Insert here your thesis' task.

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Bachelor's thesis

Development, Application and Representation of Algorithms for Discoveries with the ATLAS Forward Proton (AFP) Detector at CERN

Martin Vatrt

Department of Applied Mathematics
Supervisor: doc. Dr. André Sopczak

May 16, 2019

Acknowledgements

I want thank to Doc. Dr. André Sopczak for his help and very kind approach during the work on my thesis, and Dr. Vlasios Petousis for his explanations on the experimental setup.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as school work under the provisions of Article 60(1) of the Act.

In Prague on May 16, 2019 ……

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Vatrt, Martin. *Development, Application and Representation of Algorithms for Discoveries with the ATLAS Forward Proton (AFP) Detector at CERN.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2019.

Abstrakt

Tato práce se zabývá rozšířením výběrových kriterií pro získání signal/background procesů z dat která byla v roce 2016 a 2017 zaznamenána při AFP experimentu v CERNu. Experiment se zabývá interakcí protonů které se ve dvou protichůdných paprscích těsně míjejí. K rozšíření výběrových kritérií bylo použito strojové učení. Práce čtenáře nejprve seznamuje s AFP experimentem a co vysvětluje znamenají signal/background procesy a jakým pomocí jakých podmínek jsou z dat získávány. Následuje poté úvod do strojového učení včetně podrobného vysvětlení metody která byla pro tuto úlohu použita - neuronových sítí.

V praktické části je vysvětlen dataset s nasbíranými daty se kterým jsem pracoval, jsou zde popsány úpravy které jsem v datasetu provedl a kterou machine-learningovou techniku nakonec zvolil. Jsou zde vysvětleny dva přístupy pro klasifikaci záznamů a jak jsem s jejich kombinací došel k finálnímu výsledku.

Na závěr popisuji program webovou stránku, které byly použity jako výstup této práce.

Klíčová slova CERN, AFP, strojové učení, neuronové sítě

Abstract

The goal of this thesis is to extend selection criteria for getting signal/background samples from data collected during the AFP experiment at CERN in years 2016 and 2017. This experiment studies interaction of two near-missing protons in two crossing beams where both of them go in opposite direction. For extending the selection criteria machine learning is used.

The thesis first introduces the AFP experiment and explains signal/background processes and the current critera for retrieving them from data. Then follows an introduction to machine learning with explanation of a method which was requested for realization - neural networks.

The practical part explains the dataset I worked with and which adjustments I made in dataset. Then I explain which machine learning method I have used and why. Also two approaches for getting signal/background samples are presented and how I combined both of them for obtaining the final results.

At the end I describe the output of the thesis - webpage and script for getting desired samples.

Keywords CERN, AFP, machine learning, neural networks

Contents

Introduction **1**

1 AFP experiment **3**
1.1 Signal and background 3
1.2 Catching the muons 4
1.3 Experimental set-up 4
1.4 Current criteria for signal/background separation 6

2 Machine learning **7**
2.1 What is machnine learning 7
2.2 Neural networks 8

3 Realisation **15**
3.1 Used tools 15
3.2 Getting data from .root file 16
3.3 Dataset 18
3.4 First adjustments 20
3.5 Signal/background definition 21
3.6 Creating new columns 22
3.7 Dropping diff column 22
3.8 Using logistic regression 23
3.9 Using neural networks 23
3.10 Standartization of the dataset 23
3.11 Applying method on the whole dataset 26
3.12 Combining probabilities 27
3.13 Final output 30

Conclusion **31**

Bibliography **33**

A Acronyms **35**

B Contents of enclosed CD **37**

List of Figures

1.1 Signal - left and middle diagram, background - right diagram. [2] . 3
1.2 Experimental set-up with description. [2] 5

2.1 Model of a perceptron. [3] 8
2.2 Graph of sigmoid function. [4] 9
2.3 Lineary separable problem. 10
2.4 Lineary non-separable problem. 10
2.5 Example of 3-layer neural net. 12
2.6 Decision borders of MLP with 5 neuron in two hidden layers. . . . 13

3.1 Dataset we work with. 18
3.2 Records for one *event_n*. 20
3.3 Table of biggest weights for logistic regression. 24
3.4 Histograms of columns with biggest weights. 25
3.5 Table with basic statistics for *diff* column for each probability interval. 26
3.6 Histogram with *n_samples* for 5% probability intervals. 27
3.7 Estimated probability calculated from *diff*. 28
3.8 Table with basic statistics for final method *diff* column for each probability interval. 29
3.9 Table with basic statistics for final method *diff* column for each probability interval. 29
3.10 Probabilities for each sample. 30

Introduction

The European Organization for Nuclear Research[1] known as CERN(from French Conseil européen pour la recherche nucléaire) is one of the largest scientific research centers of the world situated at northwest suburb of Geneva on the Franco–Swiss border. It employs more than 2500 scientist and technicians from nationalities all around the world. Its research is focused on what is the universe made of and how it works. One of the main topics is the study of the Higgs boson, antimatter, extending the Standart Model of physics and much more.

The main component of CERN by which are most of the experiments done is Large Hadron Collider - LHC. LHC is the world's largest and most powerful particles accelerator which lies up to 175m underground. The LHC consists of a 27-kilometre ring of superconducting magnets with a number of accelerating structures to boost the energy of the particles along the way. Inside the accelerator, two high-energy particle beams travel at close to the speed of light before they are made to collide. The beams travel in opposite directions in separate beam pipes – two tubes kept at ultrahigh vacuum. They are guided around the accelerator ring by a strong magnetic field maintained by superconducting electromagnets. Then these particles are forced to collide at one of LHC detectors.

One of these detectors is called ATLAS. It is a general purpose detector and investigates a wide range of physics. Newly an AFP[2] detector was added to extend the basic functionality of ATLAS detector. This detector operated for the first time in 2016 and its purpose is to study protons which nearly missed each other at ATLAS and made a short interaction. The project I have collaborated with during the works on my Bachelor's thesis study are these near-miss interactions. My task was to extend the selection criteria for getting the events this project investigates based on the data collected from the ATLAS detector and AFP stations. For this task I was requested to im-

plement a suitable neural network which will do the job.

Having a new criteria and more samples to study might help the project to better understand the interaction and possibly to make new discoveries in physics.

In the next chapters I explain the experimental setup and describe the experiments in more detail. Then I explain how do the neural networks work and why they can be useful for the task.

In the practical thesis part I describe the dataset and the approaches I have chosen to find the best separation model.

Chapter 1

AFP experiment

1.1 Signal and background

At first, it is important to understand the processes which the project studies. When two beams of protons going in the opposite direction cross, there are more types of interactions which can take place, but there are only a few we are interested in. These interactions are called signal. All the other interactions, which may be detected but are not important for the analysis are called background.

When two protons fly very close to each other there is an electromagnetic interaction between them. Because both of these particles have a positive electric charge, they repeal and both particles are deflected from their original trajectory. These particles which escape the beam are caught at the AFP. Problem is that these particles can be dissolved during the interaction. Situations, where at least one proton comes from the interaction is considered as signal in our analysis (left and middle diagram in Fig. 1.1]). These intractions are called "elastic"[2].

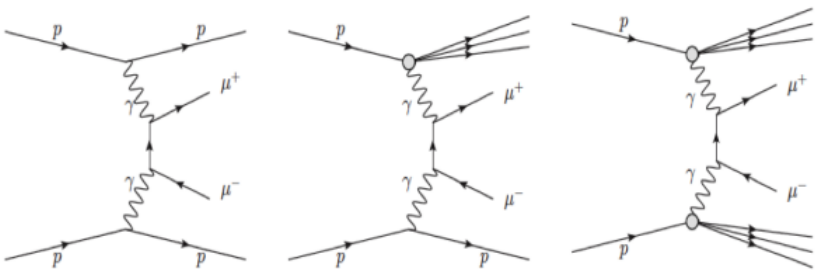


Figure 1.1: Signal - left and middle diagram, background - right diagram. [2]

Situations where both protons dissolve into more particles are considered as background (right digram in Fig. 1.1). Background may also come from different sources. For example, when two particles hit each other, many new particles are created and may be detected in the AFP. Also, when the intact proton hits some AFP component, e.g. magnet or pipe, it is smashed and creates a showers of particles we are not interested in. All these background situations need to be filtered out.

1.2 Catching the muons

From the quantum physics perspective, electromagnetic interaction is done by sending virtual photons $\gamma\gamma$ from both particles. When these two photons meet, they create a muon pair $\mu^+\mu^-$ which is shown in Fig 1.1. These muons are caught in ATLAS and are essential for finding deflected protons at the AFP.

When two protons interract they both lose a fraction of their momentum/energy. From the law of conservation of energy it is known that this lost energy must have transformed into something else. And it is these newly created muons the energy has transformed to. From Einstein's equation $E = mc^2$ we know there is a relation between energy and mass. That means that we can calculate the energy loss from elastic interaction from the mass of the muons. For this we use the following formula 1.1 [2] where s is the centre of mass energy of the proton beams, and η the pseudo-rapidity angle(with respect to the beam pipe):

$$\xi_{\mu\mu} = \frac{m_{\mu\mu}}{\sqrt{s}} e^{\pm\eta} \tag{1.1}$$

1.3 Experimental set-up

At LHC, the beam is in fact separated into bunches of protons which are 25 ns away from each other. Each segment contains about 10^{14} protons. At interaction point(IP) these segments are forced to cross each other using the magnetic fields and particles interact there, and some continue to the AFP stations. AFP contains 4 stations – two stations on the left side from the IP and two on the right side. These two stations on both sides are identified as NEAR and FAR station. NEAR station is situated 205 m away from IP and FAR station 217 m away from IP. Each AFP station contains 4 silicon planes, each having a resolution of 336x80 pixels – electro-sensitive units. The area of these planes is 50x250 m^2. Each plane is placed about 2 mm away from the beam axis to detect only deflected particles. Since the resolution of the detector in X direction is much higher than in Y direction, very often only X

direction is taken in account in the analysis.

Another aspect of the AFP are the magnets along the beam pipe. This magnetic field are used to attract particles which escape them beam evelope. Particles with lower momentum change their trajectory more significantly and have higher bias from beam axis. Based on the properties of magnets the following [2] formula was derived:

$$x(\xi) = -119\xi - 164\xi^2 \tag{1.2}$$

Using this formula we can calculate the X position of a particle if we know it's momentum loss ξ. By X position is meant the proton hit on the AFP pixel detector. This is also how we find our signal candidates since we can calculate the momentum loss of deflected protons from muons. Figure 1.2 shows the whole AFP setup:

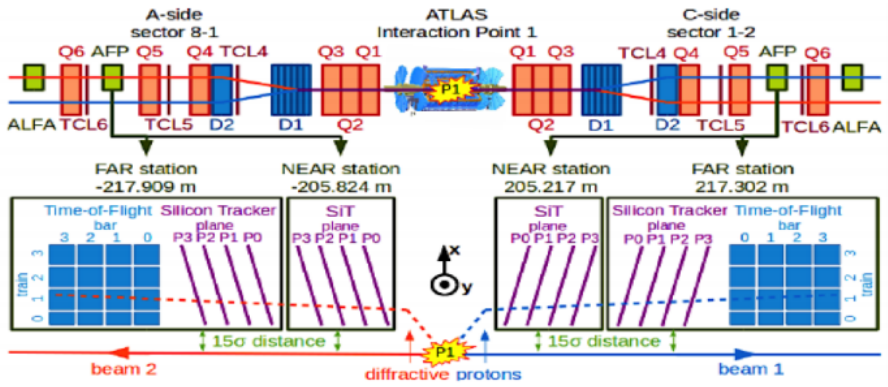


Figure 2: The layout of the beam line (seen from above) between the interaction point (IP1) and the region where the RPs are located in LHC sector 56. Dipoles (D1, D2), quadrupoles (Q1-Q6), collimators (TCL5-TCL6) are shown. The 205 NEAR and 217 FAR units are indicated, along with the timing RPs and the distance of about 15σ from the beam.

Figure 1.2: Experimental set-up with description. [2]

1.4 Current criteria for signal/background separation

The solution of the quadratic equation 1.2 is:

$$\xi_{Reco} = \frac{-119 \pm 656x}{328} \tag{1.3}$$

We consider only solution with + sign in the formula, because the other solution gives unphysical values. This value is called ξ_{Reco}. It equals to an expected momentum loss of protons based on the X position.

For the separation of signal/background following criteria are used [2]:

- $m < 70$ GeV and $m > 105$ Gev
- $0.02 < \xi_{\mu\mu} < 0.1$
- $(\xi_{\mu\mu} - \xi_{Reco})/\xi_{\mu\mu} < 0.1$
- one track at NEAR station and one track at FAR station with close X positions

The reason for the first condition is that mass in the interval $(70, 105)$ is assossiated with z boson interactions. The second condition is set because protons with $\xi_{\mu\mu}$ out of this interval are also out of the AFP acceptance - they do not hit the sensor plane. The third condition says that recorded momentum loss and expected momentum loss must be close together. The last condition is used to be clear that tracks come only from one particle.

The above criteria are very strict. They don't consider any other samples with close ξ values, even though there are still many samples which do not fulfill the second formula but are still highly probable to be a signal. The current solution does not examine other variables which could be a good predictor of being a signal - like specific X positions or specific muon weights for signal/background. To improve the selection and find more candidates the machine learning algorithm has been developed.

Chapter 2

Machine learning

2.1 What is machnine learning

The main purpose of machine learning methods is to find relations in collected data with known output based on which the method is able to predict, with certain probability, output for some new or non-classified data. With "output" is usually meant a belonging to some class or some real number. Task in which we try to assign data a corresponding class are called classification problem (shortly classification), task where we need to assign data a real value is called regression problem (shortly regression). As a typical classification problem could be predicting whether person has/has not have a specific illness, based on the reported symptoms by pacient, measured temperature etc. As a typical example of regression is predicting the price of a house based on the location it is situated, accomodation space, number of rooms etc.

To be able to do so, we typicaly need to collect a lot of data with known output. Then we split the data into training and testing set. On the training set we train the method. Then we let the method to predict classes (for classification) or real numbers (for regression) on the testing set. Because we know what the samples in testing set do equal to, we are able to calculate how succesful our method is. For classification we usually just calculate the percentage of correctly assigned samples to total number of samples. For regression we usually use MSE.

Sometimes it happens that a method has a perfect score on a training set but poor score on testing set. Reason for this is that the separation rules of the method are too complex to best fit the train set. This problem is called overfitting and there are many techniques how to reduce it, e.g cross validation, regularization, etc.

For some methods we also create a validation set on which we find the best hyperparameters of the method for current problem. Each method has its own hyperparameters. For neural nets it's typically number of neurons, for logistic regression regulatization type, for decision trees it's the depth of the tree, etc.

In our analysis we have only a small dataset we can work with. Since we have only 69 signal samples, it is not statistically possible to say which method is definitely the best. That's why I am also measuring average diff value for given probability interval in next sections as another criteria for finding the best method.

2.2 Neural networks

Neural network is a machine learning technique inspired by human brain. They work well for both regresion and classification. Its main unit is a perceptron which you can see on the picture below.

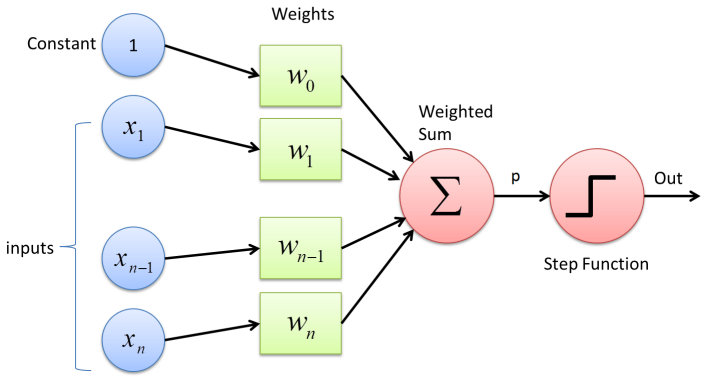


Figure 2.1: Model of a perceptron. [3]

As you can see, perceptron accepts several inputs and gives one output. It also has its own weight **w** vector with one weight for one input. There is also a special weight w_0 which is called intercept and has a constant input $x_0 = 1$.

When the neuron accepts an input from the data(one row of the dataset), we multiply all the inputs with its corresponding weight and sum them up according the following formula:

$$p = \sum_{i=0}^{n} w_i x_i \tag{2.1}$$

Then the output p goes into an activation function from which we get the output of a neuron. An example of activation function is a sigmoid function which is defined as:

$$\sigma(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1} \tag{2.2}$$

Graph of sigmoid function: We can also use hyperbolic tangent *tanh* or iden-

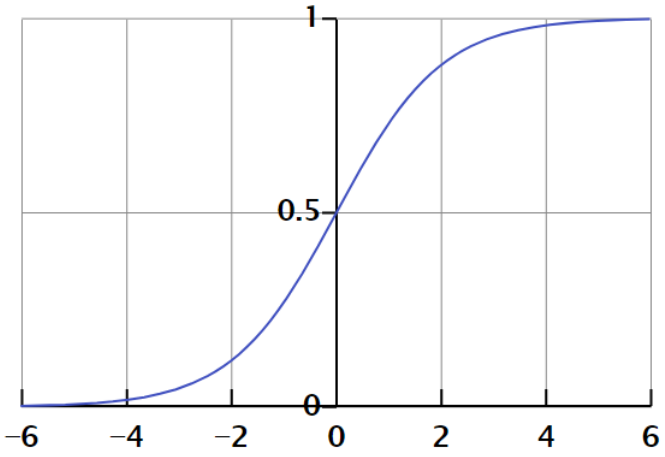


Figure 2.2: Graph of sigmoid function. [4]

tity function. Identity function is also used as output function for regression problems.

2.2.1 What one neuron actually do

Our purpose is to find the best weights for which we get the best score on training set. Because the dataset with n-columns makes up n-dimensional data space, we are in fact looking for the best coefficients of hyperplane which separates the dataspace most effectively. Below is shown an example of such a separation for 12 datapoints in two-dimensional dataspace.

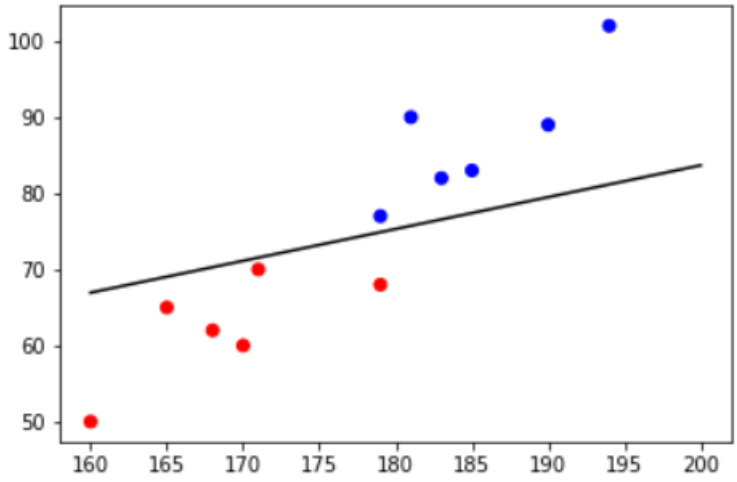


Figure 2.3: Lineary separable problem.

A problem is when data are placed in the dataspace followingly:

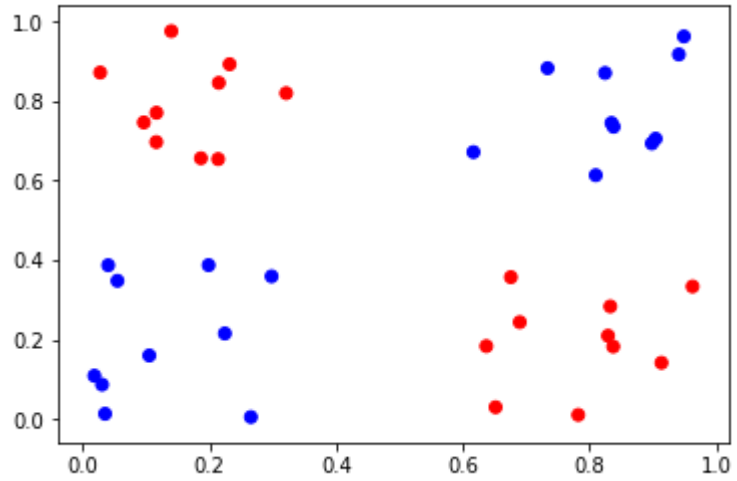


Figure 2.4: Lineary non-separable problem.

Because we cannot separate these datapoints effectively only with one line we call them lineary inseparable. Method which can perform only linear separation are called linear separators. Beside neuron we can logistic regression [6] or SVM (Support Vector Machine) [6].

2.2.2 Learning one neuron

As mentioned previously, our goal is to find the best weights for which the output vector for all the samples z have the least bias from the real output d. Mathematicaly we want to minimize formula which is called cost function. One of the most common loss functions is MSE (mean squared error) divided by number of samples [5]:

$$P(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}(d-z)^2 \tag{2.3}$$

To do this we need to find a gradient for weight vector [5]:

$$\nabla_{\mathbf{w}} P = (\frac{\partial P}{\partial w_1}, ...\frac{\partial P}{\partial w_n}) \tag{2.4}$$

When the gradient is calculated, we can use a learning technique called gradient descent. Gradient descent is a method where we calculate an average error through a set of samples, calculate gradient for weight vector and "make step" in the direction of the gradient. Making step means we update the weight vector with gradient multiplied by some typicaly small number η which is called learning rate. Below is the mathematical notation of gradient descent [5]:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta\nabla_{\mathbf{w}} P \tag{2.5}$$

When we run this iteration many times we get to a local minima. Which is really not so good because we need to find a global minima. Thankfully, for neural nets with high number of neurons all the local minimas are at similar level as global minima, so for the same number of neurons we should get similar results regardless the local minimum we get stuck in.

2.2.3 Multilayer perceptron

Neural net with mupltiple neurons is called *multilayer perceptron*(MLP). MLP always have input layer which just accepts the input from dataset, one or more hidden layers and output layer. Below is shown a simple 3-layer MLP:

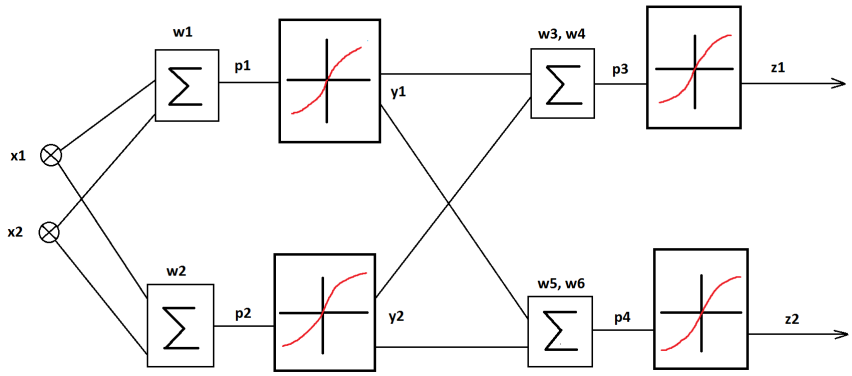


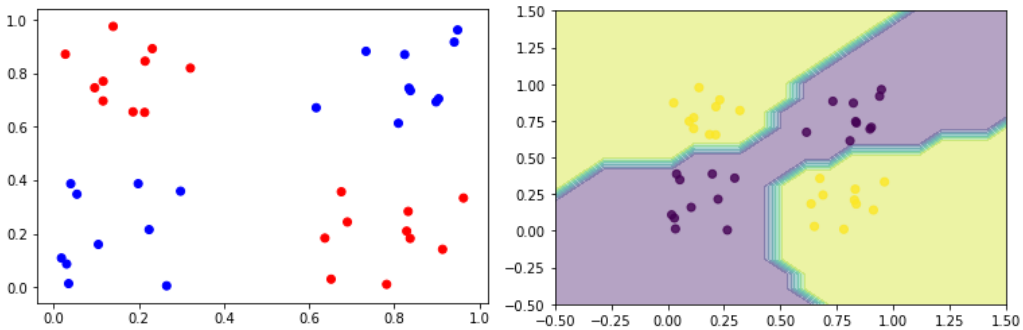
Figure 2.5: Example of 3-layer neural net.

All the outputs from previous layer work as input for next layer. For getting an output of MLP we need the input go through all layers. This is called *forward pass*.

2.2.4 One neuron vs MLP

The reason why neurons are put into networks comes from universal approximation theorem which says that with a MLP with a single hidden layer we are able to approximate any continuous functions on compact subsets of R^n.

Below is shown how MLP with two hidden layers with 5 neurons in each one deals with a lineary non-separable problems:



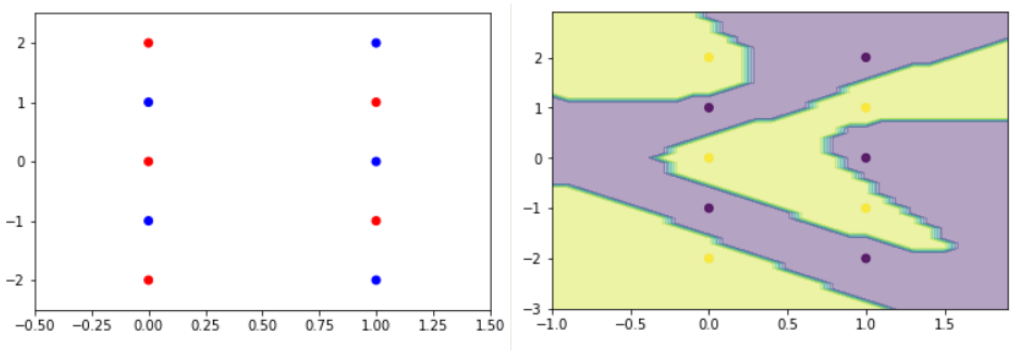


Figure 2.6: Decision borders of MLP with 5 neuron in two hidden layers.

2.2.5 Backpropagation

Training a neural network is called backpropagation. The principle is the same as for one neuron - we update the weights based on the error we get from training of samples. Only the formula is different for each layer.

For output layer we calculate the gradient followingly by chain rule:

$$\frac{\partial P}{\partial w_i} = \frac{\partial P}{\partial z}\frac{\partial z}{\partial p}\frac{\partial p}{\partial w_i} \tag{2.6}$$

If we are using MSE as loss function previous formula equals to:

$$\frac{\partial P}{\partial w_i} = -2(d-z)\sigma'(p)y_i \tag{2.7}$$

Now if we want to calculate a component i of gradient of first neuron in hidden layer from Fig.2.5 we calculate following expression:

$$\frac{\partial P}{\partial w_i} = \frac{\partial P}{\partial z_1}\frac{\partial z_1}{\partial p_3}\frac{\partial p_3}{\partial y_1}\frac{\partial y_1}{\partial p_1}\frac{\partial p_1}{\partial w_i} + \frac{\partial P}{\partial z_2}\frac{\partial z_2}{\partial p_4}\frac{\partial p_4}{\partial y_1}\frac{\partial y_1}{\partial p_1}\frac{\partial p_1}{\partial w_i} \tag{2.8}$$

$$\frac{\partial P}{\partial w_i} = \frac{\partial y_1}{\partial p_1}\frac{\partial p_1}{\partial w_i}\left(\frac{\partial P}{\partial z_1}\frac{\partial z_1}{\partial p_3}\frac{\partial p_3}{\partial y_1} + \frac{\partial P}{\partial z_2}\frac{\partial z_2}{\partial p_4}\frac{\partial p_4}{\partial y_1}\right) \tag{2.9}$$

General formula for gradient component of neuron in layer before the output is(O is the set of output neurons):

$$\frac{\partial P}{\partial w_i} = x_i\sigma'(p)\sum_{j\in O} -2(d_j - z_j)\sigma'(p_i)w_i \tag{2.10}$$

We can see that a part of the expression 2.7 is already calculated in expression 2.10. This can reduce our computational time for backpropagation. For this reason we prefer to use neural networks with more hiden layers instead of a neural network with many neurons in one hidden layer.

Chapter 3

Realisation

3.1 Used tools

3.1.1 Jupyter notebook and Python

Jupyter notebook [7] is very convenient tool for data analysis. It supports most the languages which are used for this purpose - like Julia, Python, R and much more(its name is also derived from these three languages).

An advantage of these notebooks is that we create blocks of codes which can be executed independently so we have all of our scripts on one page where we can also compare the outputs. We can also add blocks of text where we can describe what the following code does.

Work with Jupyter notebook takes place in internet browser from which we access them using URL link. Therefore, before using the notebook we need start the notebook server and then access from the browser.

From this comes another advantage of the notebook - we can easily convert the output into HTML page. I am also using this utility to create a webpage output of my thesis.

In the Jupyter notebook I am working with latest Python version. For analysis I am using the following Python libraries:

- pandas [8] - library for working with datasets
- numpy [9] - library for mathematical operations
- matplotlib [10] - library for making plots
- scikit learn [11] - library with machine learning methods

3.1.2 ROOT

ROOT[12] is a framework developed by IT staff working for CERN. It provides tools big data processing, statistical analysis, visualisation and storage. It is mainly written in C++ but integrated with other languages such as Python and R.

Even though ROOT provides a lot of useful tools, I decided to work in Python and its libraries because in my opinion it provides more dynamic work and the functionality is the same.

I still learned the ROOT commands, because all the current results done with root usage. Also the dataset is saved in special ROOT file so I needed to convert it to .csv file.

Even though I have been using for most of the parts of the thesis, I have learnt to work with ROOT in C++, also because the source codes, I was provided with, are written in C++.

3.2 Getting data from .root file

Data in ROOT are usually stored in data structure called tree which class is called `TTree`. Every tree has a list of data structure called branch which is an equivalent of a column in dataset.

For writing data into a tree, we need to create a branche and assign a local variable for them. With `TTree::Branch(const char *bname, void *add)` we create a new branch called bname loading data by variable pointed by address. Calling `TTree::Fill()` we create a new record with values stored in local variables.

Similarly for reading data from a tree we first call `TTree::SetBranchAddress(const char *bname, void *add)` and then with `TTree::GetEntry(i)` we load the value at index `i` into a variable pointed by `add`.

Trees can be saved into a `.root` file. An advantage of a `.root` files is that we can open it with ROOTs browser (which is opened by calling `new TBrowser` from ROOT terminal) and see the trees inside the file and their branches. Another advantage of `.root` files is that when we open it in the ROOT browser, they are automatically converted into histograms. A disadvantage is that only the root framework can work with this format.

For better work with tree there is a method call `TTree::Loop()` which creates a class for the tree. Creating an instance of this class loads the corresponding

tree and assigns branches local variables, which has same name as the branch. Class provides `TreeClass::Loop` which loops over the whole tree. Here we can put our own code and do what we wish to do with each record. So I loaded each row into a simple .csv file, which just contained the values separated by `';'` character.

3.3 Dataset

3.3.1 Meaning of each column

The dataset we are working with looks like this:

stationID	xLocal	yLocal	xSlope	ySlope	xiL	xiR	xiAFPx	xiAFPy	xiRecoAFP
3	-2.73788	1.694140	0.003956	-0.013791	0.020798	0.000923	0.021586	0.053883	0.022321
0	-5.13777	-14.000000	-0.000000	-0.000000	0.026487	0.001865	0.037585	-1.000000	0.040872
0	-10.68450	-4.280000	-0.000000	-0.000000	0.031497	0.000848	0.077897	-1.000000	0.080791
1	-8.68138	-0.431136	-0.000919	-0.011109	0.021971	0.000363	0.064542	0.011377	0.066803
0	-4.37762	0.720000	-0.000000	-0.000000	0.000303	0.023532	0.032517	0.034400	0.035090

n_tracks	nProtons	n_hits	n_clusters	m	aco	event_n	run_n	side
1	0	5	3	56.9576	0.001728	1312362004	1218593248	1
1	1	2	2	91.3627	0.001823	1326907476	1218528960	0
3	4	9	6	67.1976	0.009885	1310471577	1218787616	0
2	2	15	11	36.6876	0.008901	1319666649	1218790912	0
4	0	20	13	34.7084	0.007758	1326135196	1218559200	0

Figure 3.1: Dataset we work with.

For better understanding of the dataset I will distinguish three different column types based on how were the data were obtained. The first type is the columns which represent data collected at the ATLAS detector, second one is the data collected at one single AFP station and the third is the column which was created using the informations from two different AFP stations.

3.3.1.1 Data collected in ATLAS

- m - mass of the muon pair detected at ATLAS which comes from elastic interaction - our signal. Our purpose will be to find a matching proton/s by which these muons were created.

- *aco* - acoplanarity, angle between two crossing beam axes at interaction point for given *event_n*.

- *xiL*, *xiR* - momentum loss of a proton going to the left/right side of AFP. This value is calculated from muons caught in ATLAS using positive/negative η value.

- *run_n*, *event_n* - identification numbers of an experiments. Relation between these identifiers is that for one *run_n* many *event_n*s are performed. *event_n* can be seen as identification of two specific bunches of protons which collide at IP. *run_n* is the identification number of the whole experiment - e.g. identification of all bunches of protons accelerated in the same day.

3.3.1.2 Data collected at one AFP station

- *stationID* - station identificator in range from 0-3. Station 0 and 1 are NEAR and FAR stations on left side, stations 2 and 3 are NEAR and FAR stations on the right side.

- *side* - side where recordings were made. LEFT side corresponds to 0, RIGHT side corresponds to 1. With previous knowledge at which side is each station located, this column is redundant.

- *xLocal*, *yLocal* - coordinates of a track at given station.

- *xSlope*, *ySlope* - these variables are associated with track-create process from the four planes at one station. The slope is with respect to the initial proton beam.

- *xiAFPx*, *xiAFPy* - columns replaced by xRecoAFP column, approximate reconstructed xi value in X and Y direction

- *xiRecoAFP* - momentum loss of a particle calculated from *xLocal* coordinate at given station of AFP

- *n_tracks* - number of reconstructed particle trajectories which hit the AFP station. Reconstruction of a particle is done by combining recorded electro charges from all 4 plates at one station. This process is called tracking, this is where does the name of the column come from. Normally, the number of tracks corresponds to number of rows fow given station. In some cases there is less rows then number of tracks. It is because sometimes there is not enough informations for full track reconstruction, e. g. when a particle hits the border of a detector and doesn't leave and electric charge on all 4 plates.

- *n_hits* - number of activated pixels at AFP station.

- *n_clusters* - when two hits are next to each other, they are probably originating from single and are merged into clusters.

3.3.1.3 Data created using the informations from NEAR and FAR station

- *nProtons* - number of reconstructed particles using the informations from NEAR and FAR stations at the same side. Reconstruction is done by comparing coordinates of a tracks at NEAR and FAR station. When two coordinates are close together they are believed to originate from one particle. *nProtons* also includes all combinations of possible track trajectory. For example, if we have one track at NEAR station with $xLocal = -5$ and then two tracks at FAR station with $xLocal_1 = -5.5$ and $xLocal_2 = -4.5$, *nProtons* equals to 2.

It is also important to realize that the only variable which change with each row are *xLocal*, *yLocal*, *xSlope*, *ySlope*, *xiAFPx*, *xiAFPy* and *xiRecoAFP*. All the other columns are the same for one station. *n_tracks*, *n_hits*, *n_clusters* change only with *stationID*. *xiL*, *xiR* and *nProtons* change with *side*. And finally *m* and *aco* change only for different *event_n*. For better understanding of the dataset, in the table below are rows for one *event_n*:

	stationID	xLocal	yLocal	xSlope	ySlope	xiL	xiR	xiAFPx	xiAFPy	xiRecoAFP
0		-10.19940	-1.530000	-0.000000	-0.000000	0.020721	0.000368	0.074663	-1.000000	0.077444
1		-10.59840	-0.425828	-0.001111	-0.011108	0.020721	0.000368	0.077322	0.011483	0.080198
2		-4.24229	1.496070	0.010020	-0.004906	0.020721	0.000368	0.031615	0.049921	0.034052
3		-4.29630	1.242970	0.002908	-0.022540	0.020721	0.000368	0.031975	0.044859	0.034466
3		-2.88076	1.694710	0.004161	-0.013814	0.020721	0.000368	0.022538	0.053894	0.023450

	n_tracks	nProtons	n_hits	n_clusters	m	aco	event_n	run_n	side
	1	1	2	2	35.9031	0.009897	1318703235	1218579072	0
	1	1	6	4	35.9031	0.009897	1318703235	1218579072	0
	1	2	6	4	35.9031	0.009897	1318703235	1218579072	1
	2	2	12	9	35.9031	0.009897	1318703235	1218579072	1
	2	2	12	9	35.9031	0.009897	1318703235	1218579072	1

Figure 3.2: Records for one *event_n*.

3.4 First adjustments

As a first step I looked for a columns which do not bring any new informations for our analysis.

At first I have discarded *xiAFPx* and *xiAPFy*. Next I discarded *ySlope* because it also contained a lot if undefined values. I also discarded *run_n* column since it is just an identification number of the experiment.

I also replaced *xiL* and *xiR* columns with *xiMM* and *xiMMotherDir* columns to make it clear with which ξ value we are working with. *xiMM* is the momentum loss calculated from muon associated with the corresponding side, *xiMMotherDir* is the momentum loss calculated from a muon corresponding to the other side. For $side = 0$, *xiMM* equals to *xiL*, for $side = 1$ *xiMM* equals to *xiR*.

Next, I decided to work only with rows from NEAR station so I created a new columns *n_trackFar*, *n_clustersFar* and *n_hitsFar* to have all the necessary informations from both stations in one row. Rows from FAR station were dropped.

Finally, I have dropped the *stationID* column and kept only side column. With merged data from NEAR and FAR station into one row, this column is redundant.

3.5 Signal/background definition

For getting a signal I created a new *diff* variable, defined as:

$$diff = \frac{xiMM - xiRecoAFP}{xiMM} \tag{3.1}$$

This calculates how different is the calculated ξ from muons and calculated ξ value from a track.

Now for getting signal samples I put following criteria:

1. $m < 70$ GeV or $m > 105$ GeV
2. $diff < 0.1$

For background I have used similar criteria, only with $diff > 1.0$.

After applying these rules I got 69 signal samples and and 398 background samples. Samples with $diff \in (0.1, 1.0)$ weren't used for training, because may contain some signal candidates and our purpose is to find with our method. Number of samples in this set was 158.

3.6 Creating new columns

With this cleaner dataset I have created as many columns as possible which could carry some informations specific for signal/background. I focused on comparing the records at NEAR and FAR stations created columns, for which the values for signal/background distributions might be different. If there were no records at FAR station, value was set to -1.

- *xDiff*, *yDiff* - distance from the nearest track at FAR in X/Y direction. For getting the nearest track at FAR station I was comparing only *xLocal*s of the tracks. For example, for a track with coordinates (-4.0, 0.0) at NEAR station with 2 tracks at FAR station with coordinates (-5, 3.0), (-2.5, 1.0), the *xDiff* and *yDiff* values will be 1.0 and 3.0

  Reason for creating these columns was that background particles might travel differently and have different X/Y bias than signal particle.

- *tracksInc* - increase/decrease of the *n_tracks*, *n_clusters* and *n_hits* at FAR station by formula $n\_tracks_{FAR}/n\_tracks_{NEAR}$ (for tracks). The motivation for creating these columns was that we don't know exact behaviour of the particles at NEAR/FAR station. For example, if background particles had higher tendency to dissolve while traveling between stations, there should be higher *tracksInc* ratio for background than for signal.

- *hitsMean*, *clustersMean* - average number of hits and clusters per track at NEAR station calculated by formula $n_hits_{NEAR}/n_tracks_{NEAR}$(for hits). Motivation for creation of these was similar as before - if background particles had higher/lower dissolving character than signal particles, it should be reflected in these columns.

- *hitsMeanFar*, *clustersMeanFar* - same value as before, just for FAR station.

- *recoOther* - this value says if there is at least one record at FAR station. If there is, this value is 1, otherwise 0.

3.7 Dropping diff column

In the following trainings, I always drop the *diff* column. The reason is that since we classified our sample based on the values from this column, the method would assign a biggest weight to this variable and make classification only using this value. Therefore we don't find any new candidates for signal/background, we just get those we already have.

When I ran the method on train data with this columns and test it on test

data, classification succes rate was almost 100%. Running the method on the rest of the dataset gave me only exactly the same signal samples I already had and didn't find any new one - as expected.

3.8 Using logistic regression

At first I have tried to use linear separation. I have used `SGDClassificator` [13] with no regularisation and `perceptron` as loss function. I also used Logistic regression [14] with `lbfgs` solver and L2 regularisation [16].

For 69 signal samples I always randomly picked 69 background samples and merged them into one set. Then I randomly shuffled the dataset and created training and testing set in 1:1 ratio.

I ran this iteration 10 times and saved the accuracy rate on test data. The average accuracy rate was 84%, with 72% minimum and 96% maximum accuracy rate. Based on theses rates we can say, that signal and background are quite well lineary-separatable.

3.9 Using neural networks

Even though linear separation worked quite well, I also tried how good is the neural network for separation. I've used 3 and 4 layer perceptron with *tanh* activation function and stochastic gradient descend as a solver. I did not use validation set to find the real accuracy, I just wanted to try if neural network can work better than logistic regression. I prepared the training and testing samples the same way as I did with the logistic regression.

For 3-layer percepton I was putting 1-20 neurons in the hidden layer, with accuracy test in each iteration. For 4-layer neural network I put 1-5 neurons in each layer and also testing the accuracy. I ran this for 5 randomly picked background samples.

The best accuracy I got was 75% with 2 hidden layers with 5 neurons in the first one and 3 neurons in the second one which means linear separation works better for our problem. This is good for our analysis, since we would have small data for creating a validation layer for finding the hyperparameters.

3.10 Standartization of the dataset

Knowing that the data are lineary separatable, we can take advantage of logistic regression method - getting the columns with largest weights from the

learned model. To be able to do that, we need all the columns to be comparable - having the same units. Since all of our variables are on very different scale, for example *xLocal* having the values in the interval (-14,-2), while *xiMM* in the interval (0.06, 0.0002), changes of the *xiMM* will be much more weighed than changes of *xLocal*.

To make the columns comparable, we use the standartization, which makes all the columns having the same mean=0 and same variance=1. To standartize the dataset, we use the following formula [16] on every element of the dataset, (μ - mean value of the column, σ - standart devation of the column):

$$z = \frac{x - \mu}{\sigma} \tag{3.2}$$

After this, we can run the method again and see the weights.

After running the method on dataset with standartized values, success rate on the test sample increased to almost 100% for any background sample.

Here are the the examples of weights from three different methods learned on three different random background samples (*sign* is the +/- sign of the weight, *coef* is the absolute value of the weight):

Logistic reg. score: 0.971

	coef	sign
xiMM	1.621987	+
xiRecoAFP	1.193923	-
xLocal	1.117598	+
xiMMotherDir	0.580969	-
aco	0.460296	-
n_tracks	0.357848	-
xSlope	0.309636	-
nProtons	0.285599	+
n_tracksFar	0.261648	-
tracksInc	0.247290	-
yDiff	0.230926	+
xDiff	0.227117	-

Logistic reg. score: 1.0

	coef	sign
xiMM	1.621528	+
xiRecoAFP	1.041825	-
xiMMotherDir	1.037589	-
xLocal	1.000644	+
clustersMeanFar	0.424798	-
yLocal	0.361333	+
nProtons	0.269791	-
hitsInc	0.266286	+
n_tracksFar	0.238673	-
n_tracks	0.192473	-
hitsMean	0.189410	-
aco	0.178224	-

Logistic reg. score: 0.927

	coef	sign
xiMM	1.760914	+
xiRecoAFP	1.212496	-
xLocal	1.151942	+
xiMMotherDir	0.853707	-
nProtons	0.385262	+
tracksInc	0.372696	-
n_tracks	0.322053	-
m	0.258026	+
xSlope	0.241459	-
n_tracksFar	0.238164	-
n_hitsFar	0.237476	+
n_clustersFar	0.222284	+

Figure 3.3: Table of biggest weights for logistic regression.

As we can see, in all 4 cases *xiMM*, *xiRecoAFP*, *xLocal*, *xiMMotherDir* are the columns with the highest influence on signal/background separation. When we look at the histograms on Fig. 3.4 of these columns, we can see that the distributions of these values are really very different for signal/background:

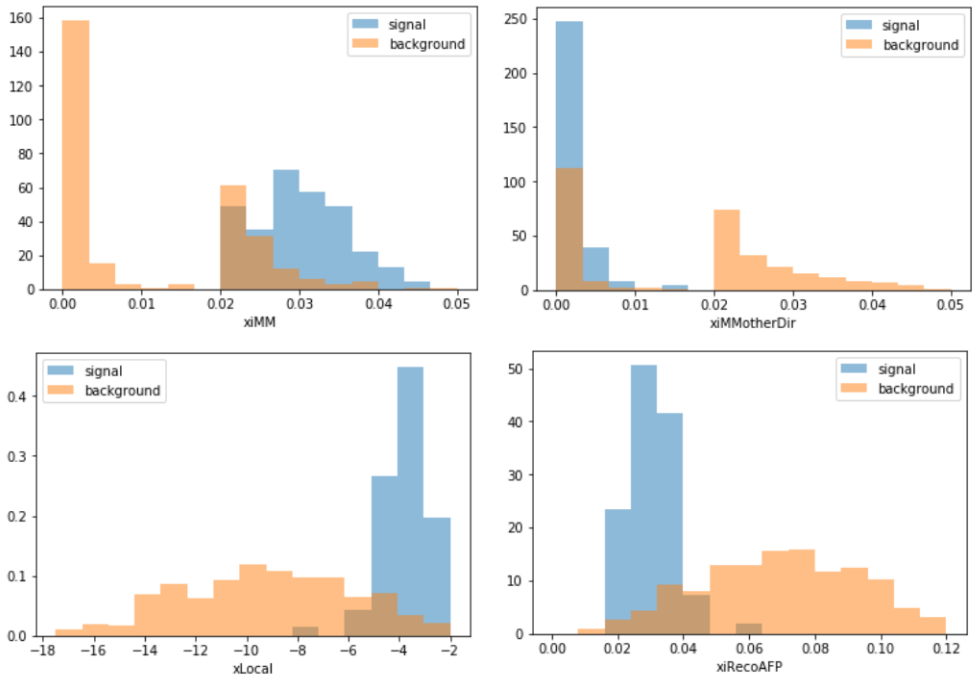


Figure 3.4: Histograms of columns with biggest weights.

It is not surprising that all these 4 concrete variables have specific distributions for signal/background.

As we can see from the first two histograms, signal is probably associated with specific momentum losses calculated from muons. Because X positions are almost lineary dependent on this momentum loss, then also X positions have specific distributions for signal/background. Now *xiRecoAFP* is also almost lineary dependant on *xLocal*, therefore also has specific signal/background distribution.

Quite misleading is the high weight for *xiMMotherDir* column. Because it is the momentum of loss the muon associated with the other side, it should not be a predictor for signal/background on the current side. This is why I decided to drop this column. The performance of logistic regression was not affected by this adjustment.

3.11 Applying method on the whole dataset

Now when we have the trained method, we can use it on the whole dataset and see how the signal/background candidates stand with real signal/background criteria - the *diff* column.

To find the best method parameters, I was randomly picking background candidates and with each call constructed a table below. By randomly picking background samples and checking the correlation with expected values, I wanted to find the best hyperplane position which separates the dataspace most efficiently and give us the most reliable signal candidates.

To do this I have created following table for each background sample. *min*, *max*, *mean* and *std* columns stand for minimum/maximum value of *diff* in the interval and for mean and standart deviation of the *diff* values in the interval. Because *diff* column values spread over a large scale, starting with 0.002 and ends with 429, I also used *mean_log* which stands for mean of *diff* values after aplying natural logarithm on them. This showed up to be better approach than a standart mean, because with one wrongly assigned sample with big *diff* value, the mean of the interval changed significantly.

For each random background sample I also calculated the correlation coeficient between *mean_log* and the starting left border of the interval(for first row it was 0.95, for second 0.90 etc.). The reason for this was that with a lower signal probability we expect the *diff* value to have higher values. Therefore the average *diff* value should grow with lower probability interval.

After training 10 methods with different background samples, here is the output of the best one I got. The correlation coefficient was -0.97.

	min	max	mean	mean_log	std	count	y
95-100%	0.001991	0.918133	0.158213	-2.626544	0.203543	90	1.00
90-95%	0.005646	0.860778	0.262786	-1.785851	0.220528	45	0.95
80-90%	0.052919	1.393980	0.632515	-0.661290	0.328540	19	0.85
70-80%	0.576360	25.131184	3.545589	0.186466	8.096980	9	0.75
60-70%	0.029546	28.502351	3.020203	-0.113174	7.668550	13	0.65
50-60%	0.428874	21.457464	2.948517	0.411907	5.842079	12	0.55
40-50%	1.028242	87.968289	10.566250	1.036473	25.906291	11	0.45
30-40%	0.521904	103.469180	20.168860	1.488548	32.882882	17	0.35
20-30%	0.644221	203.330850	32.680882	2.361865	47.027395	21	0.25
10-20%	1.706815	152.077811	33.869166	2.451279	42.785414	35	0.15
0-10%	1.069185	428.923794	99.224498	3.684771	95.857999	177	0.05

Figure 3.5: Table with basic statistics for *diff* column for each probability interval.

Here you can see the histogram of probabilities after applying the method on the whole dataset:

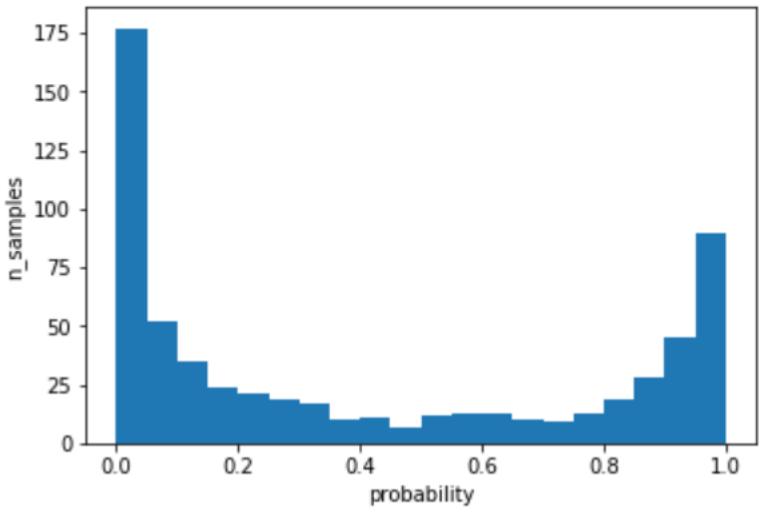


Figure 3.6: Histogram with *n_samples* for 5% probability intervals.

Most of the samples lie in the (0.0, 0.05) and (0.95, 1.0) intervals. This is because of the sigmoid output function in logistic regression (which is also in neural network). Because the linear region of sigmoid function is pretty short, most of these values are close to 1 or 0. They are still comparable – values with higher distance from plane have higher probability rate, we just aren't able to effectively say which sample is far away and which sample very far away.

3.12 Combining probabilities

With the new method we have a new criteria for separation which uses informations from the whole dataset, except for *diff*. Because the *diff* column is another important predictor for signal/background, we should somehow combine the probabilities from both methods to filter out badly classified samples and to make the final probability output more reliable.

For this purpose, we need to create a function which assigns a probability for each *diff* value. The issue is that all we know is that in the (0.0, 0.1) interval the probability is high. Thus all we can do is to make estimations.

We can expect that with a higher *diff* value the probability for signal exponencially decreases. With this knowledge we can use a formula for exponencial distribution and fit it to some estimated points. The formula is following (l and s are the parameters we need to find):

$$p(diff) = \frac{le^{-l\,diff}}{s} \tag{3.3}$$

I don't use an exact formula for exponencial distribution because I divide the values with s for better fitting with the points.

The estimated probability for *diff*=0.1 is 0.85, for *diff*=0.5 it is 0.4. For finding the best parameters I was looking for the least MSE error for s $\in (0, 10)$ and $l \in (0, 10)$. Best fitting parameters were s=1.81 and l=1.82. Below the estimated probability distribution for *diff* is given.

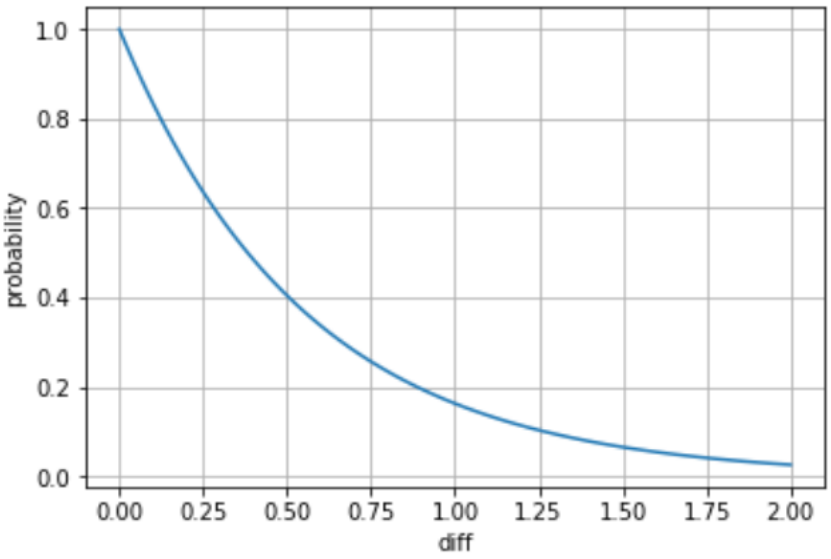


Figure 3.7: Estimated probability calculated from *diff*.

Now with two probability outputs – one from method and one from formula I have used – we need to combine these two probabilities to get the final probability output(p_m is the probability from the method, p_e probability from estimation based on *diff*):

$$p = (p_e + p_m)/2 \tag{3.4}$$

This is the distribution of final probability and the previous table with new values:

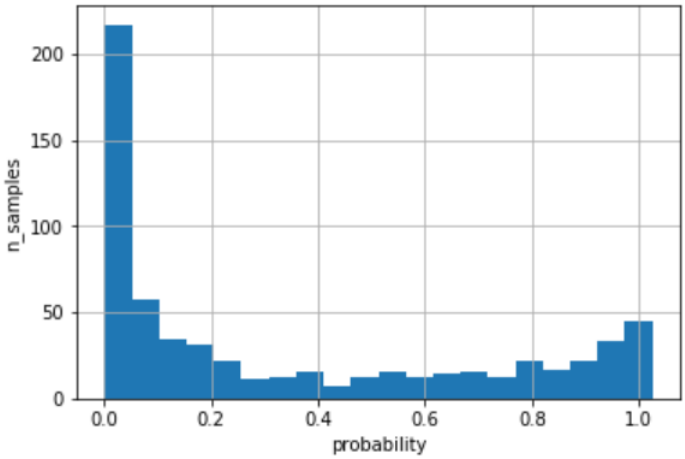


Figure 3.8: Table with basic statistics for final method *diff* column for each probability interval.

	min	max	mean	std	count
95-100%	0.005646	0.097253	0.050057	0.022669	38
90-95%	0.049302	0.141005	0.102511	0.023235	28
80-90%	0.029546	0.377532	0.244231	0.087757	15
70-80%	0.154218	0.576636	0.426514	0.122942	12
60-70%	0.341666	0.918133	0.653605	0.188064	18
50-60%	0.428874	0.961908	0.847196	0.150328	12
40-50%	0.521904	1.232732	0.974315	0.198543	10
30-40%	0.644221	28.502351	3.398505	7.553890	13
20-30%	0.940854	87.968289	10.065595	20.378990	21
10-20%	1.069185	203.330850	27.981811	44.038248	36
0-10%	1.680770	428.923794	92.180251	91.262427	217

Figure 3.9: Table with basic statistics for final method *diff* column for each probability interval.

I was also requested to make a plot with probability for each row so it will be easier to cut out the samples with lower/higher probability at given point. To do that I sorted all the rows based on the probability output. Then I changed the index of each row, so the first row has the lowest probability and last row has the highest probability. Below you can see the graph of y with dependency on row index:

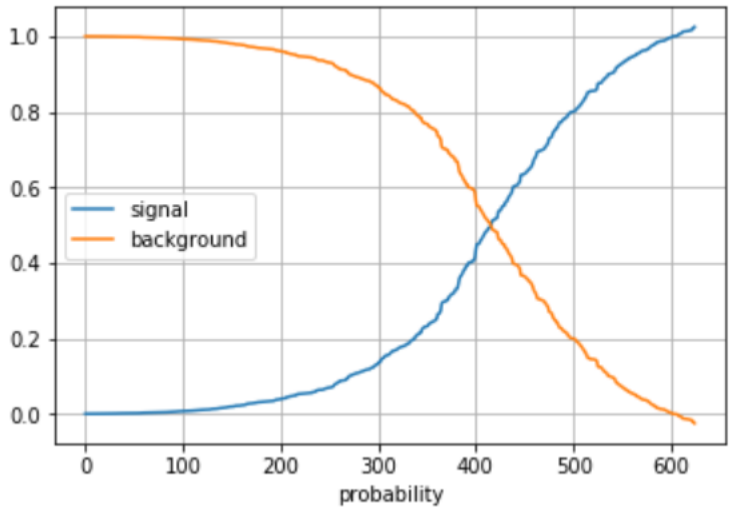


Figure 3.10: Probabilities for each sample.

3.13 Final output

As a webpage output I have exported the notebook I have worked with into a HTML format so it can be published on the internet. Before this I cut off the Python source codes and put them into a single script. I also added a tags for downloading the thesis and the script.

The script I have created just simply accepts the percantage at which we want to perform a cut. Then it returns a .csv file with samples having the probability higher/lower than requested probability.

Conclusion

The purpose of the thesis was to extend the criteria for getting signal/background using the neural network. After explaining how neural network works and for which tasks it is suitable for we tried how can neural network deal with our data.

We found out that our data are lineary separable and thus the best fitting neural net was the one with only one neuron. Because of better documentation I decided to work with LogisticRegression which is also a linear separator, only with different method for finding the weights of the model.

We also transformed our current separation rules into a probability estimation. With combination with our method output we assigned each sample a final probability we work with.

We are now able to work with more samples with still high probability for being a signal, which was the goal of this thesis. Having more signal candidates might help to understand better the elastic interactions, which is the main goals of AFP experiment.

Bibliography

[1] Who we are. [online], Available from: https://home.cern/

[2] V. Petousis and A. Sopczak, *Observation of tagged protons in events with central exclusive muon pairs with the ATLAS Forward Proton detectors at 13 TeV, ATL-COM-FWD-2019-004*. Geneva, CERN, 2019.

[3] What the Hell is Perceptron? [online], Available from: https://towardsdatascience.com/what-the-hell-is-perceptron-626217814f53

[4] Logistic function . [online], Available from: https://en.wikipedia.org/wiki/Logistic_function

[5] P.Lopez and D. Vašata, *Neural networks*. URL: https://courses.fit.cvut.cz/BI-VZD/lectures/files/BI-VZD-11-cs-slides.pdf. [Page available only for FIT CTU students - copy of presentation can be found at enclosed CD.]

[6] Generalized Linear Models. [online], Available from: https://scikit-learn.org/stable/modules/linear_model.html

[7] What is the Jupyter Notebook? [online], Available from: https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html

[8] pandas: powerful Python data analysis toolkit. [online], Available from: https://pandas.pydata.org/pandas-docs/stable/

[9] NumPy v1.16 Manual. [online], Available from: https://docs.scipy.org/doc/numpy/

[10] User's Guide. [online], Available from: https://matplotlib.org/users/index.html

[11] Documentation of scikit-learn 0.21.1. [online], Available from: https://scikit-learn.org/stable/documentation.html

[12] ROOT User's guide. [online], Available from: https://root.cern.ch/root/htmldoc/guides/users-guide/ROOTUsersGuide.html

[13] SGDClassifier. [online], Available from: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html#sklearn.linear_model.SGDClassifier

[14] Logistic Regression. [online], Available from: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

[15] MLPClassifier. [online], Available from: https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

[16] D. Vašata, *Ridge Regression*. URL: https://courses.fit.cvut.cz/BI-VZD/lectures/files/BI-VZD-09-cs-slides.pdf. [Page available only for FIT CTU students - copy of presentation can be found at enclosed CD.]

Appendix A

Acronyms

LHC Large Hadron Collider

AFP ATLAS Forward Proton detector

SVM Support-vector machine

MLP Multilayer perceptron

Appendix B

Contents of enclosed CD

- `readme.txt` ..... the file with CD contents description
- `src` ..... the directory of source codes
  - `html` ..... directory with HTML thesis output
  - `script` ..... directory with script and data
  - `thesis` ..... the directory of LaTeX source codes of the thesis
- `text` ..... the thesis text directory
  - `thesis.pdf` ..... the thesis text in PDF format