Study of rare Λ_b^0 decay using multivariate analysis techniques

Signal/Background discrimination using BDT and NN

Agostini Federico, Bottaro Federico, Pompeo Gianmarco

University of Padua

July 29, 2019

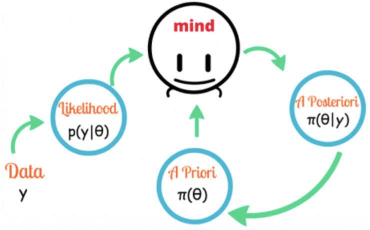
Contents

Physical overview

- We consider the decay
$$\Lambda_b^0 \to \Lambda_c^{*+}\pi^- \to \Lambda_c^+\pi^-\pi^+\pi^-$$
where $\Lambda_b^0 = udb$ and $\Lambda_c^+ = udc$
- Cardinal importance in the study of LFU (Lepton Flavor Universality) violation
- It is kynematically closed, the Λ_b^0 momentum points back to the primary vertex, large impact parameters
- Experimental setup: we study 14-TeV *pp* collisions collected at LHCb and properly selected through **trigger** and **stripping**

The dataset and the ML approach

- We have synthetic data (MC simulations, $\sim$13,500 background events and about 1500 signal) and real data: about 450,000 entries. 28 variables for classification (12 used)
- Simulated data are divided into training and test set to evaluate the algorithms (80%-20%)
- We use the data as an independent source of information, without theoretical (physical) constraints: we are turning perspective around ($\rightsquigarrow$ *bayesian approach*)

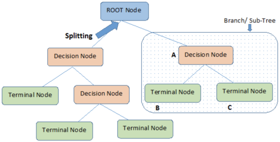


Feature filtering

In order to make sure learning actually happens, we need to disregard features that are too *informative*.

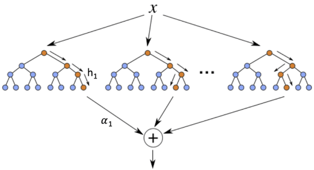
Decision Tree

Data structure composed by different nodes, suitable for regression and classification. Overfitting is avoided by adding a stopping criteria or pruning the tree ($\rightarrow$ low bias, high variance).



Decision Tree Ensembles

Combination of DT, through bagging or boosting. The trees are trained over bootstrapped samples from the training data. The variance is averaged over all the base algorithms, but correlation arises.
In boosting, each of the base algorithm is added sequentially, in order to *pay more attention* on training tuples that were misclassified by previous algorithms.



Gradient Boosted Decision Tree

Loss function is optimized using gradient descend method.

Boosted Decision Tree

R implementation and parameters

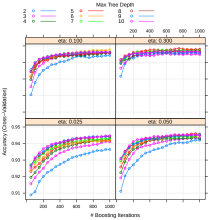
The analysis is done using the R implementation of XGBoost.
The tuning procedure involves the following parameters:

- **nrounds** Number of trees.
- **eta** Step size shrinkage used in update to prevents overfitting; it shrinks the feature weights to make the boosting process more conservative.
- **max_depth** Maximum depth of a tree. Increasing this value will make the model more complex and more likely to overfit.
- **min_child_weight** Minimum number of instances needed to be in each node. The larger, the more conservative the algorithm will be.
- **gamma** Minimum loss reduction required to make a further partition on a leaf node of the tree. The larger, the more conservative the algorithm will be.
- **colsample_bytree** Subsample ratio of columns when constructing each tree; this prevents overfitting
- **subsample** Subsample ratio of the training instances; this prevents overfitting.

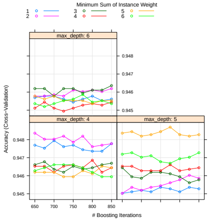
Boosted Decision Tree

Parameters tuning

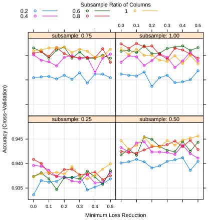
The BDT has been optimized with a *Grid Search* procedure, with cross validation, using the `R` package `caret`.



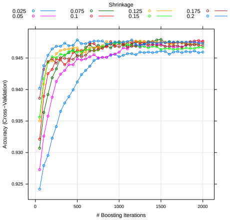
Learning rate, number of trees, maximum depth of the tree (coarse)



Maximum depth of the tree, child weight (fine)



Column and row subsample and gamma



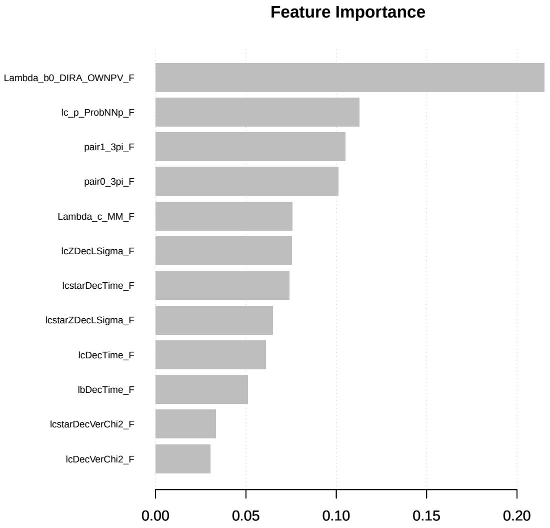
Learning rate, number of trees (fine)

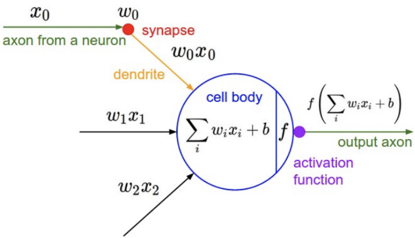
Boosted Decision Tree

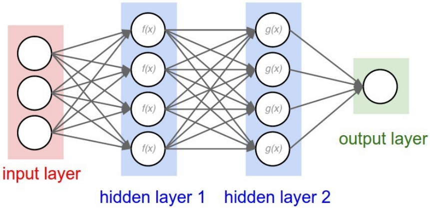
Final model

Best BDT model:

nrounds	1500
eta	0.075
max_depth	5
gamma	0.1
colsample_bytree	0.8
min_child_weight	5
subsample	1
Accuracy (training)	0.947
Accuracy (test)	0.933







A neural network is a collection of nodes organized in layers at different depths. Each node applies a specific activation function to its input.

Neural Network

Vocabulary

Activation function

Function that is applied by each neuron to its inputs in order to compute the output and establish whether the neuron is activated (it *fires*).

Loss function

It is a method to evaluate how well a specific algorithm models the given data. If predictions deviate too much from actual results, the loss function outputs a very large number.

Optimizer

Technique used to shape a model into its most accurate form possible by means of tuning its parameters.

Neural Network
Implementation

- R package: `keras`

Note

Before using it as input, the data needs to be pre-processed: mean centered to 0 and variance normalized to 1

- We tune the parameters as follows:

  activation function Chosen between sigmoid and softmax in output layer (hidden always sigmoid)

  optimizer Chosen between SGD, RMSProp, Adam

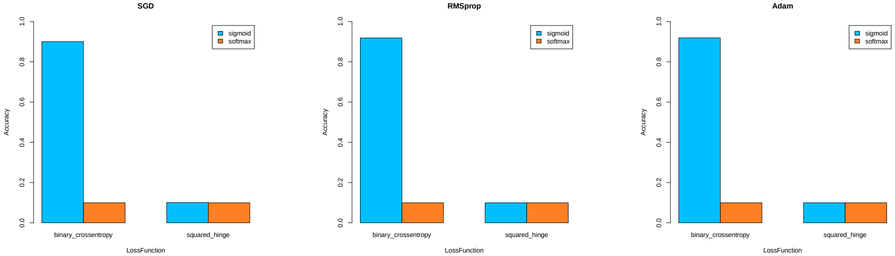
  loss function Computed with: binary cross-entropy or squared hinge loss

- These are studied in a simple 5-neuron 1-hidden layer NN; then we add complexity increasing layers (1,2,5) and hidden neurons (mostly combinations of 5 and 10)

Neural Network

Parameters tuning - 1

- These are the accuracies obtained with the different optimizers, for each loss and activation function



Best result: sigmoid, Adam, binary cross-entropy

- Fixing the best result, we investigate the accuracy with respect to hidden layers and units

Single Layer								
Nodes	1	2	3	10	30	50	100	150
Accuracy	0.901	0.901	0.901	0.921	0.920	0.919	0.920	0.919

Multiple Layers						
Nodes	(5,5)	(5,10)	(10,5)	(10,10)	(15,15)	(10,5,3,2)
Accuracy	0.920	0.919	0.918	0.919	0.918	0.901

- 1 hidden layer, 10 units is the best compromise: accuracy and computational gain

A simpler model: Logistic Regression

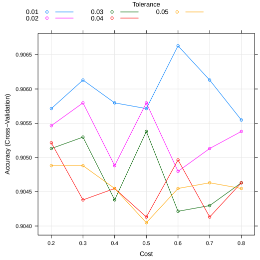
Since the amount of MC data is not too big, we try a simpler Logistic Regression model with L1 regularization.

Logistic Regression

It is a binary classifier, whose output is a real number between 0 and 1. This is a soft classifier: the output can be interpreted as the probability of belonging to a specific class.

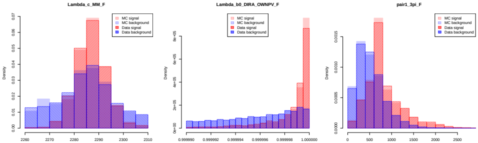
The R implementation is done with the `LiblineaR` package, while the optimization through `caret`, as before.

`Regularization`	L1
`Cost`	0.6
`Epsilon`	0.01
Accuracy (training)	0.906
Accuracy (test)	0.905



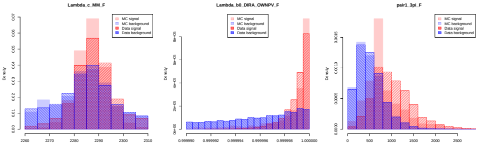
Conclusive summary

Comparison with real data



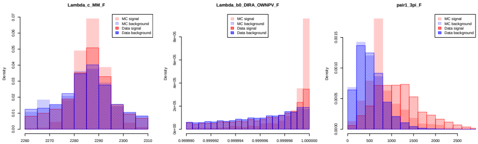
BDT

Accuracy	0.933
Signal	18000
Background	454750



NN

Accuracy	0.921
Signal	13949
Background	458801

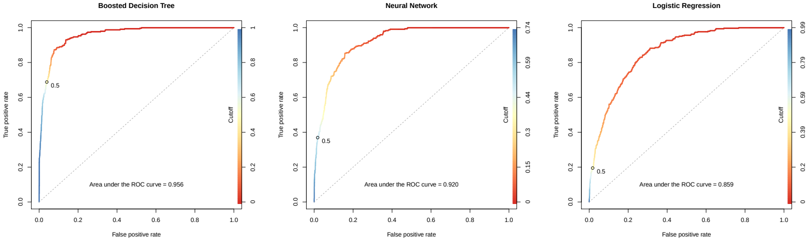


Logistic

Accuracy	0.905
Signal	9715
Background	463035

Conclusive summary
ROC curves

- ROC curves again show superiority of BDT: for threshold = 0.5, 70% true positives (about 5% false positives); AUC value confirms this



- Amount of data (especially MC) appears to be somewhat small: possible limitation to NN?

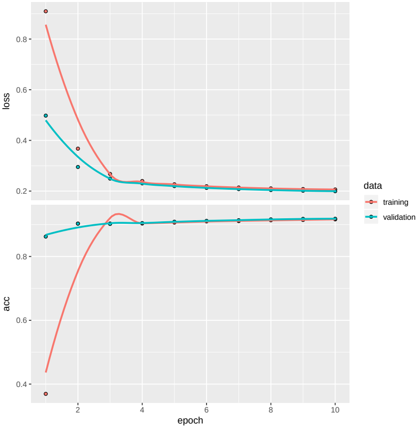
Thank you for your attention

- Activation function
  - Softmax: $\sigma(x)_i = \frac{e^{x^{z_i}}}{\sum_{j=\mathbf{1}}^{K} e^{x^{z_i}}}$ where K is the number lenght that softmax take in input
  - Sigmoid: $P(x) = \frac{1}{1+e^x}$
- Loss function:
  - Binary cross entropy: $H_p(x) = -\frac{1}{N}\sum_{i=1}^{N} x_i log(p(x_i)) + (1-x_i)log(1-p(x_i))$ where x_i is the label and $p(x)$ is the predicted probability of having one of the two possible label for all N points.
  - Squared hinge loss: $L(x) = \sum_{i=1}^{N} max(0, 1 - x_i\hat{x}_i)$ where $\hat{x}_i$ is the predicted label and x_i is the given label

BACKUP

Loss and accuracy trends over epochs

We see that the loss is highly reduced already after a limited number of epochs

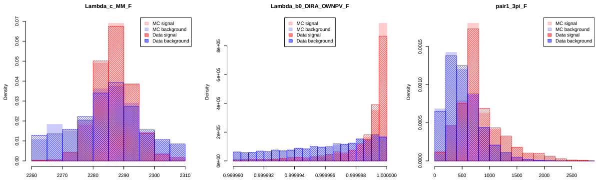


Default value: 10

Layer

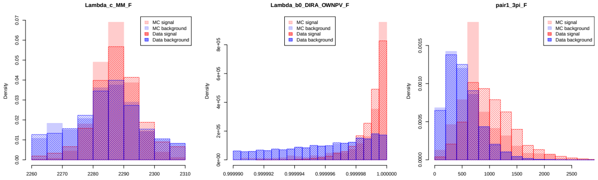
Collection of nodes operating together at the same depth within a neural network. There can be **input**, **hidden** and **output** layers.

- The implementation of both algorithms was completed successfully and in fact both give satisfactory results
- The BDT classification replicates more closely the real data



BDT

Accuracy	0.933
Signal	18000
Background	454750



NN

Accuracy	0.921
Signal	13949
Background	458801