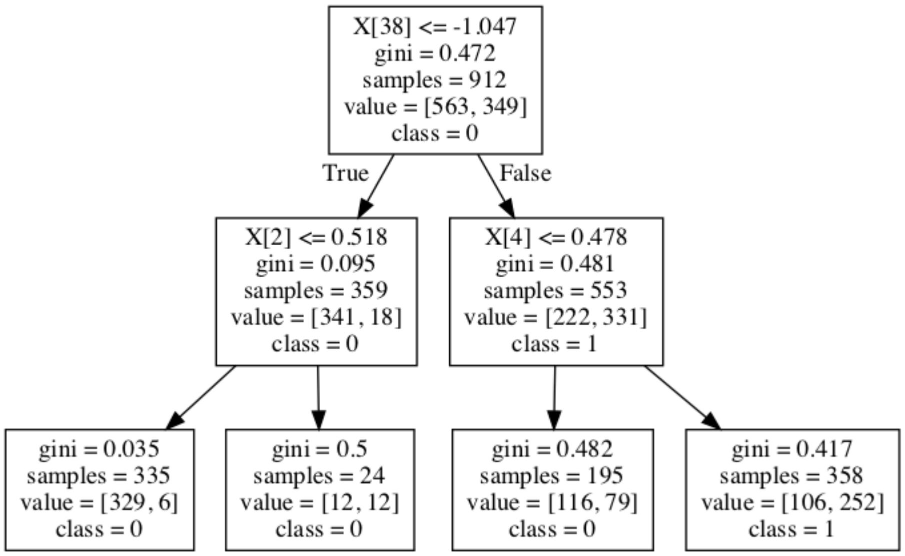

Figure 2: Tree model decision graph: **simple encoding with timestamp**

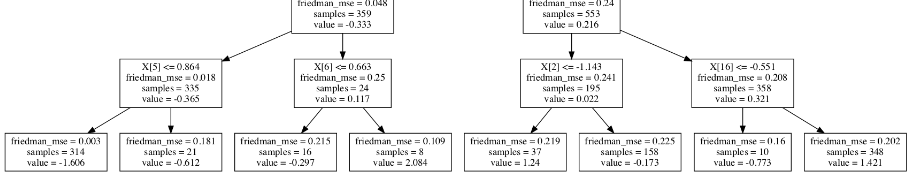

Figure 3: First GB regression tree decision graph: **simple encoding**

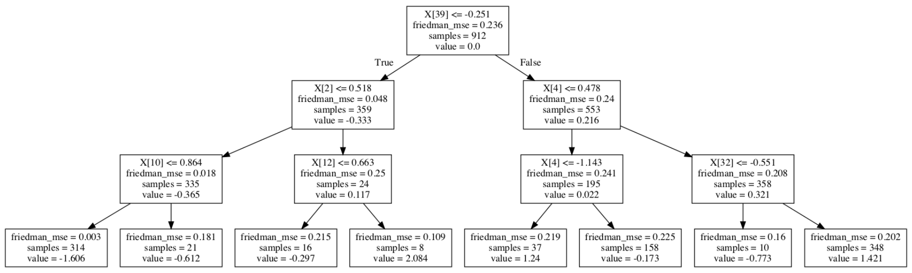

Figure 4: First GB regression tree decision graph: **simple encoding with timestamp**

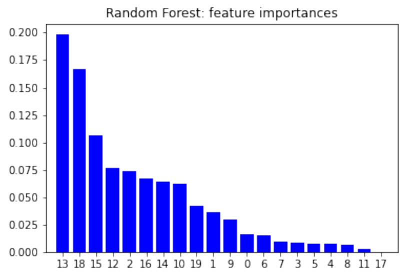

Figure 5: Forest model importance plot: **simple encoding**

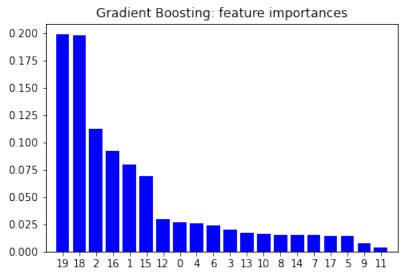

Figure 6: GB model importance plot: **simple encoding**

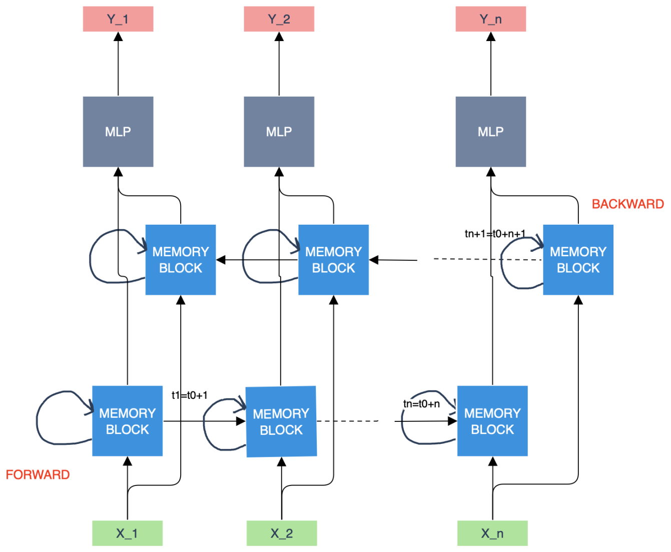

Figure 7: Overall structure of the RNNs designed in this project with LSTM or GRU memory block. Backward branch is present in the bidirectional RNN .

observed using large, in terms of number of units, and deep MLPs in this challenge. Thus, the best architecture we designed is only a MLP with two hidden layers and by using CV analysis we decided to add L_2 regularization terms in order to increase the generalization capability of the model.
The loss function to be optimized was the binary cross-entropy and the optimizer we used was Adam since it's proved to be good choice in general situations. The test accuracy was comparable with the best score returned by classic ML models and it's a small Neural Network to be trained very quickly.

Recurrent Neural Networks (RNNs) These Neural Network based models are known as very powerful DL models to deal with sequential/temporal data as in our context. For this project we decided to devise three types of standard *many-to-one* RNN, these RNNs are of similar structure, the only difference is the *memory unit* embedded; the first has *Long Short Term Memory* (LSTM), the second has *Gate Recurrent Unit* (GRU) and the last has *Bidirectional unit with LSTM unit.* The fig. 7 illustrates the models.
Once again, we realized that a deep and very large RNNs didn't improve the accuracies. We remark that GRU and LSTM based models performed similarly and Bidirectional model had highest accuracy using these datasets, this might due to the fact that *backward* information were useful for predicting the labels.
Finally, we want to point out that RNNs were extremely more expansive to train comparing with previous models and had even worse performances, so we definitely prefer the latter.

As last comment we would like to say that the DL models cannot be interpreted (at least in this case), as what we did for some of ML models. On the other hand, in general,Neural Networks are give satisfactory results, this is a trade-off one has to make very often.

Dataset	**Logistic**	**SVM (sigmoid)**	**Tree (depth=2)**
Training set	N/A	0.75	0.77
Validation set	N/A	0.61	0.6
Forest (d=2,n=6)	**GB (lr=0.01,d=3)**	**MLP** ($p = 9841$)	**GRU**
0.76	0.74	0.79	N/A
0.61	0.6	0.61	N/A
LSTM ($p = 437$)	**Bidir.** ($p = 6021$)		
0.79	0.74		
0.61	0.61		

Table 2: Model training results using **simple encoding with timestamp**.

4.2 Models training with timestamp information

In this last part we trained again the same models built previously but the datasets were enriched by timestamp information.
As before, we preprocessed the data by applying standardization technique. Since now we have one more data dimension we flattened the matrices into vectors, standardized the data and went back to the original dimension. Clearly, whereas we could feed these 2-D data into the RNNs for other models we had to transform them into 1-D data (in this case we doubled the number of variables). For this reason we 'flattened' matrices, so we had long vectors consisting of name-timestamp pairs.
The Table 2 reports the outcomes. Firstly, we noticed that the logistic models could not trained with these higher dimensional data (optimization algorithms couldn't converge) and since GRU based RNN and LSTM based RNN were interchangeable as we observed above, we decided to only consider RNN with LSTM blocks.
We can see that embedding timestamp information didn't improve the test accuracies in this case. The reason of this fact may be that the events were already *chronologically ordered* and so the temporal information gave by the timestamps were redundant. Moreover, the results imply that the time at which the events took place are not helpful.
Regarding interpretation we deduced that what the models (tree, random forest and gradient boosting models) captured as a relevant feature was always related to length of the event traces, this is another reason why we didn't noticed any accuracy increase.

5 Conclusion

In this project we implemented different ML and DL models using a training dataset consisting of event sequences and we tried to achieve as high accuracy as possible on a test dataset, we learnt that the Neural Networks designed were not good as expected, perhaps also other DL models would be not suitable because of the size of the available datasets. All models considered performed similarly and we concluded that classic ML models are enough to tackle this problem and they can be easily trained. Of course, there is still room for improvement, for instance, we can continue to work out better hyperparameters.
Finally, interpretable models indicate that the most discriminant feature of the data was the length; short length traces are very probable to fall in the class 0 (no cancer marks detected).

References

[1] https://en.wikipedia.org/wiki/ca-125.

[2] https://en.wikipedia.org/wiki/ca19-9.

[3] https://www.win.tue.nl/bpi/doku.php?id=2011:challenge.

[4] Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012.