INSTITUT FÜR INFORMATIK
Computer Vision, Computer Graphics
and Pattern Recognition

Universitätsstr. 1 D–40225 Düsseldorf

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Classification of data from the ATLAS experiment

Michael Janschek

Bachelor Thesis

Beginn der Arbeit:	10. Dezember 2015
Abgabe der Arbeit:	10. März 2016
Gutachter:	Prof. Dr. Stefan Harmeling
	Prof. Dr. Stefan Conrad

Erklärung

Hiermit versichere ich, dass ich diese Bachelor Thesis selbstständig verfasst habe. Ich habe dazu keine anderen als die angegebenen Quellen und Hilfsmittel verwendet. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Düsseldorf, den 10. März 2016

Michael Janschek

Abstract

This bachelor thesis presents the task of event selection for data analysis performed by the ATLAS experiment at CERN. A good solution of this classification problem is a key to successfully claim a discovery in particle physics, like the Higgs Boson discovery in 2012.
The Higgs Boson Machine Learning Challenge, hosted on Kaggle, serves as an example for this task and is further investigated in this work. Several strategies are presented. The Gradient Boosting package XGBoost stood out from the others; achieving good prediction accuracy as well as excellent runtime. Though it did not provide the winning submission it was acknowledged by the competition hosts with a special jury award. As event selection is part of a budgeted data analysis system in ATLAS, runtime is an important factor for algorithms to be used in actual CERN applications. It is concluded that XGBoost has a good chance of replacing common tools in particle physics.

While there have been original tools created for testing and analysing several approaches to solve the task, no new classification algorithms have been designed or programmed in this work. The focus is set primarily on the analysis of existing classifiers provided by existing packages, like Logistic Regression Classification of scikit-learn, a machine learning toolbox for Python.

Keywords: Higgs boson, Kaggle, Machine learning, Classification, Data science, scikit-learn

Acknowledgements

I would like to express my gratitude to Professor Stefan Harmeling. With the deadline approaching, he responded quickly to any questions and gave helpful feedback to several ideas.
Assistance provided by Lukas Schoden and Erik Spiegelberg for this work's final formulation was greatly appreciated.

I am thankful to my parents, Christina Janschek and Peter Janschek, for their support and encouragement throughout my study of computer science and physics.

At last I wish to thank Kathrin Dorn as she offered a continues support and endless patience during the last months.

Contents

1 Introduction

In 1964, three teams of physicists predicted the existence of a sub-atomic particle to give mass to other elementary particles. This so-called *Higgs Boson* became a key requirement of the *Standard Model of particle physics*, a collection of theories that successfully predicted many other particles.

The European Organization for Nuclear Research (CERN[1]) approved the construction of the *Large Hadron Collider* (LHC) in 1994. This ring-shaped machine with a 27-km circumference has been designed to perform experiments investigating some of the most important questions in particle physics. Finding evidence to confirm or deny the existence of the Higgs Boson was a central aim of the experiments *ATLAS* and *CMS*.

After CERN had announced the discovery July 2012, Peter Higgs and François Englert received the Nobel Prize in Physics 2013 for predicting the Higgs Boson.

1.1 Data processing of ATLAS

With the configuration used during its first runtime from 2009 to 2013, the LHC produced *events* at a rate up to 40 million per second (40MHz). Each event contained at least one *Proton-Proton* (*pp*) interaction and hundreds of new particles. These were summarized in a raw-data vector of about a hundred thousand dimensions [ABCG+15b]. This data is reduced via three levels of hard- and software filters, so-called *triggers*, discarding all but the most interesting data. This process scales the output data of ATLAS down to about 400[2] compressed events per second, adding up to 320 Mbyte of data per second, which is then recorded. This raw data is reconstructed by ATLAS to whole events in several stages. The first stage is called *event selection*. In it, easily calculable information is derived from compressed events and is used to classify them for further analysis.
This event selection estimates the probability of an event being significant to the current task, like finding the Higgs Boson. If the probability is high enough, the event will enter the next reconstruction stage to calculate other previously cut features that need more computation time. This event is called *signal* and will be recorded as part of the so-called *selection region*. The other events are referred as *background* and are not processed further for the current task.
These stages of reconstruction work time-budgeted and only calculate necessary features of an event[3]. ATLAS extracts physics information from reconstructed events, containing only 0.1 Mbytes of data. This data describes actual physical objects, like particles and their trajectory.

[1] CERN is the abbreviation of *Conseil Europèen pour la Recherche Nuclèaire*, a council founded in 1952, assigned to establish a European research organization [CER16].

[2] Used sources are not consistent about the amount of events [ABCG+15b, Gli15, ATL16].

[3] The reconstruction of a full event was planned to cost 15 seconds, though this estimation relies on technical standards of 2008 [ATL16].

1.1.1 Claiming a discovery

In particle physics, the choice of the selection region is key for improving the *statistical significance* of a discovery. Given an observation that is assumed to be caused by a specified effect, this relation, the discovery, is considered *significant*, if the chance of it being caused by any other effect is less than the *significance level*, a previously determined value. For the successful discovery of the Higgs Boson, this value was $p = 2.87 \times 10^{-7}$ [ABCG+15b]. We can conclude that *improving statistical significance* is equivalent to reducing the chance of the observation being caused by background-events.

After the discovery in 2012, CERN started to strengthen the evidence for the Higgs Boson by proving various properties that a particle has to fulfil to be the predicted object of 1964. One property is called *Higgs to tau tau decay* ($H \rightarrow \tau\tau$) and was observed for the first time in 2013 [atl13]. However, this decay has not been proven yet, as the measurement's significance is still too low for a successful discovery[4] [HH15].

1.2 The Higgs Boson Machine Learning Challenge

In May 2014 CERN announced a challenge to be hosted on the data science platform Kaggle. Counting 1785 participating teams it was the most popular challenge on the website until early 2015.

1.2.1 Kaggle

Kaggle is an internet platform for data scientists, founded in 2010. Besides hosting challenges, its services include hosting public datasets, a job market for data scientists and "Kaggle Rankings", a scoreboard based on performances of community members in Kaggle's competitions. All these services are free to an individual member, but paid by businesses, e.g. for hosting a challenge. Further, Kaggle acts in a consultancy role to structure or prepare a challenge [kag16].

1.2.2 The leaderboard

In Kaggle challenges, competitors are ranked in a leaderboard. Rank 1 is the participant who submitted a solution which achieved the best result on the evaluation method used in this challenge. In our case a submission is evaluated by its *Approximate Median Significance* (AMS), a function derived in Chap. 2.
To prevent participants from simply training algorithms by directly optimizing the leaderboard-score, Kaggle uses so-called *public* and *private leaderboards*, which use different data subsets from the whole test set to generate the AMS. For every Kaggle challenge, the data is split by 30% for the public and 70% for the private leaderboard.

During the challenge, the public leaderboard is visible to any visitor of Kaggle, so participants are able to get an evaluation of their submitted solution and work on better classification for a higher rank. The private leaderboard, which shows the final ranking

[4] Status of March 27th 2015.

Benchmark	public AMS	private AMS
random submission	0.58477	0.58648
simple window	1.54451	1.53518
naive Bayes starting kit	2.06036	2.06021
simple TMVA boosted trees	3.24954	3.19956
Multiboost	3.34085	3.40488

Table 1: Benchmarks provided by CERN

of the challenge and the differences to the public leaderboard, is accessible after reaching the final submission deadline. Only the private rank is relevant for winning a Kaggle competition. In Chap. 4 we discuss the differences in this challenge's ranking. We will use the public AMS to evaluate our classification approaches as this might enable better insight to what might cause the differences.

For comparison, the competition hosts added various benchmarks to the leaderboard [Tab.1]. A tuned submission achieved private rank 782 by using *Toolkit for Multivariate Data Analysis with ROOT* (TMVA), a software widely used in High Energy Physics [ABCG+15a].

1.2.3 Goals

While the task of *The Higgs Boson Machine Learning Challenge* is to strengthen the observation of $H \rightarrow \tau\tau$ by training a classifier for event selection, another important goal of the challenge is to promote the collaboration of data science and physics. Over the next years, CERN expects hardware improvements that make much higher rates in data processing possible. However, to utilize the hardware, a considerable improvement of algorithm speed is required [Gli15]. Beside the three prizes going to the top-three submissions, a special jury award has been offered. The team that delivers an efficient classification model in terms of accuracy, simplicity and performance requirements is acknowledged with an invitation to meet the ATLAS team at CERN. This prize marks the important properties of algorithms that will be used to overcome the increasing amount of LHC data in the future.

1.3 Overview

In Chap. 2, we will describe the structure of the challenge's dataset [Hig15] and use simple data analysis methods, to gain first insight about useful features. After we derive the formal problem from the challenge's task, we conclude that chapter by explaining the evaluation metric AMS as objective function for optimizing our classifiers.

Chap. 3 presents basic data science methodology and discusses several dependencies we need to consider in the choice of some simple approaches for classification in Sect. 3.3 and 3.4. After these we will describe the more specific and complex methods *Neural Networks* and *Gradient Boosting Classification*.

In Chap. 4 we discuss the performance of the methods presented beforehand on the challenge's data. We consider the impact of *The Higgs Boson Machine Learning Challenge* on Kaggle and on data processing at CERN.

The thesis is concluded by Chap. 5. We recapitulate the whole work and review key competences for succeeding on Kaggle.
This is closed by a short personal view.

1.3.1 The source code

The source code created for this thesis has been uploaded to Github. This repository can be accessed under the URL:

`https://github.com/gargi/BA_git`

For information regarding the source code is found in Appendix A.

2 Understanding the challenge

The discovery of the Higgs Boson in 2012 received much attention from media all over the world, sometimes being called *the god-particle*. As publicity is a defined goal of the challenge, the good response to its announcement was no surprise.

This chapter presents the key properties of the challenge. In Sect. 2.1 the data will be described and first methods will be used to gain knowledge about it. As we continue with sections 2.2 and 2.3, we will formulate the classification problem to solve and study the evaluation the challenge uses to rank the participants. We follow the documentation of the challenge[ABCG+15b].

2.1 The data

All data provided by the challenge and the opendata.cern dataset [Hig15] was created by the official ATLAS full detector simulator in a two-part process. The simulator first produces proton-proton collisions, called *events*. Then it tracks these via a virtual model of the ATLAS-detector. The resulting data emulates the statistical properties of the real events. By this procedure it is possible to exactly know, if an event is a searched *signal*, or *background*. Signal-events are generated by previously mentioned *tau tau decay of the Higgs Boson* ($H \to \tau\tau$). Background-events originate from three known processes[5] which produce radiation similar to the signal [ABCG+15b]. The feature `Label` expresses the true class of an event, "s" for signals and "b" for background.

Every event has a feature `Weight`, a result of the simulation. The weights correspond to an estimation of the number of expected events of this class in a *region* during a fixed time interval[ABCG+15b].
For instance: A group of 100 events of both classes, "s" and "b", originates from the region A, for which we expect 90% of events to be background and 10% being signal. For another region B, we expect a 50/50 distribution. The weight of an event in a region is directly related to the expected probability of its occurrence. This results in a so-called *importance weighting* and causes the mean of the weights for signal-events to be considerably smaller than for background-weights. We will see, that the challenge evaluation utilizes this to punish incorrectly classifying background as signal.

The features `Weight` and `Label` were originally only provided in the training dataset. The data used in this thesis is expanded by complete `Weight`-, `Label`-features and the Kaggle-specific features `KaggleSet` and `KaggleWeight`. The last one is a normalization of `Weight` for the total number of signal- or background-events with the same `KaggleSet` feature. This information enables us to recreate the original datasets using the opendata.cern dataset.

The physical features are separated in two types. Features containing so-called primitive data, properties of events explicitly measured by the simulated ATLAS detector, use the prefix "`PRI_`" in their names. The second type is called derived data, which are features

[5] These processes are decay of Z bosons, W bosons and pairs of top quarks. There are more processes known that resemble ($H \to \tau\tau$), for simplicity only these three are used as background for this challenge [ABCG+15b].

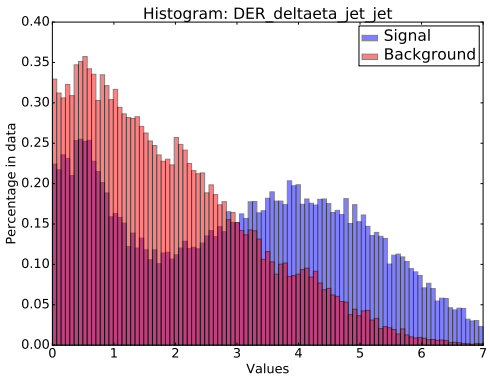


Figure 1: Histogram of *DER_deltaeta_jet_jet*

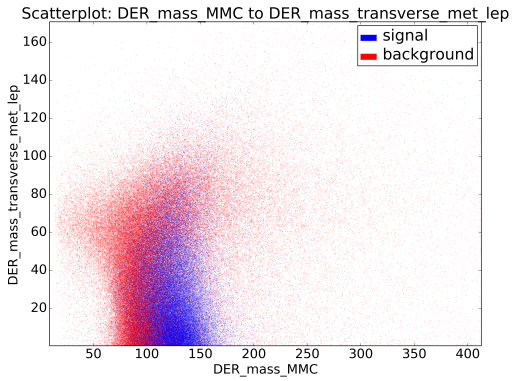


Figure 2: Scatter plot of *DER_mass_MMC* to *DER_mass_transverse_met_lep*

that have been computed from primitive features. Their labels use the prefix "DER_". Tab. 2 is a complete data vector retrieved from [Hig15].

One might expect decent knowledge in physics as key in succeeding in the challenge. However, the top participants did not use a lot domain knowledge for feature or method selection. One goal of the challenge was to set a task for data scientists without any background in physics [ABCG+15b].

2.1.1 Data visualization

To get first information about our data we use basic methods of data analysis. To evaluate single features for classification we plot a histogram of the class labels' distribution in Fig. 1. A useful way to learn about relations between features is to create *scatter plots* of all two-features combinations, see Fig. 2. Using these visualizations, we can identify features with good properties for our task.

2.2 The classification problem

We use the formal description of the challenge as in [ABCG+15b].
Let $D = (x_i, y_i, w_i),\ i \in \{1, \ldots, n\}$ be the training sample with n events, where:

- $x_i \in \mathbb{R}^d$ is a d-dimensional vector
- $y_i \in$ {b,s} is the label
- and $w_i \in \mathbb{R}^+$ is a non-negative weight.

Feature name	Feature value
EventId	100000
DER_mass_MMC	138.47
DER_mass_transverse_met_lep	51.655
DER_mass_vis	97.827
DER_pt_h	27.98
DER_deltaeta_jet_jet	0.91
DER_mass_jet_jet	124.711
DER_prodeta_jet_jet	2.666
DER_deltar_tau_lep	3.064
DER_pt_tot	41.928
DER_sum_pt	197.76
DER_pt_ratio_lep_tau	1.582
DER_met_phi_centrality	1.396
DER_lep_eta_centrality	0.2
PRI_tau_pt	32.638
PRI_tau_eta	1.017
PRI_tau_phi	0.381
PRI_lep_pt	51.626
PRI_lep_eta	2.273
PRI_lep_phi	-2.414
PRI_met	16.824
PRI_met_phi	-0.277
PRI_met_sumet	258.733
PRI_jet_num	2
PRI_jet_leading_pt	67.435
PRI_jet_leading_eta	2.15
PRI_jet_leading_phi	0.444
PRI_jet_subleading_pt	46.062
PRI_jet_subleading_eta	1.24
PRI_jet_subleading_phi	-2.475
PRI_jet_all_pt	113.497
Weight	0.00081448039868
Label	s
KaggleSet	t
KaggleWeight	0.00265331133733

Table 2: Event 100000 provided by opendata.cern data [Hig15]

The sum of signal weights S and the sum of background weights B

$$S = \sum_{y_i=s} w_i \qquad \text{and} \qquad B = \sum_{y_i=b} w_i$$

represent the *expected total number* of signal and background events, during the time of actual data recording.

Let a function

$$g : \mathbb{R}^d \to \{b, s\}$$

be a binary classifier. The set

$$G_s = \{x : g(x) = s\}$$

is called the *selection region*.

Our task is to find a function g that maximizes the AMS, that we introduce in the next section.
For a given classifier g we maintain an *index set* $\hat{G}_s = \{i : g(x_i) = s\}$ of training points that g classifies as signal and an *index set* $\hat{G}_b = \{i : g(x_i) \neq s\}$. By definition each point can only be one index set, i.e. $(\hat{G}_s \cap \hat{G}_b = \emptyset)$.

The index set can be turned into a submission file for the challenge, but requires to fit the following format.

```
EventId,RankOrder,Class
350000,2,b
350001,54493,s
...
899999,32455,b
```

All events must have a prediction `b` or `s`. `RankOrder` shall state the probability of an event being the signal `s`, compared to all other events, e.g. rank `550000` is being considered as the most signal-like event. However, predicting the right event ranking was not necessary for the challenge.

2.3 The evaluation

The evaluation of a single submission file to the challenge is based on the previously mentioned *significance level*. We define the *significance*

$$Z = \sqrt{2\left(n \ln\left(\frac{n}{\mu_b}\right) - n + \mu_b\right)} \quad , \tag{1}$$

where n is the total number of observed events and bigger than μ_b, which is the expected number of events produced by background effects.

In other words, we investigate a region in feature space and *expect* a number μ_b of events. The number n is what we *actually* observe in this region. In particle physics, a significance

of at least $Z = 5$, called a *five-sigma effect*, is regarded as sufficient to claim a discovery . This is equivalent to the previously stated *significance level* $p = 2.87 \times 10^{-7}$ [ABCG+15b].

We can substitute n with $s + b$ and μ_b with b and transform Eq. (1) to the *Approximate Median Significance* (AMS)

$$\text{AMS}' = \sqrt{2\left((s+b)\ln\left(1+\frac{s}{b}\right)-s\right)} \quad , \tag{2}$$

which resembles an estimation function that is used by high-energy physicists for optimizing the selection region for stronger discovery significance [ABCG+15b].

For the challenge, a regularization-term b_{reg} was introduced as an artificial shift to b to decrease variance of the AMS, as this makes it easier to compare the participants if the optimal signal region was small. The challenge's documentation provides no further information on the choice $b_{reg} = 10$ [ABCG+15b].

This addition to Eq. (2) makes the final evaluation-formula complete:

$$\text{AMS} = \sqrt{2\left((s+b+b_{reg})\ln\left(1+\frac{s}{b+b_{reg}}\right)-s\right)} \tag{3}$$

For simplicity, we will call it just AMS, as Eq. (2) will not have further appearances in this thesis.

2.3.1 AMS as Objective Function in Classification

In data classification, an *objective function* is a tool to estimate the accuracy of a fitted classifier. To find a good classifier we directly try to optimize the AMS.

If we partially derive Eq. (3) with respect to s, we get

$$\frac{\partial}{\partial s}\text{AMS} = \frac{\frac{s+b+b_{reg}}{(b+b_{reg})\left(\frac{s}{b+b_{reg}}+1\right)} + \ln\left(\frac{s}{b+b_{reg}}+1\right) - 1}{\sqrt{2\left((s+b+b_{reg})\ln\left(1+\frac{s}{b+b_{reg}}\right)-s\right)}} \quad . \tag{4}$$

The complexity of Eq. (4) is the reason why most participants of the challenge avoided the AMS as a primary objective function. After the actual prediction, which is made by the classifier fitted to another objective, the events are ranked according to their likelihood of being a signal. The AMS can now be used to determine an optimal decision threshold for the final classification. For most submissions, given their ranking works correctly, classifying the top 14% of most likely events as signal resulted in optimal AMS [ABCG+15a]. Many participants simply substituted the AMS with common objective functions like *squared error* that are computational efficient. Some investigated AMS further and found related functions with beneficial properties, like easier derivation [Kot15, MBM15, DMnNV15].

3 Methods of classification

There are countless possible ways in data science to classify data. The efficiency of methods varies in wide ranges for different types of problems.

In Sect. 3.1 we learn about methods that are commonly used by data scientists. Following this, we discuss criteria for the choice of classifiers in Sect. 3.2. Based on this we present *Logistic Regression* in Sect. 3.3 and *k Nearest Neighbors* in Sect. 3.4 as our approaches to the challenge. This is followed by the winning submission by Gábor Melis in Sect. 3.5. The chapter ends with Sect. 3.6, presenting *XGBoost* as winner of the *HEP meets ML Award*.

3.1 Basic data science methodology

Before we learn about criteria for choosing a good classification method, we present basic methods in data science. The performance of our approach can be influenced by the right use of these procedures.

3.1.1 Feature selection and engineering

An intuitive approach to optimize classifiers is to vary the used features of the data sets. The selection can be performed manually by considering e.g. histograms (Fig. 1) , scatter plots (Fig. 2), performance of various classifiers or automatically using techniques like PCA. Many automatic procedures tend to exclude features based on max errors, while the challenge aims to maximize AMS. Using such methods risks losing important features[win14].

In the data, the value *-999.0* is often set for certain features. This is a flag for *missing* measurement in an event as sometimes a property simply cannot be measured, because fundamental effects do not happen. The defining feature for missing values is `PRI_jet_num`, which is equal to the number of jets measured in a *pp* collision. Given an event, where `PRI_jet_num` equals *0*, there are 10 features that do not exist, only missing values in `DER_mass_MMC` are not related to this feature. The whole original dataset contains 567329 events with at least one feature missing, excluding missing `DER_mass_MMC`. This equals 70.9% of all events and has to be considered for some classifiers. We bypass this problem by cutting these features for most methods. For simplicity, we only use the feature subsets presented in Tab. 3 and the complete set of the original data.

These sets are created before and during classification, the latter as part of optimization efforts. Sets 1 and 2 are chosen beforehand as a result of the visualization of the features, e.g. `DER_mass_MMC` seems to be well separable, so it is added to most sets although it has missing values. Sets 3,4 and 5 result of an ensemble of 50 *k Nearest Neighbors* classifiers, which will be described in Sect. 3.4, each choosing 5 random features. These sets contain a number of the common features used for the best predictions generated by this ensemble. The Sets 6 and 7 are chosen by hand to improve *k Nearest Neighbors*. Set 8 results of an ensemble of *Logistic Regression* classifiers (Sect. 3.3). Like before, we choose the common features used for the best predictions of the ensemble. Set 9 is the only feature set suggested by a competitor in the challenge [win14].

Feature	Set 1	Set 2	Set 3	Set 4	Set 5	Set 6	Set 7	Set 8	Set 9
DER_mass_MMC	■	■	■	■	■	■	■		■
DER_mass_transverse_met_lep	■	■	■	■	■	■	■	■	■
DER_mass_vis	■		■	■	■	■	■	■	■
DER_pt_h	■	■	■	■	■	■	■	■	■
DER_deltaeta_jet_jet			■						■
DER_mass_jet_jet			■	■					■
DER_prodeta_jet_jet									■
DER_deltar_tau_lep								■	■
DER_pt_tot								■	■
DER_sum_pt			■	■	■		■		■
DER_pt_ratio_lep_tau	■		■			■	■	■	■
DER_met_phi_centrality	■					■	■		■
DER_lep_eta_centrality									■
PRI_tau_pt	■	■	■	■	■	■	■		■
PRI_tau_eta									
PRI_tau_phi								■	
PRI_lep_pt								■	■
PRI_lep_eta			■	■					
PRI_lep_phi			■	■				■	
PRI_met		■							■
PRI_met_phi									
PRI_met_sumet									■
PRI_jet_num			■	■	■	■	■		■
PRI_jet_leading_pt									■
PRI_jet_leading_eta									■
PRI_jet_leading_phi									
PRI_jet_subleading_pt									■
PRI_jet_subleading_eta			■						■
PRI_jet_subleading_phi									
PRI_jet_all_pt			■	■	■		■		■
Total number of features	7	5	14	11	8	8	10	9	23

■ Feature = -999.0 if PRI_jet_num equals 0 or 1
■ Feature = -999.0 if PRI_jet_num equals 0

■ Used feature for Set

Table 3: Feature subsets used by our approaches

Because of its origin in physics, the calculation of more topic-relevant features seems as a promising use of domain-knowledge. However, the top-participants of the challenge shared the observation that artificial features did not improve the AMS significantly [Mel14, Sal14, Cou14].

3.1.2 Cross-validation

Overfitting a model is a known problem in machine learning. In the leaderboards we can observe single submissions to the challenge that had big differences in ranks on the public and private leaderboard. This could be explained by overfitted classifiers, in one case a submission fell 720 ranks from its public to private rank.
A common technique to prevent overfitting is to use only one part of the training set for fitting the classifier while using the other one for testing, calculating a evaluation score, like *mean squared error* (MSE). This method is called *cross-validation*. Multiple contestants stated, that simple cross-validation is not able to reliably estimate the submission's AMS. A solution to this is to repeat this procedure with different splits of the training data, which increases the runtime of the estimation by cross-validation.

3.2 Choosing the classifiers

In order to choose an effective approach to a solution for the challenge's task, we need to consider several dependencies of a classifier.

3.2.1 Computational resources

One basic limitation of our choice are the available resources. We want our classification to efficiently use the system represented by Tab. 4.

Operating System:	Windows 8.1 64bit
CPU:	Intel i7 K875 @ 2.93GHz (4 Cores)
Memory:	16GB DDR3
GPU:	NVIDIA GeForce GTX470 @ 1.22GHz
VRAM:	1280 MB GDDR5

Table 4: Specifications of used System

While this hardware enables us to use various methods that utilize minor parallelization, it still limits the performance of classifiers dependent on high parallelization, like neural networks. If not stated differently, all presented data on *runtime* in this thesis is based on the system represented by Tab. 4.

3.2.2 Dependency on data

The choice of classifiers is dependent on the amount of data. The training set contains 250000 samples with 32 features (EventId not counted), the test set 550000 samples with 30 features. Considering this, some classification methods would be ineffective, e.g. the

implementation of *Support Vector Classification* in scikit-learn, which is quadratic in the number of examples [sci16a].

3.2.3 scikit-learn

For the approaches we implemented, we will use version 0.17 of *scikit-learn*, an open source machine learning library for Python. Originated from a "Google Summer of Code" project started 2007, it grew into a popular toolkit [sci16a]. Using scikit-learn has advantages and disadvantages, Tab. 5 considers the most important ones.

Sect. 3.3 and 3.4 describe our approach.

Advantages	
Usability	Scikit-learn is easily installed via Python-package manager systems like *pip* or *Anaconda*, its required dependencies are NumPy, SciPy, and a working C/C++ compiler[sci16b]. It is compatible with Windows 8.1 64bit.
Scale and documentation	A reason for its popularity is the amount of common algorithms this toolkit contains. It also offers good code-documentation and mathematical explanation of all models included.
Multi-core CPU support	With every version, scikit-learn improves its multi-core support for algorithms that profit of parallelization[sci16c]. This will speed up our classification considerably.
Disadvantages	
No GPU support:	While the library utilizes multi-core CPUs, GPU-parallelization is not supported by scikit learn.
(Almost) no custom scoring functions	Scikit-learn offers rarely an easy way to use custom evaluation functions for e.g. fitting a classifier. Some algorithms, like *sklearn.cross_validation*, offer a choice of prebuilt functions via a *scoring* parameter.
Speed	One focus of the development of the toolkit is to optimize and speed up the algorithms, but projects devoted to single learning techniques are probably faster.

Table 5: Comparison of scikit-learn's properties

3.3 Logistic Regression

The first model we use is *Logistic Regression*. As this implementation uses *coordinate descent* of the *LIBLINEAR* library to solve optimization problems, we expect this classifier to have a short runtime [CHL08]. This enables us to test basic assumptions about the data without spending too much time on this method, in case it does not perform well.

3.3.1 Classification method

Basically, this classifier scales each dimension of the n-dimensional data. We write a scaled datapoint x as linear combination

$$g(x) = w_1x_1 + w_2x_2 + ... + w_nx_n + c \quad , \tag{5}$$

where $x = x_1, x_2, \cdots, x_n$ is a datapoint, $w_1, w_2, ..., w_n$ are the weights for n features and c adds a *bias*. Eq. (5) can be compressed to

$$g(x) = x^T w + c \quad , \tag{6}$$

using vectors for data and weights.
Using a logistic function h, the classifier predicts the probability of a datapoint being of class *signal* or *background* based on their distance to a linear function. This function marks the points z, where h predicts equal probability for both classes, i.e. $h(g(z)) = 0.5$.
Using L2-regularization, this classification is fitted by optimizing

$$\min_{w,c} ||w||_1 + C \sum_{i=1}^{m} \log(\exp(-y_i(X_i^T w + c)) + 1) \quad , \tag{7}$$

where X_i is n-dimensional data of an event i with the *true label* y_i. A vector w weights all n features of X_i and c adds a *bias*. A positive, real number C is chosen by the user to set regularization strength [sci16a].

3.3.2 Performance and optimization

Logistic Regression achieves an AMS of $\approx$ 2.0 by using the full datasets with all features and tuned parameters. The first approach for optimization is feature selection. We achieve the best prediction using Set 8, which uses common features of best predicting runs of Logistic Regression with random feature sets. The most promising explanation for this set's success is, that the set contains no features with missing values. Other tests with more features without missing values fail to increase the AMS scores. As other possible improvement of the method, the *logisticRegressionCV* class from scikit-learn allows us to use a custom scoring method via cross-validation. For that purpose, we choose *"ROC AUC"* (AUC). During the challenge, competitors used AUC as an alternative to AMS as objective function for optimization, due to the task's relation to a ranking problem. However, AUC- is not equivalent to AMS-optimization [ABCG+15a]. In practice, these tuning efforts fail to achieve higher AMS. An explanation for this poor performance is the noise heavy data we want to classify.

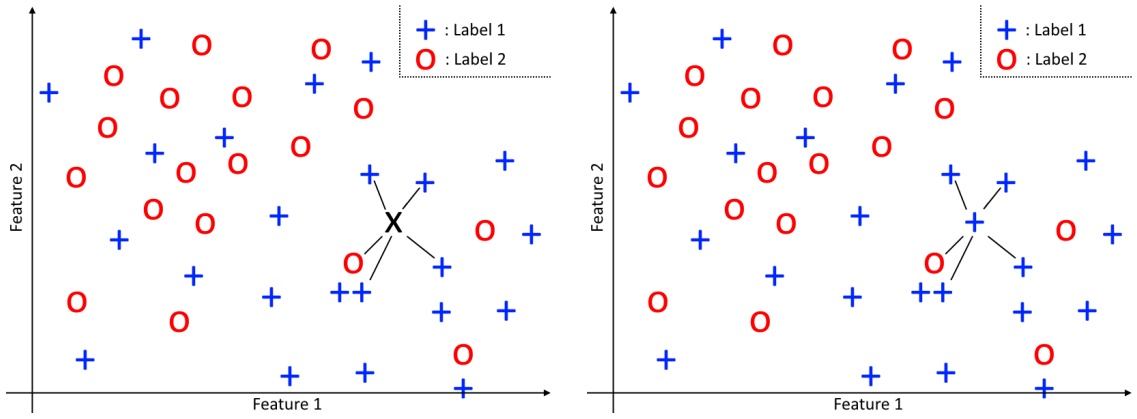
3.4 K Nearest Neighbors classification

Our next choice is *k Nearest Neighbors* (kNN), as it is a common and intuitive classification method.

3.4.1 Classification method

To describe the classification, we divide it into two steps. Like all classifiers of scikit-learn, first we *fit* the model with training data, then we *predict* the labels of our test data.

Other than most models, a kNN-classifier is fitted by simply projecting the n-dimensional data in an n-dimensional feature space. The computation heavy step is the prediction. The probability of a data point from the test set is estimated by a simple vote of the k nearest neighbors. The proximity of a point to this test data point is measured in their *Euclidian distance*. We create Fig. 3 to visualize the prediction method of kNN. By choosing odd-numbered k, we avoid possible even vote results.



(a) Insert new datapoint and find k (here =5) nearest neighbors.

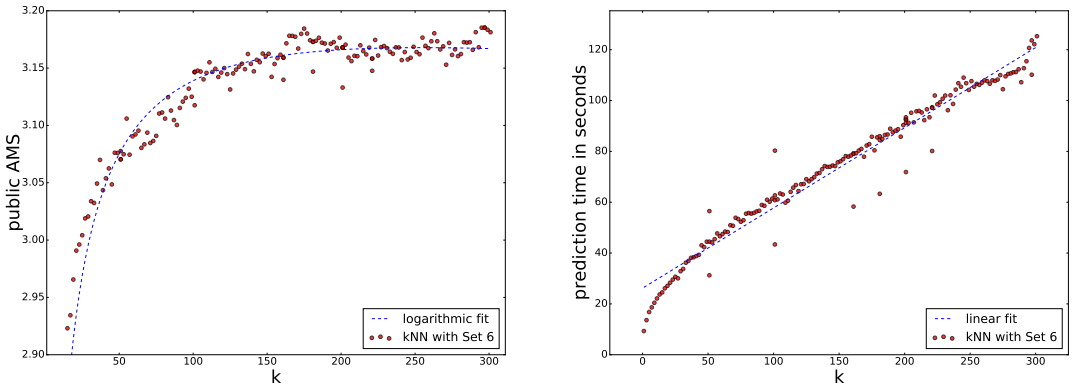
(b) Predict datapoints label by majority vote.

Figure 3: Visualization of kNN-classification with k=5

3.4.2 Performance and optimization

Compared to logistic regression, kNN achieves with default parameters and the complete data a much higher AMS. We can improve our score mainly with changing two parameters. The first option is to use feature selection on our data, as kNN is highly influenced by this choice. This also improves the runtime of the classifier, as the calculation of distances between vectors increases with their dimensionality. For testing these selections, low k like 10 or 20 are sufficient. After we find a good feature set, tuning k is the second parameter kNN mainly relies on. Larger k suppresses noise, but softens the classification boundaries and increases runtime. The sets we created with help of kNN ensembles result in worse scores than our hand-crafted sets. We achieve the best public AMS with feature set 6 and $k = 297$ after 123.67 seconds prediction time. The relation of k, prediction time and the resulting public AMS is presented by Fig. 4a and 4b, which uses data we recorded during testing.

Our results surpass those by another team that uses kNN. They identify the replacement of missing data values by the features mean as main error [PPSS14]. Considering that our optimal feature sets (Tab. 3) contain none of `PRI_jet_num`-related data, except `PRI_jet_num` itself, support this assumption.



(a) kNN results on Set 6

(b) kNN prediction time on Set 6

Figure 4: kNN performance on feature set 6

3.5 The winning submission

The three top submissions of the challenge all use classification methods that rely heavily on strong hardware. Without the ability to recreate these submissions or use various parameter changes, we focus solely on understanding the winning model, relying on [Mel15].

3.5.1 Neural Networks in general

A *neural network* is a type of machine learning model that was inspired by central nervous systems, like the human brain. Blocks or *neurons* are basic mathematical operations, like simple linear or bias functions. These neurons are grouped into so-called *layers*, which can be interpreted as weighting functions. The weights are optimized via loss functions, like *Squared Error*, during the training phase of this learning model. Usually neural networks have an *input layer*, a number of *hidden layers* and an *output layer*. A user or an external algorithm can only interact through the input layer, delivering data to the network, and the output layer, receiving an estimation on this data.
A neural network is fitted with so-called *backpropagation*. To understand this procedure, we use a mathematical expression of neural networks with one hidden layer, following [Har14]:

$$
\begin{aligned}
& z_1 = x && \text{input layer} \\
& z_2 = f_1(w_1, z_1) && \text{second layer (hidden)} \\
y = \quad & z_3 = f_2(w_2, z_2) && \text{output layer} \\
& \\
& E = \tfrac{1}{2}(y-t)^2 && \text{squared error (loss function)} \quad ,
\end{aligned}
\tag{8}
$$

where f_1 and f_2 are a differentiable functions. While the weight w_1 and the value z_1 are input of f_1, z_2 is its output. Analogous, f_2 produces the networks output, using w_2 and

z_2. E is a loss function, here *squared error*.
The output of a layer is highly dependent on this layers input. Therefore, changing a weight of a previous layer influences all following layers. If we differentiate the loss function E w.r.t. w_1 by applying the chain rule, we can observe its connection to following layers.

$$\frac{\partial E}{\partial w_1} = \frac{\partial E}{\partial z_3}\frac{\partial z_3}{\partial z_2}\frac{\partial z_2}{\partial w_1} \tag{9}$$

Backpropagation uses this connection and tunes the weights of each neuron of each layer. For instance, the layer f_2 receives the loss function derivative w.r.t. this layers output, i.e. $\frac{\partial E}{\partial z_3}$. It calculates the resulting derivation of E w.r.t. w_2:

$$\frac{\partial E}{\partial w_2} = \frac{\partial E}{\partial z_3}\frac{\partial}{\partial w_2} f_2(w_2, z_2) \tag{10}$$

The layer uses $\frac{\partial E}{\partial w_2}$ to update the weights with *gradient descent* at a sometimes automatically chosen regularization strength η. The weight approaches an optimal value with respect to the loss function.

In practice, the derivation for each layer is known and stored in the code, we only recalculate values resulting from these functions. The ability to highly parallelize backpropagation by using GPUs is the reason for its increasing popularity during the last years.

3.5.2 The method

The model developed by Gábor Melis uses an ensemble of 70 neural networks with three hidden layers containing 600 neurons each. The output layer consists of two *softmax* units, which basically normalize the weight-combinations to get a prediction for each class. Each neural network is fitted by a different split of the training data that is normalized to mean 0 and variance 1 and different weight-initializations. The hidden layers utilize *Local Winner-Take-All activation*, which cancel all weighted input values received by three blocks except for the maximum. The weighting is optimized with respect to a cross-entropy objective function using backpropagation with stochastic gradient descent. The winning submission, which has been generated by this classifier, outperforms follow-up submissions by over 0.015 AMS. It uses artificial features and was created within a runtime of 10 minutes on a GTX Titan GPU [Mel15].

3.6 XGBoost

One noteworthy classification method is *Gradient Boosting* and its implementation by Tianqi Chen and Tong Hel via the package *XGBoost*. After sharing it with other participants early in the challenge, they were rewarded with a special jury-award, for providing "a good compromise between performance and simplicity"[CGG+15]. While describing XGBoost, we evaluate this method by these criteria. For our testing, we use the python package of XGBoost version 0.47 available at `https://github.com/dmlc/xgboost`, released on 15. Jan. 2016.

3.6.1 Boosting

Assuming, we fit a model to solve a problem. A classifier is called a *strong learner* if it performs well and its prediction achieves a high accuracy. Therefore, a *weak learner* is a classifier that performs poorly, but learns from a training set, the prediction is not made by random guessing. *Boosting* is the idea to combine a set of weak learners into a single strong learner.

Given training data x with true label y, we fit a weak learner $f_1(x)$, which predicts the label $\hat{y}$.

$$y = f_1(x) + z \quad , \tag{11}$$

where z is the prediction error $|y - \hat{y}|$.
We replace the error with a second classifier in Eq. (11) such that

$$y = f_1(x) + f_2(x) \quad , \tag{12}$$

or equivalently

$$f_2(x) = y - f_1(x) \quad . \tag{13}$$

Considering we create an ensemble of weak learners, f_2 will not correct f_1's prediction perfectly. To approximate a good *fix* we fit f_2 not to the training data x, but to the error $y - f_1(x)$, the so-called *residual*.
In words: Because our solution is not perfect, we expect an error in our prediction. Therefore, we train another model to fit this error and repeat this process a set number of times.

3.6.2 Classification method

Boosting can be trained by minimizing an objective function

$$L = \sum_{j=1} l(y_j, F(x_j)) \quad , \tag{14}$$

where l is a loss function, y_j the *true* label of x_j and F is an ensemble of weak learners $\{f_1, f_2, \ldots, f_n\}$. If we use *square loss* as loss function l and differentiate this w.r.t. a weak learner $f_1(x_j)$ of the ensemble such that

$$\frac{\partial l}{\partial f_2(x_j)} = y - f_1(x_j) \quad , \tag{15}$$

we can observe that $-\frac{\partial l}{\partial f_2(x_j)}$ equals the residual $f_2(x) = y - f_1(x)$. Thus, residuals of a Boosting model can be interpreted as negative *gradients* that allow minimization of the loss function using *gradient descent*. This is called *gradient boosting*. Further, one can show that a gradient boosting algorithm can be constructed for *any* loss function [Li].

The more complex a classification model is, the higher is the chance of overfitting. Considering this, the main difference of XGBoost to other gradient boosting classifiers is the simple idea to penalize complex models in the ensemble. This is achieved by adding a regularization term $\Omega(f)$, which measures the complexity of f, to the objective function Eq.(14). Training this classifier minimizes not only the loss function, but also $\Omega(f_t)$. The

actual predictions are made by weighted *decision trees* with a chosen maximum depth, which are initialized with random values.

We visualize a tree in Fig. 5 using XGBoost's *plot_tree* method. It is one of 100 trees fitted to the original training data with a maximum depth of 3.

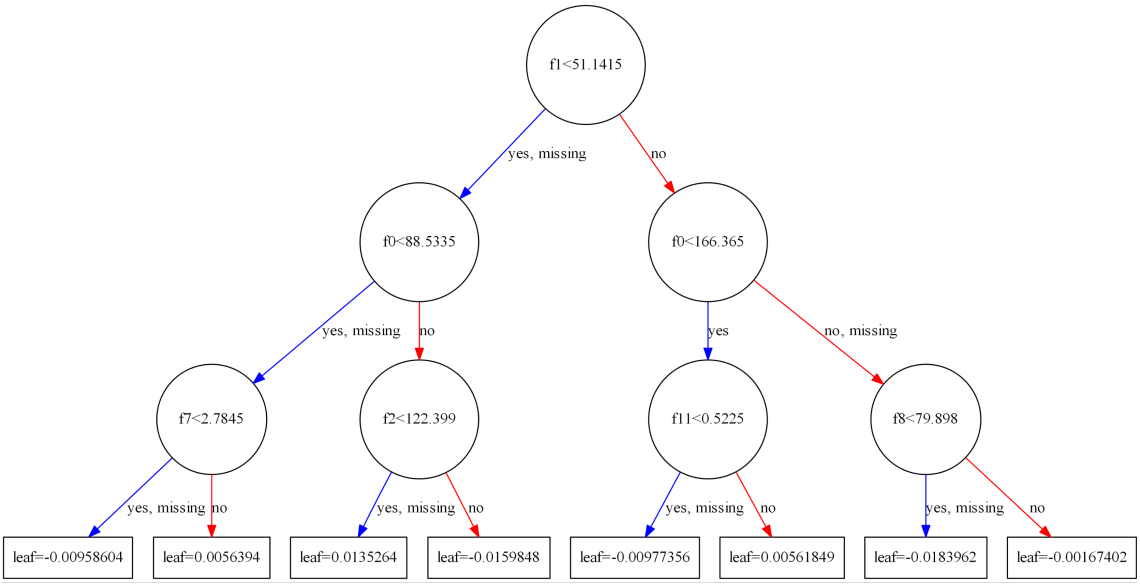


Figure 5: A decision tree generated by XGBoost.

3.6.3 Performance and optimization

Using default settings, 100 trees and AMS evaluation, XGBoost achieves a public AMS of about 3.32 using the complete, original data set. We get significantly better results by tuning parameters with respect to the public AMS, mainly increasing the number of trees up to 3000 and lowering *eta*. The latter parameter determines the *step size shrinkage* to prevent overfitting. Testing shows that *eta* is inversely related to the tree number, the more trees are used for training, the lower *eta* should be. Using all parameters, we are still not able to reproduce the exact AMS on the original data stated in [CH15]. Cutting phi-values, which is suggested by some competitors that use XGBoost [win14], does not improve the results. Further feature selection fails to resolve into better AMS. Adding artificial features proved beneficial for several submissions using the method.
In its recent version, XGBoost provides several alternative objective functions and evaluation metrics. As the package has been developed for the Kaggle challenge, it also contains AMS and AUC as metrics. While the original submission used AUC as model evalutation, we use AUC *and* AMS for training XGBoost with no noticeable disadvantages.

Many software packages implement gradient boosting, as it is an established method in machine learning. While Fig. 6 compares speed benchmarks, with all algorithms set up to fit 120 trees with depth 6 and *eta*=0.1, we consider AMS as it is the main goal of the challenge. Since our approaches rely on scikit-learn, we compare the performance of the

GradientBoostingClassifier(GBC) class to XGBoost , see Fig. 7.
The training GBC consumes over 5000 seconds, while using 100 trees of depth 12. Considering the long training time, we do not perform further training of GBC with more trees. However, it is worth mentioning that the longest training, that was performed on a XGBoost classifier for this thesis, terminated after 1666 seconds, calculating 3000 trees with a depth of 12. By preprocessing the mathematical model and using decision trees, the training complexity for a tree is reduced to $O(ndK)$ for n samples with d features, where K is the maximum depth of a decision tree[CH15].

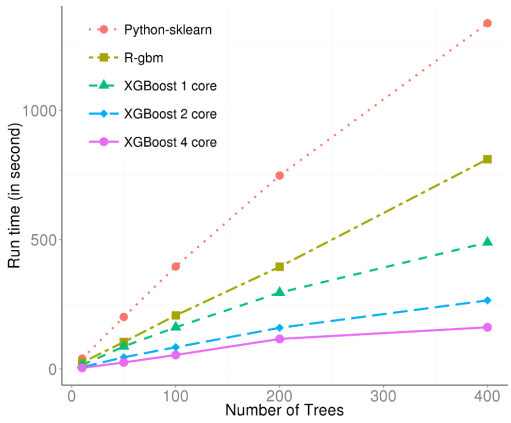


Figure 6: Speed Benchmark on challenge data [CH15]

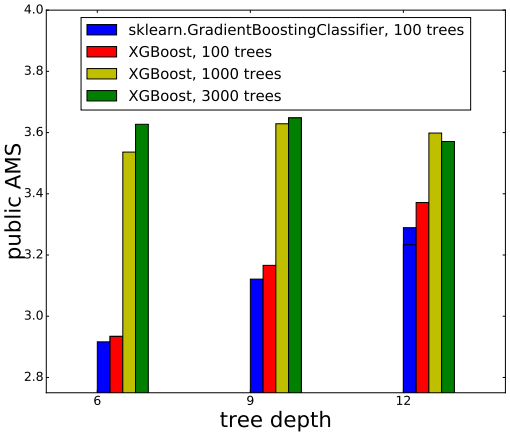


Figure 7: AMS-Comparison XGBoost and sklearn.GradientBoostingClassifier

Regarding the special award, XGBoost seems to fulfil each of its defined aspects. While many other submissions outperform the best benchmark *MultiBoost* easily as well, this package does so remarkably quickly. Training time for a benchmark surpassing result takes under 37 seconds with tuned parameters, using up to 8 CPU threads. This specific run achieved a public AMS of 3.5399. The developers made the package available early in the challenge and many other participants used it at some point to test custom feature sets or benchmark their own approaches.

It is justified to acknowledge this package, as it could have impact on tools, which are being used in high energy physics[CGG+15].

4 Impact of the challenge

We only presented some of the many approaches undertaken by the participants. There are just too many strategies to examine every single one. Many good ideas, some of them maybe groundbreaking, are lost or even discarded by their designers because they receive a worse evaluation than common models.

After comparing the challenge performances of the previously presented methods in Sect. 4.1, this chapter reviews the challenges impact on Kaggle and CERN.

4.1 Comparison of methods

Tab. 6 lists the best results we achieve and their hypothetical performance in the challenge is visualized by Fig. 8. To create this figure, we access the leaderboard data directly from Kaggle by using a Python script and plot the ranks of each leaderboard to their corresponding AMS.

As stated in Sect. 1.2.2, we choose our best submissions by their public AMS, due to the hidden private leaderboard in the running challenge.

Classifier	public AMS	private AMS	private rank
Logistic Regression*	2.04480	2.06934	1429
k Nearest Neighbor*	3.16941	3.18323	996
sklearn.GradientBoostingClassifier	3.44038	3.45097	838
XGBoost	3.66421	3.71268	65
XGBoost original submission	3.72435	3.71885	45
Winning submission (Neural Network)	3.85059	3.80581	1

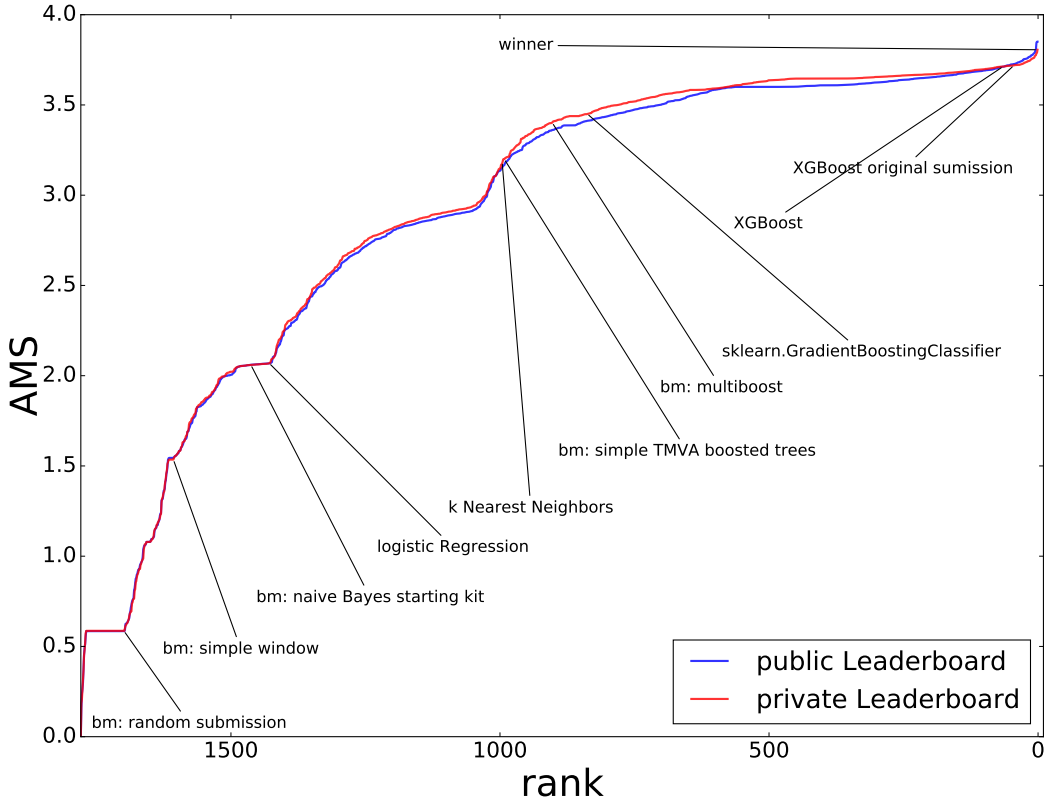
XGBoost original submission and *Winning submission* have been received from the official leaderboards and added as comparison, our approaches are marked with *.

Table 6: Best performances of used classification methods

4.1.1 Our approaches

The first approach, Logistic Regression, only achieved poor AMS results. As opposed to other presented classifiers, this method responds best to Set 8. This is not that surprising as Set 8 was crafted using an ensemble method. Its top submission is able to surpass the *Naive Bayes starting kit* benchmark by only 0.00913 AMS. Both classifiers are considered as linear models, which can be an explanation for the resemblance of their AMS. As Naive Bayes is used as benchmark in the leaderboards, one can assume an AMS of about 2.1 is the top limit for linear classification. However, few competitors reported to the forums that they used linear models, in their cases Support Vector Machines. The best submission of these classifiers resulted in a public AMS of 2.7827 [kag14c].

Our other approach, k Nearest Neighbor(kNN) classification, is able to outperform linear models significantly and scored rank 996 on the private leaderboard. While this result is



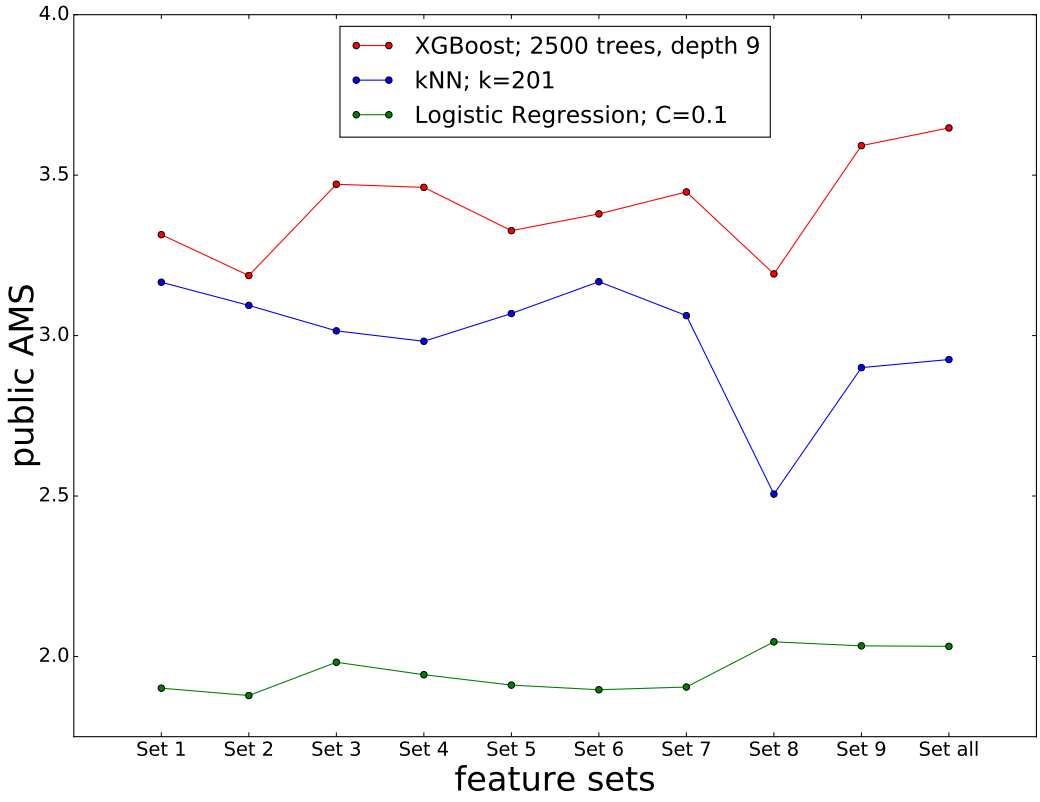
Benchmarks placed by challenge organizers have prefix bm.

Figure 8: Public and private AMS plotted to equivalent rank

still within the lower half of ranked submissions, it is noteworthily close to the untuned TMVA benchmark with a difference of 0.01633 in private AMS, corresponding to five ranks on the private leaderboard. For the point of view that TMVA is a well-established tool in particle physics, the marginally worse performance of kNN is a small success. As Fig. 9 visualizes, kNN is very sensitive to the choice of feature sets. Careful use of artificial features could give improvement to this classification and enable the model to surpass at least one more benchmark. Over all, kNN was the only method we used that profited heavily from *excluding* features of the data. This offers another possibility for improving it. An algorithm to weight features for this model on the basis of clustering could give information valuable for manual or automatic feature selection.

4.1.2 XGBoost and Neural Networks

As mentioned, XGBoost quickly gained a fair amount of popularity after it has been presented in the Kaggle forums. We can understand this by comparing it to our approaches. Using low parameters, the package surpasses Logistic Regression with only 2 trees of depth 5 with *eta*=0.1 in under 2.5 seconds in total (training and prediction time). For beating kNN, our personal best submission, XGBoost uses 40 trees with the same remaining



Note the set sizes, listed in Tab. 3.

Figure 9: Performances of classifiers for specific feature sets

parameters, having a total runtime of under 12.03 seconds.

Many submissions in the private top 500 used the package for their final submission. In Fig. 10, which appears in the next section, we can see a flat line around public rank 500. This is a group of submissions that all used XGBoost with same parameters and therefore experienced the same AMS difference with the transition to the private leaderboard [Heb14]. Other participants were able to get a higher AMS with XGBoost. The differences in evaluation can be explained with different tuning efforts, we achieve a better score with heavy testing of different parameter settings. Our best XGBoost submission, measured by the public AMS, would be ranked on place 65.

The most common tuning for any classification method was proper feature engineering, though, as previously stated, this procedure generally resulted in small improvements of AMS. However, even little improvement of classifiers could have resulted in a respectable amount of ranks, as concluded from Fig. 8.

As we mentioned, an ensemble of neural networks delivered the winning submission, but it should be mentioned that the third place was also achieved by a method of this kind. Since we did not perform experiments with neural networks in this thesis, we can not provide much insight to possibilities of improving this method. The winner of the

challenge, Gábor Melis, reportedly used a number of artificial features [Mel15]. Courtiol Pierre (Rank 3) stated that increasing the size of his ensemble can improve the result slightly [Cou14].

4.2 Impact on Kaggle and CERN

While XGBoost did not win the Higgs Boson Machine Learning Challenge, it did have a strong impact on following Kaggle competitions. The most recently closed competition *Prudential Life Insurance Assessment*[6] features an ensemble of XGBoost classifiers as winning model. This and other methods involving XGBoost finished several competitions on Kaggle in top ranks [XGB16b].

4.2.1 Future Kaggle and CERN collaborations

After *The Higgs Boson Machine Learning Challenge* had ended, many competitors criticized AMS evaluation by calculating the so-called *Leaderboard Shake-up*. In the Kaggle forums, this formula is often stated as

```
shake-up = mean[abs(private_rank - public_rank) / number_of_teams].
```

We can express this mathematically as

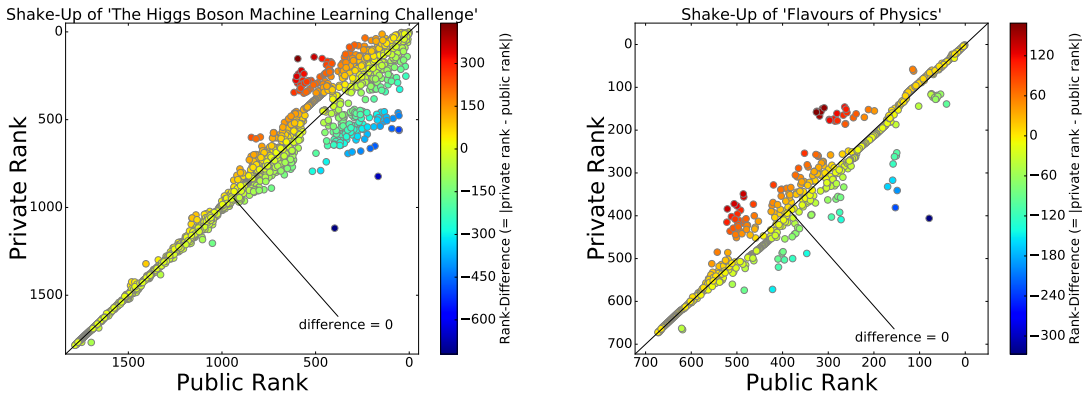
$$\text{Shake} - \text{up} := s(v, b) = \frac{1}{N * n} \sum_{i=1}^{n} |v_i - b_i| \quad , \tag{16}$$

where

- v and b are vectors of length n containing private and public ranks, sorted w.r.t. affiliation to another, so v_i and b_i are the ranks of a single participating team.
- and N is the total number of a challenges participating teams. N = 1785 in case of the *Higgs Boson Machine Learning Challenge*.
- n is the number of considered rank pairs for the calculation for a specific subset of leaderboard entries, like *Top 10% of private rank*. By definition n must be less than or equal to N.

This is a measurement for differences in ranking between public and private leaderboard. It is common for the Kaggle community to rate a competition by this criterion, a low score being good. Given the final rankings, Eq. (16) calculates a total Shake-up of 0.033 and 0.050 for the top 10% of participants. Summarized, competitors criticized the challenge mainly by two criteria. First, the amount of provided training data was too low. Community members argue that more test data used for the public AMS could prevent choosing overfitted submissions for the private leaderboard and subsequently improve the total

[6] The competition closed on 15 February 2016.
URL: https://www.kaggle.com/c/prudential-life-insurance-assessment



The leaderboard data was accessed from [kag16] with a simple Python script inspired by [Heb14].

Figure 10: Shake-up of *The Higgs Boson Machine Learning Challenge* and *Flavours of Physics*

Shake-up[kag14a, kag14b]. Second, while AMS evaluation was used due to its relation to actual particle physics, it was suboptimal for the challenge.

In CERN's following *Machine Learning Challenge* on Kaggle, *Flavours of Physics: Finding* $\tau \to \mu\mu\mu$, these two criteria were addressed. More test data was provided to the challenge and the evaluation was calculated by *weighted AUC*. Due to its favourable properties, AUC has been used by a number of top participants as objective function [DMnNV15]. Fig. 10 visualizes the Shake-up of this challenge and CERNs second Kaggle Challenge, latter having a total Shake-up of 0.038 and 0.013 for the top 10% of participants. The smaller Shake-up within the top ranks can be considered as improvement of the evaluation. However, it has not been stated that these differences resulted directly from insights gained in the first competition.

Though all the benchmarks provided by the challenge organizers have been surpassed by a considerable amount of submissions, the results ended up being controversial. An essential part of the challenge, optimizing the AMS, has been evaded by many competitors by using alternative objective functions. While this strategy opens up possibility for other research, many model designs that center AMS are probably dropped by their creators because they did not deliver significant improvements. The insight that CERN gained from the challenge taught more about winning on Kaggle than on designing new methods. All procedures that ended up in the top ranks are variations of established methods in data science, new, *game changing* strategies were not discovered in the challenge. Further, the winning submission *did not work at CERN*. The team of data scientists tasked with analysing the winners classification model, was unable to run its code due to compiling errors [Ké15].

A participating team was able to develop an original model based on theoretical results on AMS and its members were invited to *the Annual Conference on Neural Information Processing Systems 2014*(NIPS14) to present their model [MBM15]. Their resulting submission was only ranked 461th.

4.2.2 Data Analysis at CERN

Overall the challenge still seemed to be a success for CERN, as they started and closed up an additional Kaggle challenge[7] in 2015. The first and second best submissions of this competition were generated using XGBoost. Though it has not been reported that the package is currently used for classification at CERN, it is still possible that it will be implemented in future upgrades performed on the LHC and its experiments. Meanwhile, the package experiences further gain in popularity, for instance it is used for a cloud service by Alibaba.
The creators recently received the *John Chambers Award 2016* [XGB16a].

As the design of AMS was an effort to provide the competitors an objective function with physics background, the instability of AMS evaluation resulted in further research regarding usable objective functions for classification of LHC data [CGG+15]. On NIPS14, a *new* AMS was presented, having more stable properties by including the variance of *unknown background*. This could improve ATLAS' own evaluation metric [Ké14].

[7] The second challenge was called *Flavours of Physics: Finding* $\tau \to \mu\mu\mu$.

5 Conclusion

As this is the last chapter, I will briefly summarize the key statements of the whole work. After reviewing the main take-away for Kaggle and data science in physics, I close with my own, personal thoughts regarding this thesis.

5.1 Summary

After a bief introduction to the ATLAS experiment and the discovery of the Higgs Boson, we learnt about data analysis at CERN and the process of *event selection*. This topic was connected to Kaggle and *The Higgs Boson Machine Learning Challenge*, which is the works central topic. We formalized the challenges task: Label the data as signal or background events and rank them in order of signal probability, rank 550000 being the the event that most likely produced the Higgs Boson. A prediction for every single event contained in a test set shall be submitted to Kaggle and is evaluated via an estimation function, called the AMS. To succeed in the challenge, maximizing this AMS is the main goal. We prepared a first approach by presenting basic data science methods to optimize classification models in general. As we considered diverse requirements to the methods and limitations to our resources, we set up two approaches to the challenge based on the scikit learn package for Python. We started with Logistic Regression Classification, which performed poorly in the challenge. Our other approach was K Nearest Neigbhbors and was able to surpass three of five benchmarks provided by the challenges organizers. As only the model in this thesis, kNN profited significantly from careful feature selection. After our approaches we described Neural Networks, as the winning model used this type of learning method to generate the best submission, and ended Chap. 3 with XGBoost, which was acknowledged by CERN with a special award. This package was easy to use and achieved high AMS in short runtimes in comparison to other methods that ended up in top ranks on the leaderboards. It was concluded that the special *HEP meets ML Award* was justified. The fourth chapter of this thesis summarized the results of our approaches and compared these to the other methods presented before. It discussed further developments after the challenge regarding Kaggle and CERN. This included goals of CERN that the challenge fulfilled and expectations it did not satisfy.

5.2 Learned lessons regarding Kaggle

In general, ensemble methods seem to dominate most Kaggle challenges. This trend can be expected to continue as computational resources are constantly growing, allowing bigger ensembles to be run on home computers. Neural Networks will continue growing as important learning mechanism for data science and collaborating sciences. From a technical perspective, more experimentation with parallelizing known methods is reasonable. In several challenges, the evaluation metric fails to create the public leaderboard as good representation of the private leaderboard. The two-leaderboard system prevents models from succeeding, when they are actually overfitted, but also it does not give a reliable prediction of the private score. Good use of cross-validation(cv) counteracts this effect. If possible, the evaluation metric should be used for calculating the cv score to

achieve optimal model fitting. However, cv can not prevent overfitting if not used properly. Competitors stated overfitting as main threat in this challenge[Mel14, win14]. One key property of Kaggle is its active community. Though the participants of a challenge are usually competing for monetized prizes, it is still considered as scientific collaboration by many teams. However, the higher the prize money is, the less the Kaggle forum is used for discussing different approaches.

5.3 Regarding data science in physics

As future computer hardware will improve further, many methods notorious for their high demand in computational resources will become more viable for CERN research. Especially the *triggers*, which filter LHC data for the most interesting information, rely on their efficiency in data processing. If more complex models enable a more precise data selection in the same time, follow-up data analysis will benefit of higher quality data sets with less background events. In other experiments, like ALICE and LHCb, more efficient models are even more critical, as these experiments use the majority of data. They do not distinguish between signals and background, but classify most events for different purposes in later analysis[Gli15]. With the challenge, CERN was able to learn about design of these complex methods and most importantly promote the collaboration of data science and high energy physics.
To continue its current success and make important discoveries in physics CERN needs experts for both, physics *and* data science.

5.4 Own thoughts

I was surprised by the fact, that the winning model itself actually did not have any impact on data analysis at CERN, its code was not even able to be compiled. As I discovered Balázs Kégl's[8] blog near the end of the deadline for this work and read his perspective on this challenge, I found its outcome to be the opposite of my own expectations.

Regarding the results of my approaches: With 1785 submissions in total, among them many submitted by professional data scientist and particle physicists with years of experience, the competition in this challenge is big. Considering the small difference of the top submissions[9], the possibility of achieving a top 100 submission with an own approach is unrealistic for an undergraduate.
However, in my personal view, this thesis can be considered as a success for me. As main take-away from this project, I learned to investigate data science problems further before rushing to just try any classification models. Though a benefit from reading scientific papers is obvious, the importance of *comparing* papers was known to me, but never practised.

The plan for the future is to participate in actual, running Kaggle competitions and to obtain more experience, but this might be a topic for another day.

[8] Balázs Kégl is a Research Scientist at *The French National Centre for Scientific Research*(CNRS) and stated in the Kaggle forums to be one of the initiators of the Challenge.

[9] The AMS difference of the winning and rank 500 submission is 0.16933, which resembles less than 4.5% of the winning score.

References

[ABCG+15a] Claire Adam-Bourdarios, Glen Cowan, Cécile Germain, Isabelle Guyon, Balàzs Kégl, and David Rousseau. The higgs boson machine learning challenge. In *NIPS 2014 Workshop on High-energy Physics and Machine Learning*, volume 42 of *JMLR: Workshop and Conference Proceedings*, pages 19–55, 2015. URL http://jmlr.csail.mit.edu/proceedings/papers/v42/cowa14.pdf, Accessed: 2016-02-10.

[ABCG+15b] Claire Adam-Bourdarios, Glen Cowan, Cécile Germain, Isabelle Guyon, Balázs Kégl, and David Rousseau. Learning to discover: the higgs boson machine learning challenge. URL http://opendata.cern.ch/record/329, January 2015. Version 2.3.

[atl13] Evidence for Higgs Boson Decays to the $\tau^+\tau^-$ Final State with the ATLAS Detector. Technical Report ATLAS-CONF-2013-108, CERN, Geneva, Nov 2013. URL http://cds.cern.ch/record/1632191.

[ATL16] Official homepage of atlas. URL http://www.atlas.ch, 2016. Accessed: 2016-02-15.

[CER16] Official homepage of cern. URL http://www.cern.ch, 2016. Accessed: 2016-02-15.

[CGG+15] Glen Cowan, Cécile Germain, Isabelle Guyon, Balàzs Kégl, and David Rousseau, editors. *NIPS 2014 Workshop on High-energy Physics and Machine Learning*. CERN, CRNS, 2015. URL http://jmlr.csail.mit.edu/proceedings/papers/v42/edit14b.pdf, Accessed: 2016-02-09.

[CH15] Tianqi Chen and Tong He. Higgs boson discovery with boosted trees. In *NIPS 2014 Workshop on High-energy Physics and Machine Learning*, volume 42 of *JMLR: Workshop and Conference Proceedings*, pages 69–80, 2015. URL http://jmlr.csail.mit.edu/proceedings/papers/v42/chen14.pdf, Accessed: 2016-01-18.

[CHL08] Kai-Wei Chang, Cho-Jui Hsieh, and Chih-Jen Lin. Coordinate descent method for large-scale l2-loss linear support vector machines. *Journal of Machine Learning Research*, (9):1369–1398, july 2008. Accessed: 2016-02-11.

[Cou14] Pierre Courtiol. Third place model documentation. URL https://www.kaggle.com/c/higgs-boson/forums/t/10481/third-place-model-documentation/55390#post55390, 2014. Accessed: 2016-02-08.

[DMnNV15] Roberto Díaz-Morales and Ángel Navia-Vázquez. Optimization of ams using weighted auc optimized models. In *NIPS 2014 Workshop on High-energy Physics and Machine Learning*, volume 42 of *JMLR: Workshop and Conference Proceedings*, pages 109–127, 2015. URL http://jmlr.csail.mit.edu/proceedings/papers/v42/diaz14.pdf, Accessed: 2016-01-18.

[Gli15] Vladimir V. Gligorov. Real-time data analysis at the lhc: present and future. In *NIPS 2014 Workshop on High-energy Physics and Machine Learning*, volume 42 of *JMLR: Workshop and Conference Proceedings*, pages 1–18, 2015. URL http://jmlr.csail.mit.edu/proceedings/papers/v42/glig14.pdf, Accessed: 2016-02-15.

[Har14] Stefan Harmeling. Machine learning lecture 18: Neural networks continued. December 2014.

[Heb14] Jeff Hebert. Kaggle higgs classification challenge leaderboard analysis. URL http://rpubs.com/jeff-hebert/Higgs, 2014. Accessed: 2016-02-23.

[HH15] Jacob Howard and Christopher Hays. *Measurements of the Higgs Boson in the $H \to \tau\tau$ Decay Channel*. PhD thesis, Oxford U., Feb 2015. Presented 27 Mar 2015.

[Hig15] Dataset from the atlas higgs boson machine learning challenge 2014. URL http://opendata.cern.ch/record/328, 2015. Accessed: 2015-12-10.

[Ké14] Balázs Kégl. Introduction to the hepml workshop and the higgsml challenge. In *NIPS 2014 Workshop on High-energy Physics and Machine Learning*, JMLR: Workshop and Conference Proceedings, 2014. URL https://indico.lal.in2p3.fr/event/2632/session/0/contribution/1/material/slides/0.pdf, Accessed: 2016-01-18.

[Ké15] Balázs Kégl. What is wrong with data challenges. URL https://medium.com/@balazskegl/what-is-wrong-with-data-challenges-ed1f34246d44#.bg3zawx79, December 2015. Accessed: 2016-02-24.

[kag14a] Kaggle forum: Big differences between public and private leaderboard. URL https://www.kaggle.com/c/higgs-boson/forums/t/10346/big-differences-between-public-and-private-leaderboard, 2014. Accessed: 2016-02-20.

[kag14b] Kaggle forum: Quantifying leaderboard shake-up. URL https://www.kaggle.com/c/higgs-boson/forums/t/10320/quantifying-leaderboard-shake-up, 2014. Accessed: 2016-02-20.

[kag14c] Kaggle forum: Has anyone tried the support vector machine for this problem? URL https://www.kaggle.com/c/higgs-boson/forums/t/10165/has-anyone-tried-the-support-vector-machine-for-this-problem, 2014. Accessed: 2016-02-22.

[kag16] Kaggle homepage. URL https://www.kaggle.com, 2016. Accessed: 2016-02-16.

[Kot15] Wojciech Kotłowski. Consistent optimization of ams by logistic loss minimization. In *NIPS 2014 Workshop on High-energy Physics and Machine Learning*, volume 42 of *JMLR: Workshop and Conference Proceedings*, pages 57–67, 2015. URL http://jmlr.csail.mit.edu/proceedings/papers/v42/meli14.pdf, Accessed: 2016-02-09.

[Li] Cheng Li. A gentle introduction to gradient boosting. URL http://www.ccs.neu.edu/home/vip/teach/MLcourse/4_boosting/slides/gradient_boosting.pdf. Accessed: 2016-03-02.

[MBM15] Lester Mackey, Jordan Bryan, and Man Yue Mo. Weighted classification cascades for optimizing discovery significance in the higgsml challenge. In *NIPS 2014 Workshop on High-energy Physics and Machine Learning*, volume 42 of *JMLR: Workshop and Conference Proceedings*, pages 129–134, 2015. URL http://jmlr.csail.mit.edu/proceedings/papers/v42/mack14.pdf, Accessed: 2016-02-24.

[Mel14] Gábor Melis. Winning model documentation. URL https://github.com/melisgl/higgsml/blob/master/doc/model.md, 2014. Accessed: 2016-02-08.

[Mel15] Gábor Melis. Dissecting the winning solution of the higgsml challenge. In *NIPS 2014 Workshop on High-energy Physics and Machine Learning*, volume 42 of *JMLR: Workshop and Conference Proceedings*, pages 57–67, 2015. URL http://jmlr.csail.mit.edu/proceedings/papers/v42/meli14.pdf, Accessed: 2016-02-09.

[PPSS14] Jocelyn Perez, Ravi Ponmalai, Alex Silver, and Dacoda Strack. Ml2014: Higgs boson machine learning challenge. URL http://www.http://opendata.cern.ch/record/331, 2014. Accessed: 2015-12-10.

[Sal14] Tim Salimans. 2nd place documentation - higgsml. URL https://github.com/TimSalimans/HiggsML, 2014. Accessed: 2016-02-08.

[sci16a] scikit-learn homepage. URL http://scikit-learn.org/stable/index.html, 2016. Accessed: 2016-02-03.

[sci16b] scikit-learn on github. URL https://github.com/scikit-learn/scikit-learn, 2016. Accessed: 2016-02-03.

[sci16c] scikit-learn version-history. URL http://scikit-learn.org/stable/whats_new.html, 2016. Accessed: 2016-02-03.

[win14] Winning solution of kaggle higgs competition: what a single model can do? URL https://no2147483647.wordpress.com/2014/09/17/winning-solution-of-kaggle-higgs-competition-what-a-single-model-can-do/, 2014. Accessed: 2016-02-13.

[XGB16a] Xgboost demo readme. URL https://github.com/tqchen/xgboost/blob/master/demo/README.md, 2016. Accessed: 2016-02-21.

[XGB16b] Xgboost documentation. URL http://xgboost.readthedocs.org/en/latest/, 2016. Accessed: 2016-02-21.

List of Figures

List of Tables

A The Code used for this thesis

The code can be accessed on Github via the URL `https://github.com/gargi/BA_git/`. All original code is placed in this repository's directory `scripts/python/`.

To run any code it is required to unpack the data provided by opendata.cern. The file `atlas-higgs-challenge-2014-v2.csv.gz` is found in the directory `data`. Unpacking this file in place is sufficient.

Requirements

The functionality of the majority of the code has been successfully tested for Windows 8.1 64bit and Linux MINT 17.2 Rafaela. Code directly related to the xgboost package was only tested on Windows 8.1 64bit.

The Github release requires at least 500 mb free hard disk space.

Software dependencies

Any code for this thesis was created for following software dependencies, functionality for other versions has not been tested:

Software	Version used	Needed for
Python	3.4.4	everything
jupyter	4.0.6	working with IPython notebooks
Python packages		
numpy	1.10.1	everything
matplotlib	1.5.1	plotting
scikit-learn	0.17	all classification but xgboost
graphviz	2.38.0	plotting decision trees of xgboost
xgboost	0.47	classification with xgboost

An installation guide for xgboost is provided within the package's documentation [XGB16b].

Showcases

With the notebook format, IPython provides a great way for explaining code stepwise. Since Github is able to render this format online in repositories an increasing number of public repositories use this way to introduce new users to their code.

Several notebooks had been created as showcases for the code created and used in this thesis. They can be accessed and read directly on Github. They can be found in the directory `scripts/python/` and be run using jupyter and Python. PDF versions of each showcase are provided in the directory `docs/showcases/`, these files had been generated with jupyters `nbconvert` tool.

Following order is recommended:

1. `showcase_kaggleData.ipynb`
   As most functionality is used in some way on data, this showcase presents simple methods to access data relevant to Kaggle and *The Higgs Boson Machine Learning Challenge*.

2. `showcase_toolbox.ipynb`
   We introduce further tools that are used in the rest of showcases. This includes functionality important for Kaggle competitions, like building a submission file.

3. `showcase_figures.ipynb`
   This showcase combines the tools we introduced in 1. and 2. to reproduce figures used in this thesis. Figures 4a and 4b are excluded as they were created using the raster graphics editor Paint.NET.
   Fig. 5 is reproduced in `showcase_xgboost.ipynb` and Fig. 6 is cited from another work [CH15].

4. `showcase_sklearn.ipynb`
   We perform classification with scikit-learn and reproduce best submission files. This includes the creation of recording mechanics we used in testing.

5. `showcase_xgboost.ipynb`
   XGBoost is presented independent of the other classification methods.

opendata.cern starting kit

With the opendata release CERN provides software examples, which can be accessed under `http://opendata.cern.ch/record/331`. This includes a Python script to perform AMS scoring like performed by the challenge's leaderboards. To confirm evaluation scores we calculated for our submissions, we used a slightly edited version of `higgsml_opendata_kaggle.py`.

The original, unedited Python scripts provided by CERN can be found in the directory `scripts/python/HiggsML2014`.