Arbeit zur Erlangung des akademischen Grades
Master of Science

Analysis of FACT Data

From Offline Processing to Real Time Excess Rates

Kai Brügge
geboren in Dortmund

July 15, 2016

Lehrstuhl für Experimentelle Physik Vb
Fakultät Physik
Technische Universität Dortmund

Erstgutachter:	Prof. Dr. Dr. Rhode
Zweitgutachter:	Prof. Dr. Westphal
Abgabedatum:	1. September 2016

Abstract

Imaging Atmospheric Cherenkov Telescopes (IACT) like the First G-APD Cherenkov telescope (FACT) produce a continuous flow of data during measurements. (FACT) is dedicated to monitor bright TeV Blazars in the northern sky. FACTs continuous monitoring produces a dense sample of observation points over long periods of time. The usage of silicon photon detectors allows for a larger duty cycle compared to traditional Photo Multiplier Tubes (PMTs). This results in large amounts of collected data. Up to 800 GB of raw data are recorded per night in good weather conditions. To help understand the mechanisms of cosmic ray acceleration, the sources need to be observed over a wide range of wavelengths. In order to coordinate successful multi-wavelength campaigns, other experiments need to be notified quickly in case of flaring sources.

The entire data analysis process including raw data calibration, filtering, image cleaning, parametrization, energy estimation and signal/background separation needs to be performed in order to calculate the flux of observed sources. This puts strong constraints on the data analysis process. To avoid pileup of delays, the analysis software needs to process the data at a rate higher than FACT's mean trigger rate of $50\,\mathrm{Hz}$. For energy estimation and signal/background separation supervised multivariate machine learning models are used. These are trained using labeled FACT raw data produced by Monte Carlo simulations.

In this thesis, pre-trained models are applied to the data written by the telescope data acquisition system to perform effective background suppression in real time. The software is build on top of the `streams`-framework, a modular data streaming environment written in Java. It provides methods for raw data handling, calibration and calculation of image parameters for the FACT telescope. The real time analysis also features a web interface showing live analysis results and the telescope's status during measurements. The system serves as a cross check for an existing "Quick Look Analysis" and has proven to be capable of detecting flaring sources. This will be demonstrated using data from the two bright Blazars Markarian 501 and Markarian 421.

Contents

1 Introduction

When particles of cosmic origin, so-called cosmic rays, hit Earth, they interact with the molecules in the atmosphere. Given enough energy, secondary particles are produced. These can in turn interact again and produce more particles. This process sends a shower of particles towards the Earth's surface. The discovery of cosmic rays and its secondary particles in the atmosphere is attributed to the baloon experiments performed by Viktor Hess in 1912 [39]. Today, air showers can be measured from the surface. When electromagnetically charged particles move at speeds larger than the speed of light in air, they excite molecules in the atmosphere. The molecules then give off their excess energy in the form of Cherenkov light. While the light is too weak to be detected by the naked eye, Imaging Atmospheric Cherenkov Telescopes (IACTs) are sensitive enough to capture the Cherenkov light. The majority of air showers are induced by protons and heavier nuclei. The air showers of interest for IACTs however are the ones induced by gamma rays. These high energy photons can travel undisturbed through electric and magnetic fields. They travel in a straight line from the source to the observer. Measuring the energy and the direction of the photons allows to gain insight into the sources producing high energy gamma rays like active galactic nuclei (AGN) or supernova remnants (SNRs).

The First G-APD Cherenkov Telescope (FACT), is an Imaging Air Cherenkov Telescope located on the Canary Island of La Palma. At the time of its construction in 2011 it was the first IACT that used Silicon Photomultipliers (SiPMs) instead of the conventional Photomultiplier Tubes to detect Cherenkov photons. It serves as a testbed for the SiPM technology in the field of Cherenkov astronomy. A major task for FACT is the continuous monitoring of bright gamma ray sources in the Northern Hemisphere [3]. Among the monitored sources are SNRs like the Crab nebula and AGNs like Markarian 421 and 501. Many sources show periods of higher activity during which more gamma rays can be recorded coming from the source. These periods can last from minutes to weeks and are often called *flares*. In case such a flare is detected, FACT can notify other experiments to monitor the source as well. This allows for observations over large energy ranges. The mechanisms which accelerate cosmic rays and photons are not fully understood yet. A closer observation of the source's variability in both total brightness and spectral behavior can lead to a better understanding of cosmic ray production.

FACT, like all IACTs, records showers induced by both gammas and hadrons. One major task of the data analysis is to separate images created by hadron induced showers, the background, from images recorded from gamma induced showers, the signal.

A full analysis using the raw data from FACT, or any other IACT, usually includes the following steps:

Preprocessing
: Calibrate data and remove artifacts.

Extraction
: Estimate number of photons and their arrival time.

Cleaning
: Select the shower pixels.

Image Parameter
: Calculate parameters for Signal / Background separation.

Classification
: Use ML methods and simulated data to separate signal from background events.

Energy Estimation
: Use regression methods on the parameters to estimate the energy of signal events.

Traditionally many of these steps were performed by separate programs. Over the recent years software called the FACT-Tools was developed to analyze FACT raw data. It is a collection of methods including raw data handling, calibration, image parameter extraction and visualization that can be executed within a single executable. It is built on top of the `streams`-framework [5], a data streaming environment which allows for defining data flow using xml files and executing them in distributed streaming and batch computing environments like Apache Flink [4] or Apache Spark [40].

The last two steps, classification and energy estimation, use models gained from simulated data using supervised machine learning methods and are usually performed by external tools like RapidMiner [31].

The goal of this thesis is to built and run the entire data analysis process within a single `streams` process, in near real time, by online application of pre-trained machine learning models. This will allow for faster responses to flaring sources and a simpler analysis process in general. The new Real Time Analysis will feature a web interface which is served directly from within the self-contained `streams` process keeping the installation process as simple as possible.

After a short introduction to the telescope itself, the next chapters will describe the process of gaining image parameters from raw FACT data. The main focus will be on the calculation of the Hillas parameters since they constitute essential the essential group of image parameters in IACT astronomy and were among the first to be implemented into the FACT-Tools. 0 The machine learning models applied to FACT data and the theory behind are covered in some detail as its a common source of confusion. Models are then build on Monte Carlo data and applied to the FACT data stream using the FACT-Tools software explained in chapters 8 to 9.

2 FACT

Figure 2.1: Photograph of FACT with the Milky Way in the background, courtesy of Miguel Claro [15].

FACT saw its first light in October 2011. It is located on site of the Roque de los Muchachos Observatory on the island of La Palma. FACT's reflector is made up of 30 hexagonal mirrors with a total reflective area of $9.51\,\mathrm{m}$. The telescope is built on top of an old High-Energy-Gamma-Ray Astronomy (HEGRA) mount which was equipped with a new drive system. The camera consists of 1440 SiPM pixels each with a sampling rate of $2\,\mathrm{Gsamples}$. Hexagonal light concentrators are glued to the surface of each SiPM whose surface itself is of quadratic shape. SiPMs are very small when compared to photo multiplier tubes (PMTs). Typical PMTs for this application, like the ones used in the Major Atmospheric Gamma Imaging Cherenkov Telescopes (MAGIC), have a diameter of $25.4\,\mathrm{mm}$ [37]. FACT's hexagonal pixels measure $9.5\,\mathrm{mm}$ across the flat-to-flat distance.

Camera electronics, cooling components, and data acquisition (DAQ) are fitted within the camera housing. Data is transferred via Ethernet cables from the camera to a server. FACT's computers and servers are located right next to the telescope in a standard shipping container. Data is stored on site during measurement and is continuously transferred to the ISDC Data Centre for Astrophysics at the University of Geneva, Switzerland. The telescope can be operated by remote via several web interfaces.The FACT collaboration hopes to achieve full robotic operation in the not too distant future. Many of the necessary features are already in place. An alert system has been tested for a couple of months now. The system can notify the shifter via a phone call whenever the telescope goes into an anomalous state, or a monitored source shows unusual activity [29]. The long-term goal is complete robotic operation.

Since 2011 FACT has recorded approximately 6000 hours of data. Most of the observation time is spent on well-known, and bright, gamma ray sources like the Crab nebula or AGN like Mrk 501 and Mrk 421. Figure 2.2 shows common sources and their observation times.

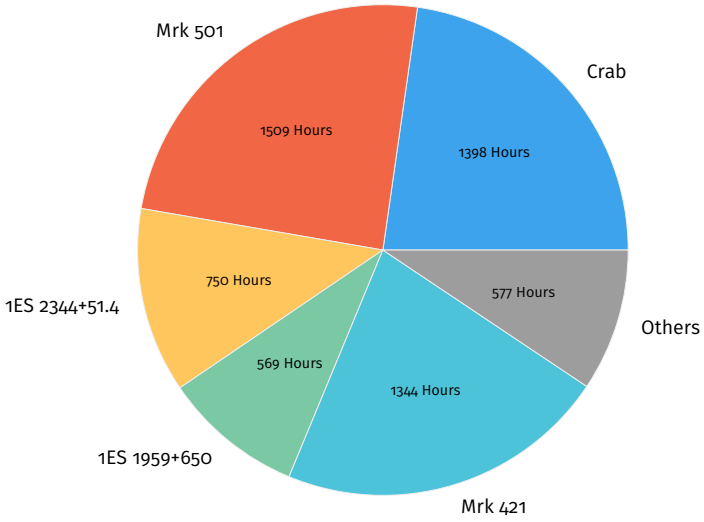

Figure 2.2: Observation times of different FACT sources from October 2011 to June 2016

One of FACT's main goals is the long-term monitoring of bright AGN type gamma ray sources. These sources can show variable behavior in terms of both total brightness and spectral properties. IACTs record so-called light curves which show the gamma ray flux versus time. In case a source shows a sudden increase in flux, it is considered to be in a flaring state. In case a flare is detected, FACT can alert other experiments to trigger multi-wavelengths observations. While IACTs can observe the highest energies on the photon spectrum, radio and satellite experiments usually have a much better energy and angular resolution. Observation over a wide energy range is essential to understand the physics of cosmic ray productions in the observed source. To quickly send flare alerts it is important to calculate the light curve as fast as possible. This is done onsite before the data gets transferred to the data center. The system introduced in this thesis is capable of producing excess rates in near real time while providing a publicly accessible web interface for visualizing the status of the Real Time Analysis.

3 Why Active Galactic Nuclei are Monitored

AGN are the central regions in so-called active galaxies. These regions are extremely bright over many wavelengths, reaching luminosities larger than the rest of the corresponding galaxy [25, ch. 18.7]. The energy required to maintain the observed light output is provided by super-massive black holes in the center of the galaxies. These black holes are estimated to have masses above $10^8\ M_\odot$ with large accretion disks around them. Some black holes produce jets with massive outflows of material parallel to the rotational axis of the accretion disk. AGN whose jets point towards an observer, standing somewhere on earth, are called blazars. They are observed over many wavelengths, combining data from different kinds of experiments into one spectral energy distribution (SED). Figure 3.1 shows an SED of the blazar Mrk 501.

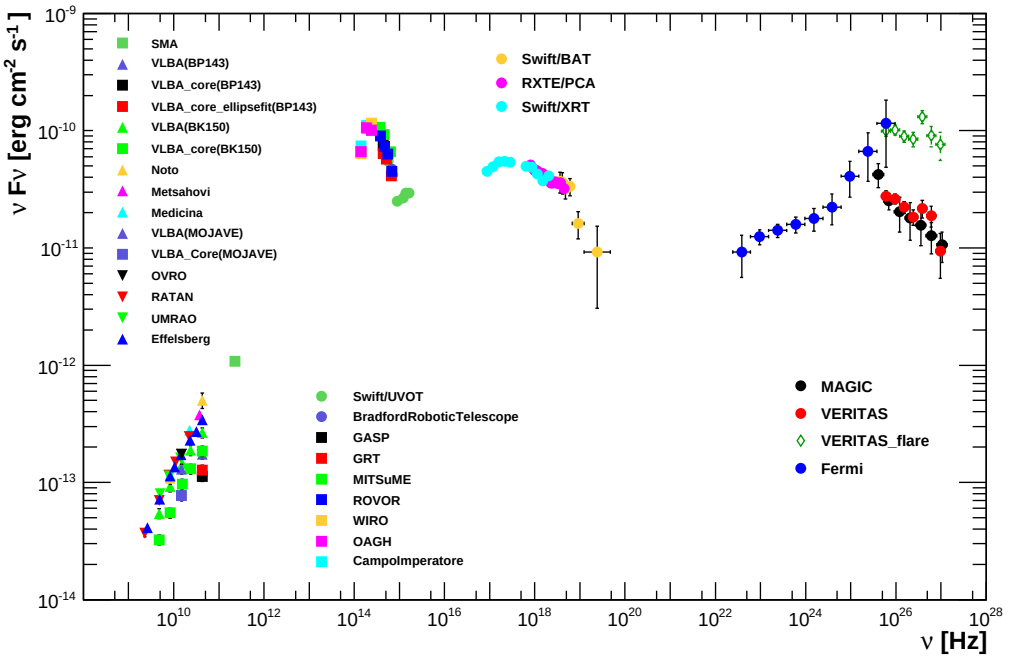

Figure 3.1: SED from the Mrk 501 blazar. All data was recorded between March and August 2009. Taken from [1] as published by the FERMI, MAGIC and VERITAS collaborations.

Blazars typically show two distinct bumps in their SED. Turbulences in the accreting gas account for much of the thermal radiation in the optical and UV wavelengths [35, ch. 10.3.2]. The low energy non-thermal radiation however is due to synchrotron radiation produced by relativistic electrons accelerated by strong magnetic fields in the jet. When the jet coincides with the line of sight of the observer, the synchrotron radiation is relativistically boosted towards the observer to even higher frequencies [8]. There are two popular approaches for explaining the high-energy, X-ray to γ-ray, part of the spectral energy distributions. Leptonic models assume that photons, either synchroton photons in the jet (Synchrotron Self-Compton (SSC)) or photons in an external field (External Compton (EC)), interact with relativistic electrons in the jet in an inverse compton process [7, ch. 8.3]. Through the inverse compton process these photons can gain even more energy. Hadronic models on the other hand assume the acceleration of relativistic protons in the jet. Given enough energy, these protons decay into pions when interacting with photons. The high-energy photon emission from the jet is in part due to Lorentz boosted $\pi^0 \rightarrow \gamma\gamma$ decay and compton and synchrotron emission from π^+, π^- secondaries [8].

Hadronic models could account for the proposed emission of ultra-high-energy (UHE) cosmic rays and neutrinos in AGN. While the origin of UHE cosmic rays near the Greisen–Zatsepin–Kuzmin (GZK) limit is still debated, AGN are among the popular contenders. In 2007 the Auger collaboration showed a correlation between high energy protons and the position of extragalactic objects thought to be active galactic nuclei with distances below $100\,\mathrm{Mpc}$ [2]. Leptonic models on the other hand can explain the high variability observed in AGN SEDs on the order of days or even hours.[1]

Monitoring AGN over wide energy bands is essential for understanding possible acceleration mechanisms of gamma-rays, UHE cosmic rays and extragalactic neutrinos in AGN; consequently, alerting other experiments of flaring sources as fast as possible is an essential part for an IACT like FACT.

[1] Variability of γ-ray emission on time scales of several minutes still poses a problem for simple leptonic models [10].

4 From Raw Data to Image Parameter

The following sections give a rough overview of the raw data processing performed during a typical FACT analysis process. These steps are performed to gain image parameter from the voltage curves recorded by the SiPMs. These image parameters are essential to the machine learning methods needed to separate signal from background events. The calculation of the important image parameters is described in more detail.

4.1 Raw Data Calibration and Filtering

FACT is triggered whenever a certain threshold of signal, in some predefined regions of the camera, is exceeded. Once triggered, a fixed number of samples is stored for each pixel. During normal data taking the number of samples, or Region of Interest (ROI), is set to 300. Each sample has a length of half a nano second due to the sampling rate of 2 GHz. The recorded data is contaminated by several types of noise due to the electronics used in the camera.

The output from the camera's SiPMs is sampled using a Domino Ring Sampler v4 (DRS4) chip. Each sample in the recorded data has a distinct offset which needs to be corrected. Special purpose calibration runs are taken to measure these offsets which can then be used to shift the samples to the proper values.

Every pixel has a fixed gain which has to be taken into account when measuring the amplitude of the signal. These gain constants are also acquired using dedicated calibration runs.

The data contains other forms of noise which are induced by the electronics. The source of these artifacts is not well understood. The artifacts can be filtered from the data using simple heuristics however.

One typical example of a calibrated signal, for a single pixel, is depicted in figure 4.2. All the signals from all of the 1440 pixel are combined into one data unit. One of these units is called *event*. Figure 4.1 shows one of these events as recorded by the FACT camera.

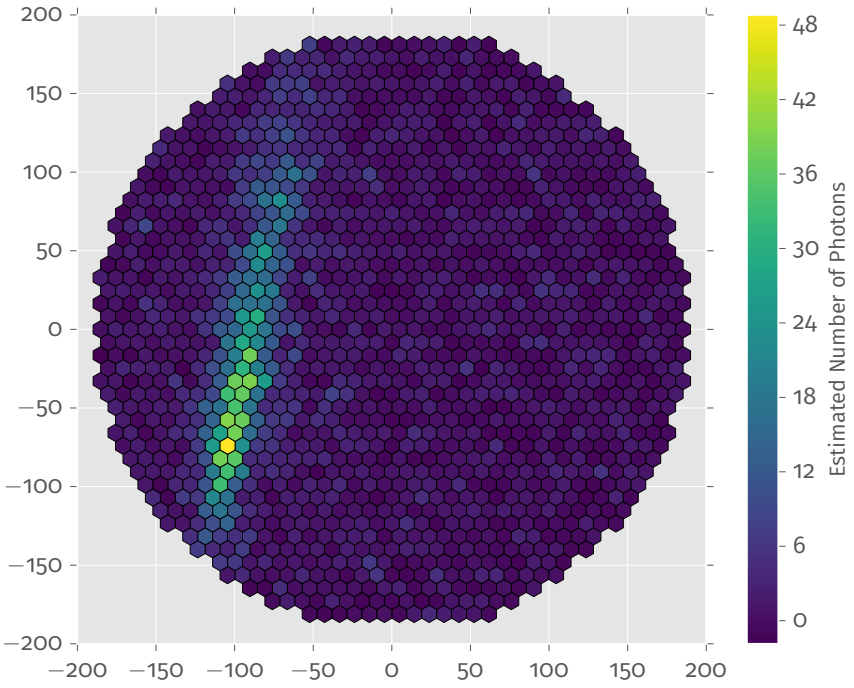

Figure 4.1: A typical FACT event. A shower is clearly visible on the left-hand side of the image.

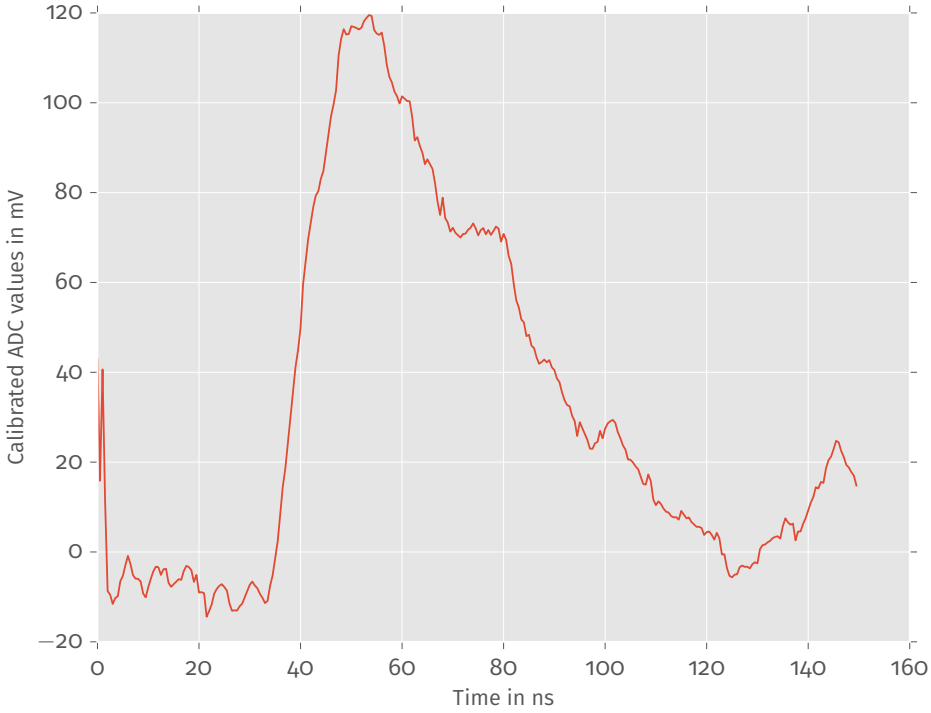

Figure 4.2: The samples recorded by a single pixel. This particular pixel lies on the edge of the shower visible in figure 4.1. There is a clear signal with a peak near the 50 ns mark.

4.2 Signal Extraction

In this step the number of recorded photons per pixel and their mean arrival time are estimated. This process is often called signal extraction. To estimate the number of photons, the time series is integrated over the length of a fixed window. The result is then multiplied by the gain factor of the corresponding pixel.

The arrival time is estimated by finding the rising edge of the signal. To reduce the influence of noise in the signal, a polynom is fitted to the pulses. The inflection point of the polynom is used as the estimator for the arrival time.

4.3 Cleaning

The data recorded by the camera contains a high amount of noise. Some noise originates in background light due to stars or other diffuse light hitting the mirror. Other noise is produced by the sensor itself or by the camera electronics. The noise creates unwanted background in the data. The process of selecting only pixels which have been hit by Cherenkov light is called *cleaning*. From these selected pixels, the shower pixels, the main image parameters are calculated. The other pixels are discarded after this step in the analysis.

4.4 Geometric Image Parameters

In his acclaimed paper from 1985 Hillas [21] used Monte Carlo simulations to find parameters which allow for the separation between gamma induced air showers and proton induced ones. Selecting signal pixels, as described in the previous step, results in an optical image of the air shower. The shape of that image can be used to distinguish between different types of showers. Hillas proposed to approximate the shape by an ellipse and use its geometric parameters for separation. The Hillas features were among the first implemented in the FACT-Tools software. The calculation of the ellipse is an integral part of any IACT analysis software and is worked out in some more detail in this section.

Let $\boldsymbol{p}$ be a two-dimensional vector of random variables.

$$\boldsymbol{p} = \begin{pmatrix} X \\ Y \end{pmatrix}$$

Let $\mathrm{E}(X)$ be the expected value of variable X and

$$\mathrm{Var}(X) = \mathrm{E}\left((X - \mathrm{E}(X))(X - \mathrm{E}(X))\right)$$

its variance. The covariance between two one-dimensional random variables is defined as

$$\operatorname{cov}(X, Y) = \mathrm{E}\left((X - \mathrm{E}(X))(Y - \mathrm{E}(Y))\right) \tag{4.1}$$

Then their covariance matrix is defined as

$$\mathrm{COV}\left(\boldsymbol{p}\right) = \begin{bmatrix} \operatorname{Var}(X) & \operatorname{cov}(X, Y) \\ \operatorname{cov}(Y, X) & \operatorname{Var}(Y) \end{bmatrix}. \tag{4.2}$$

Since all entries in the matrix are positive and the matrix is always symmetric, all of its eigenvalues will be positive as well. The eigenvalue decomposition of a covariance matrix yields a set of orthogonal vectors of which the vector belonging to the largest eigenvalue points into the direction of the largest variance. These vectors are often called the *principal components* of the covariance matrix. The eigenvalues are the variances of the distributions along these directions. Figure 4.3 shows a 2D normal distribution and its principal components. This process is also known as Principal Component Analysis (PCA)

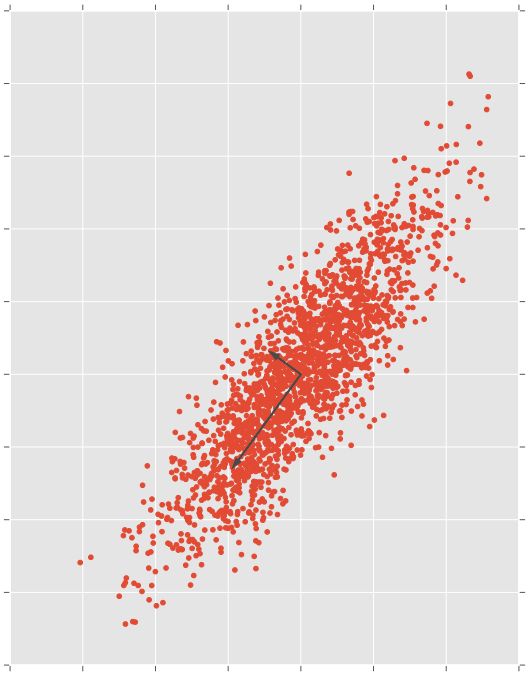

Figure 4.3: A sample of 2000 points from a 2D Gaussian distribution with eigenvectors, or principal components, of its covariance matrix.

and has a large number of real world applications. The calculation of the Hillas parameters is also based on this idea.

Assume the Cherenkov light hitting the camera plane is distributed according to some unspecified two-dimensional distribution $\boldsymbol{p}$. The previously selected pixels are samples

from the vector of random variables $\boldsymbol{p}$. However since the pixels are finite in size and gather multiple Cherenkov photons this is a *binned* measurement of the Cherenkov light that hit the camera surface. For each pixel $p \in P$ with coordinates x_p and y_p use w_p as weight to estimate the variance of the original Cherenkov light distribution. To estimate the expected value of P the *weighted sample mean* is used. It is defined as

$$\bar{x}_w = \frac{1}{W} \sum_{p \in S} w_p x_p \tag{4.3}$$

where W is the sum of all weights $W = \sum_{p \in P} w_p$. The *weighted sample variance and covariance* can be defined accordingly as

$$\mathrm{Var}(x)_w = \frac{1}{W} \sum_{p \in S} w_p (x_p - \bar{x}_w)^2 \tag{4.4}$$

and

$$\mathrm{cov}(x, y)_w = \frac{1}{W} \sum_{p \in S} w_p (x_p - \bar{x}_w)(y_p - \bar{y}_w). \tag{4.5}$$

The covariance matrix of the light distribution can now be estimated. Figure 4.4 shows a binned measurement of a 2D Gaussian with its real and estimated principal components. It is these principal components which define the shape of the ellipse. The square root of the eigenvalues of the estimated covariance matrix are the image parameters known as width and length. They are the standard deviations along the major axes. Once the eigenvectors are known, higher order moments along these axes can be calculated leading to the parameters known as m3long and m3trans which describe the skewness of the light distribution.

In physical terms, the sum of all weights W is an estimation of the number of Cherenkov photons which reached the camera surface from the air shower. It is an important image parameter, often simply called size, which is especially useful for energy estimation.

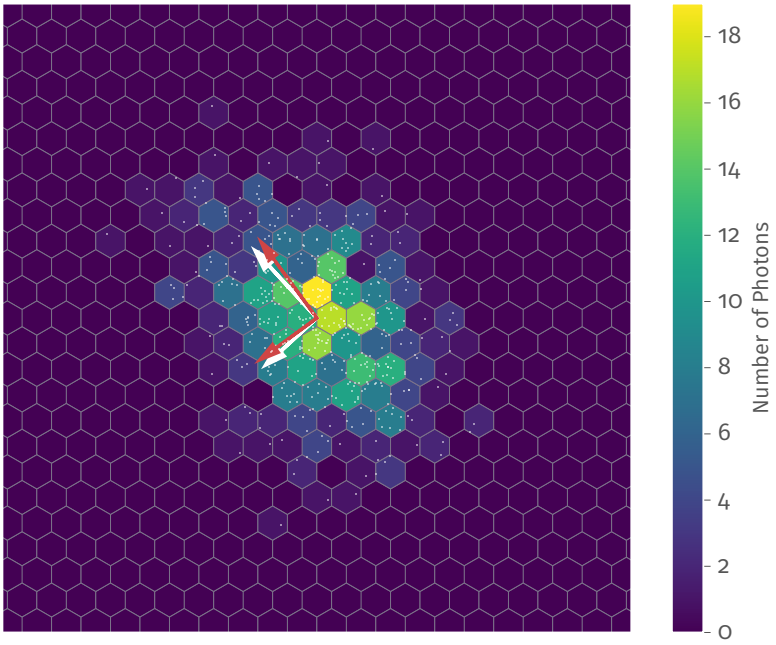

Figure 4.4: The white dots show the location of the simulated photons in the camera plane. The hexagons are the light cones of the camera. Two pairs of eigenvectors are shown in the image. In white are the true eigenvectors from the known covariance matrix used to distribute the photons. Eigenvectors estimated by the sampled photons have a red color.

4.5 Source Dependent Image Parameters

FACT observes point sources of gamma rays. The position of the showers' origin can be reconstructed from the camera image. While background showers, induced by cosmic particles, have no preferred direction and origin, the showers induced by gamma rays have to start at the position of the source in the sky. The background events, showers induced by hadronic particles, have a homogeneous distribution over the sky.

The position of the source in the sky is projected onto the camera plane. The angle and distance of the showers to that point are used to discriminate between showers from the source and others. Figure 4.5 shows the camera image containing a simulated gamma shower. The real source position as projected on the camera surface is marked as a white circle. The white triangle is the source position as reconstructed from the shape of the shower. The reconstructed position is currently calculated by applying a simple parametrization using only geometric properties of the shower width and length. In the IACT community this parameter is called Disp. Theta is the distance between the reconstructed source position and the real one. It is essential for the estimation of background flux in the final dataset as described in section 6.1.1.

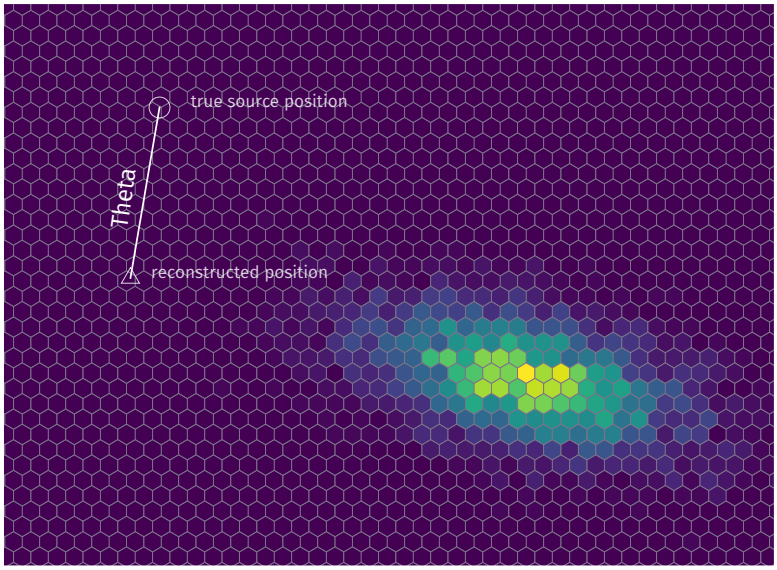

Figure 4.5: The dashed line represents the major axis of the simulated shower. The distance between the shower's center of gravity and the reconstructed source position is called Disp. The space between the reconstructed and the true source position is the most important source dependent parameter. It is often measured in radians or degree by transforming the distance in the camera to the angle of sky covered by that specific span in the camera. Hence the name Theta

More features are calculated on the image data. Describing them all in detail is beyond the scope of this thesis. In total there are approximately 50 image parameters. Some are useful to estimate background noise, others are helpful for monitoring technical properties of the telescope. Only a few are relevant for distinguishing between different shower types. However the methods used for background suppression are capable of selecting these features automatically. They are explained in the next chapter 5.

5 Application of Machine Learning Methods to FACT Data.

Machine learning is used for two major tasks in this analysis: energy estimation and gamma-hadron separation. The first uses the observed values to estimate the energy of the cosmic primary particle which started the air shower. The latter classifies the events by the type of the primary particle. After processing the raw data as described in the previous chapter the data can now be written as a two-dimensional table. Each column represents an image parameter while each event gets written into a single row. This table, or matrix, of observations will be called $\boldsymbol{X}$. The goal of the machine learning tasks is to find a function f which takes $\boldsymbol{X}$ and predicts a label y for each observation x in $\boldsymbol{X}$.

For the gamma-hadron separation task, the label is a categorical variable taking one of two values $y \in \{\text{gamma}, \text{proton}\}$. However for further calculations it will be helpful to encode these values as $y \in \{1, 0\}$. Regarding energy estimation, y is a continuous valued number encoding the energy of the primary particle. The simulated data contains the vector of true labels $\mathbf{Y}$ and can be used to train a statistical model, or decision function, $f(\boldsymbol{X}) = \mathbf{Y}$ yielding the vector of predicted labels $\mathbf{Y}$. Generally any supervised classification or regression method tries to minimize the difference between the estimated label $\mathbf{Y}$ and the true label $\mathbf{Y}$. The method used for FACT data is described in the following section. For a more in-depth mathematical review of machine learning see [19, chapter 2.4].

Table 5.1 shows some example data for a classification problem. The last column encodes the label $\mathbf{Y}$ while columns 1–5 are the data matrix $\boldsymbol{X}$. In the machine learning community one column in $\boldsymbol{X}$ is called *feature*, whereas physicists often call it *parameter*.

Table 5.1: Example data for a classification problem. Randomly selected football matches from the Bundesliga season 2015/16. From the data in columns 1–5 the outcome $\mathbf{Y}$ in column 6 can be predicted. The Location columns displays in which of the competing teams stadiums the game was held. The Temperature and Rain columns describe the (fictitious) weather conditions during the game.

Team 1	Team 2	Location	Temperature	Rain	WinningTeam
Schalke	Werder Bremen	1	5	Yes	2
BVB	Bayer Leverkusen	1	10	No	1
BVB	FSV Mainz	2	12	Yes	1
BVB	Schalke	1	20	No	1
FC Bayern	Borussia MG	2	8	No	2
Augsburg	Schalke	1	14	No	1

5.1 Decision Trees

Decision trees are a common tool for visualizing decision rules and are often used for classification tasks like the one at hand. They formalize the process of making decisions based on some external parameters. Figure 5.1 shows an example for a decision tree for the data in table 5.1.

In a decision tree each node in the tree splits the data into subsets. For simple data sets, the best parameter and split value can be selected by hand. For decision trees the "best" parameter is the one which produces the smallest tree while having the best classification performance. However, for larger and more complicated data this quickly becomes unfeasible. The problem of finding the optimal decision tree is known to be NP-complete [23], however many heuristics exist to find approximate solutions. The most popular algorithms for building trees use a top-down approach. Starting at the root node of the tree, the algorithms select the feature for splitting the data into subsets. For each subset a new node is build where the feature and split for that specific subset is calculated. This process is repeated until some termination criteria is met, e.g. a fixed number of data rows remaining in a leaf node. In many cases the split value is selected according to the *information gain* measure. In the following it will be described in some more detail.

Take any random variable Y over the alphabet Z. In the example from table 5.1 the last column, the vector of true labels, $\mathbf{Y}$ is made up of the realizations of that random variable. The alphabet contains all possible outcomes for Y, i.e. $Z_{\text{football}} = \{1, 2\}$ because either team 1 or team 2 can win. The entropy, or expected information, of a random variable is defined as

$$H(Y) = -\sum_{z \in Z} P(Y = z) \log_2 P(Y = z). \tag{5.1}$$

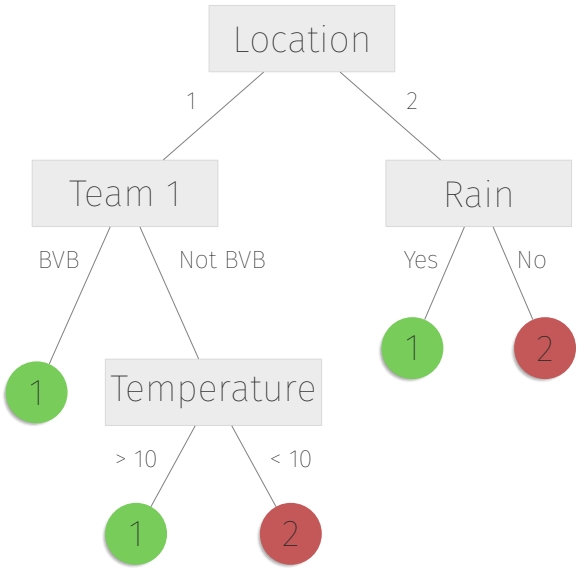

Figure 5.1: A decision tree for predicting the outcome of football games. The round nodes, or leaves, of the tree display the winning team. The tree is obviously not the optimal tree for describing the data in Table 5.1. One can easily build a smaller one by choosing Team 1 as the root node since BVB is always the better team.

Given a large set of simulated data the probability $P(Y = z)$ can simply be estimated by counting the number of occurrences where Y takes the value z.

$$P(Y = z) = \frac{|\{y \in Y \mid y = z\}|}{|Y|}$$

In a similar fashion we can define $P(Y = z \mid X = m)$, the probability of $Y = z$ under the condition that the parameter X takes on the value m

$$P(Y = z|X = m) = \frac{|\{(y, x) \in Y \times X \mid y = z, x = m\}|}{|\{x \in X \mid x = m\}|}$$

hence

$$\sum_{z \in Z} P(Y = z|X = m) = 1.$$

Take the X_{Rain} feature in the football example. The probability that team 1 wins under the condition that it's raining can be stated as

$$P(\text{Team 1 wins} \mid \text{It's raining}) = \frac{\#\text{rainy matches won by team 1}}{\#\text{rainy matches}} = \frac{1}{2}$$

Starting at the root node of the tree the data gets split into subsets. Most decision tree algorithms use binary splitting. The data is split into two subsets, according to a value of a feature, either into the left or right child node, of the root. The new subsets are used to recursively build new nodes in the tree. In our football example the $X_{\text{Team 1}}$ feature can split the data in $\binom{4}{2} = 6$ different ways. When dealing with continuous variables, as is the case for FACT data, it is assumed that there is some natural order to the values. Consider a Parameter X_{cont} having N values n_i, the values can be sorted and then partitioned at $(n_i + n_{i+1})/2$ leaving $N - 1$ possible combinations.

To find the best split in a node, maximize the Information Gain, IG, over all parameters $X \in \boldsymbol{X}$ and splits

$$\text{IG}(X, Y) = H(Y) - H(Y \mid X). \tag{5.2}$$

where $H(Y \mid X)$ is the weighted entropy of the child nodes

$$H(Y \mid X) = \frac{S_{\text{left}}}{S} H_{\text{left}}(Y) + \frac{S_{\text{right}}}{S} H_{\text{right}}(Y). \tag{5.3}$$

with S being the number of data points in the node. In a generalized setting this is equivalent to the conditional entropy

$$\begin{aligned} H(Y \mid X) &= \sum_{m \in M} P(X = m) H(Y \mid X = m) \\ &= -\sum_{m \in M} P(X = m) \sum_{z \in Z} P(Y = z \mid X = m) \log P(Y = z \mid X = m). \end{aligned} \tag{5.4}$$

Because $H(Y)$ is fixed per node, maximizing the information gain is equivalent to minimzing the entropy in the child node.

Popular algorithms for building decision trees are ID3, C4.5, CHAID and CART. They all work with a recursive top-down approach. They differ in their treatment of continuous variables and termination criteria. In this thesis the Python library `scikit-learn` [30] is used for performing machine learning tasks. It uses the CART algorithm as the underlying mechanism to build the trees. [34]

5.1.1 Feature Importance

A feature is considered to be important when it has a high impact on the classification performance of the model. There are several ways of measuring the importance in decision tree. See [33] for a more detailed discussion. In the implementation used in this thesis the importances are measured after the tree has been build. During training, each node saves its selected feature and the corresponding information gain. Importances are estimated by traversing the trees and averaging the information gain per feature.

5.1.2 Regression Trees

Regression trees work in a similar fashion to decision trees. They use a different criterion for splitting the nodes. For each possible split in each node calculate the reduction of variance of the target variable. The split with the largest reduction is chosen.

5.2 Random Forests

Originally proposed by Breiman in his 2001 paper [11], Random Forests have since become a popular choice for classifications tasks for problems across various domains and fields. The algorithm has been used successfully by all major IACT experiments i.e. H.E.S.S, MAGIC and VERITAS.

A Random Forest is an ensemble of decision trees. Each tree is trained on a random bootstrapped sample of the data $\boldsymbol{X}^*$. In contrast to a typical decision tree, only a random selection of parameters is considered for generating splits in each node. Each tree t has a different decision function $f_t(\boldsymbol{X}^*) = \mathbf{Y}_t$. The functions are accumulated to build one decision function for the whole ensemble by first building the average of all N tree responses

$$\bar{f}(\boldsymbol{X}) = \frac{1}{N} \sum_t f_t(\boldsymbol{X}). \tag{5.5}$$

This method of combining learners is called Bagging[19, chapter 8.7]. When encoding the classes as described above $\bar{f}(\boldsymbol{X})$ will yield numbers in the range $[0, 1]$. The decision function of the Random Forest is

$$\mathrm{RF}(\boldsymbol{X}_i) = \begin{cases} 1 & \text{if } \bar{f}(\boldsymbol{X}) > 0.5 \\ 0, & \text{otherwise.} \end{cases} \tag{5.6}$$

In the field of high energy physics the decision function is often extended by another parameter p

$$\mathrm{RF}(\boldsymbol{X}_i, p) = \begin{cases} 1 & \text{if } \bar{f}(\boldsymbol{X}) > p \\ 0, & \text{otherwise.} \end{cases} \tag{5.7}$$

The parameter p is called prediction threshold. It can be used to select only those events for which the classifier gave a high probability. It can be useful for lowering the false positive rate of the decision function.

The prediction threshold and its terminology is a common source of confusion when comparing results and methods from other experiments. Many publications from the MAGIC telescopes use the term *gammaness* for p and *hadroness* for $1 - p$. Others often call the value of p the *confidence cut*. Adapting the prediction threshold is not restricted to the Random Forest method. Other classification schemes that have some notion of classification probability or confidence work as well.[1]

In short: a Random Forest is a bagged ensemble of decision trees where the trees only consider a random selection of features for splitting in each node.

5.3 Performance Evaluation and Cross Validation

Once a model has been trained it needs to be evaluated. Machine learning models are evaluated by applying the models to a set of labeled data. For a binary decision problem, like the one at hand, each row in the test data can have one of four possible outcomes.

True Positives T_p Number of rows which have been correctly labeled as belonging to the positive class.

False Positives F_p Rows which have been falsely labeled positive.

True Negatives T_n The number of rows correctly labeled as negative.

False Negatives F_n Rows falsely labeled as negative.

Collecting these numbers in a matrix leads to the so-called *confusion matrix*.

$$\boldsymbol{C} = \begin{bmatrix} T_p & F_p \\ F_n & T_n \end{bmatrix}. \tag{5.8}$$

To avoid overfitting, the confusion matrix is always calculated on an independent test set, that is a set of data that was not used to train the model. Error estimates on theses values can be gained by splitting the training data into N disjoint test sets. In each iteration the classifier is then trained on $N - 1$ subsets and tested on the remaining one. This leads to N different estimates for the confusion matrix. This method is called cross validation and is widely used for classifier evaluation.

[1] In fact, other experiments sometimes use other tree-based classification schemes. The term Random Forest is often somewhat carelessly applied to these as well.

6 Signal/Background Separation

The air showers used for the simulation of FACT data are created using a program called COsmic Ray SImulations for KAscade (CORSIKA) [20]. It is used by most, if not all, experiments studying extensive air showers. It simulates all secondaries and the propagation of Cherenkov photons to the ground. The telescope itself, optics and electronics, is simulated using Camera Electronics and REflectorSimulation (CERES), a programm specifically designed to simulate the FACT hardware. CERES [12] creates FITS files in a format similar to those written by the telescope itself. Just like the real data, the simulated FITS files are processed by the FACT-Tools to create image parameters for each triggered event. Simulations are performed for gamma ray primaries, the signal class and for hadronic primaries, the background class.

Models are trained using the popular Python machine learning library scikit-learn [30]. It comes with a large number of algorithms for supervised learning, data transformation, clustering and model evaluation. Data handling is performed by the Pandas library [28], using data frames inspired by the R programming language. Visualizations are created by the Matplotlib [22] Python library.

See the configuration files in the appendix B for the exact settings used to select the data and train the models.

When using multivariate methods for signal/background separation it is essential to evaluate the performance characteristics of the models. Better performance can lead to higher detection significance and lower systematic errors for further studies on the selected set. A few essential performance indicators are used to evaluate the models. They are calculated from the entries in the confusion matrix.

Precision
: The fraction of positive events that have been correctly labeled: $\frac{T_p}{T_p+F_p}$.

Recall
: Also called True Positive Rate. The fraction of true positives and all events belonging to the positive class: $\frac{T_p}{T_p+F_n}$.

F_β score
: The harmonic mean of precision and recall: $(1+\beta^2)\frac{\text{precision}\cdot\text{recall}}{\text{precision}+\text{recall}}$.

False positive Rate
: The fraction of false positives and all events from the negative class: $\frac{F_p}{F_p+T_n}$.

Receiver Operating Characteristic

Plotting the true positive rate versus the false positive rate for all prediciton thresholds creates the so-called ROC curve. For a perfect classifier the area under the curve (AUC) would be 1.

Model performance is evaluated on independent test sets during cross-validation. However due to possible mismatches between simulated and real data the crucial test for the model is whether a signal can be seen on real data.

6.1 Performance of Machine Learning Models

The Random Forest was trained using 52 features. A full list is printed in appendix B. Figure 6.1 shows the feature importances as calculated by the Random Forest. The width feature shows a high information gain in the model, which is in line with physical expectations. In general gamma induces air showers show a narrower image in the camera while hadronic showers create more subshowers resulting in a wider shower image.

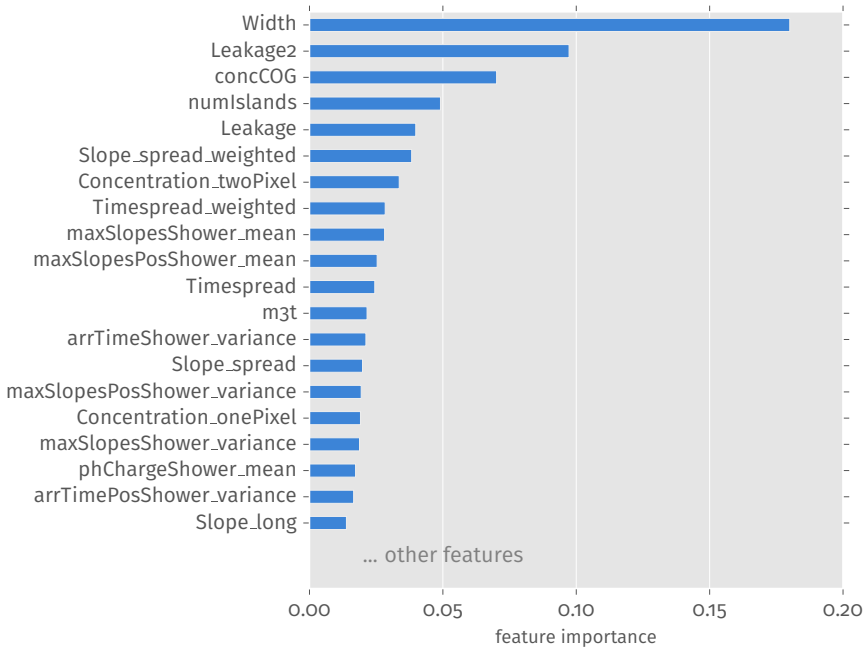

Figure 6.1: Feature importances as estimated by the RandomForest. Width is the most important parameter for this particular model.

In the following figures the integral performance measures are plotted versus the prediction threshold of the model. Figure 6.2 shows how a lower prediciton threshold decreases the recall while the precision increases. The F_β score can show a maximum at a fixed prediction threshold. However figure 6.3 shows no distinct value. The area under the ROC curve seen in figure 6.4 is independent of the chosen prediction threshold and shows a value of 0.73 ± 0.0217 . All values have been calculated in a 5 fold cross-validation. Error bars indicate the standard error of the mean.

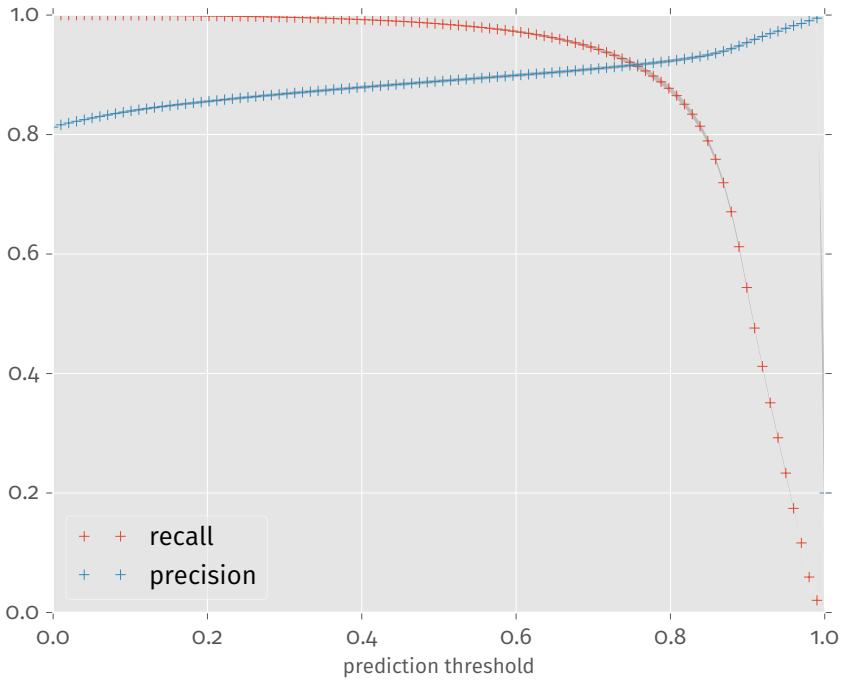

Figure 6.2: Recall and Precision of the model plotted versus the prediction threshold. As expected, the precision increases towards large prediction thresholds while the number of false positives decreases. Recall on the other hand decreases as the number of false negatives increases. The thin gray band indicates the standard deviation of the measurements gained during cross validation.

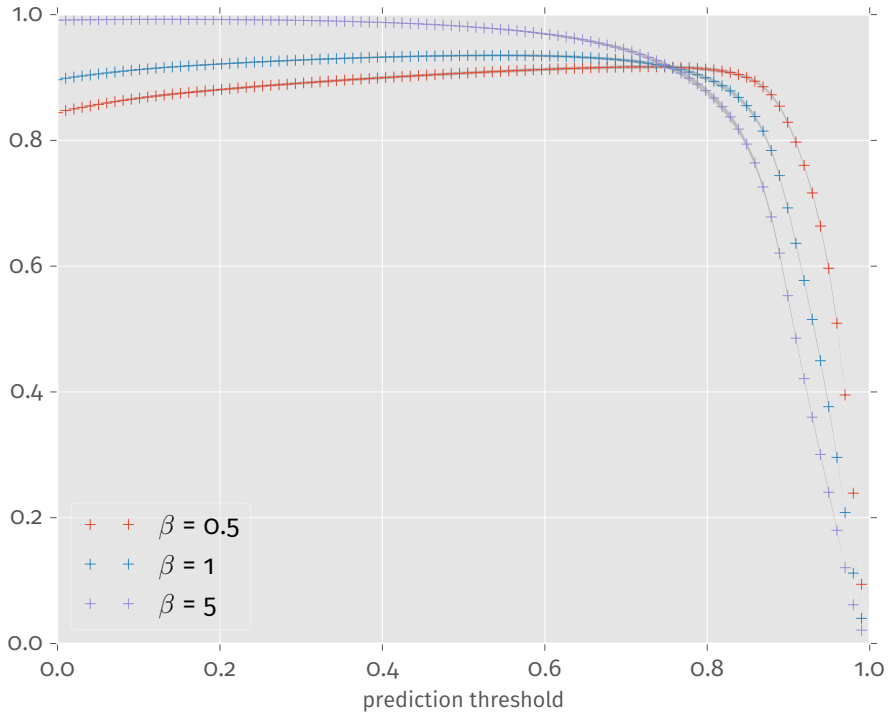

Figure 6.3: The F_β score for the Random Forest. The thin gray band indicates the standard deviation of the measurements gained during cross validation.

6.1.1 Application to Measured Data

Eventhough the multivariate models show good performance, on measurements the selected data will still be contaminated by non-gamma events. This is simply due to the fact, that the flux of cosmic ray showers clearly dominates the flux of gamma ray induced air showers. So even with a small false positive rate of the model, and a high prediction threshold, there are still many falsely labeled background events on real measurements. The amount of these *gamma-like* events which are background can be estimated by using *off measurements*. FACT mostly observes well-known point sources. The position of these sources in the camera image is known during measurements. To estimate the background FACT uses five *off positions* in the camera. During the analysis all source dependent parameters, e.g Theta see section 4.5, are calculated with respect to each of the five positions.

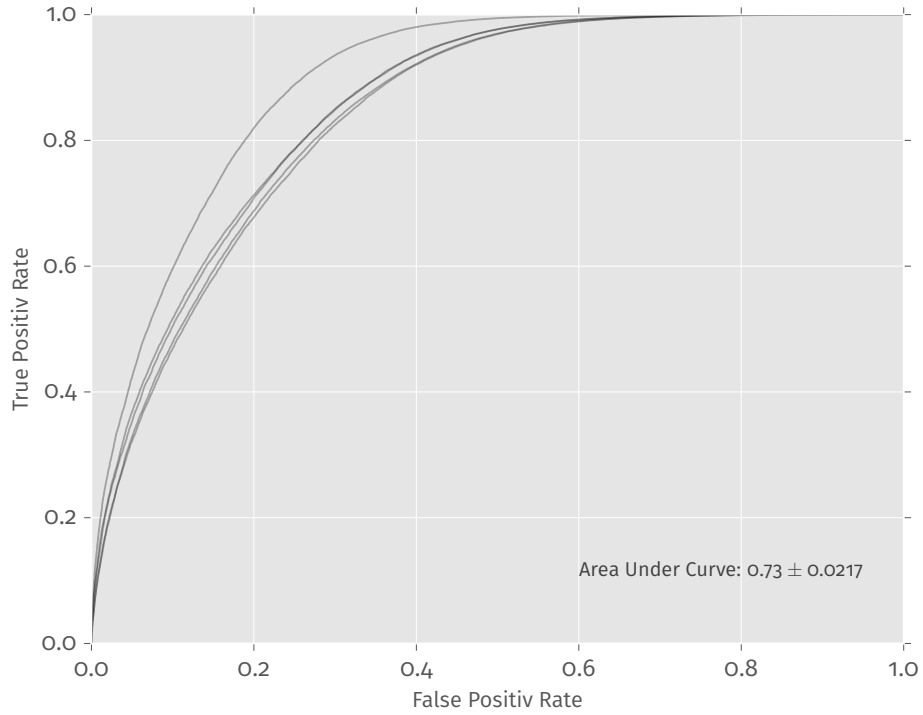

Figure 6.4: Receiver operating characteristic of the models built during cross-validation. Each of the gray lines represent one of the models build during the cross-validation process.

Figure 6.5 shows a histogram over Theta^2 on real data. An excess is clearly visible for events which have been reconstructed near the true source position. When selecting data where Theta^2 is smaller than 0.0193 the statistical significance of the excess is 36.7. For this data a prediction threshold of 0.94 is used. Both the prediciton threshold and the Theta^2 cut are chosen by iterating over all possible combinations and selecting the one where the significance according to Li & Ma [26] is highest. To create this plot approximately 87 hours of raw data were recorded while pointing at the Crab nebula.

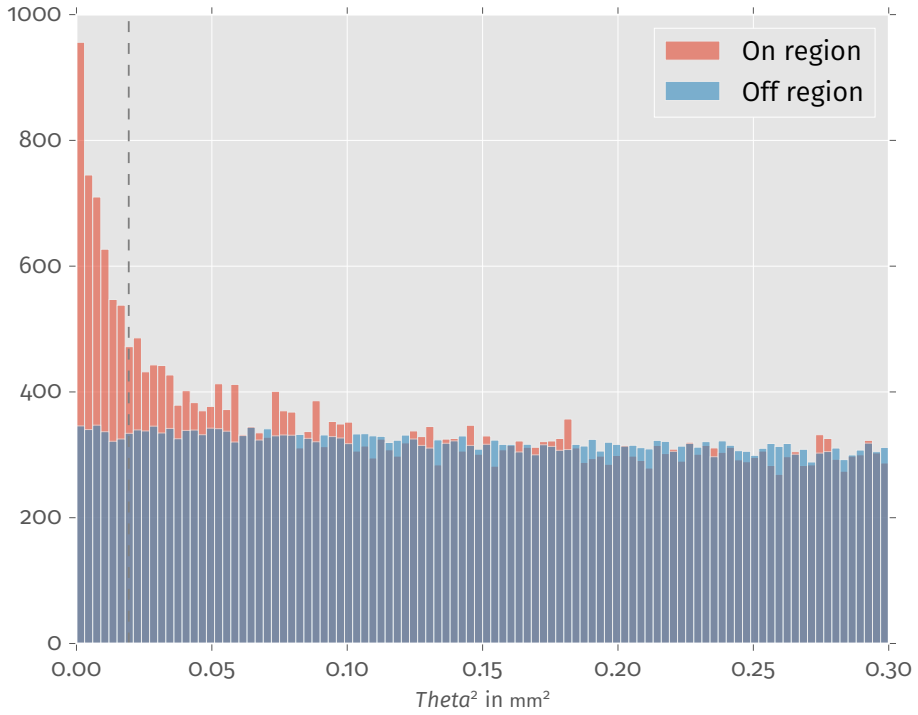

Figure 6.5: Theta^2 plot for a Crab data set containing 87 hours of data. The data was recorded between October 2013 and February 2014. The significance of detection according to Li & Ma [26] is 36.7

6.2 Performance of Source Dependent Machine Learning Models

One alternative to the approach presented above is to add features like Theta and Distance to the training inputs for the machine learning models. As shown in the previous section 6.1.1 these parameters can be used to distinguish signal events from background events to some degreee. To no surprise the quality of the model clearly exceeds that of the source independent models. Precission, Recall, F_β and ROC all show better performance when training on source dependent features. The AUC of the source dependent model is 0.93 ± 0.0058 .

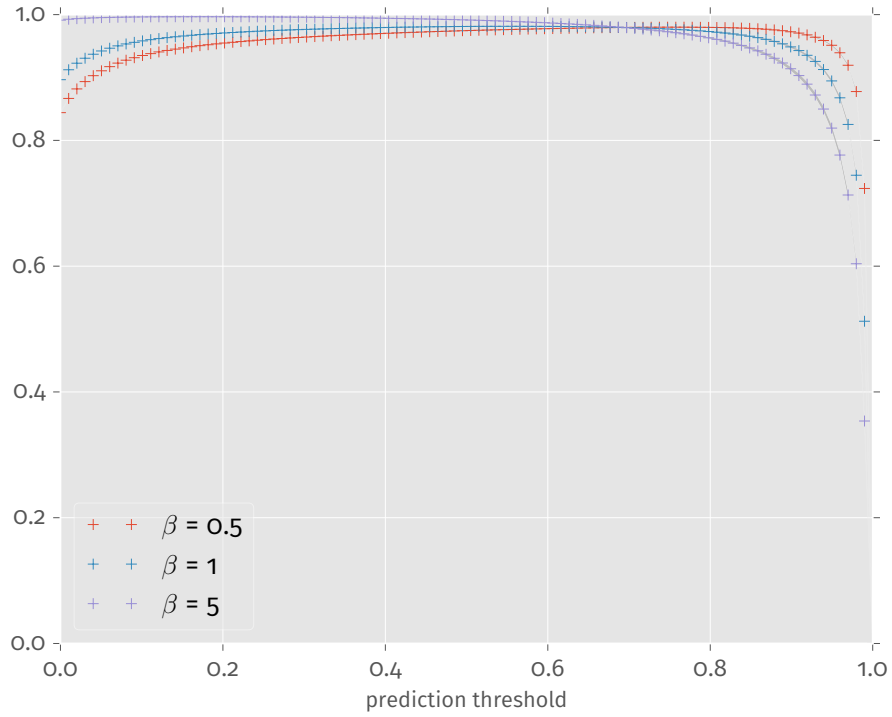

Figure 6.6: The F_β score for the Random Forest. The thin gray band indicates the standard deviation of the measurements gained during cross validation.

In figure 6.9 the feature importances of the Random Forest are shown. The decision trees in the ensemble mostly split the data based on Theta. Its feature importance outweighs the other parameters by a large margin. In the training data, all events are simulated as coming from a single source position, as is the case for real gamma rays from cosmic sources. In case the position has been reconstructed correctly, the simulated signal events all have low Theta values.

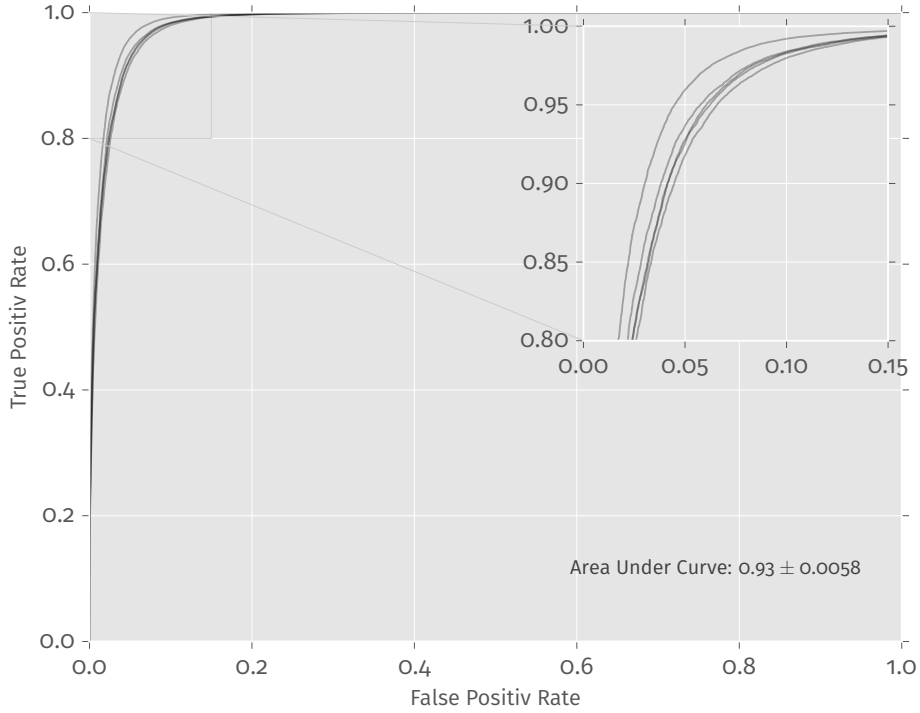

Figure 6.7: The receiver operating characteristic of each model built during cross-validation. Perfomance is certainly much better than the models seen in figure 6.4. The inset highlights the performance of the individual models. The variation in the model's performance shows once again the importance of performing cross-validation.

6.2.1 Application to Measured Data

In order to get a background estimation for a source dependent model, it has to be applied to the data once for each off position. Having five off position means applying the Random Forest five times per event, each time using the Theta and Distance values from the one of the five off positions. Background events are expected to show a uniform distribution in their reconstructed source position. However since there are far more background than signal events there is a high chance of mis-classifiying events as signal when their reconstructed source position is close to the true position. This behaviour of the model is visible in figure 6.10. The background events show a distinct increase towards smaller Theta^2 values when compared to the source independent model from section 6.1.1 where the background histogram is rather flat.

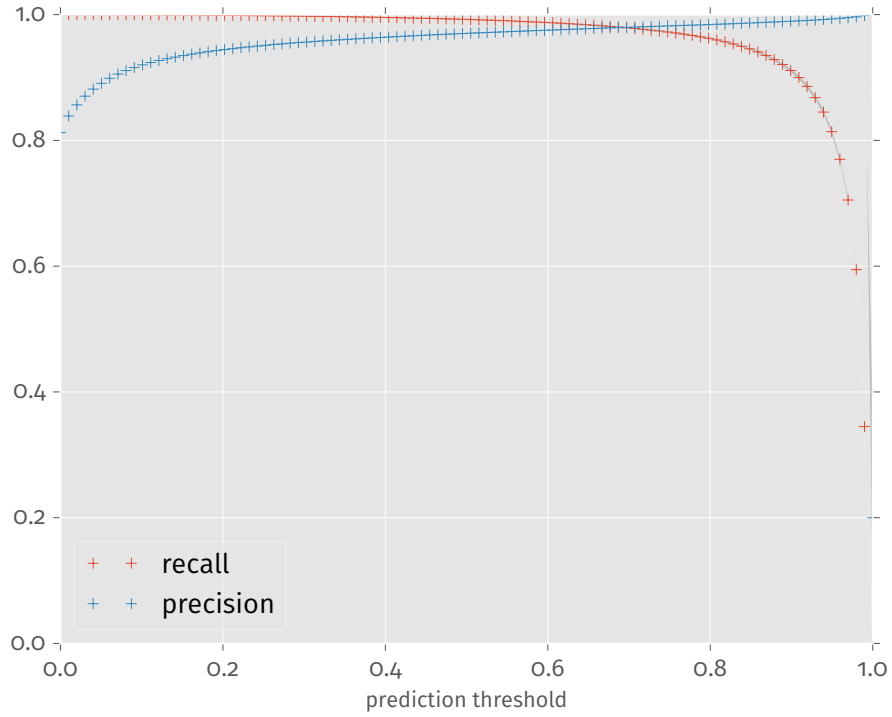

Figure 6.8: Recall and precision versus the prediction threshold of the model. The thin gray band indicates the standard deviation of the measurements gained during cross validation.

Using this model and choosing a Theta^2 cut of 0.0193 with a prediction threshold of 0.99 leads to a signal significance of 33.9.

The performance on real data of the two models does not differ much. However the source independent model shows a flat background distribution in Theta^2 and has to be applied only once. Because the source dependent model mainly splits on one or two features, it can be more prone to overtraining. For application during the real time analysis the source indepenent model is chosen.

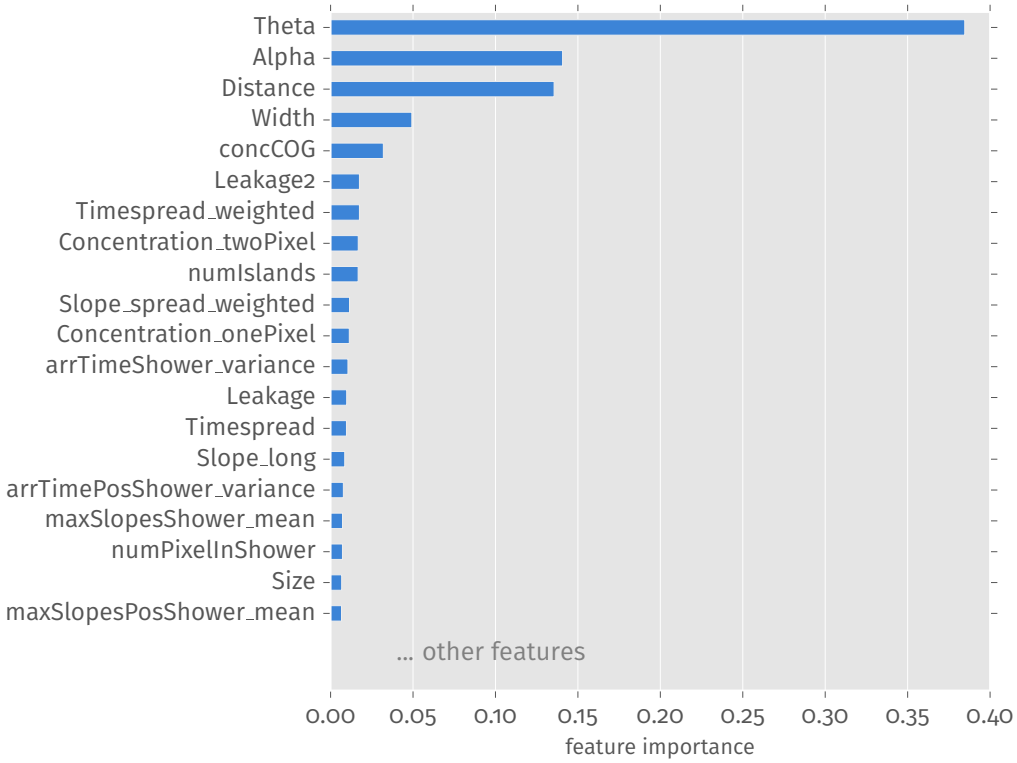

Figure 6.9: Feature importances as estimated by the RandomForest when Theta and Distance are part of the training data. The source dependent features have a large influence on the splitting performed by the decision trees.

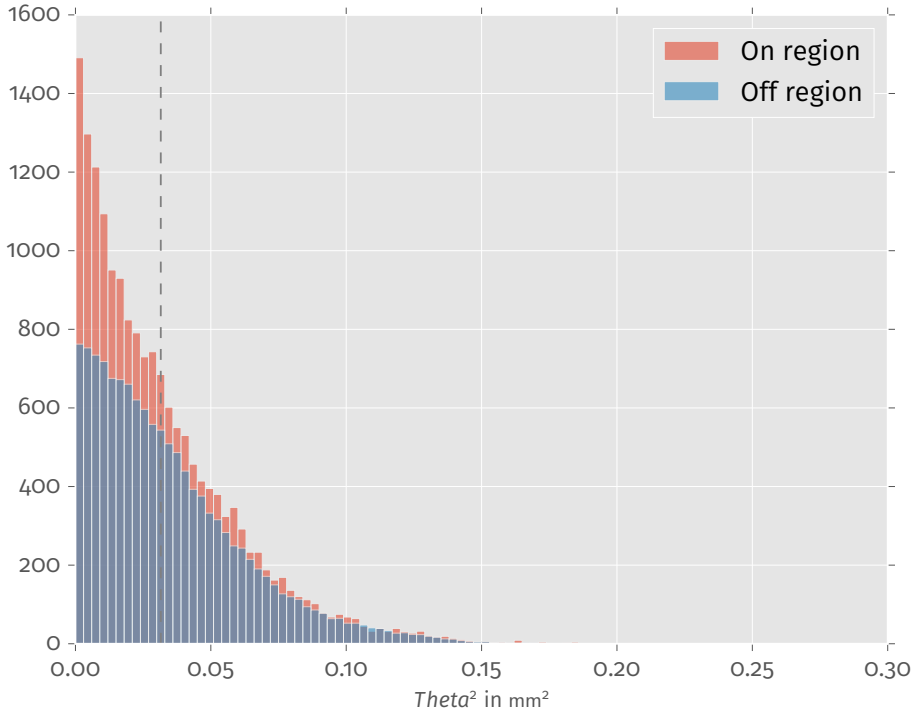

Figure 6.10: Theta plot for a Crab data set. The Li & Ma significance is 33.9.

7 Energy Estimation

Online energy estimation for the Real Time Analysis works in a similar fashion to the online separation. A Random Forest regressor is used to estimate the event energy. The regressor is trained on the same set of variables as the separator from the previous chapter. A complete list of training variables can be seen in appendix B. Feature importances are displayed in figure 7.1. The Random Forest chooses many features which are correlated to the arrival time of the individual Cherenkov photons. This appears reasonable since photons from larger showers tend to have a later arrival time on average. The size parameter also proves to be important according to the Random Forest which, considering it estimates the total amount of Cherenkov light in the shower, also seems reasonable.

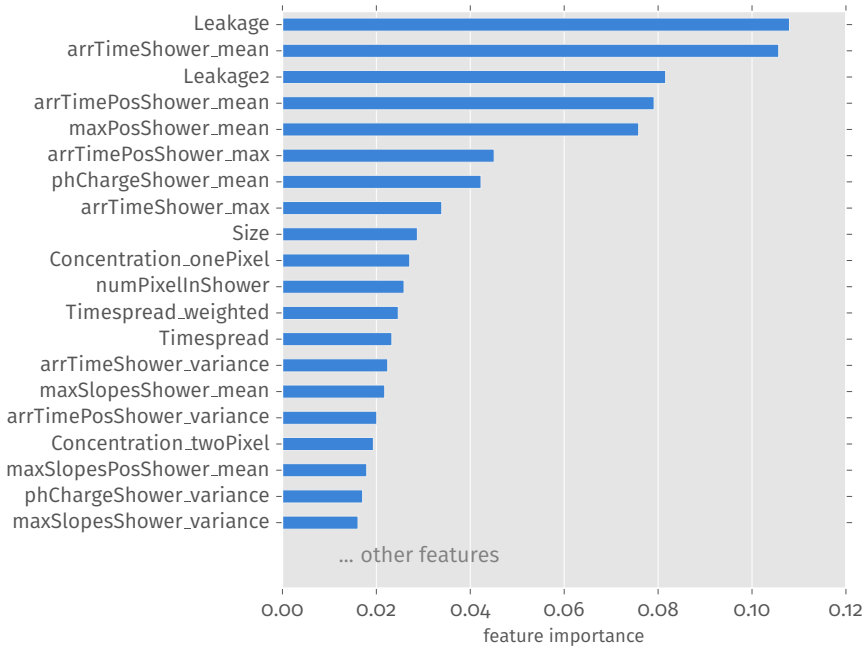

Figure 7.1: Feature importances calculated by the Random Forest regressor for energy estimation.

Figure 7.2 shows the correlation between the true, i.e. simulated, energy and the energy as estimated by the Random Forest regressor. While a correlation between the two is clearly visible, the distribution is not very narrow. In fact it gets broader for lower energies. To quantify the performance of the regressor its resolution and bias are calculated. For each bin a Gaussian distribution is fitted to $\frac{\text{estimated energy} - \text{true energy}}{\text{true energy}}$. The mean and standard deviation, bias and resolution, of the fitted Gauss are drawn in figure 7.3. The errors indicated in the figure are calculated by creating the fit on five different independent test sets in a cross validation. Both the bias and the resolution of the estimator are strongly dependent on the simulated energy. At lower energies the energy estimation is more difficult. Low energy showers create smaller images in the camera since they do not emit as much Cherenkov radiation as their high energy counterparts. Image parameters cannot be extraced as accurately on small shower images since the image resolution is limited due to the finite pixel size and only few pixels are hit by Cherenkov light in the camera. Still, for a small $4\,\mathrm{m}$ telescope, FACT shows good reconstruction performance even for smaller showers at low energies.

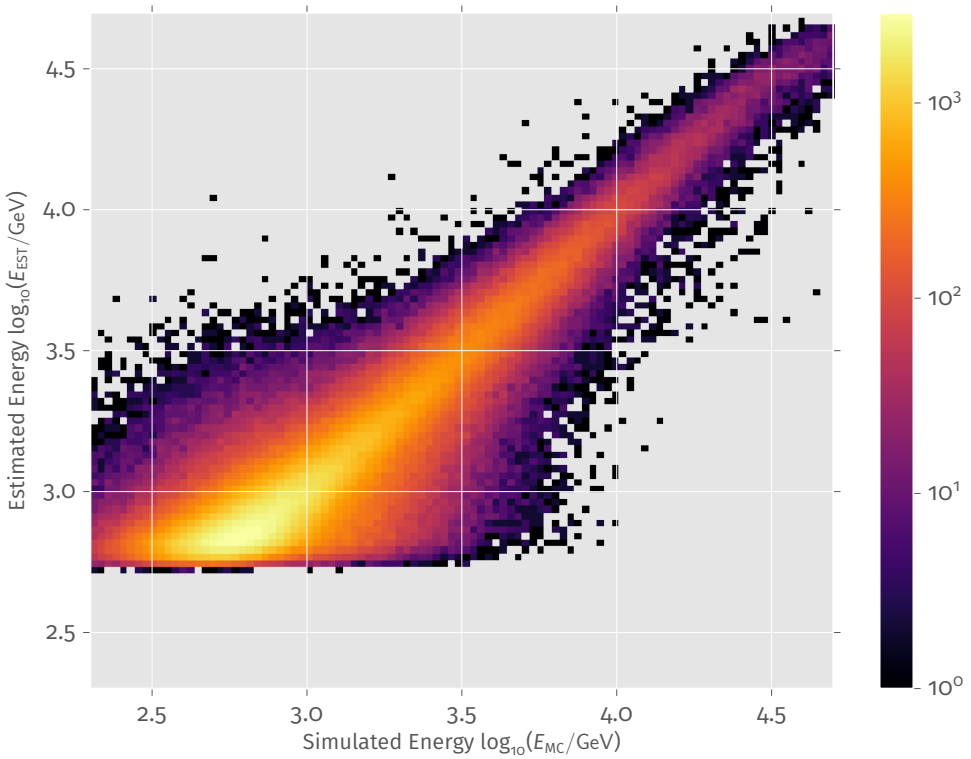

Figure 7.2: The plot shows the true event energy, known from simulations, versus the energy estimated by the Random Forest regressor.

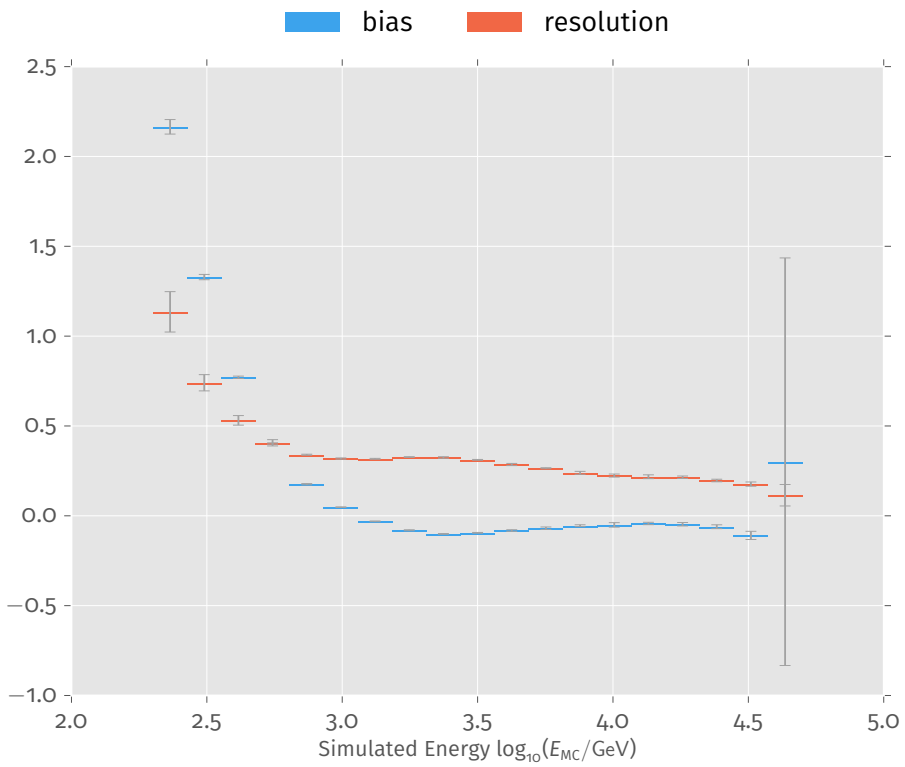

Figure 7.3: The plot shows the bias and resolution of the estimator in energy each bin. The errors are gained from 5-fold cross validation. The errors in the rightmost bin clearly show the lack of statistics in the high energy range.

When the Real Time Analysis is running the estimated energy is displayed in the web interface of the Real Time Analysis along with its error. In conjunction with the instrument response function of the telescope, the estimated energy can be used to calculate a gamma ray flux from the excess rates.

8 Software for FACT Data Processing

Once the models for energy estimation and signal/background separation have been built, they can be applied to a data stream. The software used for building the analysis is based on two major components the `streams`-framework and the FACT-Tools. Both were adapted for the purpose of building the new Real Time Analysis in this thesis. The following section describe the basic software components and concepts of the `streams`-framework and the FACT-Tools.

8.1 The `streams`-framework

The `streams`-framework [5] is a data streaming environment written in the Java programming language. It is the basis for the software which reduces FACT raw data into image parameters. The purpose of the `streams`-framework is to provide an abstract way to model the data flow of an application. It defines a number of fundamental building blocks to define a data flow graph using an Extensible Markup Language (XML) based description. A data stream, in the context of the `streams`-framework, has to have at least one *data source*. It continuously emits small packets of data, the data items. These elementary units of data are processed by small functional units called *processors*. A processor can modify the data item and pass it back to the stream. Then it can be processed by the next entity in the stream. This way processors can be chained together. Once a data item has been fully processed by all processors it can be written into a sink.

Figure 8.1 shows a simple data flow graph. It shows a data stream consisting of a single data source, N different processors and a single sink. Data items emitted by the data source get propagated to each of the processors until they reach the final node in the graph, the sink.

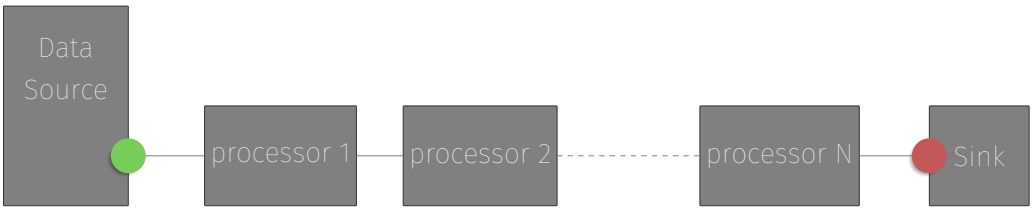

Figure 8.1: A simple data flow graph containing of a single source, a chain of processors, and a sink.

A data source can be something as simple as a local file or something more intricate like live sensor data or a web-service. A processor can perform a calculation based on the values in the data items, e.g. add 1 to each value in the item, and then write the result back to the data item for further processing. Data in a sink can be written to a file, a database or simply be queued until read by a separate streams process. Listing 8.1 is a minimal streams XML file. It reads data items from each line in a Coma Separated Values (CSV) file and passes each data item onto a chain of processors. The XML defines one process containing two processors. They are called in the order in which they appear in the file. The processors perform some calculation on the data stream using the input values provided in the XML tags. The `PrintData` processor simply echoes all values in the data item to the standard output.

Listing 8.1: An XML file defining a simple data flow for streaming data from a CSV file. In this case the CSV file is not read from a local drive but streamed from a webserver via http connection. A valid XML file needs an outer tag by definition. The `<application/>` element is chosen by convention.

```
<application>
    <stream id="sensor" class="stream.io.CsvStream"
url="http://sfb876.de/temperature.csv" />

    <process input="sensor">
      <CalculatePressure amount="5" />
      <PrintData />
    </process>
</application>
```

The `streams`-framework has its own Application Programming Interface (API) which makes it easy to implement new processors or data sources. Creating custom processors is as easy as extending the `Processor` interface from the `streams` API. Listing 8.2 shows an implementation of the `CalculatePressure` processor. The purpose of this processor is to read some fictional temperature and volume values from the data stream and apply the ideal gas law to calculate pressure. The `Processor` defines a single method, `public Data process ( Data item )` which needs to be overridden. This method gets called for every item in the data stream. To pass values from the XML to the processor during runtime the `@Parameter` annotation is used. In this example the amount of substance can be set in the XML. The annotation was introduced in an effort to create self-documenting code. The annotations and its *description* value are published in the automatically generated API documentation. Creating custom data sources or sinks works in a similar fashion. The API provides interfaces or abstract default implementations for these common functionalities.

Listing 8.2: A custom processor is created by implementing the `Processor` interface. This processor calculates pressure according to the ideal gas law. One parameter, the amount of substance, is passed into the processor from the XML file. This happens during runtime after the XML has been parsed. No recompilation is necessary to inject new values.

```
public class CalculatePressure implements Processor {

    final double k_b = 1.380E-23 // Joule per Kelvin
    final double N_A = 6.022E23 // 1 per mol

    @Parameter(required = false, defaultValue = 0.5,
                    description="Amount of substance in mol")
    double amount = 0.5;

    @Override
    public Data process ( Data item ) {
      double T = (double) item.get ("temperature") ;
      double V = (double) item.get ("volume") ;

      double p = N_A * amount  * k_b * T / V

      item.put("pressure", p) ;
      return item;
    }
}
```

Programs built on top of the streams-framework can be bundled into a single, machine independent, executable. Together with the data flow description this leads to shareable and reproducible data analysis processes. The current development goal of the streams-framework is to provide runtimes to execute the processes in large-scale distributed computing environments. Experimental runtimes exist for mapping streams processes to topologies used by Twitter's real time compute engine Storm [36]. Runtimes for Apache Spark [40] and Apache Flink [4] are under active development.

8.1.1 Local Multi-Threaded Execution

The streams-framework natively supports multi-threaded streaming. The data items are completely independent of each other. Each process streaming from a data source runs in a dedicated thread. Stream can be split and merged using thread-safe queue elements. A stream can be written to a queue using an `Enqueue` operator. Queues are data sources and can therefore natively be read by processes. There is no limitation on how many processes can read from and write to a queue. The queues are blocking whenever they reach a predefined capacity of data items. Otherwise the memory of the machine running the process might be exhausted quickly. Figure 8.2 shows how data items from a source get written to a queue element. Two different processes read from the queue, perform some calculation, and merge the stream back into a sink. A setup as shown in the picture can improve performance on multi-core machines, especially if processors A and B do long-running calculations in parallel.

8.1.2 Stream Services

Processors within a data stream often need access to meta data. For a typical FACT application this could be the current weather or the pointing position of the telescope. This information is not part of the telescope's raw data stream. Services provide a way to access this data from within a data stream. They run in their own dedicated thread and are independent of any running data streams. Services have no common contract in what functions they export. A service is defined by implementing the `Service` interface. Processors get access to the service instance through dependency injection via the XML. Similar to the `@Parameter` annotation, there is the `@Service` annotation to inject services into processors. During its lifetime a processor can call any public methods on the service. Services have no inherent mechanism to guarantee thread safety. When implementing a service one has to explicitly take concurrent access by many streams and process into account. Figure 8.3 shows two parallel streams accessing a shared service. Each stream is running in an individual thread and so is the service.

Listing 8.3: One data source streaming data into a process. The queue consumes the data and passes it on to the next processors running in parallel.

```
<application>
    <queue id="queue_1"/>
    <queue id="queue_2"/>

    <stream id="data" class="stream.io.CsvStream"
url="http://sfb876.de/data.csv" />

    <process input="data">
      <Process/>
      <Enqueue queues="queue_1" condition="%{data.temperature} == 1"/>
    </process>

    <process input="queue_1">
      <ProcessorA />
      <Enqueue queues="queue_2"/>
    </process>

    <process input="queue_1">
      <ProcessorB />
      <Enqueue queues="queue_2"/>
    </process>

    <process input="queue_2">
      <Processor/>
      <PrintData/>
    </process>
</application>
```

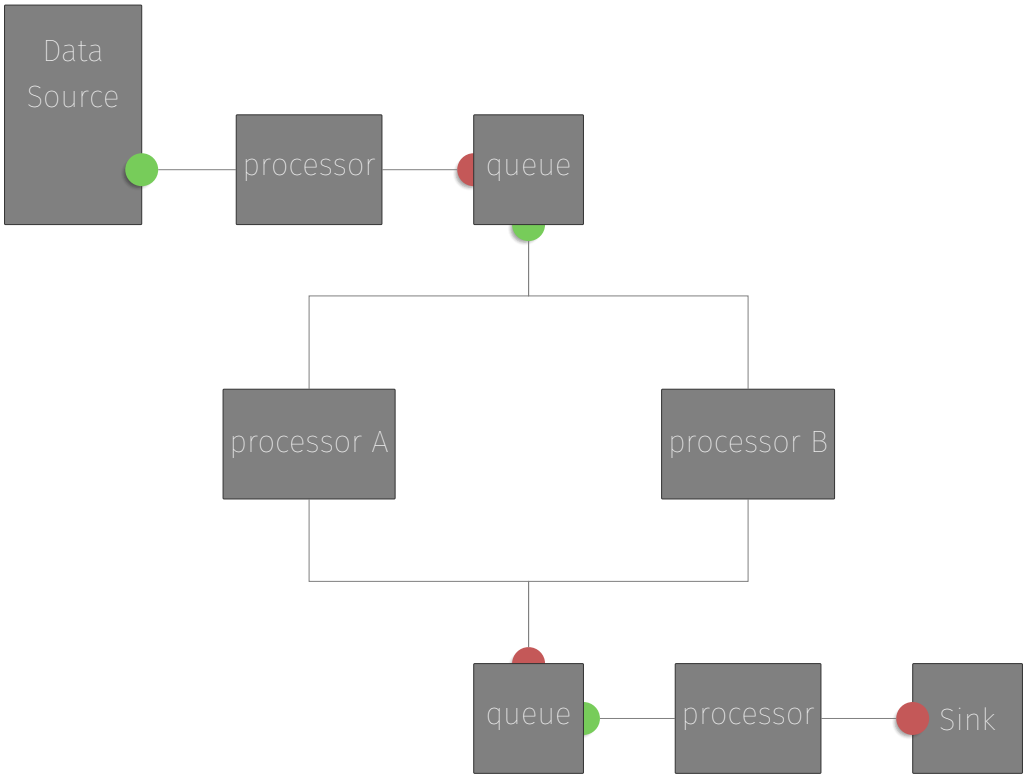

Figure 8.2: The two parallel edges in this data flow graph are executed in two separate threads. Both access the same blocking queue element for new data items. The streams get merged back into one by writing the data items back into another queue. Note that the order of elements is not necessarily the same after merging the two streams.

The data flow graph as shown in figure 8.3 can be defined in one single XML file. Listing 8.4 shows an XML with two streams. The service is defined outside of any process tag. It is accessed by its unique id by two processors.

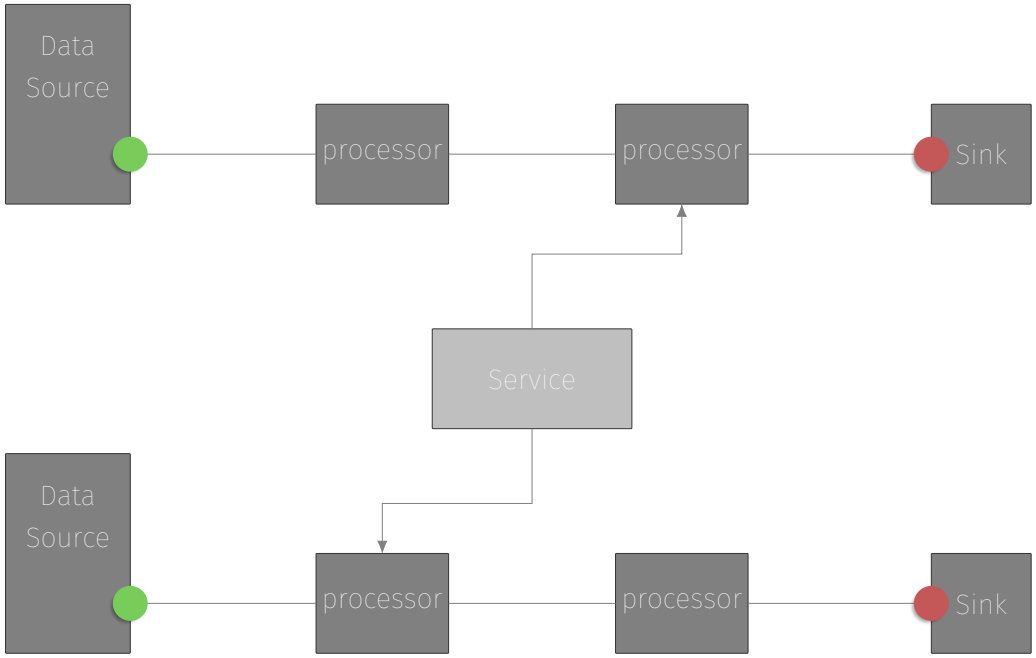

Figure 8.3: Two parallel streams accessing a service. Concurrent access to the service has to be handled by the implementation of the service.

Listing 8.4: Two data sources streaming the file a.csv and b.csv. Two processes access a single service instance with the id “example”.

```
<application>
    <service id="example" class="stream.ExampleService" />

    <stream id="a" class="stream.io.CsvStream" url="http://sfb876.de/a.csv" />
    <stream id="b" class="stream.io.CsvStream" url="http://sfb876.de/b.csv" />

    <process input="a">
      <Process1 />
      <Process2 service="example" />
      <PrintData/>
    </process>

    <process input="a">
      <Process3 />
      <Process4 service="example" />
      <PrintData/>
    </process>

</application>
```

8.2 The FACT-Tools

The program for raw data processing is called FACT-Tools. It is a collection of tools build on top of the `streams`-framework and has been under active development since 2012. A raw FACT event contains 300 samples for each of its 1440 pixels. One night of FACT measurements can create hundreds of gigabytes of data. For signal/background separation only the image parameters of an event is needed. There is not yet any automatic process to reduce the raw data to its image parameters. Users always start their analyses at the raw data level. This is due to the fact that FACT uses prototype SiPM technology to measure Cherenkov photons. The influence of low-level calibration and extraction parameters is still being investigated. The FACT-Tools provide reading routines, `streams`-framework data sources, that stream data items from the FITS files that the FACT DAQ produces. The FACT-Tools mostly contain simple stream processors that perform single tasks on the raw data. One of the first things that were imeplemented into the FACT-Tools were the low-level DRS calibration and the calculation of Hillas parameters such as width and length as described in chapter 4. Many more methods for low-level data treatment and feature extraction have been implemented in the recent years.

Traditionally the FACT DAQ writes so-called slow-control-data into FITS files as well. These files contain information about the current weather conditions, pointing information, hardware properties and similar meta-data. The FACT-Tools use stream services to give the processors access to the information in these files. A graphical user interface was build for quick visual inspection of data and evaluation of algorithms. Figure 8.4 shows a screenshot of the interface.

Data processing for FACT is usually performed in a batch manner. Users manually select files stored in some central storage and call the program to process the data. The output files are further analyzed by other programs providing machine learning methods like the ones described in chapter 5. The usual suspects for further processing of image parameters are ROOT [14], RapidMiner [31] and the scientific Python stack [38]. In a real time setting this work-flow is unfeasible. Models created by other programs have to be applied during the runtime of the FACT-Tools. The next chapter describes how the FACT-Tools are extended to allow for online application of machine learning methods.

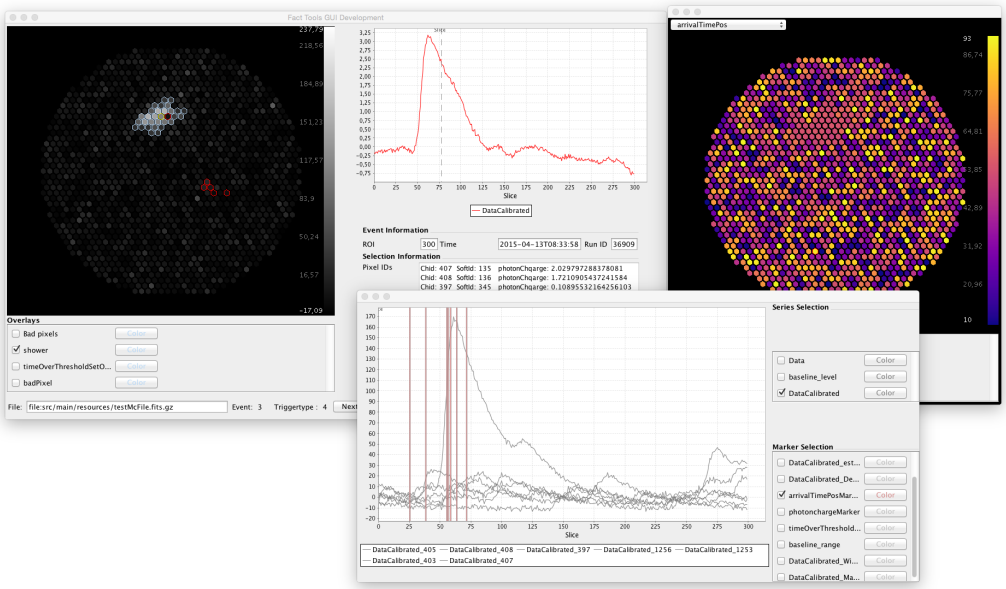

Figure 8.4: A screenshot of the FACT-Tool graphical user interface. It can show the whole camera image as well as single voltage curves. It includes a list of useful event information and supports a number of different color maps and additional visualizations. Its visualizations can be extended easily by adding special objects to the data stream, which are then automatically available for selection in the user interface.

9 The Real Time Analysis

The major goal of this thesis is to build a system to analyze FACT data live and in real time. The Real Time Analysis should be completely defined within a single XML file for the `streams`-framework. The FACT-Tools RTA will run on a single machine next to the telescope. Data will be analyzed as soon as the DAQ has finished its writing process. It includes a web server which can show live displays of the telescope's status and current excess rates. The key part of the Real Time Analysis, however, is the online application of machine learning models.

9.1 Online Model Application

The goal was to apply models to a continous stream of data items. A multitude of libraries and programs for supervised learning already exist. To create a new one simply for the purpose of applying the models in the `streams`-framework would be futile. The Predictive Model Markup Language (PMML) format is an easy way to share models between languages and programs. It is an XML-based format which defines building blocks to encode the most common multivariate models and data transformations. Plenty open-source libraries for using PMML in Java, R, Ruby and Python exist. The standard was first published in 1997. The current version 4.0 was released in 2009 and last updated in 2014. It is backed by the Data Mining Group (DMG) and in use in data mining companies such as BigML [27].

In this thesis the models are created using the Python library scikit-learn as described in chapter 6. To export the models from scikit-learn to PMML the open source library sklearn2pmml [32] is used. The import into the FACT-Tools is accomplished by the jpmml-evaluator library.

A new `PredictionService` was implemented to provide model predictions to the data stream. Besides the convenient access by a streams processor, it also allows for loading large, memory intensive models into the data flow graph, which can be accessed by many parallel streams. The service has two public methods which processors can access. For getting the result of a regression model the `Double regressionPrediction(Data item)` method can be called. It takes a data item and returns a single `Double` value as a prediction for the target variable. To predict a class for a data item the `ProbabilityDistribution classPrediction(Data item)` method is used. It returns an object containing the predicted class probabilities for the data item. This way each processor can apply a unique prediction threshold as described in section 5.2.

Listing 9.1 shows how the service can be used in conjunction with a processor. The `Signal` and `Energy` processors were added to the process to write the predictions to the stream. The values can later be written to a file or used for displaying in the RTA frontend.

Listing 9.1: A shortened example of a streams process, which performs energy estimation and signal/background separation on FACT data. Two models are loaded from a file into a service.

```
<application>
    <service id="rf" class="fact.rta.PredictionService"
url="file:./model.pmml" />
    <service id="regressor" class="fact.rta.PredictionService"
url="file:./regressor.pmml" />

    <stream id="data" class="fact.io.FitsStream"
url="http://sfb876.de/data.fits" />

    <process input="data">
      <!-- Calibration, preprocessing and image parameter calcuclation -->
      <fact.rta.Signal predictor="rf" />
      <fact.rta.Energy predictor="regressor" />
    </process>
</application>
```

9.2 Data Stability

Environmental factors can have a large impact on the data quality and trigger rates. Temperature of the camera and power supply electronics can affect the data in various ways. The data produced by FACT's Monte Carlo simulation programs are simulated for very specific background light and temperature conditions. For an offline analysis this is usually no problem. The user can simply select the data which fits the simulated environment best, e.g. dark nights and stable trigger rates. For an online analysis however no such data selection can take place. To make sure the model was trained on valid ranges of the parameter space the image parameters have to be stable over long periods. Figure 9.1 shows a two-dimensional histogram of the width parameter. During the six months plotted here, the distribution seems rather stable. The position of the peaks and the overall shape of the distribution stays the same.

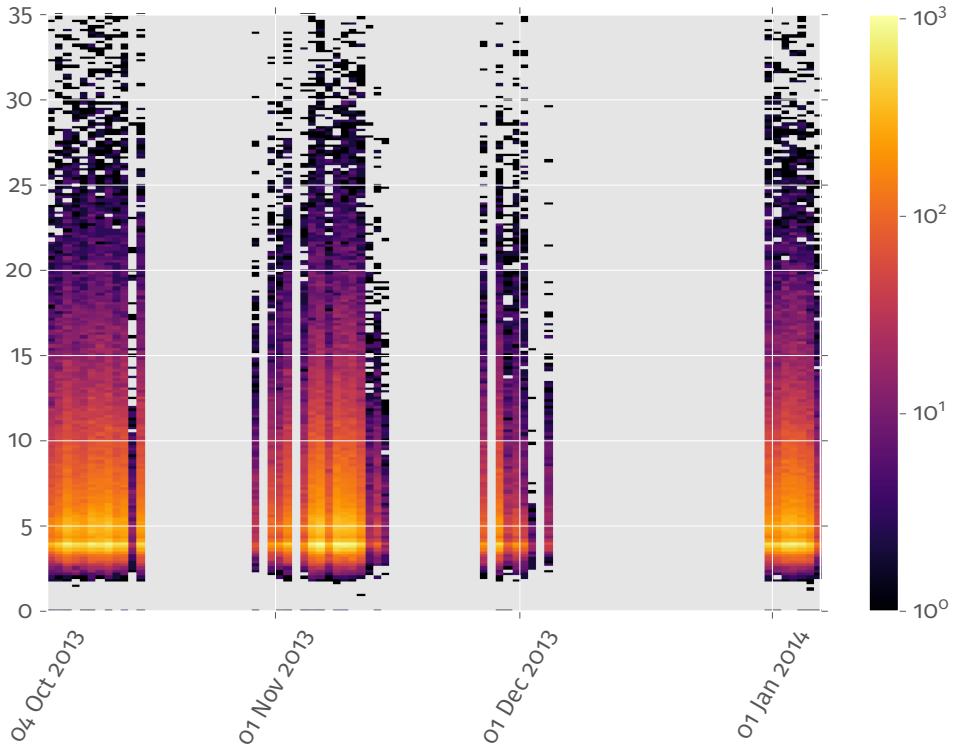

Figure 9.1: The image parameter known as width (see section 4.4) over the course of three months. The single distributions in the bins have not been normalized with respect to the total amount of entries.

9.3 Excess Rates

As seen in subsection 6.1.1 data can be divided into gamma-like events from the on-region and gamma-like events coming from the off-region. The infamous Theta^2 plots are a common tool among IACT experimenters to compare instrument performance and source detection capabilities. So are curves of excess rates. The excess in one fixed width time bin is defined as

$$N_{\text{excess}} = N_{\text{On}} - N_{\text{Off}} \cdot \alpha \tag{9.1}$$

where N_{On} is the total number of gamma-like events coming from the source region, N_{Off} the total number of gamma-like events coming from the off positions and α is $1/\text{number of off regions}$. Assume N_{On} and N_{Off} are independent random variables distributed according to the poisson distributions $P_{\lambda_{\text{On}}}$ and $P_{\lambda_{\text{Off}}}$ with $\text{Var}(P_{\lambda_{\text{On}}}) = \lambda_{\text{On}}$ and $\text{Var}(P_{\lambda_{\text{Off}}}) = \lambda_{\text{Off}}$. For any constant factor $\beta \in \mathbb{R}$ and two independent random variables X and Y

$$\text{Var}(X - \beta Y) = \text{Var}(X) + \beta^2 \,\text{Var}(Y)$$

holds. Hence one can calculate the standard deviation of the excess rate assuming $\lambda_{\text{On}} = N_{\text{On}}$ and $\lambda_{\text{Off}} = N_{\text{Off}}$ as

$$\sigma(N_{\text{excess}}) = \sqrt{N_{\text{On}} + \alpha^2 N_{\text{Off}}}. \tag{9.2}$$

Figure 9.2 shows the excess curve of the AGN Markarian 501 during the month of June 2014 in a 20 minute binning. The error bars indicate the standard deviation of the excess rate as calculated above. The activity of the source shows a clear peak around June 24th. At that time FACT, as well as the HESS telescope, sent out a flare alert [16][17]. This plot confirms that the RTA is sensitive enough to measure flaring states of cosmic gamma ray sources and could be used to send out flare alerts.

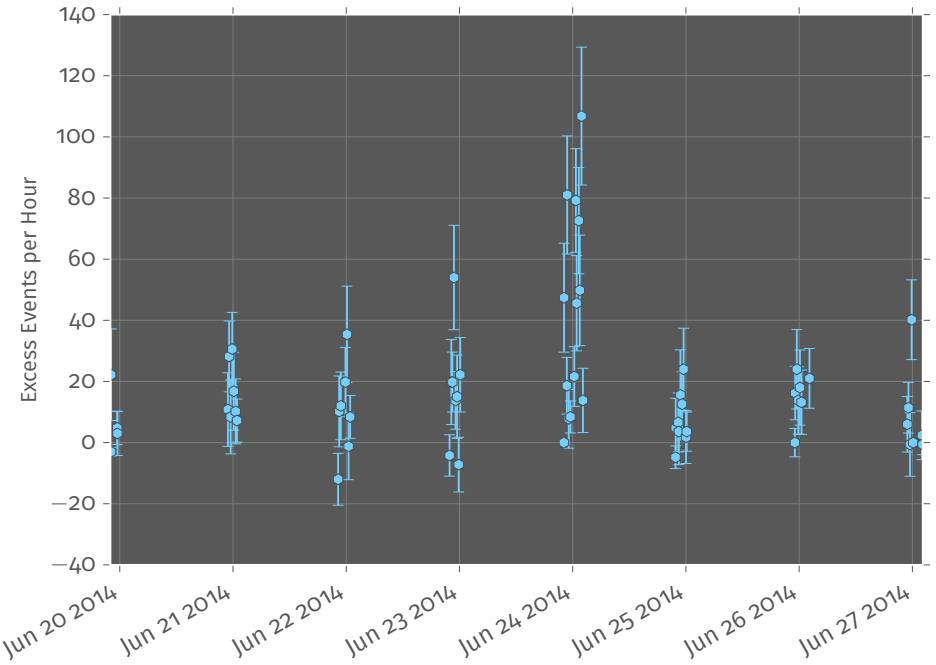

Figure 9.2: Observations of the active galactic nucleus Markarian 501 during June 2014. This plot was produced by applying the RTA process to archival data. The data was grouped into 20 minute bins. The peak at June 24th conincides with a flare detected by the FACT and HESS experiments. These points have not been corrected for dead time of the sensors.

9.4 Runtime Considerations

The `streams`-framework has proven to be capable of calculating image parameters for Cherenkov telescopes with very high event rates [13]. However, the RTA has to perform a multitude of additional tasks, such as raw data calibration, running a webserver and application of machine learning models. The RTA certainly needs to be capable of analyzing more events per second than the FACT trigger rate. Figure 9.3 shows the FACT trigger rate over the time period of a whole year. Each little point is the trigger rate of a single data run. The mean of the trigger rate is at 47.7 with a standard deviation of 28.5. The red line shows a running mean over 900 samples. It has a noticable periodicity. The thin gray line in the figure indicates the moon phase. Evidently there is a connection between

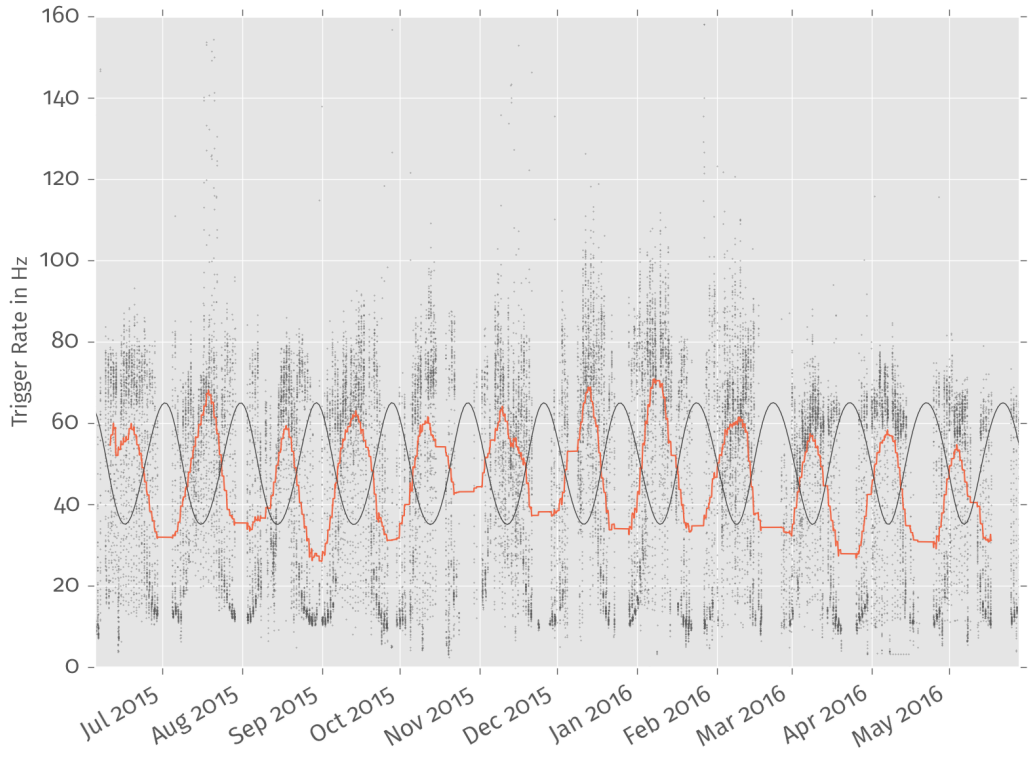

Figure 9.3: The FACT trigger between June 2015 and June 2016. This plot shows all data runs that were taken. Technical runs, like calibration related data, are of no meaning to the Real Time Analysis and therefore are not taken into account. The mean trigger during the period is at 48 Hz.

the moon phase and the mean trigger rate. Moon light gets reflected off the ground or scattered in the atmosphere creating more background light in the camera. More light in the camera leads to a higher trigger threshold resulting in a lower trigger rate.

The Real Time Analysis has to able to respond to peaks in the trigger rate without accumulating much delay. To allow for fluctuations in the data rate the minimum event rate that has to be achieved by the FACT-Tools RTA is chosen to be 100 events per second.

To get a sense of the machinery which is needed to perform the described tasks, a simple measurement, in a single threaded environment, is performed. The goal is to find out how much relative runtime is needed for each of the steps in the FACT analysis. Hoping of course that it will not be dominated by the model application. To measure the runtime of each single processor, a new `PerformanceMeasuringProcess` class was created. It can be applied just like any other process. Listing 9.2 shows a quick example. It can write its result to a JSON [24] file specified in the `url` parameter or simply print it to the standard output.

Listing 9.2: A shortened example of a streams process using the `PerformanceMeasuringProcess` class.

```
<application>
    <stream id="data" class="fact.io.FitsStream"
url="http://sfb876.de/data.fits" />

    <process class="fact.PerformanceMeasuringProcess"
url="file:./somefile.json" input="data">
      <!-- Calibration, preprocessing and image parameter calcuclation -->
    </process>
</application>
```

The measurements were performed on a single core on Java Virtual Machine (JVM) version 1.8.0-60 with the JVMs default setting. Figure 9.4 shows the relative runtime for single processors applied during the RTA. The runtime of the processors responsible for preprocessing the data clearly dominates all others. The processors to calculate image parameters are too fast to be seen here. The plot shows that the application of the model is not a large factor in overall processing time.

In figure 9.5 the runtime per thread is displayed. This is one single streams process started with a single XML file. Each thread can analyze about 20 events per seconds. In total 205 ± 11.7 events per second can be analyzed when using 10 threads. The original performance goal of 100 events per second is surpassed by a factor of two.

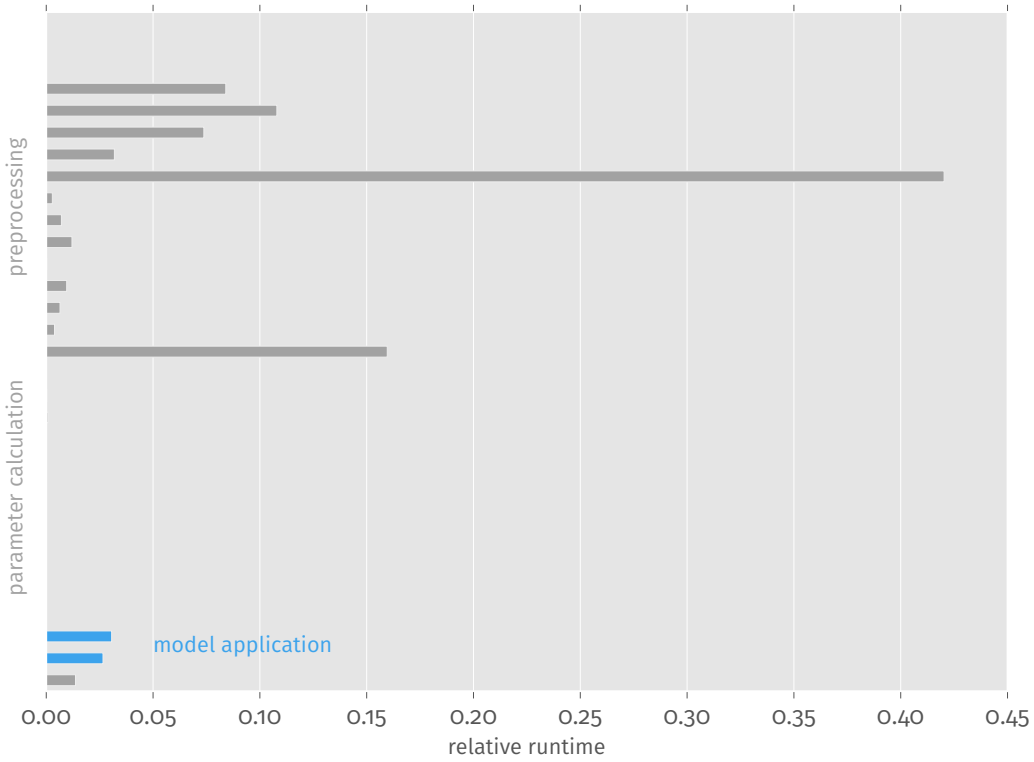

Figure 9.4: Relative runtime of processors during the RTA. The preprocessing takes the largest amount of time since. For raw data treatment many iterations over the raw data array (1440 ×300 elements) have to be performed. Application of the model is a relatively small factor compared to the rest.

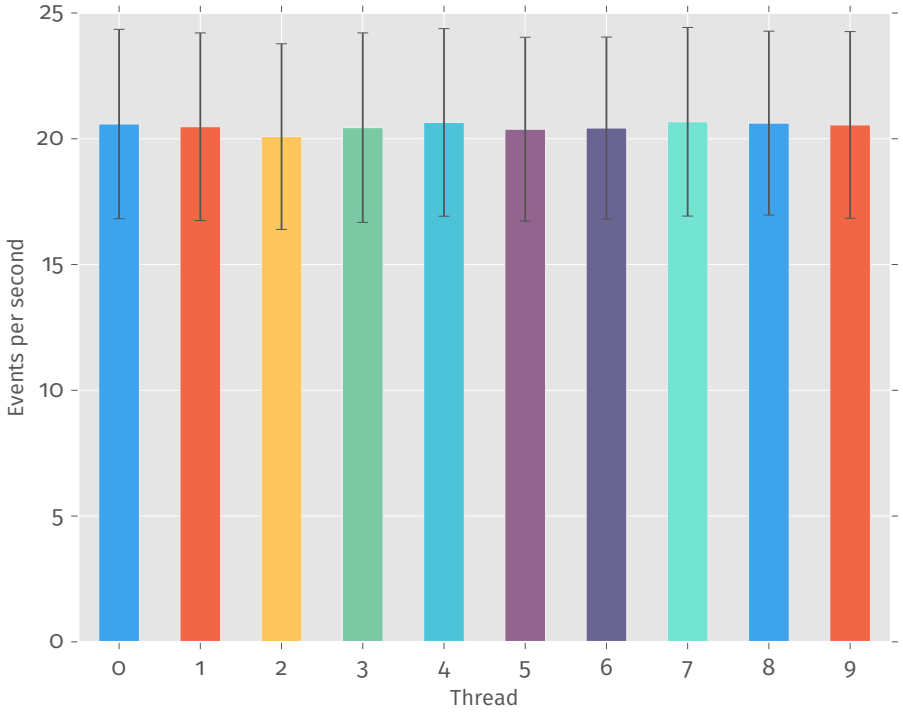

Figure 9.5: Runtime per thread in events per second. In total the performance goal is surpassed by a factor of two. For this measurement the FACT raw data was read from the hard disk using the Kryo[18] serialization format.

9.5 The Web-Frontend

The last step for building the RTA is the website. The whole RTA should be bundled into a single executable. This is why a webserver has to be embedded in the application. This is no problem when using one of the many web-related Java microframeworks which became popular in recent years. In this case the Spark Web Framework[1] was chosen for no other reason than having an aesthetically pleasing homepage. The following key components will be visualized on the website:

1. A camera image of the most recent gamma-like event.
2. Machine status such as the current memory consumption and free hard disk space.
3. The current data rate of the analysis for monitoring the status of the RTA.
4. A plot of binned excess rates.

To achieve this another streams service was build. The `RTAWebService`. Processes can connect to the service and update it with new data. The data will be cached in the services for a given amount of time. The data is then exposed via a simple representational state transfer (REST) interface. A client, a web browser openened by someone interested in seeing RTA results, uses a handful of AJAX requests to poll the REST endpoints for new data. To display the FACT camera image a JS module was build which uses the D3 [9] library to display the colored hexagons. For more information about the technical details and implementation of the frontend see the notes in appendix C.

9.5.1 Persitent Storage of Results

Results of the RTA have to be persisted to a hard drive. For the sake of simplicity an SQLite database was chosen. Besides being easy to use, it is also the simplest solution to creating a single executable program for the RTA. No other dependencies, e.g. a MySQL server, have to be installed for running the RTA. So far the database consists of two tables. One table contains a row for each analyzed event, the other contains one row per run. A FACT *run* is a distinct unit of measurement with a fixed duration which typically ranges from one to five minutes. Each event is connected to a run via a foreign key relation. The run table is filled with effective observation time and information about the status of the analyzed data, e.g. whether the run was succesfully analyzed or not.

The `RTAWebService` is responsible for writing to the database in regular intervals.

[1] Not to be confused with the Apache Spark Framework for Big Data processing

9.6 Putting it all Together

Starting the program is as simple as calling the FACT-Tools executable with the XML file containing the RTA services and processors. The inputs are plain FACT raw data files as written by the DAQ. To perform signal/background separation and energy estimation to paths to the models, the PMML files, have to be set. Once the stream starts running, the web interface can be seen by pointing a webbrowser to the host running the RTA. Final perfomance, in terms of analyzed events per second, depends on the specific data format written by the telescope's DAQ.

10 Conclusions and Future Plans

The key part of IACT data analysis is certainly the signal/background separation. The data is contaminated heavily by background noise. The background events dominate the signal by many orders of magnitude. To effectively separate signal from background, supervised machine learning approaches have proven to be a successful method. Not just in Cherenkov astronomy but in many areas of high energy physics. To apply these techniques, image parameters have to be extracted from the raw data. The calculation of the Hillas parameters, see chapter4, was the first essential step in creating the FACT-Tools. The FACT-Tools contain the full range of of methods for calibration, filtering, image parameter calculation and metadata handling needed for analyzing FACT data. Both measured data as well as simulated data can be processed by the FACT-Tools. The processed data can then be used to train and apply supervised machine learning models. The weapon of choice for signal/background separation is the Random Forest. Its ubiquitous use throughout the whole IACT community prompted the detailed explanation of decision trees in chapter 5. The need for streamed processing of FACT data motivated the effort to apply machine learning models to an online data stream. After testing the concepts of online model application in the contribution to the European Conference on Machine Learning (ECML) [6] this thesis successfully applied these concepts in a production setting. Machine learning methods are pervasive in today's data analysis processes in every area of research. An abundance of tools for creating multivariate models exist in all major programming languages. The PMML format allows for quick interchange of models between the several machine learning tools and the FACT-Tools.

A real time analysis has strong requirements on usability, scalability, maintainability and processing speed. Results need to be available as soon as possible. At the same time they must be persisted in a reliable way which can be used for further analysis. This was achieved by connecting the data stream to a database via a single abstract interface point. The strict runtime constraints could be fulfilled and even surpassed by a factor of two. Additionally the new RTA features a web interface showing live analysis results and data rates using simple visualizations. Figure 10.1 shows a screenshot of the web interface.

The next step is to deploy the system at the telescope site. Once the system is running stable, models with better sensitivities can be investigated. In the near future more Monte Carlo simulations will be necessary to produce proper fluxes from the measured excess rates.

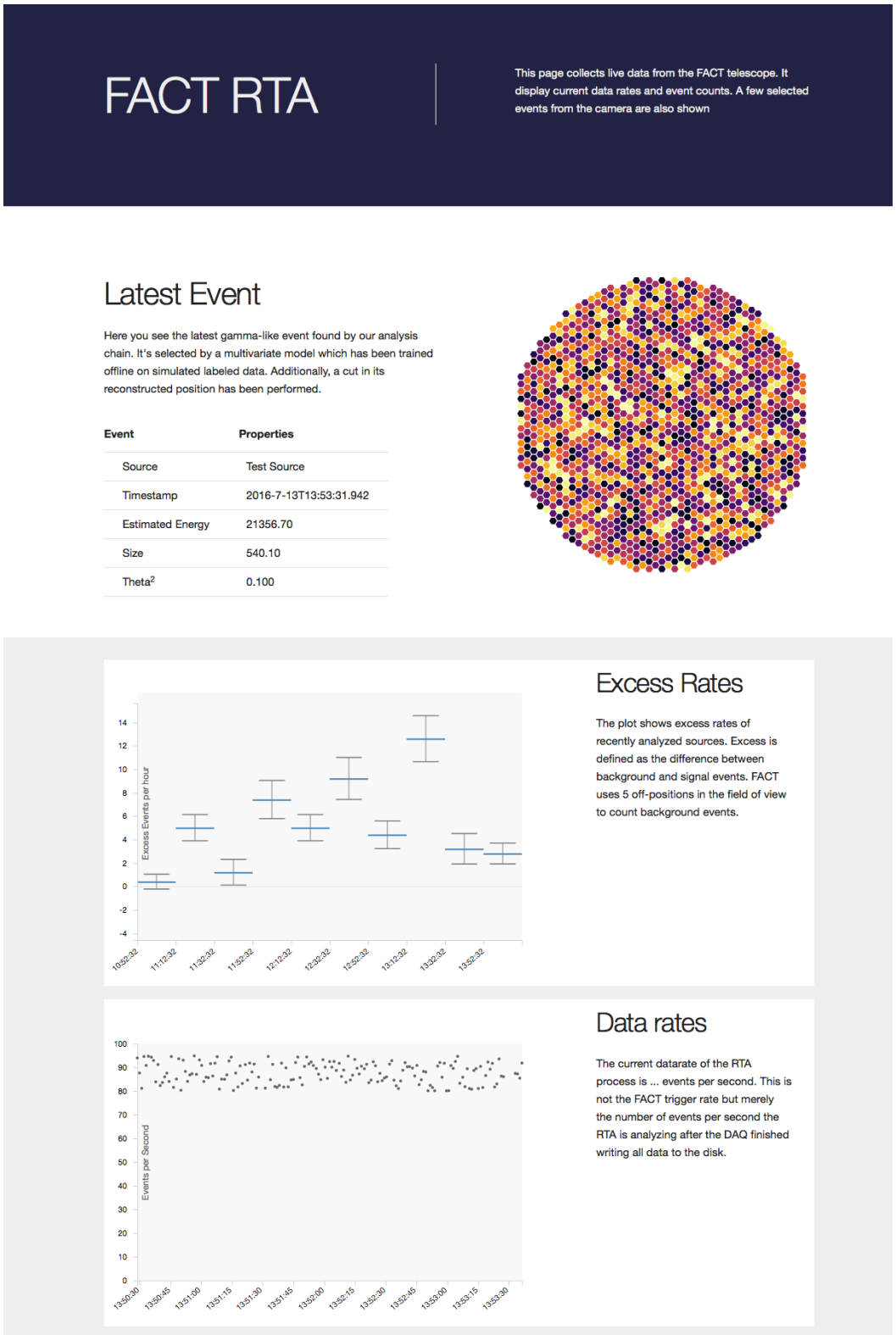

Figure 10.1: A screenshot of the web interface for the RTA. Excess rates are plotted live in the browser.

A More Excess Rates

The plot in figure A.1 shows excess rates of Markarian 501 over the summer of 2014. The two flares can easily spotted in this 120 minute binning. The timing of these flare completely coincides with the traditional FACT analysis.

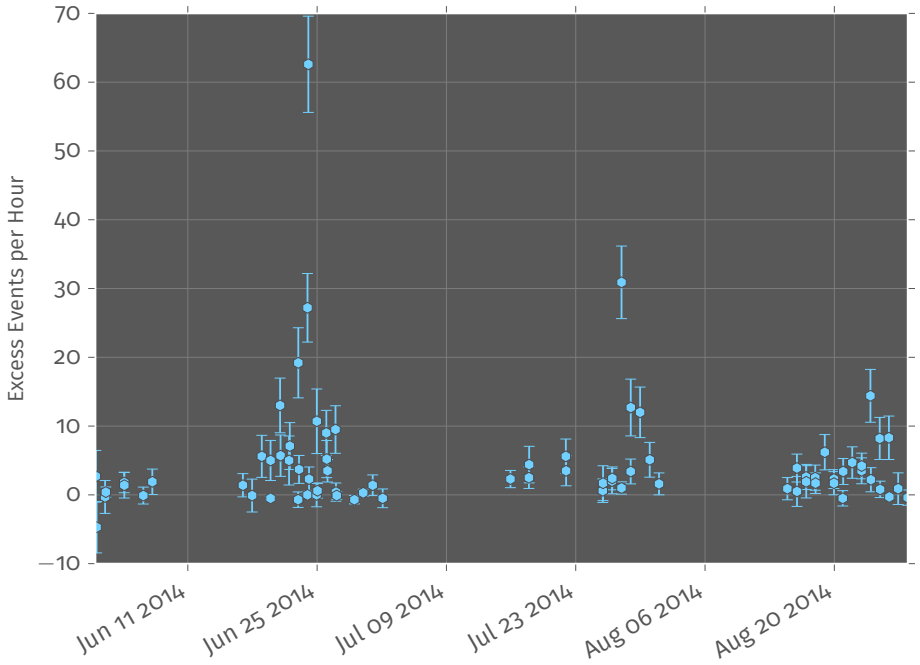

Figure A.1: Observations of the active galactic nucleus Markarian 501 during the summer of 2014. This plot was prodcued by applying the RTA process to archival data. The data was grouped into 120 minute bins. The high peaks coincide with flares seen by FACT and other instruments. These points have not been corrected for dead time of the sensors.

B Learner Configurations

The configuration files used for training the classifiers. Contents should be more or less self explanatory. For more information visit https://github.com/fact-project/classifier-tools

Listing B.1: The config file used to train the energy regressor.

```
# hmm realy tasty config for realy tasty classification stuff

#the classifier to use
classifier : |

    ensemble.RandomForestRegressor(n_estim
    max_features='sqrt', n_jobs=24,
    max_depth=15)

#randomly sample the data if you dont want to use the whole set, -1 for all samples
sample : -1

#query to apply to the data before separation
query : '(Leakage < 0.2) & (numPixelInShower > 8)'

# define th number of cross validations to perform
num_cross_validations : 5

# training_variables = ['Size', 'Length', 'Width', 'numIslands', 'Leakage',
#         'm3l', 'm3t', 'Disp', 'Timespread', 'arrivalTimeMean',
#         'ConcCore']
#specify the variables to train on
training_variables : ['ConcCore',
 'Concentration_onePixel',
 'Concentration_twoPixel',
 'Leakage',
 'Leakage2',
 'Size',
 'Slope_long',
 'Slope_spread',
 'Slope_spread_weighted',
 'Slope_trans',
 # 'Distance',
 # 'Theta',
 # 'Alpha',
 'Timespread',
 'Timespread_weighted',
 'Width',
 'arrTimePosShower_kurtosis',
 'arrTimePosShower_max',
 'arrTimePosShower_mean',
 'arrTimePosShower_min',
 'arrTimePosShower_skewness',
 'arrTimePosShower_variance',
 'arrTimeShower_kurtosis',
 'arrTimeShower_max',
 'arrTimeShower_mean',
 'arrTimeShower_min',
 'arrTimeShower_skewness',
 'arrTimeShower_variance',
 'concCOG',
 'm3l',
 'm3t',
 'maxPosShower_kurtosis',
 'maxPosShower_max',
 'maxPosShower_mean',
 'maxPosShower_min',
 'maxPosShower_skewness',
 'maxPosShower_variance',
 'maxSlopesPosShower_kurtosis',
 'maxSlopesPosShower_max',
 'maxSlopesPosShower_mean',
 'maxSlopesPosShower_min',
 'maxSlopesPosShower_skewness',
 'maxSlopesPosShower_variance',
 'maxSlopesShower_kurtosis',
 'maxSlopesShower_max',
 'maxSlopesShower_mean',
 'maxSlopesShower_min',
 'maxSlopesShower_skewness',
 'maxSlopesShower_variance',
 'numIslands',
 'numPixelInShower',
 'phChargeShower_kurtosis',
 'phChargeShower_max',
 'phChargeShower_mean',
 'phChargeShower_min',
 'phChargeShower_skewness',
 'phChargeShower_variance',
 # 'photonchargeMean'
 ]
```

Listing B.2: The config file used to train the source independent model.

```
# hmm realy tasty config for realy tasty classification stuff

#the classifier to use
classifier : |

    ensemble.RandomForestClassifier(n_esti
    max_features='sqrt', n_jobs=24,
    max_depth=15,
    criterion='entropy')

#randomly sample the data if you dont want to use the whole set, -1 for all stuff
sample : -1

#query to apply to the data before separation
query : '(Leakage < 0.2) & (numPixelInShower > 8)'

# define th number of cross validations to perform
num_cross_validations : 5

# training_variables = ['Size', 'Length', 'Width', 'numIslands', 'Leakage',
#         'm3l', 'm3t', 'Disp', 'Timespread', 'arrivalTimeMean',
#        'ConcCore']
#specify the variables to train on
training_variables : ['ConcCore',
 'Concentration_onePixel',
 'Concentration_twoPixel',
 'Leakage',
 'Leakage2',
 'Size',
 'Slope_long',
 'Slope_spread',
 'Slope_spread_weighted',
 'Slope_trans',
 # 'Distance',
 # 'Theta',
 # 'Alpha',
 'Timespread',
 'Timespread_weighted',
 'Width',
 'arrTimePosShower_kurtosis',
 'arrTimePosShower_max',
 'arrTimePosShower_mean',
 'arrTimePosShower_min',
 'arrTimePosShower_skewness',
 'arrTimePosShower_variance',
 'arrTimeShower_kurtosis',
 'arrTimeShower_max',
 'arrTimeShower_mean',
 'arrTimeShower_min',
 'arrTimeShower_skewness',
 'arrTimeShower_variance',
 'concCOG',
 'm3l',
 'm3t',
 'maxPosShower_kurtosis',
 'maxPosShower_max',
 'maxPosShower_mean',
 'maxPosShower_min',
 'maxPosShower_skewness',
 'maxPosShower_variance',
 'maxSlopesPosShower_kurtosis',
 'maxSlopesPosShower_max',
 'maxSlopesPosShower_mean',
 'maxSlopesPosShower_min',
 'maxSlopesPosShower_skewness',
 'maxSlopesPosShower_variance',
 'maxSlopesShower_kurtosis',
 'maxSlopesShower_max',
 'maxSlopesShower_mean',
 'maxSlopesShower_min',
 'maxSlopesShower_skewness',
 'maxSlopesShower_variance',
 'numIslands',
 'numPixelInShower',
 'phChargeShower_kurtosis',
 'phChargeShower_max',
 'phChargeShower_mean',
 'phChargeShower_min',
 'phChargeShower_skewness',
 'phChargeShower_variance',
 # 'photonchargeMean'
 ]
```

Listing B.3: The config file used to train the source dependent model.

```
# hmm realy tasty config for realy tasty classification stuff

#the classifier to use
classifier : |

    ensemble.RandomForestClassifier(n_esti
    max_features='sqrt', n_jobs=24,
    max_depth=15,
    criterion='entropy')


#randomly sample the data if you dont want to use the whole set, -1 for all stuff
sample : -1

#query to apply to the data before separation
query : '(Leakage < 0.2) & (numPixelInShower > 8)'

# define th number of cross validations to perform
num_cross_validations : 5

# training_variables = ['Size', 'Length', 'Width', 'numIslands', 'Leakage',
#         'm3l', 'm3t', 'Disp', 'Timespread', 'arrivalTimeMean',
#        'ConcCore']
#specify the variables to train on
training_variables : ['ConcCore',
 'Concentration_onePixel',
 'Concentration_twoPixel',
 'Leakage',
 'Leakage2',
 'Size',
 'Slope_long',
 'Slope_spread',
 'Slope_spread_weighted',
 'Slope_trans',
 'Distance',
 'Theta',
 'Alpha',
 'Timespread',
 'Timespread_weighted',
 'Width',
 'arrTimePosShower_kurtosis',
 'arrTimePosShower_max',
 'arrTimePosShower_mean',
 'arrTimePosShower_min',
 'arrTimePosShower_skewness',
 'arrTimePosShower_variance',
 'arrTimeShower_kurtosis',
 'arrTimeShower_max',
 'arrTimeShower_mean',
 'arrTimeShower_min',
 'arrTimeShower_skewness',
 'arrTimeShower_variance',
 'concCOG',
 'm3l',
 'm3t',
 'maxPosShower_kurtosis',
 'maxPosShower_max',
 'maxPosShower_mean',
 'maxPosShower_min',
 'maxPosShower_skewness',
 'maxPosShower_variance',
 'maxSlopesPosShower_kurtosis',
 'maxSlopesPosShower_max',
 'maxSlopesPosShower_mean',
 'maxSlopesPosShower_min',
 'maxSlopesPosShower_skewness',
 'maxSlopesPosShower_variance',
 'maxSlopesShower_kurtosis',
 'maxSlopesShower_max',
 'maxSlopesShower_mean',
 'maxSlopesShower_min',
 'maxSlopesShower_skewness',
 'maxSlopesShower_variance',
 'numIslands',
 'numPixelInShower',
 'phChargeShower_kurtosis',
 'phChargeShower_max',
 'phChargeShower_mean',
 'phChargeShower_min',
 'phChargeShower_skewness',
 'phChargeShower_variance',
 # 'photonchargeMean'
 ]
```

C Notes on Implementation

The backend for the web interface is mainly bundled into the `RTAWebService` class. Most of the data from the stream is provided to the `RTAWebService` by the `RTAProcessor`. with one important exception: the `DataRate` processor needs to write the current data rate to the service. The data rate has to be measured at the start of the process chain. This was done to accomplish a regular updating data rate plot in the web interface. In the future it might be wise to simplify that step and simply bundle all the informaiton for the website into the `RTAProcessor`. The `RTAWebService` is also responsible for writing results to a database.

The frontend consists of several small javascript dependencies for visualization purposes. Data for the frontend (or any other application for that matter) is accessible through a REST interface. The data rate plot and the excess curves plot are independent modules managed by the node package manager (NPM). For deployment of node modules to the frontend, the Browserify tool is used. The visualization modules rely heavily on the D3 library. While writing this text, D3 version 4 was published with many new features and simplifications. The code for the RTA frontend was written for version 3 but it is planned to upgrade in the future.

All the plots are published a separate node modules and can be downloaded from GitHub.

Hexmap The camera visualization `https://github.com/mackaiver/hexmap`

Datarate Plot The animated plot of the current datarate `https://github.com/mackaiver/datarate_d3`

Lightcurve Plot The plot of excess rates of the `https://github.com/mackaiver/lightcurve_d3`

This way most of the frontend code is completely seperate from the main repository of the FACT Tools. The FACT-Tools are also open source and on GitHub: `https://github.com/fact-project/fact-tools/` For any additional information please contact me at `kai.bruegge@tu-dortmund.de`.

D Notes on Reproducibility

Given all needed data is available (which should be the case on the official thesis DVD) a simple `make` command in the `thesis` directory should suffice to build the whole thesis. This document was compiled using TexLive 2015 https://www.tug.org/texlive/.

All plotting scripts are written in Python 3.5.To use them it is necessary to execute `pip install .` in the `ma_scripts` folder. All dependencies are listed below.

```
algopy==0.5.3
appnope==0.1.0
arrow==0.7.0
astral==0.9
astropy==1.0.6
bokeh==0.11.1
boto3==1.3.1
botocore==1.4.25
click==6.6
cloudpickle==0.2.1
cycler==0.10.0
dask==0.9.0
decorator==4.0.10
distributed==1.10.2
docopt==0.6.2
docutils==0.12
drmaa==0.7.6
ephem==3.7.6.0
fact==0.5.0
ipykernel==4.3.1
ipython==4.1.2
ipython_genutils==0.1.0
ipywidgets==4.1.1
Jinja2==2.8
jmespath==0.9.0
joblib==0.9.4
jsonschema==2.4.0
jupyter==1.0.0
jupyter_client==4.2.2
jupyter_console==4.1.1
jupyter_core==4.1.0
-e
    git+git@github.com:fact-project/classifier_tools.git@2be8979f6e5fbef45569156fb66411c4af222cd5#egg
llvmlite==0.10.0
locket==0.2.0
-e
    git+git@bitbucket.org:mackaiver/ma.git@17148fd3044f2de68ce8d4073faf85c1e322de08#egg=ma_thesis
MarkupSafe==0.23
matplotlib==1.5.1
mistune==0.7.2
mpmath==0.19
msgpack_python==0.4.7
natsort==4.0.3
```

```
nbconvert==4.2.0
nbformat==4.0.1
networkx==1.11
notebook==4.2.0
numdifftools==0.9.14
numexpr==2.5.2
numpy==1.11.0
pandas==0.18.1
path.py==0.0.0
pexpect==4.0.1
pickleshare==0.5
Pillow==3.2.0
Pint==0.7.2
pltutils==0.1
psutil==4.2.0
ptyprocess==0.5
py==1.4.31
pycrypto==2.6.1
pyflakes==1.0.0
Pygments==2.1.3
pymongo==3.2
PyMySQL==0.6.7
pyparsing==2.1.4
pytest==2.9.1
python-dateutil==2.5.3
pytz==2016.4
PyYAML==3.11
pyzmq==15.2.0
qtconsole==4.2.1
requests==2.10.0
ruamel.yaml==0.11.11
s3fs==0.0.5
scikit-image==0.12.3
scikit-learn==0.17.1
scipy==0.17.1
Shapely==1.5.13
simple-crypt==4.1.7
simplegeneric==0.8.1
six==1.10.0
sklearn==0.0
sklearn-pandas==1.1.0
sklearn2pmml==0.8.7
SQLAlchemy==1.0.9
sympy==1.0
tables==3.2.2
tblib==1.3.0
terminado==0.5
toolz==0.8.0
tornado==4.3
tqdm==3.1.4
traitlets==4.2.1
uncertainties==2.4.8.1
```

E Abbreviations and Acronyms

DAC Digital Analog Converter

ERM Entity Relationship Model

CTA Cherenkov Telescope Array

KB Kilo Byte

EULA End User License Agreement

NACK Negative Acknowledgment

BVB Ballspielverein Borussia

ESRF European Synchrotron Radiation Facility

HTTP Hyper Text Transfer Protocoll

AGN active galactic nuclei

ADC Analog Digital Converter

RTA real time analysis

UDP User Datagram Protocoll

Ns Nano Second

GRB Gamma Ray Burst

LST Large Size Telescope

MST Medium Size Telescope

SST Small Size Telescope

JDK Java Development Kit

JVM Java Virtual Machine

FoV Field of View

SNR supernova remnant

DAQ data acquisition

SFB Sonderforschungsbereich

ZMQ Zero Message Queue

FACT First G-APD Cherenkov Telescope

SiPM Silicon Photomultiplier

TCP Transmission Control Protocol

PMT photo multiplier tube

IACT Imaging Atmospheric Cherenkov Telescope

XML Extensible Markup Language

IQR Inter Quartile Range

PCA Principal Component Analysis

ROI Region of Interest

DRS4 Domino Ring Sampler v4

CSV Coma Separated Values

SED spectral energy distribution

MAGIC Major Atmospheric Gamma Imaging Cherenkov Telescopes

HEGRA High-Energy-Gamma-Ray Astronomy

CORSIKA COsmic Ray SImulations for KAscade

API Application Programming Interface

PMML Predictive Model Markup Language

CERES Camera Electronics and REflectorSimulation

REST representational state transfer

JS Java Script

AJAX asynchronous JavaScript and XML

ECML European Conference on Machine Learning

UHE ultra-high-energy

GZK Greisen–Zatsepin–Kuzmin

SSC Synchrotron Self-Compton

EC External Compton

Bibliography

[1] A. A. Abdo et al. "Insights into the High-energy γ-ray Emission of Markarian 501 from Extensive Multifrequency Observations in the Fermi Era". In: *apj* 727, 129 (2011-02), p. 129. DOI: 10.1088/0004-637X/727/2/129. arXiv: 1011.5260 [astro-ph.HE].

[2] "Correlation of the Highest-Energy Cosmic Rays with Nearby Extragalactic Objects". In: *Science* 318.5852 (2007). Ed. by J. Abraham et al., pp. 938–943. ISSN: 0036-8075. DOI: 10.1126/science.1151124. eprint: http://science.sciencemag.org/content/318/5852/938.full.pdf. URL: http://science.sciencemag.org/content/318/5852/938.

[3] H. Anderhub et al. "Design and operation of FACT – the first G-APD Cherenkov telescope". In: *Journal of Instrumentation* 8.06 (2013), P06008. URL: http://stacks.iop.org/1748-0221/8/i=06/a=P06008.

[4] *Apache Flink is an open source platform for distributed stream and batch data processing*. URL: https://flink.apache.org/ (visited on 06/15/2016).

[5] C. Bockermann. "Mining Big Data Streams for Multiple Concepts". PhD thesis. TU Dortmund University, 2015. DOI: 10.17877/DE290R-16437. URL: https://eldorado.tu-dortmund.de/handle/2003/34363.

[6] C. Bockermann et al. "Online Analysis of High-Volume Data Streams in Astroparticle Physics". In: *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2015, Porto, Portugal, September 7-11, 2015, Proceedings, Part III*. Ed. by A. Bifet et al. Cham: Springer International Publishing, 2015, pp. 100–115. ISBN: 978-3-319-23461-8. DOI: 10.1007/978-3-319-23461-8_7. URL: http://dx.doi.org/10.1007/978-3-319-23461-8_7.

[7] M. Boettcher, D. E. Harris, and H. Krawczynski. *Relativistic Jets from Active Galactic Nuclei*. John Wiley and Sons, 2012.

[8] M. Boettcher et al. "Leptonic and Hadronic Modeling of Fermi-Detected Blazars". In: *Astrophys. J.* 768 (2013), p. 54. DOI: 10.1088/0004-637X/768/1/54. arXiv: 1304.0605 [astro-ph.HE].

[9] M. Bostock. *Data-Driven Documents*. URL: https://d3js.org/ (visited on 06/15/2016).

[10] M. Böttcher. “Models for the spectral energy distributions and variability of blazars”. In: *Fermi meets Jansky - AGN in Radio and Gamma Rays. Proceedings of a Workshop held 21-23 June, 2010 at the Max-Planck-Institut f. Radioastronomie, Bonn, Germany. Edited by Tuomas Savolainen, Eduardo Ros, Richard W. Porcas and J. Anton Zensus. Bonn: Max-Planck-Institut f. Radioastronomie, 2010.* Ed. by T. Savolainen et al. 2010-06.

[11] L. Breiman. “Random Forests”. English. In: *Machine Learning* 45.1 (2001), pp. 5–32. ISSN: 0885-6125. DOI: 10.1023/A:1010933404324. URL: http://dx.doi.org/10.1023/A%3A1010933404324.

[12] T. Bretz, D. Dorner, et al. for the FACT. “CheObs goes Monte Carlo”. In: *Proceedings of the 31th ICRC*. 2009.

[13] K. Brügge. “Evaluating throughput performance of the streams-framework on Cherenkov Telescope Array data”. Bachelorarbeit. TU Dortmund, 2015.

[14] R. Brun and F. Rademakers. “ROOT - An Object Oriented Data Analysis Framework”. In: *AIHENP’96 Workshop*. URL: http://root.cern.ch/.

[15] M. Claro. *Astrophotography by Miguel Claro*. URL: http://www.miguelclaro.com (visited on 06/15/2016).

[16] Dorner, Daniela for the FACT. *FACT Quick Look Analysis*. URL: http://fact-project.org/monitoring.

[17] D. Dorner et al. for the FACT. “FACT - Monitoring Blazars at Very High Energies”. In: *Fifth International Fermi Symposium Nagoya, Japan, October 20-24, 2014*. 2015. arXiv: 1502.02582 [astro-ph.IM]. URL: https://inspirehep.net/record/1343481/files/arXiv:1502.02582.pdf.

[18] Esoteric Software. *kryo Github Repository*. URL: https://github.com/EsotericSoftware/kryo.

[19] T. J. Hastie, R. J. Tibshirani, and J. H. Friedman. *The elements of statistical learning : data mining, inference, and prediction*. Springer series in statistics. PDF version of the book. 10th Print (January 2013). New York: Springer, 2009. ISBN: 978-0-387-84857-0. URL: http://statweb.stanford.edu/~tibs/ElemStatLearn/.

[20] D. Heck for the KASCADE. “Extensive air shower simulations with CORSIKA and the influence of highenergy hadronic interaction models”. In: *From e+ e- to Heavy Ion Collisions: Proceedings of the 30th International Symposium on Multiparticle Dynamics (ISMD 2000), Tihany, Hungary, October 9-15, 2000*. 2000, pp. 252–259. arXiv: astro-ph/0103073 [astro-ph]. URL: http://alice.cern.ch/format/showfull?sysnb=2244056.

[21] A. M. Hillas. “Cerenkov light images of EAS produced by primary gamma”. In: *International Cosmic Ray Conference* 3 (1985-08), pp. 445–448.

[22] J. D. Hunter. "Matplotlib: A 2D Graphics Environment". In: *Computing in Science and Engineering* 9.3 (2007), pp. 90–95. DOI: http://dx.doi.org/10.1109/MCSE.2007.55. URL: http://scitation.aip.org/content/aip/journal/cise/9/3/10.1109/MCSE.2007.55.

[23] L. Hyafil and R. L. Rivest. "Constructing optimal binary decision trees is NP-complete". In: *Information Processing Letters* 5.1 (1976), pp. 15–17. ISSN: 0020-0190. DOI: http://dx.doi.org/10.1016/0020-0190(76)90095-8. URL: http://www.sciencedirect.com/science/article/pii/0020019076900958.

[24] *The JSON Data Interchange Format*. Tech. rep. Standard ECMA-404 1st Edition / October 2013. ECMA, 2013-10. URL: http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf.

[25] H. Karttunen et al. *Fundamental astronomy*. Springer Science & Business Media, 2007.

[26] T.-P. Li and Y.-Q. Ma. "Analysis methods for results in gamma-ray astronomy". In: 272 (1983), pp. 317–324. DOI: 10.1086/161295.

[27] *Machine Learning for everyone*. URL: https://bigml.com/ (visited on 06/15/2016).

[28] W. McKinney. "Data Structures for Statistical Computing in Python". In: *Proceedings of the 9th Python in Science Conference*. Ed. by S. van der Walt and J. Millman. 2010, pp. 51–56. URL: http://pandas.pydata.org/.

[29] M. Noehte, D. Neise, and M. Müller for the FACT. *FACT shifhelper repository*. URL: https://github.com/fact-project/shifthelper.

[30] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[31] O. Ritthoff et al. *YALE: Yet Another Learning Environment*.

[32] V. Ruusmann. *Python library for converting Scikit-Learn models to PMML*. URL: https://github.com/jpmml/sklearn2pmml (visited on 06/15/2016).

[33] M. Sandri and P. Zuccolotto. "A Bias Correction Algorithm for the Gini Variable Importance Measure in Classification Trees". In: *Journal of Computational and Graphical Statistics* 17.3 (2008), pp. 611–628. DOI: 10.1198/106186008X344522. eprint: http://dx.doi.org/10.1198/106186008X344522. URL: http://dx.doi.org/10.1198/106186008X344522.

[34] Scikit-Learn. *Decision Trees. Tree algorithms: ID3, C4.5, C5.0 and CART*. URL: http://scikit-learn.org/stable/modules/tree.html#tree-algorithms-id3-c4-5-c5-0-and-cart (visited on 06/15/2016).

[35] T. Stanev. *High energy cosmic rays*. Springer Science & Business Media, 2010.

[36] A. Toshniwal et al. “Storm@Twitter”. In: *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*. SIGMOD ’14. Snowbird, Utah, USA: ACM, 2014, pp. 147–156. ISBN: 978-1-4503-2376-5. DOI: 10.1145/2588555.2595641. URL: http://doi.acm.org/10.1145/2588555.2595641.

[37] D. B. Tridon et al. “Performance of the Camera of the MAGIC II Telescope”. In: (2009). eprint: arXiv:0906.5448.

[38] S. v. d. Walt, S. C. Colbert, and G. Varoquaux. “The NumPy Array: A Structure for Efficient Numerical Computation”. In: *Computing in Science and Engineering* 13.2 (2011), pp. 22–30. DOI: http://dx.doi.org/10.1109/MCSE.2011.37. URL: http://scitation.aip.org/content/aip/journal/cise/13/2/10.1109/MCSE.2011.37.

[39] “When Victor Hess Discovered Cosmic Rays in a Hydrogen Baloon. On Its Centenary, Celebrating a Ride That Advanced Physics”. In: *NY Times* (2012-08-07).

[40] M. Zaharia et al. “Spark: Cluster Computing with Working Sets”. In: *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*. HotCloud’10. Boston, MA: USENIX Association, 2010, pp. 10–10. URL: http://dl.acm.org/citation.cfm?id=1863103.1863113.

Eidesstattliche Versicherung

Ich versichere hiermit an Eides statt, dass ich die vorliegende Abschlussarbeit mit dem Titel “Analysis of FACT Data” selbstständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt, sowie wörtliche und sinngemäße Zitate kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Ort, Datum	Unterschrift

Belehrung

Wer vorsätzlich gegen eine die Täuschung über Prüfungsleistungen betreffende Regelung einer Hochschulprüfungsordnung verstößt, handelt ordnungswidrig. Die Ordnungswidrigkeit kann mit einer Geldbuße von bis zu 50 000 € geahndet werden. Zuständige Verwaltungsbehörde für die Verfolgung und Ahndung von Ordnungswidrigkeiten ist der Kanzler/die Kanzlerin der Technischen Universität Dortmund. Im Falle eines mehrfachen oder sonstigen schwerwiegenden Täuschungsversuches kann der Prüfling zudem exmatrikuliert werden (§ 63 Abs. 5 Hochschulgesetz –HG–).

Die Abgabe einer falschen Versicherung an Eides statt wird mit Freiheitsstrafe bis zu 3 Jahren oder mit Geldstrafe bestraft.

Die Technische Universität Dortmund wird ggf. elektronische Vergleichswerkzeuge (wie z. B. die Software “turnitin”) zur Überprüfung von Ordnungswidrigkeiten in Prüfungsverfahren nutzen.

Die oben stehende Belehrung habe ich zur Kenntnis genommen.

Ort, Datum	Unterschrift