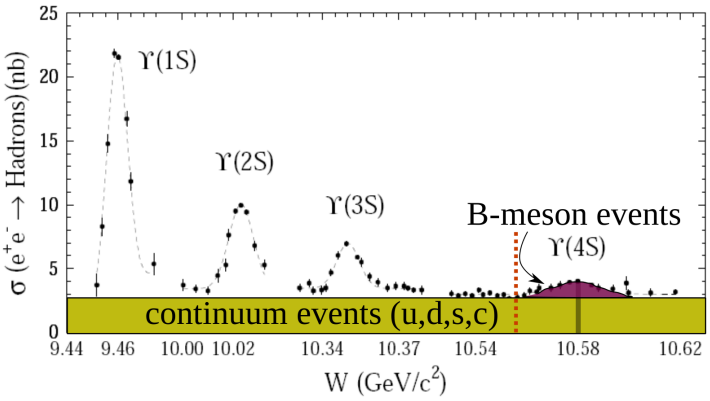
Introduction

In this exercise we will learn the basic usage of three widely used machine learning frameworks. The exercises 2-4 are independent from one another.

We use a Monte Carlo simulated dataset from the Belle II experiment. The Belle II experiment is located in Tsukuba, Japan at the SuperKEKB asymmetric electron-positron collider, which operates at a center of mass energy of 10.58 GeV. If an electron-positron pair annihilates into a virtual photon two different physical processes are likely to happen: either an intermediate $\Upsilon(4S)$ resonance (a bound $b\bar{b}$ state) is produced, which decays into two entangled B mesons $e^+e^- \to \Upsilon(4S) \to B\bar{B}$, or two unbound quarks are produced $e^+e^- \to q\bar{q}$ ($q \in \{udsc\}$), as shown in the figure below. We call the former a $B\bar{B}$ event, and the later a continuum event.



In this exercise we want to identify the continuum events in order to discard them in our analysis. This is called continuum suppression.

1 Understanding the data

Understanding the data is usually an important and time-consuming step during data analysis. Detecting errors due to data collection, preprocessing the data to cast it in a usable format for the task at hand, check for differences between simulated and real data, choosing sensible pre-cuts to minimize the required computing resources and systematic uncertainties. We will keep this part very short here.

Our dataset contains:

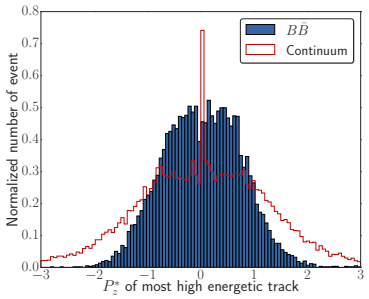
- 12 features for each of the 10 most high energetic tracks in each event:
  - the four-momentum (P_x, P_y, P_z, E);
  - the particle identification information for Kaons Kid, electrons eid, muons $muid$, and protons $prid$;
  - the impact parameters of the track d_x, d_y, d_z and the $pValue$ of the track fit.
- 8 features for each of the 10 most high energetic unmatched clusters in each event:

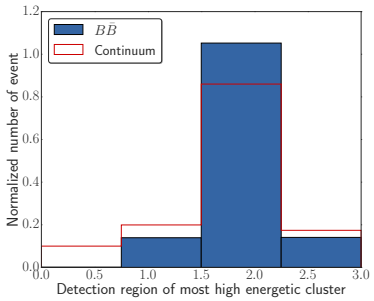
  - the four-momentum (P_x, P_y, P_z, E);
  - the number of hits per cluster $clusterNHits$, a time information $clusterTiming$, the ratio of 3x3 and 5x5 crystal cells around the center of the cluster $clusterE9E25$ and the region of the cluster $clusterReg$: backward, barrel or forward.

- the Monte Carlo truth $isContinuumEvent$.

The are two ROOT files: `data.root` contains $\approx$ 30000 continuum and 10000 $B\bar{B}$ events, `data_validation.root` contains nearly the same amount of data represents an independent test sample. In addition both datasets are also available as pickled `pandas.DataFrames` in the corresponding files named `data.df` and `data_validation.df`.

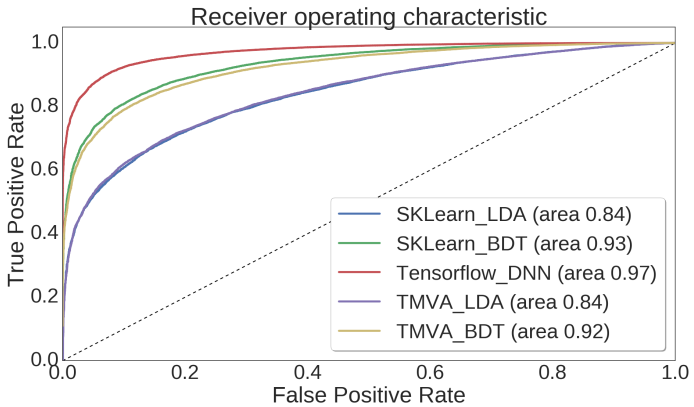
The two figures below show a feature of the most high energetic track and cluster, respectively. Events without any tracks or clusters are stored as 0, hence the peaks at 0.





Look at the distributions for some of the variables and familiarize yourself with the dataset. You can use the python script `plot.py` to do this, for instance try `python3 plot.py Pz_T0` to reproduce the plot above.

Now you can start with one of the three exercises below. All examples write out a result text file, containing the estimated continuum probabilities for each event in the independent validation dataset. At the end, you can compare the different frameworks by executing `python3 evaluate.py`. Using the default settings it should look like this in the end:



2 TMVA

The Toolkit for Multivariate Data Analysis with ROOT (TMVA) is traditionally used in high energy physics. It is shipped with ROOT and integrates well in the workflow of data analysis in high energy physics, which typically employs ROOT's TTree. The documentation is available here `http://tmva.sourceforge.net/docu/TMVAUsersGuide.pdf`.

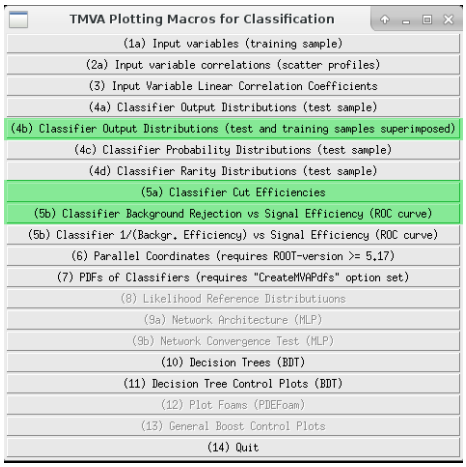
> Take a look into the ROOT Macro `TMVATrain.C`. Train a Linear Discrimant and a Boosted Decision Tree using this macro by executing it with ROOT `root -l TMVATrain.C`. Use the output of TMVA to answer the following questions: Which method is better? Are the methods overfitted? Which variables contribute the most to the classification?

```
--- Factory                : Ranking input variables (method specific)...
--- LDA                    : Ranking result (top variable is best ranked)
--- LDA                    : ----------------------------------------
--- LDA                    : Rank : Variable         : Discr. power
--- LDA                    : ----------------------------------------
--- LDA                    :    1 : Energy_C1        : 1.774e-01
--- LDA                    :    2 : clusterE9E25_C2  : 1.698e-01
--- LDA                    :    3 : prid_T2          : 1.695e-01
--- LDA                    :    4 : eid_T0           : 1.691e-01
--- LDA                    :    5 : prid_T1          : 1.585e-01
--- LDA                    :    6 : muid_T0          : 1.214e-01
--- LDA                    :    7 : prid_T0          : 1.129e-01
--- LDA                    :    8 : Kid_T2           : 1.055e-01
--- LDA                    :    9 : Energy_T2        : 1.024e-01
```

```
--- Factory                :
--- Factory                : Evaluation results ranked by best signal efficiency and purity (area)
--- Factory                : ------------------------------------------------------------------------------
--- Factory                : MVA              Signal efficiency at bkg eff.(error):       | Sepa-    Signifi-
--- Factory                : Method:          @B=0.01    @B=0.10    @B=0.30    ROC-integ. | ration:  cance:
--- Factory                : ------------------------------------------------------------------------------
--- Factory                : BDT            : 0.532(04)  0.783(03)  0.906(02)    0.919    | 0.561    1.470
--- Factory                : LDA            : 0.330(03)  0.622(03)  0.794(03)    0.842    | 0.360    0.911
--- Factory                : ------------------------------------------------------------------------------
--- Factory                :
--- Factory                : Testing efficiency compared to training efficiency (overtraining check)
--- Factory                : ------------------------------------------------------------------------------
--- Factory                : MVA              Signal efficiency: from test sample (from training sample)
--- Factory                : Method:          @B=0.01             @B=0.10            @B=0.30
--- Factory                : ------------------------------------------------------------------------------
--- Factory                : BDT            : 0.532 (0.884)       0.783 (0.928)      0.906 (0.957)
--- Factory                : LDA            : 0.330 (0.345)       0.622 (0.628)      0.794 (0.792)
--- Factory                : ------------------------------------------------------------------------------
```

TMVA automatically splits the provided dataset into a training and test dataset, and can create a number of control plots to check for overfitting and evaluate the performance of the classifiers. This functionality is provided by the TMVAGui

> Start TMVAGui using `root -e "TMVA::TMVAGui(\"TMVA.root\")"`. Look at the control plots 4b, 5a and 5b; and think about which conclusions you can derive from them.



At this point it should be clear that our classifiers are heavily overfitted. Overfitting can be avoided by decreasing the number of degrees of freedom, which are controlled by the hyper-parameters of the method. The TMVA BDT has a lot of them, see `http://tmva.sourceforge.net/optionRef.html` for more information.

> Take a look at the available configuration options of the BDT method. Afterwards change the configuration of the BDT to avoid overfitting and improve its performance. A good starting point is the configuration of the BDT which is already included in the macro but commented out.

The line looks like this:

```
//factory->BookMethod(TMVA::Types::kBDT, "BDT",
 "H:V:NTrees=100:BoostType=Grad:Shrinkage=0.1:
 UseBaggedBoost:BaggedSampleFraction=0.5:nCuts=100:
 MaxDepth=3:IgnoreNegWeightsInTraining");
```

Advanced

Another possibility to reduce overfitting, is to restrict the dimensionality of the input space. You can define arbitrary functions of your input variables by adding new variables e.g. `Pt := sqrt(Px**2 + Py**2)`. Unfortunately this works only for the Factory and not for the Reader. There is also the possibility to add transformations of the input features e.g. `VarTransform=N,D` added to the configuration string of a method would normalizes and decorrelates the input features before passing them on to the method.

> Reduce the number of variables by exploiting the cylindrical symmetry of the detector. Do you encounter any problems?

TMVA provides a large number of methods and options. In the example above we used a linear discriminant and a BDT, however you are free to test out other methods and options. The code for training an SVM or MLP is already included in the macro, but is commented out. Be careful, the SVM and

MLP may take very long to train and are less ,,stable" than one would expect. You may want to continue playing around with these methods in your free time.

3 SKLearn

Scikit-Learn (aka SKlearn) is a machine learning library build on top of the scipy project, which aims to provide a full data analysis stack for science written in python. Hence SKLearn integrates well with famous python libraries like numpy, pandas and matplotlib. You can find more information on these frameworks here `http://scikit-learn.org` and here `https://www.scipy.org`.

> Take a look into the SKLearn example `SKLearnTrain.py`. Train a Linear Discrimant and a Boosted Decision Tree using this python file by executing it with python `python3 SKLearnTrain.py`. Use the output of SKLearn to answer the following questions: Which method is better? Are the methods overfitted? Which variables contribute the most to the classification?

The methods are already quite good although we used the default hyper-parameters. You can check out the available options online `http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html`

> Take a look at the available configuration options of the BDT method. Afterwards change the configuration of the BDT to improve its performance. A good starting point is the default configuration of the BDT.

The line looks like this:

```
#bdt = sklearn.ensemble.GradientBoostingClassifier(
   n_estimators=150, subsample=0.5,
   max_depth=4, min_samples_leaf=100)
```

Advanced

We can automatize the process of searching the optimal hyper-parameter set. A simple approach is the systematic search on a grid. Of course you could use any optimization algorithm, however SKLearn already implements the grid-search via the `GridSearchCV` class. CV stands for cross validation, this algorithm automatically splits the dataset into N pieces and uses N-1 to train, and the last one to estimate the performance, it does this N times choosing a different piece to test each time.

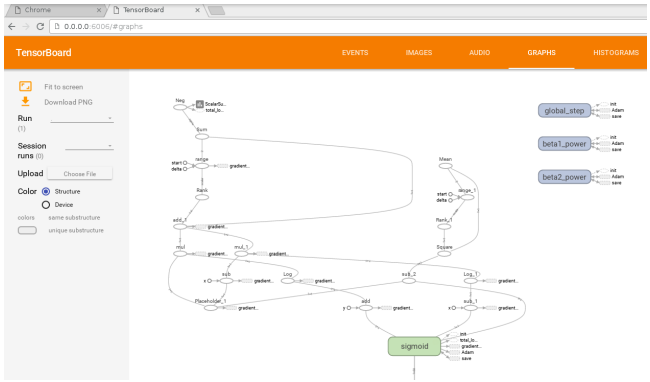
> Perform a grid search to find the optimal hyperparameter set for the BDT.

The example file contains already code to train a Quadratic Discrimant, a SVM and an MLP. Unfortunately the MLP implementation is not in the stable SKLearn version included, only in the current development version. You may want to continue playing around with these methods in your free time.

4 Tensorflow

Tensorflow a the Deep Learning Framework developed by google. It is possible to create very sophisticated machine learning algorithms using this framework and train them on large data samples. On the other hand, training such a model can take a long time even if one employs GPU clusters. Therefore an already trained model is included in the example directory. You can find more information here `https://www.tensorflow.org/`.

You can visualize the model using the tensorboard. Start it using `tensorboard --logdir .`, next open `http://0.0.0.0:6006` in your web-browser and click on GRAPHS. It should look like in the figure below. Follow the flow of the tensors through the network starting at the nodes called placeholder.



Apply the trained model by executing `python3 TensorflowApply.py` and compare it to the classifiers of the previous exercises using the `evaluate.py` script.

The code used to train the Tensorflow model is in the file `TensorflowTrain.py`, you can take a look if you are interested. The performance of the deep neural network outperforms the other methods, but to be fair, we cheated! Do you see the two advantages of the Tensorflow model that the other methods didn't have?