Multivariate methods and continuum suppression

Belle Analysis Workshop 2024

Rahul Tiwary

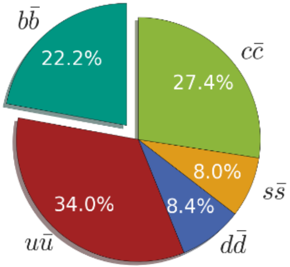
TIFR, Mumbai

Outline

- Motivation
- Feature variables
- BDT
- Performance
- Overtraining
- Hyperparameter optimization
- Summary

Motivation

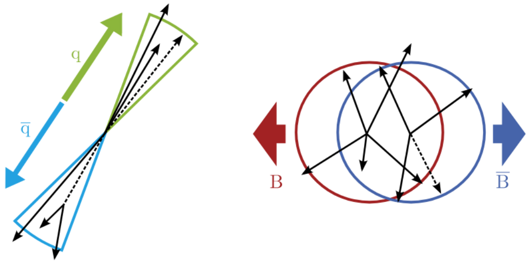
- At Belle II we study electron positron collision data.
- e^+e^- machines like Belle II are referred to as B-factories, but they also have other products in the shelf.



Relative cross sections of various $e^+e^- \to q\bar{q}$ procesess at $\Upsilon(4S)$ resonance

Motivation

- In order to study properties of B decay we need to supress backround from $e^+e^- \rightarrow$ light quark events.
- We can exploit the topological differences between $B\overline{B}$ and light quark events



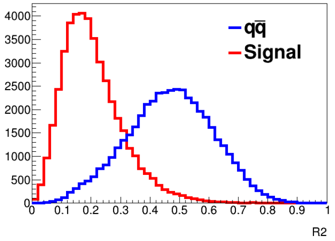
Topology of B decays is spherical compared to jet-like light quark events.

Feature variables

- We utilise a multitude of custom made variables to fight continuum background.
- Some of these variables are listed below:
    - Fox-Wolfram moments
    - Cleo cones
    - Magnitude of thrust for signal B, ROE
    - Cosines like $cos(\theta_{T_B})$, $cos(\theta_{T_B T_{ROE}})$, $cos(\theta^*_B)$.
- A brief summary of these varibales can be found in
    - The Physics of B Factories
    - Belle II Phyisics Book.

Feature variables: R_2

- $H_k = \sum_{i,j} |\tilde{p_i}||\tilde{p_j}| P_k(\cos\theta_{ij})$, kth order Fox-Wolfram moment. $\tilde{p_i}$ is momentum of ith particle.
- θ_{ij} is angle between $\tilde{p_i}$ and $\tilde{p_j}$, P_k is kth order Legendre polynomial
- $R_k = H_k/H_0$.



Plot for R_2 variable

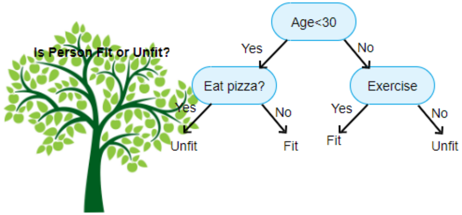
B decay daughters have spherical topology with large θ_{ij}. The cosine for larger values of theta is small hence the B events populate the lower region of R_2

Tools

- Variables described in previous slides show a different shape for signal B and continuum background events.
- The idea is to combine the individual weak classifiers to create a strong classifier.
- Multivariate analyzers (MVA) are the tools which are used to perform this task.
- I will discuss BDT, a popular MVA method and some general concepts.

Tools: Decision tree

- A Decision Tree (DT) is an MVA which classifies events by applying rectangular cuts on input variables.
- The cuts are applied in order to minimize entropy.
- Entropy → Gini Index
- Gini Index: p (1 – p), p (purity) = S / (S + B)



A DT to classify people as healthy or unhealthy. The input variables are Age, Eat pizza and Exercise

To understand more about DT refer "Leo Breiman et. al Classification and Regression Trees (1984)".

Tools: Boosted Decision tree (BDT)

- A single DT is a weak-learner[1].
- Combine many DT's to prepare a strong classifier. Boost!!
- Ensemble output of DT's $\mathcal{O} = \sum_t w_t h_t$, here w_t is the weight for tree h_t.
- Iteratively add more trees to the ensemble $\rightarrow$ increase classification power.
- New weights calculated by minimizing a loss function.
- Gradient boost $\Rightarrow$ uses gradient decent to re-weight
- Stochastic gradient boost[2] $\Rightarrow$ use a randomly drawn sub-sample instead of full sample at each boosting step.

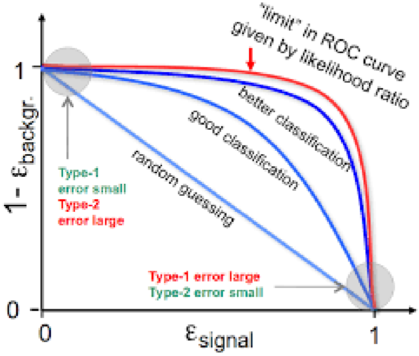
[1] simple classifier with few parameters

[2] Stochastic gradient boosting, Freidman (2000)

Performance

- The performance of an MVA can be checked by plotting the Reciever operator charecteristic curve (ROC).
- ROC: ROC is prepared by applying the training results to MC datasets, then we use the MVA output variable (MVAout) as a metric to findout signal selection efficiency and background rejection for varioous cut values of MVAout.
- Area under ROC tells us about the power of classifier, for a random classifier ROC integral = 0.5 and for a perfect classifier ROC integral = 1.0.

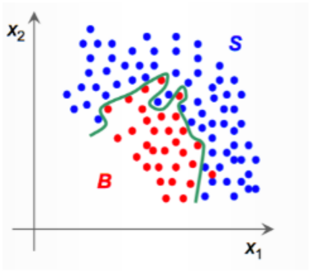
Performance

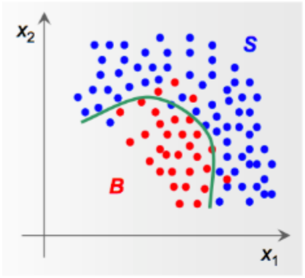


ROC for various classifiers

- We can improve the ROC by adding more feature variables. Note: This is not always guaranteed! it depends on whether the new variable is contributing some extra information to the existing set of feature variables.

Overtraining





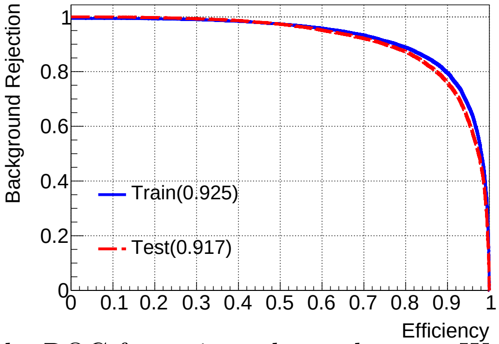
Example of overtrained (left) and properly trained (right) classifiers

- The left plot shows decision boundary for an overtrained classifier. Notice how it has learned the statistical fluctuations of the sample!
- The right plot shows a general decision boundary for a properly trained classifier.
- Overtrained MVA: A classifier which has learned statistical fluctuations of the training dataset, it is biased!!!.

Overtraining check

- Overtraining check can be done by looking at ROC and Kolmogorov smrinov test resuts.
- We apply the results of MVA training to the train and test datasets. Next we plot the ROC for train and test dataset in the same canvas.
- If the classifier is not biased we should observe similar distribution of ROC for train and test dataset.
- This implies that the classifier is not biased and doesn't performs equally well for an independant sample.

Overtraining check:

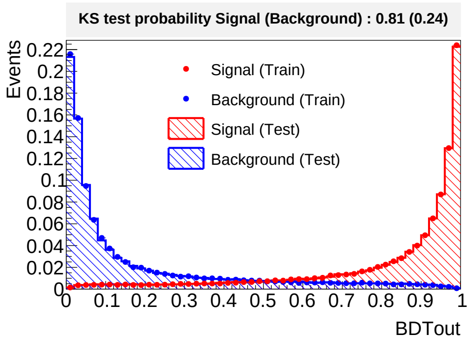


Above we plot the ROC for train and test dataset. We see that the performance of the classifier is similar for both the dataset (we can do a bit better I guess!).

Overtraining check

- Next we look at Kolmogorov smirnov (KS) test.
- The MVAout distribution of signal and background events from train and test sample are taken and we perform KS test to get a p value. The p value is equal to 1 if both the distributions are exactly the same, 0 if they are completely different.
- The idea is that for overtrained scenarios the MVA will be biased towards the train dataset and hence the KS test value will be close to zero.
- A 95% confidence level can be chosen for KS test p value, this means we can say cases where the p value is below 0.05 as overtrained with a 95% confidence limit.

Overtraining check



Here we show the results of a KS test for a toy MC sample, we observe no significant overtraining.

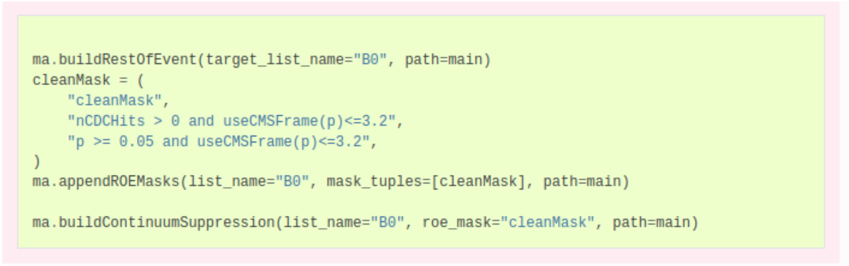
Hyperparameters

- Hyperparameters: Tunable parameters of the MVA which decide how the training will be performed.
- FastBDT hyperparameters:
  - nTrees : Number of trees
  - nLevels : Number of stages for split in a tree
  - nCuts : Granularity for finding split value = 2^{nCuts}
  - shrinkage : Learning rate
- Better to prune the tree (fix nLevels) to avoid overtraining.
- nTrees and shrinkage are anti-correlated.

Summary

- Discussed continuum background events encountered in analyses involving B decays.
- A brief overview of BDTs and some general concepts.
- Backup: Implementing a BDT in BASF2.

BASF2 Implementation: Saving variables

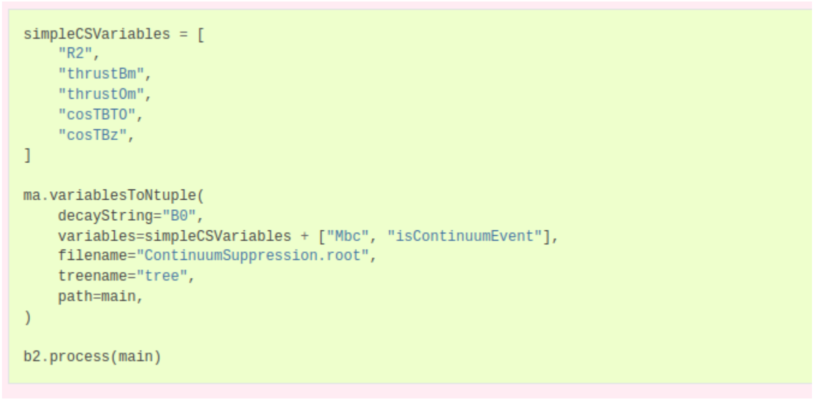


```
ma.buildRestOfEvent(target_list_name="B0", path=main)
cleanMask = (
    "cleanMask",
    "nCDCHits > 0 and useCMSFrame(p)<=3.2",
    "p >= 0.05 and useCMSFrame(p)<=3.2",
)
ma.appendROEMasks(list_name="B0", mask_tuples=[cleanMask], path=main)

ma.buildContinuumSuppression(list_name="B0", roe_mask="cleanMask", path=main)
```

- **Rest-of-event** (ROE): Everythingin the event excluding tracks and clusters of reconstructed signal B meson.
- **cleanMask** : Selection cuts for tracks and clusters to be included in ROE calculation.

BASF2 Implementation: Saving variables



```
simpleCSVariables = [
    "R2",
    "thrustBm",
    "thrustOm",
    "cosTBTO",
    "cosTBz",
]

ma.variablesToNtuple(
    decayString="B0",
    variables=simpleCSVariables + ["Mbc", "isContinuumEvent"],
    filename="ContinuumSuppression.root",
    treename="tree",
    path=main,
)

b2.process(main)
```

BASF2 Implementation: Training

- We train the MVA using MC events, the MVA learns topological differences between signal and background events and helps in classification.
- We prepare two independant datasets of MC called **train** and **test** sample.
- Both the **train** and **test** sample are prepared with equal number of signal and background events.
- We use the **train** MC sample for training the MVA, the **test** MC sample is used as an independant sample to check for overtraining.

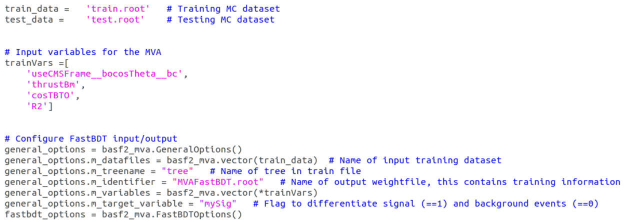
Hyperparamter optimization

- How to find the best set of hyperparameters?
- Use the default parameters in FastBDT, understand the sample on which the default parameters were optimized and then perform training on a similar sample!
- Manually do optimization
    - Random search : Self explanatory
    - Grid search : Give the set of values of hyperparameters and let the code search for the best. For example, we give nTrees(100, 200, 300) and nCuts(3, 4, 5) as an input and the code provides nTrees = 200, nCuts = 3 as the best choice.
    - Bayesian optimization : This algorithm takes bayesian priors in to account while searching the parameter space.
- Example codes: MVA repository.

BASF2 application: Training

- I have prepared an example code to perform training of FastBDT.
- Path : /group/belle2/users/rahul/BAW_2021_CS
- MVATrain.py ⇒ Training script
- train.root and test.root ⇒ train and test datasets
- MVAFastBDT.root ⇒ weight file obtained after training
- Training_result.pdf ⇒ results of training
- The code can be executed in BASF2 environment using the command: "basf2 MVATrain.py". Once the code is finished you should get "MVAFastBDT.root" and "Training_result.pdf" files as the output.

BASF2 application: Training



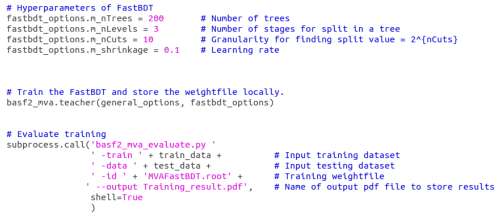
```
train_data =   'train.root'   # Training MC dataset
test_data =    'test.root'    # Testing MC dataset


# Input variables for the MVA
trainVars =[
    'useCMSFrame__bocosTheta__bc',
    'thrustBm',
    'cosTBTO',
    'R2']


# Configure FastBDT input/output
general_options = basf2_mva.GeneralOptions()
general_options.m_datafiles = basf2_mva.vector(train_data)  # Name of input training dataset
general_options.m_treename = "tree"   # Name of tree in train file
general_options.m_identifier = "MVAFastBDT.root"   # Name of output weightfile, this contains training information
general_options.m_variables = basf2_mva.vector(*trainVars)
general_options.m_target_variable = "mySig"   # Flag to differentiate signal (==1) and background events (==0)
fastbdt_options = basf2_mva.FastBDTOptions()
```

- Here we show the first part of python script used for training.
- We basically define the training and testing datasets as well as the name of feature variables.
- We also define the name of tree (tree), the name of output weightfile (MVAFastBDT.root) and the name of flag (mySig) to differentiate signal and background events.

BASF2 application: Training



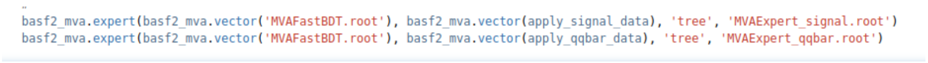
```
# Hyperparameters of FastBDT
fastbdt_options.m_nTrees = 200      # Number of trees
fastbdt_options.m_nLevels = 3       # Number of stages for split in a tree
fastbdt_options.m_nCuts = 10        # Granularity for finding split value = 2^{nCuts}
fastbdt_options.m_shrinkage = 0.1   # Learning rate


# Train the FastBDT and store the weightfile locally.
basf2_mva.teacher(general_options, fastbdt_options)


# Evaluate training
subprocess.call('basf2_mva_evaluate.py '
                ' -train ' + train_data +         # Input training dataset
                ' -data ' + test_data +           # Input testing dataset
                ' -id ' + 'MVAFastBDT.root' +     # Training weightfile
               ' --output Training_result.pdf',   # Name of output pdf file to store results
                shell=True
                )
```

- Here we show the second part of python script used for training.
- We can adjust the various hyperparameters by changing their values.
- Perform the training using **basf2_mva.teacher** module.
- Use **basf2_mva_evaluate.py** script to evaluate results of training.

BASF2 application: MVA on nTuples



```
basf2_mva.expert(basf2_mva.vector('MVAFastBDT.root'), basf2_mva.vector(apply_signal_data), 'tree', 'MVAExpert_signal.root')
basf2_mva.expert(basf2_mva.vector('MVAFastBDT.root'), basf2_mva.vector(apply_qqbar_data), 'tree', 'MVAExpert_qqbar.root')
```

- We use **basf2_mva.expert** module to apply the results of training on nTuples.
- The inputs are (from left to right) ⇒ MVA weight file (generated during training), path of input root file (on which we wish to apply MVA), name of the tree (in the input root file), path of output root file (where MVA output will be saved).