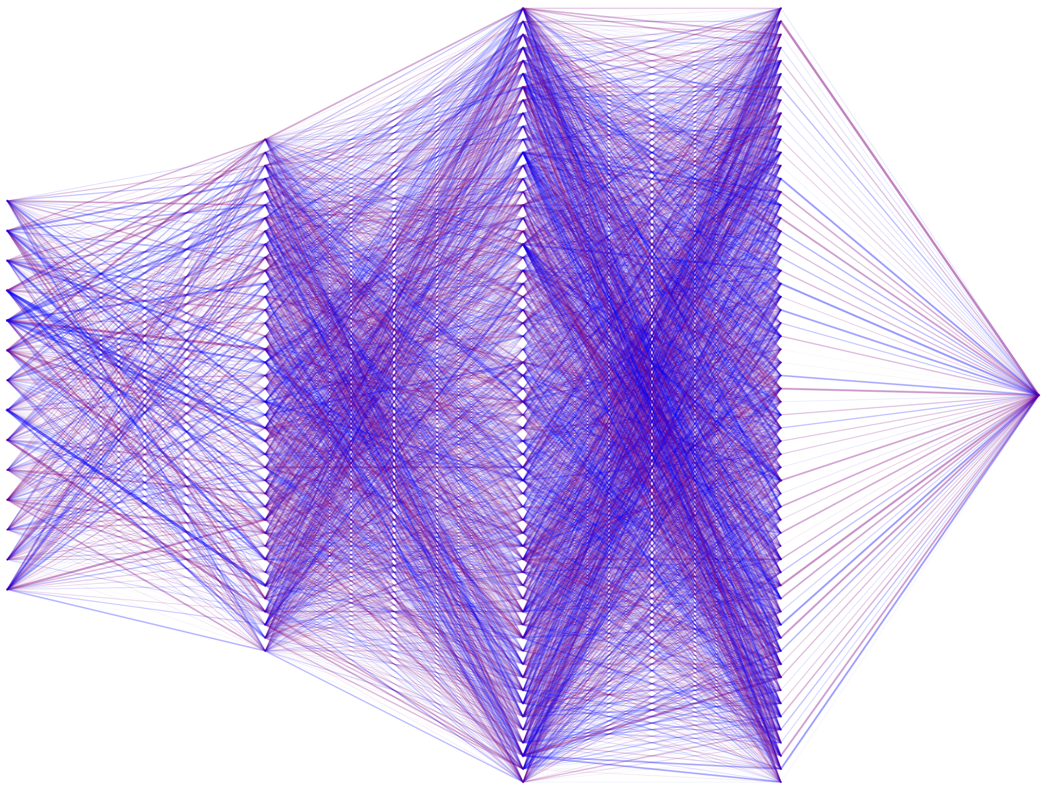
University of Zurich, Physik-Institut, autumn semester 2020

Bachelor Thesis

Supervision: Dr. Rafael Silva Coutinho, Jonas Eschle, Dr. Patrick Owen, Prof. Nicola Serra

Comparison of Prediction Uncertainties Using Bayesian Neural Networks and Boosted Decision Tree Ensembles with B Decays



Andreas Wiemeyer

April 5, 2023

Contents

Abstract

Bayesian Neural Networks and Boosted Decision Tree ensembles were trained for the signal-background separation for B-meson decays measured at the LHCb. The two types of classifiers were compared regarding the uncertainties on their predictions for signal, background and a third class of unknown events. For both types the behaviour was found to be similar, yielding a slight difference in the uncertainties for signal/background and unknown events. Furthermore it was tested whether one of the classifier types is better at generalising beyond the training data. No significant differences were found except for cases where the training data had different cuts applied. In these cases the Bayesian Neural Network was better than the Boosted Decision Tree ensembles at generalising to other files.

1 The Standard Model and New Physics

The Standard Model of particle Physics (SM) is a theory that states the elementary particles which all matter is made up of and describes how these particles interact with one another. The SM has been extremely successful in providing experimental predictions. Since the discovery of the Higgs boson in 2012, all elementary particles of the SM have been found. Nevertheless, there are reasons to believe that a theory beyond the SM is needed. The SM for instance does not explain gravity or other large scale phenomena. It does also not explain some experimental observations such as the neutrino oscillation. To test whether there are other experimental inaccuracies, the decay ratios of particles are studied using particle accelerators. This search for deviating observations is often called the search for new physics (NP). Research in these areas promises to give valuable insights for new theories beyond the SM. An important research facility is the Large Hadron Collider (LHC), the worlds largest and highest energy particle accelerator, built and operated by the European Organization for Nuclear Research (CERN). It is mainly used as a proton-proton synchrotron and reaches single-beam energies of up to 6.5 TeV. When these high energy beams collide, the constituents of the protons create heavier particles that decay into other particles along different decay channels. To investigate them, the LHC is equipped with multiple detectors, the largest of which are called Atlas, CMS, Alice and LHCb. Some detectors are designed to understand specific aspects of the decays, the behaviour of certain decay products for instance. The LHCb experiment, from which the data for the thesis project was taken, focuses on heavy flavour physics, such as decays of B-mesons. The specific decays that have been used in this thesis are the following:

$$B^0 \to K^* e^+ e^- \tag{1.1}$$

$$B^0 \to K^* J/\Psi(\to e^+ e^-) \tag{1.2}$$

$$B^0 \to K^* J/\Psi(\to \mu^+ \mu^-) \tag{1.3}$$

$$B^0 \to K^+ \pi^- \mu^+ \mu^- \tag{1.4}$$

$$B^0 \to K^+ \pi^- e^+ e^- \tag{1.5}$$

1.1 Detector setup

The LHCb detector, which was used to gather data for these decays, is shown in figure 1.1. It does not cover the full angle around the collision point because the relevant types of hadrons do not scatter far from the beam at high energies. An array of different components is used to gather information about the processes taking place in the detector. There are multiple layers (Vertex Locator, TT, T1-T3) that track the movement of charged particles. These are made of Kapton/Al straws or – where higher precision is needed – silicon-strips sensors. The tracking of the particles near the interaction point is done by a component called VELO, enabling a more precise location of the primary decay vertices. To retain information about charge and mass of a particle, a large magnet with an integrated field of 4 T m is set up to deflect the particles, such that charge and mass can be inferred from a particles path. Finally, there are multiple modules responsible for the identification of particle types. RICH1 and RICH2 are Ring Imaging Cherenkov counters, mainly responsible for π-K separation by measuring the velocity of the particle. The electromagnetic (ECAL) and hadronic (HCAL) calorimeters measure electrons, photons and hadrons position and energy. The muon stations (M1-M5), which are composed of wire chambers and gas electron multipliers interleaved with material to stop other particles, identify muons and determine their locations.

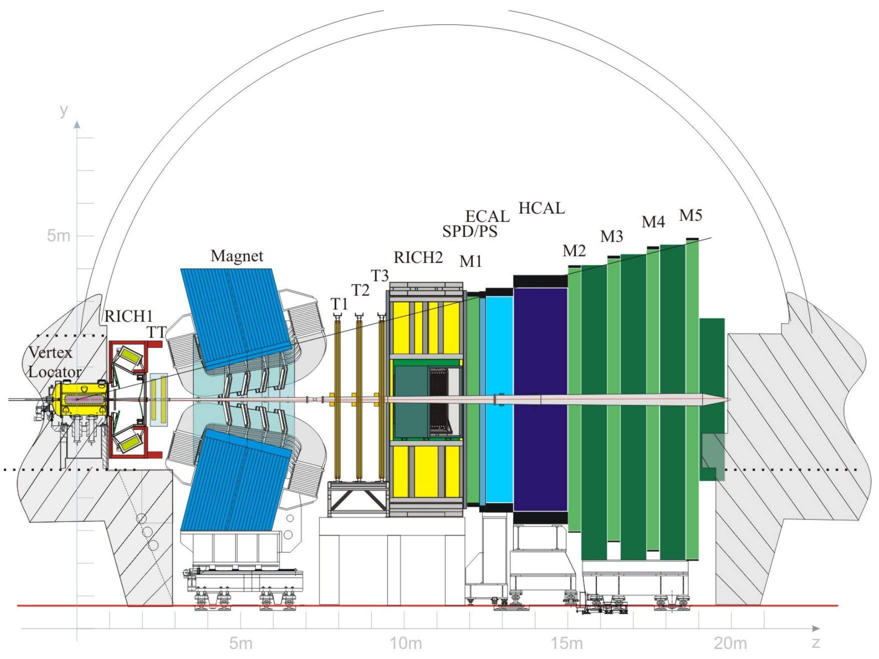


Fig. 1.1: The LHCb detector at CERN that determines the properties of the decay products [**2**, 2].

Without filtering, the detector would produce far too much data to store, since about 10 million proton collisions are registered every second. Because of this, there are multiple trigger stages, that decide which events to store and which to discard. In the first part of the selection the frequency of measured events is still too high to make elaborate computations. The transverse energy serves as a first selector (among other things), because particles with high transverse energy are easier to measure. This reduces the bandwidth enough to be able to partially recreate tracks, make more elaborate selections and save what is left to disk [2].

The offline analysis is done afterwords using the stored data. In short, the paths of particles are reconstructed more accurately than during the trigger stage to find the vertices at which decays happened. Using the information from the particle identification and the reconstructed paths one can figure out which particles decayed into which. In this data there is still a lot of background coming from noise in the electronics, random combination of particle tracks or other physics processes, which is in a first step reduced by only selecting events where certain measurements are within defined regions. Some kinds of background are more difficult to filter out because they require multiple variables to be considered for a clean separation. Therefore, a multivariate analysis (MVA) can be performed using machine learning algorithms. This part of the selection is what the thesis project aims to improve and will be returned to later. After the selection one can determine physical observables. One can for instance compute the efficiency of the selection and with it compute the branching ratios of different channels. These ratios can then in principle be used to test the SM and guide the way to NP. However, results are only statistically significant, if the selection is clean enough, which is why improvements is this area are very important.

The selection needs to deal with different kinds of background. One kind of background is due to falsely reconstructed decay vertices. It can happen, that decay products, which actually stem from different decays or other outside sources, have their paths reconstructed in such a way, that they are taken to stem from the same decay. This kind of background is called combinatorial background and it is the main target of the previously mentioned MVA.

To get cleaner results an algorithm is needed that can differentiate combinatorial background from signal. Given a data set with background and signal examples (see section 2.5), such an algorithm can be obtained with machine learning methods, of which several were used for the thesis project. Some algorithms that are commonly used in the field are Boosted Decision Trees (BDTs, see Section 3.1), which serve as a baseline comparison. They are compared to the more complex Neural Networks (NNs, see section 2.2) and finally to Bayesian Neural Networks (BNNs, see section 2.3). These are an interesting variant of NNs that can output a distribution of predictions from which a measure of certainty can be deduced. It was tested whether this makes them more robust to misclassifications of similar but different or even completely unknown events. One would expect that such events receive a prediction with a large uncertainty. This way one could know that one actually does not know well or even be able to tell that these are events of a different kind. In contrast, simpler methods, such as BDTs, are expected to sometimes mark completely unknown events as signal, with no associated uncertainty such that they cannot be distinguished from actual signal.

BNNs thus might enable researchers to identify odd events and find solutions to eliminate them. In the end this would allow for cleaner signal yields and more significant tests of the SM. The following chapter introduces the theory behind the above mentioned methods in more detail. The third chapter documents their practical application and the fourth chapter concerns the evaluation of the results.

2 Machine Learning Methodology

An algorithm can be said to learn, when it improves through experience. This is what differentiates machine learning algorithms from ordinary algorithms. Ordinarily, once an algorithm is written, it will keep performing its task in the same fashion resulting in equal output for equal input. Machine learning algorithms on the other hand are set up such that they can learn from the data they process and gradually become better at fulfilling their task until their peak performance is reached and then the algorithm is frozen. This definition encompasses a wide array of methods, not only the ones used for the thesis.

Since the right type of machine learning method depends largely on the type of task, the problem first needs to be characterized. Multiple data sets (for details see section 2.5) will be used, of which some contain only background, some only signal and some are mixed. The goal is to be able to separate the mixed data sets into background and signal: a binary classification problem. The sets that are already separated can be used to train the classifiers. The basic idea is to create a mixed data set from them, such that the correct label is known. The classifier is then set up to do a classification and since the correct solution is known, one can give feedback step by step to reduce the mistakes the algorithm makes. This process is called supervised learning. Once it is finished and a classification is learned, it can be applied to datasets where the true labels are not known (such as the mixed sets).

The following sections give a more concrete and mathematical idea of how the feedback loop works. Since this mechanism is different for all of the methods, the discussion is split into three sections.

2.1 Boosted Decision Trees

As the name suggests, boosted decision trees (BDTs) are an adaption of ordinary decision trees. A decision tree is a simple tool with which one can divide a set of items into subsets. Following the metaphor of a tree, the decision tree is made up of nodes and branches (see figure 2.1). At the start there is a single branch – the root or stem if you will – which divides into more and more branches, corresponding to subsets, at each node. Each node represents a decision with which the superset is subdivided. If items have a numerical value x for instance, a node might split into three subsets with $x > 4$, $x = 4$ and $x < 4$. With a clever choice of decisions at the nodes, a useful division into subsets can be obtained. One could for instance end up with m subsets that contain only (or mostly) signal events and n subsets that contain only (or mostly) background events.

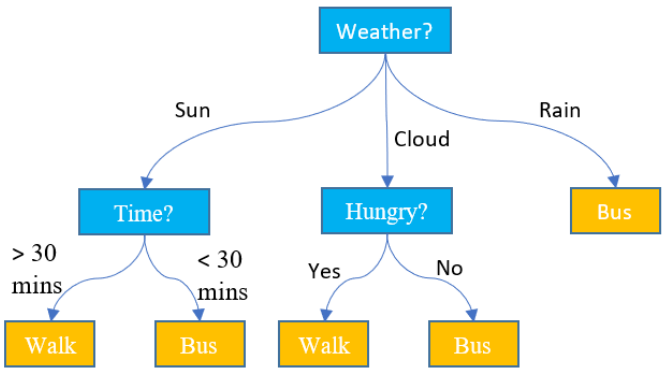


Fig. 2.1: The structure of a decision tree visualised[3].

A single tree typically is not a very performant classifier since it is prone to not generalise well to new data. There are several methods that are designed to overcome this problem by automatically generating many trees and joining them together. BDTs are one of these methods. They differ from other methods (like random forests), in that the tree generation works iteratively, meaning that the generation of each new tree is informed by the trees generated before it. The procedure works roughly as follows:

1. A first tree, a so called weak learner, is generated to obtain a baseline classification on which to improve.
2. The difference between the true class and the predicted class is used to compute the loss of each prediction.
3. A second tree is generated to predict the loss associated with the initial predictions.
4. By subtracting the predicted loss (usually multiplied with a learning rate $l < 1$) from the initial prediction a modified classification is obtained, that is more accurate than the initial one.
5. The modified prediction is used as new baseline and steps 2-5 are repeated n times.

There are many libraries which implement the BDT algorithm. The three libraries used for the thesis project are Scikit-learn, XGBoost and LightGBM. For a comparison of these libraries see section 3.1.

2.2 Neural Networks

Neural networks (NNs) are inspired by the structure of brains, where electrochemical pulses are passed from neuron to neuron. In the artificial version, neurons are modelled as entries of vectors and elecrochemical pulses are nonlinear functions, mapping from one set of vector entries to another. The initial vector serving as argument for the functions encodes the data one is working with, the so called input layer l_0. To obtain an algorithm that can learn, a mathematical operation with free parameters transforms it into a second vector (or layer) l_1. This in done in three steps with a weight matrix W, a bias vector b (these are the free parameters, both initially random) and an activation function f_a. First, l_0 is multiplied with the weight matrix, whose entries w_{ij}, are called weights and can be understood to determine the importance of the j-th entry in l_0 for the i-th entry in l_1. Afterwords the bias is added and the activation function is applied to get l_1. The activation function and the bias are used to introduce non-linearity, which is needed for the network to work. Usually the described operation is performed multiple times consecutively, such that there are n vectors (or layers) $l_1 \dots l_n$ in the end. One can thus generalise the operation mapping from one layer to the next as

$$l_{i+1} = f_{a_i}(W_i l_i + b_i). \tag{2.1}$$

The final layer l_n is called output layer. The goal of the consecutive operations is to obtain a useful output layer from the input layer. For the thesis project, the purpose of the output layer is to enable a classification into background and signal. To obtain this, the right weights W_i and biases b_i need to be found. To do so a measure called loss L is defined and then minimised. Depending on the problem, different loss functions work best, for the thesis project categorical cross-entropy was used. It can be defined as

$$L = -\frac{1}{N}\sum_{i=1}^{N} \vec{y}_{true,i} \cdot log(\vec{y}_{output,i}) \tag{2.2}$$

where the j-th entry of $\vec{y}_{true,i}$ and $\vec{y}_{output,i}$ determine whether the i-th event belongs to the j-th class. The entries of $\vec{y}_{true,i}$ are binary and are given by the known classification of the labelled training set. The output of the neural network $\vec{y}_{output,i}$ depends on the choice of the output layer, for the thesis project it has entries ranging from 0 to 1. To minimise the loss, one typically uses a modified or approximated version of gradient descent, like stochastic gradient descent (SGD). The idea behind SGD is to approximate the gradient of the loss with respect to W_i and b_i by computing it for a random subset – called batch – of the training data. One then subtracts the approximate gradient (scaled with a learning rate $0 < l < 1$) from W_i and b_i. If the learning rate is aptly chosen one will move towards a minimum of the loss. For each training epoch this is repeated until all of the training data was selected for the batch exactly once. This way the algorithm learns a lot faster and finds a more stable minimum than if one were to compute the gradient with the whole data set.

Because it is difficult to choose the right amount of free parameters, NNs often have too many degrees of freedom. This means that they can learn not just the true correlations within the data, but also remember random statistical noise within the training set. This effect is called overtraining or overfitting. When an algorithm is overfitting, its performance on the training set continues to increase, but the performance on other data (called out-of-sample data) decreases, because it does not share its random patterns. The more irrelevant patterns are learned, the more the network will be mislead on out-of-sample data, resulting in a worse performance. An illustrative analogy of this effect can be found in ordinary fits (see figure 2.2).

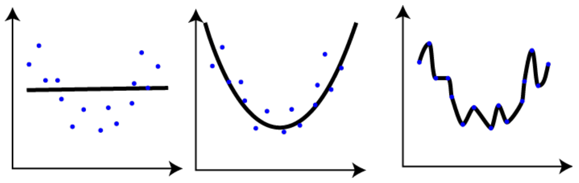


Fig. 2.2: A distribution that is underfitted (left), fitted just right (middle) and overfitted (right)[5].

One way to account for overfitting is to split the labelled data into a statistically independent training and test set. The former is used for training and the latter is used to get a measure of the out-of-sample performance. Because the test data is statistically independent, it will not contain the same random patterns as the training data and give an unbiased measure of the algorithms performance. By monitoring when the performance on the test data starts to decrease, one can find the ideal amount of training epochs.

Another issue arises when the training data is not exactly the same as the data one wants to classify. For the thesis project, it was only possible to obtain a labelled data set that approximates the real data (see section 2.5). This gives rise to the question of whether the classifier can generalise to the real out-of-sample data. There is no simple answer to this question and because of this, the evaluation of an algorithms performance on such data is difficult. A central motivation of the thesis project is to find ways in which an algorithm could be evaluated when going beyond the training data. This is discussed in more detail in section 2.4.

BDTs and NNs do not come with a good measure of how certain their classifications are. Such a measure would be of special interest for the generalisation to real data, which might contain previously unseen kinds of events. A possible solution is seen in BNNs, which are introduced in the next section.

2.3 Bayesian Neural Networks

As the name suggests, BNNs are an implementation of Bayesian statistics in the NN framework. The school owes its name to the famous statistician and philosopher Thomas Bayes. His approach to statistics is the most influential one next to the so-called frequentist approach. His ideas center around Bayesian inference, which relates different probability quantities to another.

Consider an hypothesis H explaining the observation of some data D. The data is a set of events, which have the combined probability $P(D)$ to be measured. The Hypothesis H can be given a probability $P(H)$, called prior, which says how likely H is to be the correct description of reality. Often the hypothesis is a function of a set of parameters θ_i, such that the prior $P(H(\theta_i))$ is a probability density function in the θ_i space. Next there is the likelihood function $P(D|H(\theta_i))$ which says how likely it is that the measurements D would be taken, if $H(\theta_i)$ were the true description of reality.

For the frequentist approach the hypothesis is looked for, which maximizes the likelihood function. The underlying assumption being that the θ_i must have a fixed value and the goal of the scientist is to find this correct value. On the Bayesian approach one does not settle on a fixed set of θ_i because one takes these values to be distributions by nature, which are not reducible to a single value. Because of this one is interested in how likely an hypothesis it to be true for the *whole range* of θ_i, given the data D. This is given by the posterior $P(H(\theta_i)|D)$, which is obtained with Bayesian inference as follows:

$$P(H(\theta_i)|D) = \frac{P(D|H(\theta_i))P(H(\theta_i))}{P(D)} \tag{2.3}$$

In a neural network the training parameters (W_i, b_i) can be understood to define an hypothesis $H(W_i, b_i)$ explaining the observed training data (X_i, y_i). Regular NNs are set up to find the fixed set of parameters for which the likelihood $P((X_i, y_i)|H(W_i, b_i))$ is maximized, they thus implement a frequentist approach. For the Bayesian approach, the parameters are understood to be distributions by nature. One thus wants to find the probability that – given the data – the network correctly codifies the classification for the whole parameter space. This is given by the posterior $P(H(W_i, b_i)|(X_i, y_i))$. With the posterior one could classify other X_j and obtain the right probability distribution y_j for their classifification. In practice this is not possible because one cannot precisely compute the posterior. However, one option is to use variational inference to approximate the posterior.

For variational inference a distribution $q_{\phi_i}(W_i, b_i)$ replaces the exact posterior. The distribution has its own parameters ϕ_i and is a function of the weights and biases of the network. During the training of the network, the parameters ϕ_i are fitted so that q_{ϕ_i} approximates the true posterior. To do so a different loss metric called *ELBO* is utilized. The weights and biases are no longer single values, since the posterior is to be computed for the whole parameter space. Thus, they are replaced by prior distributions with further parameters, from which they are sampled each time a prediction is made. For cases where it is hard to know the priors in advance, empirical Bayes can be and was used. With this variant the parameters of the priors are trained along with the ϕ_i, such that one only needs to choose the right parametrisation (for more information see [**22-23**, 1]). A normal distribution is a common default choice.

Because of the stochastic weights and biases and the new loss function, BNN layers converge less quickly and easily. To remedy this problem, one can rely mostly on normal layers and have only the last few layers implement empirical Bayes. This lets the network learn a lot faster and still gives decently satisfying results from a Bayesian point of view [**27**, 1].

A BNN differs from ordinary NNs and BDTs in that it is not deterministic. This means, that even after the training is completed, the algorithms output is not predetermined by the input. Since weights and biases are sampled for each prediction from their prior, the networks response function is different each time. Thus if a prediction is repeatedly made for the same event, a random distribution of outputs will be obtained. From the shape of this distribution one can read out how certain the network is about its prediction. The standard deviation of the distribution serves as a simple measure of certainty, with the prediction being given by the mean, the most likely value. The usefulness of this measure of certainty will be discussed in section 4.

2.4 Evaluating the Performance of an Algorithm

When it comes to the labelled test set, there are some easy to compute quantitative measures to tell whether the learned algorithm performs well. Firstly there is the accuracy, the percentage of events that are classified correctly. This measure can be unreliable though, since 1) it depends on the choice of the threshold for a prediction to count as signal and 2) imbalanced data sets can give misleading accuracies. In the labelled data used for the thesis project, over 93% of the events are signal, such that a trivial algorithm, that classifies everything as signal, would have an accuracy of over 93%. It is easy to see that further measures are needed to identify useful algorithms. One can split the predictive success of an algorithm into four categories:

1. True positive rate: $\text{TPR} = \frac{\text{amount of correct signal classifications}}{\text{amount of signal events}}$
2. False positive rate: $\text{FPR} = \frac{\text{amount of incorrect signal classifications}}{\text{amount of signal events}}$
3. True negative rate: $\text{TNR} = \frac{\text{amount of correct background classifications}}{\text{amount of background events}}$
4. False negative rate: $\text{FNR} = \frac{\text{amount of incorrect background classifications}}{\text{amount of background events}}$

An algorithm that classifies everything as signal only has a good TPR. By requiring also a good FPR, one can make sure that the signal is pure. One can make use of the TPR and the FPR to get a measure of the classifiers quality without having to choose a threshold for the classification. To do so the two ratios are plotted for different choices of the threshold. The resulting plot is called receiver operating characteristic curve or ROC-curve for short. Conventionally, the TPR is chosen as y-axis and the FPR as x-axis. For any threshold one wants the TPR to approach one and the FPR to approach zero. The curve for a perfect algorithm, which assigns all signal events a higher predictions than any background event, is shown in figure 2.3. For realistic data sets that are not neatly separable, the TPR increases with the FPR. This makes sense, because as the threshold for signal classifications is lowered, both more correct signal classifications and more false signal classifications are allowed. For a simple quantification of classifiers quality we can integrate the ROC-curve, obtaining the area under the ROC curve, called AUROC. This is the measure that was focused on for the thesis project, leaving the problem specific question of how to choose the classification threshold aside for later steps in the analysis. Sometimes the labelled data set can be so small, that there are large fluctuations in the AUROC score depending on which subset is used for testing. K-fold cross valdiation (CV) can be used solve this problem. For this method multiple classifiers are trained by dividing the labelled data into k subsets, training the classifiers on $k-1$ subsets and using them to predict on the left over subset. One then computes the AUROC score of the algorithms on the k different test sets and takes the average.

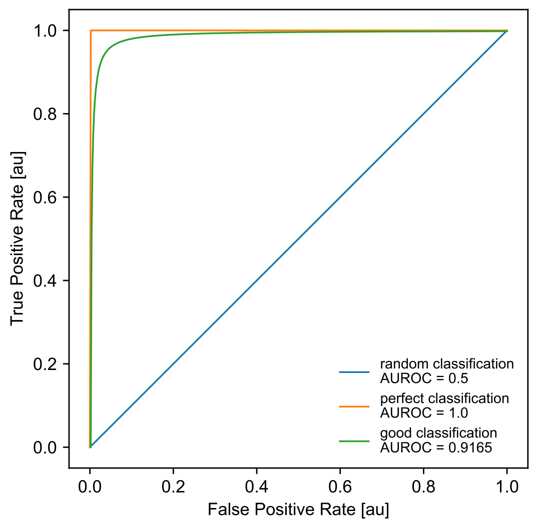


Fig. 2.3: The ROC-curves of different algorithms and the area under them (AUROC)

Since these measures cannot be computed for data sets other than the labelled data (*unlabelled* data from now on), other measures need to be found for these sets. Three criterions of quality were checked for. Firstly it was checked how well the algorithm handles unknown events by classifying different types of modified or generated data (see section 4.2).

The second criterion of quality is how nicely the classification works for the fits, which are made in later stages of the analysis. To evaluate this, the different classifiers are used to cut out the background before fitting the data. If a result looks nicer, it does not necessarily mean that it is more accurate. This will be discussed again when looking at the results in section 4.4.

The third criterion of quality is how well the Classifier generalises to other decay channels. It would be desirable to have a classifier that can be used on more decay channels than the one it was trained on. One way to test whether a classifier generalises well is to compare the predictions made for MC files of different decay paths. The difficulty in such a comparison is that some channels might in fact look more less like typical signal than others. If this is the case it would not be a weakness of the classifier to assign different predictions to such channels. A more elaborate discussion is thus needed and can be found in section 4.3.

2.5 Examining the Data

A machine learning classification can only work if good data is available. The data that is to be analysed and motivates the thesis was collected in 2016 with the LHCb experiment described in section 1. Along with the real data, files generated using Monte-Carlo simulation (MC) are used. These data sets are available for various decay channels listed below:

$B^0 \to K^* e^+ e^-$	MC and real data
$B^0 \to K^* J/\Psi(\to e^+ e^-)$	MC and real data
$B^0 \to K^* J/\Psi(\to \mu^+ \mu^-)$	MC and real data
$B^0 \to K^+ \pi^- \mu^+ \mu^-$	real data
$B^0 \to K^+ \pi^- e^+ e^-$	real data

The real data sets are a mix of background and signal, on which a classification needs to be performed. The MC files contain only simulated signal and can be used for training and for the performance measures mentioned in section 2.4. For the labelled training data a set of pure signal events and one of pure background events is needed. The signal is taken from the MC simulation for the $B^0 \to K^* e^+ e^-$ decay. The background is taken from a real data set by cutting off the region where the B_0 mass, the signal region, is above 5450 MeV/c^2. Above this threshold the signal contribution should be diminishingly small, so that the cut sample can be considered pure background. Granting this, there are still possible problems with the training data.

Firstly, the signal is simulated, meaning that there might be differences to signal events found in real data. Secondly, the background is cut out at a high energy region, but the real data also contains lower energy background. The algorithm thus needs to be able to generalise from higher to lower energy background. For future research it could be tried to use background from higher and lower energy regions to remedy this problem. Lastly, the signal set is much larger than the background set, containing 88'521 events, about 15 times more than the background set with 5'785 events. A data set with such a large ratio in class sizes is called an *imbalanced* data set and can be difficult to deal with. An algorithm might get stuck assigning every event a prediction of one, since this will already give a decent accuracy (over 93% would be classified correctly for the thesis project).

2.5.1 The feature space and its transformation

For the data used for thesis project, every event comes with thousands of features. Some of these are direct measurements and some of these are deduced quantities. Many of the features are redundant. Some have missing entries and some are not relevant for the classification, meaning that the amount of features can be greatly reduced. This reduction is important not only because it makes algorithms quicker to train but also because it reduces the risk of overfitting. Thus only 14 features corresponding to the quantities shown in table 2.1 were selected for the algorithm to work with.

The machine learning library XGBoost comes with a function to roughly estimate how important these features are for the classifier. The following features were found to be most

Feature name	Physical meaning
$B_{DTF\ PV}\ \chi^2$	χ^2 of the DecayTreeFitter fit of the primary vertex
$B_{FD}\ \chi^2$ PV	Flight distance significance of B-meson in units of χ^2 w.r.t. the related vertex
B SV χ^2	χ^2 of the B-meson decay vertex position
B_{DIRA} PV	Direction angle of B-meson w.r.t. the related Vertex
$H_{Min\ IP}\ \chi^2$ PV	χ^2 of the impact parameter of the hadron
B_{PT}	Transverse momentum of the B-meson
$L_{Min\ IP}\ \chi^2$ PV	Minimal impact parameter of the lepton χ^2 w.r.t. the related vertex
$H_{Min\ PT}$	Minimum transverse momentum of the hadron
$L_{Max\ PT}$	Maximal transverse momentum of the lepton
$B_{IP}\ \chi^2$ PV	Impact parameter of B-meson χ^2 w.r.t. the related vertex
$L_{Max\ IP}\ \chi^2$ PV	Maximal impact parameter of the lepton χ^2 w.r.t. the related vertex
$L_{Min\ PT}$	Minimal transverse momentum of the lepton
K^* SV χ^2	χ^2 of the K^*-meson decay vertex position
J/Ψ SV χ^2	χ^2 of the J/Ψ-meson decay vertex position

Table 2.1: The 14 features that were used for the classification and the corresponding physical quantities

important by XGBoosts *feature_importances_*:

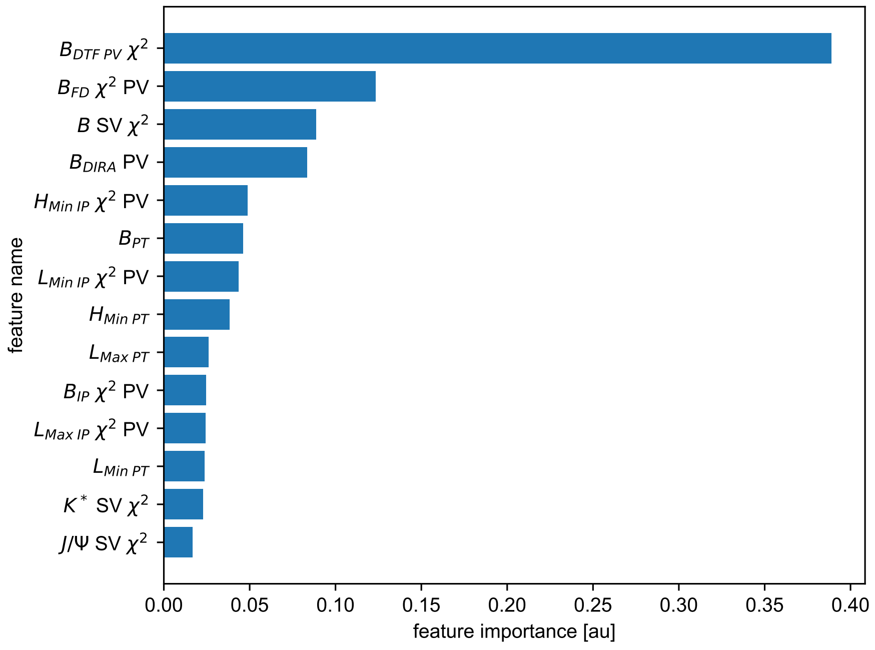


Fig. 2.4: The features found to be most relevant using XGBoosts *feature_importances_*

To get a better understanding of how a separation can be obtained and how clean this separation can be, the 4 most important features (according to XGBoosts *feature_importances_*) have been histogrammed for signal and background. The distributions for the different kinds of white noise from section 2.4 are also shown:

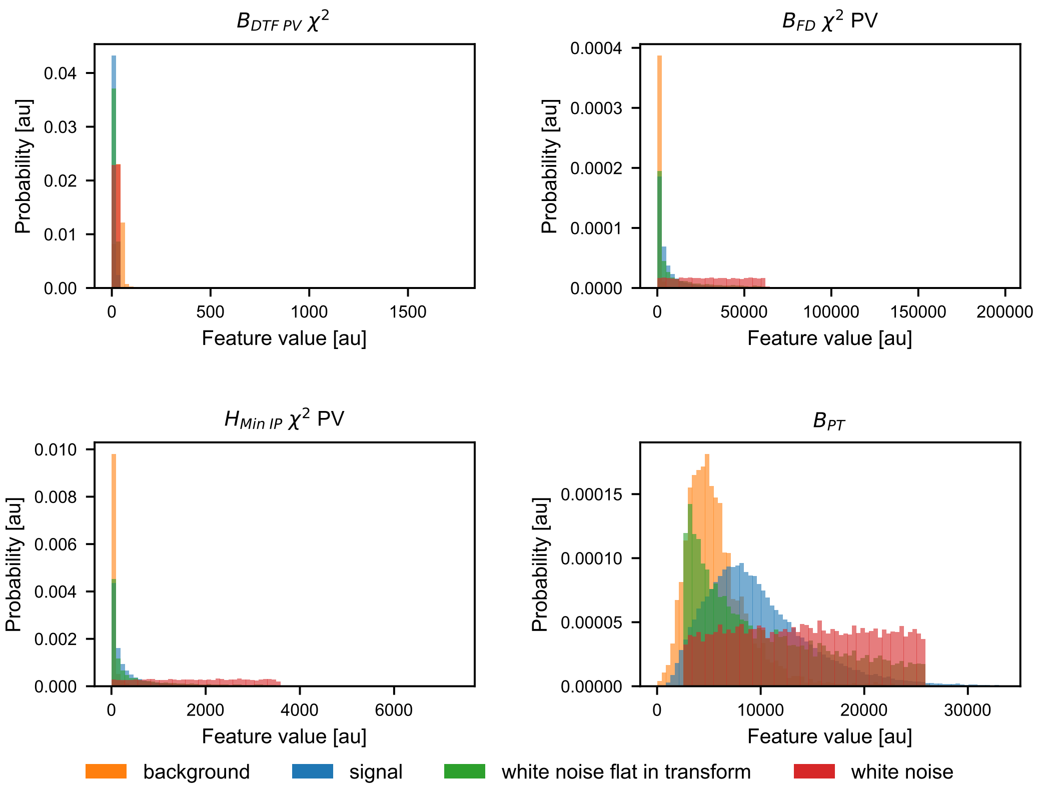


Fig. 2.5: The distribution of the 4 most important features (sorted from most to least important) for signal and background events compared to two kinds of white noise.

Many features follow an exponential-like distribution, such that a logarithmic transformation of the data can yield a better resolution as it makes it easier for neural networks to pick up on the difference between background and signal. Following a standard procedure in data preprocessing, the features were also shifted to have a mean of zero and scaled to have a standard deviation of one. This ensures that all of the features have an equally large effect on the network for the initial weights. After applying the transformations, the distribution of the features looks as follows:

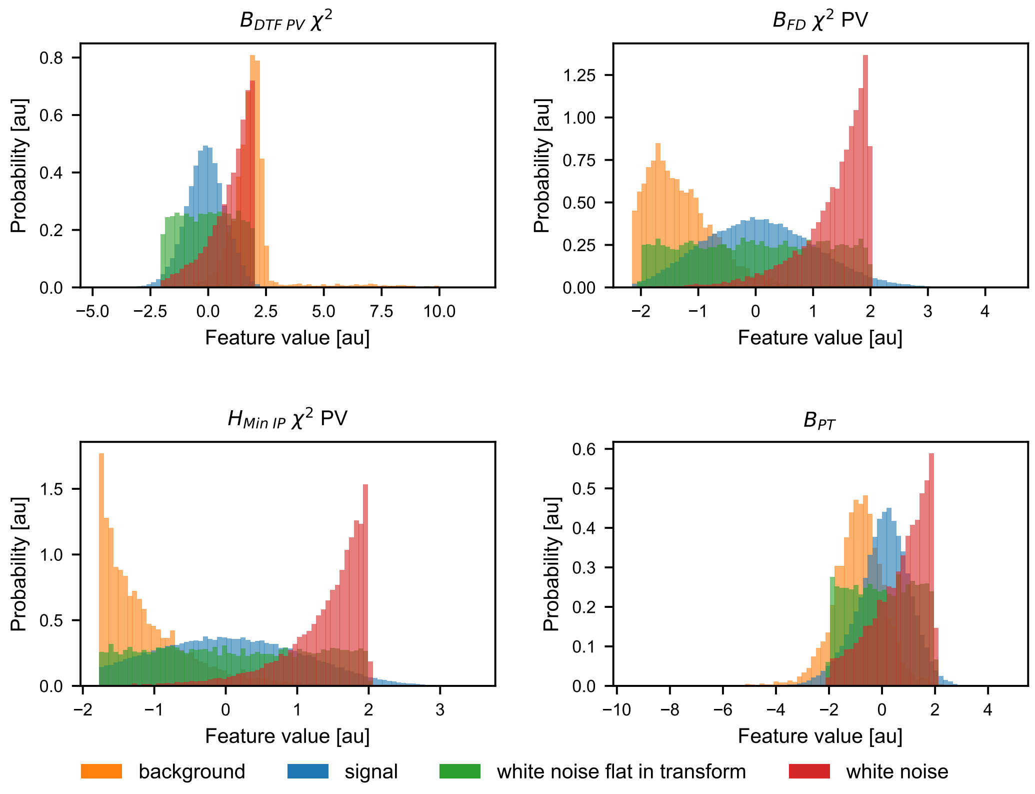


Fig. 2.6: The distribution of the 4 most important features (sorted from most to least important) for signal and background events after taking the logarithm and normalisation compared to two kinds of white noise.

Figure 2.6 illustrates nicely, why certain features were found to be most important by XGBoost (see figure 2.4). For the most important features the overlap of signal and background is small, meaning that the region where events are similarly likely to be background and signal is smaller.

3 Training the Classifiers

The methods described in sections 3.1, 2.2 and 2.3 were implemented using different machine learning libraries. First the BDTs were trained to quickly obtain a baseline, followed by ordinary NNs to have guidance for the geometry of the BNNs, which were trained last. The primary interest for the thesis is whether the measure of uncertainty from BNNs provides a significant advantage over BDTs and NNs. For a better comparison, BDT ensembles were trained from which an uncertainty can be obtained as well (see section 3.1.1).

3.1 Implementation of the Boosted Decision Trees

Three libraries were used to train BDTs: scikit-learn, XGBoost and LightGBM. Scikit-learn is the most accessible and comes with many useful functions and a lot of beginner friendly documentation. XGBoost has become somewhat of a standard in the field for its solid performance and relative ease of use. LightGBM is the newest of the three libraries and is becoming increasingly popular because it is by far the fastest to train and competes well with XGBoost performance wise. Because the implementation is similar for all of the libraries, the details will not be gone over for each of them. For all of the libraries it was quite easy to train an algorithm that performs decently well. The first step to improve the performance of the algorithms was tuning the hyperparameters. The resulting parameter values are shown in the appendix A.

When using a large proportion of the labelled data as training set, it was found that the test set was too small to give a stable measure of the performance. For a more reliable measure k-fold cross-validation (CV, see section 2.4), was used. Figure **??** shows how much the tuning effected the performance as a function of k. The errors bars indicate the standard deviation of the AUROC score across 20 different k-fold splits.

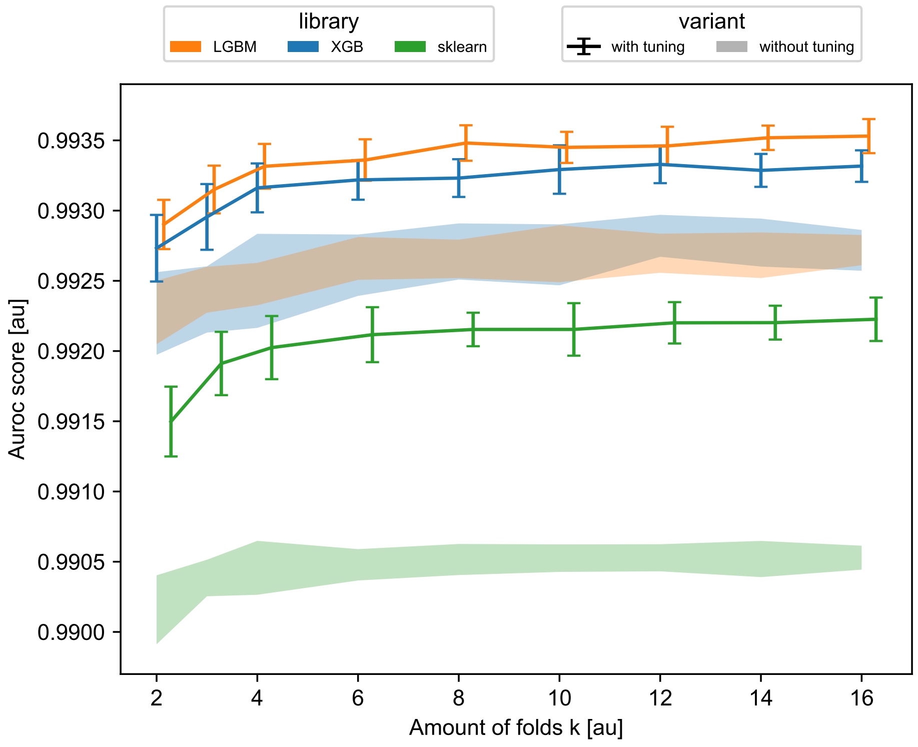


Fig. 3.1: The AUROC score of the different BDTs with and without tuning of the hyperparameters computed with k-fold CV and averaged across 20 different fold seeds. The shaded areas fill the area between the first standard deviation of the AUROC score for BDTs without hyperparameter tuning

More effort could have been invested into the tuning, but these results are good enough for having a benchmark. The training and prediction times for the different classifiers can be found in appendix D.

3.1.1 Boosted Decision Tree Ensembles

As a final improvement of the BDTs and to obtain a comparison for the BNNs measure of uncertainty, an ensemble of BDTs was trained. What this means is that multiple BDTs are trained to each make a prediction which are then joined into a single prediction. For the thesis 20 different BDTs were taken from the k fold CV evaluation and their predictions were joined by taking the average, with the standard deviation giving a predictions uncertainty. The motivation behind this method is to avoid overfitting on patterns that only show up in some training sets. If one takes the average, the random fluctuations caused by these patterns should be diminished.

The method was applied for a range of values of k, giving the performance increases shown in figure 3.2.

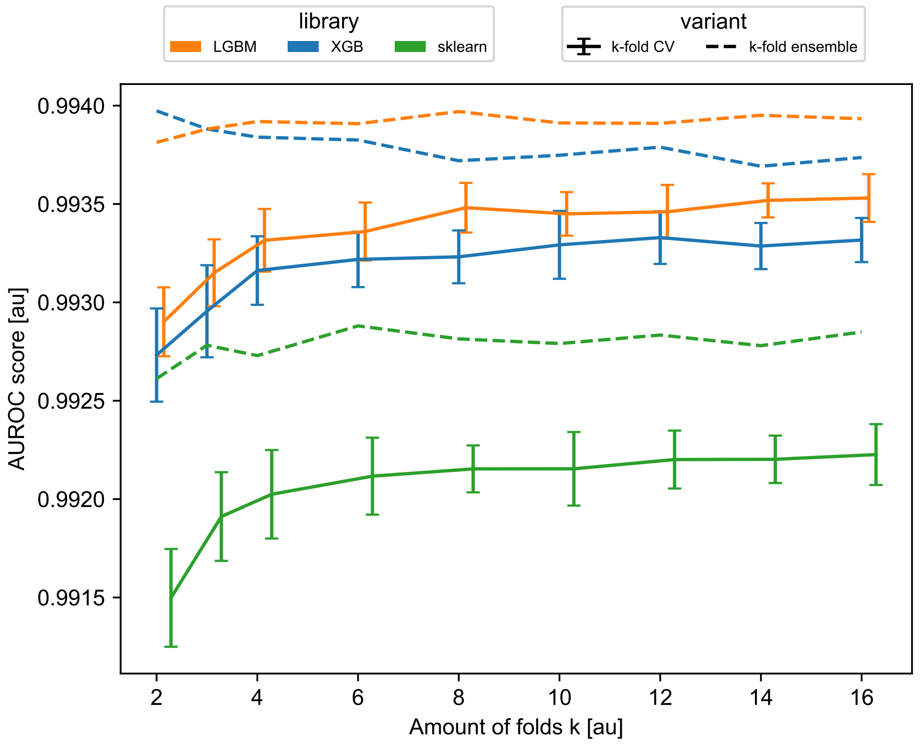


Fig. 3.2: The AUROC score of the different BDT ensembles taken from k-fold CV compared to simple k fold CV BDTs (all with tuned hyperparameters)

Across all values of k averaging yields a stable performance increase on the labelled data. Because of this, the further analysis will focus on BDT ensembles as a benchmark for NNs and BNNs. The previous performance measures seem to indicate that scikit-learn is not a viable alternative to XGBoost and LightGBM. We will nevertheless continue to evaluate the BDT ensembles for all Libraries because there might be significant differences when it comes to the generalisation to other data types. The measure of uncertainty of the BDT ensembles will be examined in more detail in section 4.1.

3.2 Implementation of a Neural Network

As expected, it was much harder to set up a working NN than a BDT. The first difficulty is that there is far less guidance for choosing the initial hyperparameters, like the networks geometry. Since this is crucial for the performance (a wrong choice can make the network completely useless) a lot more trial an error was needed. The network was set up as feedforward NN, using the Sequential model of *tensorflow.keras*. Four layers with 40 to 60 nodes were eventually settled on, details can be found in appendix B.

The tuning of the network was made a lot easier by transforming the data as shown in figure 2.6. After the transformation it was possible to drastically reduce the amount of layers and nodes per layer, making the network faster to train and making it easier to tune other parameters or test new optimizations. The final hyperparameters are found in appendix B.

In order to not overfit, a feature called early stopping was implemented. The early stopping callback was set up to stop training when the AUROC score computed for an independent test set has not increased over the last 5 epochs. In this case the network has most likely started to overfit and the training can be stopped.

Since the training can be quite computationally expensive, something like a k-fold CV averaged over 20 seeds to evaluate the performance was not tried. Instead, the labelled data was split into a training set and a test set to compute the AUROC score. This was done for 20 random train/test splits, to obtain a measure of how reliable the score is. Furthermore different split ratios were tried to check how much the network depends on the amount of available data, the results are shown in table 3.1.

Train/Test split	AUROC score
90%/%10	0.9942 ± 0.0007
80%/%20	0.9942 ± 0.0007
70%/%30	0.9942 ± 0.0008
60%/%40	0.9941 ± 0.0007
50%/%50	0.9941 ± 0.0007

Table 3.1: The AUROC score of the NN averaged over 20 random data splits for different ratios of training and testing data and the time it took to train and predict

For the further evaluation, a classifier with a 70%/30% training/test split ratio was used. For the BNN this split ratio will be motivated in section 3.3. For the main research interest the NN only serves as a guidance to constructing the BNN, so these results are satisfactory.

3.3 Implementation of a Bayesian Neural Network

The Bayesian neural network was even more difficult to set up. It has the same geometry with some of the layers being *DenseVariational* layers from the TensorFlow Probability library. Since these layers implement Empirical Bayes, they need to have more free parameters to fit the distribution of the weights and biases and the posterior shape, three times as many parameters were used for the thesis (see section 2.3). The fitting of these parameters is much more error prone, which is why a wrong amount of variational layers or a wrong ordering can prevent the network from learning. For the thesis project a single variational layer was placed at the end of the network. A further difficulty is, that one needs to define the shape from which weights and biases are sampled as well as the initial parametrisation of the distribution. A wrong choice can again prevent the network from learning.

The hyperparameters that were eventually settled on can be found in appendix C. Again early stopping and the same evaluation method was used to find the following AUROC scores for different train/test splits:

Train/Test split	AUROC
90/10	0.9925 ± 0.0006
80/20	0.9926 ± 0.0006
70/30	0.9927 ± 0.0006
60/40	0.9929 ± 0.0009
50/50	0.9926 ± 0.0006

Table 3.2: The AUROC score of the BNN averaged over 20 random data splits for different ratios of training and testing data

The classifier that used for the evaluation was trained with a 70%/30% train/test split, because the test set needs to be somewhat large for the training of a meta classifier introduced in section 4.2.

4 Evaluation of the Classifiers and Discussion

For the performance on the labelled data set, the AUROC scores indicate, which classifiers are best. A summary of the results is shown in table 4.1. The BDT ensembles lack a confidence interval because they are only trained once due to computational intensity. Figure 3.2 suggests that the scores are somewhat stable though.

Classifier	AUROC
Neural Network	0.9948 ± 0.0008
XGBoost ensemble	0.9940
LightGBM ensemble	0.9939
given algorithm	0.9933
scikit-learn ensemble	0.9928
Bayesian Neural Network	0.9927 ± 0.0006

Table 4.1: The AUROC scores of the different classifiers sorted in descending order

The classifier called *given algorithm* is a benchmark MVA score that already came with the data. It was obtained with an XGBoost classifier, that was optimized with a Grid Search and randomly selected from one of 10 classifiers obtained with 10-fold CV. The training data is almost the same as the one used for this theses, expect for the background cut being at 5600 MeV/c^2 rather than 5450 MeV/c^2. The given algorithm has a significantly worse AUROC score on the labelled data than some of the newly trained classifiers. These results are not the primary interest of the thesis though. The evaluation will focus on the performance on unlabelled data, since it is most important how the algorithms generalise beyond the labelled data. As mentioned in section 2.4, evaluating the performance on such data is a difficult task. The following sections show how well the different classifiers fulfill the criteria of quality from section 2.4. Three research questions are central for this evaluation.

1. Is the uncertainty supplied by the BNN different from the one of the BDT ensemble?
2. Are the uncertainties useful for the detection of unknown events?
3. How well do the classifiers perform applied to different channels?
4. How useful is the classifier for the further analysis?

4.1 Frequentist Versus Bayesian Uncertainty

As mentioned in section 2.4 several types of unknown events were generated to be predicted on. These are events for which:

1. Every feature is sampled from a flat distribution in the feature space (white noise flat)
2. Every feature is sampled from a distribution that is flat in the transformed feature space (white noise flat in transform)
3. Gaussian noise proportional to the standard the deviation of the feature and scaled by different factors is added to the features (noisy 10%, 30%, ...)

Since BDTs are easier to train, there would be no reason to use a BNN if it did not provide another benefit. It was thus checked whether there is a significant difference between the BDT ensembles and the BNN when it comes to predicting on these events. To do so, the uncertainties assigned to different types of unknown events were visualised and compared.

Figures 4.1, 4.2, 4.3 and 4.4 show the predictions assigned to events within a given percentile of their respective data sets. The width of the bars represent the uncertainties of the predictions.

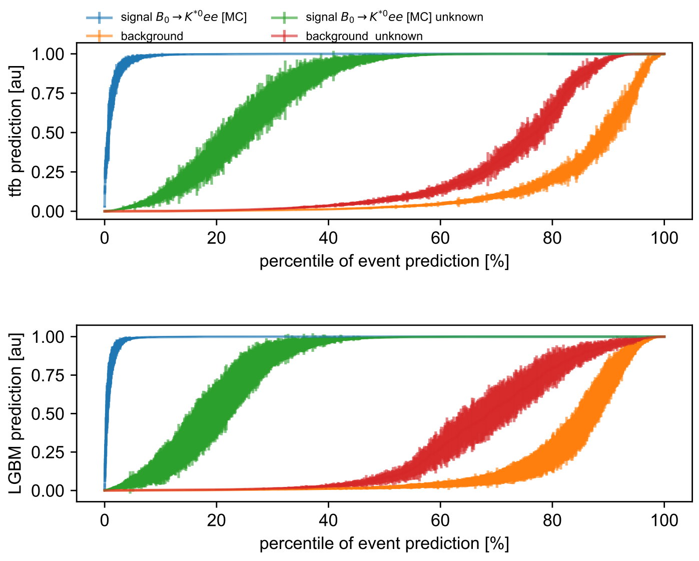


Fig. 4.1: The prediction value and its uncertainty for events where gaussian noise proportional to 10% of a features standard deviation is added compared to the original labelled data

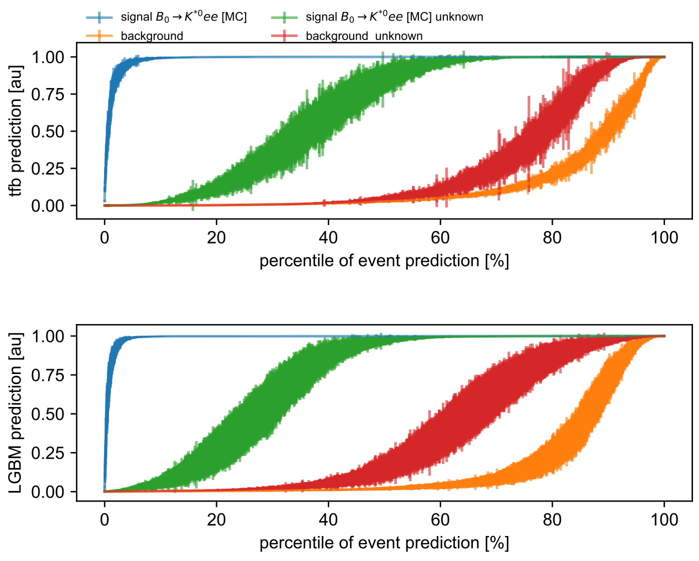


Fig. 4.2: The prediction value and its uncertainty for events where gaussian noise proportional to 30% of a features standard deviation is added compared to the original labelled data

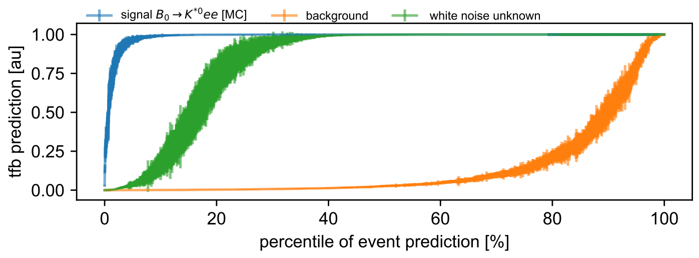


Fig. 4.3: The prediction value and its uncertainty for white noise that is uniformly distributed in the transformed feature space compared to the original labelled data

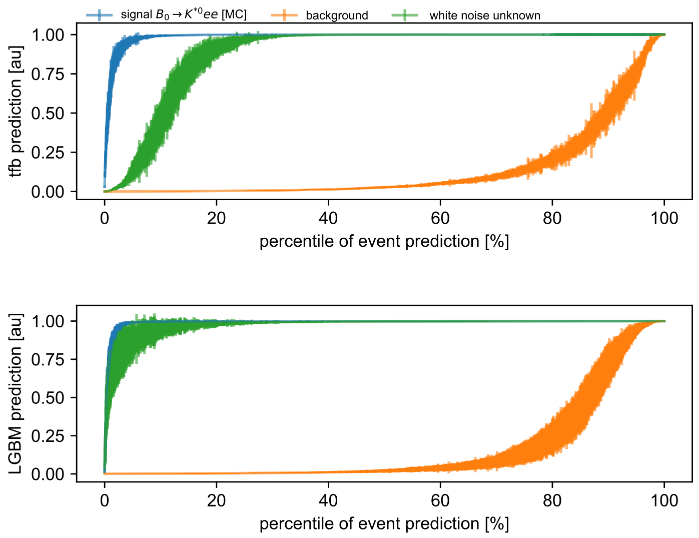


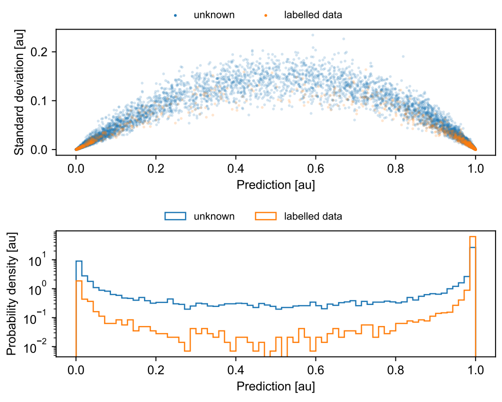
Fig. 4.4: The prediction value and its uncertainty for white noise that is uniformly distributed in the feature space compared to the original labelled data

The figures above are for the LightGBM ensemble, but similar results are obtained with scikit-learn and XGBoost. While there are differences for white noise that is uniformly distributed in the feature space (figure 4.4), the behavior is strikingly similar for all other types of unknown events. The fact that this holds for all of the BDT ensembles suggests that the similarity is not random. Rather it suggests that some types of unknown events are actually more or less like signal and that all of the classifiers can pick up on this. White noise that is uniformly distributed in the feature space might be an exception because it is drastically different from real events, which are distributed exponentially (this can be seen in figure 2.5).

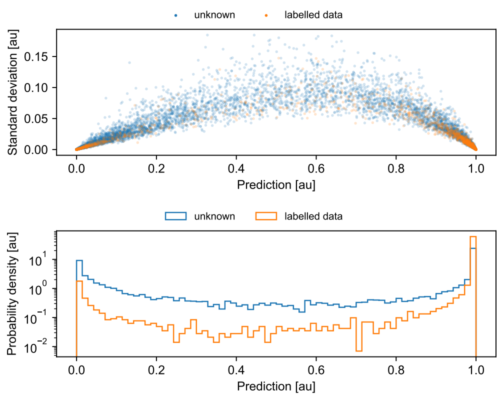
There thus is no evidence so far that the BNNs uncertainty differs substantially from the BDT ensembles. The fact that differences only become significant for completely different types of data (white noise in figures 4.3, 4.4) indicates, that issues only come up when one moves too far from what the networks were trained on. Within the realm of what events realistically look like, all classifiers give similar predictions and uncertainties. The differences seem to be only up to tuning. It is still possible that BNNs might be helpful when completely different events are predicted on, but it is unclear what these use cases would look like. As a consequence it is unclear how to evaluate whether BNNs would actually perform better in these cases.

4.2 Using Uncertainties for Classification

A central motivation behind making predictions with uncertainties was to be able to identify unknown events. For this to be the case, a classifiers output for unknown events would need to be differently distributed in the prediction-uncertainty space. Figures 4.5, 4.6, 4.7 and 4.8 show how events of different types are scattered in this space.

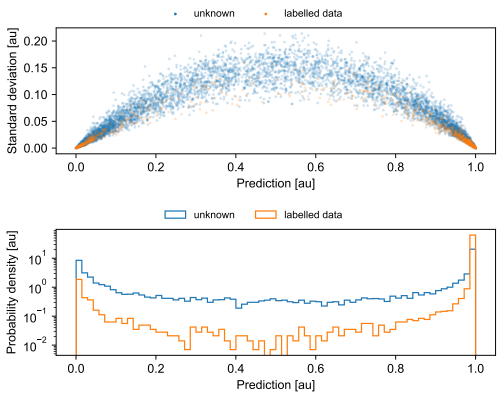


(a) for the LightGBM ensemble

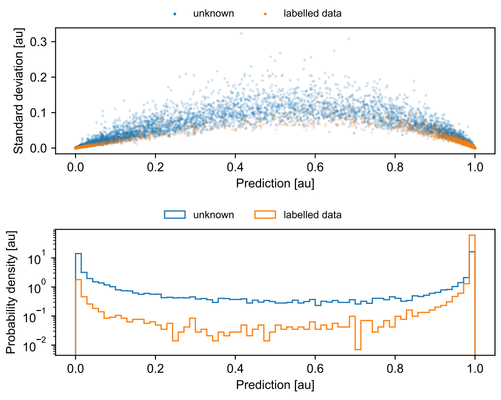


(b) for the BNN

Fig. 4.5: The predictions and uncertainties assigned to 10000 events from the labelled data set and the data set where gaussian noise proportional to 10% of a features standard deviation is added

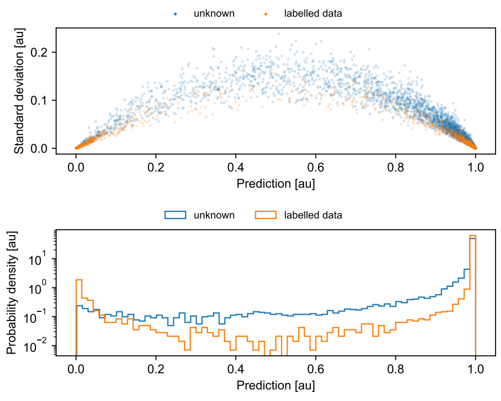


(a) for the LightGBM ensemble

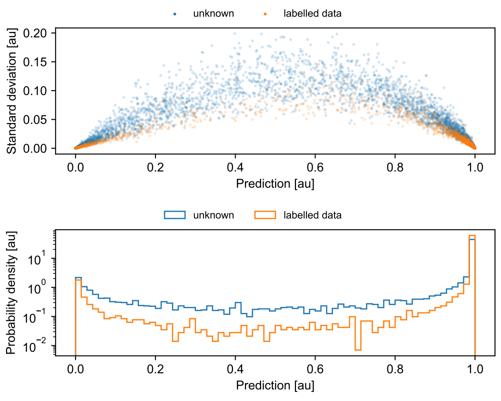


(b) for the BNN

Fig. 4.6: The predictions and uncertainties assigned to 10000 events from the labelled data set and data set where gaussian noise proportional to 30% of a features standard deviation is added

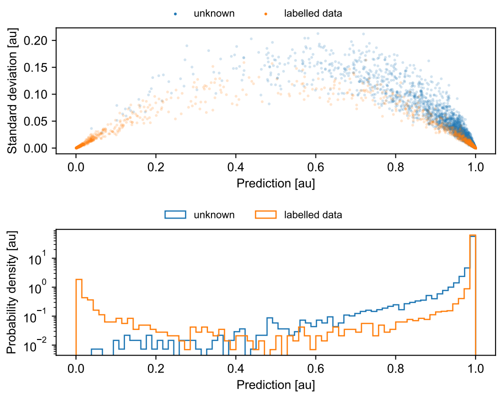


(a) for the LightGBM ensemble

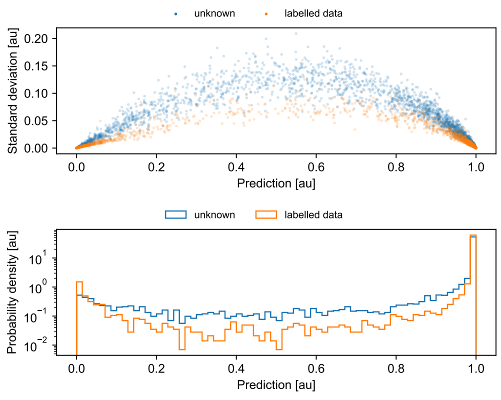


(b) for the BNN

Fig. 4.7: The predictions and uncertainties assigned to 10000 events from the labelled data set and the white noise that is uniformly distributed in the transformed feature space



(a) for the LightGBM ensemble



(b) for the BNN

Fig. 4.8: The predictions and uncertainties assigned to 10000 events from the labelled data set and the white noise that is uniformly distributed in the feature space

The figures again show that there are no obvious differences between the BDT ensembles and the BNN. For none of the classifiers, the unknown events would be recognizable as a different species in the scatter plots.

From these plots it is hard to get a more quantitative understanding of how neatly unknown events can be separated. One could imagine a use case, where it is known that events of some strange kind will show up in the data, i.e. a known unknown is present. For this use case one could train a second order BDT, that classifies events into unknown data and labelled data given their assigned predictions and uncertainties. This was done for the events discussed above. The resulting AUROC scores in figure 4.9 give a quantitative measure of how neatly the separation could be performed for the different classifiers outputs. Figure 4.10 shows how much the AUROC score improves over the case where only a simple cut in the prediction is made and the uncertainty is not considered.

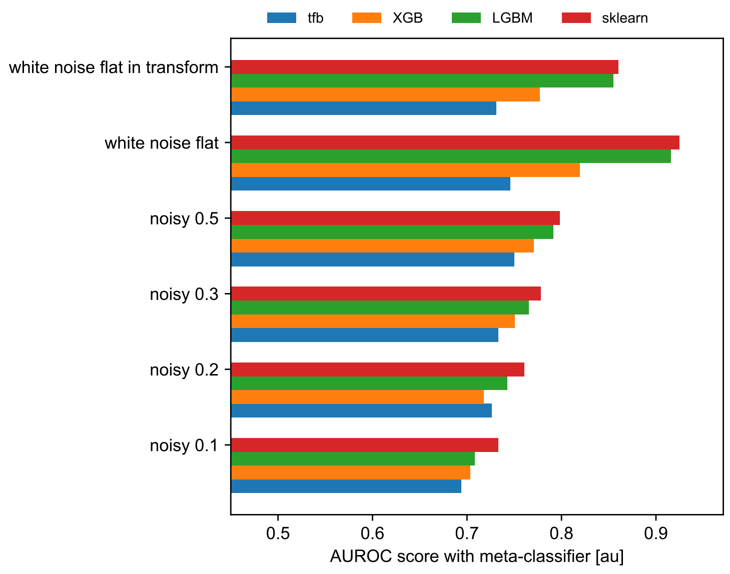


Fig. 4.9: The AUROC score of the meta classifier separating unknown events from the labelled data

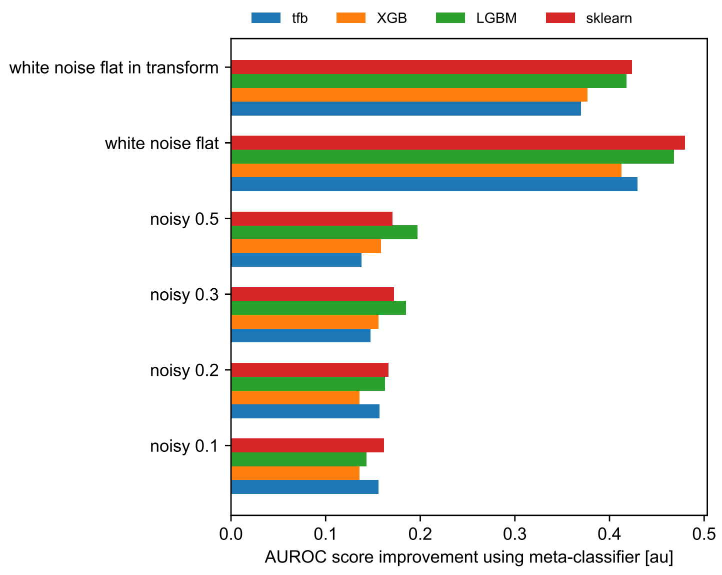


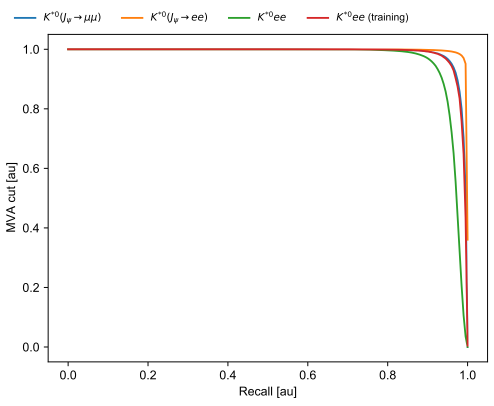
Fig. 4.10: The AUROC score improvement gained by using the meta classifier to separate unknown events from the labelled data (instead of just using the predictions)

Again the only significant differences are in the white noise. The BNN is clearly worse at separating white noise from the labelled data. This is somewhat surprising because in figures 4.3 and 4.4 it looks like there is a larger difference between signal and white noise for the BNN. A possible explanation is that there is a slight difference between the prediction plateau for white noise and for signal that can not be seen in the plots. For noisy data the performance differences across classifiers are most likely dependent on the optimisation.

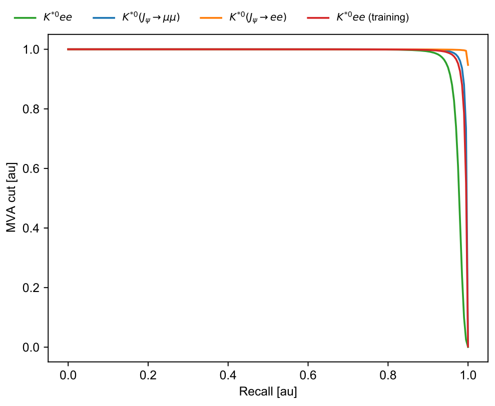
There are no apparent advantages to using a BNN.

4.3 Generalisation to other decay channels

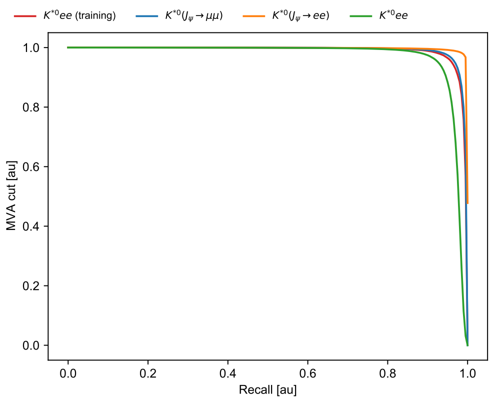
It would be desirable to have a classifier, that works equally well for all decay channels. Assuming that the background is the same for all data files, this is fulfilled if the recall of the classifier behaves in the same way for all channels. This can be examined by studying the predictions for MC files for the different paths. For a comparison that does not depend on the cut, one can visualise the recall as a function of the cut. This comparison is shown in figures 4.11d to **??**.



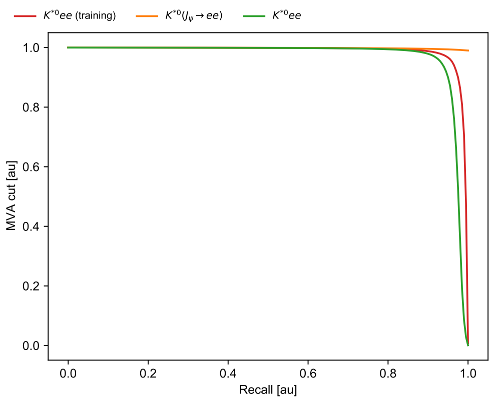
(a) for the BNN



(b) for the LightGBM ensemble



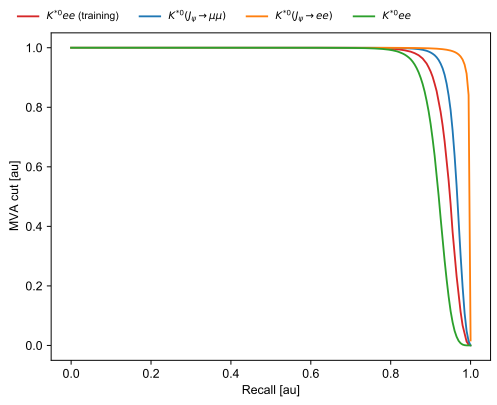
(c) for the NN



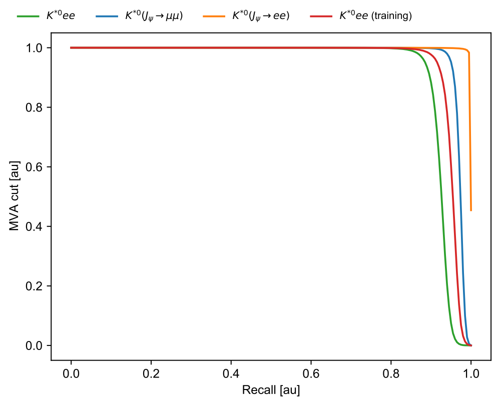
(d) for the given algorithm

Fig. 4.11: The MVA cut required to obtain a given recall for different MC files with classifiers trained on $K^*e^+e^-$ MC

What these figures show is that some channels receive higher predictions than others for all classifiers. Most clearly $B^0 \to K^*J/\Psi(\to e^+e^-)$ receives higher predictions on average than the other channels. This behaviour could be due to events from this MC file actually looking more like typical signal or due to the classifiers generalising poorly. To determine what is causing the disparity, a cross check was performed by training the classifiers on J/Ψ signal and redoing the plots.



(a) for the BNN

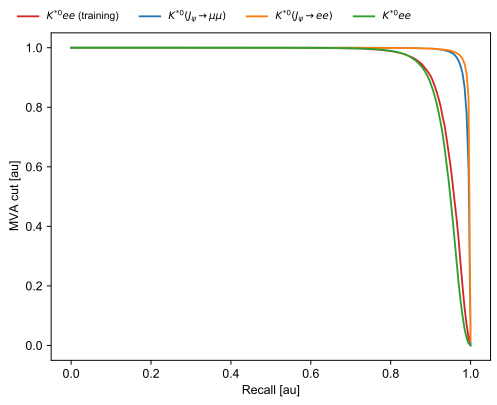


(b) for the LightGBM ensemble

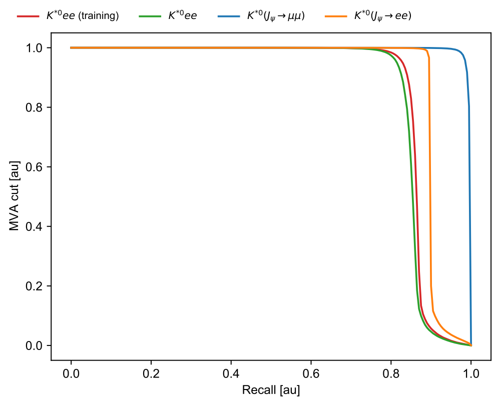
Fig. 4.12: The MVA cut required to obtain a given recall for different MC files with classifiers trained on $K^*J/\Psi(\to e^+e^-)$ MC

Again the classifiers assign higher predictions to the $K^*J/\Psi(\to e^+e^-)$ channel, thus strengthening the conclusion that signal events from this MC file look more like typical signal. The difference observed in the $K^*J/\Psi(\to e^+e^-)$ MC file might very well be due to cuts and not due to the channel being different. Information on this is still pending. The fact that the $K^*e^+e^-$ file used for training gives a different curve than another $K^*e^+e^-$ MC file suggest that cuts are indeed responsible.

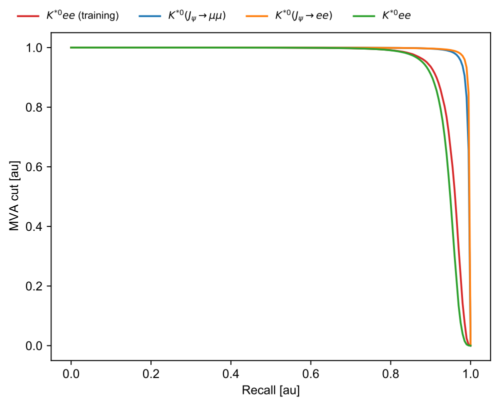
Another disparity common to all classifiers is, that the $J/\Psi(\to e^+e^-)$ channel receives higher predictions than the $J/\Psi(\to \mu^+\mu^-)$ channel. Since the algorithms are trained on e^+e^- data, this is not especially surprising as we expect muons to have a different signature than electrons. In order to make sure that this holds, it was also tested by training on $J/\Psi(\to \mu^+\mu^-)$ files and redoing the plots.



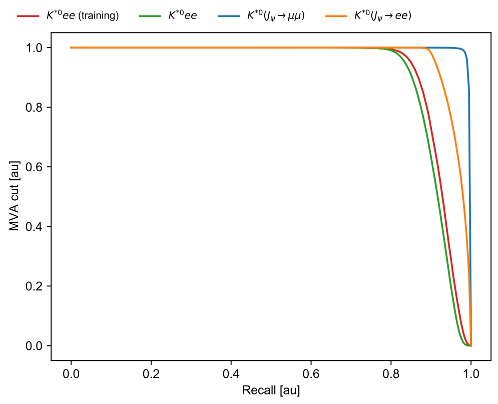
(a) for the BNN



(b) for the LightGBM ensemble



(c) for the NN



(d) for the XGBoost ensemble

Fig. 4.13: The MVA cut required to obtain a given recall for different MC files with classifiers trained on $K^*J/\Psi(\to \mu^+\mu^-)$ MC

The results for this cross-check are surprising. There is a significant difference between the NN/BNN and the BDTs. For the BDTs there is a drastic difference between the $J/\Psi(\to \mu^+\mu^-)$ channel and the $J/\Psi(\to e^+e^-)$ channel (the results for scikit-learn are comparable to those for LGBM in figure **??**). In comparison the NNs are way better at generalising to the $J/\Psi(\to e^+e^-)$ channel. In fact, they still give higher predictions to this channel. This suggests that NNs are less prone to overfitting on the $J/\Psi(\to \mu^+\mu^-)$ file. Checking the feature distribution reveals that the MC has a different cut on the $L_{Min\ PT}$ feature (see figure 4.14). Because this opens up an area where background events can be neatly separated from signal events, the BDTs will most likely learn to make a decision that classifies events below the cut-off in $L_{Min\ PT}$ as background. This decision does not work that well for other channels, thus causing the poor generalisation. The NNs might be less prone to this problem because they do not work with sharp cut-offs and will not place as much of an emphasis on a single feature decision. The presented hypothesis is difficult to prove because both types of classifiers are somewhat of a black box, but it seems highly probable. When one is working with files that have different cuts applied, it thus might be advisable to use NNs instead of BDTs.

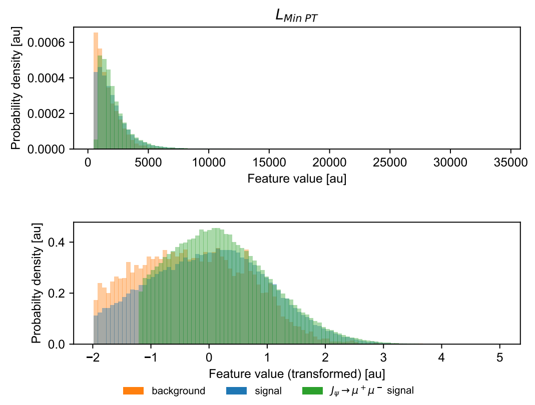


Fig. 4.14: The distribution in the $L_{Min\ PT}$ feature of signal and background from the labelled data set compared to the $J/\Psi(\to \mu^+\mu^-)$ MC file

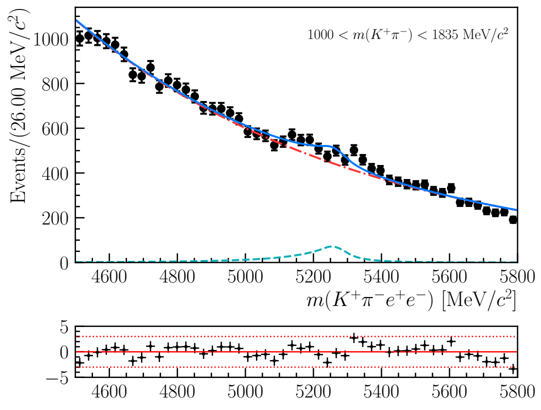
4.4 Usability for Further Analysis

The mass fits from the analysis were redone using the prediction outputs from different classifiers. For all of the plots, events with predictions lower than a certain threshold (0.999) were cut out to obtain a cleaner signal peak. To make sure that the results are comparable, the predictions of the classifiers were shifted, such that all classifiers have the same recall on the $K^*e^+e^-$ MC training file when cutting at 0.999. Since the mass fits are for $B^0 \to K^+\pi^-e^+e^-/\mu^+\mu^-$ it would have been better to match the cuts on a $K^+\pi^-e^+e^-$ MC file, but no file with the required features was available.

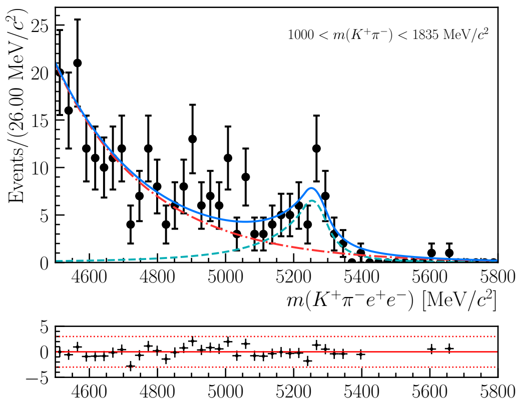
Fits for 4 different cases were plotted:

1. $B^0 \to K^+\pi^-e^+e^-$ with lower energy hadrons $1000 < m(K^+\pi^-) < 1835\,\mathrm{MeV/c^2}$
2. $B^0 \to K^+\pi^-e^+e^-$ with higher energy hadrons $1895 < m(K^+\pi^-) < 2600\,\mathrm{MeV/c^2}$
3. $B^0 \to K^+\pi^-\mu^+\mu^-$ with lower energy hadrons $1000 < m(K^+\pi^-) < 1835\,\mathrm{MeV/c^2}$
4. $B^0 \to K^+\pi^-\mu^+\mu^-$ with higher energy hadrons $1895 < m(K^+\pi^-) < 2600\,\mathrm{MeV/c^2}$

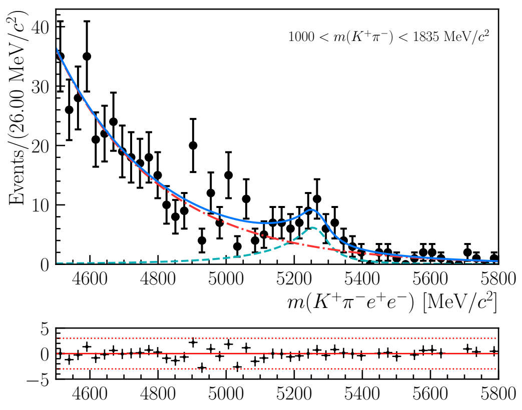
The results for $K^+\pi^-e^+e^-$ with lower energy hadrons are shown in figure 4.15, those with higher energy hadrons in figure 4.16. For the $K^+\pi^-\mu^+\mu^-$ channel only a few selected plots are shown in figure 4.17 because they do not provide many new insights. It is not easy to interpret, what the mass fits tell us about the classifiers that were used to obtain them. There are some straightforward measures of fit quality, such as having a more visible peak, more reasonable residuals and lower uncertainties. These measures of quality however do not guarantee that the classifier that were used perform better. A classifier might for instance skew the results of the fits, which could not be seen in the plots. This must be kept in mind when considering the plots and the conclusions drawn from them.



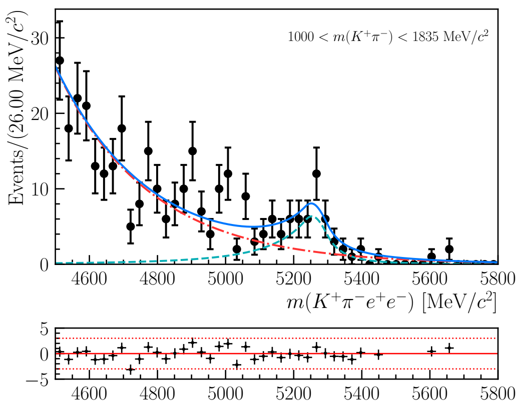
(a) without MVA



(b) with given MVA

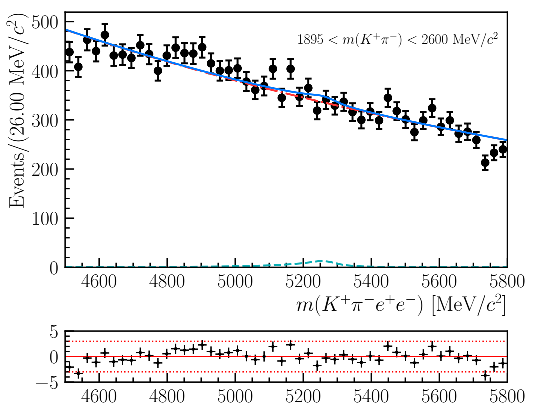


(c) with BNN MVA

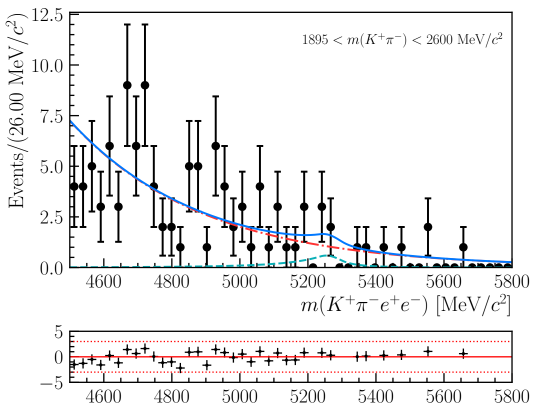


(d) with LightGBM ensemble MVA

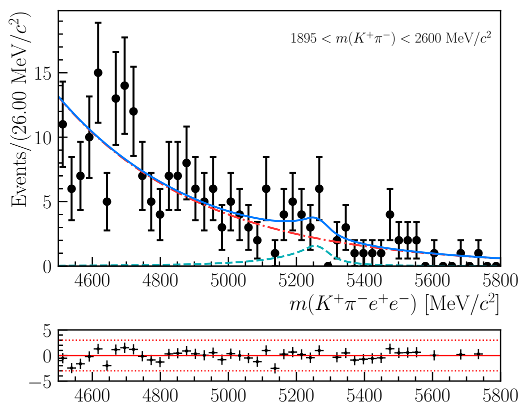
Fig. 4.15: Mass fit for the $B^0 \to K^+\pi^-e^+e^-$ channel with $1000 < m(K^+\pi^-) < 1835\,\mathrm{MeV/c^2}$



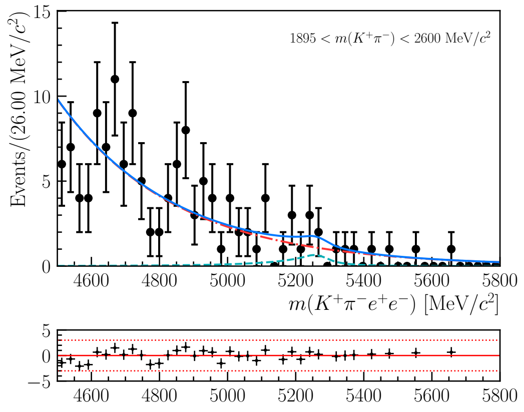
(a) without MVA



(b) with given MVA



(c) with BNN MVA

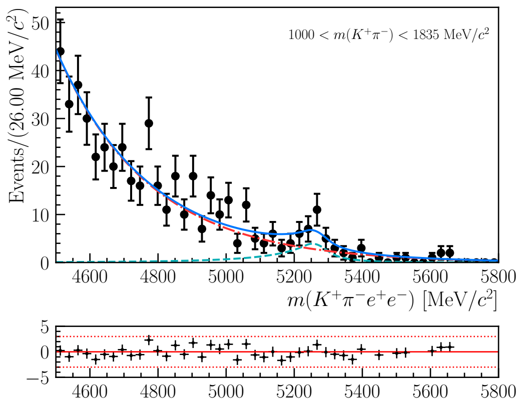


(d) with LightGBM ensemble MVA

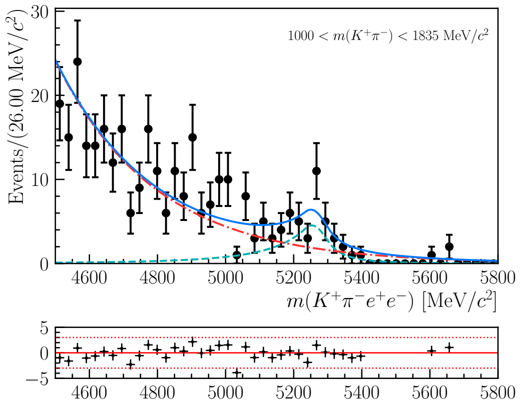
Fig. 4.16: Mass fit for the $B^0 \to K^+\pi^-e^+e^-$ channel with $1895 < m(K^+\pi^-) < 2600\,\text{MeV/c}^2$

Figure 4.15a illustrates how a classification into signal and combinatorial background is absolutely necessary to get a meaningful fit. The fits for LightGBM are really similar to the ones obtained with the given algorithm. It does not look like the performance increase on the labelled data translates into a performance increase on real data.

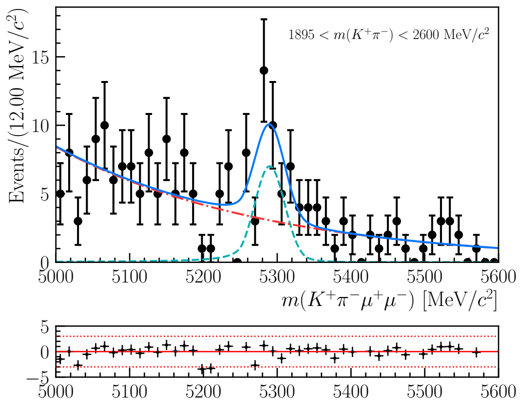
The BNN tends do give slightly worse results (see for instance figure 4.15c) except for the higher energy region (figure 4.16c). In this fit, the BNN as well as the NN give a more visible peak, this is not seen for the $\mu^+\mu^-$ channel though. Because of this and since there are very little events, this can not be considered a meaningful results. Further investigations would be needed to determine if NNs are advantageous in this energy region.



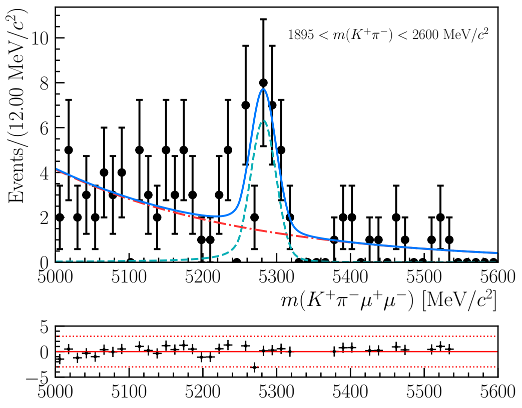
(a) with XGBoost ensemble MVA



(b) with scikit-learn ensemble MVA



(c) with XGBoost ensemble MVA



(d) with scikit-learn ensemble MVA

Fig. 4.17: Comparison of the mass fits obtained with the XGBoost ensemble and the scikit-learn ensemble for the $B^0 \to K^+\pi^-e^+e^-$ channel with $1000 < m(K^+\pi^-) < 1835\,\text{MeV/c}^2$ (4.17a and 4.17b) and the $B^0 \to K^+\pi^-\mu^+\mu^-$ channel with $1895 < m(K^+\pi^-) < 2600\,\text{MeV/c}^2$ (4.17c and 4.17d)

Quantitative measures for the goodness of the fits can be found in the signal/background yields and their respective uncertainties. They have been compared in figure 4.18 and roughly match the observations made so far.

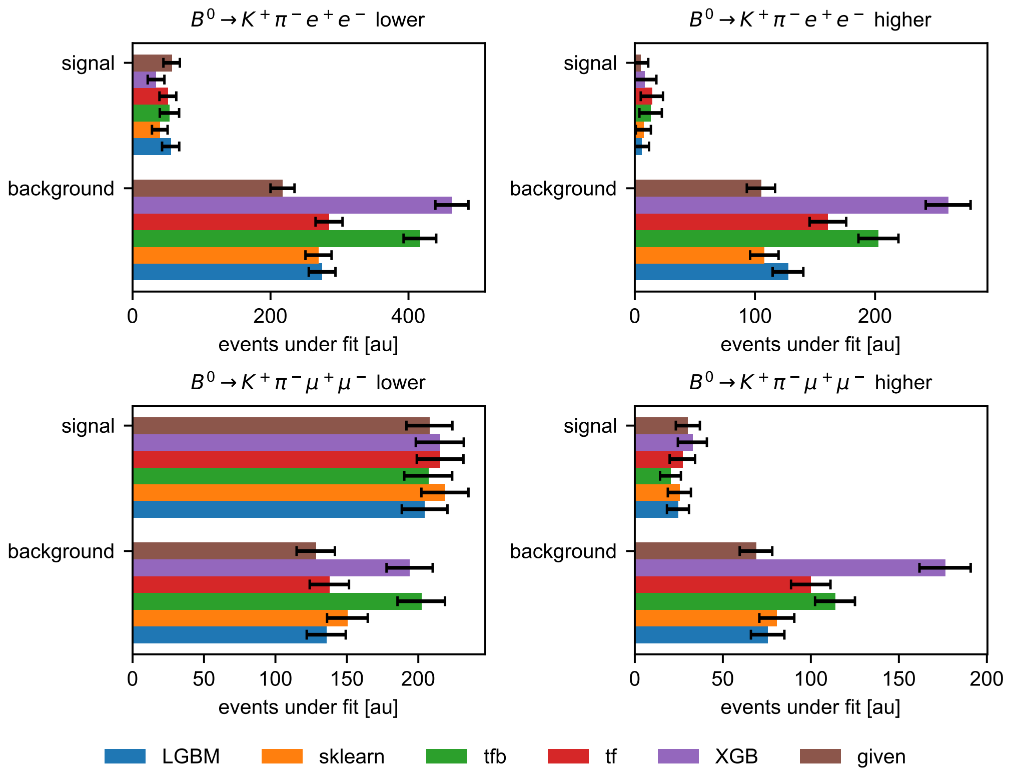


Fig. 4.18: The signal and background yields for fits of data that was cut with MVA scores from different classifiers.

5 Conclusion

It was found that BDT ensembles and BNNs give comparable uncertainties and predictions. Larger differences only showed up for types of unknown events, that are drastically different from what one would measure. Both for the BDT ensembles and the BNN, the outputs do make some unknown events identifiable from their distributions in the prediction-uncertainty space. They can however be separated somewhat cleanly if the type of unknown events are known and can be trained on. This separation works about as well for the BDT ensembles and the BNN, with advantages on the BDTs side for white noise.

All classifiers were found to give higher predictions to certain datasets, suggesting that events in these files are actually easier to differentiate from background, which might just be due to different cuts though. Training on $J/\Psi(\rightarrow \mu^+\mu^-)$ gave weak evidence that NNs are better at generalising onto data sets with different cuts. For future research a mix of different MC signal files for the training data to improve the generalisation power of the algorithms could be used. Similarly one could try to add events with noisy features (such as the ones generated for this thesis) into the training data to try and improve generalisation to real data.

When looking at the mass fits obtained with the different classifiers, the obtained results look similar and for a good comparison, further studies would be required. There are some areas where further investigations might shine light on possible use cases for BNNs, but they did not prove to be beneficial so far. Because of the much more involved tuning that is needed to obtain a working BNN, ensembles of BDTs seems to be the more reasonable choice for this kind of analysis.

Acknowledgements

I want to thank the research group for their helpful comments during our weekly meetings. In particular I want to thank Nathan Baudis of his cooperation in redoing the mass fits and his patient explanation of his work. Finally I want to thank Jonas Eschle for his thorough corrections on drafts of the thesis and for the thoughtful discussions.

Bibliography

[1] Laurent Valentin Jospin et al. “Hands-on Bayesian Neural Networks - a Tutorial for Deep Learning Users”. In: *ACM Computing Surveys* 1.1 (2020), pp. 891–921.

[2] The LHCb Collaboration et al. “The LHCb Detector at the LHC”. In: *Journal of Instrumentation* 3.08 (Aug. 2008), S08005–S08005. DOI: 10.1088/1748-0221/3/08/s08005. URL: https://doi.org/10.1088/1748-0221/3/08/s08005.

[3] Jake Hoare. *What is a Decision Tree?* 2018. URL: https://www.displayr.com/what-is-a-decision-tree/ (visited on 11/26/2020).

[4] Aarshay Jain. *Complete Machine Learning Guide to Parameter Tuning in Gradient Boosting (GBM) in Python.* 2016. URL: https://www.analyticsvidhya.com/blog/2016/02/complete-guide-parameter-tuning-gradient-boosting-gbm-python/ (visited on 01/10/2021).

[5] Jesse Johnson. *General regression and over fitting.* 2013. URL: https://shapeofdata.wordpress.com/2013/03/26/general-regression-and-over-fitting/ (visited on 11/16/2020).

A BDT Hyperparameters

Hyperparameter	Default Value	Optimised Value
learning_rate	0.1	0.15
n_estimators	100	70
max_depth	3	9
min_samples_split	2	600
min_samples_leaf	1	150
max_features	None	8

Table A.1: The hyperparameters that were tuned for the scikit-learn BDT and their corresponding default values

For XGBoost and LightGBM the hyperparameter tuning was done quickly by hand, only for scikit-learn a more methodological approach was tried. For scikit-learn a succession of Grid Searches was performed to find the best values of the hyperparameters across different

Hyperparameter	Default Value	Optimised Value
learning_rate	0.3	0.1
n_estimators	100	1000
max_depth	6	5
subsample	1	0.8
colsample_bytree	1	0.8

Table A.2: The hyperparameters that were tuned for the XGBoost BDT and their corresponding default values

Hyperparameter	Default Value	Optimised Value
num_leaves	31	2^8
min_data_in_leaf	20	300

Table A.3: The hyperparameters that were tuned for the LightGBM BDT and their corresponding default values

Hyperparameter	Default Value	Optimised Value
nodes per layer	–	14/40/60/60/1
optimizer	–	Adam
learning_rate	0.001	0.0001
training epochs	–	400 (at most)
loss function	–	binary cross-entropy
activation function	–	leaky relu
output layer activation	–	sigmoid
batch size	–	64

Table B.1: The hyperparameters of the TensorFlow NN and the corresponding default value (if available)

intervals. The method is described in more detail in [4].

B Hyperparameters of the Neural Network

C Hyperparameters of the Bayesian Neural Network

Hyperparameter	Default Value	Optimised Value
nodes per layer	–	14/40/60/60v/1
optimizer	–	Adam
learning_rate	0.001	0.0001
training epochs	–	400 (at most)
loss function	–	binary cross-entropy
activation function	–	leaky relu
output layer activation	–	sigmoid
batch size	–	64
prior shape	–	Normal
posterior shape	–	Normal

Table C.1: The hyperparameters of the TensorFlow Probability BNN and the corresponding default values (if available). A v in the nodes per layer indicated that a layer implements variational inference (see sec 2.3)

D Training and Prediction times

Because prediction times are basically constant across the amount k of folds and training time is linear for high k, we can simply compare the times for 16-fold CV. As table D.1 shows, LightGBM is in a league of its own when it comes to training time. XGBoost could also easily outperform sklearn, but the tuning of the hyperparameters is really computationally expensive.

	Prediction Time [s]		Training Time [s]	
Library	without tuning	with tuning	without tuning	with tuning
sklearn	0.12	0.22	928	1061
XGBoost	0.18	1.09	123	703
LightGBM	0.16	0.38	10	23

Table D.1: The training and prediction times of the different BDTs for 16 fold CV averaged across 20 seeds

The prediction time might seem irrelevant compared to the training time, but this is somewhat misleading. For the table above only the labelled data was predicted on, which makes up about 2% of the total data. For sklearn and XGBoost the total prediction time still does not make a big difference, but for LightGBM it roughly doubles the total computation time.