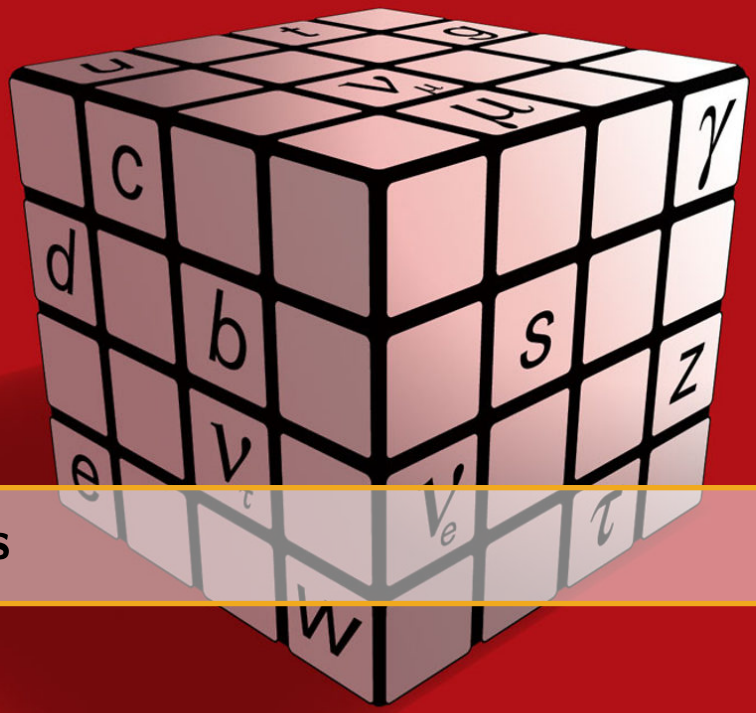


Table of Contents

1. Reconstruction, Cleaning and Analysis Techniques

Shall I refuse my dinner because I do not fully understand the process of digestion? ~ Oliver Heaviside

As explained in Chapter **??**, an IceCube event consists of a series of DOM hits. One hit contains charge, timing and positional information. The collection of multiple hits has to be translated (reconstructed), to certain properties of a particle. Some examples are the direction, position, track length, etc. Fast reconstructions of these properties, such as the ones necessary in online filtering, are usually done with simple algorithms. More sophisticated algorithms are performed offline on smaller data samples.

Because the in-ice IceCube detector is sparsely distributed, it is not straightforward to unambiguously reconstruct the particle (interactions). The scattering and absorption of photons, tilt of ice sheets, bubble column, etc. lead to uncertainties and make reconstruction challenging. Over the years, multiple reconstruction methods have been developed in the collaboration. Several reconstruction algorithms have been used in this analysis and are explained in more detail in this chapter. We start with the algorithms that reconstruct where the particle entered and traversed the detector, that reconstruct the direction of the particle estimate the direction uncertainty.

A number of hits in an event can often be attributed to DOM noise. There are several techniques possible to remove these hits, some of which are used in this analysis. They are also discussed in more detail below.

Finally, we discuss the methods that were used to discriminate signal events from background events with machine learning techniques.

1.1 Reconstruction

1.1.1 Likelihood

Reconstruction algorithms usually have no unique solutions to describe the set of measured values of an event. The likelihood $\mathcal{L}(\vec{x}|\vec{a})$ describes the probability of a set of parameters $\vec{a}$ to be expressed in a set of experimentally measured values $\vec{x}$. The parameters, $\vec{a}$, typically define the particle's characteristics (energy, direction, position, type, etc.) while the measured values $\vec{x}$ are determined from the detector response (number of PE, timing, position of hit DOMs, etc.). This likelihood is equal to the cumulative probability

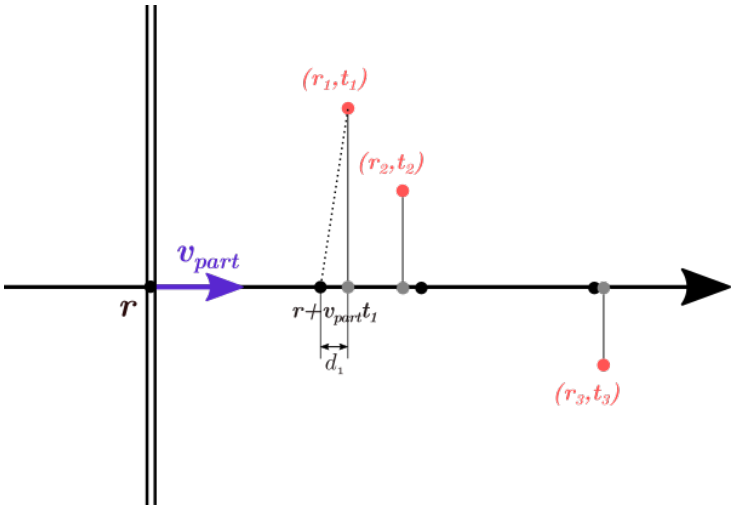


Figure 1.1: Figure illustrating how `LineFit` works. The position, $\vec{r}$, and velocity, $\vec{v}_{\text{part}}$ minimizing the distance of the DOMs to the track is calculated. The dotted line is one of the distances that is minimized in Eq. 1.4.

$$\mathcal{L}(\vec{x}|\vec{a}) = \prod_i p(x_i|\vec{a}), \tag{1.1}$$

where $p(x, \vec{a})$ is the probability that we measure a certain value x from a set of independent values $\vec{x}$ given an initial set of parameters $\vec{a}$. The maximum likelihood method is used to estimate the unknown parameters $\vec{a}$. This is done by maximizing the likelihood $\mathcal{L}$. The reconstruction algorithms below rely on analyzing parameters that assume a single, long track

$$\vec{a} = (\vec{r_0}, t_0, \hat{\vec{p}}, E_0), \tag{1.2}$$

where $\vec{r_0}$ is the position vector of the particle at a time t_0 with a direction $\hat{\vec{p}}$ and initial energy E_0.

1.1.2 `Line-Fit`

One of the simplest approaches in constructing a parameter profile is by calculating the track that, overall, has the closest approach of all the hit optical modules. This is called `Line-Fit` (LF) [1]. If we assume that a particle starts at a position $\vec{r}$ at a time 0 and travels at a velocity of $\vec{v}_{\text{part}}$, then its position at any given time is

$$\vec{r'} = \vec{r} + \vec{v}_{\text{part}} t. \tag{1.3}$$

We want to calculate the best possible estimate of the velocity $\vec{v}_{\text{part}}$ and an initial position $\vec{r}$. Each DOM has a known location, $\vec{r}_i$, and measured time of a pulse, t_i. In this algorithm one assumes that a wavefront perpendicular to the particle's direction is traveling along with the particle. If the velocity $\vec{v}_{\text{part}}$ is fixed, then the position of the particle at later times is known (black points in Figure 1.1). However, the Cherenkov wavefront should be set at an angle and because scattering, PMT jitter, noise, etc. are not taken into account, this will not agree with the DOM position projected along the particle path (grey dots). The unknown velocity $\vec{v}_{\text{part}}$ and position $\vec{r}$ are the analytical solutions after minimizing the distances d_i as shown in the figure*

*Minimizing $r_i - r'$ (dotted line in Figure 1.1) is the same as minimizing d.

$$
\begin{aligned}
S(\vec{r}, \vec{v}_{\text{part}}) &\equiv \sum_{i=1}^{N_{\text{hit}}} \rho^2(\vec{r}, \vec{v}_{\text{part}}, \vec{r}_i, t_i) \\
&= \sum_{i=1}^{N_{\text{hit}}} (\vec{r}_i - \vec{r} - \vec{v}_{\text{part}} t_i)^2 \,,
\end{aligned}
\tag{1.4}
$$

where $\rho(\vec{r}, \vec{v}_{\text{part}}, \vec{r}_i, t_i) = |\vec{r}_i - \vec{r} - \vec{v}_{\text{part}} t_i|$ and N_{hit} is the number of pulse hits. The analytical solution by minimizing this equation is equal to

$$
\vec{r} = \langle \vec{r}_i \rangle - \vec{v}_{\text{part}} \langle t_i \rangle \quad \text{and} \quad \vec{v}_{\text{part}} = \frac{\langle \vec{r}_i t_i \rangle - \langle \vec{r}_i \rangle \langle t_i \rangle}{\langle t_i^2 \rangle - \langle t_i \rangle^2},
\tag{1.5}
$$

where $\langle x \rangle$ denotes the average of a parameter x over all hits i.

Because this is an analytical equation, this algorithm is very fast and therefore often used in online processing.

1.1.2.1 `Improved Line-Fit`

`Line-Fit` is often used as a seed track for other, more computationally heavy algorithms (such as SPE, see Section 1.1.3). The simplifications that are used in LF can lead to angular deviations that converge to a local minimum instead of the global. The LF algorithm assumes that all hits will be near the track and hits far away from the track enter the least squares computation quadratically. Therefore, far away hits often dominate the reconstruction even though the simple algorithms doesn't account for

1. The Cherenkov emission profile.
2. The scattering effects of the ice medium.
3. Noise hits that occur far from the track.

To reduce the effects of outlier, it was found that a basic filter could identify these scattered hits, and improve accuracy by almost a factor of two by removing them from the dataset. More formally, for each hit h_i that consists of a charge, position and timing, the algorithm looks at all neighboring hits within a neighborhood of radius μ, and if there exists a neighboring hit h_j with a time stamp that is t earlier than h_i, then h_i is considered a scattered hit, and is not used in the basic reconstruction algorithm. Optimal values of μ and t were found to be 156 m and 778 ns by tuning them on simulated muon data [2].

This "delay cleaning" is done by computing a Huberfit on the remaining data points and minimizing

$$
\sum_{i=1}^{N_{\text{hit}}} \phi(\rho(\vec{r}, \vec{v}_{\text{part}}, \vec{r}_i, t_i)),
\tag{1.6}
$$

where ρ is defined in Eq. 1.4 and the Huber penalty function ϕ is defined as

$$
\phi(\rho) \equiv \begin{cases} \rho^2 & \text{if } \rho < \mu \\ \mu\,(2\rho - \mu) & \text{if } \rho \geq \mu \end{cases} .
\tag{1.7}
$$

An example of the Huber penalty function is given in Figure 1.2. Because of the overal performance increase of this method, all LF computations were done with the improved version (although still often referred to as "Line-Fit").

1.1.3 SPE and MPE

A more intricate method of track reconstruction is done by taking the geometrical shape of the Cherenkov cone into account and relying on simulation fits where a seed track is implemented (usually from the fast `Line-Fit` algorithm) [1].

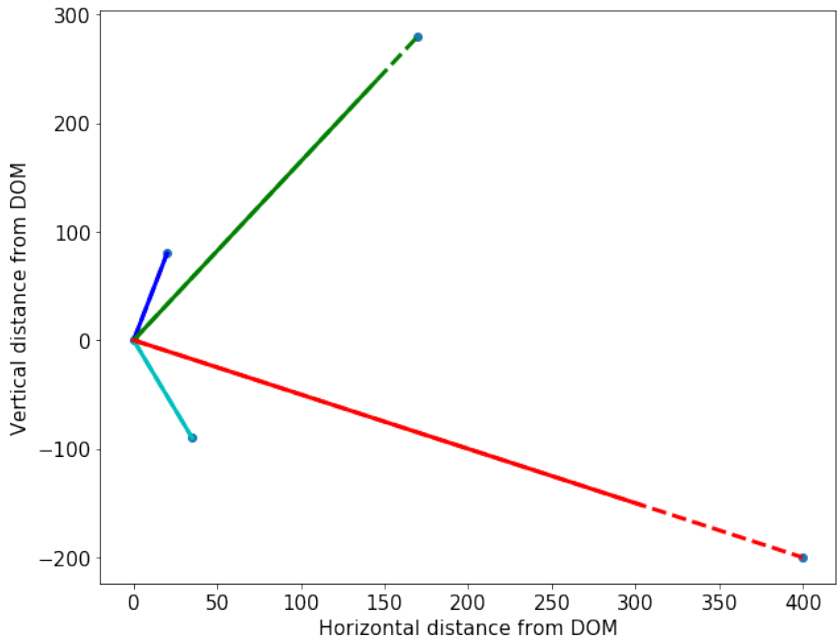


Figure 1.2: Illustration of Huber penalty for possible hit locations. The origin was set as the assumed particle position, $\vec{r} + \vec{v}_{\text{part}} t_i$. The solid line shows the result of $\phi(\rho)$ where the dotted lines show values for ρ.

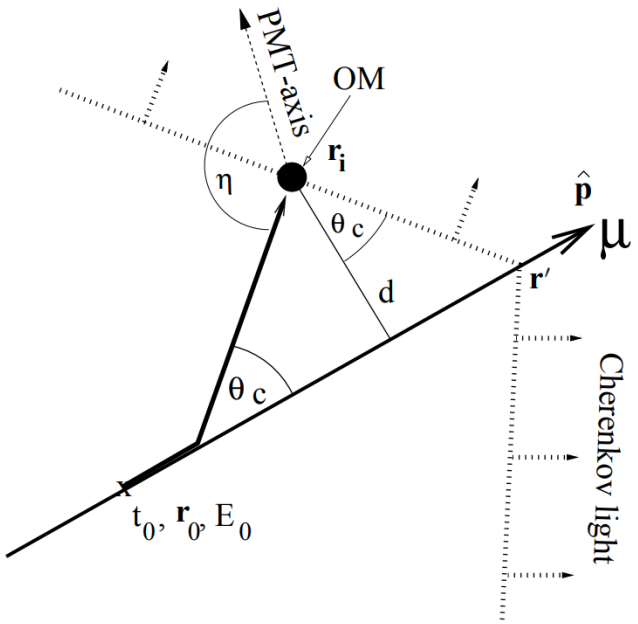


Figure 1.3: Figure illustrating a muon track passing close by an optical module and defining the parameters used in the reconstruction algorithms. Illustration from Ref. [1].

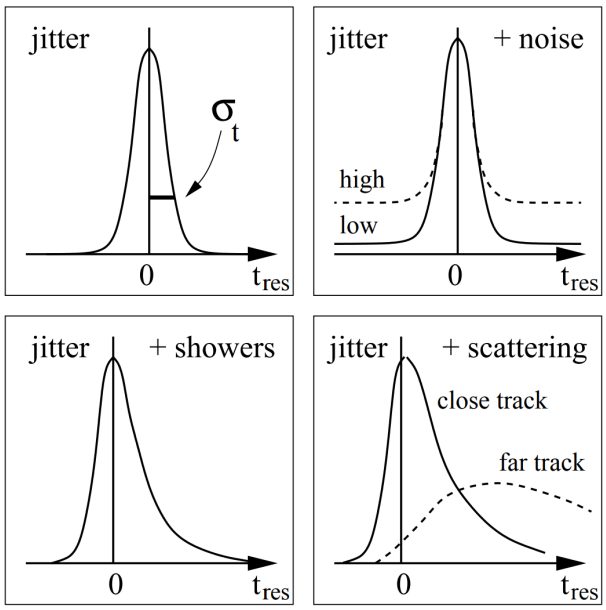


Figure 1.4: Schematic distributions of t_{res} where several effects are included. *Top left:* PMT jitter effects. *Top right:* jitter and random noise effects. *Bottom left:* adding indirect Cherenkov light from stochastic deposits along the track of a high-energy particle. *Bottom right:* adding scattering delay effects. Figure from Ref. [1].

Let us assume a particle is traveling close to a DOM with parameters defined in Eq. 1.2 as illustrated in Figure 1.3. The minimal distance of the track to the DOM is equal to d and the PMT-axis (downwards relative to DOM) has an angle offset of η degrees of the Cherenkov wave direction. If Cherenkov photons would travel undelayed directly from the primary particle to an OM, the *time residual* (time between the observed hit time and the "expected" time) is a delta function centered around 0, where

$$t_{\text{res}} \equiv t_{\text{hit}} - t_{\text{geo}}, \tag{1.8}$$

with

$$t_{\text{geo}} = t_0 + \frac{\vec{p} \cdot (\vec{r}_i - \vec{r}_0) + d \cdot \tan(\theta_c)}{c_{\text{vac}}}, \tag{1.9}$$

which is equal to the time of the particle to travel from the position $\vec{r}_0$ to $\vec{r}'$ as illustrated in the figure. The accompanying Cherenkov wavefront that sent out photons at a time t_0 from $\vec{r}_0$ will cross the DOM when the particle is at a position $\vec{r}'$. Due to noise effects, PMT jitter, light from secondary interactions, DOM orientation, etc. the time residual is smeared and shifted as can be seen in Figure **??**. Stochastic energy deposits along the track of a high-energy particle (as described in Section **??**) lead to a significant shift of the time residual since these photons arrive after the Cherenkov cone.

A time likelihood function $\mathcal{L}_{\text{time}}$ can be constructed from the p.d.f. for arrival times of single photons i, $p1$, at the locations of the hit DOMs. The p.d.f. of single photons was estimated with photon simulations in ice and fitted to a *Pandel function*. How this was done and for more information on this function, I refer the reader to Ref. [1].

$$\mathcal{L}_{\text{time}} = \sum_{i=1}^{N_{\text{hit}}} p_1(t_{\text{res}} | \vec{a} = d_i, \eta_i, ...). \tag{1.10}$$

An initial particle position and direction are found by maximizing the likelihood and iterated a couple of times to find the global maximum instead of a local. This fitting is called the *Single PhotoElectron (SPE) fit*.

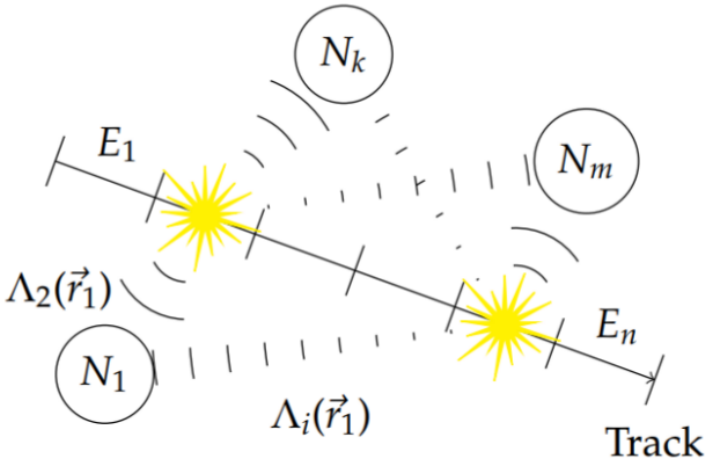


Figure 1.5: Illustration of the working principles of the `Millipede` toolkit. A track is subdivided into segments that each deposit a certain energy, E_i. Different segments can contribute to the total number photons seen per DOM, N_k.

Unfortunately, the description of single photons arriving at the optical modules is not fully correct since electrical and optical signal channels can only resolve multiple photons separated by a few 100 ns and $\approx$ 10 ns, respectively. Within this time window, only the arrival time of the first pulse is recorded.

In the *Multi-PhotoElectron (MPE) fit*, one accounts for the fact that the early photons in a DOM hit scattered less in the ice. The p.d.f. for the first photon out of a total of N to arrive with a time residual of t_{res} is

$$p_N^1(t_{\text{res}}) = N \cdot p_1(t_{\text{res}}) \cdot \left(\int_{t_{\text{res}}}^{\infty} p_1(t)dt \right)^{(N-1)} = N \cdot p_1(t_{\text{res}}) \cdot (1 - P_1(t_{\text{res}}))^{(N-1)}, \tag{1.11}$$

where P_1 is the cumulative distribution of the single photon p.d.f. The MPE likelihood computation is done by replacing p_1 in Eq. 1.10 with p_N^1.

1.1.4 `Millipede`

To have a better handle on the particle energy and stochastic cascades along the track, the module `Millipede` was developed [3, 4]. The number of photons seen at each optical module depends on multiple factors that were mentioned throughout this text, such as the scattering and absorption in the ice, timing jitter, etc. In this module, the expected number of photons is assumed to depend on the energy that was deposited along a track and a *light yield factor*, Λ, that depends on the DOM position and the location of emission

$$\begin{aligned} N_{\text{exp},k} &= \rho_k + \sum_{i=1}^{n} \Lambda(\vec{r}_k, \vec{r}_i')E_i \\ &= \rho_k + \vec{\Lambda}(\vec{r}_k, \vec{r}_i') \cdot \vec{E}, \end{aligned} \tag{1.12}$$

where k refers to a certain DOM and i refers to a certain energy deposit such as illustrated in Figure 1.5. ρ is the average expected number of noise photons and is determined by the duration of the event. The light yield factor and energy are written as vectors to shorten the expressions.

The likelihood is assumed to follow a Poisson distribution with $N_{\text{seen},k}$ the number of hits that occurred and a mean value equal to the number of expected photons, $N_{\text{exp},k}$

$$\mathcal{L}_k = \frac{\left(\vec{\Lambda} \cdot \vec{E} + \rho_k\right)^{N_{\text{seen},k}}}{N_{\text{seen},k}!} e^{-\vec{\Lambda} \cdot \vec{E} - \rho_k}. \tag{1.13}$$

For easier, faster and more accurate computation the logarithm of the likelihood is used

$$\begin{aligned}\ln \mathcal{L}_k &= N_{\text{seen},k} \ln \left(\rho_k + \sum_{i=1}^{n} \Lambda(\vec{r}_k, \vec{r}_i') E_i \right) - \ln \left(N_{\text{seen},k}! \right) - \sum_{i=1}^{n} \Lambda(\vec{r}_k, \vec{r}_i') E_i - \rho_k \\ &= N_{\text{seen},k} \ln \left(\rho_k + \vec{\Lambda}(\vec{r}_k) \cdot \vec{E} \right) - \vec{\Lambda}(\vec{r}_k) \cdot \vec{E} - \rho_k - \ln \left(N_{\text{seen},k}! \right)\end{aligned} \tag{1.14}$$

Assuming that a total of m DOMs registered hits, we can maximize the total likelihood with respect to the energy gives

$$\nabla_{\vec{E}} \ln \mathcal{L} = \nabla_{\vec{E}} \sum_{k=1}^{m} \ln \mathcal{L}_k = \sum_{k=1}^{m} \left(\frac{N_{\text{seen},k} \vec{\Lambda}(\vec{r}_k)}{\vec{\Lambda}(\vec{r}_k) \cdot \vec{E} + \rho_k} - \vec{\Lambda}(\vec{r}_k) \right) = 0. \tag{1.15}$$

This equation holds if all terms in the sum vanish, i.e. if for all DOMs holds that

$$\begin{aligned} N_{\text{seen},k} &= \vec{\Lambda}(\vec{r}_k) \cdot \vec{E} + \rho_k \\ &\overset{\text{Eq.1.12}}{=} N_{\text{exp},k}. \end{aligned} \tag{1.16}$$

This can be written in a set of linear equations

$$\vec{N} - \vec{\rho} = \mathbf{\Lambda} \vec{E}, \tag{1.17}$$

where

$$\mathbf{\Lambda} = \begin{pmatrix} \Lambda(\vec{r}_1, \vec{r}_1') & \cdots & \Lambda(\vec{r}_1, \vec{r}_n') \\ \vdots & \ddots & \cdots \\ \Lambda(\vec{r}_m, \vec{r}_1') & \cdots & \Lambda(\vec{r}_m, \vec{r}_n') \end{pmatrix}, \tag{1.18}$$

is the *response matrix*. This has to be inverted to find the energies in the vector $\vec{E}$. It describes the DOM response to light output from certain segments along a track. The entries in this matrix come from simulations that produce spline tables. Simplified sources, such as minimum ionizing muons and isotropically emitting point sources are simulated in Monte Carlo simulations at certain discrete points. Interpolation is done using spline functions. More information, such as how timing information can be implemented, can be found in Refs. [3, 5].

1.1.5 `FiniteReco`

Charged particles such as muons and electrons can be created from neutrino interactions. Electrons are stopped within a couple tens or hundreds of meters, but muons can travel up to kilometers and do not have a spherical light deposit. Because neutrinos do not produce Cherenkov light, this "sudden" light production from muons can look like a *starting track* in the IceCube detector. Charged particles that travel through matter also lose energy and can therefore be stopped, giving rise to *stopping tracks*. `FiniteReco` is a module that tries to reconstruct if particles are starting, stopping, contained or throughgoing. The hit DOMs around a seed track are checked to have seen light and the first and last emission points along the track are used to check the possible hypotheses. Because SMPs are assumed to create long tracks, the module can be used to remove tracks that start of stop within the detector but far away from the edge.

Because the edges of the detector are not well defined*, the likelihoods of individual DOMs to have seen a hit lead to a total likelihood that doesn't give a conclusive answer, but the starting and stopping probabilities can be compared to a throughgoing track hypothesis.

*Imagine a cascade 20 m below the lowest DOMs. It is still possible for light to reach the bottom modules of the detector.

1.1.6 `Paraboloid`

In Sections 1.1.2 and 1.1.3, we discussed how a particle's direction could be estimated with likelihood estimations. The `Paraboloid` module tries to provide an estimate for the error on this direction. A highly energetic muon with hundreds of hit DOMs will lead to a much better directional resolution than a dim track where only a handful of DOMs are hit. In general, the likelihood space around the estimated direction is scanned and compared to the likelihood of the initial track estimation. If the likelihood space around the minimum is relatively flat, there is a larger uncertainty in the directional estimation. With this method, the module also tries to parametrize if the maximum likelihood found from the reconstruction algorithms is far from or close to the global maximum likelihood. `Paraboloid` constructs a grid of zenith and azimuth points near the minimum and for each point on the grid it does a three-parameter minimization for the vertex holding the zenith and azimuth constant. The likelihood values for each point on the grid are then fit to paraboloid using a χ^2 minimization since the shape of the log-likelihood space near the minimum should have a paraboloid (2D parabola) shape. Of importance are the parameters of the corresponding error, which is assumed to correspond to an ellipse for the 1σ contours.

The module computes the lengths of the semimajor and semiminor axes of the 1σ error ellipse σ_1 and σ_2*. It was found that the quadratic mean of both uncertanties provides a good single-valued estimate for the angle uncertainty

$$\sigma_{\text{para}} = \sqrt{\frac{\sigma_1^2 + \sigma_2^2}{2}}. \tag{1.19}$$

Since σ_1 and σ_2 should follow univariate Gaussian distributions, σ_{para} will be the radius parameter in a bivariate distribution, meaning that the mean should be set to $1.177\sigma_{\text{para}}$†. The variable σ_{para} describes an uncertainty on the direction estimation. But to know if this uncertainty is distributed like described above, one needs to compare the reconstructed direction with the true particle direction. This means, that if we calculate the great circle distance between the MC truth of the signal particle with the reconstruction direction and divide it with σ_{para}, the distribution should peak at 1.177 (mean). This variable is called the *paraboloid pull* and should be compared to an energy related variable (here the number of hit DOMs (NCh) is used). In Figure 1.6, it is clear that there is an offset, which is seen in all analyses, that can be explained by multiple factors in Monte Carlo simulations but mainly stems from our incomplete knowledge and non-perfect simulations of the ice.

More information can be found in Ref. [6].

1.2 Pulse cleaning

As explained in Section **??**, each DOM in IceCube has an intrinsic noise rate. This dark noise is observed in every triggered event and seen as random hits in the detector added to the hit pattern of tracks and cascades. These spurious hits are a large nuisance factor in event reconstructions, leading to misidentification and errors in the result. Noise cleaning should be done in early stages of event processing and analysis to reduce a large rate of bad reconstructed events that pass cut selections. One of the most conservative ways is to only look at HLC hits (*HLC cleaning*), but is too demanding for most low-energetic events that will have multiple hits from isolated DOMs.

Another, more conservative method, is the `seededRT` algorithm. This method relies on the "RT-cut", which was already implemented in the time of AMANDA operations. R is a designed radius and T refers to the time between multiple possible pulse times (e.g. the pulse of one DOM starts during the time window of a second DOM's pulse, or stops during the time window). The

*The confidence intervals σ_θ and σ_ϕ can be found by rotating the minor and minor axes σ_1 and σ_2. How this is done can be found in the literature, but is of no importance here.

†The CDF of the Rayleigh distribution $1 - e^{-x^2/2\sigma^2}$ is equal to the containment for a bivariate normal distribution. Implementing $x = 1.177\sigma$ yields a factor of 0.5.

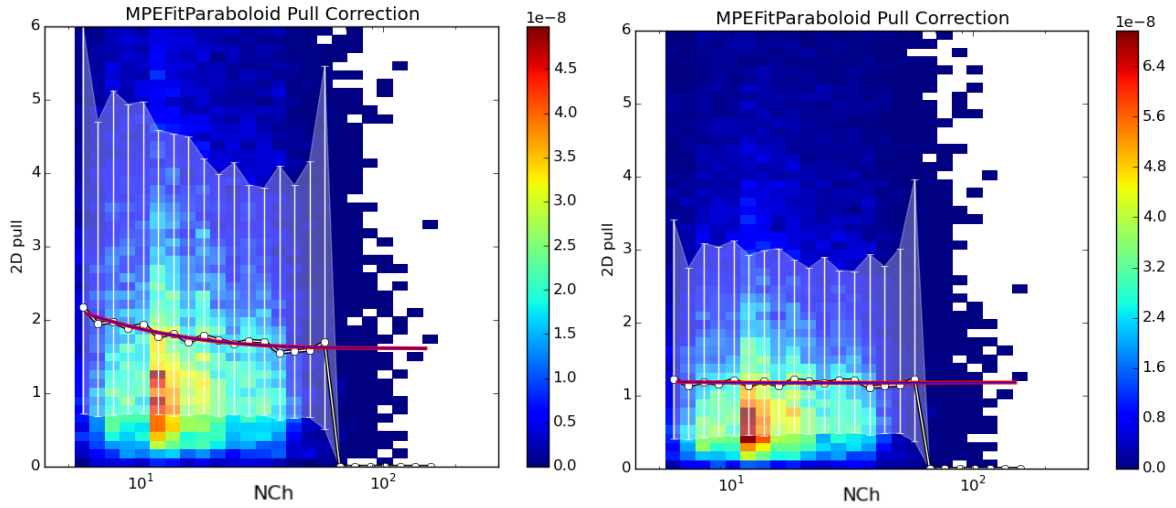


Figure 1.6: *Left:* Paraboloid pull in function of the number of hit DOMs shows that the global average (red/purple line) is not centered at 1.177. *Right:* Pull after correction.

full description can be found in Ref. [7], but can be summarized as follows: DOMs are required to be in a temporal and spacial coincidence that is physically possible (e.g. signal between DOMs cannot exceed the speed of light in vacuum). This method is however computationally expensive since all DOM pairs have to be looped over*. The `seededRT` algorithm takes a subset of seeds that are considered to be mostly signal related hits. These seeds can be provided by, for example, using HLC information. By adding all further hits found within the seed's *RT*-range to the list of seed hits and iterating until a convergence, only those (SLC) hits are kept that cluster around the initial seed hits. Outlying noise hits are supposedly not added and thus removed in the cleaned output. This method does not scale as drastically as the original *RT*-cut method.

1.3 `IceHive`

In Section **??**, it was explained how multiple triggers were combined into one global trigger. In a first step, Q-frames are simply re-split into the individual events that belong to the different subtriggers. In about 15% of cases the data read out in one of these P-frames contains more than one primary interaction. This pile-up effect is referred to as *coincident events*. It is a direct result of the traversal time of a couple of microseconds in the detector†, the large flux of low-energetic events and the trigger time windows of a couple of microseconds. This can be problematic for reconstructions, as can be seen in Figure 1.7 where two downgoing muons can be reconstructed as an upgoing track.

There are two modules that try to clean events more thoroughly than pulse cleaning alone. The first is `TopolocalSplitter` (TS), which starts from the Q-frames and loops over pulses and splits the event into clusters of pulses that contain at least a number of causally‡ connected pulses within a certain time window. Some extra cleaning, similar to `seededRT` cleaning, is done in addition and can split coincident events that have overlapping readout windows, but are geometrically separated.

The second module, and also used in this analysis, is called `IceHive`. A full description can be found in the doctoral thesis of M. Zoll [8]. The module consists of two main parts: one that splits events and handles coincident events, `HiveSplitter`, and another that has a refined pulse cleaning, `HiveCleaning`.

*The number of pairs for n DOMs is equal to $\frac{1}{2}n(n-1)$ and scales with n^2.

†The speed of light in vacuum is equal to ≈ 0.3 m/ns, meaning the particle travels around 100 m in 0.3 μs, without accounting for the delayed photon propagation necessary for detection.

‡The time between two DOM hits cannot be less than the time that light may have taken.

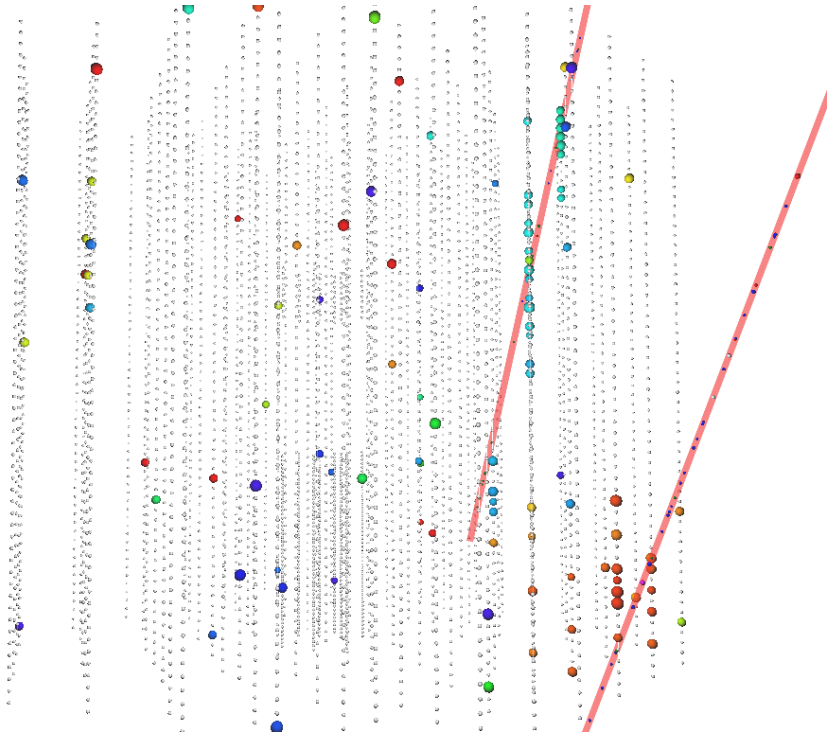


Figure 1.7: Event display of a simulated coincident event of two downgoing muons. The colors of the event range from red (early) to blue (late). The first muon hits the bottom of the detector, while the second traverses mainly the upper part. These events are often reconstructed as a single up-going event and therefore result in a large background contribution. The scattered isolated hits are due to noise effects and mostly removed by pulse cleaning.

1.3.1 `HiveSplitter`

The module assumes that individual particles will create *clusters* of hits in the detector. A cluster can grow within a certain time window, but is separated from another cluster if it's not spatially connected. An initial cluster is formed if the multiplicity of hits exceeds a certain threshold (usually 3 or 4). The main difference in this module versus `TopologicalSplitter` is that it uses hexagons to describe the detector instead of assuming a spherical parameterization. It makes more sense to optimize the search volume, where hits are clustered together, with a shape that describes the detector well and uses a discrete spacing between larger volumes instead of a uniformly growing sphere. The hexagonal shape is set by defining three heights. The first height is defined along the string of the hit DOM and is equal to the vertical distance along the string. The second height is the vertical height along the neighboring strings. The third hight is the vertical height along the next-to-neighboring strings. An example is shown in Figure 1.8.

When the active region is set, there is an additional check to see if DOMs can be "connected". `IceHive` assumes certain emission profiles (for both cascades as tracks) where light is produced. Three possible connections are assumed:

1. Hits occur at the same time, but at a spatial distance in agreement with the Cherenkov emission profile (hits C&1 and 2&3 in Figure 1.8).
2. Hits occur at a different time and a different location, but in agreement with the Cherenkov emission profile (hits C&2 in Figure 1.8).
3. Hits on topologically identical sites of an emission pattern that has moved along with the propagation of the particle (hits C&3 and hits 1&2 on Figure 1.8).

These clusters are finally separated into different P-frames and thus regarded as distinct events.

1.3.2 `HiveCleaning`

Additionally, a similar cleaning as explained in Section 1.2 can be performed. Isolated hits that do not have neighboring hits occurring within a certain distance and time window, are

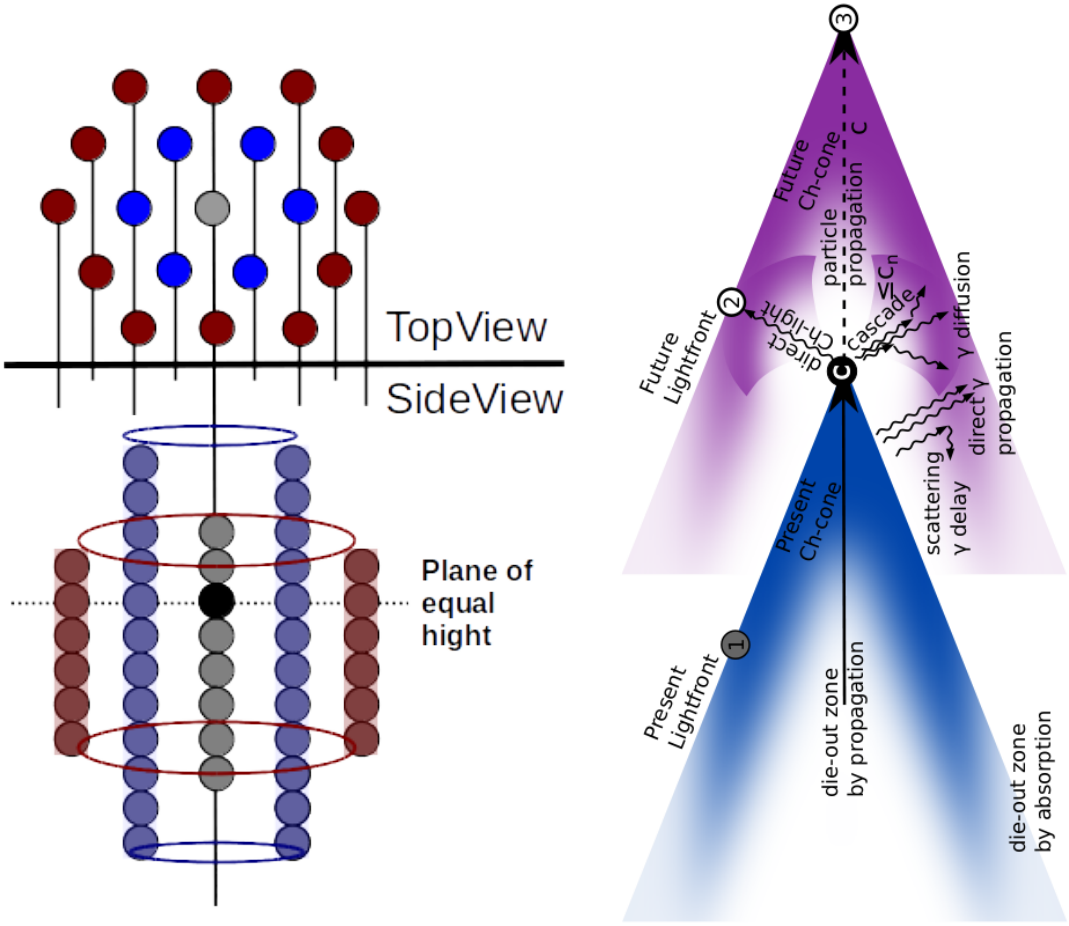


Figure 1.8: *Left:* The black circle illustrates a DOM that triggered a hit in the detector. The grey circles symbolize the DOMs along the string of the hit DOM. The number of DOMs that can be included in the active volume depends on the height defined by the module. The blue/purple DOMs belong to the neighboring strings and the red/brown DOMs to the next-to-neighboring strings. The heights of both these sets of DOMs are also set by the module. This example shows $h_2 > h_1 > h_3$, the heights are also asymmetric in this example. *Right:* Illustration of Cherenkov emission profile of a traversing particle. Both figures from Ref. [8].

removed. The main difference between this and `seededRT` cleaning is that the module again uses the hexagons as defined in the previous section.

1.3.3 Remark

The usage of `IceHive` has a great performance in separating coincident events, but often "overperforms" and splits clusters of hits that are originating from the same particle. This is predominantly the case for dim tracks that have large separations in between clusters (most of the triggered SMP events are of this type). It is because of this that the module `CoincSuite` was designed.

1.4 CoincSuite

Several testing algorithms allow to check if two or more split P-frames can originate from a single event. Five different scenarios were tested in this analysis, the first four are also chronologically shown in Figure 1.9:

1. Cluster alignment: the reconstructed direction of the individual clusters is compared to the direction of a reconstruction that uses the combined hits (HypoFit). The directions

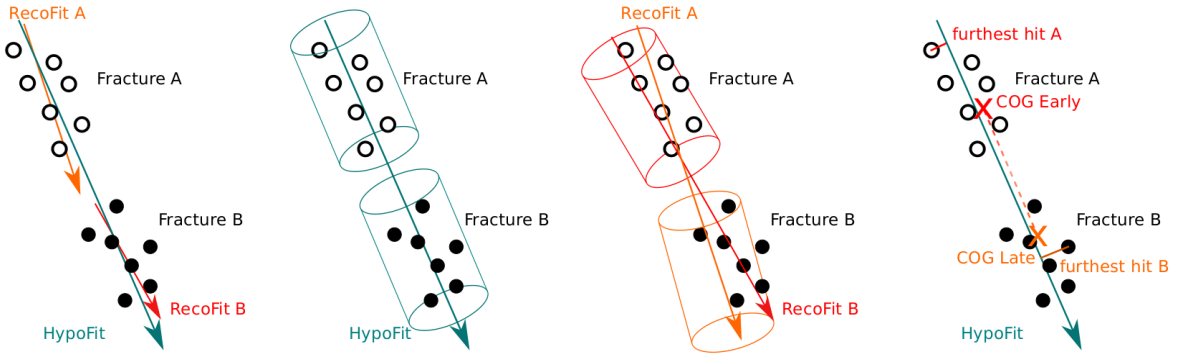


Figure 1.9: Schematic illustrations of possible recombination scenarios with the `CoincSuite` module. Two events are compared to each other or the combined event (HypoFit) in several different ways.

should be within a certain criticle angle.
2. Cylinder cluster containment: the DOMs of the individual clusters should be able to be grouped together in a cylinder that has its center and direction along the HypoFit.
3. Cylinder cluster alignment: a cylinder around the reconstruction of each cluster is draw. The cylinders should overlap within a certain fraction.
4. COG* connection: the second quarter of the COG of the first cluster and the third quarter of the COG of the second cluster are computed. These COG should lie close enough and have to be in the vicinity of the HypoFit.
5. Velocity test: tests if the velocity of the HypoFit is close to the speed of light.

The combination of `IceHive`, which does a very good job in cleaning events, but often overperformes and splits events that shouldn't be, and `CoincSuite`, which recombines events that were wrongfully split, leads to a very powerful tool to clean events.

1.5 Analysis techniques

Komt hier iets dat je gebruikt in het volgende hoofdstuk maar nog niet hebt uitgelegd?

1.6 Boosted Decision Tree classifiers

Given a certain event with a fixed set of variables that are constructed in an analysis, the question remains if the event is in fact a *signal* or *background* event. One can rely on Monte Carlo simulations to get a handle on the variable distribution in both sets. The most general and still widely used method is to use a cut-and-count approach where a cut is placed on a certain variable that discards events that fail the requirement. A Boosted Decision Tree (BDT) inspects a set of set variables and classifies an event with a score that ranges from -1 to 1. The higher the score, the more an event is regarded as signal-like. How this is done is given in more detail below. Boosted decision trees rely on multiple individual trees. Therefore, we will first explain how a single tree classification works.

1.6.1 Stucture

The goal of a decision tree is to determine if an event is signal- or background like. It uses a tree-like structure where certain selection criteria are used at different nodes as illustrated in Figure 1.10.

A decision tree is a binary tree that places an event into a certain node depending on the selection at a node. The depth of a decision tree can be arbitrarily long, but is determined by a set of criteria defined in Section 1.6.2. An event consists of a certain set of variables

*Similar to COM (Center Of Mass), the COG is a weighted average position of the hit DOMs. DOMs that register more light get a heigher weight.

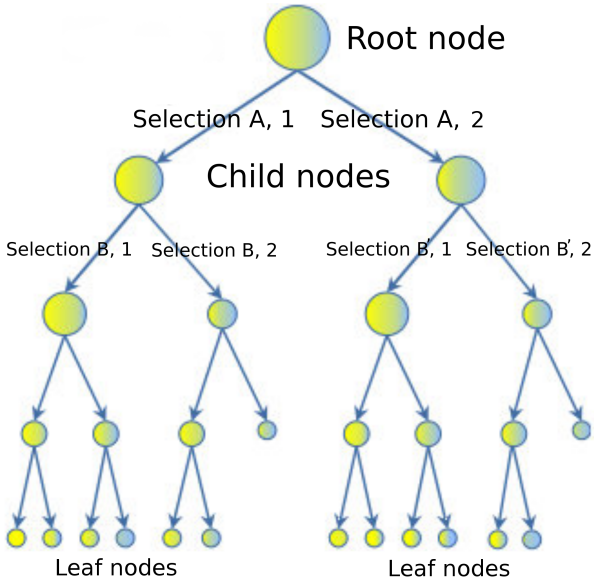


Figure 1.10: Illustration of decision tree scheme. Selection B and B' denote a selection on the same variable, but other requirement.

$X = x_1, x_2, ...x_n$ that are used in the classification. Before any selection criteria, the event is said to be represented by the *root node*. A binary selection then determines to which *child node* the event should be classified, for example:

$$\begin{aligned} \text{Selection A} &= x_1 > y_1 \quad \text{(option 1)} \\ &= x_1 \leq y_1 \quad \text{(option 2)}, \end{aligned} \tag{1.20}$$

where y_1 is the cut value for variable x_1. Similarly, the other selections deterimine where the event is eventually placed. The last nodes are referred to as *leaf nodes* and hold the probabilities of whether an event is more signal- or background-like. These probabilities are translated into a score ranging between -1 (background) and 1 (signal).

1.6.2 Training

To construct a decision tree, one first has to "train" the algorithm. Given a certain "signal set" and "background set", all variables used in the BDT are histogrammed and at each bin for each variable the "best cut" is set at the first node selection. To determine the optimal cut, we first define the purity of a node, p, by

$$p = \frac{\sum_s w_s}{\sum_s w_s + \sum_b w_b}, \tag{1.21}$$

where w_s and w_b refer to the weights of the signal and background events and the Gini index, g,

$$g(p) = p(1-p), \tag{1.22}$$

is used as a separation variable in this work*. Using the Gini index, the separation gain determines the effectiveness of the cut

$$\Delta S = g_p \cdot \sum w_p - \left(g_l \cdot \sum w_l + g_r \cdot \sum w_r \right), \tag{1.23}$$

where g_p and w_p denote the Gini index and weights of the parent nodes and similarly for the left and right child nodes. The cut that gives the highest separation gain is subsequently selected. The algorithm stops when one of the following criteria is met:

*Other possible separation variables include the cross entropy $-p \cdot \ln(p) - (1-p) \cdot \ln(1-p)$ or the misclassification error $1 - \max(p, 1-p)$

- a node only consists of signal or background events;
- a certain predefined maximal depth is reached;
- splitting would cause a child node to have less than a predefined minimal amount of events left;

and therefore determines the size of a tree. These selection criteria are necessary to avoid overtraining (see Section 1.6.4).

1.6.3 Boosting

As already implied in the text above, a BDT consists of a *forest* of decision trees. A user specified number of individual decision trees are trained sequentially, with a boosting process in between each training. Boosting consists of adjusting the weights of individual events according to whether the previously trained tree classifies them correctly. In this work, the AdaBoost* algorithm was used for boosting in which the score of an event is a weighted average of the scores the event receives from each tree in the forest [9].

A BDT may informally be called a "boosted decision tree", but it must be understood that there are actually many trees (typically hundreds), and that boosting is a process that occurs between the training of consecutive trees. The approach makes use of the power of numbers: many weak single decision trees combined can be more powerful than one very good decision tree. In general, boosting follows the following steps:

1. Train a weak model on training data.
2. Compute the error of the model on each training example.
3. Give higher importance to examples on which the model made mistakes.
4. Re-train the model using "importance weighted" training examples.
5. Go back to step 2.

An example is given in Appendix **??**.

If $I(s, y)$ is a function equal to 0 when, after tree classification, the sample test score, s, is equal to its true identity, y, then the error rate for a tree is equal to

$$\epsilon = \frac{\sum_i w_i I(s, y)}{\sum_i w_i}. \tag{1.24}$$

The boosting factor for the tree is defined as

$$\alpha = \beta \cdot \ln\left(\frac{1-\epsilon}{\epsilon}\right), \tag{1.25}$$

with β a user defined *boosting beta* and changes the weight of the tree to

$$\begin{array}{cc} w' = w \cdot \exp(\alpha), & w' = w \cdot \exp(-\alpha) \\ \text{(correct classification)} & \text{(incorrect classification)}, \end{array} \tag{1.26}$$

after which the weights are renormalized so that $\sum w' = 1$. The process is repeated until the number of predefined trees is reached.

Due to its definition, the boost factor α will give good classifiers (which have low error rates) large boost factors. Events that are misclassified are then given larger weights, making the algorithm more likely to classify them correctly in the subsequent tree classifier.

Once the entire BDT is trained, the events can be given a score based on the multiple tree classifiers. This is done by taking the weighted average of all the scores in the individual tree classifiers, using its boost factor α as the weight of the tree. The score of an event i is then given by

*Short for Adaptive Boosting.

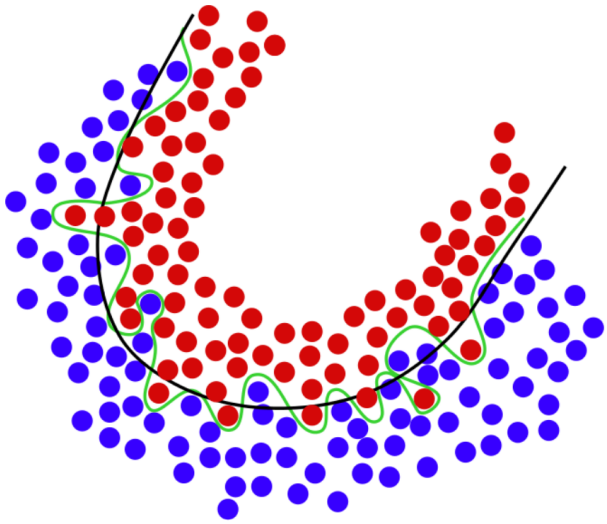


Figure 1.11: Example of overtraining. In black, the theoretical line between background (blue points) and signal (red points) is given. An overtrained BDT will have a (perfect) selection such as the green line. Illustration from [10].

$$s_i = \frac{\sum_m \alpha_m \cdot s_{i,m}}{\sum \alpha_m}, \tag{1.27}$$

where we loop over the individual trees denoted with index m.

1.6.4 Overtraining

BDTs are very powerful tools, but if not used correctly could lead to problems that are not easy to spot at first sight. Assume we train our BDT with a certain signal set and background set. If the BDT is trained up to the point of classifying statistical fluctuations, there is said to be *training sample overtraining*. An illustrative example is given in Figure 1.11. Another example is data/MC overtraining. The former can be dealt with with the use of *pruning*, while the latter should show clear data/MC agreement.

1.6.4.1 Pruning

The problem with overtraining is essentially that there are certain splits in a classifier tree that are too specific and less important. In the method of *cost complexity pruning*, for each node the complexity is calculated as

$$\rho = \frac{\Delta S}{n_{\text{leaves}} - 1}, \tag{1.28}$$

with ΔS the separation gain as defined in Eq. 1.23. The subtree of the node with the smallest value of ρ is removed and this is done repeatedly until a desired *pruning strength** is reached.

BDTs are always trained and tested with different subsets of the same type (background or signal) of event. If the training set has significantly different scores than the testing set, the sample is most probably overtrained and one has to change the BDT input parameters. Typically, one uses Kolmogorov-Smirnov testing to indicate if the BDT score distributions from the training and testing sets could follow the same distribution pattern. As a rule of thumb, a $p_{KS} \lesssim 0.01$ indicates overtraining beyond the level of comfort.

1.6.4.2 Data-MC agreement

Nodig?

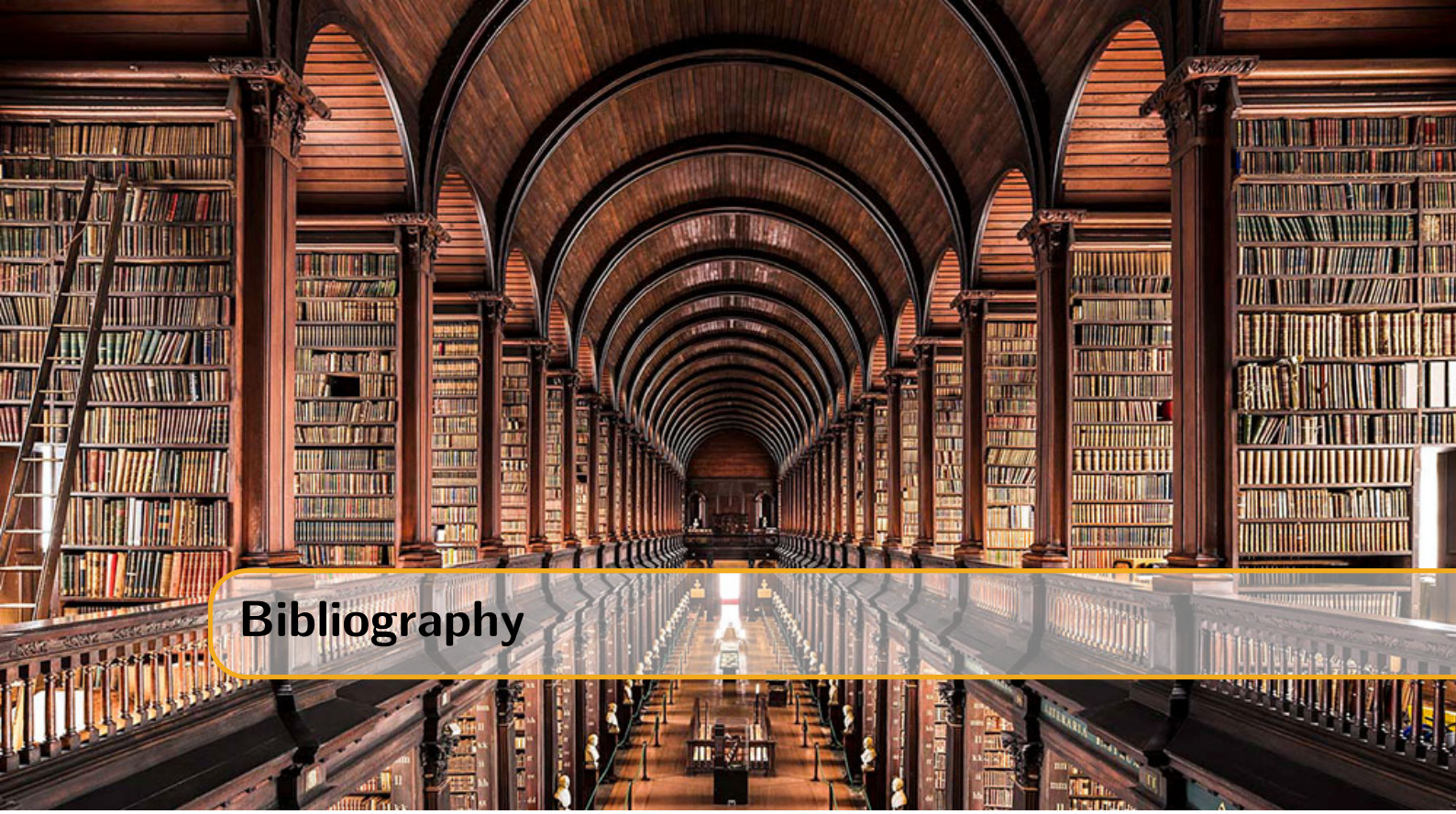
*A parameter on a scale from 0 to 100, which specifies the percentage of the pruning sequence to actually execute. 0 means no pruning is done and 100 signifies only a single root node remaining.

1.7 Minimal-Redundancy-Maximum-Relevance

Having multiple variables that are able to discriminate signal from background events is a neccesary tool to ensure to conclude statements about a certain theory or exotic phenomenon. Single variables can show promising results, but when multiple variables are highly correlated much of the discriminative power diminishes. When using BDTs, analyzers often try to include variables and remove them if they show to be highly correlated in a trial-and-error fashion.

In this analysis, I made use of a technique that was originally developed for data in biological sciences but can be used for most analyses that involve "data mining ". Variables from a large sample set were selected with the condition of minimal-Redundancy-Maximal-Relevance (mRMR). To optimize the characterization of a certain class of events with a set of variables, these variables are selected with a *maximal relevance*. "Relevance" is characterized in terms of correlation of mutual information. Because combinations of individually good features do not necessarily lead to good classification performance, there is the additional requirement of *minimal redundancy* [11].

In this analysis, mRMR was used to rank variables from a large set according to their importance and proved to lead to low correlated variables (see Figure ???).



Bibliography

[1] J. Ahrens et al. “Muon track reconstruction and data selection techniques in AMANDA”. In: *Nucl. Instrum. Meth.* A524 (2004), pages 169–194. DOI: 10.1016/j.nima.2004.01.065. arXiv: astro-ph/0407044 [astro-ph] (cited on pages 4–7).

[2] M. G. Aartsen et al. “Improvement in Fast Particle Track Reconstruction with Robust Statistics”. In: *Nucl. Instrum. Meth.* A736 (2014), pages 143–149. DOI: 10.1016/j.nima.2013.10.074. arXiv: 1308.5501 [astro-ph.IM] (cited on page 5).

[3] IceCube collaboration. URL: https://internal-apps.icecube.wisc.edu/reports/data/icecube/2014/04/001/icecube_201404001_v1.pdf (cited on pages 8, 9).

[4] M. G. Aartsen et al. “Energy Reconstruction Methods in the IceCube Neutrino Telescope”. In: *JINST* 9 (2014), P03009. DOI: 10.1088/1748-0221/9/03/P03009. arXiv: 1311.4767 [physics.ins-det] (cited on page 8).

[5] Stef Verpoest. URL: https://lib.ugent.be/fulltxt/RUG01/002/479/620/RUG01-002479620_2018_0001_AC.pdf (cited on page 9).

[6] Till Neunhoffer. “Estimating the angular resolution of tracks in neutrino telescopes based on a likelihood analysis”. In: *Astropart. Phys.* 25 (2006), pages 220–225. DOI: 10.1016/j.astropartphys.2006.01.002. arXiv: astro-ph/0403367 [astro-ph] (cited on page 10).

[7] URL: https://wiki.icecube.wisc.edu/index.php/SLC_hit_cleaning (cited on page 11).

[8] URL: https://www.dissertations.se/dissertation/fef1f8e4ba/ (cited on pages 11, 13).

[9] Yoav Freund and Robert E Schapire. “A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting”. In: *Journal of Computer and System Sciences* 55.1 (1997), pages 119–139. ISSN: 0022-0000. DOI: https://doi.org/10.1006/jcss.1997.1504. URL: http://www.sciencedirect.com/science/article/pii/S002200009791504X (cited on page 16).

[10] S. Fink C. Böser and S. Röcker. *Introduction to boosted decision trees: A multivariate approach to clasification problems*. Presented at the KSETA Doctoral Workshop, Berlin, Germany. July 2014 (cited on page 17).

[11] Hanchuan Peng, Fuhui Long, and C. Ding. "Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 27.8 (Aug. 2005), pages 1226–1238. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2005.159 (cited on page 18).