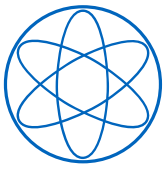


Department of Physics
Technical University of Munich

Investigation of Machine Learning Algorithms to Predict the Synchrotron Peak of Blazars

Bachelor's Thesis

Tobias Kerscher
(03697278)

Examiner:	Prof. Dr. Elisa Resconi
Advisor:	Dr. Theo Glauch
Submission Date:	16th August 2021

"All models are wrong, but some are useful."

- George E. P. Box

Contents

List of Figures

List of Tables

Acronyms

AGN Active Galactic Nuclei. iv, 1, 3, 4, 6, 33, 53

CART Classification and Regression Trees. 14, 16, 17

FCN Fully Connected Networks. iv, vi, 23, 24, 46, 54

FFN feedforward network. 22

GBRT Gradient Boosted Regression Trees. 18

GRU Gated Recurrent Units. 25

HSP High-energy cutoff blazar. 6

ISP Intermediate-energy cutoff blazar. 6

LASSO Least absolute shrinkage and selection operator. 19, 21

LSP Low-energy cutoff blazar. 6

LSTM Long Short-Term Memory. 25

MLP multilayer perceptron. 22

MSE Mean Squared Error. 10, 12, 16

OOB Out-of-Bag. 17, 49, 50, 52

RNN Recurrent Neural Network. iv, vi, 24, 46, 53, 54

RSS Residual Sum of Squares. 16

SED Spectral Energy Distribution. iv, 1, 3, 4, 5, 7, 27, 31, 33, 34, 35, 36, 37, 49, 52, 53, 54

SGD Stochastic Gradient Descent. 23

CHAPTER 1

Introduction

Especially in astronomy processing of large data sets has always been part of a physicist work. Ever since the forecast of the return of Halley's Comet in the 18th century computation based on large amount of data have been distributed on a dedicated group of workers, which were referred to as *computers* [Grier, 2013]. Even in the Apollo Missions in the middle of the 20th century with the uprising of electronics human computers were still employed to support them [Greicius, 2016]. Only in the late 20th century electronics became powerful enough to take over their work and ultimately their name.

While it might appear odd to choose a seemingly informatics based theme for a bachelor thesis in physics, it is actually a métier of physics older than informatics itself. In fact machine learning and especially nowadays popular deep learning are a fundamental part of a physicist repertoire. The *Large Hadron Collider (LHC)* at *Cern* in Genf for example produces too much data for a human to handle making the usage of artificial intelligences crucial [Guest, Cranmer, and Whiteson, 2018].

The central theme of this thesis are *blazars*, a subcategory of the *Active Galactic Nuclei (AGN) zoo*. It shows some of the most extreme features known in the universe such as a supermassive black hole, relativistic jets radiating well into γ-ray regions and superluminal velocities. Although they have been subject to many studies they are still not completely understood [Perlman, 2013]. With the uprising of machine learning and especially deep learning only recently have their methods been applied to blazars (e.g. [Fraga et al., 2020], [Kovačević et al., 2020]). However to our knowledge, while there have been classification models trained, there are no models to predict the actual synchrotron peak ν_{peak} from the *Spectral Energy Distribution (SED)*, which is the indicator the classification is based on.

The goal of this thesis is to apply machine learning methods to predict the synchrotron peak ν_{peak} with a prediction interval using the *Spectral Energy Distribution (SED)* of blazars. Because a best model or approach cannot be deduced beforehand and there are no other models to compare to, multiple models as well as different ways of data preprocessing are compared. The used algorithms consist not only of the popular deep learning but also the more traditional ones *random forest* and *gradient boosting* to see if the increase in complexity and thus labour actually results in an increased performance.

Chapter 2 introduces *AGN* including the *unified model* with an emphasis on blazars and concludes with numerical models to simulate the SED of a blazar.

Chapter 3 gives an overview of the used machine learning algorithms. While it was tried to keep a balance between too shallow and too detailed explanations, since this is still a physics thesis, it is not strictly necessary to have read this chapter to understand the results but it is recommended nevertheless. Since it requires no prior knowledge it should provide an easy entry to this field. The chapter dedicates one section per algorithm. Once can therefore also return to this chapter after examining the results and only read the sections one is interested in.

Chapter 4 presents the data set used in this thesis, explains and justify the data preprocessing including data augmentation that was done on it, followed by showing different types of feature sets used to train the models later on, consisting of *binning*, *compressed sensing* and an *autoencoder*.

Chapter 5 shows and evaluates the trained models using histogram of their predictions and plots of prediction intervals. There was one model per feature set and algorithm trained thus in total 9 models. It ends with a comparative summary of the models using various metrics.

Chapter 6 presents *BlaSE*, a ready to use tool for predicting the synchrotron peak of blazars including a 95% prediction interval based on the results of this work. The new tool is then applied to the *4LAC-DR2* catalogue.

Chapter 7 ends the thesis not only with a conclusion and an outlook, but also with some suggestions for future works on this topic.

CHAPTER 2

Blazars

Blazars are among the most extreme objects in the universe being powered by supermassive black holes [Padovani et al., 2017] and are the central theme of this thesis. They show interesting properties making them even candidates for neutrino and comsic-ray sources needed in the developing research area of multimessenger astronomy [Petropoulou et al., 2020].

This chapter first introduces the blazar as a specimen of the *AGN zoo* on the basis of the *unified model*, followed by numerical methods to model their emission behavior as described by their *Spectral Energy Distribution (SED)*.

2.1 Blazars as a Subclass of AGN

Already in 1907 has E. A. Fath documented his observation of the spiral nebula NGC 1068 which stood out from all other nebulas he has observed by showing not the expected star like absorption lines but dominant emission lines typical for planetary nebula [Perlman, 2013] and is today even considered to be a candidate for a neutrino source as indicated by the data of the IceCube experiment [IceCube Collaboration et al., 2019]. Discoveries of similar objects followed, whose properties were systematized by Seyfert in 1943 and are thus now known as *Seyfert galaxies*. Most of their radiation is produced by their nuclei and shows exceptionally broad emission lines. This led to the hypothesis, that there is a distinct class of galaxies, which is now known as *Active Galactic Nuclei (AGN)*.

The class of AGN holds a variety of different objects, thus often the term of an *AGN zoo* is used. However, they all share intrinsic properties such as the earlier mentioned broad emission lines and the high luminosity from the nuclei, which actually appears superluminal due to relativistic effects, but also a high variability and a continuos radiation starting from the infrared up to in some cases even γ-rays [Perlman, 2013]. The Existence of these similarities despite their various subtypes led to the development of the *unified model* by Urry and Padovani, 1995, which is illustrated in figure 2.1.

It is nowadays widely accepted, that AGN are driven by a still growing supermassive black hole ($\gtrsim 10^6 M_\odot$) in the center of the galaxy, which is fed by an accretion disk. This explains the high luminosity and variability as it allows up to half the gravitational energy to be converted into radiation and heat. The disk is surrounded by an optical thick "*torus*", which explains the behavior of many AGN in the infrared

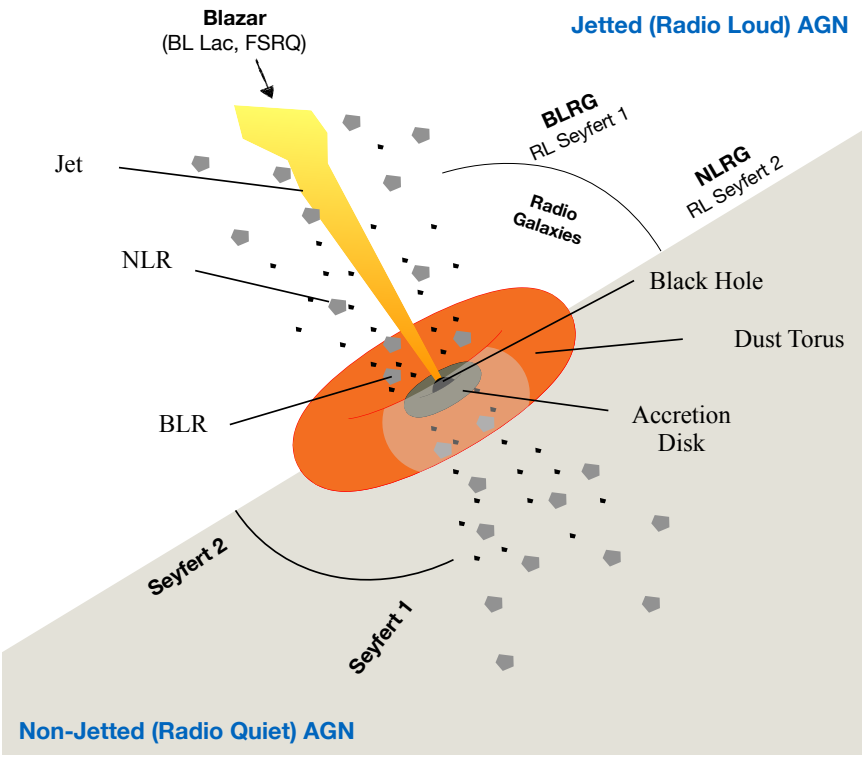


Figure 2.1: Illustration of AGN as described by the unified model featuring both jetted and non-jetted version. At different viewing angles are further subclasses marked. Kindly provided by Theo Glauch.

[Padovani et al., 2017]. While the form of a torus is a valid assumption, it is by no mean necessary and the name is historically based. In fact, newer data suggests that a patchy structure is more likely. The nucleus is further enveloped by warm gas, which explains the emission lines features. Only around 10-20% of all AGN also posses a highly relativistic jet, dividing the zoo into *jetted* and *non-jetted* AGN, often also referred to as *radio loud* and *radio quiet*. Further subdivision are highly dependent on the viewing angle [Perlman, 2013].

An especially interesting although rather rare subclass are jetted AGN with a small viewing angle ($\lesssim 15 - 20°$) and thus pointing the jet towards us known as *blazars* [Padovani et al., 2017]. The jet's output is strong and therefore dominates the radiation making the emission lines practically invisible. The radiation goes well into γ-ray frequencies, were many blazars deposit most of their radiation energy and were in fact for the longest time the only known extragalactic source of radiation in the GeV and even TeV region. Figure 2.2 shows an *Spectral Energy Distribution (SED)* of the blazar *PKS 2155-304* as an example, featuring the broad emission spectrum and the especially in X-ray present high variability. It also shows the characteristic double bump. Figure 2.3 shows a model representation of an imaginary blazar to make the bumps more obvious. The first bump is produced by synchrotron radiation caused by relativistic electrons in the jet, while the second is mainly caused by inverse compton scattering, i.e. the photon gaining energy from free electrons. The necessary seed photons can originate from various sources such as the synchrotron radiation, thermal radiation from the surrounding gas or even the cosmic microwave background [Perlman, 2013]. Another possible explanation for the second bump are hadronic processes like pion decay [Padovani et al., 2017].

Figure 2.2: SED of the blazar PKS 2155-304 using data generated by the *VOUBlazar* tool. It shows strong radiation over a broad frequency range up to several hundred GeV with especially high variability in the X-ray region.

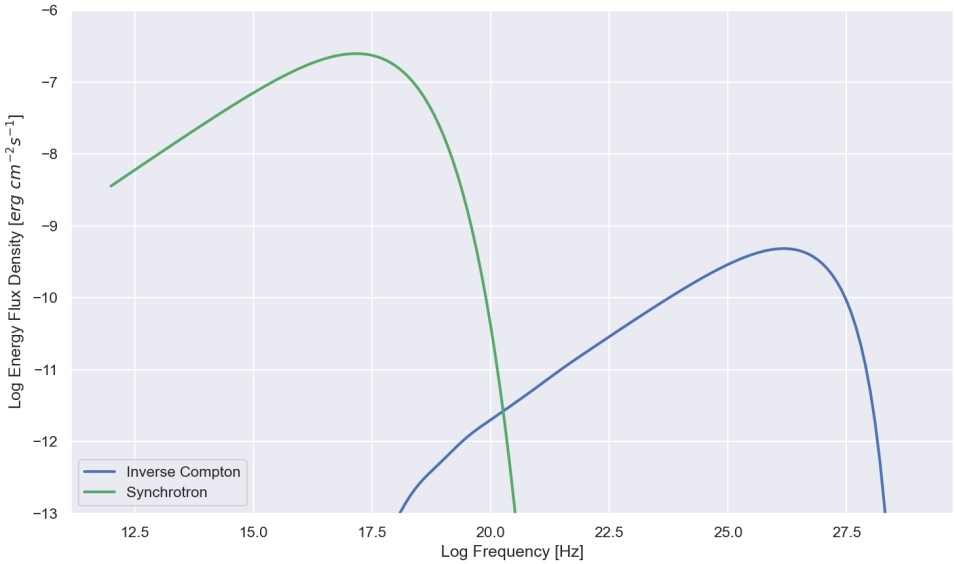


Figure 2.3: Modeled SED of a blazar. It shows the intrinsic two bumps produced by synchrotron and inverse compton radiation. Adapted from Zabalza, 2015

.

Category	Frequency range
Low-energy cutoff blazar (LSP)	$\nu_{peak} \leq 10^{14}$Hz
Intermediate-energy cutoff blazar (ISP)	10^{14}Hz $\leq \nu_{peak} \leq 10^{15}$Hz
High-energy cutoff blazar (HSP)	$\nu_{peak} \geq 10^{15}$Hz

Table 2.1: Classification of blazars based on their synchrotron peak ν_{peak} (Padovani et al., 2017).

The peak of the first bump, i.e. of the synchrotron radiation ν_{peak} is commonly used to further subdivide blazars into three categories as listed in table 2.1.

2.2 Numerical Models

The numerical modeling of blazars are a tedious and complex problem resulting in models being hand-tailored for specific objects (e.g. Petropoulou et al., 2020, Cerruti et al., 2018). The process is further complicated by the fact that blazars and even AGN in general are not completely understood, yet [Padovani et al., 2017]. However, since here the models won't be used to fit existing objects, but rather to simulate non-existing blazar-like objects, more simple models are already sufficient. Here the python package *naima* was used, whose models will be briefly introduced in the following [Zabalza, 2015].

As mentioned in the previous section, the black hole is surrounded by layers of gas, which is needed for not only the synchrotron radiation, but also the inverse compton scattering. Therefore, one must model the corresponding particle distribution for which *naima* provides several empirical functions mapping particle energy to its density. The following lists the distribution functions provided by *naima* each having additional empirical parameters and containing its predecessor as special case.

- **Power Law**

$$f(E) = A \left(\frac{E}{E_0}\right)^{-\alpha} \tag{2.1}$$

- **Log Parabola**

$$f(E) = A \left(\frac{E}{E_0}\right)^{-\alpha-\beta\log\left(\frac{E}{E_0}\right)} \tag{2.2}$$

- **Exponential Cutoff Power Law**

$$f(E) = A \left(\frac{E}{E_0}\right)^{-\alpha} \exp\left[-\left(\frac{E}{E_{cutoff}}\right)^{\beta}\right] \tag{2.3}$$

- **Exponential Cutoff Broken Power Law**

$$f(E) = \exp\left[-\left(\frac{E}{E_{cutoff}}\right)^{\beta}\right] \begin{cases} A \left(\frac{E}{E_0}\right)^{-\alpha_1} & : E < E_{break} \\ A \left(\frac{E_{break}}{E_0}\right)^{\alpha_2-\alpha_1} \left(\frac{E}{E_0}\right)^{-\alpha_2} & : E > E_{break} \end{cases} \tag{2.4}$$

Naima obviously also includes radiation models taking the particle distribution and

Name	Temperature	Energy Density
Cosmic Microwave Background	2.72 K	0.261 eV cm^{-3}
Near Infrared	30 K	0.5 eV cm^{-3}
Far Infrared	3000 K	1 eV cm^{-3}

Table 2.2: Default photon sources as provided by naima [Zabalza, 2015]

Property	Value
Exponential Cutoff Power Law	
A	1×10^{36} eV^{-1}
E_0	1×10^{12} eV
α	2.1
E_{break}	1.3×10^{13} eV
β	1.0

Property	Value
Synchrotron	
B	1×10^{-8} T
Inverse Compton	
Sources	CMB, NIR, FIR
SED	
Distance	1.5 kpc

Table 2.3: Properties of example blazar in figure 2.3

additional parameters to produce the corresponding components of the SED. There are in total four radiation models provided, however only the synchrotron radiation and inverse compton are of interest here. The synchrotron model is parameterized by the isotropic magnetic strength, the inverse compton model by a list the photon sources. The photon source can either be specified as thermal radiation of specific temperature or monochromatic radiation of specific energy alongside the sources energy density. There are three default values for the photon source as listed in table 2.2. Both models additionally allow to specify the minimum and maximum electron energy as well as the blazar's distance. An example of an model produced by using *naima* is shown in figure 2.3 using parameters as described in table 2.3.

CHAPTER 3

Machine Learning

The origins of machine learning can be dated back to October 1950 when Alan Turing brought up the question wether machines can think [Turing, 1950]. Already 8 years later Rosenblatt tried to mimic the human brain by creating a neural network using *perceptron* as building blocks [Rosenblatt, 1958]. Machine learning has always been an active field of study, but only recently since computers became powerful enough has the research gained momentum. One of the better known breakthroughs are IBM's *Deep Blue*, beating world chess champion Garry Kasparov in 1997 [Harding and Barden, 2011] and Google's *AlphaGo* beating Go master Lee Se-dol in 2016 [Borowiec, 2016].

Nowadays machine learning is often referenced by the buzzwords *Big Data* and *Artificial Intelligence* and especially the younger generations interact with it on a daily basis. According to *Dimensions* in 2020 over a quarter a million publications were made in the field of *Artificial Intelligence and Image Processing* [Digital Science, n.d.]. In the recent years a lot of books about the topic of machine learning have been published: Hastie, Tibshirani, and Friedman, 2009; Bishop, 2016; Goodfellow, Bengio, and Courville, 2016; Du and S., 2019; Brunton and Kutz, 2019; Berk, 2020.

This chapter is based on these books and starts with a brief introduction to the general concept of machine learning, followed by explaining the algorithms used in this thesis.

3.1 Basic Concepts

At the heart of any machine learning algorithms is data. Mathematically speaking can the data be seen as a tensor, with either fixed or variable dimension. The latter is often referenced as *sequential data*. A grayscale picture for example is also data usually stored as a matrix consisting of pixel, where 0.0 means black and 1.0 white. A video adds time as a third dimension. If its duration is not fixed it can also be seen as sequential data. For now we will focus on one dimensional data. Obviously, a single data point (or picture to keep the frame) is not enough to learn anything, thus one needs a whole *data set*. An especially prominent data set in the machine learning community is about the casualties in the titanic accident, even leading to an open competition [Kaggle, 2012]. Table 3.1 shows an excerpt of the actual dataset. Each passenger is represented by a vector containing information also known as a set of *features* such as sex, age and class one booked and maps it to the *response*, in

Surv.	Class	Name	Sex	Age	Siblings/ Spouse	Parents/ Children	Fare (£)
No	3	Mr. Owen Harris Braund	m	22	1	0	7.25
Yes	2	Miss. Emily Rugg	f	21	0	0	10.5
Yes	3	Miss. Laina Heikkinen	f	26	0	0	7.925
Yes	1	Mr. Hugh Woolner	m	46	0	0	35.5
No	3	Mr. William Henry Allen	m	35	0	0	8.05

Table 3.1: Excerpt of the Titanic dataset taken from Stanford University, 2016

this case their survival.

Consider a dataset consisting of the input X and the response Y. We assume that the dataset was produced by a *true model* f

$$Y = f(X) + \epsilon \tag{3.1}$$

where ϵ is random noise with the expected value $E[\epsilon] = 0$ often assumed to be normal distributed. The goal of machine learning is now to learn a *model* $\hat{f}$ to approximate the true model to get an estimation $\hat{y}$ of the true response in a process called *training*. Because even a perfect approximation leaves an error of ϵ it is also known as *irreducible error*. The various methods of training a model can be divided into three categories:

- **Supervised Learning**
  Input and response are known beforehand. The response is used as *ground truth* to train the model by reducing a *loss function*. We can further subdivide this category based on the response. If the response is discrete it's called a *classification* problem and a possible loss function is the *logistic regression*:

  $$L(y, \hat{y}) = -\frac{1}{n}\sum_{i=1}^{n} y_i \log(\hat{y}_i) - (1 - y_i)\log(\hat{y}_i) \tag{3.2}$$

  If on the response is continuous, it's called a *regression* problem and a possible loss function is the *Mean Squared Error (MSE)*:

  $$L(y, \hat{y}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{3.3}$$

  The index i denotes the i-th sample. It is common to normalize the loss function to keep it comparable across different data sets.

- **Unsupervised Learning**
  Only the input is known, the response and thus a ground truth is not available. Instead the training tries to enforce a specific property on the response. The loss L is thus how well this property is fulfilled. For example creates *Principal Component Analysis (PCA)* as the name suggests *principal components* as

response. Another example is *sparse coding*, where the goal is to find a basis in which the input is sparse. There the response is the sparse encoding. There also exist unsupervised classification.

- **Reinforcement Learning**
  Neither the input nor the response is known beforehand. Instead they are discovered by the algorithm by performing actions on its own. The response is given as feedback to the action hence the reinforcement. This feedback is often depicted as a score like in a game. It's increased if a action should be repeated and decreased if it should be avoided. The model here often called a *agent* tries to maximize its score, thus the loss function L would be the negative score. This type of machine learning is useful in scenarios where the environment can change on itself or as a reaction to the model. Autonomous driving is a prominent example of such a algorithm. The agent discovers more of its surroundings as it moves. If it obeys the traffic law it gets positive feedback, if it causes damage it gets negative feedback, ultimately improving its driving skills through experience.
  We will ignore this type of machine learning as we won't be using it in this thesis.

The performance of the training process scales with the number of samples, however often increasing the data set size is not feasible or even possible. One can artificially increase the dataset by *data augmentation*, i.e. creating new data samples based on already existing ones. For example one can randomly delete features or translate a picture.

The model can be described by its parameters θ. The goal of the training process is to find the optimal parameters θ^* by minimizing the loss function L. In the special case of supervised learning this can be described as

$$\theta^* = \arg\min_{\theta} L(y, \hat{f}(\theta; y)) \tag{3.4}$$

Once $\hat{f}$ is obtained it can either be used directly to deduce new information like how much impact each features had on the response also known as *feature importance* (did the sex influence the survival rate?), or to predict the response on unseen data (*generalization*).

To measure the accuracy of these entities one can use the *Bootstrap* method. Usually one needs multiple sets to calculate them, which are often not available. Bootstrap now simulates multiple sets by randomly drawing datasets with replacement, i.e. a single data point can occur multiple times. For example to calculate the variance of a function $g(X)$ on the dataset, one can use bootstrap to calculate its variance. By creating B bootstrap datasets X_i^* the variance is given as

$$Var(g(X)) = \frac{1}{B-1} \sum_{i=1}^{B} (g(X_i^*) - \bar{g}^*)^2 \tag{3.5}$$

$$\text{where} \qquad \bar{g}^* = \frac{1}{B} \sum_{i=1}^{B} g(X_i^*)$$

Figure 3.1 illustrates the method of bootstrapping.

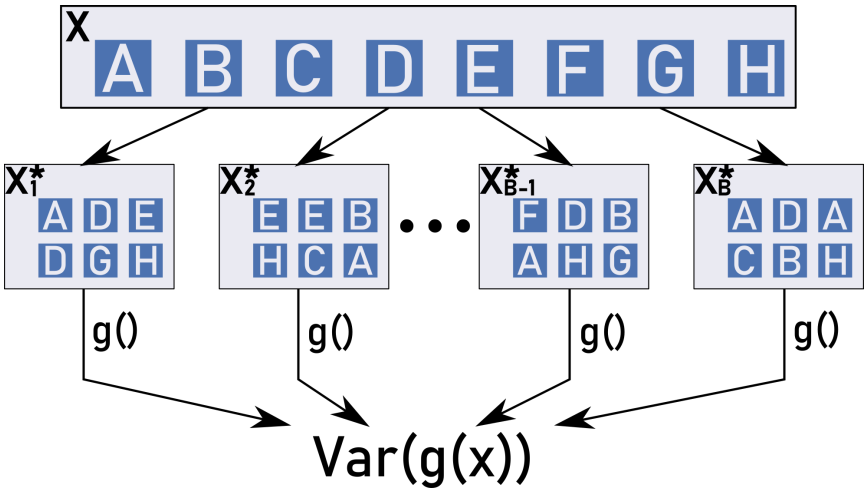


Figure 3.1: Illustration of bootstrapping.

In supervised generalization one always wants evaluate how well a model performs on unseen data the so called *generalization error*. Because the very nature of unseen data is that it's not available, the error can only be approximated. To determine the performance the dataset is usually split into a *training set* and a *test set*. The first one is used to train the model, the later functions as unseen data. It can be shown that the expected MSE on an unseen data can be decomposed into a sum

$$E[y_0 - \hat{f}(x_0)]^2 = \text{Var}(\hat{f}(x_0)) + (\text{Bias}(\hat{f}(x_0)))^2 + Var(\epsilon) \tag{3.6}$$

consisting of the following three parts:

- **Variance Error**
  Variance refers to how much the model would vary for different training sets. A high variance means that the model picked up noise from the dataset.
- **Bias Error**
  The bias is the error produced by simplifying the dataset. If the bias is high the model failed to generalize the dataset.
- **Irreducable Error**
  This one was mentioned earlier. It's the error intrinsic to the data itself and thus cannot be reduced.

Most models have additional *hyperparameters* that are not trained but rather set before the training process. These can be used to adjust bias and variance. The process of *hyperparameter tuning* consists of training a model and then calculating the error on unseen data. Using the test set to determine the hyperparameters should be avoided at any cost as this means it was indirectly used to train the model and thus can no longer be considered unseen.
Instead one either creates a third set often called *validation set*, or alternatively if the dataset is rather small *cross validation*. There exist multiple variants of this algorithm. The following describes what is commonly referred to as *Leave-one-out cross validation*. First the training set is split into N smaller sets (*folds*). The model is trained N times, always putting a different fold aside, which is then used to calculate the error. The total error is the average of these errors. Figure 3.2 shows a corresponding

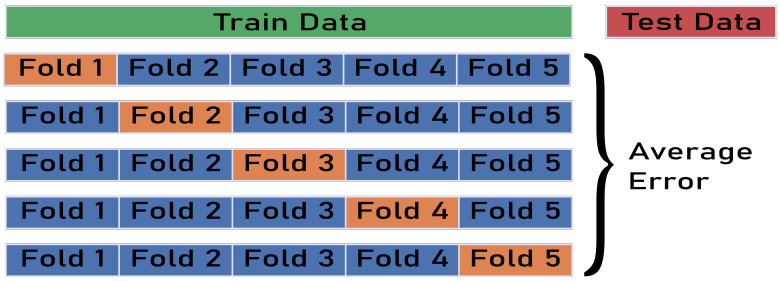


Figure 3.2: Schematic of the leave-one-out cross validation. The data set is split into a test (red) and train (green) set, the later is further split into 5 folds (blue/orange). The model is trained 5 times every time leaving a different fold out (orange). The errors of the 5 trained models is calculated for each using this fold and are finally averaged.

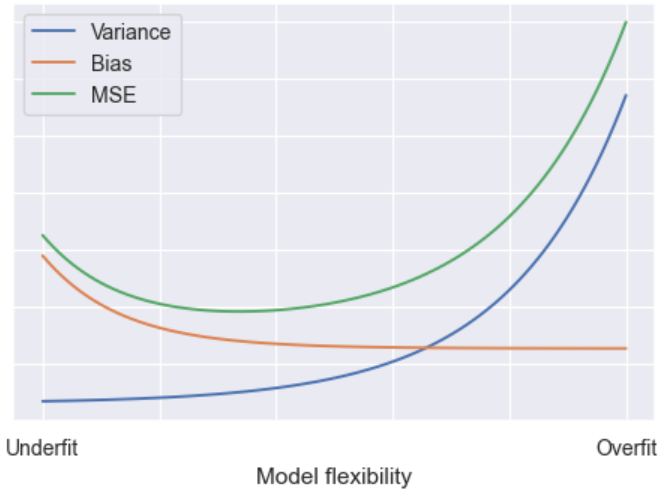
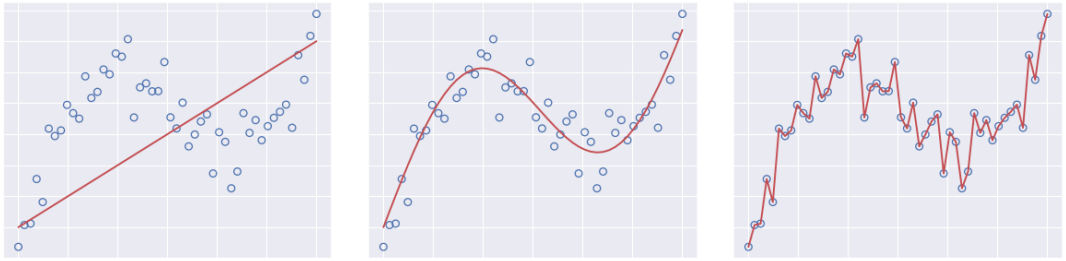


Figure 3.3: Bias-Variance tradeoff of an example model. With increasing model flexibility the bias de- and the variance increases. The minimum of the generalization error lies in between.

diagram. This process is usually repeated many times until a good set of hyperparameters is found. It is beneficial to create different folds for each run.

Ideally a model has low variance and low bias. However as depicted in figure 3.3 one usually increases while the other decreases. One therefore speaks of the *bias-variance-tradeoff*. The two extremes of this tradeoff creates another way to categorize the model error illustrated by figure 3.4:

- **Underfitting** (*low variance, high bias*)
  The model oversimplifies the data and thus fails to generalize. This happens more likely to simple models.
  In the titanic data set this would be for example the model only using the sex as criteria.
- **Overfitting** (*high variance, low bias*)
  The model follows the noise very closely and thus fails to generalize. This happens more likely to complex models.
  In the titanic data set this would be for example the model remembering the names of passenger.



(a) Underfit **(b)** Good Fit **(c)** Overfit

Figure 3.4: Bias-Variance tuning on an example model.

In the end no model can get rid of all error and thus perfect prediction are not possible. There are several methods to estimate the prediction interval, often specific to a machine learning algorithm. We will discuss them alongside the corresponding algorithms.

As we will see in the following chapters there are many machine learning algorithms. One might wonder which one is the best. However, the no-free-lunch-theorem states that there is no best algorithm [Wolpert and Marcready, 1997].

Theorem 3.1 (No-Free-Lunch Theorem [Du and S., 2019])
Given the set of all functions $\mathcal{F}$ and a set of benchmark functions $\mathcal{F}_1$, if algorithm A_1 is better on average than algorithm A_2 on $\mathcal{F}_1$. then algorithm A_2 must be better than algorithm A_1 on $\mathcal{F} - \mathcal{F}_1$.

3.2 Decision Tree

While overshadowed by later developed machine learning algorithms, decision trees are not only playing an important part in many of those algorithms but is also one of the most intuitive to understand and thus makes a great entry point. They can be seen as a type of flow chart with no recursion, i.e. a sequence of if-else questions leading to the result. Due to the branching at each decision and ending in a node, it is as usual in computer science depicted as an upside down tree, hence the name *decision tree*. Figure 3.5 shows a illustration of an actual decision tree trained on the titanic dataset.

One can easily create a decision tree if the underlying model has already been understood, which is usually not the case in machine learning. Here it is the goal to retrieve the necessary decisions from the data itself using features and the corresponding responses. It is thus a supervised algorithm. Because each decision splits the data in a *true* and a *false* set it can also be seen as a partition problem.There are many algorithms to create a decision tree. In the following the *Classification and Regression Trees (CART)* algorithm is introduced.

To keep things simple for the beginning a classification problem is used, therefore a *classification tree* will be grown. Starting with the whole dataset, first for each feature all possible splits are determined. Features can be put in the following two categories:

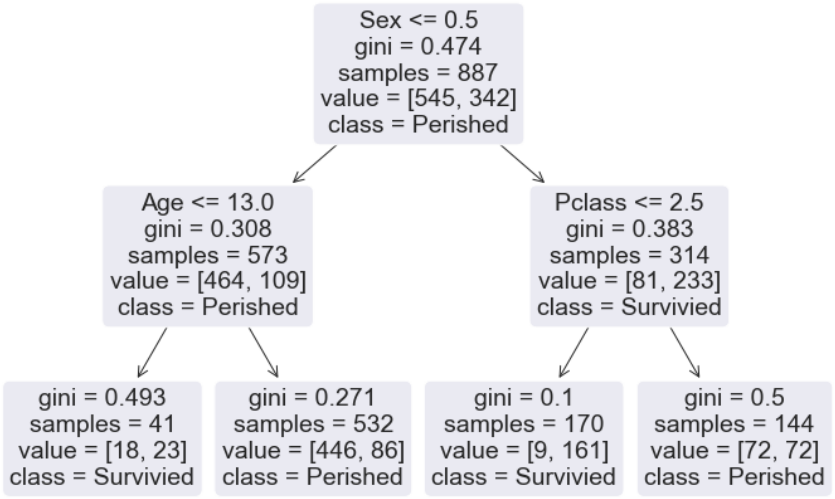


Figure 3.5: Decision tree for the titanic dataset with a maximal depth of two. Each node shows from top to bottom the condition, the gini impurity, number of samples in node, samples in each category and the assigned class. For the left subnode the condition holds, for the right not. Male was labeled as zero, female as one. Therefore the left half are male, the right are female passenger. Interrestengly sex had the most impact on survival chances.

- **Ordinal**
  The feature has a natural ordering, which is conserved by the algorithm. This can be for example the product rating on an online shop("good", "mediocre", "bad"). For k possible values there are therefore $k-1$ possible splits. Note that continuous variables like age or price also fall into this category. While there is an infinite amount of possible values, there is only a finite and thus discrete subset present in the dataset.
- **Categorial**
  The feature has no natural ordering and the algorithm must consider all possible combinations of values. For example the values (red, green, blue) can be either split into (red,green),(blue), (blue),(green,blue) or (green), (red,blue). For k possible values there are therefore $2^{k-1}-1$ possible splits.

For each new subset called a *node* the *impurity* is calculated, which indicates how heterogenous the subset respective to the response is. It is zero if all responses are equal and has a maximum for an equal distribution of all possible responses. There are multiple formulas to calculate the impurity, here the *Gini impurity* or *Gini index* is presented

$$I(A) = \sum_{i=1}^{k} p_i(A)(1 - p_i(A)) \tag{3.7}$$

where A is the node in question, $p_i(A)$ is the proportion of the i-th class in A and k is again the number of possible responses.

The impurity alone is not enough to determine the best split as surely a split in half is more beneficial than separating a single case. Instead the improvement of a split s

of a set A into A_L and A_R is defined as

$$\Delta I(s, A) = I(A) - p(A_L)I(A_L) - p(A_R)I(A_R) \tag{3.8}$$

where $p(A_i)$ is analogous to its previous definition the proportion of A_i in A.

CART considers all possible splits of all features and selects the split s with the biggest improvement $\Delta I(s, A)$. This process is repeated for each new formed node independently until it's either pure or a subset fulfills a certain criterion. Terminating nodes are called *leaf* and are assigned a response by a majority vote of the training samples inside it. Trees where all leafs are pure are called *fully grown*. A criterion can for example be:

- Depth of tree
- Amount of samples in subset
- Minimum improvement necessary

By extending the *CART* algorithm to regression problems, regression trees are obtained. Instead of a majority vote, the response assigned to a leaf is the average. Furthermore, the Gini impurity can no longer be used as the distances between responses would get distorted. Instead the impurity of a subset is calculated using the *Residual Sum of Squares (RSS)*

$$I(A) = \sum_i (y_i - \bar{y}(A))^2 \tag{3.9}$$

where $\bar{y}(A)$ is the mean response in A.

Note that RSS differs from MSE by the lack of normalization. The weighting of the impurity used to calculate the improvement of a split is therefore already implicitly present. The improvement simplifies to

$$\Delta I(s, A) = I(A) - I(A_L) - I(A_R) \tag{3.10}$$

Any unseen data that is dropped down the tree gets the response of the leaf assigned.

While decision trees are easy to understand and interpret, they are prone to high variance. Algorithms that build on decision trees and tackle this problem will be introduced in the following chapters.

3.3 Bagging and Random Forest

Bagging is a portmanteau and stands for *bootstrap aggregating*. As the name implies it's based on the bootstrap method and elevates its concept from measuring accuracy to training models. Instead of a single model a whole *ensemble* of multiple models is trained using a different bootstrap dataset each time. To get a prediction the result of each model in the ensemble is either averaged for regression problems or the majority vote is taken for classification problems.

For identical distributed random variables with variance σ^2 and positive pairwise correlation ρ it can be shown that variance of the average over N values is [Hastie, Tibshirani, and Friedman, 2009]

$$\rho\sigma^2 + \frac{1-\rho}{N}\sigma^2 \tag{3.11}$$

Averaging reduces the variance. Therefore bagging is especially useful for models with high variance and low bias such as decision trees. With increasing number of models in the ensemble the second summand vanishes, but the first remains. This is where *random forest* take over.

Random forest introduced in Breiman, 2001 also creates an ensemble of decision trees, but slightly varies the process of growing the trees. CART considers at each split all features. If there's a strong feature, i.e. one that divides the outcome exceptionally well like sex in the titanic dataset, it will most likely always be selected first and the trees become correlated (high ρ). Random forest restricts the number of features CART can choose from by randomly selecting less features than there are available at each split independently to reduce the correlation and thus ρ. Ultimately, this reduces the variance of the ensemble as shown before. How many features are chosen is a hyperparameter.

Due to the very nature of bootstrapping there exist for each model in the ensemble a set of data points which have not been used to train the model, hence the name *Out-of-Bag (OOB)*. They can be used to estimate the models' and thus the entires ensemble's *Out-of-Bag error* as an alternative to cross validation. As they are a byproduct of the training process they are computational faster.

Meinshausen, 2006 extended the concept of Random Forest in regression and created what he called *Quantile Regression Forest* which does not only predict the expected value of an input, but also its prediction interval. Only the algorithm itself and not the math behind will be discussed. First each training sample with input X_i and response Y_i is assigned a weight $w_{i,t}$ equal to the inverse number of total samples in the leaf $L_t(X_i)$ it belongs to independently for each tree t. For a new input X the α-quantile is then

$$Q_\alpha(X) = \inf_Y \sum_{i=1}^{n}\sum_{t=1}^{B} w_{i,t} 1_{\{L_t(X_i)=L_t(X)\}} 1_{\{Y_i \leq Y\}} \geq \alpha \tag{3.12}$$

The sum consists of the weights $w_{i,t}$ of all training samples among all trees that ended in the same leaf as the new input and have a response less or equal than Y. The quantile is then the infimum of Y with respect to the sum, which can be used as usual to create prediction intervals, e.g. a 90% prediction interval can be calculated as

$$I_{90\%}(x) = [Q_{0.05}(x), Q_{0.95}(x)] \tag{3.13}$$

3.4 Gradient Boosted Regression Trees

Like bagging *boosting* utilizes an ensemble of models to improve their performance, but trains the individual models sequential using knowledge from previous iteration. The training samples are weighted in the training process. After each model added to the ensemble the weights of correct predicted samples are reduced and the one of falsely predicted ones increased. Subsequently models are therefore used to predict hard to learn edge cases. Boosting furthermore uses *weak learners*, i.e. models

that are more prone to underfitting, which can get as extreme as decision trees with only two leafs, often referred as *stumps*.

Boosting can be applied to almost any machine learning algorithm, but decision trees are the common choice. The process of boosting is shown in the case of regression trees (*Boosted Regression Trees*). The first tree $\hat{f}^1$ is trained as usual. After each new tree $\hat{f}^b$ the residuals r_i used to determine the impurity in the decision tree are updated based on the previous residuals using the learning rate λ as a hyperparameter:

$$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i)$$

After B iteration and thus B trained trees the boosted model is as follows

$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x)$$

Gradient Boosted Regression Trees (GBRT) alternates the algorithm by utilizing the concept of *Gradient Descent*, i.e. functions are updated according to $f_{i+1} = f_i + \alpha \nabla L(f)$ with a loss function L and a learning rate α which converges to zero loss.

There are multiple possible loss functions that can be used in GBRT. Common are the L1 and L2 norm, but one can also use the Huber Loss to calculate quantile and thus get prediction intervals analog to the previous section

$$L(y, f(x)) = \begin{cases} [y - f(x)]^2 & \text{for } |y - f(x)| \leq \delta_m \\ 2\delta_m |y - f(x)| - \delta_m^2 & \text{otherwise} \end{cases}$$
$$\text{where} \quad \delta_m = \alpha\text{th-quantile}\{|y_i - f(x_i)|\}$$

One has to keep in mind, that because boosting focuses on hard to fit samples it will for arbitrary many iterations eventually start to overfit making the number of iterations a crucial hyperparameter. This distinguishes it from random forests as their performance saturates with increasing ensemble size. The overfitting can be dampened by using regularization, which is a technique introduced in the following sections. Notwithstanding, GBRT is one of the standard algorithm in machine learning capable of tackling even complex task. An example of particular interest to physicist might be ability to detect the higgs boson [Chen and He, 2014].

3.5 Compressed Sensing and Dictionary Learning

The goal of *Compressed sensing* is to reconstruct an unknown signal from randomly spaced measurement. It is therefore an unsupervised learning algorithm. The algorithm exploits the fact that most signals are sparse in a specific domain. For example is music sparse in the fourier space as chords have only specific possible frequencies. Even images tend to be sparse in the wavelet basis, which is actively used in magnetic resonance drastically increasing the patient throughput[Lustig et al., 2008]. Du and S., 2019 and especially Brunton and Kutz, 2019 give a good explanation of the math behind the algorithms in this chapter. The following is a summary of the key concepts based on those two books.

Given a signal $x \in \mathbb{R}^n$ with a sparse representation s in Ψ (e.g. Fourier-Basis) we can describe the measurements $y \in \mathbb{R}^p$ mathematically by a measurement matrix $C \in \mathbb{R}^{p \times n}$

$$y = Cx = C\Psi s \tag{3.14}$$

Assume that only the measurements are known. Then the signal can reconstructed by first calculating $\hat{s}$

$$\hat{s} = \arg\min_s \|s\|_0 \text{ subject to } y = C\Psi s \tag{3.15}$$

where $\|\cdot\|_0$ stands for the l_0 pseudo-norm, which is equal to number of non-zero entries, also known as *cardinality*.

The minimization of the l_0 norm is computational very expensive and thus not feasible in practice. It can however, if certain conditions on the measurement matrix C are holding, be relaxed to a convex l_1 minimization problem

$$\hat{s} = \arg\min_s \|s\|_1 \text{ subject to } y = C\Psi s \tag{3.16}$$

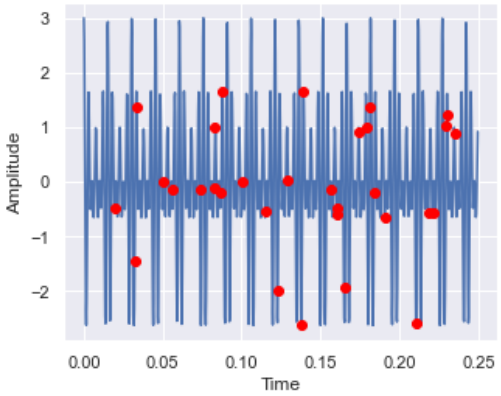
A popular algorithm to solve that minimization problem is the *Least absolute shrinkage and selection operator (LASSO)*. For a equation $b = Ax$ it minimizes the l_2 norm of the residual $Ax - b$ plus the l_1 norm of x itself. While the l_2 tends to equalize all entries in x, the l_1 norm enforces sparsity. The weight of these two can tuned by a hyperparameter λ. This is especially useful as often times the actual cardinality of the sparse signal is not known beforehand. For compressed sensing LASSO is the following

$$\hat{s} = \arg\min_s \|C\Psi s - y\|_2 + \lambda \|s\|_1 \tag{3.17}$$

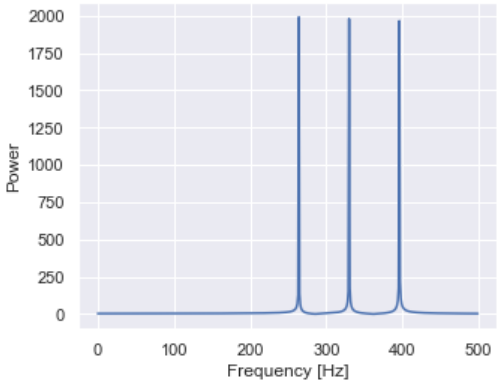
Figure 3.6 shows an example of compressed sensing algorithm in action. Signals up to 396 Hz were reconstructed from only 100 measurements, which seems to contradict the Nyquist-Shannon theorem telling us that only frequencies up to 50 Hz should be possible to reconstruct. However, compressed sensing exploits sparsity (only 3 frequencies present) and thus has stronger assumption about the signal. An even more important property is the randomness in the location of measure points. Because of this some measure points are very close together resulting in a locally high sampling rate. If the measure points were equal distributed one could only reconstruct frequencies up to 50 Hz as Nyquist-Shannon states.

The most important hyperparameter in compressed sensing is the choice of an appropriate base like fourier for music. It is not always clear which to choose and even if there's a good chance that there's still some performance gain by using a to the signal tailored basis. To produce such a tailored basis is the goal of *dictionary learning* which is explained in more detail in Mairal et al., 2009.

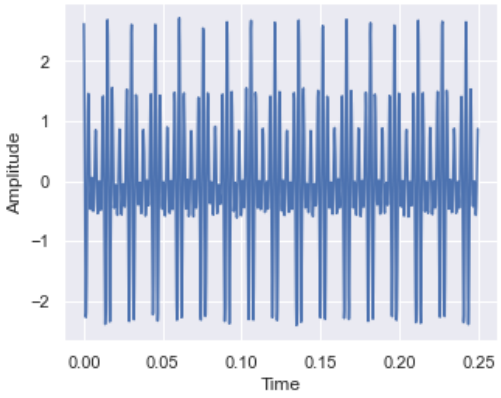
Given a training set of signals $X = \{x_1, \ldots, x_n\} \in \mathbb{R}^{m \times n}$ one wants to find a dictionary $D \in \mathbb{R}^{m \times k}$ whose columns are basis vectors (in this context also often referred as *atoms*) by minimizing a loss function



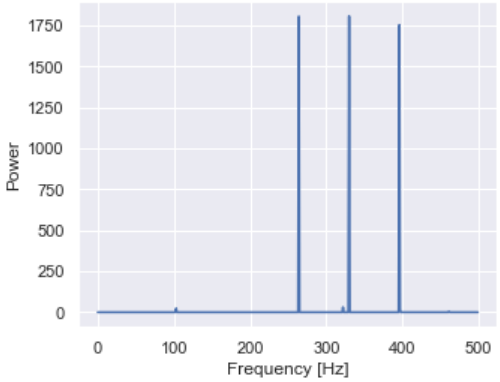
(a) Signal with measurements in red



(b) Power spectrum of signal



(c) Reconstructed signal



(d) Reconstructed power spectrum

Figure 3.6: Example of compressed sensing with a C major accord as signal. Signal was reconstructed from 100 measurements (marked as red dots in the signal) using the discrete cosine transformation as sparse basis and LASSO($\lambda = 0.08$) as minimizer.

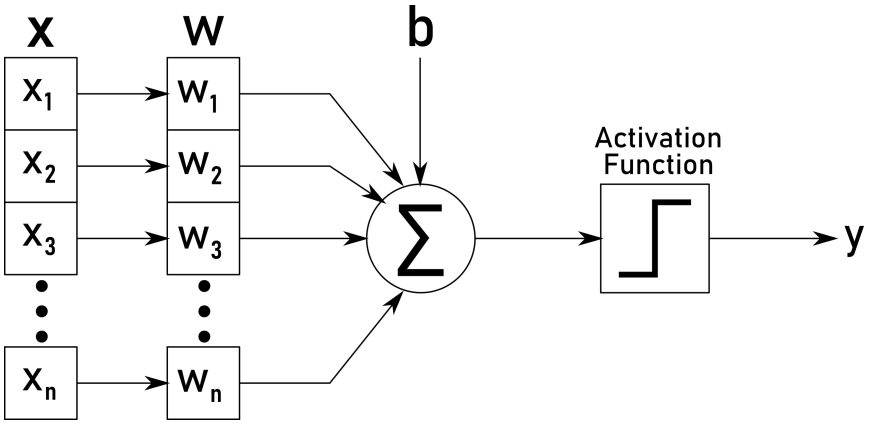


Figure 3.7: Illustration of a perceptron with input vector x, weight vector w, bias b and output y with a step activation function.

$$\min_{D \in C, \alpha \in \mathbb{R}^{k \times n}} \frac{1}{n} \sum_{i=1}^{n} \left(\frac{1}{2} \|x_i - D\alpha_i\|_2^2 + \lambda \|\alpha_i\|_1 \right) \tag{3.18}$$
$$C \triangleq \left\{ D \in \mathbb{R}^{m \times k} \text{ s.t. } \forall \text{ columns } d_j : \|d_j\|_2^2 \leq 1 \right\}$$

which is similar to the loss function in LASSO. The constraint on the columns and thus the basis vectors of the dictionary is to prevent the sparsity constraint being circumvented by decreasing α while increasing D.

Dictionary learning is obviously also an unsupervised learning algorithm.

3.6 Neural Networks

Neural Networks especially in the realm of deep learning are at the heart of most modern public known artificial intelligences. Most elaborate image filters now widely used are implemented using deep learning, e.g. is CartoonGAN a neural network which "cartoonizes" images [Chen, Lai, and Liu, 2018]. Unfortunately, the astonishing performance also lead to malicious usage widely known as "Deep Fakes" [Nguyen et al., 2019].

There are hardly any new books published about machine learning without at least mentioning neural networks. In fact, all books stated at the beginning of this chapter cover neural network. Goodfellow, Bengio, and Courville, 2016 even exclusively handles deep learning as the title suggests and is together with Du and S., 2019 the main source for this section, which is by no means a complete summary, but handles only the parts used in this thesis as the topic is to broad to handle fully here.

The motivation behind neural networks was to mimic the human brain by modeling its building blocks, i.e. the neurons. While there are many models of a neuron that can be used in machine learning, the one widely used today and even became synonymous for neurons is the perceptron developed by Rosenblatt in 1958 [Rosenblatt, 1958], which was also the first one that was not trained by hand but trained itself using data.

The perceptron as depicted in figure 3.7 is based on a linear model, i.e. it forms the weighted sum over its inputs x and adds a bias b, before putting the sum in

Name	Function
Sigmoid	$\frac{1}{1+\exp(-x)}$
ReLU	$\max(0, x)$
Tanh	$\tanh(x)$
Softplus	$\log(1+\exp(x))$

Table 3.2: Activation functions

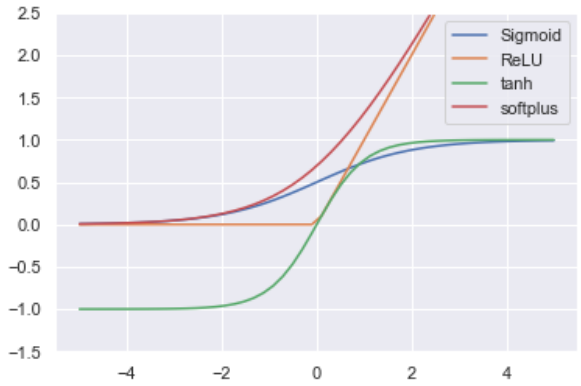


Figure 3.8: Plots of activation functions

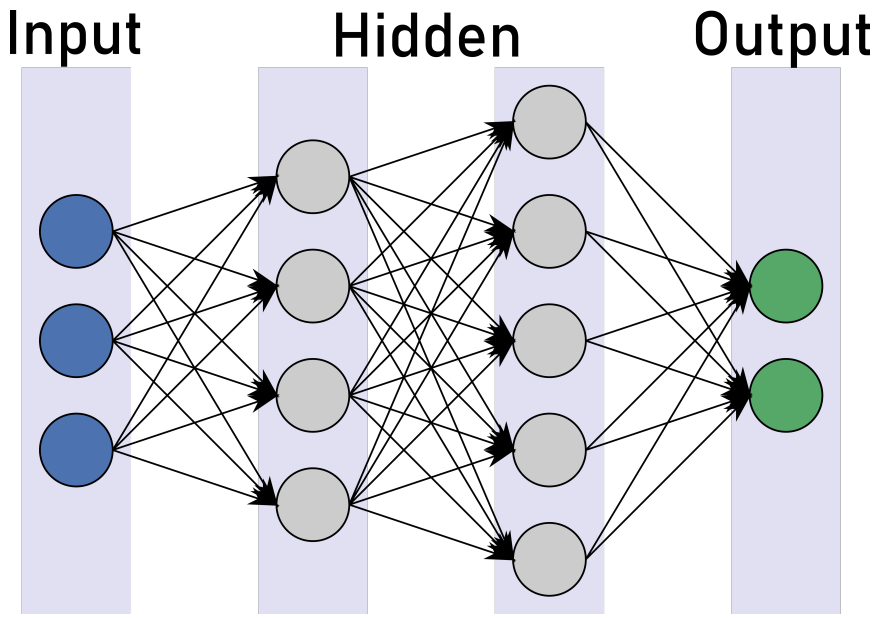


Figure 3.9: Fully Connected Network

the *activation function* $f()$. There are various activation functions in use and its only prerequisite is its nonlinearity. Table 3.2 and figure 3.8 show some popular ones. The original perceptron is a binary classifier using a simple step function as activation function denoting the two classes as -1 and 1:

$$y = f(\vec{w}^T \vec{x} + b) \tag{3.19}$$

$$\text{where} \quad f(x) = \begin{cases} 1 & \text{for } x > 0 \\ -1 & \text{for } x \leq 0 \end{cases}$$

Among the many limitations of such linear models is the most famously its inability to learn the XOR function which lead to neural networks becoming unpopular. This was later overcome by simply joining many perceptrons, thus creating a *multilayer perceptron (MLP)* organized in layers as depicted in 3.9. The first layer is the *input layer*, the last the *output layer*, while the layers in between are referred to as *hidden layers*. Networks where the outputs of the previous layer is only used in neurons of the next one are called *feedforward network (FFN)* and if all outputs of the previous

layer are used in every neuron of the next layer are called *Fully Connected Networks (FCN)*. Such networks are capable of much more complex models than binary classification, as stated by the *universal approximation theorem*:

Theorem 3.2 (Universal Approximation Theorem [Goodfellow et al., 2016])
[A] feedforward network with a linear output layer and at least one hidden with any "squashing" activation function [...] can approximate any Borel measurable function from one finite-dimensional space to another with any desired non-zero amount of error, provided that the network is given enough hidden units.

During the training process first a training sample X is put into the network to calculate a specified loss, which is known as the *forward pass*. The gradient is then calculated by consecutive applying of the chain rule starting by the loss function until the input is reached again. Because the flow is now in the opposite directions this process is known as *backpropagation*. This is sometimes falsely referred to as the algorithm to train the network. There are several training algorithm which usually base on gradient descent (the bias b is here part of w)

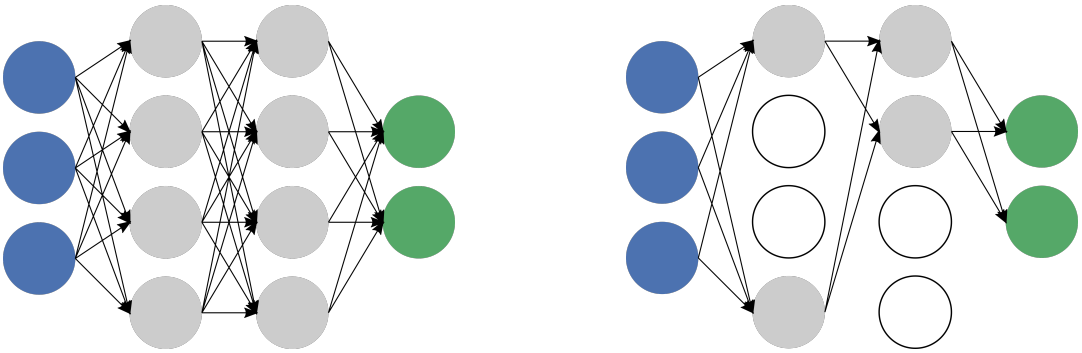
$$w \leftarrow w - \alpha \nabla L(y, \hat{f}(x)) \tag{3.20}$$

where α is the learning rate and a hyperparameter.

A popular example is *Stochastic Gradient Descent (SGD)*, which is a compromise between calculating the gradient for the whole dataset and for each data sample individually by forming random subsets of the data set called *mini-batch*. The first approach is often technically not possible and tends to get stuck in local minima, while the second one is too noisy. *Adam* builds upon this idea and includes first and second order momentum. There are many more algorithms, unfortunately there is again no one best algorithm but must be determined individually. Because of often times large amount of variables involved in a neural network second-order gradient methods are not technical possible, despite converging faster, i.e. in fewer iterations.

In order to be able to do the first forward pass, the weights must have been initialized. Choosing them has a great impact on not only the time needed to train the model, but even on its success. They must not be zero because then the gradient would also be zero, thus rendering the gradient descent useless. The also have to be randomly distributed in order to break symmetry as gradient descent cannot achieve this. If the model were symmetric, all neurons of a layer would be the same and thus cannot adopt to complex function. Glorot and Bengio, 2010 gives a great insight on this topic. In a nutshell one wants the weights such that the gradient won't collapse or explode. A common heuristic to achieve this knwon as *Xavier Initialization* is to set the bias to zero and uniformly distribute the weights in the interval $\left[-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}\right]$ where n is the number of neurons of the previous layer.

Especially in neural network is it usual to use *regularization*, i.e. adding a loss function based on the model itself and not the data. A common one is *ridge regression* also known as *weight decay*. It adds the l_2 norm of the weights to the cost function and thus enforces the network to use all neurons and not rely on only a few specialist neurons (setting the weights of the other neurons to zero). Its second name got it because through gradient descent the weights are effectively exponentially decaying. *Dropout* (figure 3.10) takes this idea a step further by making it a part of the network itself. Applied as a intermediate layer after a hidden layer it discards randomly selected values only during training. This also prevents specialist neurons as they can



(a) Neural Network without Dropout **(b)** Neural Network with Dropout

Figure 3.10: Illustration of dropout. Neurons depicted by empty circles are ignored. It is somewhat disputed wether dropout should include the input neurons.

by chance be silenced. During predictions it acts like an ensemble algorithm as there have been multiple (entangled) networks with the same function trained.

A more complex problem requires unsurprisingly a more complex network. One often refers in this context to the *capacity* of the model. If the capacity is too low, it will heavily underfit, but with increasing capacity the model becomes harder and harder to train. It needs more data to prevent it from overfitting as it will happily just remember the data set if it fits in its capacity. Even with the right capacity the model will eventually start to increase its generalization error again and overfit. A common approach is to just stop training once the error raises again, known as *early stopping*.

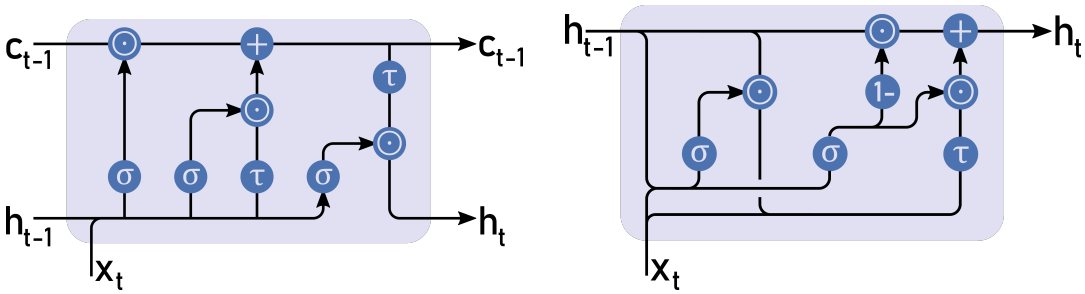
Increasing the depth, i.e. adding more layers, increases the capacity better than adding more neurons to the layers. Unfortunately, such *deep neural networks* are hard to train, as the chance of the gradient vanishing or exploding increase with each layer. Glorot and Bengio, 2010 is again a great source to understand this problem. Ioffe and Szegedy, 2015 accomplished a breakthrough in this matter by introducing *batch normalization*. In a nutshell is it an additional layer like dropout whose job is to restore a gaussian distribution with learnable mean and variance. This sanitizes the gradient and allows much deeper networks. The reasoning behind that is the same as for the weight initialization but won't be discussed here.

There exist a form of neural networks known as *Recurrent Neural Network (RNN)* that unlike all previously mentioned machine learning algorithm is able to directly process sequential data of variable length. By carrying over a *hidden state* $\vec{h}^{(t)}$ the same neural network denoted by its parameters θ can be reapplied for each item $\vec{x}^{(t)}$ in the sequence:

$$\vec{h}_t = f(\vec{h}_{t-1}, \vec{x}_t; \theta) \tag{3.21}$$

The last hidden state can then be further used. For example can a RNN process the weather of the last 10 days and pass the hidden state to a conventional FCN to predict the weather of the next day. The hidden state can also be passed to another RNN to produce another sequence. The trick here is that the second RNN can emit a special token to mark the end of the sequence. This approach is used in natural language translation.

A hurdle in training RNNs is to prevent it from forgetting information from the



(a) LSTM with two hidden states h_t and c_t **(b)** GRU with one hidden state h

Figure 3.11: Illustration of a LSTM and a GRU cell for RNNs. σ is the sigmoid and τ is the tanh activation function. $\odot$ is the element-wise multiplication. Based on Phi, 2018.

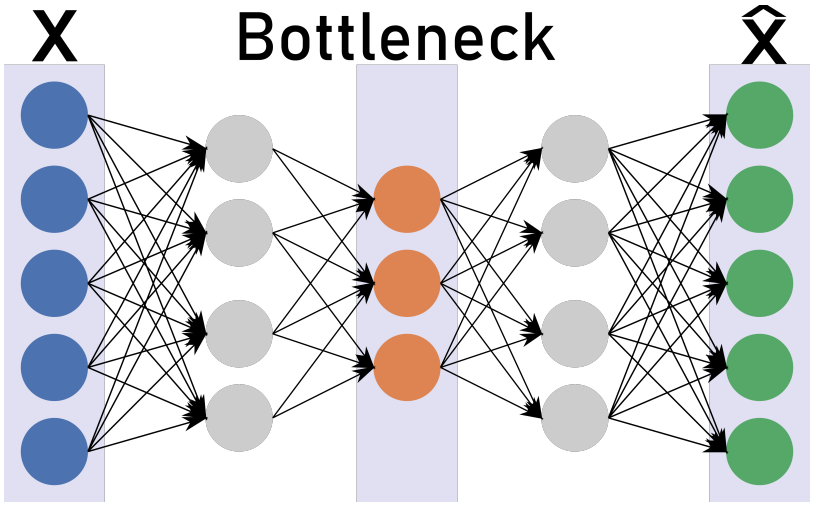


Figure 3.12: Illustration of an autoencoder. It tries to recreate the original but has to compress the input in order to be able to pass it through the bottleneck.

beginning of the sequence. This is usually achieved by having a direct path for the gradient to flow back to the beginning. Figure 3.11 shows two of prominent representatives models used: *Long Short-Term Memory (LSTM)* and *Gated Recurrent Units (GRU)*.

Neural networks can also be used in an unsupervised fashion for *feature extraction*, i.e. compressing the input and thus reducing its dimensionality. Such neural networks are called *Autoencoder* and are illustrated in figure 3.12. They work by forcing the input through a bottleneck and try to reconstruct the original input, thus the loss is the difference between reconstructed and original input. The bottleneck can than be used to train another model.

CHAPTER 4

Preprocessing and Feature Engineering

Feature Engineering describes the process of modeling new features (*feature extraction*) from or selecting a subset (*feature selection*) of the raw input data [Du and S., 2019]. While time consuming and rarely automatable, it's still worth to spend most of the time to refine features, since the performance of machine learning models are highly sensitive to their quality [Heaton, 2017].

The dataset was created using the three blazar catalogues 4LAC [The Fermi-LAT collaboration, 2019], 3HSP [Chang et al., 2019] and 5BZCat [Massaro et al., 2015] to get a list of known blazar[1]. The actual SEDs that are used as input were created by the *VOUBlazar* tool [Chang, Brandt, and Giommi, 2019], which internally uses many more catalogues to create an as complete SED as possible.

This chapter shows the preprocessing of the raw input data as generated from the *VOUBlazar* tool and presents several engineered features, which will be used later to train models. Doing so specifically tackles the problem of having a variable length in the raw input while the models require a fixed size, and the problem of bias in the data.

4.1 Data Preparation

The *VOUBlazar* tool creates a human readably representation of a SED by merging several astrophysical catalogues [Chang, Brandt, and Giommi, 2019]. Figure 4.1 shows an example output. It consists of several measurements each including the frequency, energy flux, the time of the measurement and some metadata about the source the tool retrieved the data.

The metadata holds no useful information for determining ν_{peak} and thus will be ignored. The measurement time is for most measurements a fixed default value, i.e. it's missing and thus not useful and will also be ignored. Therefore the first step in the preprocessing is to parse the text file to retrieve a machine readable version of frequency, flux and its errors for each measurement.

Some measurements have zero frequency or zero flux, which makes physically speaking no sense and are thus discarded. Furthermore, measurements with flux outside their error range are also discarded. The error was deemed to provide little information to the task as it is often times dominated by the sources variability and won't

[1] A special thanks to Paolo Giommi for refining the ν_{peak} on several hundred blazars

1 matched source 187.27790 2.05240 99 0.000

Frequency Hz	nufnu erg/cm2/s	nufnu unc. upper	nufnu unc. lower	start time MJD	end time MJD	Flag	Catalog	Reference
1.400E+09	5.178E-13	5.179E-13	5.177E-13	55000.0000	55000.0000	Det	FIRST	White et al. 1997, ApJ, 475 479
2.418E+17	6.085E-11	6.103E-11	6.068E-11	55000.0000	55000.0000	Det	4XMM-DR10	Webb et al. A&A, 641, A136
7.253E+16	7.759E-11	7.786E-11	7.733E-11	55000.0000	55000.0000	Det	4XMM-DR10	Webb et al. A&A, 641, A136
1.813E+17	5.460E-11	5.477E-11	5.443E-11	55000.0000	55000.0000	Det	4XMM-DR10	Webb et al. A&A, 641, A136
3.627E+17	5.153E-11	5.169E-11	5.136E-11	55000.0000	55000.0000	Det	4XMM-DR10	Webb et al. A&A, 641, A136
7.253E+17	6.316E-11	6.344E-11	6.288E-11	55000.0000	55000.0000	Det	4XMM-DR10	Webb et al. A&A, 641, A136
1.692E+18	8.087E-11	8.143E-11	8.031E-11	55000.0000	55000.0000	Det	4XMM-DR10	Webb et al. A&A, 641, A136
2.418E+17	3.697E-11	3.703E-11	3.692E-11	55000.0000	55000.0000	Det	4XMM-DR10	Webb et al. A&A, 641, A136
7.253E+16	4.762E-11	4.772E-11	4.752E-11	55000.0000	55000.0000	Det	4XMM-DR10	Webb et al. A&A, 641, A136
1.813E+17	3.513E-11	3.519E-11	3.507E-11	55000.0000	55000.0000	Det	4XMM-DR10	Webb et al. A&A, 641, A136
3.627E+17	3.274E-11	3.279E-11	3.269E-11	55000.0000	55000.0000	Det	4XMM-DR10	Webb et al. A&A, 641, A136
7.253E+17	3.798E-11	3.807E-11	3.790E-11	55000.0000	55000.0000	Det	4XMM-DR10	Webb et al. A&A, 641, A136
1.692E+18	4.736E-11	4.753E-11	4.718E-11	55000.0000	55000.0000	Det	4XMM-DR10	Webb et al. A&A, 641, A136
2.418E+17	3.924E-11	3.943E-11	3.905E-11	55000.0000	55000.0000	Det	4XMM-DR10	Webb et al. A&A, 641, A136
7.253E+16	3.990E-11	4.011E-11	3.969E-11	55000.0000	55000.0000	Det	4XMM-DR10	Webb et al. A&A, 641, A136
1.813E+17	2.725E-11	2.738E-11	2.713E-11	55000.0000	55000.0000	Det	4XMM-DR10	Webb et al. A&A, 641, A136
3.627E+17	2.758E-11	2.770E-11	2.746E-11	55000.0000	55000.0000	Det	4XMM-DR10	Webb et al. A&A, 641, A136
7.253E+17	3.534E-11	3.554E-11	3.513E-11	55000.0000	55000.0000	Det	4XMM-DR10	Webb et al. A&A, 641, A136
1.692E+18	6.727E-11	6.792E-11	6.662E-11	55000.0000	55000.0000	Det	4XMM-DR10	Webb et al. A&A, 641, A136
2.418E+17	4.441E-11	4.458E-11	4.424E-11	55000.0000	55000.0000	Det	4XMM-DR10	Webb et al. A&A, 641, A136
7.253E+16	5.903E-11	5.928E-11	5.878E-11	55000.0000	55000.0000	Det	4XMM-DR10	Webb et al. A&A, 641, A136

Figure 4.1: Output of the *VOUBlazar* tool for an example blazar. Only the first few lines are shown. In total it has over 10,000 lines which makes it one of the bigger ones.

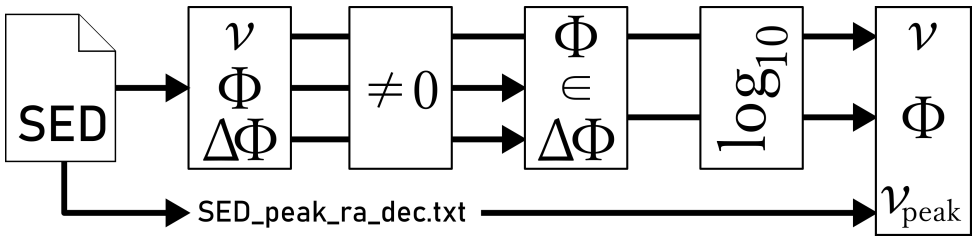
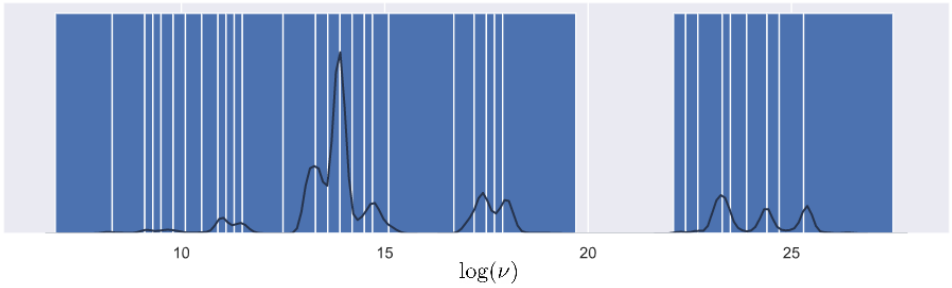


Figure 4.2: Schematic of the preprocessing step to retrieve machine readable data. ν denotes the frequency and Φ the flux density.

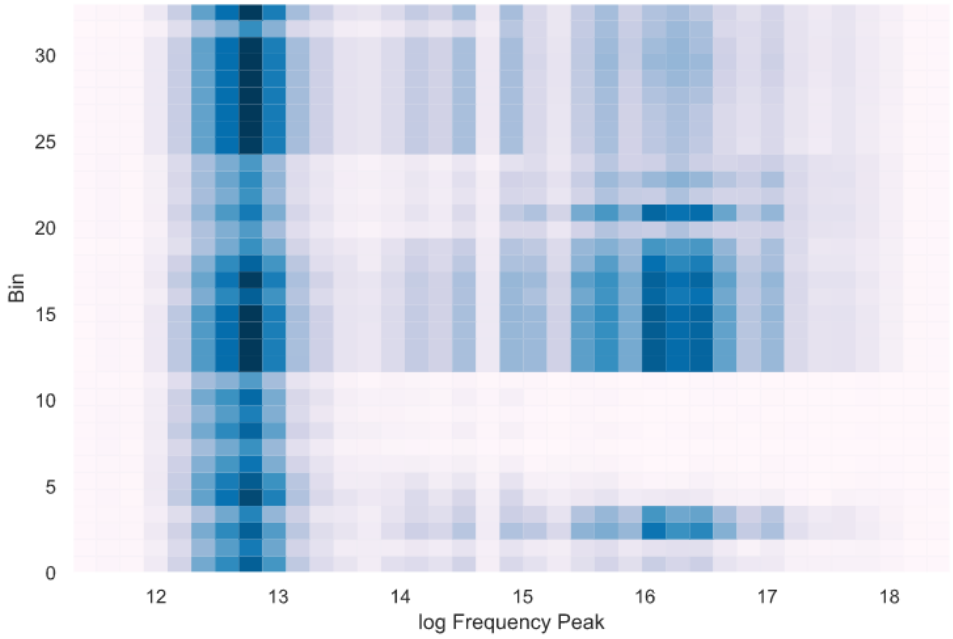
be used further. At this step of the preprocessing only flux and frequency with sane values remain. It was found, that the overall performance of the models significantly increases if the logarithm of flux and frequency were used. Here the decadic logarithm was chosen. Lastly, the ground truth, i.e. the ν_{peak} for the blazar was determined by hand and is stored also in decadic logarithm in the text files name. One such a filename is *SED_13.24_133.7036_20.1085.txt* and consists of the blazar's ν_{peak} and its coordinate. Since each file only consists of measurements of one blazar, this process is repeated for each blazar in the dataset. Figure 4.2 visualizes the process so far.

The dataset contains 3,793 blazars with annotated ν_{peak}. Before any further feature engineering a test set containing 700 of these blazars is put aside for later evaluation.

Figure 4.3 shows a visualization of the training set. It's rather obvious to see, that some bins are only filled for blazars with ν_{peak} in certain ranges and are thus strongly biased. If not for the histogram, this might have gone unnoticed during the training. To show this, a simple random forest is trained on this biased data set (see 4.4a). While the predictions seem rather good, it actually exploited heavily the bias in the dataset. The corresponding feature importance score (see 4.4c) conclude, that the random forest is especially interested in frequencies in the logarithmic range of 10.9 - 11.1. As figure 4.4e shows, wether that specific bin is empty or not already separates the data set very well. Unfortunately, this also means that this random forest is useless for any new unseen data, which does not follow that bias.

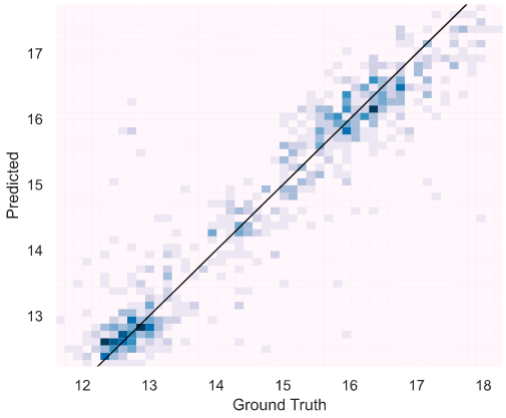


(a) Binning edges width frequency density

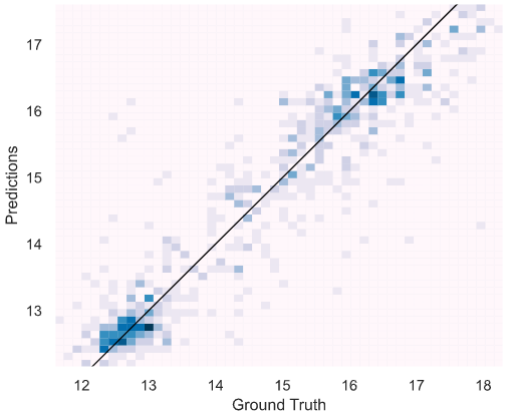


(b) Histogram of Bins Filled against ν_{peak}

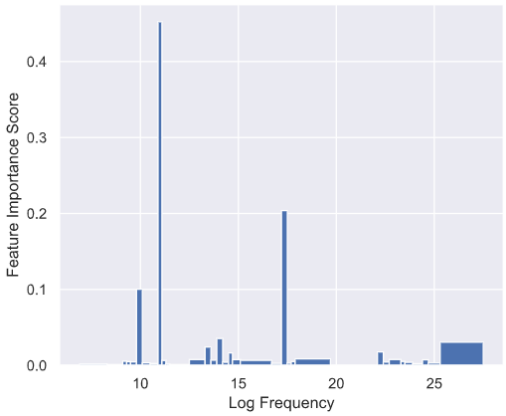
Figure 4.3: Visualization of the dataset used to train the models. Since some frequency ranges are more dense with measurements than other the bins are not uniform, but are adjusted to get a more uniform filling. a) shows the bin edges including the frequency density of the measurements. Note that the goal was to get the histogram uniform, thus some bins seem to get more data than others. b) shows the histogram for these bins against the ν_{peak}. One can clearly see clustering of the bins with some only present in specific ranges, thus being biased.



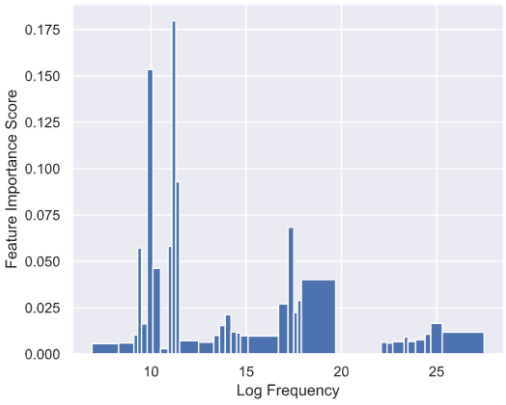
(a) Biased predictions



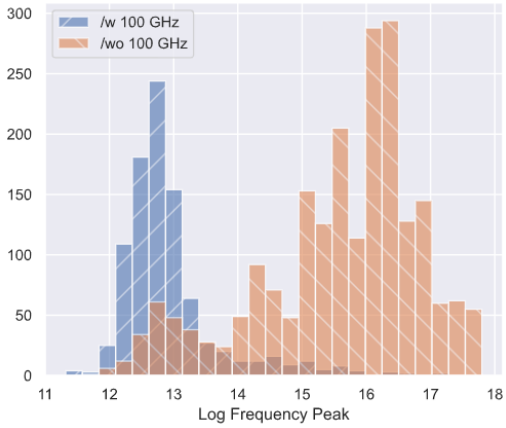
(b) Bias reduced predictions



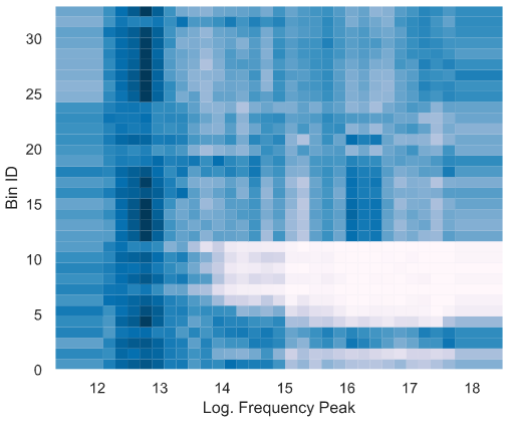
(c) Feature importance biased random forest



(d) Feature importance bias reduced random forest



(e) Distribution based on existence of samples around 100 GHz



(f) Histogram of bias reduced data set

Figure 4.4: Before (left) and after(right) of data augmentation to reduce bias, visualized by two separately trained random forests. While both seem to perform well, the biased one relies strongly on the log frequency range 10.9 - 11.1. E) shows that even solely on wether this bin is empty or not, the peak is already well separated. The unbiased random forest considers more frequencies. Lastly, F) shows a histogram of the enhanced dataset.

In order to reduce the bias in the dataset it is augmented by oversampling especially samples with underrepresented peak frequencies while undersampling especially overrepresented frequencies. To do so, first the SED are binned (into label bins) according to their ν_{peak}. Afterwards, for every copies of randomly selected members are made until a specified amount in every label bin is reached. However, 19 from the 34 frequency bins (as in figure 4.3a) are randomly selected with replacement to be deleted in the copy. The probability of a frequency bin to be chosen is proportional to the amount of total (i.e. not regarding the label bins) non empty frequency by the power of four, to better smooth the distribution. Copies with less than 10 sample points are discarded as ν_{peak} is assumed to be no longer determinable. With this data augmentation the dataset grow to a total of 10,233 SEDs, while on average 12 distinct not necessarily empty bins were deleted. Figure 4.4f shows the histogram of the enhanced data set. While the bias was drastically reduced, it's still present and must still be considered during training.

4.2 Binning

In the previous section binning was already used to demonstrate the bias in the dataset. However no justification for the choice of the binning edges were made, which is made up for here. Starting with the set of all frequencies present in the dataset, they are rounded to a specific width, e.g. to the next 0.1 for a width of 0.1. Gaps in between are merged with the surrounding bins if they are small enough. Bins with hardly any data inside are also merged. Finally, biased bins are removed.

Figure 4.6 shows the histogram for four different widths. Their exact edges can be found in the appendix. Wider bins seem to suffer from bias, while narrower bins ignore more data. Figure 4.5 shows the error distribution for random forests trained for each bin configuration. It seems all four configuration achieve about the same performance. Therefore, the second configuration was chosen since it provides the best balance between bias and frequency coverage.

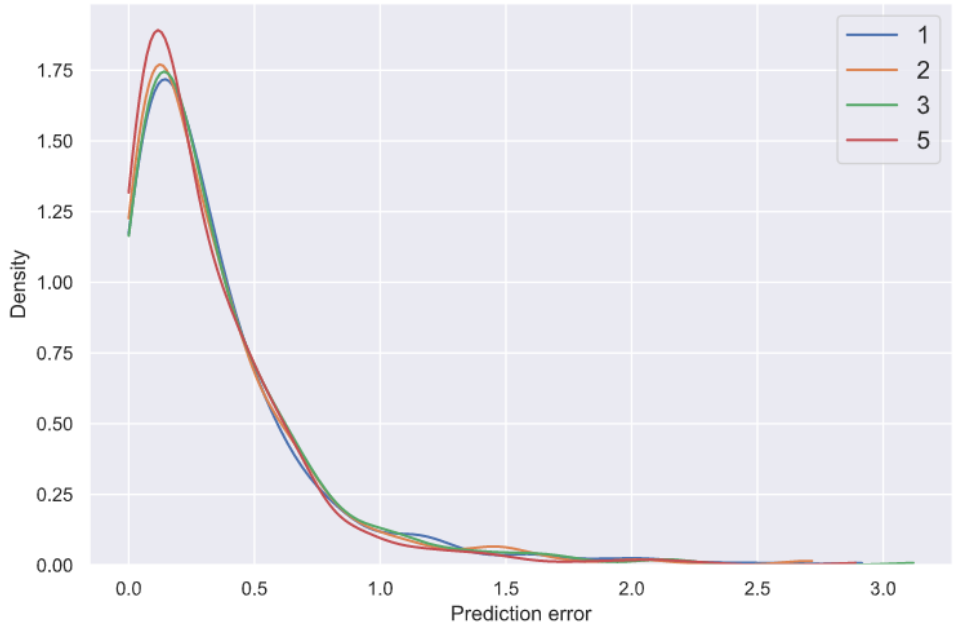
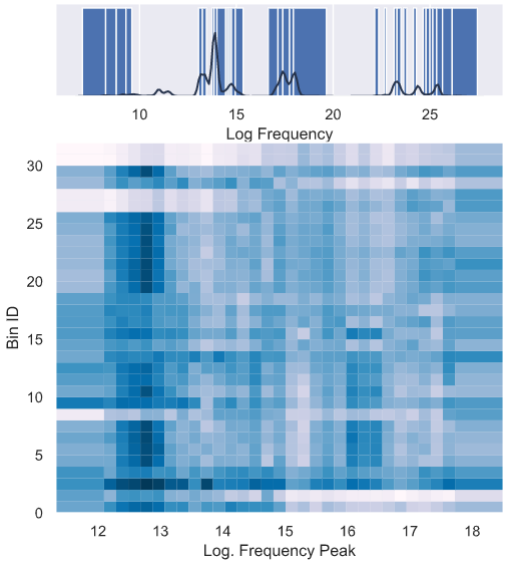
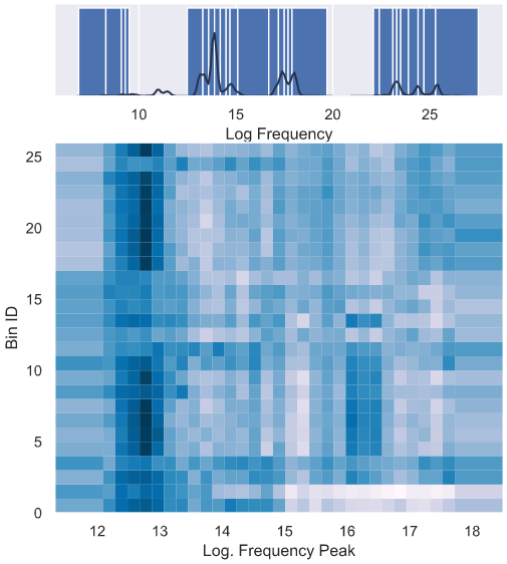


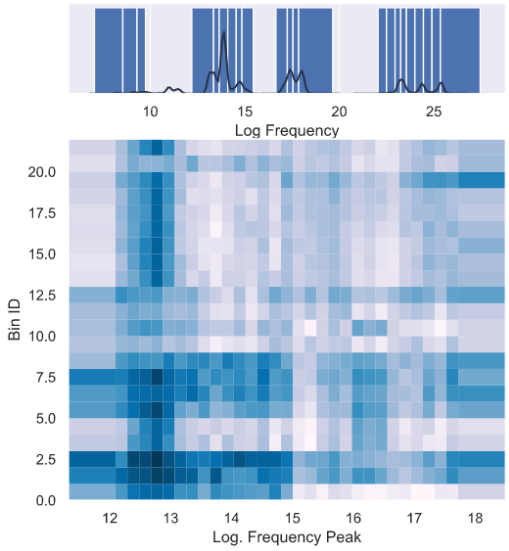
Figure 4.5: Performance of different binnings



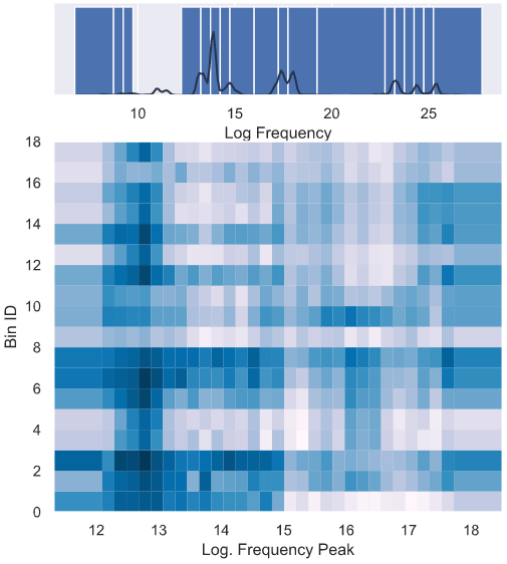
(a) 0.1 width



(b) 0.2 width



(c) 0.3 width



(d) 0.5 width

Figure 4.6: Histograms for various binning configurations. Biased bins have been removed. Additionally, an kernel density estimate of the distribution of ν_{peak} is plotted on top of the bin edges.

4.3 Compressed Sensing

A major drawback of the binning was that biased bins had to be ignored. Instead of ignoring them it would be better to fill in missing data and thus remove the bias. One way to achieve this is to use *compressed sensing* as described in section 3.5. Here the implementation of *scikit-learn* is used [Pedregosa et al., 2011]. An important assumption for this to work is that there exist a sparse representation of a SED. While basis like the Fourier-basis might be a reasonable choice, it makes more sense to create a basis tailored to the problem to enhance the algorithms performance. Thus, as a first step one wants to create such a basis, i.e *Dictionay learning*.

Dictionary learning requires a set set of complete signals, which in the case for blazar or AGN in general does not exist, since for some frequency ranges no telescopes exist. It this therefore necessary to fall back on synthetic data. Here a set of 500,000 synthetic SEDs were created using *naima* as described in section 2.2. The exponential cutoff power law was chosen as particle distribution as it seems to provide a good balance between complexity and completeness. As radiation models were synchrotron with variable magnetic strength and inverse compton with three different thermal photon sources and varying temperature, both sharing the same maximum electron energy. The ranges on which these parameters were sampled from are listed in table 4.1. While the param ranges seem to be ridiculously wide, it was chosen on purpose since it would harm the performance to not simulate exotic SEDs while a few "garbage" ones would not. The later one will at best result in some basis vectors that later simply won't be chosen by the algorithm. Since the goal is to find a sparse representation this does not matter at all. Nevertheless, a more reasonable selected parameter space might improve the performance.

The SEDs are binned into ranges from 6.9 to 27.5 into widths of 0.1 of the logarithmic frequency space matching the range of the real dataset, thus 206 bins in total. Following the default parameter of *scikit-learn* 206 basis vectors were chosen, thus creating a square dictionary. Figure 4.7 shows a selection of basis vectors of the dictionary.

Figure 4.8 shows two reconstructed SEDs from the augmented training set using compressed sensing with the Lasso algorithm ($\lambda = 0.001$ with a tolerance of 10^{-4}) and the previously described dictionary. It fills in missing measurements with reasonable data as long as there is enough real data nearby. If this is not the case like in the second example the algorithm starts to hallucinate and completely makes up new data. However, since the main interest lies inside the first bump weird data at other regions should hopefully not harm the performance of the predicting models later on. This assumption has to be tested though.

Param	Range
A	1×10^{29} to $1 \times 10^{34}\,\mathrm{eV}^{-1}$
E_0	1×10^{12} to $1 \times 10^{15}\,\mathrm{eV}$
α	-2.0 to 3.0
E_{cut}	1×10^{8} to $1 \times 10^{13}\,\mathrm{eV}$
β	-2.0 to 3.0
B	1×10^{-9} to $1 \times 10^{-7}\,\mathrm{T}$

Param	Range
E_{max}	1×10^{10} to $1 \times 10^{20}\,\mathrm{eV}$
T_{NIR}	1500 to 6000 K
T_{FIR}	5.0 to 800 K
ρ_{NIR}	0.5 to 4.0 $\mathrm{eV\,cm^{-3}}$
ρ_{FIR}	0.1 to 2.1 $\mathrm{eV\,cm^{-3}}$

Table 4.1: Parameters of the synthetic dataset used for dictionary learning.

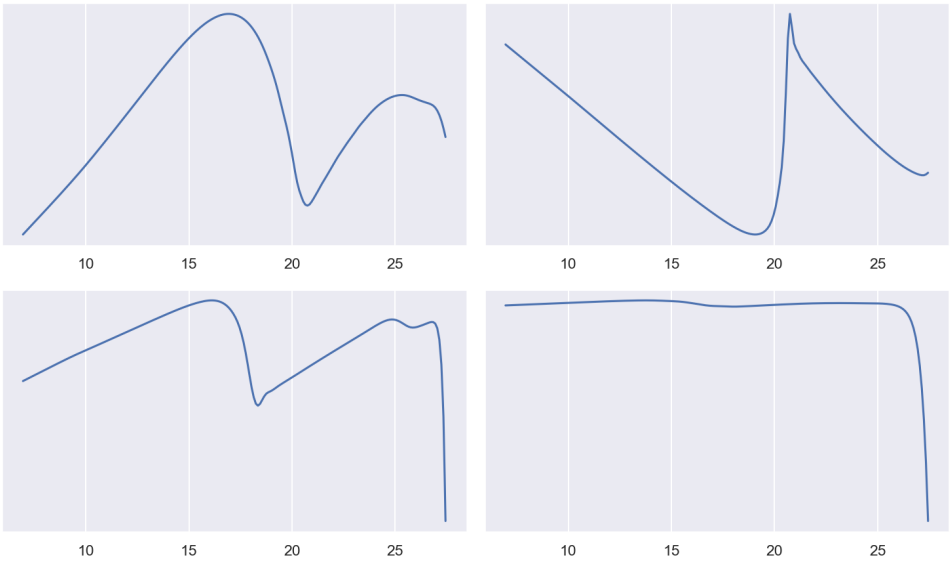


Figure 4.7: Selection of basis vectors produced by dictionary learning for SEDs. Due to the nature of the algorithm the exact flux densities have no physical meaning and are thus omitted. Most of the basis vectors resemble SED very well, although maybe upside-down. Only a few have no resemblance at all like the last one.

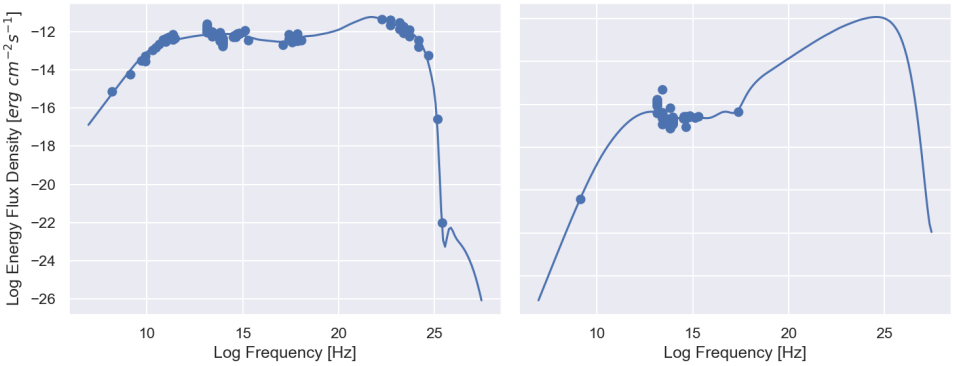


Figure 4.8: Example of a reconstructed SED using compressed sensing and the tailored dictionary. The Lasso algorithm was used with $\lambda = 0.001$ and a tolerance of 10^{-4}.

4.4 Autoencoder

Smaller bin widths seem beneficial as it allow for more detail. However, since most bins would be empty this way broader ones were chosen previously. One way to circumvent the problem is to compress the binned data such that the data becomes dense. Here it is achieved by utilizing an autoencoder which tries to encode the whole SED and decodes or reconstruct it given the encoded data and the frequencies at which the flux was measured. By reapplying the frequencies the encoder does not need to remember where the measurements happened and thus the encoded data should not include wether a specific frequency was measured ultimately reducing the bias. To speed up the training process, the data was binned in the same manner as for the compressed sensing, i.e. all bins have 0.1 width.

Parameter	Value
Learning rate	6.756×10^{-3}
Weight decay	1.014×10^{-5}
Num layers	2
Hidden size	16
Encoder dropout	37.10%
Decoder dropout	1.19%

Table 4.2: Hyperparameters for autoencoder. Note that a layer consists of two GRUs (forward and backward) and the hidden size is per GRU, thus the total hidden state is actually 128.

The architecture of the encoder is illustrated in figure 4.9. Both encoder and decoder consists of two layers of bidirectional GRUs including dropout, while the decoder additionally features a final linear layer to output only the flux for each frequency. The used hyperparameters are listed in table 4.2 and were deduced by a randomized search.

Figure 4.10 shows two example of the autoencoder reconstructing an SED. While it is in most cases very accurate it seem to struggle with regions showing high variability. Since this also occurs in regions where the ν_{peak} is located it might hurt the performance.

For models predicting ν_{peak} the final hidden state of the encoder is used as input and consists of 64 values in total. Considering that the original input featured 412 values (frequency and flux for each of the 206 bins) a compression by a factor of 6.4 was achieved.

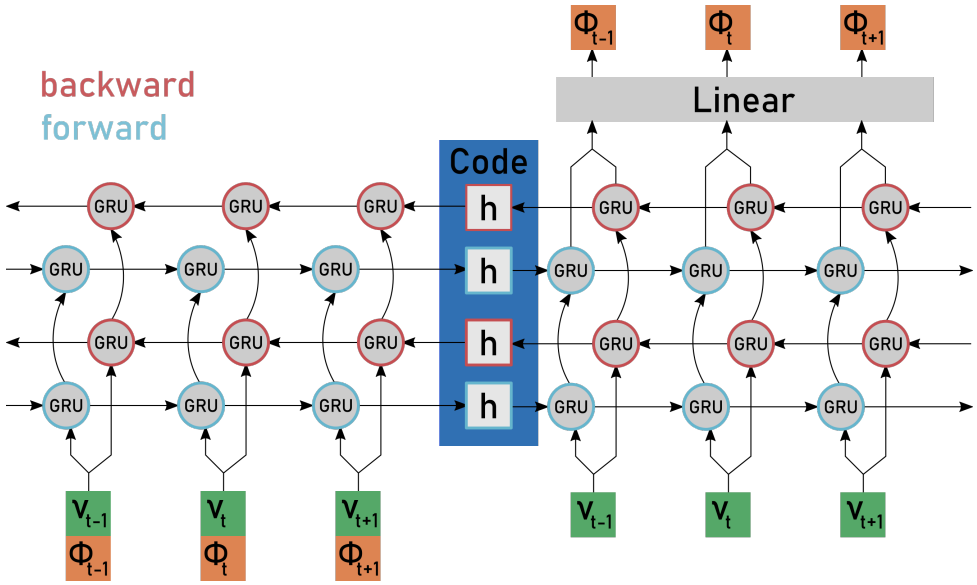


Figure 4.9: Architecture of the autoencoder. It consists of two layer bidirectional GRUs for both encoder and decoder, i.e. 8 independent GRUs in total. The encoder is feeded with both the flux Φ and the frequency ν. The accumulated last hidden states of each layer form the compressed data and is used as initial hidden state for the decoder, which now only gets the frequencies. The output of the GRUs of the decoder are put through a linear layer to obtain the flux for each frequency.

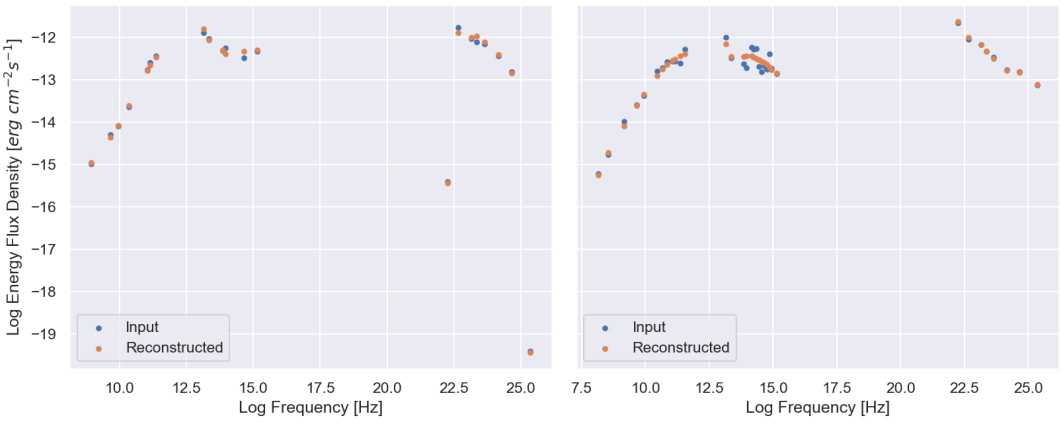


Figure 4.10: Examples of the autoencoder reconstructing SEDs. Most are well reconstructed like the first example. However, in some cases like the second one the autoencoder starts to smooth the flux.

CHAPTER 5

Predictive Models

Using the features engineered in the previous chapter several machine learning algorithms as described previously will be used to train one model per feature set and algorithm. Their task is to predict the ν_{peak} of an SED with an 95% prediction interval. This chapter is organized by the algorithm used and ends with an overview and comparison of the models' performances, which are measured using the test set created at the beginning of the data preprocessing. The test set as mentioned before has not been augmented like the training data set and thus represents unseen data as one would get from the *VOUBlazar* tool for new blazars.

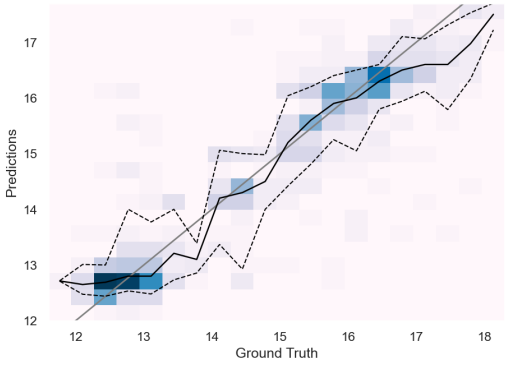
The algorithms used to train the predictive models were chosen based on a survey performed by *Kaggle*. Ignoring linear regression, the three most used algorithms were *Random Forest*, *Gradient Boosting* and *Neural Networks* [Kaggle, 2020], which all have been discussed previously. For the actual implementation of these algorithms *scikit-learn* [Pedregosa et al., 2011] were chosen for the first two, while for neural networks *PyTorch* [Paszke et al., 2019] was chosen, with the addition of *PyTorch Lightning* [Falcon, 2019] and *Tune* [Liaw et al., 2018] to enhance model training. Additionally *TensorBoard*, which is part of *TensorFlow*[Abadi et al., 2015] was used to monitor the training of neural networks.

5.1 Random Forest

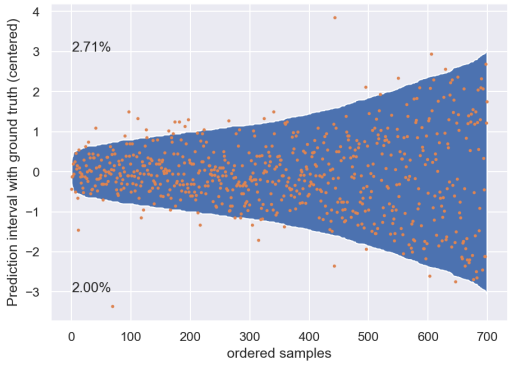
Parameter	Binning	Compressed Sensing	Autoencoder
#Trees	500	1,000	1,000
λ		8×10^{-4}	
Tolerance		8×10^{-4}	

Table 5.1: Hyperparameters used for random forest training after tuning.

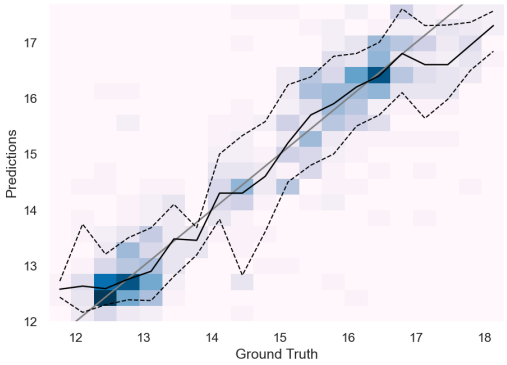
The implementation of random forest in *scikit-learn* does not provide the distribution of the training set on the leafs, which is required in equation 3.12 to calculate the quantiles. This problem can be circumvented by fully growing the trees in the forest, thus each training sample has the weight one. Because of this, the only remaining tunable hyper parameter besides those intrinsic to the features is the number of trees in the forest. Table 5.1 lists the used hyperparameters, which have been deduced after a randomized grid search. The number of trees has been chosen as the maximal amount after which no further improvement has been possible.



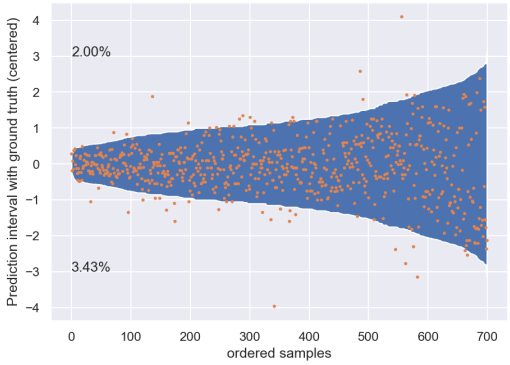
(a) Predictions for binned features



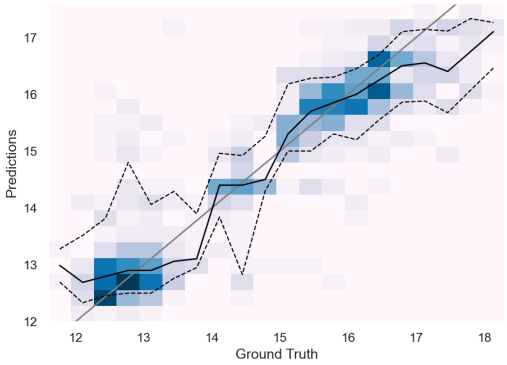
(b) Prediction Intervals for binned features



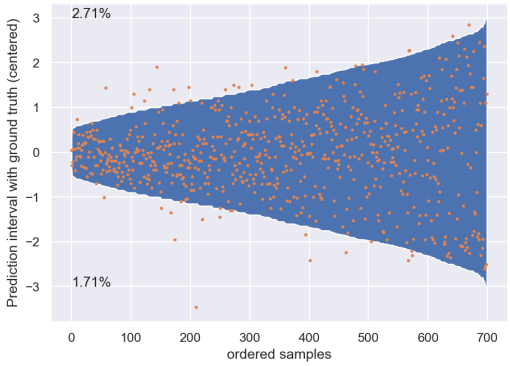
(c) Predictions for compressed sensing



(d) Prediction Intervals for compressed sensing

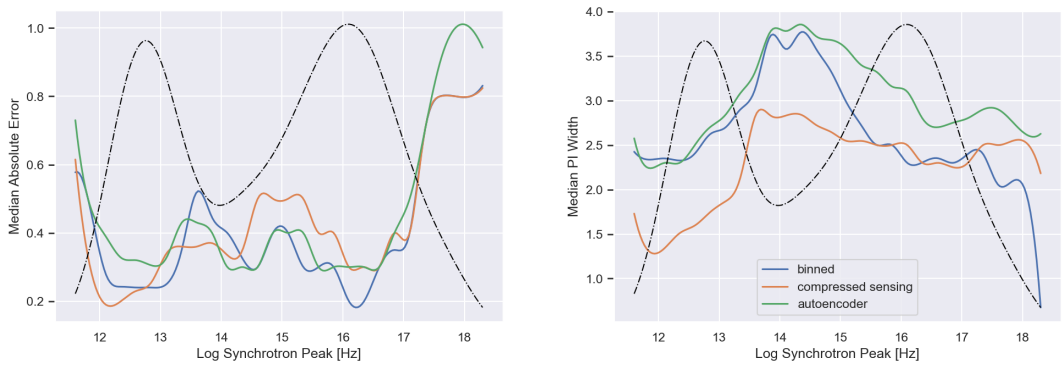


(e) Prediction for autoencoder



(f) Prediction Intervals for autoencoder

Figure 5.1: Illustration of the performance of random forest for each feature set. The left column shows a histogram of the predicted ν_{peak} against the ground truth with the median as a solid and the 80% quantile interval as dashed black lines. The right column shows the centered prediction interval (95%) sorted by their width with the ground truth marked as orange dots. Additionally the percentage of outliners above and below their respective prediction interval is shown.



(a) Median absolute error

(b) Median PI width

Figure 5.2: Median absolute error of the prediction and the median prediction interval width for random forest on the test set for each feature set. The curve was created by a moving window and smoothed with a cubic spline. The dashed line shows the density of the ground truth in the test set.

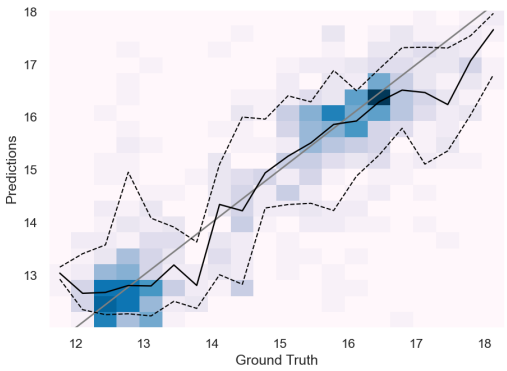
Figure 5.1 illustrates the performance and figure 5.2 the errors for the random forests trained on each feature set. All models are capable of predicting ν_{peak} with an acceptable performance, but show problems with edge cases, i.e. those with very small or very huge ν_{peak}. While some prediction have low prediction intervals, more than half were bigger than ± 1. Notable is that the compressed sensing indeed improved the performance as one can see by the median more strictly following the ground truth in the histogram and the prediction interval being narrower, although with a slightly worse coverage. The autoencoder on the other hand shows the worst performance in both prediction and prediction intervals. The errors are well behaved and show no unexpected behavior.

5.2 Gradient Boosting

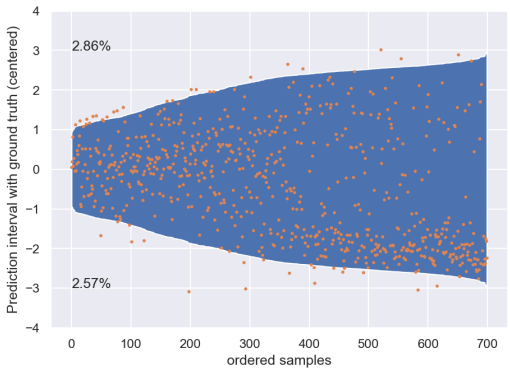
Gradient boosting is a special case since it is not a single model per feature set but three, one for each quantile: 2.5%, 50% and 97.5%. Thus, there are also three sets of hyperparameters as listed in table 5.2. They have been deduced after a randomized grid search.

The performance of this algorithm is like before illustrated in figure 5.3. The predictions on the binned feature set is only good on a small range of ν_{peak} and thus hardly usable. To make things worse, the prediction intervals are also way to wide to be of any use. The usage of compressed sensing does improve the performance, making the prediction even useful, the prediction intervals however are still to wide and thus not useable. The autoencoder outperforms the binned feature set but could not even reach the performance of the compressed sensing. Not only is the prediction interval still too broad to be usable, it also shows a strong tendency of overestimating ν_{peak}.

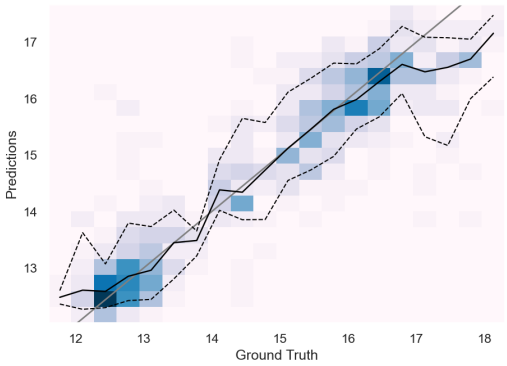
The errors as illustrated in figure 5.4 show interesting behavior as only the binned feature set struggles with predicting peaks around 100 THz. Furthermore, while having similar predicting performance the compressed sensing and autoencoder produce better prediction intervals on different frequency ranges, thus combining them would improve the performance. Unfortunately, that would still be not good enough to be usable. Apparently the gradient boosting algorithm boosted the noise



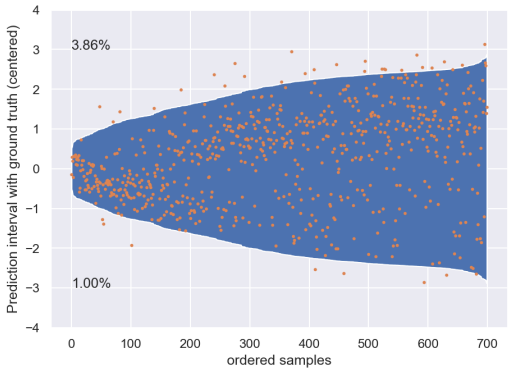
(a) Predictions for binned features



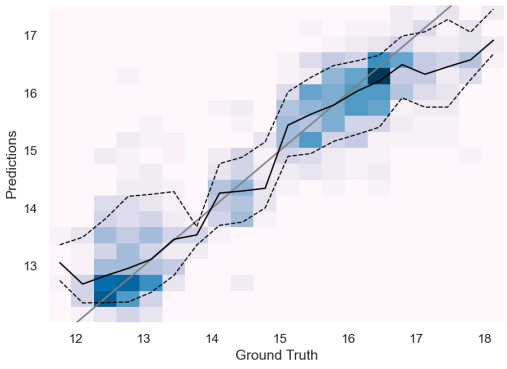
(b) Prediction Intervals for binned features



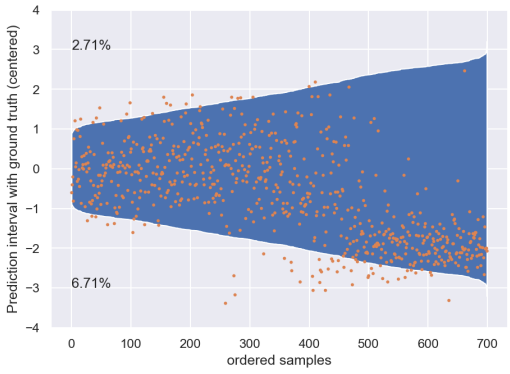
(c) Predictions for compressed sensing



(d) Prediction Intervals for compressed sensing



(e) Prediction for autoencoder

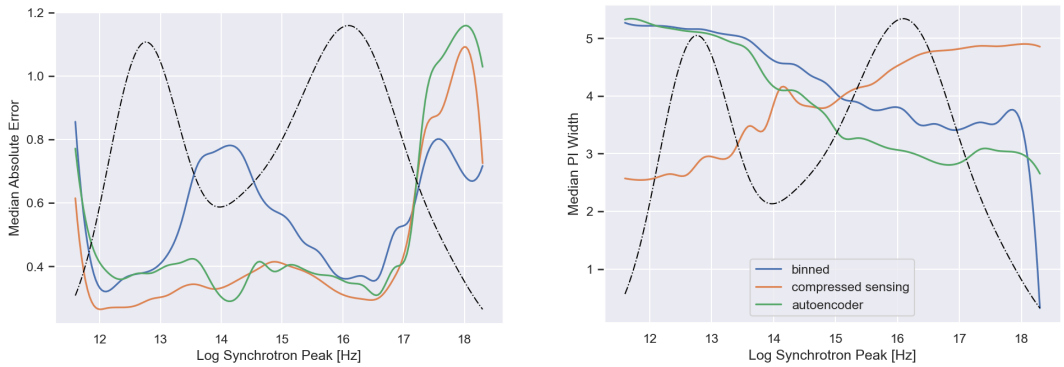


(f) Prediction Interval for autoencoder

Figure 5.3: Illustration of the performance of gradient boosting for each feature set. The left column shows a histogram of the predicted ν_{peak} against the ground truth with the median as a solid and the 80% quantile interval as dashed black lines. The right column shows the centered prediction interval (95%) sorted by their width with the ground truth marked as orange dots. Additionally the percentage of outliners above and below their respective prediction interval is shown.

Parameter	Binning	Compressed Sensing	Autoencoder
# Trees (2.5%)	100	175	250
Learning rate (2.5%)	0.2	0.01	0.2
Max tree depth (2.5%)	2	2	2
Min samples leaf (2.5%)	30	10	20
Min samples split (2.5%)	10	20	30
# Trees (50%)	150	250	350
Learning rate (50%)	0.05	0.05	0.15
Max tree depth (50%)	20	20	25
Min samples leaf (50%)	1	5	20
Min samples split (50%)	20	5	2
# Trees (97.5%)	100	150	150
Learning rate (97.5%)	0.01	0.1	0.01
Max tree depth (97.5%)	15	2	2
Min samples leaf (97.5%)	30	25	30
Min samples split (97.5%)	20	5	30
λ		1×10^{-4}	
Tolerance		5×10^{-4}	

Table 5.2: Hyperparameters used for gradient boosting training after tuning. Note that for each quantile a separate model was trained and tuned, thus each model has a different set of hyperparameters.



(a) Median absolute error

(b) Median PI width

Figure 5.4: Median absolute error of the prediction and the median prediction interval width for gradient boosting on the test set for each feature set. The curve was created by a moving window and smoothed with a cubic spline. The dashed line shows the density of the ground truth in the test set.

to much and thus was not able to generalize the data. It seems that the algorithm is not suitable for this particular data set.

5.3 Neural Network

Parameter	Binning	Compressed Sensing	Autoencoder
Input size	52	206	64
Layer 1	152 (11.62%)	120 (3.05%)	112 (29.84%)
Layer 2	80 (14.95%)	104 (3.16%)	80 (6.84%)
Layer 3	72 (2.46%)	80 (33.71%)	64 (17.86%)
Layer 4	48 (3.21%)	64 (19.72%)	64 (1.82%)
Learning rate	1.810×10^{-3}	9.788×10^{-4}	6.658×10^{-3}
Weight decay	1.635×10^{-5}	2.429×10^{-4}	4.235×10^{-5}
λ		1×10^{-4}	
Tolerance		5×10^{-4}	

Table 5.3: Hyperparameters used for neural networks training after tuning. Each layer has two hyperparameters: The number of neurons in that layer and the dropout chance noted in parentheses behind.

For neural networks a method to calculate prediction intervals have yet to be discussed. In fact, there are numerous algorithms to accomplish this as layed out by Cavlo-Pardo, Mancini, and Olmo, 2021. In that paper a novel approach has been proposed using an ensemble of slightly different neural networks architectures replacing the classic dropout with a bernoulli mask and only predicting the value. The prediction interval is then calculated using the ensemble. However, this method's performance turned out to be unsatisfactory as in this case it only produced a nearly constant sized prediction interval. Instead it is calculated as described by Lakshminarayanan, Pritzel, and Blundell, 2017, where the network not only predicts the actual value but also the standard deviation. The loss to train the model is than the negative log likelihood of a gaussian distribution. The neural networks used in this work outputs the actual ν_{peak} with the squared standard deviation σ^2 and were trained by minimizing double the negative log likelihood following the previously established naming scheme for algorithms:

$$L(y, \hat{y}(x)) = -2 \log p(y|x) = \log \sigma^2(x) + \frac{(y - \hat{y}(x))^2}{\sigma^2(x)} + \text{const} \tag{5.1}$$

The results are further improved by utilizing an ensemble consisting of $M = 20$ members. The individual results are combined using the following equations:

$$\hat{y}_*(x) = \frac{1}{M} \sum_{m=1}^{M} \hat{y}_m(x) \tag{5.2}$$

$$\sigma_*^2(x) = \frac{1}{M} \left(\sum_{m=1}^{M} \sigma_m^2(x) + \hat{y}_m^2(x) \right) - \hat{y}_*^2(x) \tag{5.3}$$

Each neural network is 5 layers deep, while each hidden layer consists of a linear layer, followed by batch normalization, dropout and ends with ReLU as activation function. Deeper networks had no benefit in performance while increasing the time

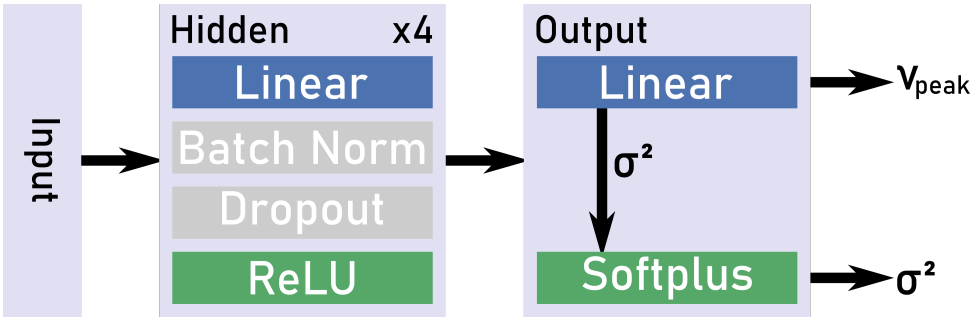


Figure 5.5: Illustration of neural network architecture. It consists of 4 hidden layers with a linear layer, batch normalization, dropout and ReLU as activation function. The output layer is a linear one outputting ν_{peak} and σ^2. A softplus further ensures $\sigma^2 > 0$.

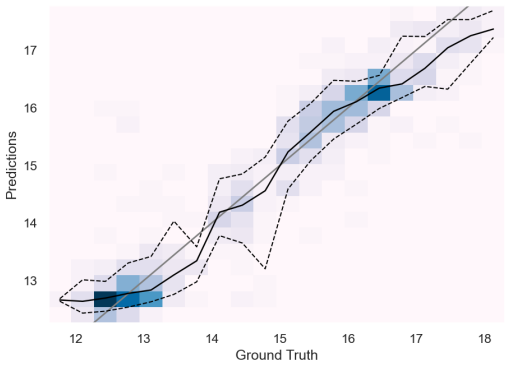
needed for training. The output layer following the last hidden layer is a simple linear layer outputting ν_{peak} and σ^2. To the later one was additionally a softplus applied to enforce positive deviation. The architecture is illustrated in figure 5.5. Additionally, both the input and output are standardized, i.e. scaled such that each individual input and output have zero mean and unit deviation. Unfortunately, this is a problem to the binning feature set, since it uses zero to indicate missing data, which is now a valid value. This is circumvented by reassigning zero to to these values after standardization and adding a mask to the input with one marking valid and zero missing data, thus expanding the input dimension from 26 to 52.

Each individual model has been trained for up to 800 epochs, but only the best one of each ensemble member with regard to the loss on an validation set has been saved, which has been achieved in most cases in only a few hundred epochs. The validation set holds 1,000 samples. Both sets are organized in batches of 64 samples. If the performance did not improve for 10 epochs, the learning rate has been reduced to a tenth starting with the values listed earlier.

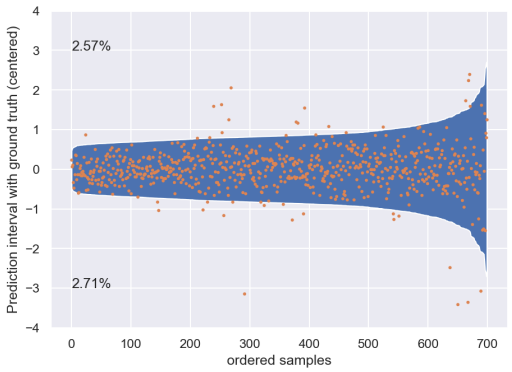
Figure 5.6 shows the performance of the neural network ensemble for each feature set. It is the best performing algorithm so far especially in regard to prediction interval width. Surprisingly, despite earlier results the usage of compressed sensing seem to actually slightly harm the performance. An explanation to this might be that the neural network performs so well, that it became susceptible to the noise introduced by compressed sensing if there is only few data available outweighing its benefits. As stated earlier, compressed sensing fails in these cases and starts to hallucinate. The autoencoder despite outperforming compressed sensing still underlies to the binned data, which is thus again the best feature set.

The neural networks error distribution is shown in figure 5.7 and behave for the prediction error as expected. The distribution for the median prediction interval however is remarkable as they are the only ones where the edge cases, i.e. samples with very low or very high ν_{peak}, are not predicted with relative to other frequency ranges wide intervals. Especially the one trained on binned data handles these cases very well.

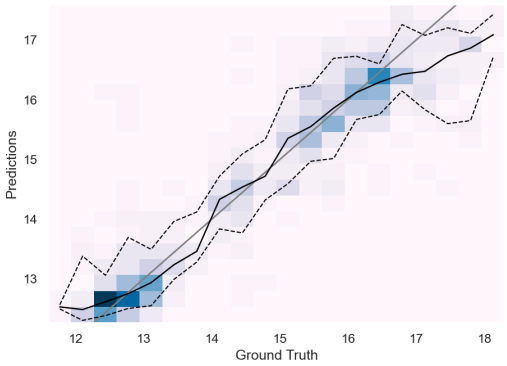
To further justify the usage of an ensemble instead of a single model, figure 5.8 shows the performance of a single ensemble member trained on the binned feature set. The edges cases, i.e. very small or high ν_{peak}, which were already problematic in earlier models are more well behaved in the ensemble. Especially the prediction interval



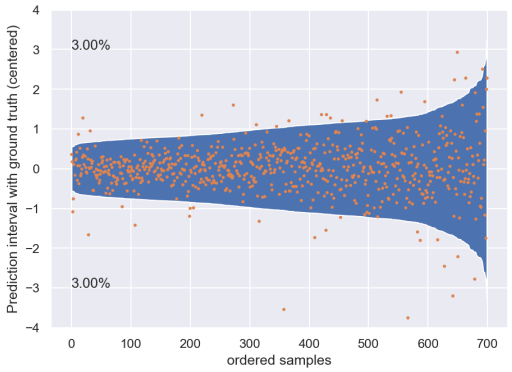
(a) Predictions for binned features



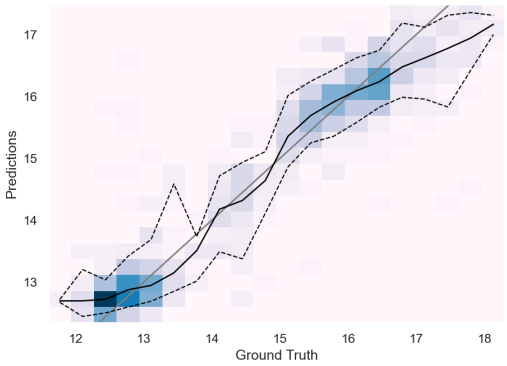
(b) Prediction Intervals for binned features



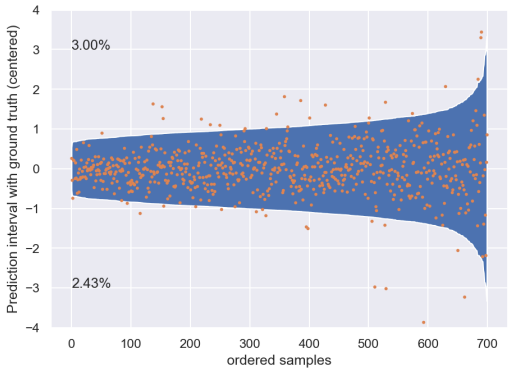
(c) Predictions for compressed sensing



(d) Prediction Intervals for compressed sensing

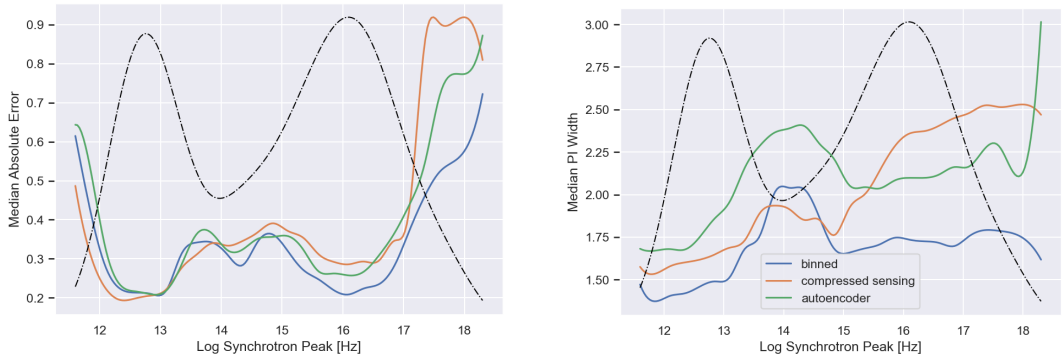


(e) Prediction for autoencoder



(f) Prediction Interval for autoencoder

Figure 5.6: Illustration of the performance of neural networks for each feature set. The left column shows a histogram of the predicted ν_{peak} against the ground truth with the median as a solid and the 80% quantile interval as dashed black lines. The right column shows the centered prediction interval (95%) sorted by their width with the ground truth marked as orange dots. Additionally the percentage of outliners above and below their respective prediction interval is shown.



(a) Median absolute error **(b)** Median PI width

Figure 5.7: Median absolute error of the prediction and the median prediction interval width for neural networks on the test set for each feature set. The curve was created by a moving window and smoothed with a cubic spline. The dashed line shows the density of the ground truth in the test set.

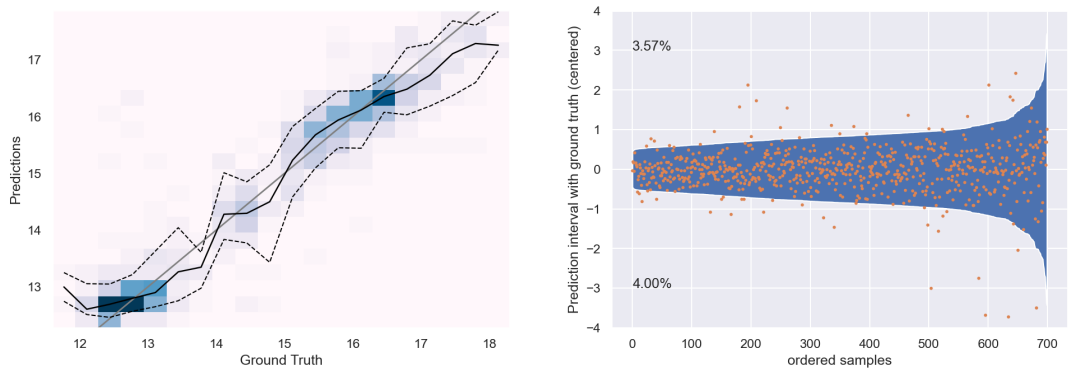


Figure 5.8: Performance of a single member in the neural network ensemble trained on binned data. Again, the left one shows the histogram with the median as solid and the 90% and 10% as dashed black lines, while the right one shows the prediction interval width including coverage rate as indicated with ratio of outliners above and below.

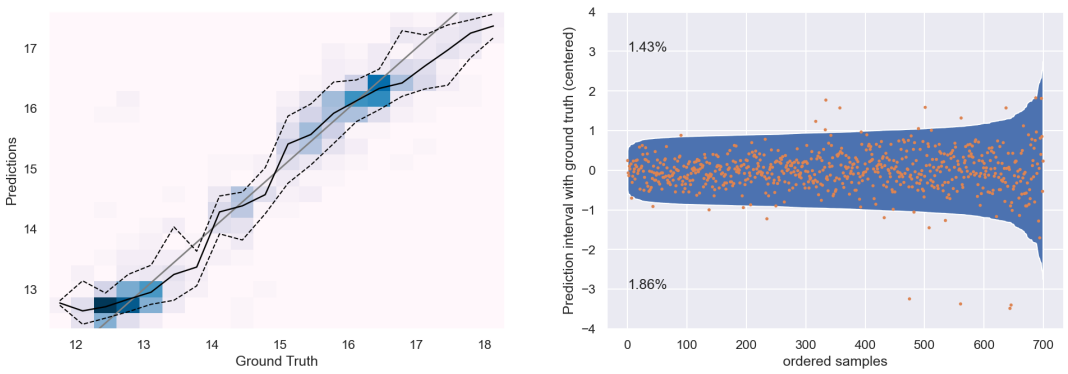


Figure 5.9: Performance of a combination using an RNN encoder and a FCN estimator. the predicted ν_{peak} against the ground truth with the median as a solid and the 80% quantile interval as dashed black lines. The right column shows the centered prediction interval (95%) sorted by their width with the ground truth marked as orange dots. Additionally the percentage of outliners above and below their respective prediction interval is shown. Note that this model is very likely biased.

Parameter	Value
Layer 1	144 (17.10%)
Layer 2	88 (26.39%)
Layer 3	72 (2.11%)
Layer 4	48 (18.23%)
Learning rate	3.105×10^{-3}
Weight decay	5.485×10^{-5}

Table 5.4: Hyperparameters after tuning used to train the direct combination of RNN encoder with an FCN. Each layer has two hyperparameters: The number of neurons in that layer and the dropout chance noted in parentheses behind.

has not only shrunk in width but even shows a better coverage.

One might wonder wether the extra step with the autoencoder hurts the performance in comparison to a direct approach of training the RNN encoder alongside the FCN estimator. Such a model was trained to answer the question. Like the other ones before it consists of an ensemble of 20 individually trained models using the hyperparameters shown in table 5.4. Its performance is illustrated in figure 5.9 and is similar to the unbiased ones shown before. Therefore the autoencoder did not harm the performance. Surprisingly, it is still slightly worse than the one trained on the binned feature set.

5.4 Summary

Table 5.5 gives a quick overview of all previously shown models by providing the median absolute error and median prediction interval width with the 25% and 75% quantile calculated on the test set. As indicated by the median absolute error most models perform similar on the prediction precision with the exception of gradient boost performing far worse and neural network far better both trained on binned data. A different picture is drawn by the median prediction interval width: As mentioned earlier the gradient boost has too wide prediction intervals to be of any use. The random forest has far narrower prediction intervals but even the best one is out-

	Binning	Compressed Sensing	Autoencoder
Random Forest	$0.300^{+0.313}_{-0.170}$	$0.376^{+0.324}_{-0.189}$	$0.400^{+0.310}_{-0.200}$
Gradient Boost	$0.469^{+0.427}_{-0.257}$	$0.356^{+0.330}_{-0.195}$	$0.403^{+0.333}_{-0.220}$
Neural Network	$0.265^{+0.242}_{-0.133}$	$0.314^{+0.257}_{-0.173}$	$0.316^{+0.255}_{-0.167}$

(a) Median absolute error

	Binning	Compressed Sensing	Autoencoder
Random Forest	$2.583^{+1.325}_{-0.683}$	$2.301^{+1.084}_{-0.501}$	$3.100^{+0.973}_{-0.900}$
Gradient Boost	$4.675^{+0.445}_{-1.176}$	$4.264^{+0.544}_{-1.158}$	$3.804^{+1.058}_{-0.835}$
Neural Network	$1.687^{+0.307}_{-0.197}$	$2.074^{+0.307}_{-0.197}$	$2.085^{+0.417}_{-0.261}$

(b) Median Prediction Interval Width

Table 5.5: Summary of the models performance showing the median absolute error of the prediction and the median prediction interval width including their 25% and 75% quantile.

performed by any neural network. Once again the neural network trained on binned data performs the best making it the overall leader. Interestingly, from a prediction interval point of view each algorithm performed best on a different feature set.

CHAPTER 6

Implementation: BlaSE

Following rhe results obtained through this thesis a tool was created with the intent of being usable out of the box: *BlaSE* (**Bla**zar **S**ynchtroton Peak **E**stimator) [1] uses an ensemble of neural networks processing a binned version of the blazar's SED as described earlier to predict ν_{peak} with an 95% prediction interval.

The training set has been split 5-folds to allow bagging, i.e. training with only 4 out 5 splits to leaf the remaining as unseen data, and underwent the same data preprocessing including augmentation as described before. This makes it possible to evaluate the model on the whole training set by using an OOB estimation. For each possible bagging 5 models were trained, thus for each data set member an ensemble of 5 models can be used. For truly unseen data even 25 models are available.

Metric	Value
Median Absolute Error	$0.255^{+0.204}_{-0.140}$
Median Prediction Interval Width	$1.702^{+0.268}_{-0.204}$

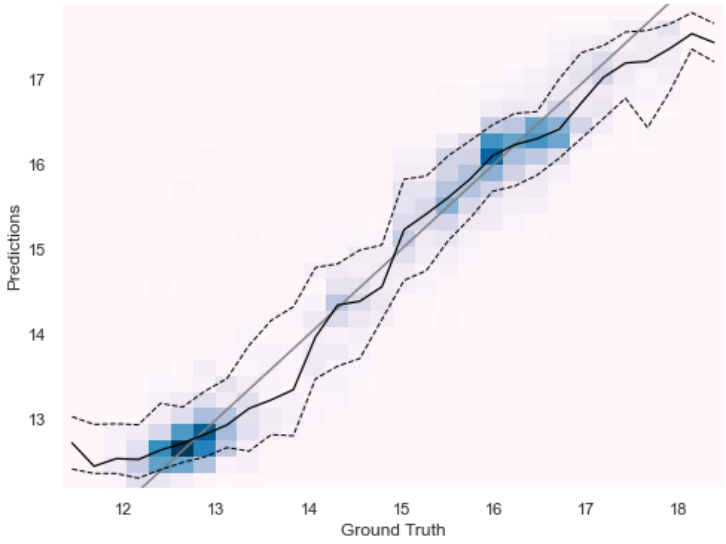
Table 6.1: Performance of BlaSE showing the median absolute error and the median prediction interval width including their 25% and 75% quantile.

Figure 6.1 and table 6.1 show the performance of BlaSE evaluated on the whole training set using OOB estimation. The prediction precision further improved with respect to same method trained without bagging while the prediction interval widths are equally distributed.

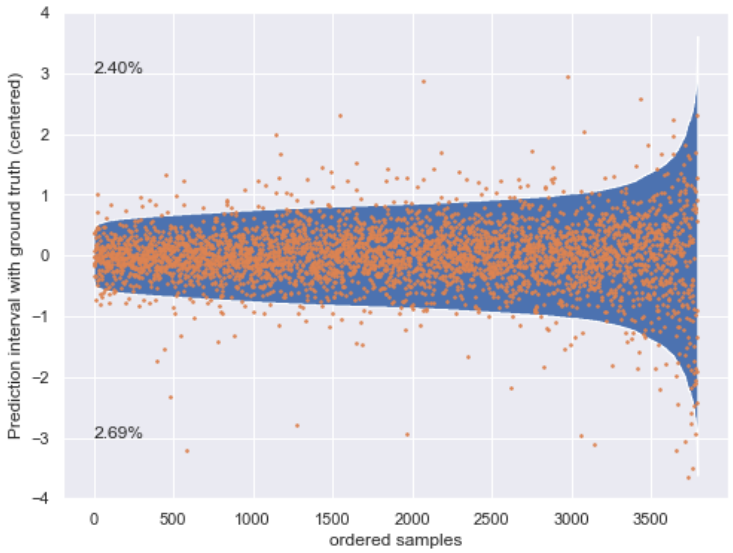
Previously no justification for the assumption of a gaussian distributed error was made. This is now made up for as it is easier with more evaluation samples due to the OOB estimations. Figure 6.2 shows the estimation error distribution for selected prediction intervals denoted as 1σ alongside the expected and a fitted gaussian distribution. One can not only see that the estimation error is indeed gaussian distributed, but also that the calculated prediction interval matches the fitted one.

To be of practical use a model should never produce overconfident but false estimations, i.e. huge prediction error but a small prediction interval. To test this *BlaSE* was fed with made up data that show no resemblance to typical SEDs of blazars.

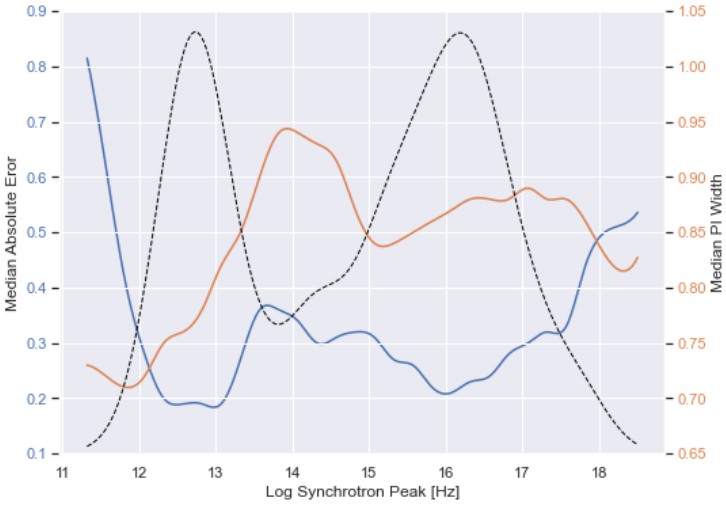
[1] https://github.com/tkerscher/blase or via *pip install blase*



(a) Histogram of the prediction including the median as a solid as well as the 10% and 90% quantile as dashed black lines.



(b) Centered prediction inerval widths with the ground truth marked as orange dots. Additionally, the coverage is denoted as percentage of outliners above and below the intervals.



(c) Distribution of median absolute error in prediction and median prediction interval width. Both curves are produced by a moving average and smoothed using a cubic spline. Additionally, shown as dotted black line is the distribution of the test set.

Figure 6.1: Illustration BlaSE's performance as established before using the OOB estimation, i.e. the whole unaugmented training set is used.

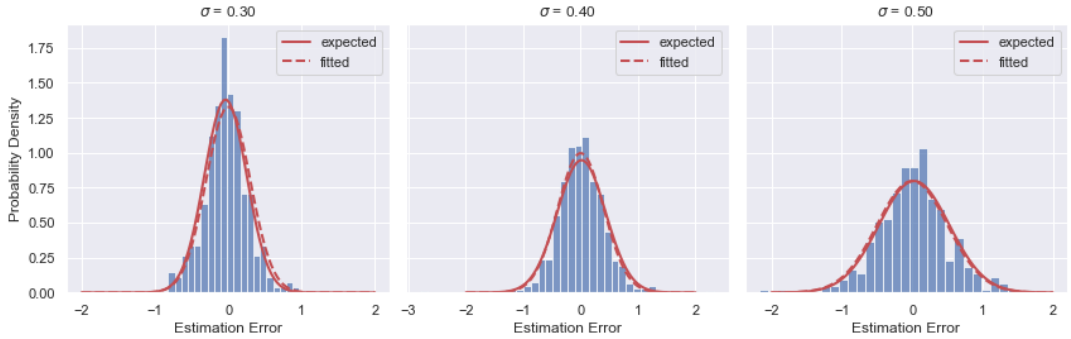


Figure 6.2: Distribution of estimation error with a specific prediction interval noted as 1σ alongside the expected gaussian distribution as solid and the fitted one as dashed line.

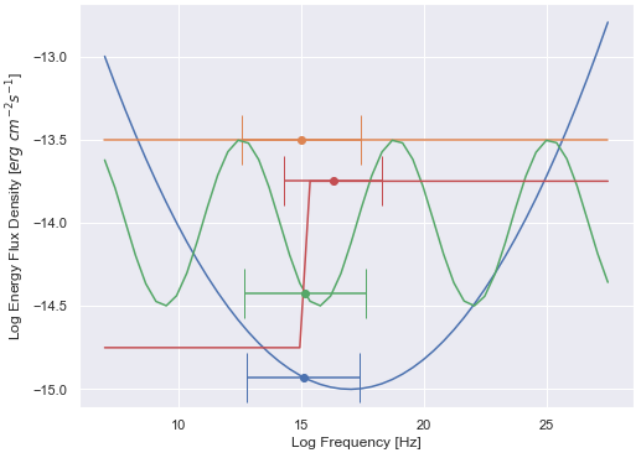


Figure 6.3: Plots of non blazar-like data.

Function	Peak	Error
$(x-17.0)^2/50-15.0$	15.100	2.290
-13.5 *(const)*	15.011	2.409
$\cos(x)/2-14.0$	15.150	2.480
$\Theta(x-15.0)-14.75$	16.308	2.008

Table 6.2: Prediction of non blazar-like data.

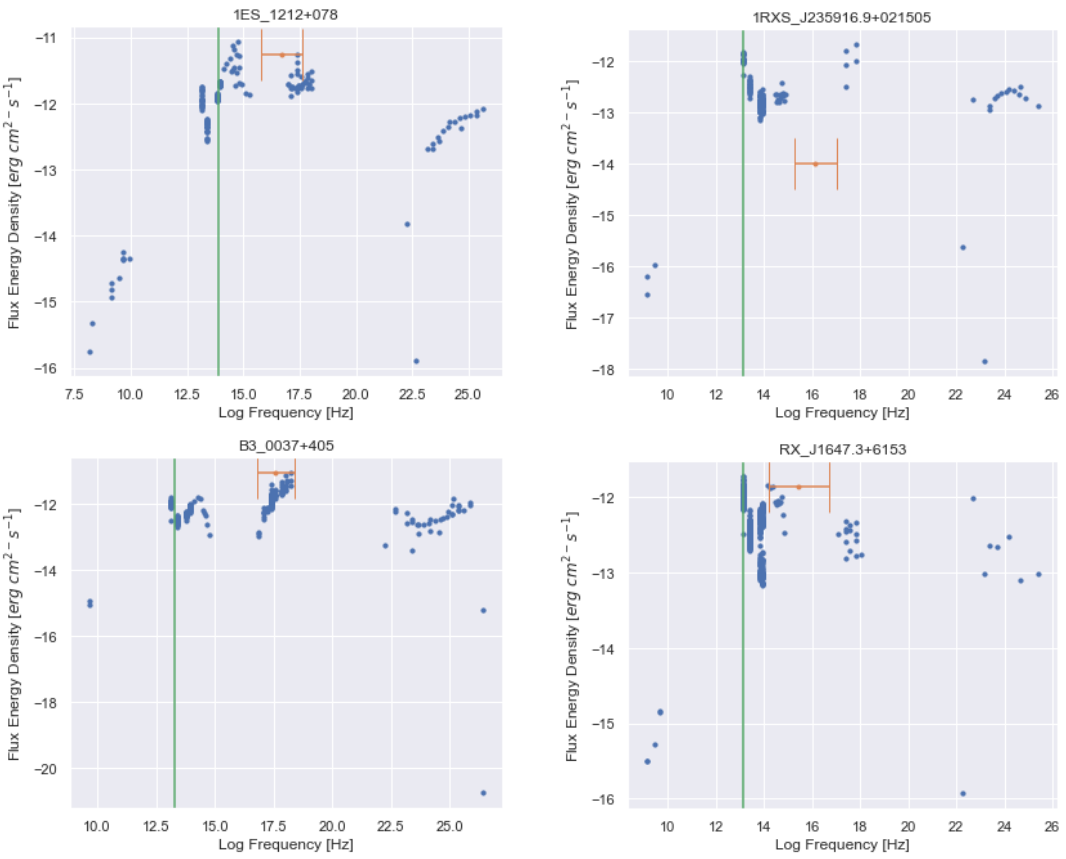


Figure 6.4: Plots of SEDs where *BlaSE* disagrees with the *4LAC-DR2* catalogue about the class of the blazar as determined by the synchrotron peak. The vertical green line shows the catalogue value while the orange shows the new estimate with the error bar indicating the 95% prediction interval.

Figure 6.3 and table 6.2 shows the functions used to generate the data as well as the produced estimates. All estimates show a rather large prediction interval above 2.0 clearly indicating that the model is not sure about its prediction. The latter is most of the times around 15.1, which seems to be the models *"default"* estimation.

Finally, *BlaSE* was applied to the *4LAC-DR2* [The Fermi-LAT collaboration, 2019] catalogue. In its actual version it consists of 3,511 entries of which 1,053 have not been used to train the model and 706 were not even assigned a ν_{peak}. If a entry was also part of the training set like before an OOB estimate was made, thus for all entries in the catalogue a genuine estimate was made. The results can be found in *BlaSE*'s repository.

Figure 6.4 shows some particular interesting examples as on these the *4LAC-DR2* catalogue and *BlaSE* disagree about the blazar's classification. A common pitfall here is to confuse the synchrotron with the galaxy's thermal background radiation. The neural network managed to overcome that challenge on several occasions.

CHAPTER 7

Conclusion and Outlook

The thesis started off with an introduction to blazars as part of the AGN zoo as well as an explanation of the utilized machine learning methods and algorithms. This was followed by the modelling the data preparation process in which the data set as produced by the *VOUBlazar* was parsed and sanitized, i.e. removing illogical entries such as zero flux or values outside their error bounds. Only frequency and energy flux density were kept to form the base feature set, which furthermore were transformed into the decadic log space as this deemed to improve the performance. Before any further data preprocessing a test set consisting of 700 SEDs were put aside from the initial 3,793.

It has been shown that the data set suffers from bias as some values of ν_{peak} are more prominently represented than others, thus making it beneficial for latter models to neglect the rare ones. Furthermore, some frequency ranges around 100 GHz were measured especially for a narrow range of ν_{peak} (mainly LSPs) allowing the models to predict them by the mere existence of such frequencies without any meaningful generalization while producing good results on the test set, ultimately rendering the models useless.

The bias caused by the uneven ν_{peak} distribution was significantly reduced by oversampling especially rare samples while also deleting especially prominent frequencies, which also led to an increased training set size of around 10,000. This however did not fix the second source of bias.

Three different feature sets based on the base one were proposed: Binning, reconstruction through compressed sensing and compressing by an autoencoder. Different and uneven binning widths with the goal of an even density were tested but turned out to make little difference. Since biased bins had to be neglected the one with the best frequency coverage without being to fine-meshed was chosen. The compressed sensing tried to reconstruct missing data and thus removing the bias allowing to use all the available data. For this purpose a custom basis in which the SEDs have a sparse representation was made. Lastly, the autoencoder used bidirectional RNN for both the en- and decoder to compress the binned data. This allowed smaller bins without leaving most of them empty, but the biased ones had once again to be neglected.

The feature sets were used as input for three different machine learning algorithm thus totalling in 9 distinct models. The algorithms were random forest, gradient boosting and neural networks. Their task was to not only predict ν_{peak} but also to

produce a prediction interval. The random forest showed decent performance on all feature sets and improved with compressed sensing. The gradient boosting however could not provide useful results. The neural networks uses a gaussian log likelihood as loss function and performed on all feature sets very well with the one trained on the binned one being the overall best one. It seems that this algorithm performed that well, that it became susceptible to the noise introduced by the other feature sets. Also noteworthy is that the second best model prediction wise was the random forest trained on binned data, which was magnitudes faster to train and far easier to utilize than the other algorithms. It shows that simpler models and algorithm are worth considering especially if time and resources are scarce.

Using the results of this thesis *BlaSE*, a ready to use tool to predict ν_{peak} with a prediction interval was made. Following the previously best model it consisted of an ensemble of neural networks trained using a gaussian log likelihood. By further using bagging, the whole training set could be used for evaluation. Finally, the tool was used to produce estimates for the *4LAC-DR2* catalogue and not only filled in 706 missing synchrotron peaks but also overcome some caveats in their estimation such as confusing the galaxy's thermal radiation with the synchrotron to a certain degree.

Unfortunately, due to the very nature of a bachelor thesis some ideas could not be tested. The original design of the autoencoder was trained on synthetic data, providing the encoder only a subset of the actual data to mimic real measurements while the decoder tries to reconstruct the whole SED. The training took several days but finally failed to reconstruct real SED leaving no time for a second try. Since the simulation of SEDs is a difficult task this was might the result of bad synthetic data. One might also want to try replacing the RNN of the decoder with a simple FCN. Since only frequency and energy flux density were used there might be room for improvement in the selection of data features. Because all feature sets used some sort of binning the variability of the blazar was also lost providing even more data to gather.

Ultimately, the most important thing for machine learning is the amount of available data. With new sources of data becoming available in the next years such as the James Webb Space Telescope or multi messenger astronomy more accurate models will be possible even using the methods showed in this thesis.

APPENDIX A

Binning

	Begin	End
1	6.90	8.30
2	8.30	9.10
3	9.10	9.30
4	9.30	9.50
5	9.50	9.80
6	9.80	10.10
7	10.10	10.50
8	10.50	10.90
9	10.90	11.10
10	11.10	11.30
11	11.30	11.50
12	11.50	12.50
13	12.50	13.30
14	13.30	13.60
15	13.60	13.90
16	13.90	14.20
17	14.20	14.50
18	14.50	14.70
19	14.70	15.10
20	15.10	16.70
21	16.70	17.20
22	17.20	17.50
23	17.50	17.70
24	17.70	17.90
25	17.90	19.70
26	22.10	22.40
27	22.40	22.70
28	22.70	23.30
29	23.30	23.50
30	23.50	23.90
31	23.90	24.40
32	24.40	24.70
33	24.70	25.30
34	25.30	27.50

Table A.1: Bin edges used for data augmentation.

	Begin	End
1	7.05	8.25
2	8.25	8.80
3	8.80	9.30
4	9.30	9.60
5	13.05	13.25
6	13.25	13.45
7	13.75	13.85
8	13.85	14.00
9	14.00	14.45
10	14.45	14.55
11	14.55	14.65
12	14.65	14.75
13	14.75	14.95
14	14.95	15.35
15	16.65	17.15
16	17.15	17.40
17	17.40	17.75
18	17.75	17.95
19	17.95	19.65
20	22.15	22.35
21	22.65	22.75
22	23.15	23.30
23	23.30	23.50
24	23.65	23.80
25	24.15	24.30
26	24.65	24.80
27	24.80	25.00
28	25.00	25.15
29	25.15	25.35
30	25.35	25.65
31	25.65	26.15
32	26.15	27.45

Table A.2: Bin edges for 0.1 base width. Values are given in log space.

	Begin	End
1	6.90	8.30
2	8.30	9.10
3	9.10	9.30
4	9.30	9.50
5	12.50	13.30
6	13.30	13.60
7	13.60	13.90
8	13.90	14.20
9	14.20	14.50
10	14.50	14.70
11	14.70	15.10
12	15.10	16.70
13	16.70	17.20
14	17.20	17.50
15	17.50	17.70
16	17.70	17.90
17	17.90	19.70
18	22.10	22.40
19	22.40	23.10
20	23.10	23.30
21	23.30	23.50
22	23.50	23.90
23	23.90	24.40
24	24.40	24.70
25	24.70	25.30
26	25.30	27.50

Table A.3: Bin edges for 0.2 base width. Values are given in log space.

	Begin	End
1	7.05	8.55
2	8.55	9.30
3	9.30	9.75
4	12.20	13.35
5	13.35	13.65
6	13.65	14.10
7	14.10	14.55
8	14.55	14.85
9	14.85	15.45
10	16.65	17.25
11	17.25	17.55
12	17.55	17.85
13	17.85	19.65
14	22.05	22.50
15	22.50	22.95
16	22.95	23.25
17	23.25	23.55
18	23.55	24.00
19	24.00	24.45
20	24.45	24.90
21	24.90	25.35
22	25.35	27.45

Table A.4: Bin edges for 0.3 base width. Values are given in log space.

	Begin	End
1	6.75	8.75
2	8.75	9.25
3	9.25	9.75
4	12.25	13.25
5	13.25	13.75
6	13.75	14.25
7	14.25	14.75
8	14.75	16.00
9	16.00	17.25
10	17.75	19.25
11	19.25	22.75
12	22.75	23.25
13	23.25	23.75
14	23.75	24.25
15	24.25	24.75
16	24.75	25.25
17	25.25	27.75

Table A.5: Bin edges for 0.5 base width. Values are given in log space.

Bibliography

Abadi, Martín et al. (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. URL: https://www.tensorflow.org/.

Berk, Richard A. (2020). *Statistical Learning from a Regression Perspective*. 3rd ed. Springer International Publishing.

Bishop, Christopher M. (2016). *Pattern recognition and machine learning*. Springer.

Borowiec, Steven (Mar. 15, 2016). *AlphaGo seals 4-1 victory over Go grandmaster Lee Sedol*. URL: https://www.theguardian.com/technology/2016/mar/15/googles-alphago-seals-4-1-victory-over-grandmaster-lee-sedol.

Breiman, Leo (2001). In: *Machine Learning* 45.1, pp. 5–32. DOI: 10.1023/a:1010933404324. URL: https://doi.org/10.1023/a:1010933404324.

Brunton, Steven L. and J. Nathan Kutz (2019). *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*. Cambridge University Press. DOI: 10.1017/9781108380690.

Cavlo-Pardo, Hector, Tillio Mancini, and Jose Olmo (2021). *Ensemble Predictors for Deep Neural Networks*. eprint: arXiv:2010.04044v2.

Cerruti, M. et al. (2018). "Lepto-hadronic single-zone models for the electromagnetic and neutrino emission of TXS 0506+056". In: DOI: 10.1093/mnrasl/sly210. eprint: arXiv:1807.04335.

Chang, Yu-Ling, Carlos Brandt, and Paolo Giommi (2019). "The Open Universe VOU-Blazars tool". In: DOI: 10.1016/j.ascom.2019.100350. eprint: arXiv:1909.11455.

Chang, Yu-Ling et al. (2019). "The 3HSP catalogue of Extreme & High Synchrotron Peaked Blazars". In: DOI: 10.1051/0004-6361/201834526. eprint: arXiv:1909.08279.

Chen, Tianqi and Tong He (2014). "Higgs Boson Discovery with Boosted Trees". In: *Proceedings of the 2014 International Conference on High-Energy Physics and Machine Learning - Volume 42*. HEPML'14. JMLR.org, 69–80.

Chen, Yang, Yu-Kun Lai, and Yong-Jin Liu (2018). "CartoonGAN: Generative Adversarial Networks for Photo Cartoonization". In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9465–9474. DOI: 10.1109/CVPR.2018.00986.

Digital Science (n.d.). *Dimensions*. URL: https://app.dimensions.ai/discover/publication?search_mode=content&and_facet_for=2746 (visited on 05/26/2021).

Du, K.-L and Swamy M N S. (2019). *Neural networks and statistical learning*. 2nd ed. Springer.

Falcon WA, et al. (2019). "PyTorch Lightning". In: *GitHub* 3. URL: https://github.com/PyTorchLightning/pytorch-lightning.

Fraga, Bernardo M. O. et al. (2020). "Deep Learning Blazar Classification based on Multi-frequency Spectral Energy Distribution Data". In: DOI: 10.1093/mnras/stab1349. eprint: arXiv:2012.15340.

Glorot, Xavier and Yoshua Bengio (2010). "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Yee Whye Teh and Mike Titterington. Vol. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR, pp. 249–256. URL: http://proceedings.mlr.press/v9/glorot10a.html.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep learning*. The MIT Press.

Greicius, Tony (2016). *When Computers Were Human*. URL: https://www.nasa.gov/feature/jpl/when-computers-were-human.

Grier, David Alan (Dec. 2013). *When Computers Were Human*. Princeton University Press. DOI: 10.1515/9781400849369. URL: https://doi.org/10.1515/9781400849369.

Guest, Dan, Kyle Cranmer, and Daniel Whiteson (2018). "Deep Learning and its Application to LHC Physics". In: DOI: 10.1146/annurev-nucl-101917-021019. eprint: arXiv:1806.11484.

Harding, L. and L. Barden (May 12, 2011). "From the archive, 12 May 1997: Deep Blue win a giant step for computerkind". In: *The Guardian*. URL: https://www.theguardian.com/theguardian/2011/may/12/deep-blue-beats-kasparov-1997.

Hastie, Trevor, Robert Tibshirani, and J. H. Friedman (2009). *The elements of statistical learning: data mining, inference, and prediction*. 2nd ed. Springer.

Heaton, Jeff (2017). "An Empirical Analysis of Feature Engineering for Predictive Modeling". In: DOI: 10.1109/SECON.2016.7506650. eprint: arXiv:1701.07852.

IceCube Collaboration et al. (2019). "Time-integrated Neutrino Source Searches with 10 years of IceCube Data". In: DOI: 10.1103/PhysRevLett.124.051103. eprint: arXiv:1910.08488.

Ioffe, Sergey and Christian Szegedy (2015). "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *Proceedings of the 32nd International Conference on Machine Learning*. Ed. by Francis Bach and David Blei. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, pp. 448–456. URL: http://proceedings.mlr.press/v37/ioffe15.html.

Kaggle (Sept. 28, 2012). *Titanic - Machine Learning from Disaster*. URL: https://www.kaggle.com/c/titanic (visited on 05/26/2021).

Kaggle (2020). *State of Machine Learning and Data Science 2020*.

Kovačević, Miloš et al. (2020). "Classification of Blazar Candidates of Uncertain Type from the Fermi LAT 8-Year Source Catalog with an Artificial Neural Network". In: DOI: 10.1093/mnras/staa394. eprint: arXiv:2002.10256.

Lakshminarayanan, Balaji, Alexander Pritzel, and Charles Blundell (2017). *Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles*. eprint: arXiv:1612.01474v3.

Liaw, Richard et al. (2018). "Tune: A Research Platform for Distributed Model Selection and Training". In: *arXiv preprint arXiv:1807.05118*.

Lustig, Michael et al. (2008). "Compressed Sensing MRI". In: *IEEE Signal Processing Magazine* 25.2, pp. 72–82. DOI: 10.1109/MSP.2007.914728.

Mairal, Julien et al. (2009). "Online Dictionary Learning for Sparse Coding". In: *Proceedings of the 26th Annual International Conference on Machine Learning*. ICML '09. Montreal, Quebec, Canada: Association for Computing Machinery, 689–696.

ISBN: 9781605585161. DOI: 10.1145/1553374.1553463. URL: https://doi.org/10.1145/1553374.1553463.

Massaro, Enrico et al. (2015). "The 5th edition of the Roma-BZCAT. A short presentation". In: DOI: 10.1007/s10509-015-2254-2. eprint: arXiv:1502.07755.

Meinshausen, Nicolai (Dec. 2006). "Quantile Regression Forests". In: *J. Mach. Learn. Res.* 7, 983–999. ISSN: 1532-4435.

Nguyen, Thanh Thi et al. (2019). *Deep Learning for Deepfakes Creation and Detection: A Survey*. eprint: arXiv:1909.11573.

Padovani, P. et al. (2017). "Active Galactic Nuclei: what's in a name?" In: DOI: 10.1007/s00159-017-0102-9. eprint: arXiv:1707.07134.

Paszke, Adam et al. (2019). "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

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

Perlman, Eric S. (2013). "Active Galactic Nuclei". In: *Planets, Stars and Stellar Systems*. Springer Netherlands, pp. 305–386. DOI: 10.1007/978-94-007-5609-0_7. URL: https://doi.org/10.1007/978-94-007-5609-0_7.

Petropoulou, Maria et al. (2020). "Comprehensive Multimessenger Modeling of the Extreme Blazar 3HSP J095507.9+355101 and Predictions for IceCube". In: DOI: 10.3847/1538-4357/aba8a0. eprint: arXiv:2005.07218.

Phi, Michael (2018). *Illustrated Guide to LSTM's and GRU's: A step by step explanation*. URL: https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21.

Rosenblatt, F. (1958). "The perceptron: A probabilistic model for information storage and organization in the brain." In: *Psychological Review* 65.6, 386–408. DOI: 10.1037/h0042519.

Stanford University (Apr. 9, 2016). *A Titanic Probability*. URL: https://web.stanford.edu/class/archive/cs/cs109/cs109.1166/problem12.html (visited on 05/26/2021).

The Fermi-LAT collaboration (2019). "The Fourth Catalog of Active Galactic Nuclei Detected by the Fermi Large Area Telescope". In: DOI: 10.3847/1538-4357/ab791e. eprint: arXiv:1905.10771.

Turing, A. M. (Oct. 1950). "I.—COMPUTING MACHINERY AND INTELLIGENCE". In: *Mind* LIX.236, pp. 433–460. ISSN: 0026-4423. DOI: 10.1093/mind/LIX.236.433. eprint: https://academic.oup.com/mind/article-pdf/LIX/236/433/30123314/lix-236-433.pdf. URL: https://doi.org/10.1093/mind/LIX.236.433.

Urry, C. Megan and Paolo Padovani (1995). "Unified Schemes for Radio-Loud Active Galactic Nuclei". In: DOI: 10.1086/133630. eprint: arXiv:astro-ph/9506063.

Wolpert, David H. and William G. Marcready (Apr. 1997). "No Free Lunch Theorems for Optimization". In: *IEEE Transactions on Evolutionary Computation* 1.1.

Zabalza, V. (2015). "naima: a Python package for inference of relativistic particle energy distributions from observed nonthermal spectra". In: *Proc. of International Cosmic Ray Conference 2015*, p. 922. eprint: 1509.03319.