Master's Thesis

Full Detector Simulation of Pair Monitor at 250 GeV ILC and Application of Machine Learning

Ahmed Mustahid

Department of Physics
Graduate School of Science
Tohoku University
2019

Full Detector Simulation of Pair Monitor at 250 GeV ILC and Application of Machine Learning

Ahmed Mustahid

Abstract

Determination of beam parameters is an essential part of beam monitoring system. Forward Calorimeters in the ILC play the major role for this purpose. In this research, first a simulation of a forward detector namely, pair monitor has been executed by utilizing the already implemented geometry of another forward calorimeter, namely BeamCal. The later part contains the evaluation of sensitivity of different variables due to horizontal and vertical beam sizes. Motivated by an approximate analytical formula of the beam electric field and the evaluation of the sensitivity of the variable to the beam sizes, machine learning has been used to determined the horizontal beam size. Due to the small value of the vertical beam sizes, it has high statistical fluctuations across events and its prediction will require a statistics of at least 10 bunches per one combination of horizontal and vertical beam.

Acknowledgments

I am greatly indebted to my parents who have worked hard for ensuring a good education for me. It would not be possible for me to do what I am doing now, without their sacrifices.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

20-th century high energy physics was revolutionised by means of the introduction of clever detector technologies which helped made new discoveries. In the 21-st century, the requirement of higher energy has made it challenging for experiments to carry out new investigations. In this century, the world is also being changed in a way that happened never before by the advent of artificial intelligence. As far as its application is concerned, high energy physics has immense potential. Artificial intelligence or machine learning has made it possible to predict certain quantities, acts which were considered cumbersome in the past. Human beings rely on a model to predict the patterns created by a certain variable. It is the other way around, when it comes to machine learning; it predicts a model by learning from the patterns. This behaviour can have high implication in the detector R&D and beam physics research. This research is an attempt to show the effectiveness and limitations of machine learning in the context of beam physics.

Linear colliders like International Linear Collider (ILC), require very dense beams at the collision point in order to ensure maximum luminosity in a single pass. Beam monitoring is required to make sure the adequate luminosity can be provided. Accurate determination of beam parameters is essential for successful beam monitoring. This research investigates into the behaviour different parameters with varying values of horizontal and vertical beam sizes. In an earlier research [1], the observables from pair monitor that are sensitive to the values of beam sizes were determined. An attempt to use image recognition technique to determine beam sizes from raw incoherent pair backgrounds (without realisitic magnetic field/detector effect), was employed in [2]. Utilizing the observables, this research shows the potential of machine learning to *predict* the beam sizes.

After a short introduction of the ILC and physics processes relevant to this study, the initial part of this research discusses the simulation procedure under the current scenario. Afterwards, the parameters sensitive to horizontal and vertical beam sizes are discussed and their sensitivity to the beam sizes, are compared. It then suggests what conditions must be satisfied by those parameters so that machine learning techniques can be applied. Finally, a suggestion about what a future research on this topic would look like is discussed.

Chapter 2

International Linear Collider

2.1 Overview of the accelerator

The International Linear Collider (ILC) is an electron-positron collider, proposed to be built near Kitakami at Iwate Prefecture, located in the north east Japan. Having a length of about 31 km, it will have a center-of-mass energy of 250 GeV at the first run. ILC will play a major role in our current understanding of the universe. Especially, it will help us investigate into the puzzling problems in the physics of the 21st century, such as, matter-antimatter asymmetry, existence of dark matter and the properties of the Higgs field. ILC will help achieve this goal by means of its clean environment, high luminosity and beam polarization.

The following sections would describe the chief components of the accelerator system.

2.2 Accelerator Components

The accelerator consists of superconducting RF main Linacs, electron source, positron source, damping rings, ring to main Linac and a beam delivery system.

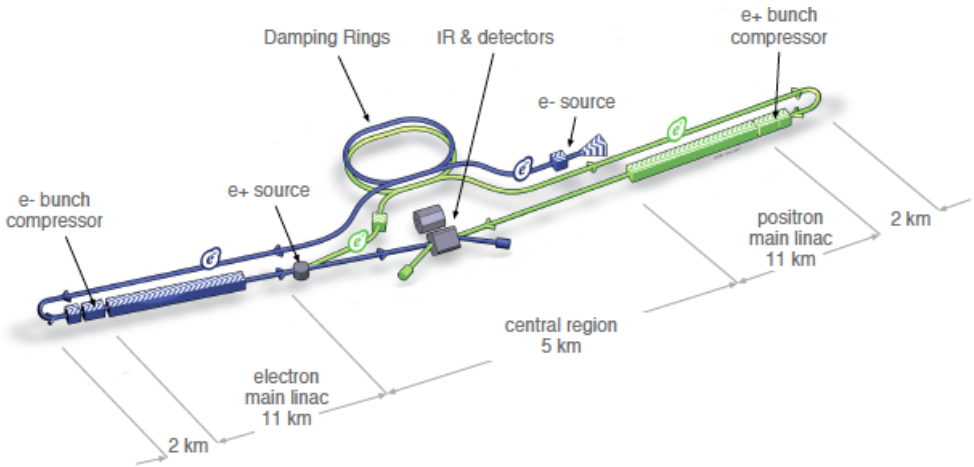


Figure 2.1: A schematic of the ILC [3]

Figure 2.2: Cryomodules to be used at ILC [4]

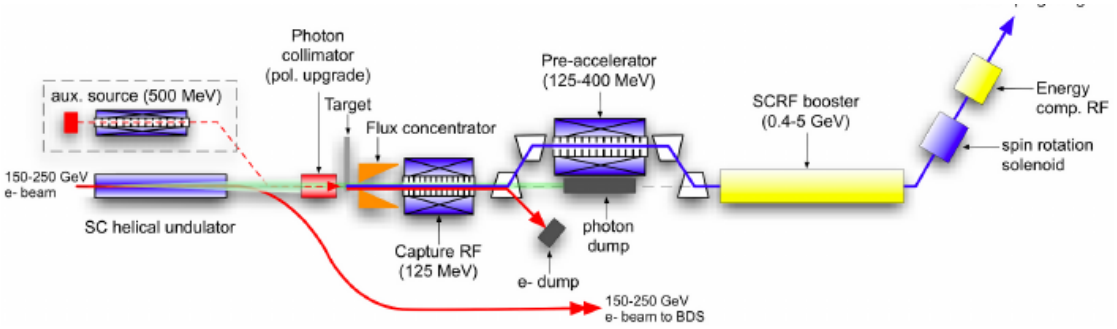


Figure 2.3: Positron source at the ILC [5]

2.2.1 Superconducting RF Main Linacs

After an initial acceleration in upstream bunch compressors to 15 GeV, 7400 1-m long nine-cell niobium cavities accelerate the particles tp 250 GeV.The cavities are operated at about 2K, which is maintained constantly by immersing them into saturated He-II bath. Liquid Helium is supplied from a total 10-12 cryogenic plants. Each plant can cool a continuous length of 2.5 km of Linac. The main linac is made with the curvature of the earth in mind to ensure smooth supply of Helium.

2.2.2 Electron Source

Electron beam with 90% polarisation is produced by applying LASER to GaAs photocathode in a DC gun. After bunching and accelearating to 76 MeV by means of normal conducting structures, superconducting solenoids rotate the spin vector into vertical and finally the beam is injected into a damping ring.

2.2.3 Positron Source

When electron beam is passed the electron beam through a helical undulator, photons ar e generated because of synchotron radiation and these photons are directed to a rotating Ti-alloy target with 0.4 radiation length. This results in the production of electron-positron pairs. The electrons and photons generated from this process are separated and dumped. This process will generate positrons with a polarisation of 30%.

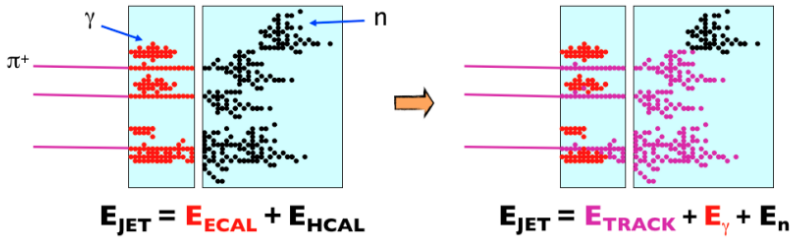


Figure 2.4: Tradiational approach (left) vs Particle flow approach(right) [7]

2.2.4 Damping Rings

The purpose of the damping ring is to lower the emittance of the beams. Electrons and positron damping rings operate at an energy of 5 GeV. In each of these rings, damping is accomplished by means of 54 superferric wigglers, each 2.1 m long. The wigglers operate at 4.5K, with a peak field requirement of 2.16T.

2.2.5 Ring to Main Linac

Ring to main Linac consists of a 5 GeV transport line which is about 15 km long, betatron and energy collimation system, a 180^o turn-around, spin rotators to polarise beam in a specific direction and a two stage bunch compressor.

2.2.6 Beam Delivery System

Beam delivery system (BDS) transports the electron-positron beam to make them collide at the IP with a crossing angle of 14 mrad, from the exit of the high energy linacs. Besides, removing beam halo to reduce background, and measurement of energy and polarisation before and after the collision are also carried out by the BDS. BDS accommodates two detectors about IP in a *push-pull* configuration. This system is designed for full upgrade of average beam power of 14MW so that they do not have to be replaced during an upgrade to 1 TeV.

2.3 International Large Detector

With a view to gaining optimal particle-flow (PFA) performance, which facilitates three-dimensional imaging capability of events, International Large Detector (ILD) design consists of a hybrid tracking system, which includes a combination of silicon tracking with a time projection chamber and a calorimeter system, all mounted inside a 3.5 T solenoid. A detailed description of ILD is available in the ILC Technical Design Report volume 4 [6].

2.3.1 Particle Flow Concept

Particle flow approach exploits the high momentum resolution capability of trackers for charged particles and high granularity of calorimeteres to explicitly reconstruct neutral hadrons and thus distinguish between neutral particles. Such a scheme ensures less use of relatively imprecise calorimetric measurement and calls for highly

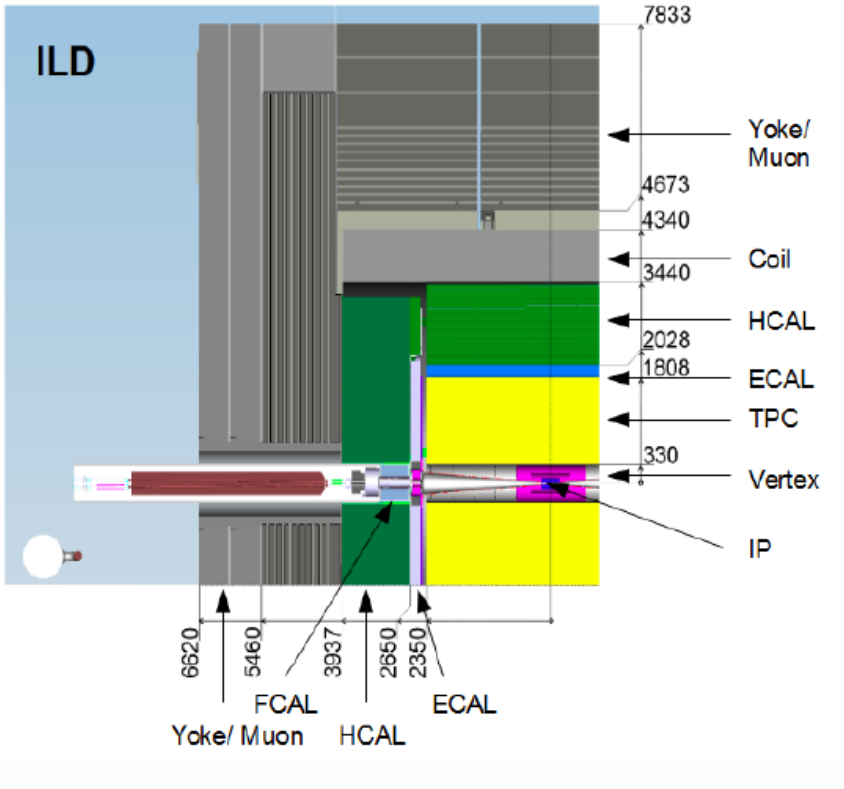


Figure 2.5: Cross Section of the ILD detector system [9]

dense and segmented calorimeters, made of materials having small radiation length and Moliere radius. Besides, the requirements of large distance between interaction point and calorimeter under a large magnetic field, which allow patricles to drift apart and sweep charged particles away from the neutrals, make particle reconstruction by PFA superior to previously used energy flow scheme [8]. A schematic diagram of the PFA is shown in figure 2.3.1.

2.3.2 Overview of detector components

Vertex System

With a view to achieving the ILC goal of precise measurement of signals from heavy (charm and bottom) quarks and τ leptons, vertex system of ILD has been optimised for spatial resolution better than 3 μm near the IP, a material budget below 0.15% X_0/layer, first detector layer within a radius of about 1.6 cm and very low pixel occupancy. Besides, saving of power consumption by means of power pulsing mechanism and a high radiation tolerance are the features of this system.

Silicon Tracking System

The silicon tracking system helps improve the momentum resolution by means of silicon inner and external tracker, which provide precise space points before and after the TPC. Besides, a set of seven silicon disks provide precise traking at small angles in the forward region where TPC does not have any coverage.

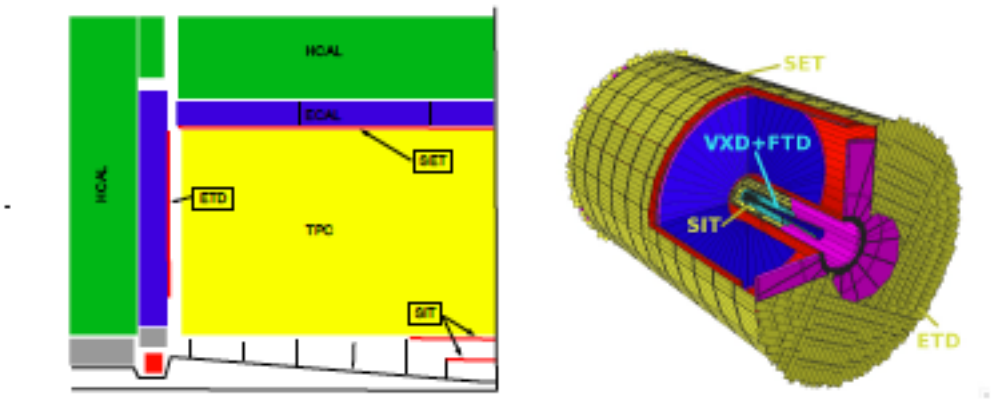


Figure 2.6: Silicon tracking system of the ILC[10]

TPC System

Time Projection Chambers provide three dimensional information of particle with precise resolution. Its low material budget substantially reduces the backgrounds due to Beamstrahlung. By means of this design of TPC, a point resolution better than $100\mu m$ and a double hit resolution of less than 2 mm becomes possible.

Electromagnetic Calorimeter System

ECal forms the first part of hadron showers and thus helps discriminate between different hadrons. With pixel size less than the Mollier radius, ECal can distinguish between overlapping showers, can do pattern recognition of the showers and can reconstruct photons even in presence of particles nearby.

Hadronic Calorimeter System

HCal separates the charged hadrons from the neutral ones and thus contributes highly to particle flow resolution for jet energies upto 100 GeV.

Forward Calorimetry

Lumical and BeamCal described below are the forward calorimeters in ILD.

LumiCal

Lumical carried out precision measurement of luminosity by means of Bhabha scattering. Its small pad size is very suitable for accurate measurement of showers with very small polar angles.

BeamCal

BeamCal are hit by massive amount of Beamstrahlung pairs and along with Pair Monitor it makes bunch-by-bunch measurement of luminosity. It employs a shower finding algorithm which can detect pair backgrounds even at low polar angles.

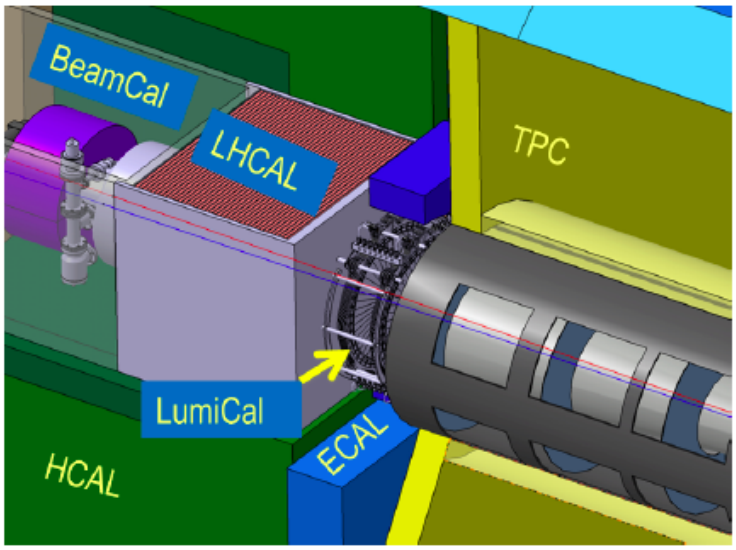


Figure 2.7: Forward calorimeters of the ILC [11]

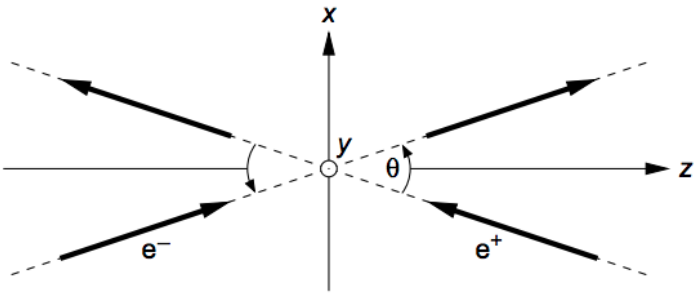


Figure 2.8: ILD coordinate system along with the beam axes [12]

2.3.3 Detector Coordinate System

With a Cartesian right handed coordinate system and the origin being at the nominal IP, z-axis lies along the mean beam direction such that the $p_z^- > 0$ and the y-axis lies along the vertical direction. The crossing angle being on the horizontal plane, causes the incoming (outgoing) beam point to the negative (outgoing) x-axis. A more detailed description of the coordinate system is available in [12].

Because of the inclusion of the crossing angle into the detector design, the forward instruments, centered about the outgoing beams, are tilted by an angle of 7 mrad with respect to the ILD coordinates. In fig 2.8, ILD coordinate system is illustrated in the form of x,z and y axes, represented by the horizontal, vertical and the axis pointing out of the paper respectively. The tilted axes are the beam axes tilted by a crossing angle with respect to the z axis of the ILD coordinate system. The arrows denote the directions of the incoming and outgoing beams.

For a visual depiction of the position of the detectors with respect to the ILD coordinate system, refer to fig. 4.1 provided at section 4.1.

2.3.4 Magnetic System

In order to align the magnetic field parallel to the beam axis with optimal precision, the conventional solenoid field from the detector is superimposed with a dipole field, produced by dedicated dipole windings installed in the detector solenoid. The resulting configuration with the magnetic field aligned to the outgoing beam, is called anti-DID (Detector Integrated Dipole). Presence of anti-DID ensures that majority of the soft e^+e^- pairs, produced from the beam collision induced beamstrahlung, are directed into the holes of the outgoing beams [13].

Anti-DID dipole windings have a reverse polarity compared to that of previously adopted DID scheme. DID scheme was deprecated because of inferior background deposition as compared to the anti-DID. A detailed simulation result of the comparison is available at [14].

Chapter 3

Pair Monitor

3.1 Introduction

Successful run of ILC demands unprecedentedly high luminosity. Because of this goal, an accurate measurement of luminosity is required. While elastic scattering (Bhabha scattering) was used in early lepton colliders such as LEP, presence of several methods to determine luminosity will increase reliability in the obtained results. Given the fact that the beam size of ILC will be of nano-meter order, application of such methods are considered crucial. With this end in view, a method of measuring nanometer beam size, using incoherent e^+e^- pair backgrounds, generated from beam collision induced process namely, beamstrahlung, has been suggested in [15].

For transverse Gaussian beam distributions, luminosity can be defined as [16]:

$$L = \frac{N_1 N_2}{4\pi\sigma_x\sigma_y} n_b f H_D \tag{3.1}$$

where N_1 and N_2 are particle per bunch for incoming and outgoing beams respectively and transverse collision area is described as $4\pi\sigma_x\sigma_y$. n_b, f, H_D are number of bunches in a train, collision frequency and enhancement factor due to the *pinching* of particles when they cross the field of the opposite bunch.

Assuming $N_1 = N_2 = N$, equation 3.1 can also be expressed as:

$$L \propto H_D \frac{N}{\sigma_x} N n_b f_r \frac{1}{\sigma_y} \tag{3.2}$$

N/σ_x can be shown to be related to the beamstrahlung emitted, whereas $1/\sigma_y$ is limited by the ability to achieve and preserve a small beam emittance and to squeeze the beam to a very small size.

3.2 Beamstrahlung

When a force applied to a charged particle causes it to travel through a curved trajectory, it emits energetic photons, known as Beamstrahlung. Beamstrahlung causes the particle to collide at an energy less than its nominal energy. As such, physics performance of the experiment becomes worse. Beamstrahlung can be described as a critical energy $\hbar\omega_c$ required to emit a photon [17],

$$\hbar\omega_c = \frac{3}{2}\frac{\hbar\gamma^3 c}{\rho} \tag{3.3}$$

where, γ, c, ρ are the relativistic gamma parameter, speed of light and bending radius of the particle trajectory respectively. Beamstrahlung parameter is defined as,

$$\Upsilon = \frac{2}{3}\frac{\hbar\omega_c}{E} \tag{3.4}$$

The Beamstrahlung spectrum can be described by the Sokolov-Ternov spectrum,

$$\frac{d\dot{\omega}}{d\omega} = \frac{\alpha}{\sqrt{3}\pi\gamma^2}\left[\int_x^\infty K_{5/3}(x')dx' + \frac{\hbar\omega}{E}\frac{\hbar\omega}{E-\hbar\omega}K_{2/3}(x)\right] \tag{3.5}$$

Here, $x = \frac{\omega}{\omega_c}\frac{E}{E-\hbar\omega}$ and $K_{5/3}, K_{2/3}$ are modified Bessel functions. If $\Upsilon \ll 1$, the power of the photon radiation is proportional to Υ^2:

$$P = \frac{e^2}{6\pi\epsilon_0}\frac{c}{\rho^2}\gamma^4 = \frac{2}{3}\frac{r_e c}{\lambda_c^2}mc^2\Upsilon^2 \tag{3.6}$$

where $\lambda_c = \hbar/(mc)$. The average Beastrahlung parameter is,

$$\langle\Upsilon\rangle = \frac{5}{6}\frac{Nr_e}{\alpha\sigma_z(\sigma_x+\sigma_y)} \tag{3.7}$$

The maximum value of Υ parameter is,

$$\Upsilon_{max} \approx \frac{12}{5}\langle\Upsilon\rangle \tag{3.8}$$

In classical regime, $\Upsilon \ll 1$ and denotes synchotron radiation. For $\Upsilon \gg 1$ the radiation is partially suppressed because the critical energy exceeds the beam energy.

3.3 Choice of Beam shape

n_γ, the average number of photon emitted per beam particle can be expressed as [17]:

$$n_\gamma \propto \Upsilon\frac{\sigma_z}{\gamma} \propto \frac{N}{\sigma_x+\sigma_y} \tag{3.9}$$

Similarly, the average energy of each photon,

$$E_\gamma \propto \Upsilon\frac{1}{\gamma} \propto \frac{N}{\sigma_z(\sigma_x+\sigma_y)} \tag{3.10}$$

Number of photons is the more important parameter because it is dependent on only σ_x and σ_y. In order to reduce Beamstrahlung, the denominator $(\sigma_x+\sigma_y)$ of equation 3.9, has to be large, whereas, for high luminosity the product $\sigma_x\sigma_y$ need to be small. Flat beam with $\sigma_x \gg \sigma_y$ is the right choice for this purpose because damping ring naturally delivers beams with larger horizontal emittance and small vertical emittance. In case of flat beam i.e. for $\sigma_x \gg \sigma_y$, N/σ_x from equation 3.2 refers to the number of Beamstrahlung photons. Incoherent processes are the major source of background and pair monitor is designed for detection of these particles.

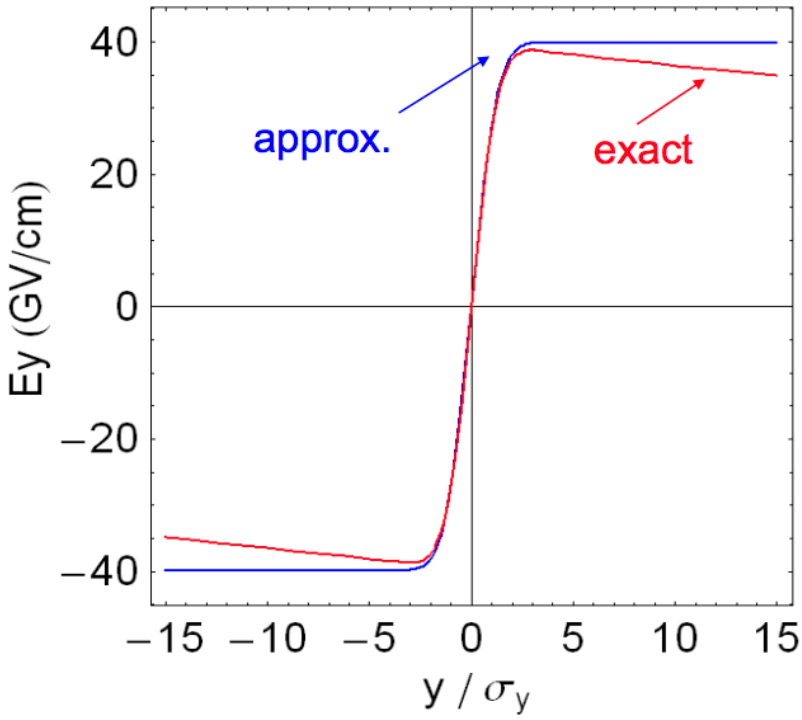


Figure 3.1: Exact and approximate E_y vs y/σ_y [18]

3.4 Pair Background

The photons produced as a result of Beamstrahlung can either directly interact with oncoming electron or positron, or they might pair create to produce e^+e^- pairs. The energy in the former case is low and might case additional interactions leading to various physical processes such as Breit-Wheeler ($\gamma\gamma \rightarrow e^+e^-$), Landau-Lifschitz ($ee\gamma\gamma \rightarrow e^+e^-e^+e^-$), Bethe-Heitler ($e\gamma\gamma \rightarrow e^{\pm}e^+e^-$) [15]. This case of Beamstrahlung *real* photons interacting with themselves or oncoming e^+e^- is called incoherent process.

3.5 Electric field due to a relativistic flat beam [18]

Electric field $\boldsymbol{E}$ due to a beam can be computed by applying Gauss's law:

$$\int_S \boldsymbol{E}.\boldsymbol{n}\ da = \frac{Q}{\epsilon_0} \tag{3.11}$$

For a beam round Gaussian beam, the charge density is expressed as ,

$$\rho(r) = \frac{Ne}{2\pi\sigma^2} exp(-\frac{r^2}{2\sigma^2}) \tag{3.12}$$

The electric charge inside the bunch at aradial distance r can be computed by

$$Q = \int \rho(r')r'dr'd\phi = \frac{Ne}{2\pi\sigma^2}\int_0^{2\pi} d\phi \int_0^r r' exp(-\frac{r'^2}{2\sigma^2})dr' \tag{3.13}$$

Equation 3.13 can be equated with the left hand side of the equation 3.11, that equals $E2\pi rL$, for cylinder of finite length L, to get the electric field:

$$\boldsymbol{E} = \frac{Ne}{2\pi\epsilon_0 r L}\left(1 - exp\left(-\frac{r^2}{2\sigma^2}\right)\right)\hat{r} \tag{3.14}$$

Similar calculation can be done for a relativistic flat beam where $\sigma_x \gg \sigma_y$. An exact calculation with Gauss's law in a closed form results in Basetti-Erskine formula with $\sigma_x > \sigma_y$:

$$E_x = \frac{Ne}{2\epsilon_0\zeta} Im\left(W((x+iy)/\zeta) - exp(-(\frac{x^2}{2\sigma_x^2} + \frac{y^2}{2\sigma_y^2}))W((x\frac{\sigma_y}{\sigma_x}) + iy\frac{\sigma_x}{\sigma_y})/\zeta)\right) \tag{3.15}$$

$$\times\, exp(-(\frac{(z-ct)^2}{2\sigma_y^2})/(\sqrt{(}2\pi)\sigma_z)) \tag{3.16}$$

$$E_y = \frac{Ne}{2\epsilon_0\zeta} Re\left(W((x+iy)/\zeta) - exp(-(\frac{x^2}{2\sigma_x^2} + \frac{y^2}{2\sigma_y^2}))W((x\frac{\sigma_y}{\sigma_x}) + iy\frac{\sigma_x}{\sigma_y})/\zeta)\right) \tag{3.17}$$

$$\times\, exp(-(\frac{(z-ct)^2}{2\sigma_y^2})/(\sqrt{(}2\pi)\sigma_z)) \tag{3.18}$$

$$\tag{3.19}$$

where $\zeta = \sqrt{(}2(\sigma_x^2 - \sigma_y^2))$ and W is the complex error function.

$$W(t) = exp(-t^2)(1 + iErfi(t)) \tag{3.20}$$

where $Erfi(t) = Erf(it)/i$ and $Erf(t) = \frac{2}{\sqrt{(}\pi)}\int_0^t exp(-p^2)dp$

But the approximate calculation is very straightforward and it helps us to visualize the electric field better. For a flat beam $\sigma_x \gg \sigma_y$, an assumption that the beam is infinitely wide with a cosntant density per unit length can be made:

$$\rho(x) \approx \frac{1}{\sqrt{2\pi}\sigma_x} \tag{3.21}$$

and the charge density in the y is taken to be a Gaussian with finite shape,

$$\rho(y) \approx \frac{1}{\sqrt{2\pi}\sigma_y} exp\left[-\frac{1}{2}\left(\frac{y}{\sigma_y}\right)^2\right] \tag{3.22}$$

Now using equation 3.11 for finite box of length Δz and height δx,

$$E_y(y,z)\Delta x\Delta z \approx \frac{QN\rho(x)\rho(z)\Delta x\Delta z}{\epsilon_0}\int_0^y \rho(y')dy' \tag{3.23}$$

$$E_y(y,z) = \frac{QN}{2\sqrt{(}2\pi)\epsilon_0\sigma_x} Erf(\frac{y}{\sqrt{2}\sigma_y})\rho(z) \tag{3.24}$$

Assuming a Gaussian distribution for z, the peak value of the electric field is,

$$E_y = \frac{QN}{4\pi\epsilon_0\sigma_x\sigma_z} \tag{3.25}$$

Therefore it is seen that the electric field in the y direction that causes maximum angular kick to the produced pair particles, is mostly dependent on σ_x in case of flat beams.

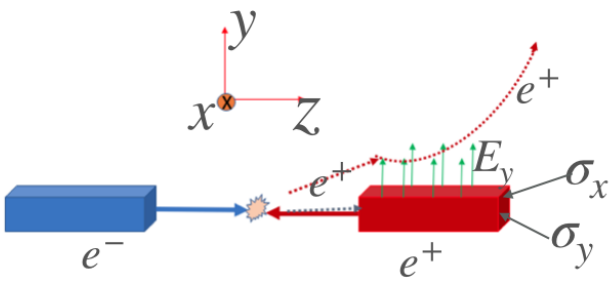


Figure 3.2: Principle of deflection of incoherent pair background

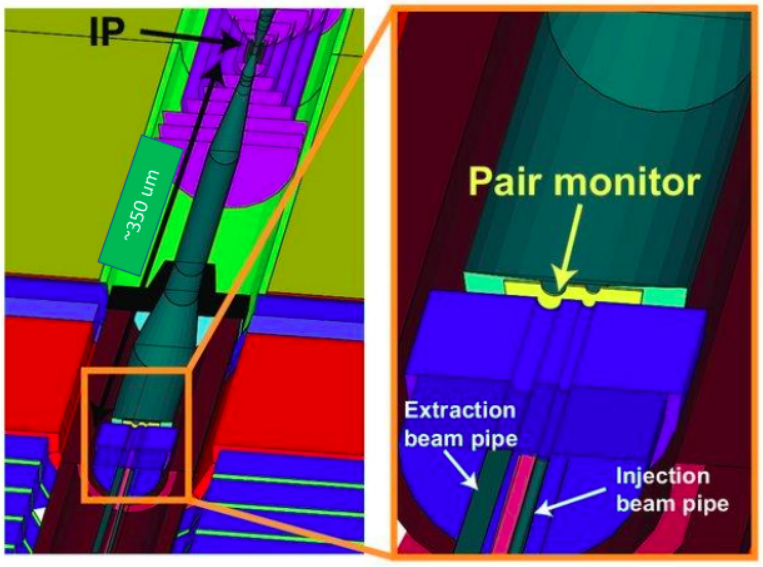


Figure 3.3: Detector geometry and location of the proposed pair monitor [19]

3.6 Pair Monitor

Figure 3.2 visualizes the production of incoherent pair background and their deflection by the vertical electric field. The beam shape and sizes are not scaled to the actual ones. The nominal values of horizontal (σ_x) and vertical (σ_y) beam sizes are 729 nm and 7.7 nm respectively for 250 GeV. Pair monitor is a silicon pixel detector that detects these incoherent pair background particles.

The pixel size of pair monitor ($400 \times 400\ \mu m^2$ [19]) is fine enough to detect the position of the particles passing through it. Its thickness is proposed to be about 300 μm and the distance from the interaction point is about 350 cm. Because of this small value of its thickness it cannot determine the energy of particles.

Chapter 4

Simulation Procedure

4.1 Simulation Tools and Method

In this study full detector simulation of pair monitor has been undertaken, with the realization that the first layer of the BeamCal mimics pair monitor readout signal. Simulation of 25 sets of beam sizes, with 5 sets for each of the horizontal (σ_x) and vertical (σ_y) beam sizes, were simulated. The current simulation assumes the vertical displacement (δ_y) between two beams to be zero.

While the first work on full simulation was carried out by [1], the current work is reflective of the latest ILD configuration and software packages for PFA (see 2.3.1) based event reconstruction.

The simulation consists of three steps:

- Simulation of Beam-Beam Interaction by CAIN [20]
- Flying the incoherent pair particles to the detectors and carrying out of the detector reactions by DD4hepSimulation package [21]
- Extracting the particle hit information using MARLIN [22] and LCIO [23] packages

4.2 Simulation of Beam-Beam Interaction

CAIN by means of which, beam-beam interaction has been carried out, is a FORTRAN based Monte-Carlo code for the interaction involving high energy electron, positron, and photons. Each of the σ_x and σ_y parameters, with values of 0.8, 1.0, 1.2, 1.4 and 1.6 times the nominal RMS values of the beam sizes ($\sigma_x^* = 729.0\,nm$, $\sigma_y^* = 7.7\ nm$) [24] have been simulated. As such, each dataset of σ_x corresponds to 5 different datasets of σ_y and vice versa. According to the convention of CAIN user's manual, the electron and positron beams have been simulated as right and left going beams respectively.

Table 4.1 describes the different parameter values used in the CAIN simulation. A detailed explanation of CAIN simulation method related to this study has been described in [2].

Energy (one beam)	125 GeV
Number of bunches	1312
Bunch Population	2 $\times 10^{10}$
Number of macro-particles	100000
Collision Rate	Number of bunches $\times$ 5 Hz
Horizontal emittance	10
Vertical emittance	35
IP Horizontal β function	13.0 mm
IP Vertical β function	0.41 mm
Slope (=$\phi_{cross}/2$)	7 mrad
External Field (B_x, B_y, B_z)	(0,0,3.5T)
Constant Field QED	BeamStrahlung Polarization Maximum event probability per time step= 0.5
Gaussian Tail cutoff $n_x, n_y, n_z, n_\epsilon$	4.5 (units of respective σ)
Polarization vector	$(\zeta_x, \zeta_y, \zeta_z)$ Electron Beam = (0,0,-1) Positron Beam = (0,0,1)
Beam-Beam Field	Horizontal bins = 32 Vertical bins = 128 Horizontal mesh width (Left and Right beams)= 12$\times\sigma_x$

Table 4.1: CAIN Simulation Parameters

Parameter Name	Value
Crossing Angle Boost	7×10^{-3} rad
Field equation	*Mag_UsualEqRhs*
Filed Stepper	*HelixSimpleRunge*
Filter Tracker	*edep1kev*
Physics List	*QGSP_BERT*
Physics Range cut	0.1 mm

Table 4.2: Paramaters used in DD4HEP toolkit

4.3 Simulation by DD4hepSimulation Package

DD4hep toolkit brings together the ROOT Geometry package for detector construction, visualization etc. and Geant4 [25] simulation toolkit for simulation of detector response. Its purpose is accurate interpretation of event throughout the full experiment life cycle and carrying out all data processing applications such as simulation, reconstruction, online trigger and data analysis, under a simple API (application programming interface).

DD4hep toolkit allows compact detector description, written in **XML** [26] language, to define an ideal detector used during the conceptual design phase of an experiment. In the current study, the ideal detector description namely, **ILD-l5-v05** [27], incorporates realistic detector geometry, solenoid field map, anti-DID field map and magnetic fields in the forward focusing magnets, into the DD4hep toolkit by means of an aforementioned XML file of the same name. This study has been performed using the DD4hep API implemented in Python programming language [28].

The values of the parameters used in this simulation are shown in table 4.2.

4.3.1 Visualization of the detector geometry

Fig 4.1 depicts the detector construction implemented by **ILD-l5-v05**, as a logarithmic color-map plot of number of radiation lengths in a bin with respect to the ILD detector coordinate system (see 2.3.3), where y-axis points out of the page.

The tilted forward calorimetry is clearly visible here. Besides, the detector of interest i.e. BeamCal, which is about 300 cm away from the IP, is seen to have a very high radiation length.

4.3.2 Visualization of the magnetic field

The effect of anti-DID field as introduced in sec. 2.3.4, can be seen in the figure 4.2. Figure 4.2a shows that the polarities of the B_x field, about the z-axis, are opposite to each other. The resultant magnetic fields exert force along the outgoing beams so that most of the beam backgrounds can escape through the beam pipes. Additionally, the extra winding responsible for the ant-DID effect, is seen to have very dark color, meaning magnetic field is highly concentrated in that region.

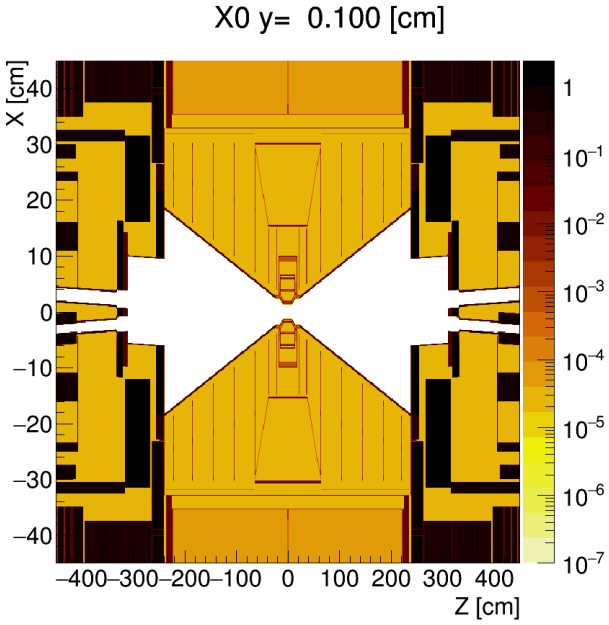
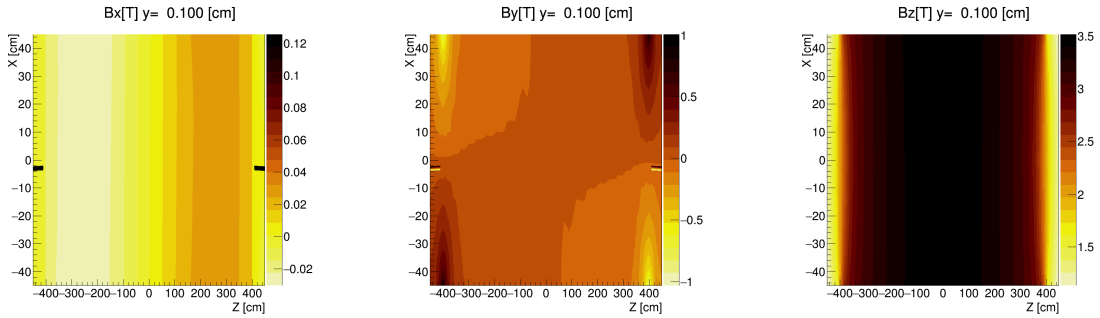


Figure 4.1: Cross Section of the ILD detector system [29]



(a) $B_x(T)$ (b) $B_y(T)$ (c) $B_z(T)$

Figure 4.2: Magnetic Field Visualization [29]

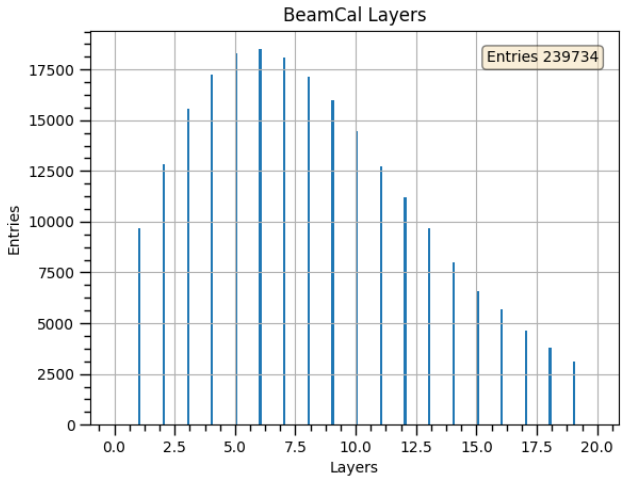


Figure 4.3: BeamCal layers

4.3.3 Extraction of particle hit information using Marlin and LCIO packages

Marlin (Modular Analysis and Reconstruction for the LINear collider) is a software framework based on ILCSOFT [30]. It implements a *Processor* to digitise hit collections which then extracts the digitised collections through LCIO data model. LCIO is a persistency framework developed solely for linear colliders. The purpose of LCIO is unification of software used in the ILC [23].

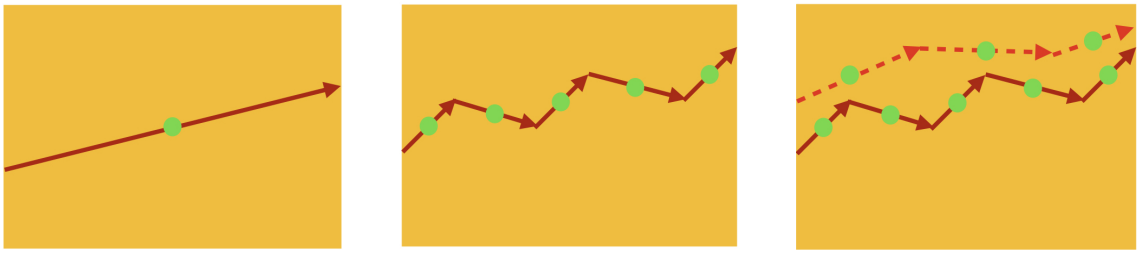
Marlin along with LCIO helps analysis progress by just defining a processor inside a steering file. In the current study, ILCSoft version 02-00-02, MARLIN version 01-16 and LCIO version 02-12-01, have been used.

Digitised information of an event are accessed inside a processor through the *LCEvent* class. Event information inside the BeamCal are accessed as *LCCollection* type object by means of *SimCalorimeterHit* class inside the processor. Each element of this collection corresponds to hits at the BeamCal cells. Each of this hit denote only the position of the cell hit by the particle and not the exact position i.e. coordinate values or layer number of the hit. Therefore, these collections are decoded by means of *CellIDDecoder* and layer number corresponding to every element of the BeamCal collection is obtained by passing the address of the element into the decoder. The layer values after decoding is shown in figure 4.3

For the first layer, number of contributions by all *MCParticles*, inside each aforementioned element, is obtained using *getNMCContributions()*, which is a member function of *SimCalorimeterHit* class.

Such contributions contain the energy, position, pdg etc. information calculated by Geant4 at the midpoint between every two Geant4 *steps* inside a single cell of the first layer of detector as shown in fig. 4.4. The thickness of the first layer is about $300\mu m$ nad has been exaggerated in fig. 4.4 for convenience of explanation. Following three cases can be used to describe the process.

- Figure 4.4a: An *MCParticle* passes through the detector without any leaving any intermediate *step*. Geant4, by default stores information (creates *steps*) at the entrance and exit of a detector volume. Therefore, *getNMCContributions()*



(a) One *MCParticle*, one step

(b) One *MCParticle*, multiple steps

(c) Multiple *MCParticles*multiple step

Figure 4.4: Three cases of *steps* inside one cell of the first layer of BeamCal (layer thickness exaggerated)

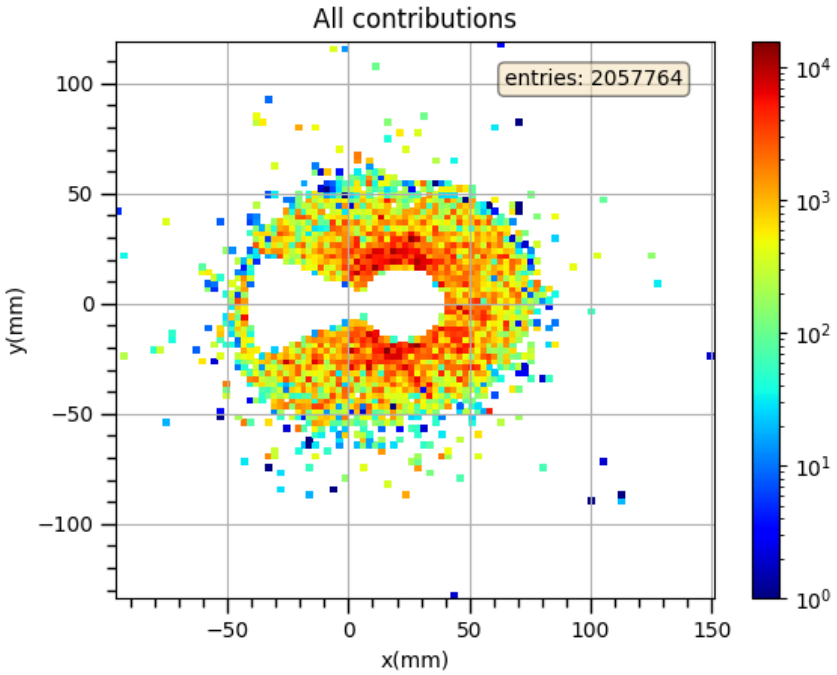


Figure 4.5: Position distribution of the BeamCal readout signal before rotation.

will return a value of 1 corresponding to the midpoint of the points at the entrance and exit.

- Figure 4.4b: An *MCParticle* while passing through the layer undergoes scattering. There are 5 midpoints between the *steps* combined. Therefore, *getNMCContributions()* will return a value of 5.

- Figure 4.4c: Two *MCParticles* while passing through the layer undergoes scattering. One leaves 5 midpoints, while the other (shown in segmented line) leaves 3 midpoints. Therefore, *getNMCContributions()* will return a value of 8.

Such complication arises because the *SimCalorimeterHit* class gives access only to cell IDs of the BeamCal. In each of the above cases, only the first midpoint between the *steps* corresponding to a single type of *MCParticle* is collected.

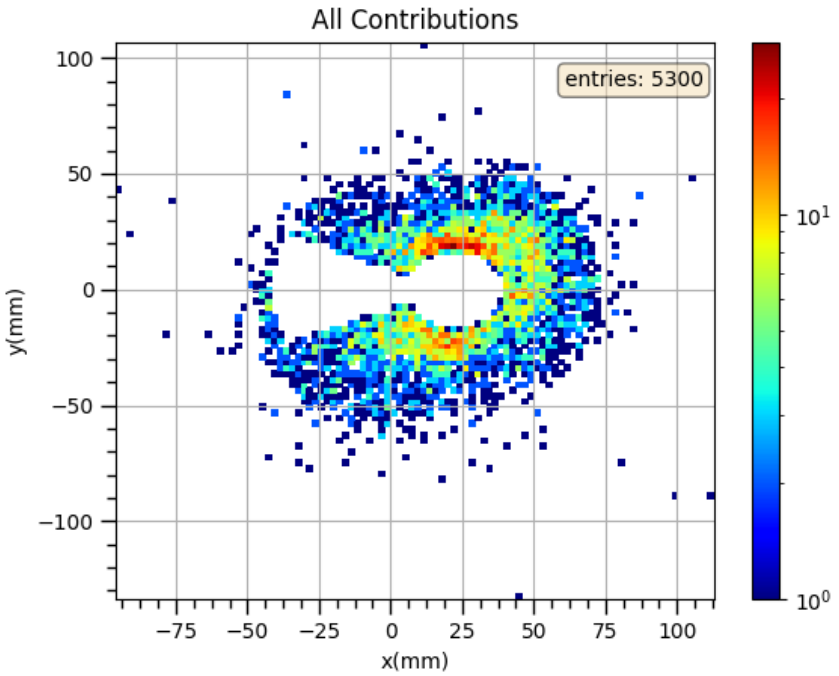


Figure 4.6: Position distribution of the BeamCal readout signal after rotation.

As seen in figure 4.6, after the above scheme to keep only the midpoint between the first two *steps* of a single type of *MCParticle* is applied, the first layer readout produces 5300 signals (4.6) instead of much higher entries in figure 4.5.

4.3.4 Plots of different variables

A plot of the position (x-coordinate vs y-coordinate) distribution corresponding to 50 bunches at the positive z-axis for beam sizes 0.8 times the nominal value for both σ_x and σ_y, is shown in figure 4.8. The origin of the figure 4.7 has a shift because of the tilt of the forward detectors as mentioned before. The circular region of the keyhole shape is the hole due to the outgoing beampipe. After rotation by an angle equal to the crossing angle (7 mrad) about the y-axis, the outgoing beampipe is centered at the origin of the axes.

As mentioned in section 4.1, simulation of 25 sets of parameters with each value of σ_x corresponding to 5 values of σ_y and vice versa, have been carried out.

The plots 4.9 and 4.10 shows the azimuthal angle, $\phi = tan^{-1}(y/x)$ distributions for different values of σ_x and σ_y respectively. They have been plotted without rotation. That is, they correspond to the x vs y plot 4.7. *Beam size pair* means one of the 25 combinations of the horizontal and vertical beam sizes.

The ϕ plots of different σ_x (figure 4.9), vary across the whole region $[-3, 3]$ with small statistical fluctuations. But, the fluctuations in the ϕ plot (figure 4.10), that corresponds to different values of σ_y, are very high and clear correlation of the plots with the values of σ_y becomes visible after 50 bunches (figure 4.15) within the ϕ range of $[0, 1]$.

Finally, the ϕ plots for different values of σ_x and σ_y for 15000 bunches are plotted in figure 4.17 and 4.16 respectively. Even after this statistic, no drastic change of ϕ

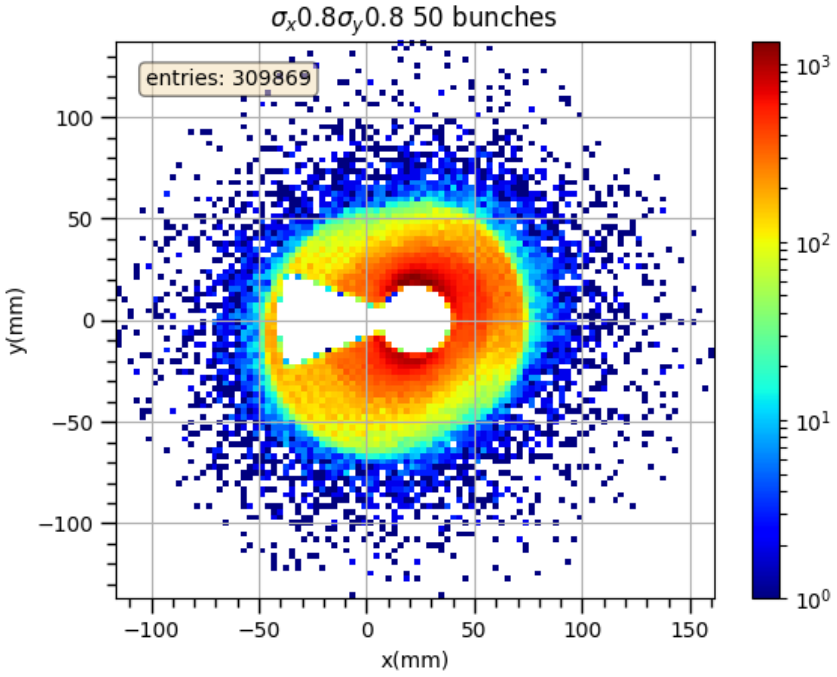


Figure 4.7: Position(mm) distribution of the BeamCal readout signal before rotation.

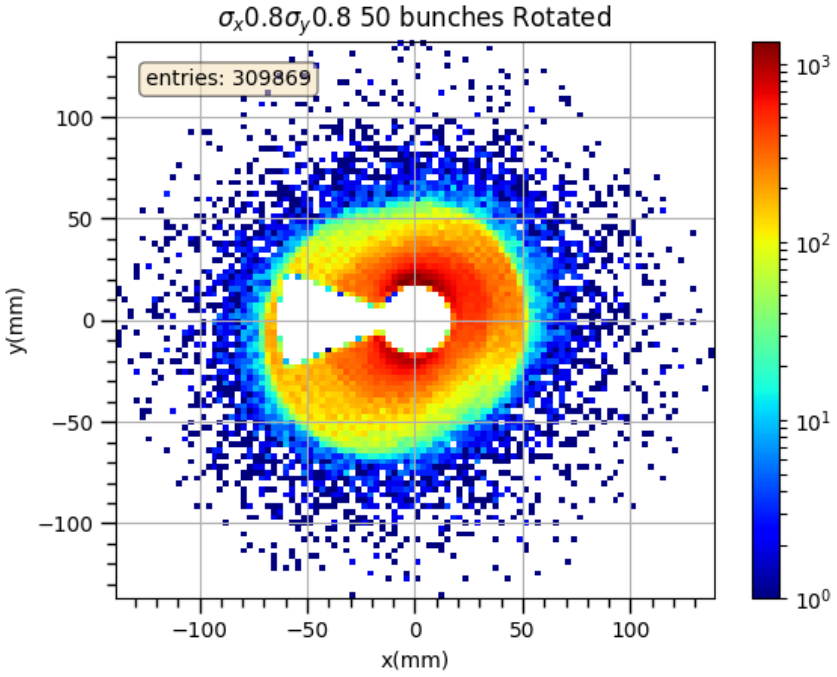


Figure 4.8: Position(mm) distribution of the BeamCal readout signal after rotation.

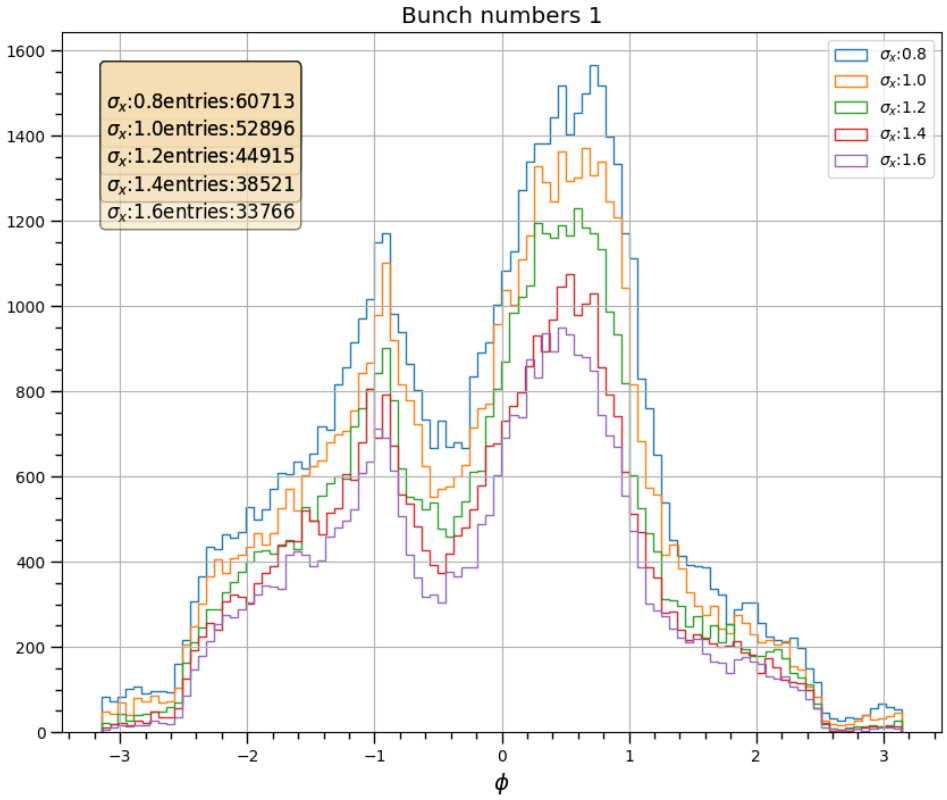


Figure 4.9: ϕ (radian) plots for different values of σ_x (1 bunch/beam size pair)

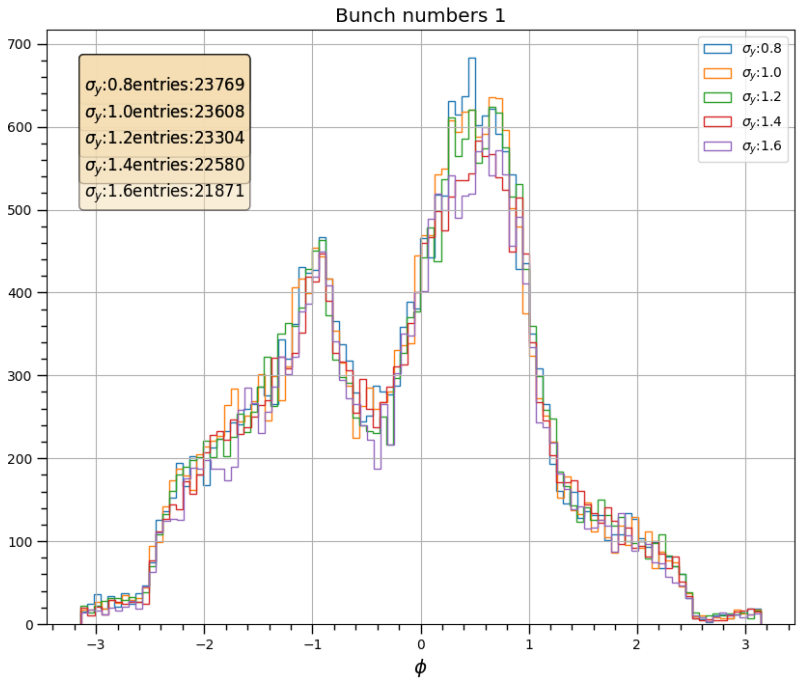


Figure 4.10: ϕ (radian) plots for different values of σ_y (1 bunch/beam size pair)

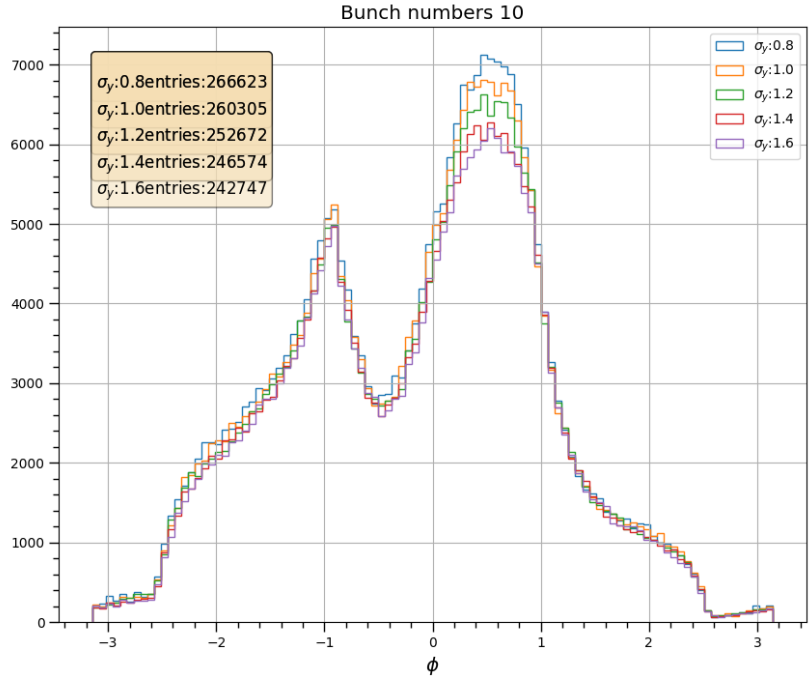


Figure 4.11: ϕ (radian) plots for different values of σ_y (10 bunch/beam size pair)

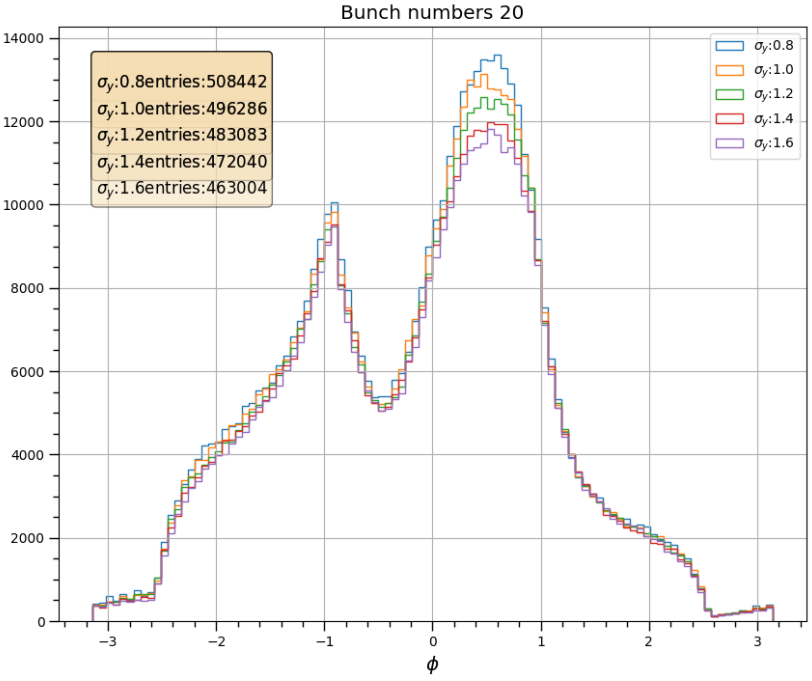


Figure 4.12: ϕ (radian) plots for different values of σ_y (10 bunch/beam size pair)

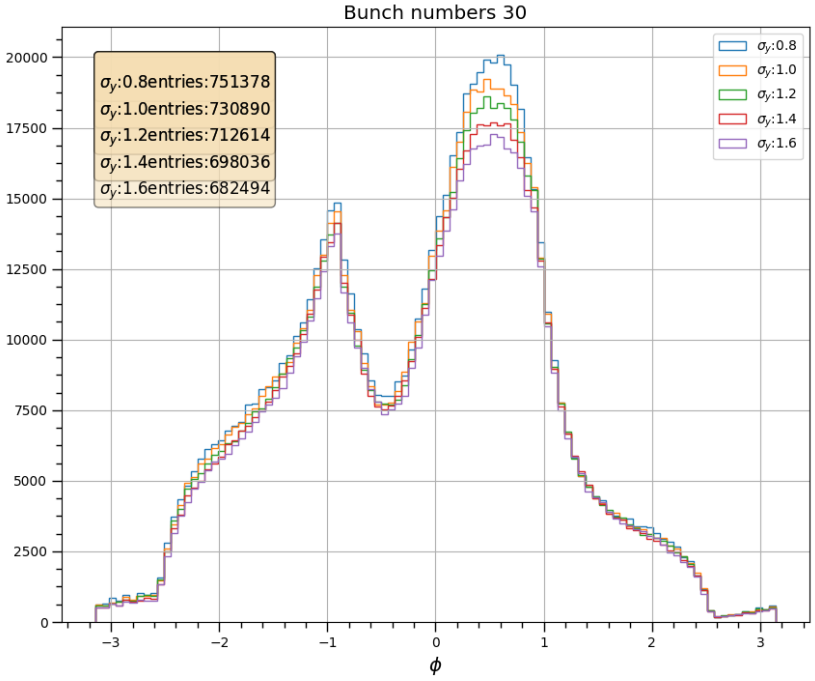


Figure 4.13: ϕ (radian) plots for different values of σ_y (30 bunch/beam size pair)

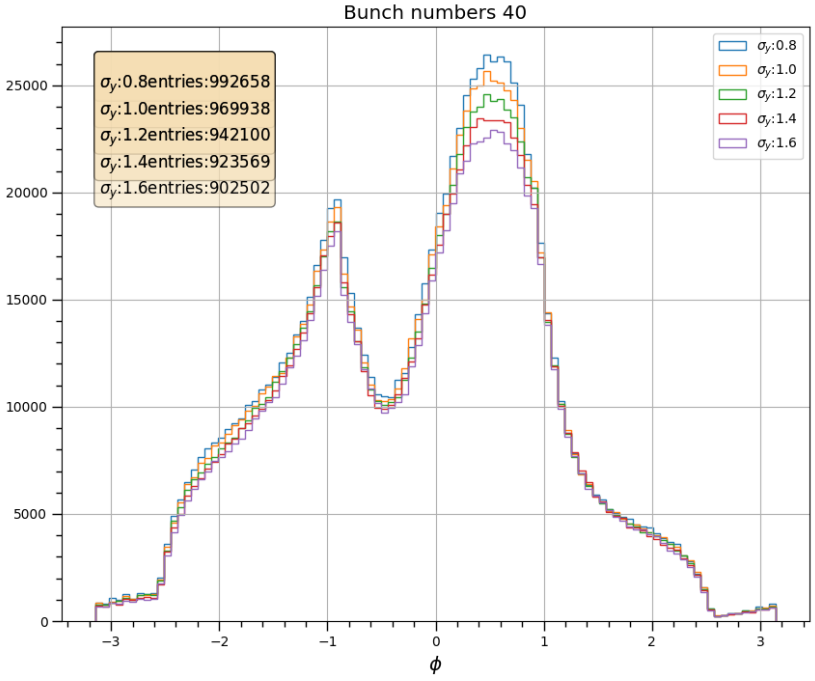


Figure 4.14: ϕ (radian) plots for different values of σ_y (40 bunch/beam size pair)

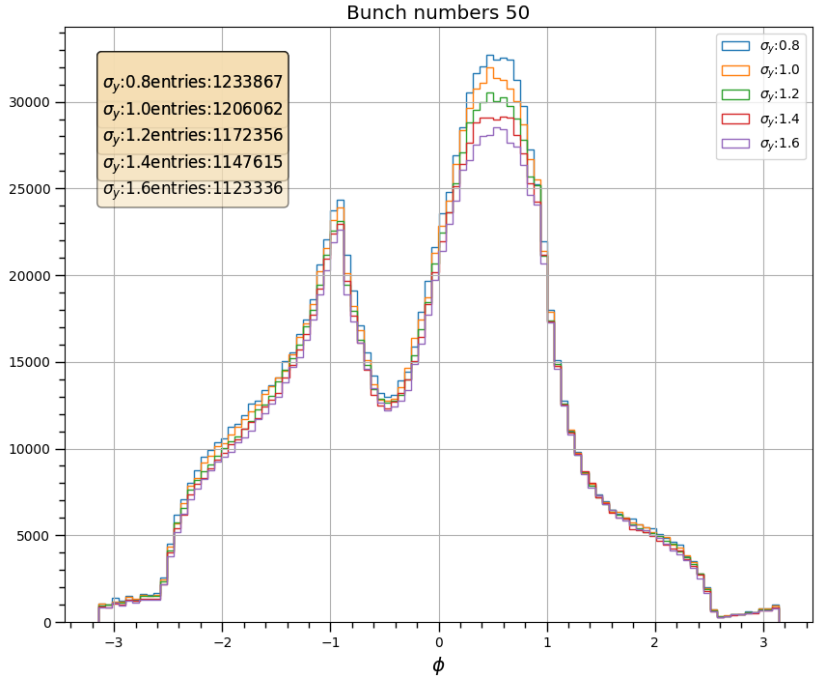


Figure 4.15: ϕ (radian) plots for different values of σ_y (50 bunch/beam size pair)

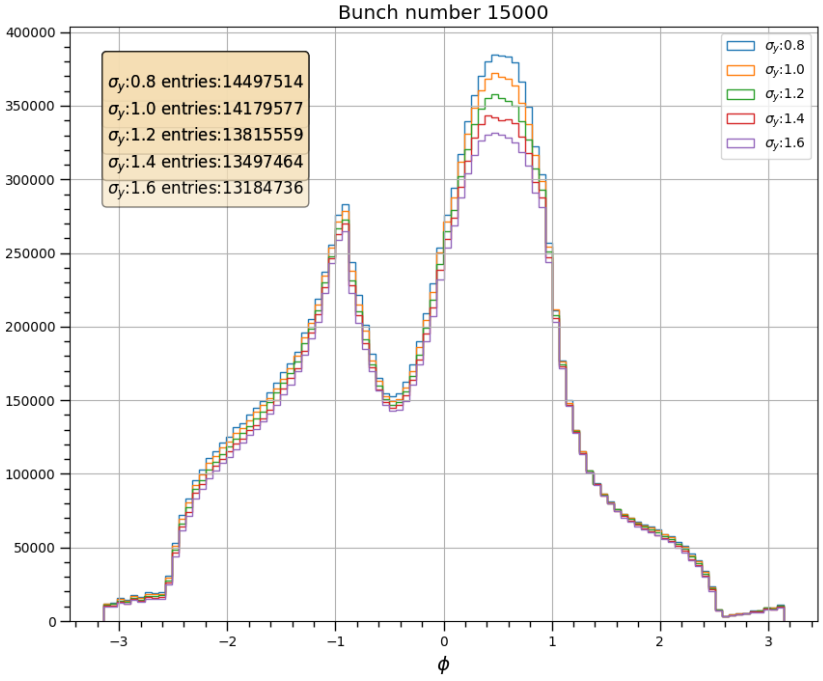


Figure 4.16: ϕ (radian) plots for different values of σ_y (15000 bunch/beam size pair)

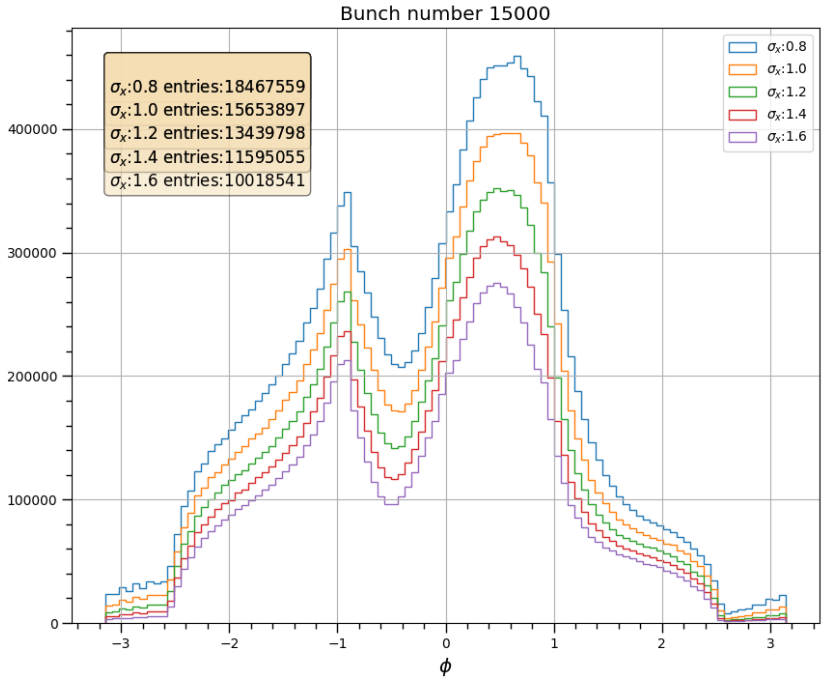


Figure 4.17: ϕ (radian) plots for different values of σ_x (15000 bunch/beam size pair)

plots is visible outside the ϕ range of $[0, 1]$.

The plots of ρ, the distance of the particles from the center are plotted corresponding to 50 bunches for different values of σ_x. The plots corresponding to each σ_x value differ across the range $[20, 80]$.

On the other hand, even though the same plots for different values of σ_y show some difference in that range, it is not as drastic as the case of changing σ_x.

4.4 Conclusion

From the distributions as seen in figures 4.17, 4.16, 4.20 and 4.21, it can be seen that the number of particles is a major discriminant between different values of both horizontal and vertical beam sizes. Especially, the number of particle hit increases drastically when the values of σ_x is decreased even for one bunch crossing. In case of decreasing values of σ_y, the number of hits increase slightly in most of the range in both ρ and ϕ distributions. The influence of the keyhole shape (as seen in fig. 4.7) is responsible for the asymmetry of the ϕ distribution (fig. 4.17, 4.16) about the origin. The keyhole shape is also responsible for the entries of ρ plots (figure 4.20) to peak, after the ρ value exceeds about 40 mm. The energy distribution peaks at about 0.1 MeV, and the number sharply falls onward. This is because of the very low energy of the incoherent backgrounds hitting the pair monitor.

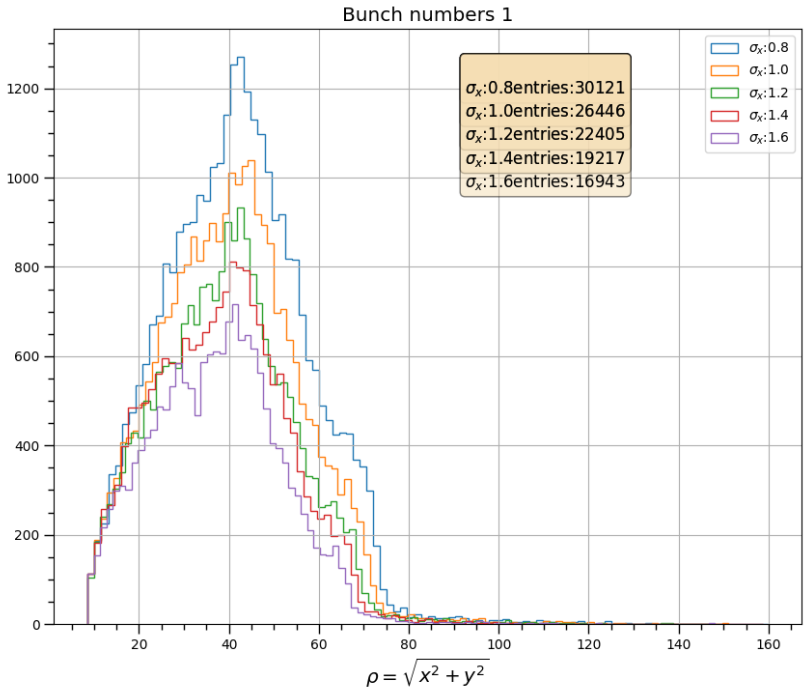


Figure 4.18: $\rho = \sqrt{x^2 + y^2}$ (mm) plots for different values of σ_x (1 bunch/beam size pair)

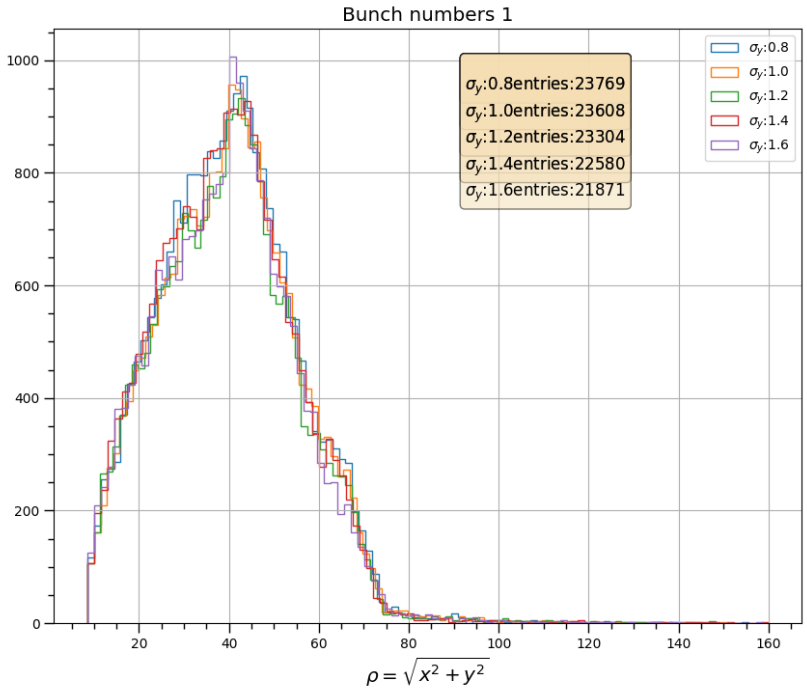


Figure 4.19: $\rho = \sqrt{x^2 + y^2}$ (mm) plots for different values of σ_y (1 bunch/beam size pair)

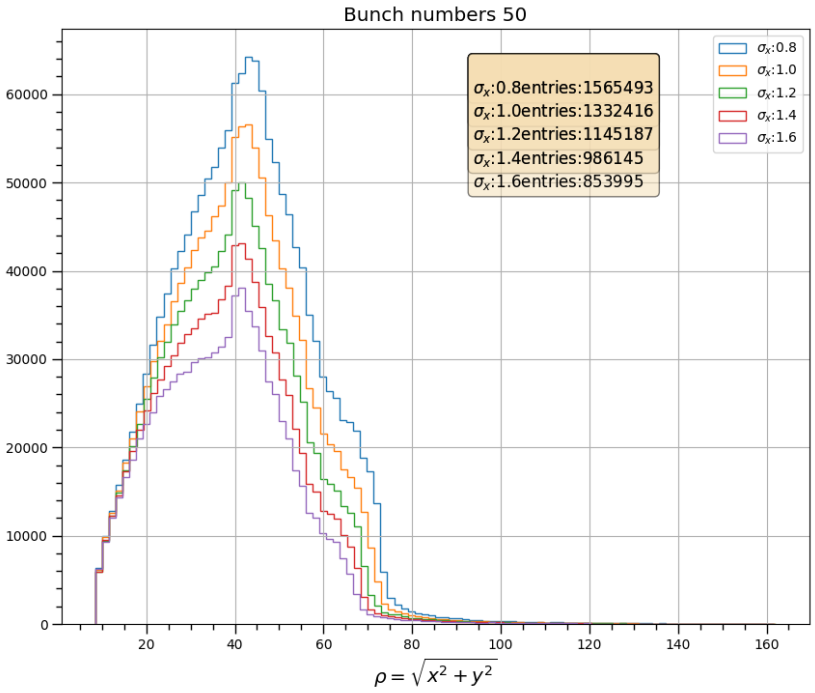


Figure 4.20: $\rho = \sqrt{x^2 + y^2}$ (mm) plots for different values of σ_x (50 bunches/beam size pair)

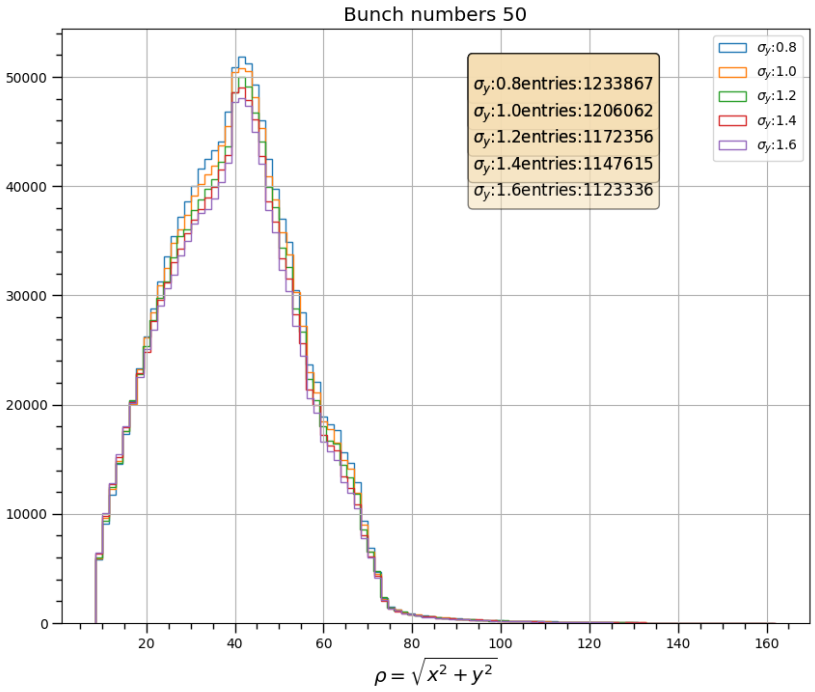


Figure 4.21: $\rho = \sqrt{x^2 + y^2}$ (mm) plots for different values of σ_y (50 bunches/beam size pair)

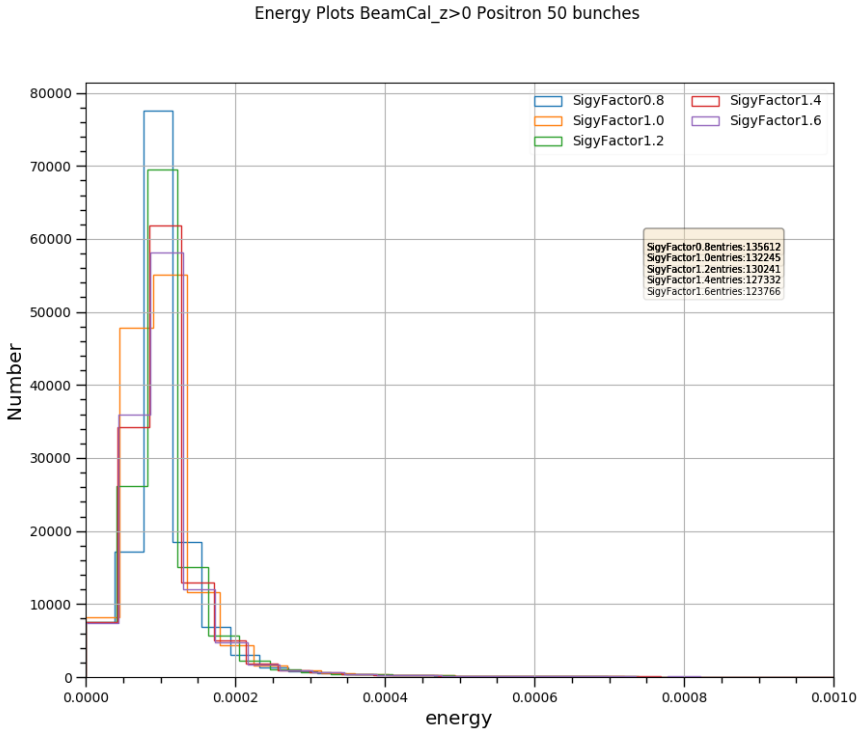


Figure 4.22: Energy(GeV) plots for different values of σ_x (50 bunhes/beam size)

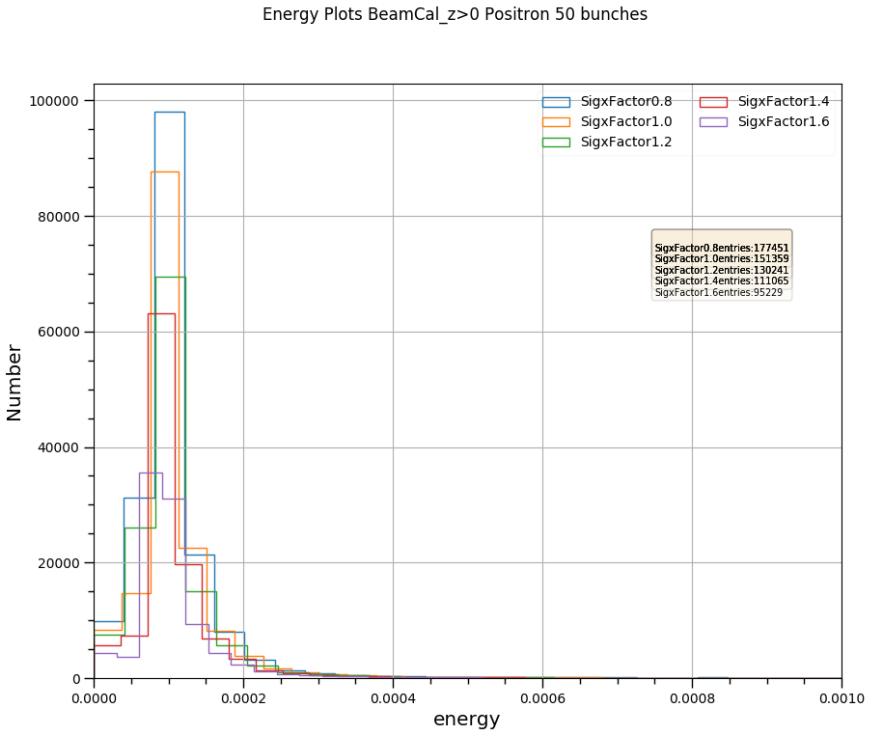


Figure 4.23: Energy(GeV) plots for different values of σ_y (50 bunches/beam size pair)

Chapter 5

Machine Learning and Its Application

5.1 Introduction

Machine learning is an intersection of computer science and statistics and its study involves the design of algorithms in order to draw statistical inferences from observation. Machine learning can be categorized into supervised learning, unsupervised learning, reinforcement learning etc. The current study deals with an application of supervised learning.

In the language of machine learning, any observation can be considered to be an unknown function $y = g(x)$ where x denotes the independent variable. The purpose of machine learning is to define a hypothesis $\mathbb{H}$ which contains a set of many functions, and to choose a certain function h from $\mathbb{H}$, so that $h \approx g$ in a strictly mathematical sense.

An important aspect of prediction by means of machine learning is an assumption about the dataset. Different algorithms make different assumptions about the dataset. That is why, for a successful prediction, it is necessary to check if these assumptions about the properties of the dataset correspond to the dataset that is being dealt with.

5.2 Current Dataset

The parameters that were used to distinguish the patterns of σ_x and σ_y were mainly some functions of distance of particle hit from origin (in this case $\rho = \sqrt{x^2 + y^2}$) or number of entries N corresponding to an event. For example, in [1], maximum value of ρ (ρ_{max}) and total number of hits were used to show a general pattern of decrease in the values of ρ_{max} and N with increasing values of σ_x and σ_y.

In order to investigate into the effects of changing σ_x and σ_y on these parameters *violin* plots have been used. Violin plot is a combination of box plot and Kernel Density Estimation (KDE) plot of the respective histogram. For example, in the figure 5.7a, the box plot is shown at the middle of identical KDE plots of the histogram of ρ on both of its sides. The first point from the bottom of the box plot at the straight line denotes the minimum value, whereas the last point at the top of the straight line is the maximum value. The first edge of the box plot from

the bottom denotes the first quartile (25%), the white point at the middle denotes the median (the second quartile) and the the last edge of the box denotes the third quartile (75%).

In the figure 5.4, the violin plots denote the change of number of entries with the change of σ_x and σ_y for 214036 events(or bunches;1 event=1 bunch). In the figure 5.3, the KDE plots for number of entries corresponding to each value of σ_x resemble Gaussian distribution. Additionally, the distance between every consecutive mean is more than 2 standard deviations.

In the figure 5.4, the violin plots denote the change of number of entries with the change of σ_y. Evidently, the influence of σ_x on number of entries is much higher than that of σ_y. This is a direct consequence of the equation 3.23, which denotes the electric field due to a flat beam.

In both of these plots, the means seem to be falling linearly with the values of σ_x and σ_y. But while the mean of distributions due to varying σ_x show consistent linear decrease with σ_x values, that in case of changing σ_y do not show this behaviour across different samples of the dataset. In other words, the variance of the estimator *mean* is high across different samples of the dataset of varying σ_y. But when the value of σ_x is fixed to 1.0, the means of distribution of both number of entries and ρ_{mean} decreases with the values of σ_y, as shown in figure 5.6.

That is the relationship of number of entries N and ρ_{mean} with σ_x and σ_y is very similar.

Besides, the plots of 5.8 and 5.9 do not show any correlation with ρ_{Max} that was used in the earlier study. The plots 4.17 and 4.17, show clear change of number of entries with the change of beam size parameters. A similar analysis with that of the peaks of these plots (i.e. ρ_{Mode}) show similar pattern as that of ρ_{Mean}. Because all these parameters show similar behaviour with the change of horizontal and vertical beam sizes, only the number of entries have been taken as the parameter that can distinguish values of σ_x. But since this value is a consequence of electric field shown in equation 3.23, only the value of σ_x can be determined from this parameter.

5.3 Principle

The chief idea of machine learning is to predict a vector of outcomes of a number of input variables, given a certain dataset $\mathbb{D}(\mathbf{X},\mathbf{y})$, where $\mathbf{X}$ is the matrix corresponding to an input vector $\mathbf{x}$ and $\mathbf{y}$ is the vector of all the outcomes. The prediction can be denoted to be a parametrized function $\mathbf{f}(\mathbf{X};\boldsymbol{\theta})$: $\mathbf{x} \rightarrow$ y, where θ minimizes a well defined cost function $\mathbf{C}(\mathbf{y}, \mathbf{f}(\mathbf{X};\boldsymbol{\theta}))$.

The dataset $\mathbb{D}(\mathbf{X},\mathbf{y})$ is divided into mutually exclusive $\mathbb{D}(\boldsymbol{X}_{train}, \boldsymbol{y}_{train})$ and $\mathbb{D}(\boldsymbol{X}_{test}, \boldsymbol{y}_{test})$. The first is used for obtaining $\hat{\boldsymbol{\theta}}$=argmin$_\theta$ $\mathbf{C}(\mathbf{y}, \mathbf{f}(\mathbf{X};\boldsymbol{\theta})$ and this $\hat{\boldsymbol{\theta}}$ is used on the training set cost function to obtain output error $E_{out} = \boldsymbol{C}(\boldsymbol{y}_{train}, \boldsymbol{f}(\boldsymbol{X}_{train}; \hat{\boldsymbol{\theta}}))$. The output (validation) error thus expressed by the cost function will most of the time be equal or greater than the input (training) error.

Based on the type of outputs, machine learning algorithms can be divided mainly into two types:

- Regression: Produces continuous output variables. It can also be divided into parametric methods and non-parametric methods. The first one includes

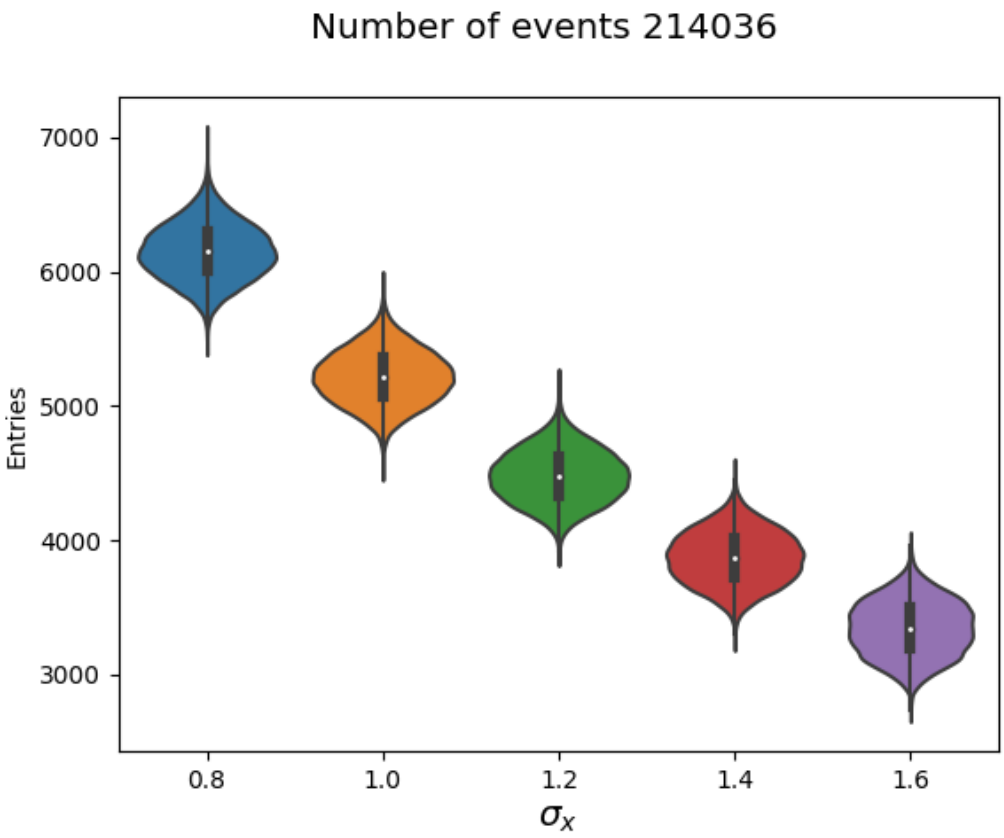


Figure 5.1: Violin plots of number entries per bunch vs σ_x

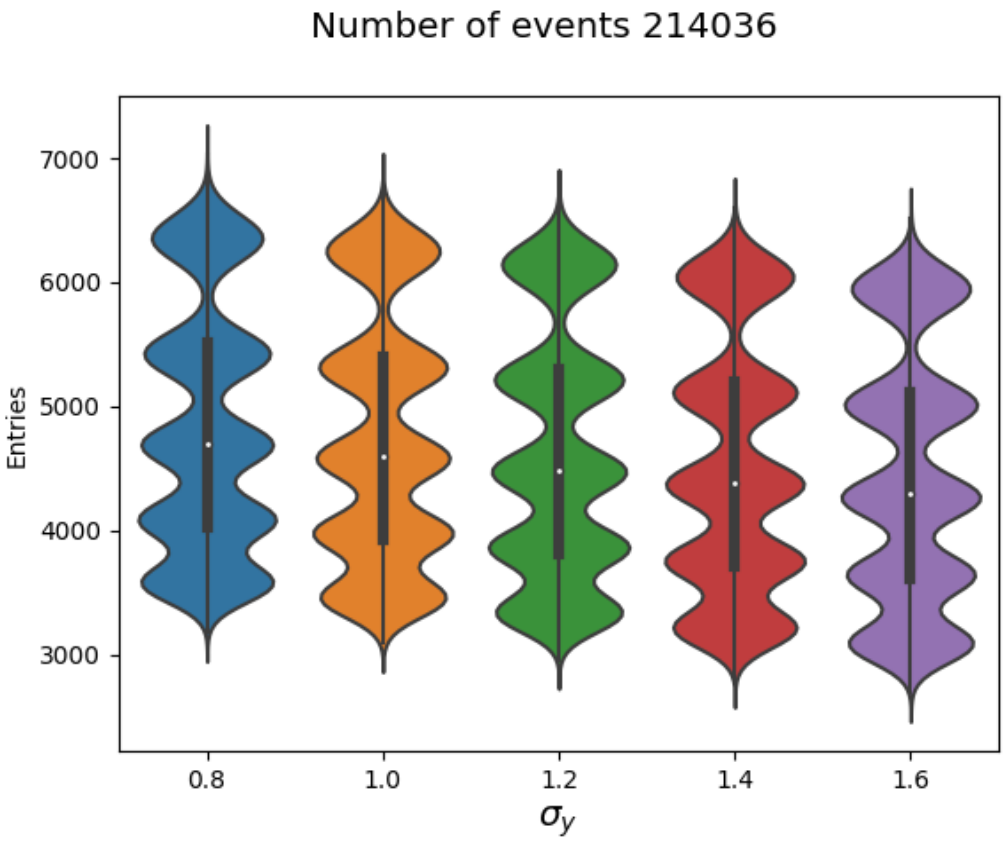


Figure 5.2: Violin plots of number entries per bunch vs σ_x

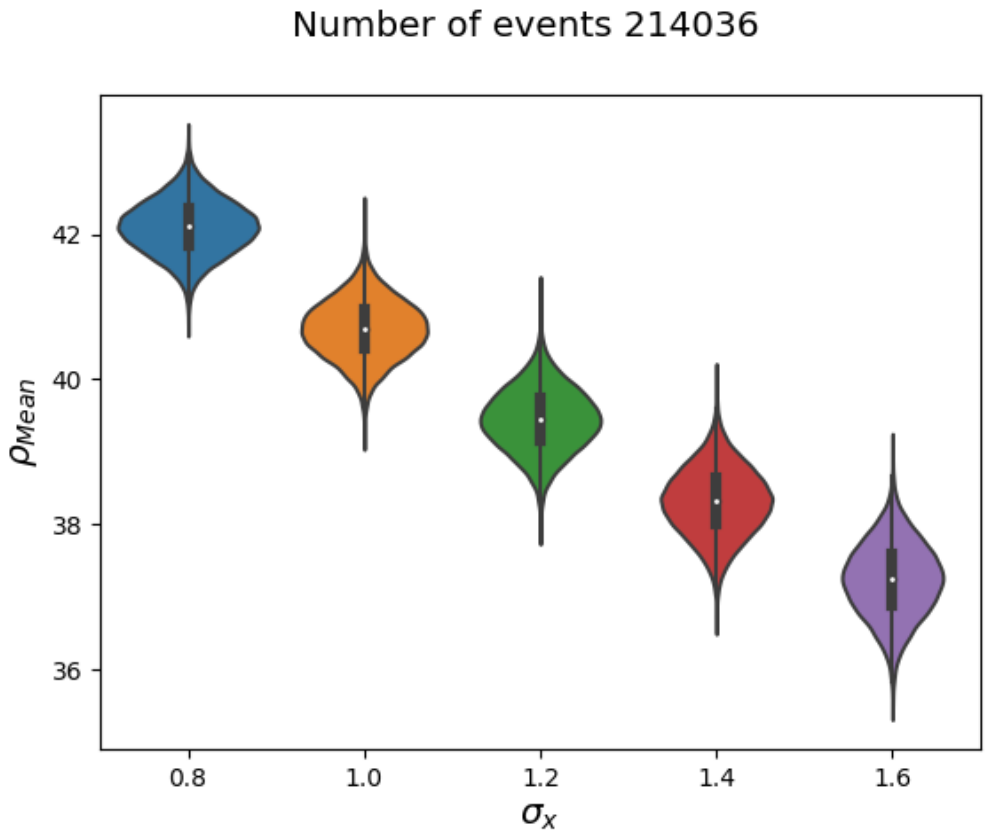


Figure 5.3: Violin plots of mean value of ρ distribution ρ_{Mean} per bunch vs σ_x

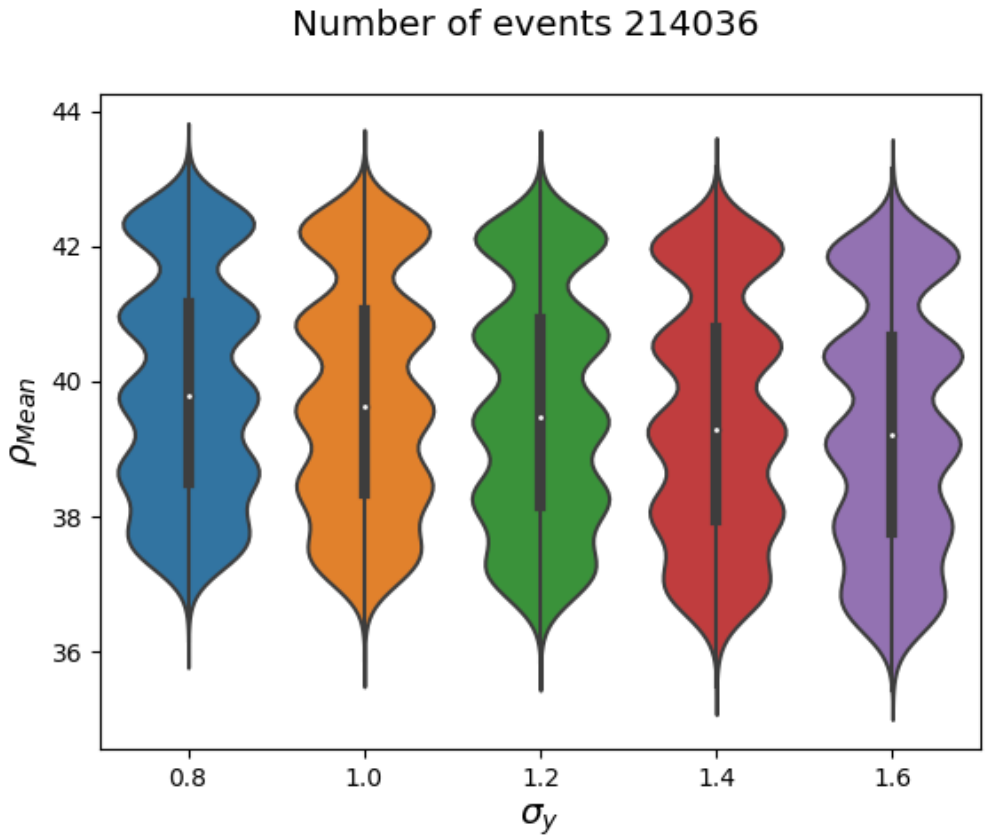


Figure 5.4: Violin plots of mean value of ρ distribution ρ_{Mean} per bunch vs σ_y

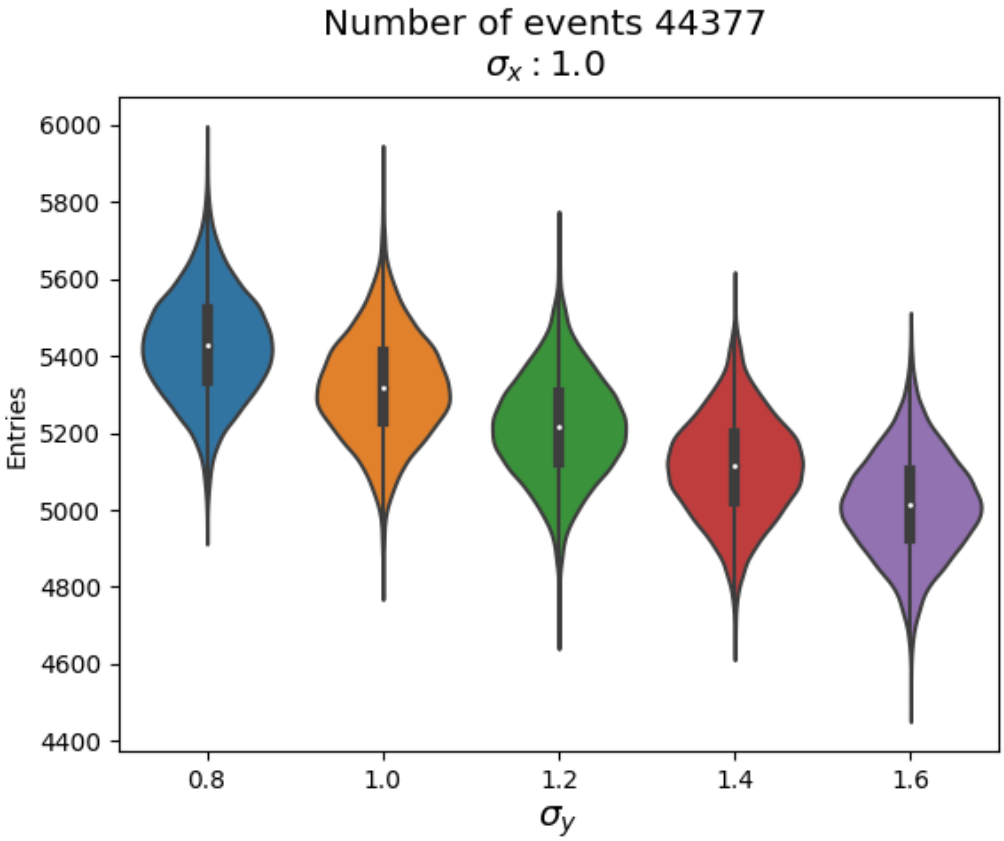


Figure 5.5: Violin plots of number of entries per bunch vs σ_y for σ_x : 1.0

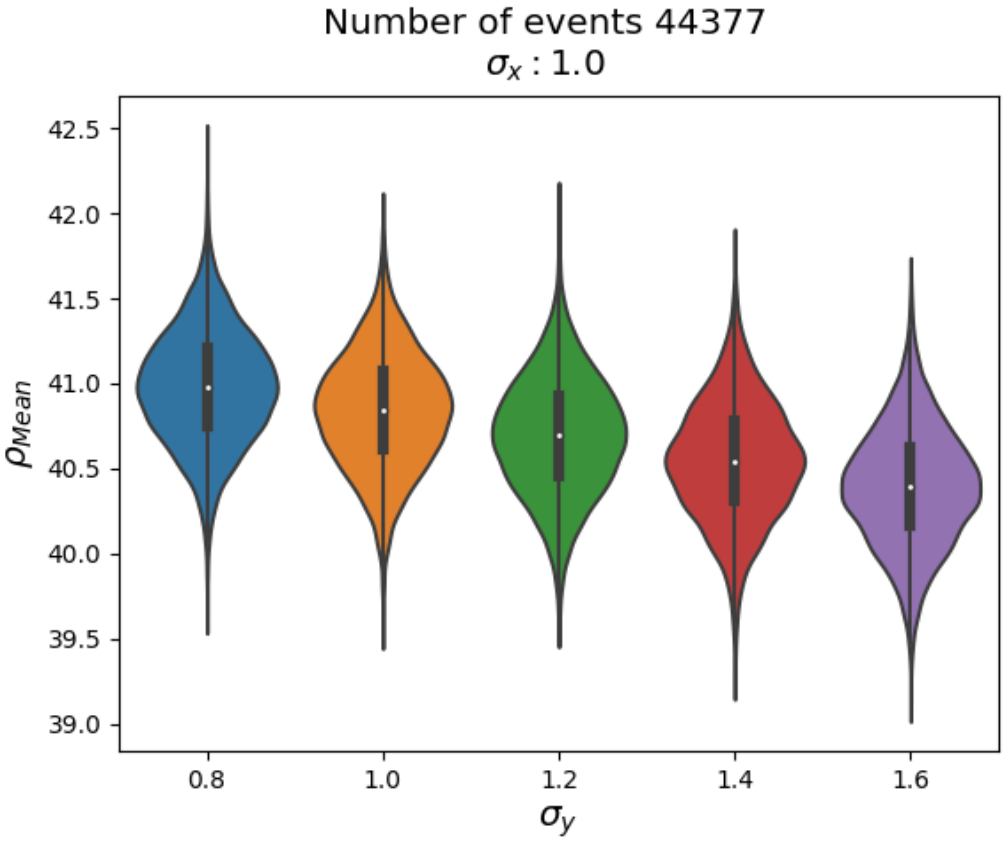
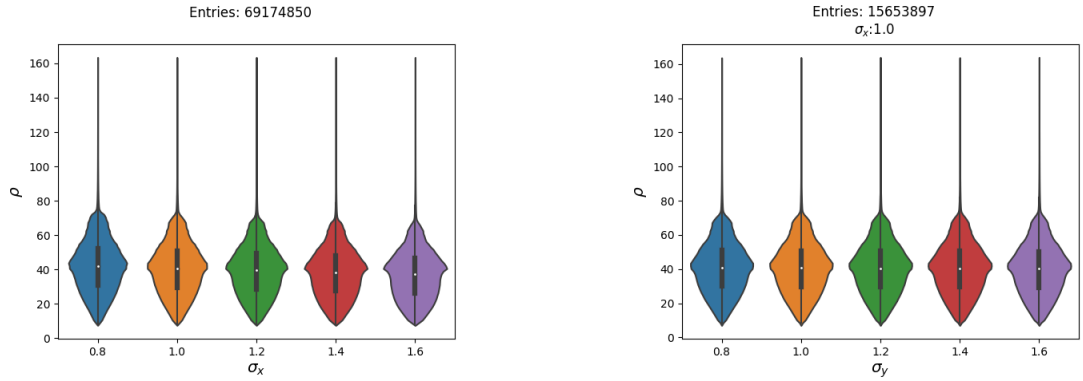


Figure 5.6: Violin plots of number of entries per bunch and ρ_{Mean} for σ_x : 1.0



(a) Violin plots of ρ for σ_x

(b) Violin plots of ρ for σ_y

Figure 5.7: Violin plots of ρ

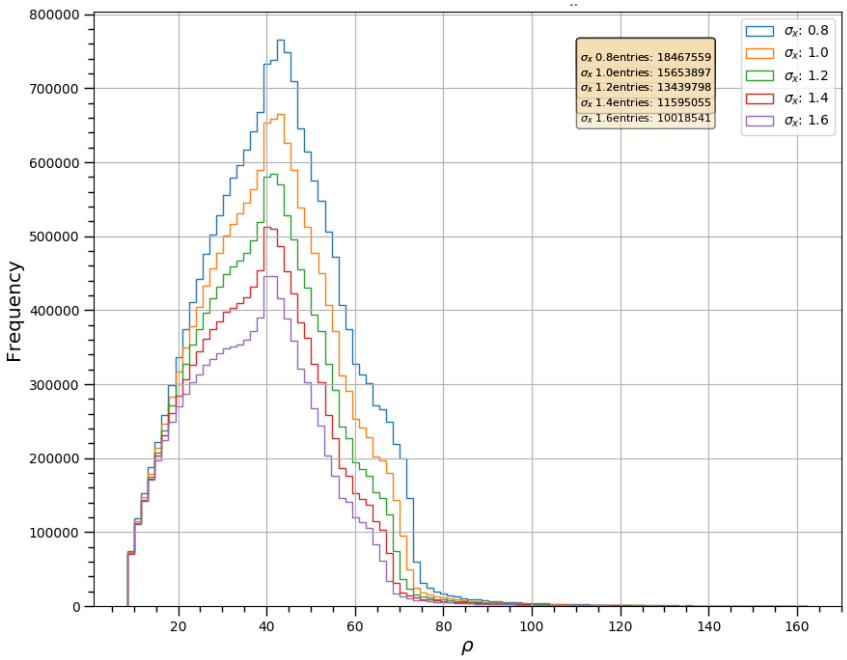


Figure 5.8: plot of $\rho = \sqrt{x^2 + y^2}$ for changing values of σ_x (without any cut)

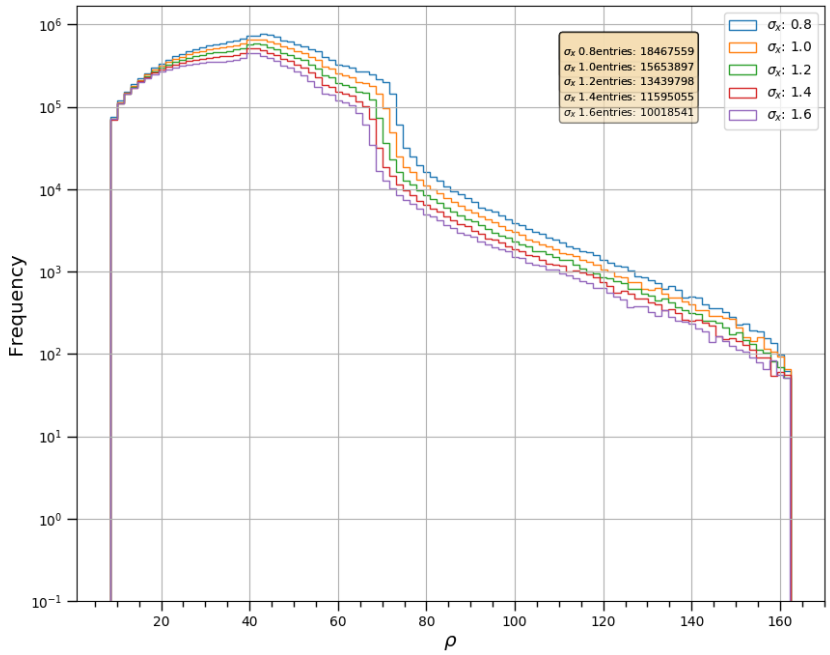


Figure 5.9: Log plot of $\rho = \sqrt{x^2 + y^2}$ for changing values of σ_x (without any cut)

generalized linear model, neural networks etc. and the latter includes decision trees, k-nearest neighbours etc.

- Classification: Produces discreet output variables. It can also be divided into generative and discriminative methods. The first includes kernel density estimation, Gaussian mixture models etc. whereas the latter includes artificial neural network, boosted decision tree, support vector machine etc.

The relevant concepts [31],[32],[33],[34] are summarised in the remainder of this section.

5.4 Bias-Variance Tradeoff

It is not enough to have low error from training datasets only. Performance of a machine learning algorithm is judged, based on its performance over data that are independent from the training datasets. The field of statistical learning suggests an educated way to judge and improve the algorithm performance by means of qualitative assessment of quantities namely, bias and variance.

Bias is said to be the performance of the algorithm in presence of an infinite training data. Variance means how the performance of an algorithm varies over different training datasets. Bias-variance analysis is discussed here based on a regression setting.

In the training dataset $\mathbb{D} = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}$ each point is $(\boldsymbol{x}_i, y_i)$ drawn from a probability distribution $\mathbb{P}(\boldsymbol{X}, \boldsymbol{Y})$ and the input vector $\boldsymbol{x}_i$ corresponds label $y_i \in \mathbb{R}$.

Now we consider test points $(\boldsymbol{x}, y)$ and formulate quantities relevant to evaluation of the machine learning algorithm. Because same test label y can be produced by multiple test input vectors, expected test label $\bar{y}$ is calculated for a given test input vector $\boldsymbol{x}$.

$$\bar{y} = E_{y|\boldsymbol{x}}(\boldsymbol{Y}) = \int_y yP(y|\boldsymbol{x})dy \tag{5.1}$$

A machine learning algorithm of our choice $\mathbb{A}$ can be applied on the training dataset $\mathbb{D}$ so that it learns a hypothesis function h_D i.e. $h_D = \mathbb{A}(\mathbb{D})$. This output hypothesis function h_D is fixed over the dataset $\mathbb{D}$ and as such expected test squared error for a fixed h_D over the dataset can be obtained as:

$$E_{(x,y)\sim\mathbb{P}}[(h_D(\boldsymbol{x}) - y)^2] = \int_x \int_y (h_D(\boldsymbol{x}) - y)^2 P(\boldsymbol{x}, y)dyd\boldsymbol{x} \tag{5.2}$$

Different hypothesis functions h_D will be produced for different training datasets. Every dataset is formed by drawing n points from a distribution $\mathbb{P}^n(\boldsymbol{X}, \boldsymbol{Y})$. Thus, the expected output function for a fixed algorithm $\mathbb{A}$ can be defined as:

$$\bar{h} = E_{\mathbb{D}\sim\mathbb{P}^n}[h_D] = \int_{\mathbb{D}} h_D P(\mathbb{D})d\mathbb{D} \tag{5.3}$$

Finally, for a given algorithm $\mathbb{A}$, the expected test error for all hypothesis functions will be:

$$E_{\substack{(x,y)\sim\mathbb{P} \\ \mathbb{D}\sim\mathbb{P}^n}}[(h_D(\boldsymbol{x}) - y)^2] = \int_{\mathbb{D}} \int_x \int_y (h_D(\boldsymbol{x}) - y)^2 P(\boldsymbol{x}, y)P(\mathbb{D})dyd\boldsymbol{x}d\mathbb{D} \tag{5.4}$$

Equation 5.4 gives us an analyticsl expression for quantifying the expected test error for an algorithm $\mathbb{A}$. This equation can now be decomposed into a bias term, a variance term and a noise term.

$$\begin{aligned} E_{\boldsymbol{x},y,\mathbb{D}}[(h_D(\boldsymbol{x}) - y)^2] = E_{\boldsymbol{x},y,\mathbb{D}}[(h_D((\boldsymbol{x}) - \bar{h}) + (\bar{h} - y))^2] \\ = E_{\boldsymbol{x},\mathbb{D}}[(h_D(\boldsymbol{x}) - \bar{h}(\boldsymbol{x}))^2] + 2\ E_{\boldsymbol{x},y,\mathbb{D}}[((h_D(\boldsymbol{x}) - \bar{h}(\boldsymbol{x}))(\bar{h}(\boldsymbol{x}) - y))] \\ + E_{\boldsymbol{x},y}[(\bar{h}(\boldsymbol{x}) - y)^2] \end{aligned} \tag{5.5}$$

The 2nd term of equation 5.5 can be evaluated to be 0.

$$E_{\boldsymbol{x},y,\mathbb{D}}[((h_D(\boldsymbol{x}) - \bar{h}(\boldsymbol{x}))(\bar{h}(\boldsymbol{x}) - y))] = E_{\boldsymbol{x},y}[E_{\mathbb{D}}[((h_D(\boldsymbol{x}) - \bar{h}(\boldsymbol{x}))]](\bar{h}(\boldsymbol{x}) - y) \tag{5.6}$$

$$= E_{\boldsymbol{x},y}[E_{\mathbb{D}}[((h_D(\boldsymbol{x})] - \bar{h}(\boldsymbol{x}))](\bar{h}(\boldsymbol{x}) - y) \tag{5.7}$$

$$= E_{\boldsymbol{x},y}[\bar{h}(\boldsymbol{x}) - \bar{h}(\boldsymbol{x}))](\bar{h}(\boldsymbol{x}) - y) = 0 \tag{5.8}$$

Therefore equation 5.5 gets reduced to

$$E_{\boldsymbol{x},y,\mathbb{D}}[(h_D(\boldsymbol{x}) - y)^2] = E_{\boldsymbol{x},\mathbb{D}}[(h_D(\boldsymbol{x}) - \bar{h}(\boldsymbol{x}))^2] + E_{\boldsymbol{x},y}[(\bar{h}(\boldsymbol{x}) - y)^2]$$

The second term of the above equation can be broken down into,

$$E_{\boldsymbol{x},y}[(\bar{h}(\boldsymbol{x}) - y)^2] = E_{\boldsymbol{x},y}[(\bar{h}(\boldsymbol{x}) - \bar{y}(\boldsymbol{x}) + (\bar{y}(\boldsymbol{x}) - y))^2 \tag{5.9}$$

$$\begin{aligned} &= E_{\boldsymbol{x},y}[((\bar{y}(\boldsymbol{x}) - y(\boldsymbol{x}))^2] + E_{\boldsymbol{x}}[((\bar{h}(\boldsymbol{x}) - \bar{y}(\boldsymbol{x}))^2] \\ &\quad +2\ E_{\boldsymbol{x},y}[((\bar{y}(\boldsymbol{x}) - y(\boldsymbol{x}))((\bar{h}(\boldsymbol{x}) - \bar{y}(\boldsymbol{x}))] \end{aligned} \tag{5.10}$$

The third term of 5.9 can be

$$E_{\boldsymbol{x},y}[(\bar{y}(\boldsymbol{x}) - y(\boldsymbol{x}))(\bar{h}(\boldsymbol{x}) - \bar{y}(\boldsymbol{x}))] = E_{\boldsymbol{x}}[E_{y|\boldsymbol{x}}[(\bar{y}(\boldsymbol{x}) - y(\boldsymbol{x}))](\bar{h}(\boldsymbol{x}) - \bar{y}(\boldsymbol{x}))] \tag{5.11}$$

$$= E_{\boldsymbol{x}}[(\bar{y}(\boldsymbol{x}) - E_{y|\boldsymbol{x}}[y(\boldsymbol{x})])(\bar{h}(\boldsymbol{x}) - \bar{y}(\boldsymbol{x}))] \tag{5.12}$$

$$= 0 \tag{5.13}$$

since $E_{y|\boldsymbol{x}}[y(\boldsymbol{x})] = \bar{y}(\boldsymbol{x})$.

Finally, we obtain,

$$E_{\boldsymbol{x},y,\mathbb{D}}[(h_D(\boldsymbol{x}) - y)^2] = E_{\boldsymbol{x},\mathbb{D}}[(h_D(\boldsymbol{x}) - \bar{h}(\boldsymbol{x}))^2] + E_{\boldsymbol{x},y}[(y - \bar{y}(\boldsymbol{x}))^2] + E_{\boldsymbol{x}}[(\bar{h}(\boldsymbol{x}) - \bar{y}(\boldsymbol{x}))^2] \tag{5.14}$$

The first three terms of equation 5.14 correspond to variance, noise and squared bias respectively. When evaluated on test data, the first term shows, how much the function that learned from training data, varies between different training datasets. If the training datasets are too distinct from one another, the flactuation between datasets is high and so variance will also be high.

The second term denotes noise. The expected label is much different from the chosen label means the data does not have the label it is expected to have. There is no way to compensate for this because this is the characteristic of the data.

The third term denotes the squared bias. This denotes how the learned function differs from the actual labels in an infinite data limit. It arises because the learned function is *biased* towards a particular solution, which is different from the solution inherent to the data.

5.5 Gradient Descent and Newton's Method

Most machine learning methods come with the same ingredients the input $\boldsymbol{X}$, model $g(\boldsymbol{\theta})$ and the cost function $C(\boldsymbol{X}; g(\boldsymbol{\theta}))$. The prediction is made by minimizing $C(X; g(\boldsymbol{\theta})$ for learned parameters $\boldsymbol{\theta}$.

The minimization is made by updating $\boldsymbol{\theta}$ as long as the the gradient of the cost function is large and negative.

In this context, minimizing the cost function is fundamentally minimization of error. Therefore, error $E(\boldsymbol{\theta})$ is equal to the cost function $C(\boldsymbol{X}; g(\boldsymbol{\theta}))$. For example, in case of linear regression this is the mean square error. Error $E(\boldsymbol{\theta})$ is the sum of errors corresponding to all of the n data points.

$$E(\boldsymbol{\theta}) = \sum_{i=1}^{n} e_i(\boldsymbol{x}_i, \boldsymbol{\theta}) \tag{5.15}$$

The main idea of gradient descent is to update the parameters and then evaluate the cost function iteratively.

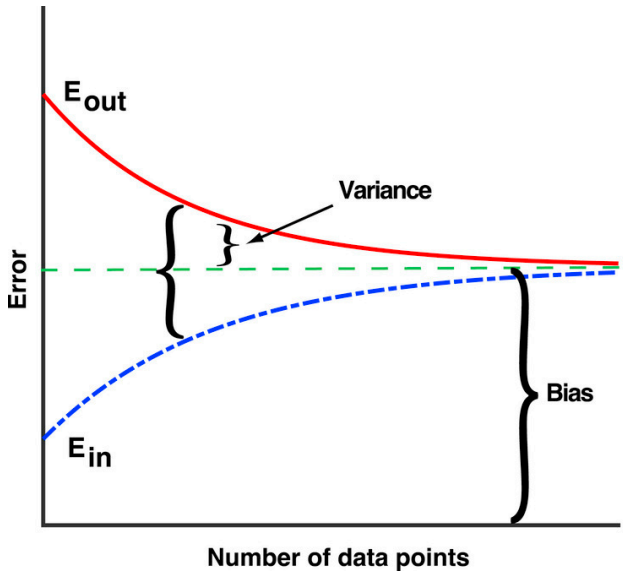


Figure 5.10: Bias is error in presence of infinite data points [31]

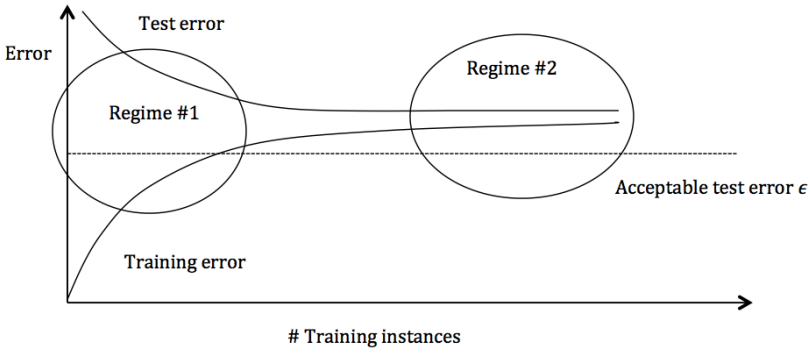


Figure 5.11: Recognizing the causes of error [32]

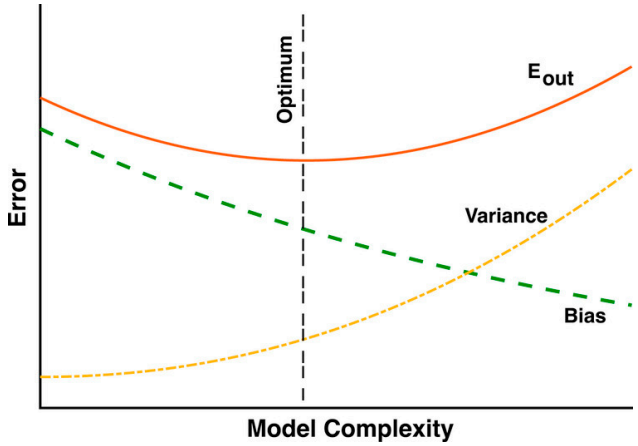


Figure 5.12: Error vs Model Complexity: An optimum balance of bias and variance leads to low generalization error [31]

$$\boldsymbol{v}_t = \eta_t \nabla_\theta E(\boldsymbol{\theta}_t) \tag{5.16}$$
$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \boldsymbol{v}_t \tag{5.17}$$

where $\nabla_\theta E(\boldsymbol{\theta}_t)$ is the gradient of $E(\boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$, η_t is the learning rate which controls the step to be taken in the direction of gradient at a step t.

According to Newton's method, step $\boldsymbol{v}$ is chosen so that the second order Taylor expansion of $E(\boldsymbol{\theta})$ evaluated about $\boldsymbol{v}$ is minimized,

$$E(\boldsymbol{\theta} + \boldsymbol{v}) \approx E(\boldsymbol{\theta}) + \nabla_\theta E(\boldsymbol{\theta})\boldsymbol{v} + \frac{1}{2}\boldsymbol{v}^T H(\boldsymbol{\theta})\boldsymbol{v} \tag{5.18}$$

where $H(\boldsymbol{\theta})$ is the Hessian matrix of second derivatives. The above equation equation is minimized for an optimal value $\boldsymbol{v} = \boldsymbol{v}_{opt}$. i.e. $\nabla_\theta E(\boldsymbol{\theta} + \boldsymbol{v}_{opt}) = 0$. So we get,

$$0 = \nabla_\theta E(\theta) + H(\boldsymbol{\theta})\boldsymbol{v}_{opt} \tag{5.19}$$

Rearranging 5.19, we get the following,

$$\boldsymbol{v}_t = H^{-1}(\boldsymbol{\theta}_t)\nabla_\theta E(\boldsymbol{\theta}_t) \tag{5.20}$$
$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \boldsymbol{v}_t \tag{5.21}$$

For one dimension equation 5.18 is,

$$E(\theta + v) \approx E(\theta_c) + \delta_\theta E(\theta)v + \frac{1}{2}\delta_\theta^2 E(\theta)v^2 \tag{5.22}$$

Differentiating with respect to v and evaluating for $\theta_{min} = \theta - v$,

$$\theta_{min} = \theta - [\delta_\theta^2 E(\theta)]^{-1}\delta_\theta E(\theta) \tag{5.23}$$

Comparing with equation 5.16,

$$\eta_{opt} = [\delta_\theta^2 E(\theta)]^{-1} \tag{5.24}$$

From the above calculations, it can be seen that Newton's method updates the learning parameter at every step by calculating the Hessian matrix. In other words, if the curvature is very steep the learning parameter is adapted to take a small value and if the curvature is small the learning parameter takes a large value. Gradient descent (GD) is inspired by Newton's method, but instead of computing the Hessian matrix and updating the learning parameter at every step, GD keeps the value of learning rate constant. Given the extremely high number of parameters in machine learning models, this is done in order to reduce the computational expenses required to calculate the Hessian matrix of n^2 elements for n parameters.

5.6 Linear Regression

Linear regression is based on the following three assumptions:

- The output labels are continuous real numbers i.e. $y_i \in \mathbb{R}$.

- The input vector $\boldsymbol{x}$ has a linear relationship with the output y.
- The noise of the output is a zero mean Gaussian. It can be viewed as a direct consequence of the central limit theorem.

The last two assumptions can be expressed as:

$$y_i = \boldsymbol{w}^T \boldsymbol{x}_i + \epsilon_i \tag{5.25}$$

where, ϵ_i denotes Gaussian noise or, $\epsilon_i \sim N(0, \sigma^2)$. In other words, y_i is a Gaussian with a mean of $\boldsymbol{w}^T \boldsymbol{x}_i$ and a variance of σ^2 i.e. $y_i|\boldsymbol{x}_i \sim N(\boldsymbol{w}^T \boldsymbol{x}, \sigma^2)$ and $\boldsymbol{w}$, the parameter of the model, is the slope. Input vector and parameter vector corresponding to one observation and p features are $\boldsymbol{x_i}$ and $\boldsymbol{w}$ respectively.

Baased on the idea of how much probable it is for an input vector $\boldsymbol{x_i}$ to produce an output y_i given a weight vector $\boldsymbol{w}$, the probability distribution function of the model can be expressed as:

$$P(y_i|\boldsymbol{x}_i) = \frac{1}{2\pi\sigma^2} exp\left(-\frac{\boldsymbol{w}^T \boldsymbol{x}_i - y_i}{2\sigma^2}\right) \tag{5.26}$$

The model parameter $\boldsymbol{w}$ is estimated by creating a log-profile likelihood function and minimizing it. It is commonly known as maximum likelihood estimation (MLE).

$$\begin{aligned}
\boldsymbol{w} &= \underset{\boldsymbol{w}\in\mathbb{R}^p}{argmax}\ P(y_1, \boldsymbol{x_1}, \ldots, y_n, \boldsymbol{x_n}|\boldsymbol{w}) && (5.27)\\
&= \underset{\boldsymbol{w}\in\mathbb{R}^p}{argmax}\ \prod_{i=1}^{n} P(y_i, \boldsymbol{x_i}|\boldsymbol{w}) && (5.28)\\
&= \underset{\boldsymbol{w}\in\mathbb{R}^p}{argmax}\ \prod_{i=1}^{n} P(y_i|\boldsymbol{x_i}, \boldsymbol{w}) P(\boldsymbol{x_i}|\boldsymbol{w}) && (5.29)\\
&= \underset{\boldsymbol{w}\in\mathbb{R}^p}{argmax}\ \prod_{i=1}^{n} P(y_i|\boldsymbol{x_i}, \boldsymbol{w}) P(\boldsymbol{x_i}) && (5.30)\\
&= \underset{\boldsymbol{w}\in\mathbb{R}^p}{argmax}\ \prod_{i=1}^{n} P(y_i|\boldsymbol{x_i}, \boldsymbol{w}) && (5.31)\\
&= \underset{\boldsymbol{w}\in\mathbb{R}^p}{argmax}\ \sum_{i=1}^{n} logP(y_i|\boldsymbol{x_i}, \boldsymbol{w}) && (5.32)\\
&= \underset{\boldsymbol{w}\in\mathbb{R}^p}{argmax}\ \sum_{i=1}^{n} log\left(exp\left(-\frac{\boldsymbol{w}^T \boldsymbol{x_i} - y_i}{2\sigma^2}\right)\right) && (5.33)\\
&= \underset{\boldsymbol{w}\in\mathbb{R}^p}{argmax}\ \frac{-1}{2\sigma^2} \sum_{i=1}^{n} (\boldsymbol{w}^T \boldsymbol{x_i} - y_i)^2 && (5.34)\\
&= \underset{\boldsymbol{w}\in\mathbb{R}^p}{argmin}\ \frac{1}{n} \sum_{i=1}^{n} (\boldsymbol{w}^T \boldsymbol{x_i} - y_i)^2 && (5.35)
\end{aligned}$$

Equation 5.25 shows the cost function C in this case is mean square error.

$$C(\boldsymbol{w}) = \frac{1}{n} \sum_{i=1}^{n} (\boldsymbol{w}^T \boldsymbol{x_i} - y_i)^2 \tag{5.36}$$

When the input vectors $\boldsymbol{x_i}$ having n observations for each of the p features, are stacked into a matrix $\boldsymbol{X}$ of dimension $n \times p$, equation 5.25 can be written as,

$$\boldsymbol{w} = \underset{\boldsymbol{w}\in\mathbb{R}^p}{argmin}||\boldsymbol{w}^T\boldsymbol{X} - y_i||_2^2 \tag{5.37}$$

$\boldsymbol{w}$ can be estimated by means of optimization algorithm such as gradient descent or by straightforward differentiation. The latter gives:

$$\boldsymbol{w} = (\boldsymbol{X^T X})^{-1}\boldsymbol{X^T y} \tag{5.38}$$

given $(\boldsymbol{X^T X})$ is invertible, which is the case when rank$(\boldsymbol{X}) = p$ and $n >> p$.

Taking the noise term in equation 5.14 to be σ^2, the expectation values of in-sample (training) $\bar{E}_{in}$ and out of sample (validation) $\bar{E}_{out}$ errors can be derived from 5.14 to get the following equations:

$$\bar{E}_{in} = \sigma^2\left(1 - \frac{p}{n}\right)\bar{E}_{out} = \sigma^2\left(1 + \frac{p}{n}\right) \tag{5.39}$$

Therefore, the generalization error:

$$|\bar{E}_{in} - \bar{E}_{out}| = 2\sigma^2\frac{p}{n} \tag{5.40}$$

Therefore, if $p >> n$ or in the case of high noise, the generalization error would be high.

MLE maximizes the probability for a given parameter $\boldsymbol{w}$ to have to a dataset $\{(\boldsymbol{x_i}, y_i)\}_{i=1}^n$. The opposite scenario, where the probability for a given dataset to have parameter $\boldsymbol{w}$ is maximised, is called maximum a posteriori estimation (MAP). MAP is the basis of regularized linear regression namely, *Ridge* regression. Regularized linear regression enables us to ameliorate the generalization error 5.40, in case of $p \approx n$.

Application

The intercept and slope values obtained from training are 2.45045216 and -0.0002714 respectively. In figure 5.13, the output of linear regression on validation set has been shown. The plots show a Gaussian distribution with peak about the actual value, which are visualized by different colors. Any overlap between these colors show mismatch of prediction with the actual value. It can be seen that out of about 600 labels a maximum overlap of about 10 entries have occurred. Because in the current case the number of inputs n is much higher than the number of regression classes $p = 5$, which correspond to the values of σ_x used for training, according to equation 5.40, the generalization error is very low.

The distributions of the errors of the prediction of the factors from linear regression are shown in figures 5.14 and 5.15. Assuming the acceptable tolerance of error of each of the factors to be 0.05, the probability that the error e lies within the range $[-0.05, 0.05]$ is denoted by $P(-0.05 \leq e \leq 0.05)$. These values for all the factors are listed in the table 5.3.

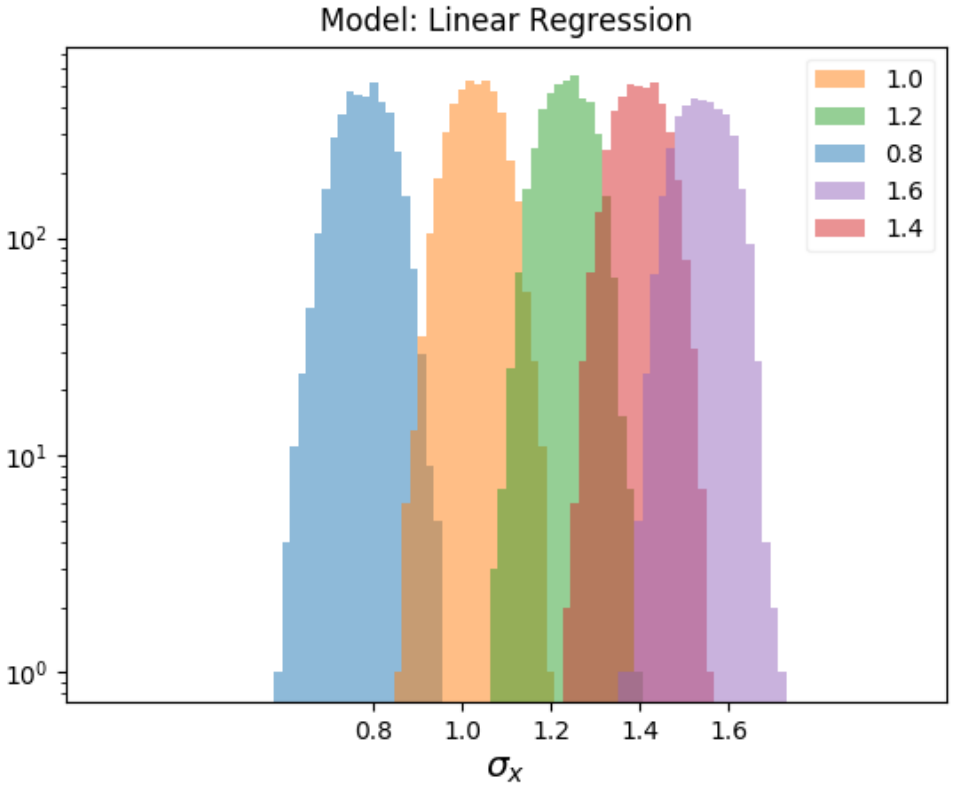


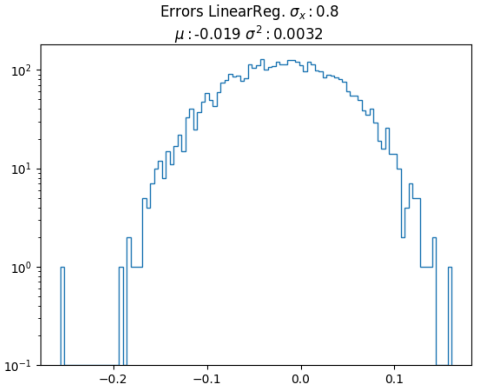
Figure 5.13: Prediction of Linear regression on validation set

5.7 Ridge Regression

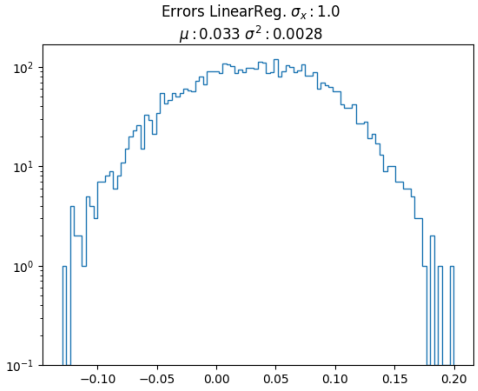
Estimation by MAP requires an assumption about the probability distribution of the prior $\boldsymbol{w}$. A very natural assumption is a Gaussian with zero mean:

$$P(\boldsymbol{w}) = \frac{1}{2\pi\sigma^2} exp\left(-\frac{\boldsymbol{w}^T\boldsymbol{w}}{2\tau^2}\right) \tag{5.41}$$

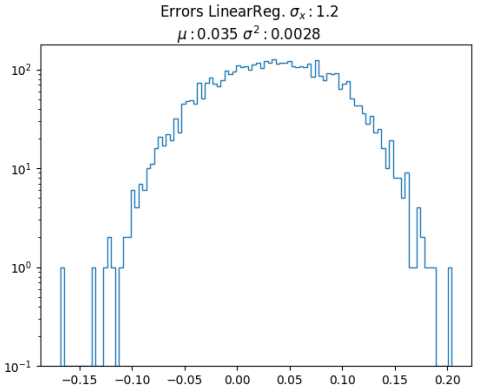
Now the maximisation of the probability distribution $P(\boldsymbol{w}|\boldsymbol{x_1}, y_1, \ldots, \boldsymbol{x_i}, y_i)$ can be calculated in the following method:



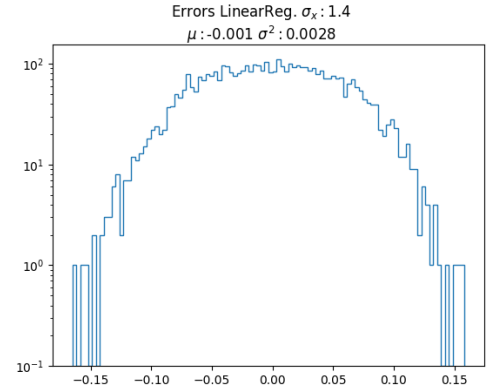
(a) Errors for σ_x factor 0.8



(b) Errors for σ_x factor 1.0



(c) Errors for σ_x factor 1.2



(d) Errors for σ_x factor 1.4

Figure 5.14: Errors for predictions for the factors of σ_x (Linear regression)

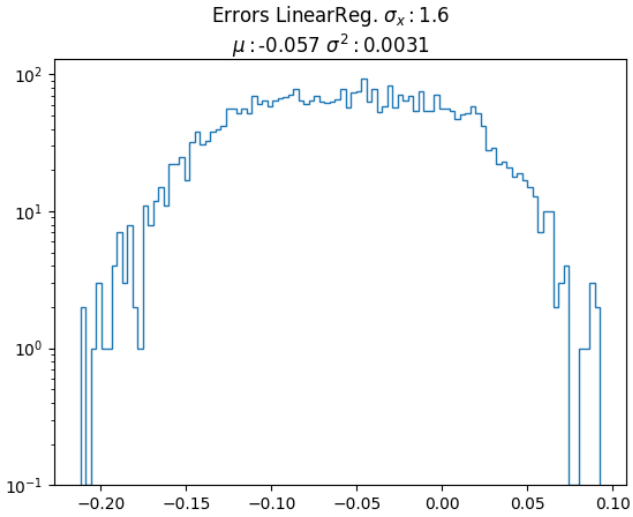


Figure 5.15: Errors for predictions for σ_x factor 1.6 (Linear regression)

$$\boldsymbol{w} = \underset{\boldsymbol{w}\in\mathbb{R}^p}{argmax}\ P(\boldsymbol{w}|\boldsymbol{x_1}, y_1, \ldots, \boldsymbol{x_i}, y_i) \tag{5.42}$$

$$= \underset{\boldsymbol{w}\in\mathbb{R}^p}{argmax}\ \frac{P(\boldsymbol{x_1}, y_1, \ldots, \boldsymbol{x_i}, y_i|\boldsymbol{w})P(\boldsymbol{w})}{P(\boldsymbol{x_1}, y_1, \ldots, \boldsymbol{x_i}, y_i)} \tag{5.43}$$

$$= \underset{\boldsymbol{w}\in\mathbb{R}^p}{argmax}\ P(\boldsymbol{x_1}, y_1, \ldots, \boldsymbol{x_i}, y_i|\boldsymbol{w})P(\boldsymbol{w}) \tag{5.44}$$

$$= \underset{\boldsymbol{w}\in\mathbb{R}^p}{argmax}\ P(\boldsymbol{x_1}, y_1, \ldots, \boldsymbol{x_i}, y_i|\boldsymbol{w})P(\boldsymbol{w}) \tag{5.45}$$

$$= \underset{\boldsymbol{w}\in\mathbb{R}^p}{argmax}\ \left(\prod_{i=1}^{n} P(y_i, \boldsymbol{x}_i|\boldsymbol{w})\right) P(\boldsymbol{w}) \tag{5.46}$$

$$= \underset{\boldsymbol{w}\in\mathbb{R}^p}{argmax}\ \left(\prod_{i=1}^{n} P(y_i|\boldsymbol{x}_i, \boldsymbol{w})\right) P(\boldsymbol{w}) \tag{5.47}$$

$$= \underset{\boldsymbol{w}\in\mathbb{R}^p}{argmax}\ \sum_{i=1}^{n} logP(y_i|\boldsymbol{x}_i, \boldsymbol{w}) + logP(\boldsymbol{w}) \tag{5.48}$$

$$= \underset{\boldsymbol{w}\in\mathbb{R}^p}{argmax}\ \frac{1}{2\sigma^2}\sum_{i=1}^{n}(\boldsymbol{x}_i^T\boldsymbol{w} - y_i)^2 + \frac{1}{2\tau^2}\boldsymbol{w}^T\boldsymbol{w} \tag{5.49}$$

$$= \underset{\boldsymbol{w}\in\mathbb{R}^p}{argmin}\ \frac{1}{n}\sum_{i=1}^{n}(\boldsymbol{x}_i^T\boldsymbol{w} - y_i)^2 + \lambda||\boldsymbol{w}||_2^2 \tag{5.50}$$

where $\lambda = \frac{\sigma^2}{nr^2}$.

The eauation 5.50 is said to have a regularizer defined as the L_2 norm of the parameter vector that is optimized. While the optimization can be carried by gradient descent algorithm, a straightforward differentiation gives the following:

$$\boldsymbol{w}(\lambda) = (\boldsymbol{X}^T\boldsymbol{X} + \lambda I_{p\times p})^{-1}\boldsymbol{X}^T\boldsymbol{y} \tag{5.51}$$

If $\boldsymbol{X}$ is orthogonal,

$$\boldsymbol{w}_{Ridge}(\lambda) = \frac{\boldsymbol{w}_{LR}}{1+\lambda} \tag{5.52}$$

where $\boldsymbol{w}_{LR}$ is the parameter vector obtained by linear regression (sec. 5.6).

5.7.1 Application

Ridge regression does not show much different behaviour as compared to linear regression even with varying values of λ regularizer. The function of this regularizer is to penalize the predictor with less effect on the output values. In other words, in case of multiple inputs into the model, the input with low influence on the output (beam size factor in this case) will be ignored by means of the regularizer.

Because of only the presence of one input parameter, ridge regression in the current setting is not any different from the linear regression. But in presence of multiple input parameters in future studies, the first choice of start can be ridge regression instead of linear regression.

5.8 K-nearest neighbors

K-nearest neighbours carry out prediction based on the assumption that similar input vectors will produce similar labels. A group $S_{\boldsymbol{x}} \subseteq \mathbb{D}$ of k input vectors have the same label, if the distance between the test input vector $\boldsymbol{x}$ and any input vector $\boldsymbol{x}'$ in the dataset $\mathbb{D}$, excluding $S_{\boldsymbol{x}}$, i.e. $\mathbb{D} \setminus S_{\boldsymbol{x}}$, is at least equal to the maximum distance between $\boldsymbol{x}$ and input vectors $\boldsymbol{x}'' \in S_{\boldsymbol{x}}$.

Mathematically speaking, for $S_{\boldsymbol{x}} \subseteq \mathbb{D}$ where $|S_{\boldsymbol{x}}| = k$ and $(\boldsymbol{x}',\, y') \in \mathbb{D} \setminus S_{\boldsymbol{x}}$ and the following condition must be satisfied for a group of same label.

$$dist(\boldsymbol{x}, \boldsymbol{x}') \geq \max_{(\boldsymbol{x}'', y'') \in S_{\boldsymbol{x}}} dist(\boldsymbol{x}, \boldsymbol{x}'') \tag{5.53}$$

After such region is determined, the output of this group highest occurrence of a label within that region.

$$h(\boldsymbol{x}) = mean(\{y'' : (\boldsymbol{x}'', y'') \in S_{\boldsymbol{x}}\}) \tag{5.54}$$

The distance as shown in equation 5.53 is most commonly chosen to be the Minnkowski distance. The distance between two vectors $\boldsymbol{x}$ and $\boldsymbol{z}$ of dimenstion p, is defined as:

$$dist(\boldsymbol{x}, \boldsymbol{z}) = \left(\sum_{i=1}^{p} |x_i - z_i|^r \right)^{\frac{1}{r}} \tag{5.55}$$

During training the distance has to be calculated between every test data point and all points in the dataset. This process is computationally expensive. Higher accuracy is possible when number of training datapoints n is large. In order to improve the speed of calculation, the idea of k-Dimensional tree is introduced into the concept of k-nearesr neighbours.

5.9 k-Dimensional Trees

A better approach to proceed with the knn algorithm is to first divide the dataset into two partitions and look for k-nearest neighbours only at the partition where the test data is located, because any other point inside other partitions are farther than the any point within the partition the test point is in.

Let us assume when the dataset is divided into two partitions, the distance between the test point $\boldsymbol{x}_t$ and another point x at the other partition is $d = d_1 + d_2$, where d_1 is the part of the distance inside the partition with test point, $\boldsymbol{x}_t$ and d_2 is the distance within the other. If d_w is the distance between x_t and the partition wall, $d_1 \geq d_w$

$$\begin{aligned} d(\boldsymbol{x}_t, \boldsymbol{x}) &= d_1 + d_2 && (5.56)\\ &\geq d_w + d_2 && (5.57)\\ &\geq d_w && (5.58) \end{aligned}$$

showing that any point beyond the partition with the test point is farther than the k-nearest neighbours. It essentially excludes the need to compute distance for points beyond the partition, thereby reducing the amount of computation.

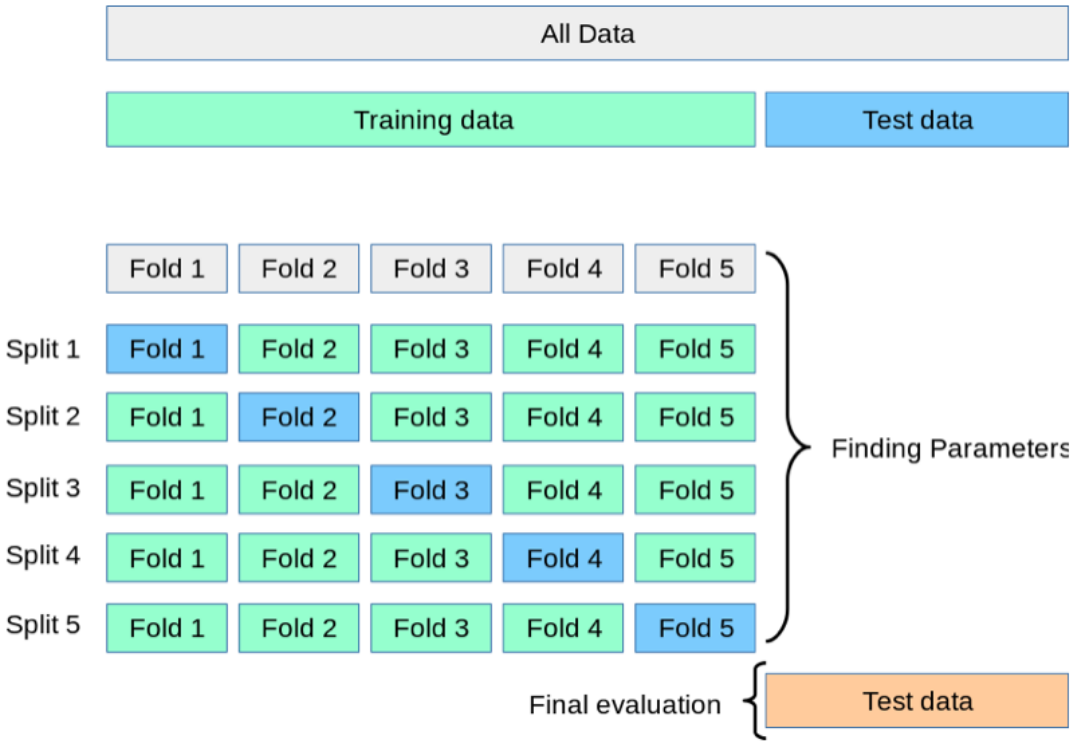


Figure 5.16: Method of cross-validation [35]

At the beginning, the data points are divided into half about their median. The data points larger than the median are sent to one node and the ones smaller than the median are sent to the other.The process is repeated until the last node is reached.

There might be case when the test point is very close to a partition. Therefore, it is required to search if any other node at the leaves can have nearest neighbour. *Tree pruning* will help us assign the particle to the best possible node. This is done by comparing the aforementioned d_w with the maximum distance of the nearest neighbour inside the partition of the test point, d_{nn}^{max}. If $d_w \geq d_{nn}^{max}$, the test point stays at the same node. In other words, it draws a circle of radius d_{nn}^{max} and if the circle intersects a partition, the point gets one level up and performs the same task and prunes parts of the trees, the partition of which is not intersected by the respective radius.

In the current study, the optimum value of k or the number of neighbours, has been chosen by means of a technique known as cross validation (figure 5.16). In this process, the training data is split into multiple *fold*s. In the first iteration (or *split*), one of the say 5 folds, is used as validation and the rest are used for training. The value of error is computed from the validation fold. The process is repeated by changing the validation/training sets repeatedly until all folds have been used as vaidation/training sets. The value of errors are noted for all these cases and the average of all these gives the *Cross-validate*d (CV) errors .

The CV errors are computed for different values of k and the k value which produced the lowest CV error, was chosen. In figure 5.17, the lowest error was produced for k=10. Even though the error keeps falling beyond this value, the change is not significant and k=10 has been chosen for the best computational performance.

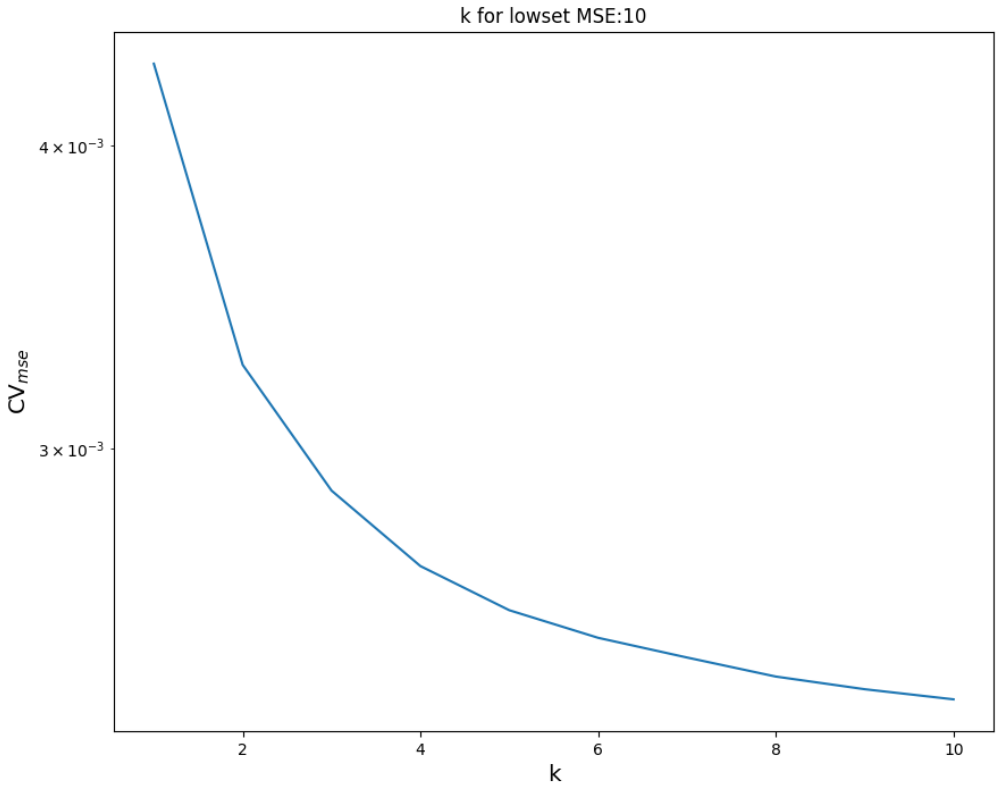


Figure 5.17: Determining k value using cross validation

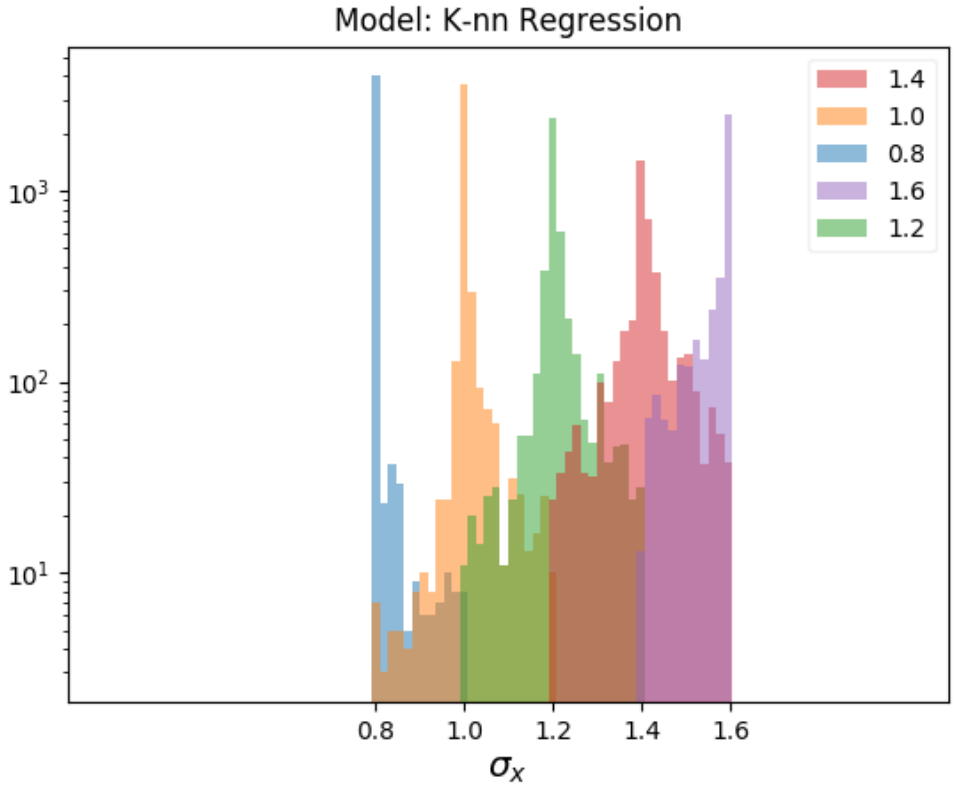


Figure 5.18: Prediction of k-nn with kD trees

5.9.1 Application

With very sharp peaks, K-D trees in figure 5.18 show a superior performance compared to the linear regression. The plots of the errors of the beam size factors, have very sharp peaks close to 0. The probability $P(-0.05 \leq e \leq 0.05)$ for k-D trees are shown in table 5.3.

5.10 Decision Trees

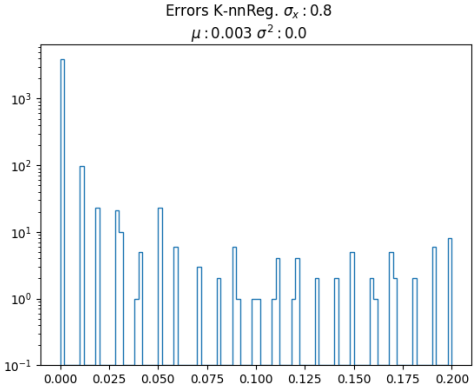
The goal of decision tree is to recursively divide the region of training data into multiple regions of pure labels. Every step of this recursion consists of separating the data based on a simple threshold $\geq t$. This threshold is determined chosen at every step of recursion so that the resulting node is purer than the former. Finally, the split is carried out so that the final leaves contain only elements of a single label.

Advantage of decision tree over k-nn is that it does not require to store the training dataset but only the tree structure. Besides they are faster because the inputs just need to descend down the tree without caring about the neighbours.

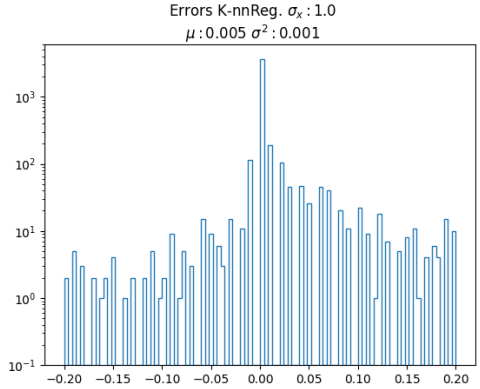
The purity mentioned above is determined by Gini impurity or entropy for classification trees or mean square errors in case of regression trees.

For a dataset $S = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}$, and $y_i \in \{1, \ldots, c\}$ where c is the number of classes, the probability of picking up a certain label k can be denoted as $p_k = \frac{|S_k|}{S}$, where $S_k \subseteq S$, $S_k = \{(\boldsymbol{x}, y) : y = k\}$ and $S = S_1 \cup \cdots \cup S_c$

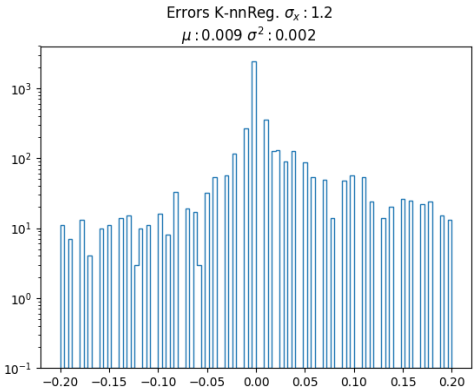
Gini impurity of a leaf, $G(S)$ is defined as:



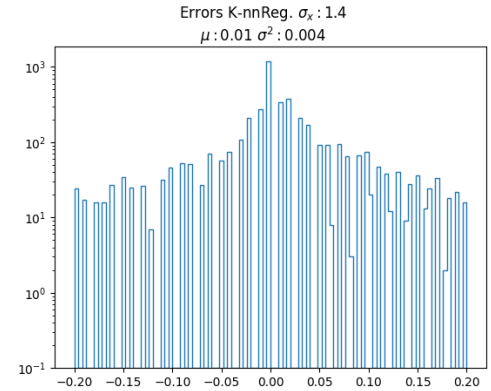
(a) Errors for σ_x factor 0.8



(b) Errors for σ_x factor 1.0



(c) Errors for σ_x factor 1.2



(d) Errors for σ_x factor 1.4

Figure 5.19: Errors for predictions for the factors of σ_x (k-Dimensional trees)

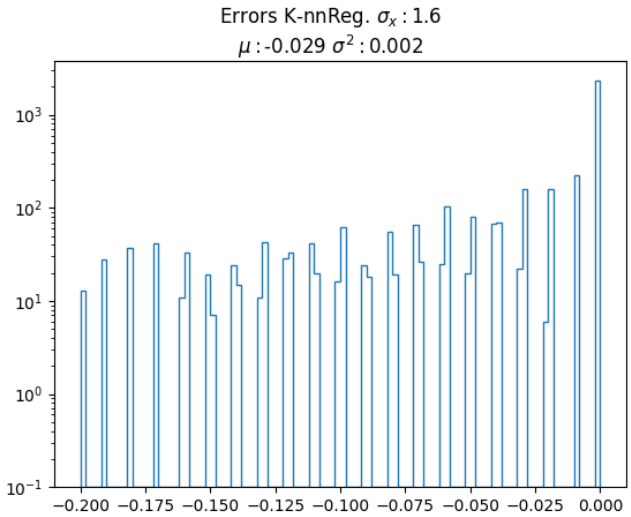


Figure 5.20: Errors for predictions for σ_x factor 1.6 (k-Dimensional trees)

$$G(S) = \sum_{k=1}^{c} p_k(1 - p_k) \tag{5.59}$$

From equation 5.59, it can be seen that, when one leave is pure i.e. $p_k = 1$, the corresponding Gini impurity is 0. If a tree has two leaves, the maximum impurity corresponding to one leaf can therefore be 0.5.

In a tree, if the dataset that goes to the left leaves is denoted by S_L and that to the right by S_R, where $S = S_L \cup S_R$, S being the input dataset and $S_L \cap S_R = \phi$, the Gini impurity of that tree $G^T(S)$ can be expressed as:

$$G^T(S) = \frac{|S_L|}{|S|} G^T(S_L) + \frac{|S_R|}{|S|} G^T(S_R) \tag{5.60}$$

An alternative to Gini impurity is cross-entropy, which for a leaf, can be defined as:

$$D = -\sum_{k=1}^{c} p_k log p_k \tag{5.61}$$

It can be shown that, D from equation 5.61, will be 0, if p_k is either 0 or 1. That is, entropy will be minimum if the leaf includes only one class.

Decision trees can also be used for regression. The splits correspond to the creation of non overlapping regions $R_1, \ldots, R_j$. Each of these regions will produce the mean of all training data inside that region as the output when any test point falls into that region.

The regions are specified by partitioned regions inside boxes and they are created by minimizing the following:

$$\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \bar{y}_{R_j})^2 \tag{5.62}$$

where, $\bar{y}_{R_j}$ is the label averaged from the training labels in R_j. Because looking for the all the square errors for all possible combinations of regions and trying to minimize the error is very computationally expensive. That is. the best split is made at each particular step. In this, threshold value s and input value corresponding to the a node X_j, are selected so the the square error is minimized. One leaf can be denoted by a region $R_1(j, s) = \{X | X_j < s\}$, and the other by $R_2(j, s) = \{X | X_j \geq s\}$, for input value X. In regression setting, the values of (j, s) are fixed so that the following value is the least.

$$\sum_{i:x_i \in R_1(j,s)} (y_i - \bar{y}_{R_1})^2 + \sum_{i:x_i \in R_2(j,s)} (y_i - \bar{y}_{R_2})^2 \tag{5.63}$$

where $\bar{y}_{R_1}$ and $\bar{y}_{R_2}$ are the mean predictions of all the training input in these two regions.

5.11 Random Forest

The purpose of decision tree is to create pure leaves at the last node. This behaviour makes decision tree predictions data specific. In other words, if a dataset is split

into half and fed into decision tree, it will predict differently for both the halves. This is called high variance and one way random forest eradicates this weakness of decision tree, is called bagging.

Bagging is the abbreviation of two: Bootstrapping and Aggregating. The first step of bootstrapping is to create multiple datasets $\mathbb{D}^{*(1)}, \ldots, \mathbb{D}^{*(B)}$ by sampling n points with replacement from dataset $\mathbb{D}$. The idea is that, instead of obtaining multiple new datasets, distinct datasets are obtained by repeatedly sampling from the original dataset. Bickel and Friedman and Singh have shown that he bootstrap estimator is always close to the estimator obtained from the population. The estimators obtained from these bootstrapped samples are averaged to obtain an aggregate of the predictions.

For n independent observations, $Z_1, \ldots, Z_n$, with a variance σ^2, the variance of the mean $\bar{Z}$ of the observations is σ^2/n. That is, averaging a set of observations reduces variance. This is why, a *bagged* decision tree has a lower variance compared to the decision tree without bagging.

If the predictions from B different training datasets are denoted by $\hat{f}^1(x), \ldots, \hat{f}^B(x)$, the average of them will have a low variance.

$$\hat{f}_{avg}(x) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}^b(x) \tag{5.64}$$

On the other hand, B different datasets can be produced by sampling repeatedly from a single dataset D. Training done on the b-th bootstrapped dataset gives an output of $\hat{f}^{*b}$ and the average of them produces an output of low variance.

$$\hat{f}_{bag} = \frac{1}{B}\sum_{b=1}^{B} \hat{f}^{*b}(x) \tag{5.65}$$

In case of regression trees, B regression trees are trained with B bootstrapped dataset and their average result is the prediction.

Random Forest employs a special trick to decorrelate the B regression trees. At each split, m out of p inputs are allowed to be passed where roughly $m \approx \sqrt{p}$. When the dataset has a strong predictor along with some other modereate ones, the first split will always be done based on the strong predictor. This will cause the trees corresponding to the bootstrapped datasets to be more or less similar. Whe m inputs are randomly chosen and passed to to the split, $\frac{p-m}{p}$ parts of the of input at a split will not consider the strong predictor. This will prevent correlation between the trees for bootstrapped data. Finally, the outputs obtained from the trees are averaged. This case is bagging if $p = m$.

5.11.1 Application

In the current case, because of the presence of only one predictor, the random forest used is just a bagged decision tree. According to table 5.3, its performance is similar to the k-D trees.

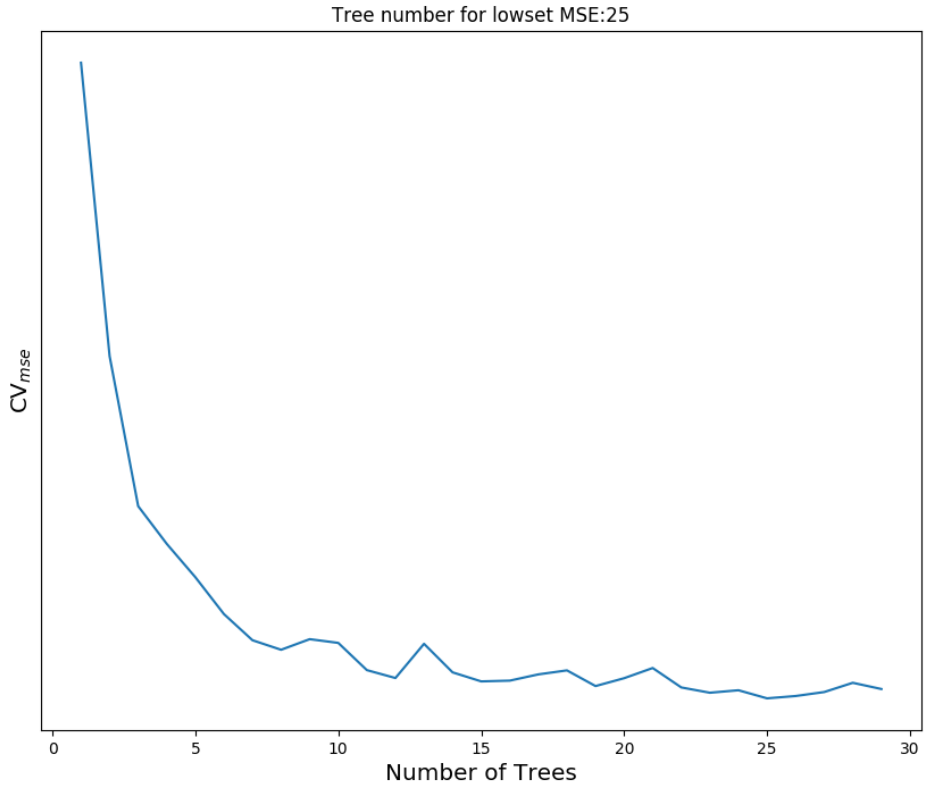


Figure 5.21: Determining number of trees for random forest using cross validation

Model	Mean Square Error	Mean Average Error	R^2
K-nn with K-D Tress	0.000520	0.002600	0.993471
Random Forest	0.000372	0.002677	0.995325
Linear Regression	0.002559	0.041080	0.967874
Ridge Regression	0.002559	0.041080	0.967874

Table 5.1: Metrics on validation set for different models

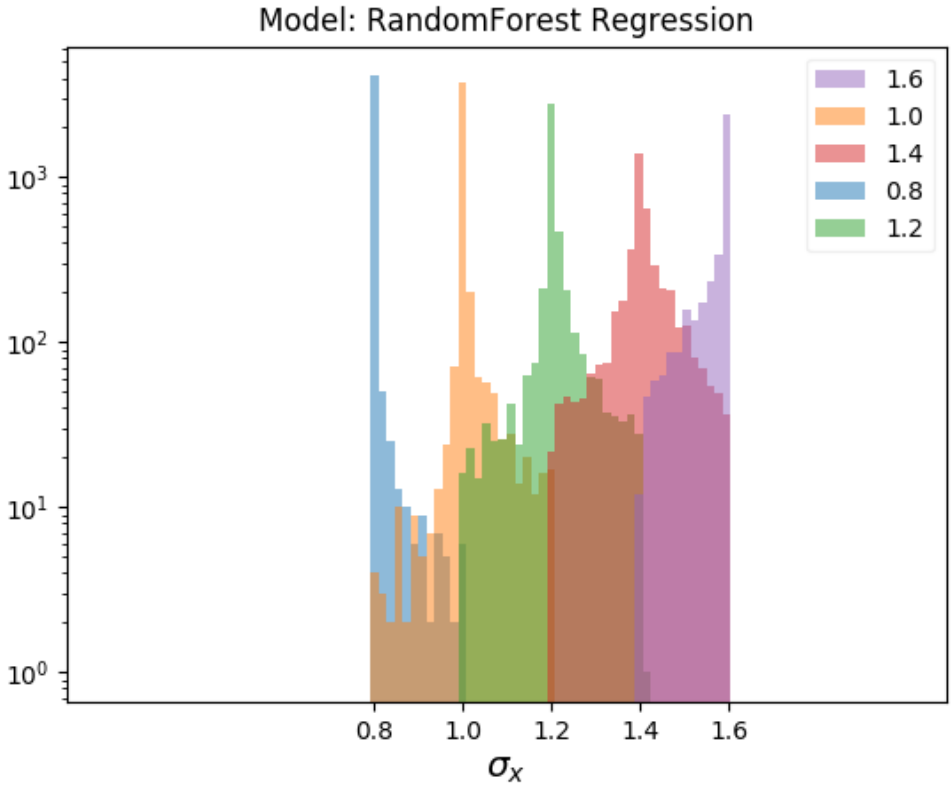


Figure 5.22: Prediction of random forest on validation set

5.12 Neural Network

In case of linear regression (equation 5.25), the learning is done only on parameter $\boldsymbol{w}$. But instead of using the input $\boldsymbol{x}$ directly, a learnable version of the function of $\boldsymbol{x}$ can be used as learning parameters as well as $\boldsymbol{w}$. In a general case of linear models like that of linear regression, the output can be defined as the following:

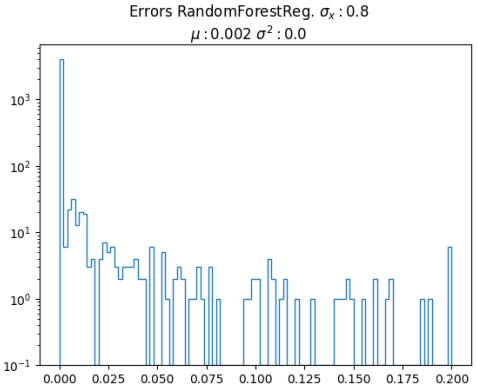
$$y(\boldsymbol{x}, \boldsymbol{w}) = f\left(\sum_{j=1}^{M} w_j \phi_j(\boldsymbol{x})\right) \tag{5.66}$$

where f is a non-linear activation function in case of classification or an identity function in case of regression. Neural networks exploit the ability of the function ϕ_j to learn along with the coefficients $\{w_j\}$. In other words, neural networks is a series of functional transformations.

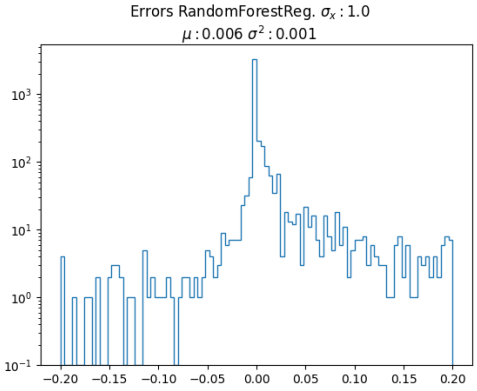
Neural network contains one or multiple layers, with each layer containing one or multiple nodes. Each node corresponds to a certain ϕ_j, that is input into the next layer. The first layer having M nodes, is M linear combinations of the input variables $x_1, \ldots, x_D$

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \tag{5.67}$$

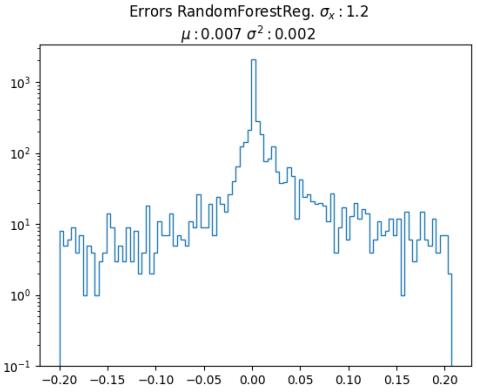
where $j = 1, \ldots, m$ The superscript (1) denotes the first layer, $w_{ji}^{(1)}$ the denotes



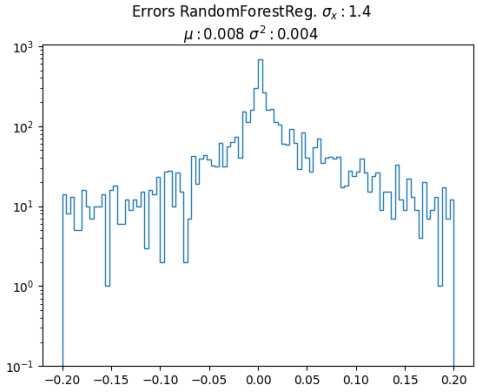
(a) Errors for σ_x factor 0.8



(b) Errors for σ_x factor 1.0



(c) Errors for σ_x factor 1.2



(d) Errors for σ_x factor 1.4

Figure 5.23: Errors for predictions for the factors of σ_x (Random Forest)

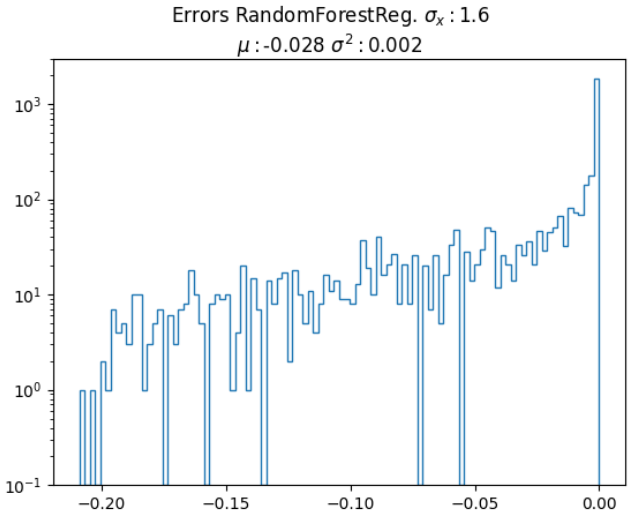


Figure 5.24: Errors for predictions for σ_x factor 1.6 (Random Forest)

Layer	Number of nodes
Input Layer	1
Hidden Layer 1	1024
Hidden Layer 2	512
Hidden Layer 3	256
Hidden Layer 4	128
Hidden Layer 5	18
Output Layer	1

Table 5.2: Neural Network Architecture used for the current study

weights and $w_{j0}^{(1)}$ denotes the bias. A non-linearity known as activation function is applied to this output,

$$z_j = h(a_j) \tag{5.68}$$

The quantities with activation function applied, are inside the hidden layers. These are the layers between the inputs and outputs. In the current study, the function h has been chosen to be Rectified Linear Unit (RELU). RELU is defined as,

$$\sigma(z) = max(0, z) \tag{5.69}$$

The outputs from the first layer (equation 5.68) are then linearly combined to give the output activations.

$$a_k = \sum_{j=1}^{M} w_{ji}^{(1)} z_i + w_{j0}^{(1)} \tag{5.70}$$

where $k = 1, \ldots, K$ is the total number of outputs.

Input layer output (from equation 5.67) and hidden layer output (from equation 5.70) can be combined to give the result at the output layer. For regression, which has been used in the current study, the function $\sigma(z)$ corresponding to the output layer is identity function.

$$y_k(\boldsymbol{x}, \boldsymbol{w}) = \sigma \left(\sum_{j=1}^{M} w_{kj}^{(2)} h \left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \right) + w_{k0}^{(2)} \right) \tag{5.71}$$

The biases in the equation 5.67, can be absorbed into the weights by taking an input $x_0 = 1$ so that,

$$a_j = \sum_{i=0}^{D} w_{ji}^{(1)} x_i \tag{5.72}$$

Similarly, from equation 5.71, we get,

$$y_k(\boldsymbol{x}, \boldsymbol{w}) = \sigma \left(\sum_{j=0}^{M} w_{kj}^{(2)} h \left(\sum_{i=0}^{D} w_{ji}^{(1)} x_i \right) \right) \tag{5.73}$$

The above method of calculating the output is called forward propagation. Figure 5.25, shows the network corresponding to the equation 5.73.

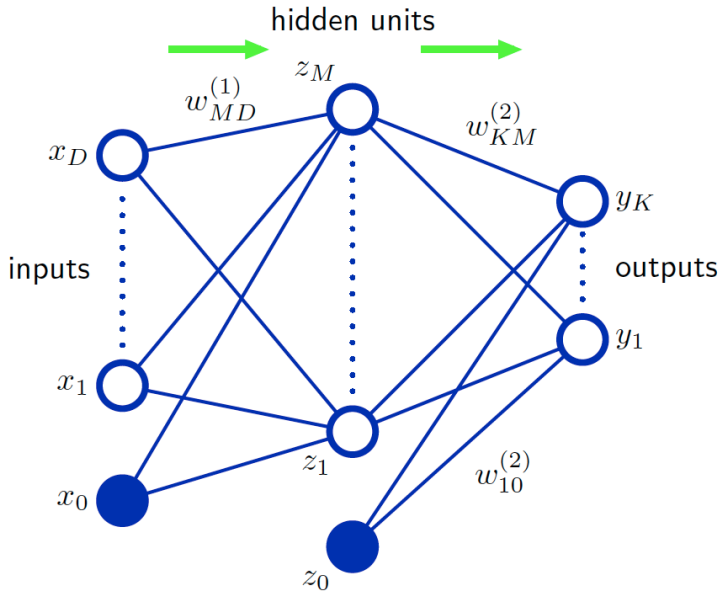


Figure 5.25: Neural Network [33]

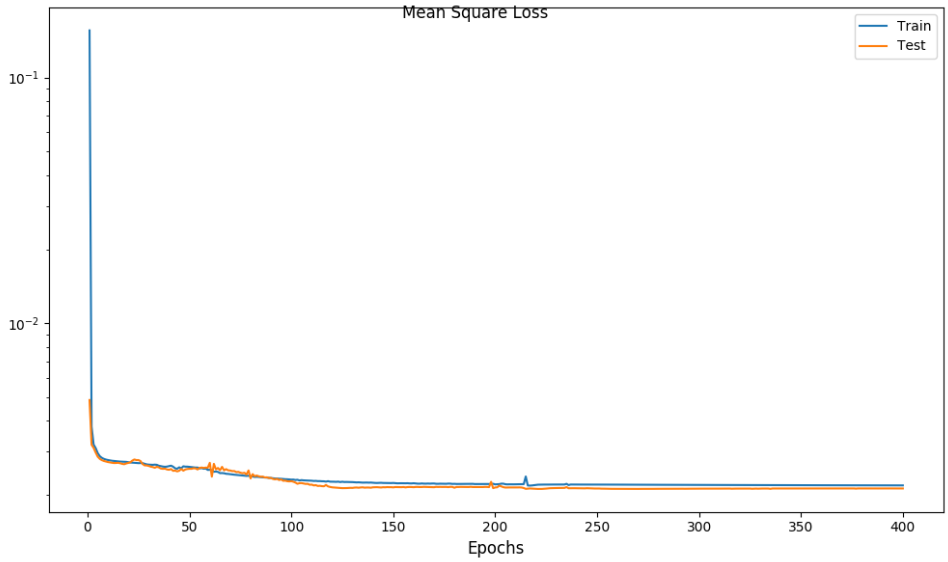


Figure 5.26: Training and validation loss

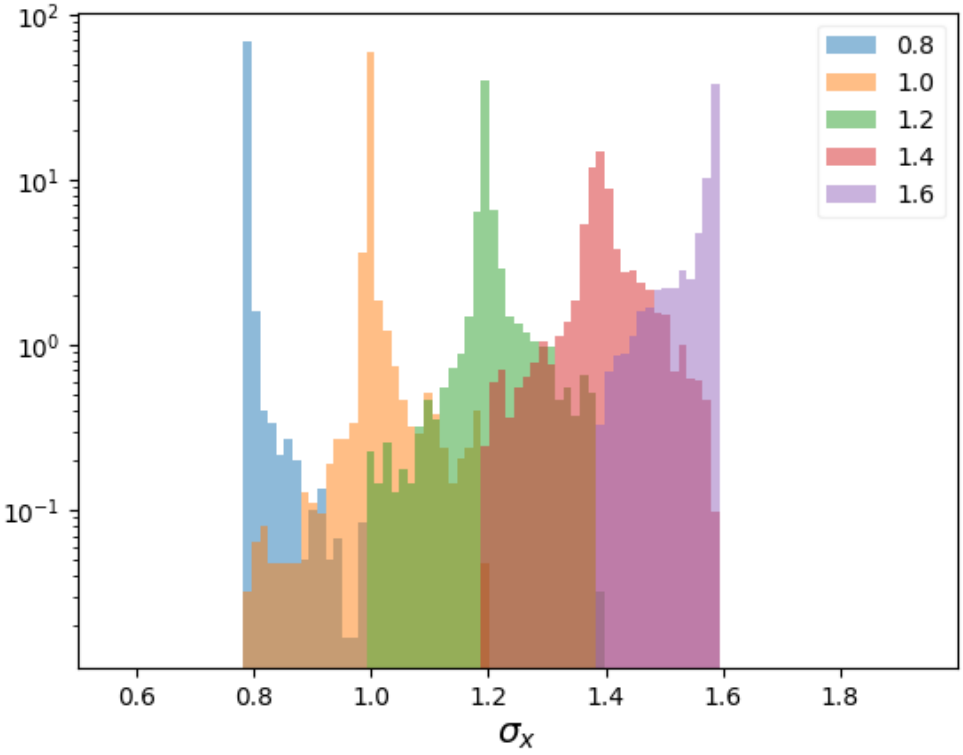


Figure 5.27: Neural Network output on validation data

The output obtained in this way is evaluated by means of a cost function, $E(\boldsymbol{w})$ which in the current study is the mean square error, which is defined as the following:

$$E(\boldsymbol{w}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i(\boldsymbol{w}))^2 \tag{5.74}$$

for an input $(\boldsymbol{x}_i, y_i)$, where $\boldsymbol{x}_i \in \mathbb{R}^{d+1}$ and prediction $\hat{y}_i(\boldsymbol{w})$

Like the other machine learning algorithms mentioned in the earlier sections, the error function (equation 5.74) can also be minimized by means of gradient descent (GD) described in section 5.5. But the presence of multiple layers causes the error function to be a composite function of all the weight parameters used in the earlier layers. Because of this brute force calculation of GD is not feasible in this case. Instead a special algorithm known as *backpropagation* is used.

Change of the cost function with respect to the weighted input in a layer, i.e. the error $\Delta_j^{(l)}$ corresponding to the j-th neuron in the l-th layer can be defined as,

$$\Delta_j^{(l)} = \frac{\partial E}{\partial z_j^{(l)}} = \frac{\partial E}{\partial a_j^{(l)}}\frac{\partial \sigma(z_j^{(l)})}{\partial z_j^{(l)}} \tag{5.75}$$

where $a_j^{(l)} = \sigma(z_j^{(l)})$ i.e a non-linearity (in this case RELU), applied to the output $z_j^{(l)}$.

Since error in layer l is propagated from the subsequent layer $l+1$, chain rule of differentiation can be used to write,

$$\Delta_j^{(l)} = \frac{\partial E}{\partial z_j^{(l)}} = \sum_k \frac{\partial E}{\partial z_k^{(l+1)}} \frac{\partial z_k^{(l+1)}}{\partial z_j^{(l)}} \tag{5.76}$$

$$= \sum_k \Delta_k^{(l+1)} \frac{\partial z_k^{(l+1)}}{\partial z_j^{(l)}} \tag{5.77}$$

$$= \left(\sum_k \Delta_k^{(l+1)} w_{kj}^{(l+1)} \right) \frac{\partial \sigma(z_j^{(l)})}{\partial z_j^{(l)}} \tag{5.78}$$

When final error is differentiated by the weight of the k-th neuron of a specific layer l, the result can be expressed by means of the product of $\Delta_j^{(l)}$ from above and the output of the k-th from the previous layer, $l-1$. In such way the derivative of E for weights at all layers and neurons can be computed.

$$\frac{\partial E}{\partial w_{jk}^{(l)}} = \frac{\partial E}{\partial z_j^{(l)}} \frac{\partial z_j^{(l)}}{\partial w_{jk}^{(l)}} = \Delta_j^{(l)} a_k^{(l-1)} \tag{5.79}$$

This approach makes it possible to update the weights:

$$\boldsymbol{w}^{(l+1)} := \boldsymbol{w}^{(l)} - \eta \frac{\partial E}{\partial \boldsymbol{w}^{(l)}} \tag{5.80}$$

where the elements of the vector $\boldsymbol{w}^{(l)}$ correspond to all the weights in the neurons of a particular layer l and η denotes the learning parameter.

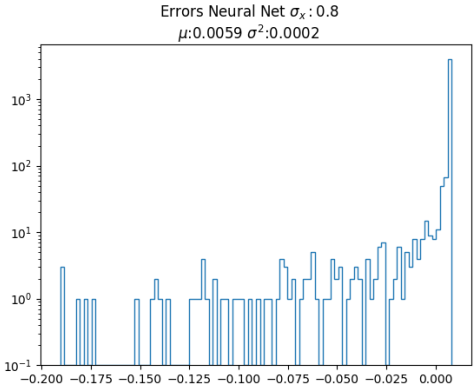
Implementing the above calculation from scratch is a lot of work. In the current study, the advantage of *Autograd*, which automatically computes the backpropagation steps shown above, of *PyTorch* has been taken.

To sum up, a neural network consists of the following steps:

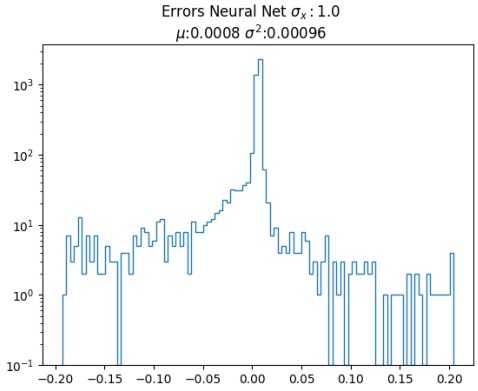
- Activation at the input layer: Calculate $a_j^{(l)}$ for all the neurons at a layer.
- Feedforward: Calculate the final output like that of equation 5.71.
- Calculate error using the error function (mean square error for regression or categorical cross entropy function for classification)
- Backpropagate the error using equation 5.78
- Update the parameters using equation 5.80

5.12.1 Application

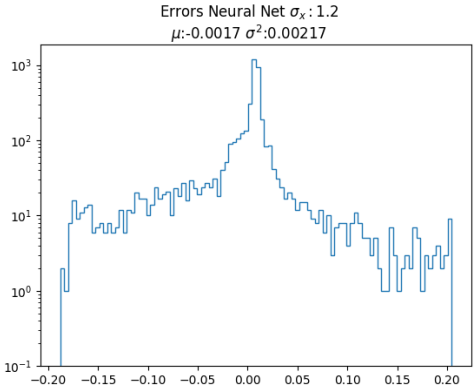
In the current dataset, in order to make the neural network work effectively, *MinMaxScaler*[36] has been used. From the figure 5.26, it can be seen that both the training and validation error are almost equal for most of the epochs. The trained model corresponding to the epoch that gives the lowest validation error, has been used. From table 5.3, neural network has performance similar to both k-D trees and random forest except for the beam size factor 1.6.



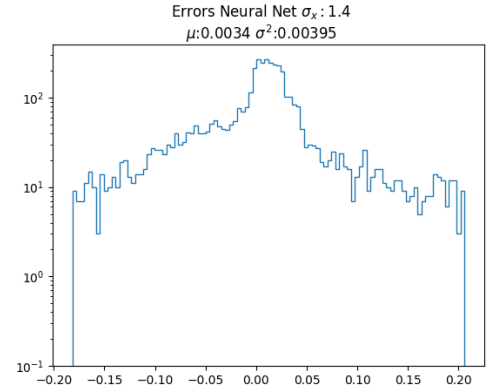
(a) Errors for σ_x factor 0.8



(b) Errors for σ_x factor 1.0



(c) Errors for σ_x factor 1.2



(d) Errors for σ_x factor 1.4

Figure 5.28: Errors for predictions for the factors of σ_x (neural network)

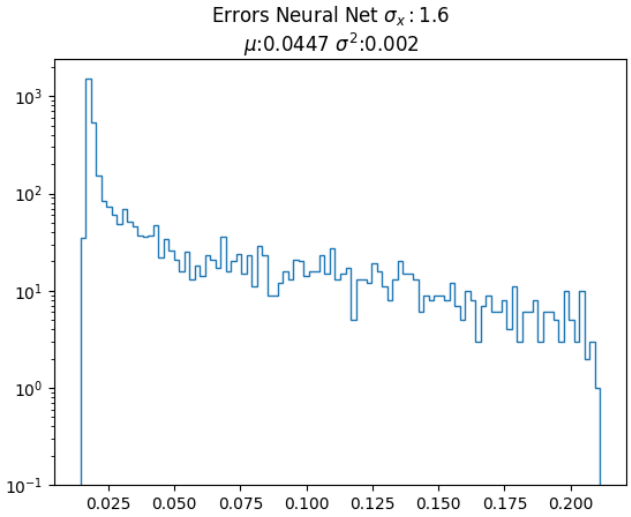


Figure 5.29: Errors for predictions for σ_x factor 1.6 (neural network)

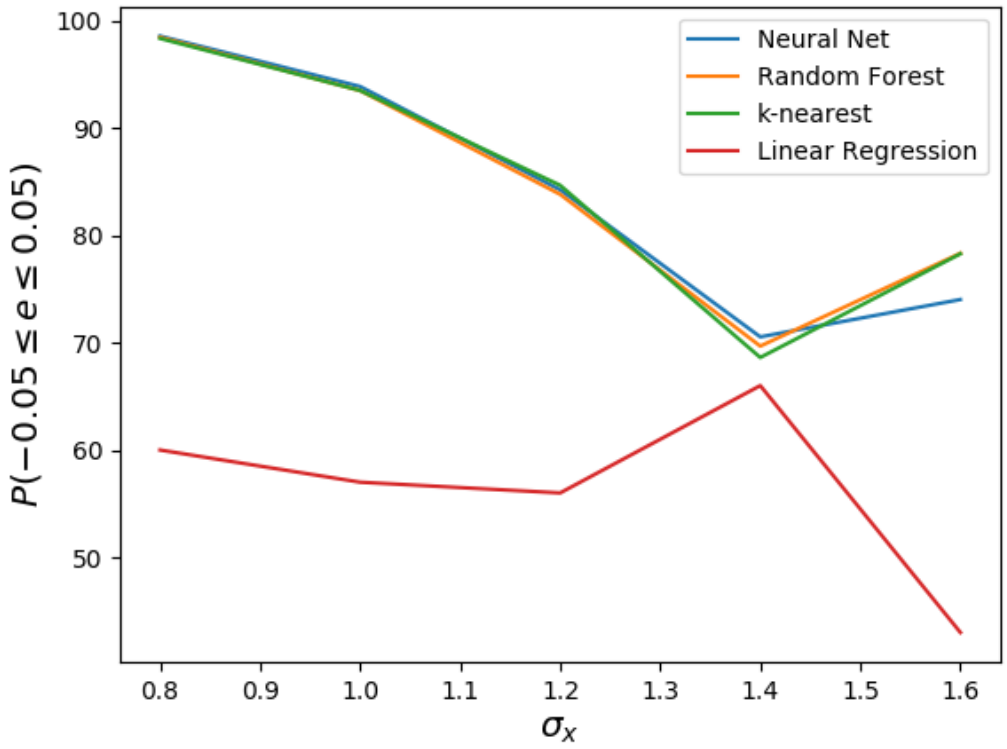


Figure 5.30: Probability of the factor errors to be within -0.05 to 0.05

Beam Size Factor	Neural Net(%)	Random Forest(%)	k-D Tree(%)	Linear Reg.(%)
0.8	99	98	99	60
1.0	94	93	94	57
1.2	84	84	85	56
1.4	71	70	69	66
1.6	74	78	78	43

Table 5.3: Table of $P(-0.05 \leq e \leq 0.05)$ for different algorithms

5.13 Convolutional Neural Network

Convolutional Neural network, or ConvNet architectures assumes that the inputs are images, and exploits the locality and translational invariance of an image. ConvNets have been historically developed based on the principle of edge detection. A very fundamental edge detection method is to detect vertical or horizontal edges in an image. ConvNets in their initial layers learn vertical or horizontal edges and learns more complicated features such as round shape etc. in the later layers.

Instead of using the one dimensional layered structures of neural nets, ConvNets use volumes of neurons having length, height and width. In case of an RGB image, the length and heights represent the image dimension and the width represents the dimension corresponding to the color channels (red, green and blue). In other words, an RGB image is represented as a three dimensional matrix. The matrix is then *convolved* with a filter of a specific shape. Filter is also a volume where the width of

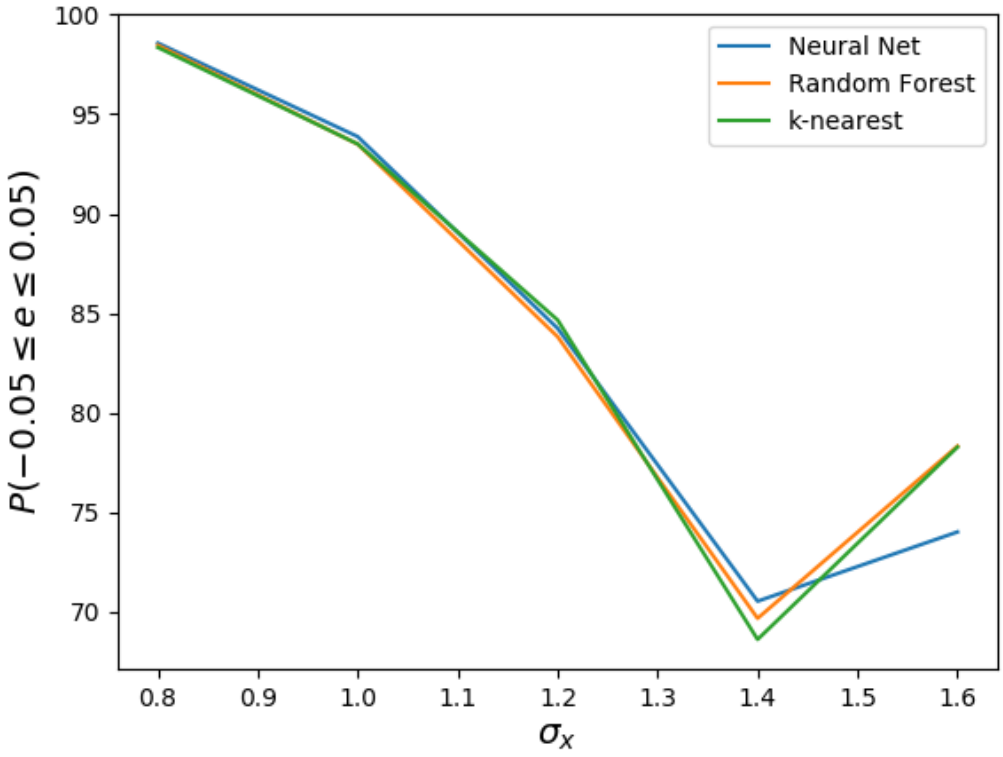


Figure 5.31: Probability of the factor errors to be within -0.05 to 0.05 (leaving linear regression)

a filter corresponds to its channel numbers. The channel number of the filter must equal the channel number of the input (in this case, the RGB image). The elements of the filter matrix are learnable parameters or the *weights*. In other words, each horizontal slice of a filter volume, can be considered as a neural network consisting of multiple layers.

Filters can be more than one in number, in which case the output volume after convolving with filters, contains a channel number equal to the number of filters convolved. Convolution is carried out by element-wise multiplication of filter elements with the image elements. This element-wise multiplication is done independently for each channel. The output corresponding to this operation is the sum of all values, obtained after element-wise multiplication, across the channel.

When convolutional layer of the form, $g = C_\Gamma(\boldsymbol{f})$, is applied on a p-dimensional input, $\boldsymbol{f} = (f_1(x), \ldots, f_p(x))$ by means of multiple filters $\Gamma = (\gamma_{l,l'})$ for $l = 1, \ldots, q$, $l' = 1, \ldots, p$ and an activation function ξ, the output can be represented as follows:

$$g_l(x) = \xi\left(\sum_{l'=1}^{p}(f_{l'} * \gamma_{l,l'})(x)\right) \tag{5.81}$$

and the convlution operation is defined as follows:

$$(f_{l'} * \gamma_{l,l'})(x) = \int_\Omega f(x - x')\gamma(x')dx' \tag{5.82}$$

where $\Omega = [0, 1]^d$; d stands for dimension of the space where the operation is being carried out. In case of image $d = 2$. When applied to an image the $\boldsymbol{f}$ corresponds to an RGB image having 3 channels i.e $\boldsymbol{f} = (f_1(x), f_2(x), f_3(x))$. Equation 5.81 describes the aforementioned element wise multiplication of image having p channels, with q filters and a subsequent addition over the p input channels.

Equation 5.82, describes the element wise multiplication along with a translation. This translation is called *stride*. Convolution along with each stride of the filter over the image produces one element of the output volume. Dimension of one plane of the resulting volume is therefore $[\frac{W-F}{S} + 1, \frac{W-F}{S} + 1]$, where W, F, S are one side of the square image, one side of the filter plane and stride value respectively. In order to avoid the shrinking of image dimension over this operation, the image matrix can be padded with 0 values along its two dimensions. The resulting dimension after P zero-paddings is $[\frac{W-F+2P}{S} + 1, \frac{W-F+2P}{S} + 1]$. Therefore in order to keep the image shape unchanged over this operation, the padding should be chosen such that the output size remains as W.

The next step of ConvNet is called pooling, which downsizes the image by extracting information from certain part of the image. In the currrent study, *MaxPool* layer has been applied at the end of convolution layers. This method replaces a small region (in the current study, 2×2 neurons) from a plane of the volume, by a single neuron, which produces the maximum output. For $l = 1, \ldots, q$ it is defined as,

$$g_l(x) = P(\{f_l(x') : x' \in \mathbb{N}(x)\}) \tag{5.83}$$

where $\mathbb{N} \subset \Omega$ is the neighbouring region about x and P is a permutation invariant function such as max.

Combining all of the above, a ConvNet can be described as the following,

Layer	In Channel	Out Channel	Kernel Size	Stride	Padding
1	1	64	(3,3)	(1,1)	1
RELU					
2	64	64	(3,3)	(1,1)	1
RELU					
MaxPool2D			(2,2)	(2,2)	

Table 5.4: ConvNet Layers

Layer	Number of nodes
Hidden Layer 1	1024
Hidden Layer 2	512
Output Layer	25

Table 5.5: Hidden layer of neural networks after convolution layers

$$U_\Theta(f) = (C_{\Gamma^{(K)}} \cdots P \cdots C_{\Gamma^{(2)}} \cdots C_{\Gamma^{(1)}})(f) \tag{5.84}$$

where $\Theta = \{\Gamma^{(1)}, \ldots, \Gamma^{(K)}\}$, which are the vectors containing the parameters of the filters.

Finally, the neurons of the last conv-layer are flattened and connected to multiple layers of neural networks, known as Fully-Connected Layers (FC layers). The number of neurons at the last layer of this network equal the number of classes that have to be predicted.

Because it is a classification task the cost function used is categorical cross entropy described below:

$$E(\boldsymbol{w}) = -\sum_{i=1}^{n}\sum_{m=0}^{M-1} (y_{im}\ log[\hat{y}_{im}](\boldsymbol{w}) + (1 - y_{im})\ log[1 - \hat{y}_{im}(\boldsymbol{w})]) \tag{5.85}$$

where y is the number corresponding to the M classes i.e. $y \in \{0, 1, \ldots, M-1\}$ and the probability of predicting a class, given input, or $p(y_i = m|\boldsymbol{x}; \boldsymbol{w})$ is denoted by $\hat{y}_{im}(\boldsymbol{w})$. In the case of more than one classes, *one vs. all* approach is adopted where the m-th class of i-th sample under consideration is assigned $y_{im} = 1$ and all others are assigned 0. Frameworks like *PyTorch* includes this feature as a default and it does not need *one-hot* encoding.

5.13.1 Application

Matrix corresponding to the histograms of horizontal and vertical positions of the particle hits were created and then trained by ConvNet layers described in table 5.4. Generally speaking, these matrix contain more information about particle hits than just number of entries or ρ_{mean} or ρ_{mode}. That is why, this method can be the natural way to determine both horizontal and vertical beam sizes simultaneously.

In this study, 25 different classes corresponding to 5 σ_x values and 5 σ_x values for each σ_x, were created and their matrix were obtained for about 14000 events.

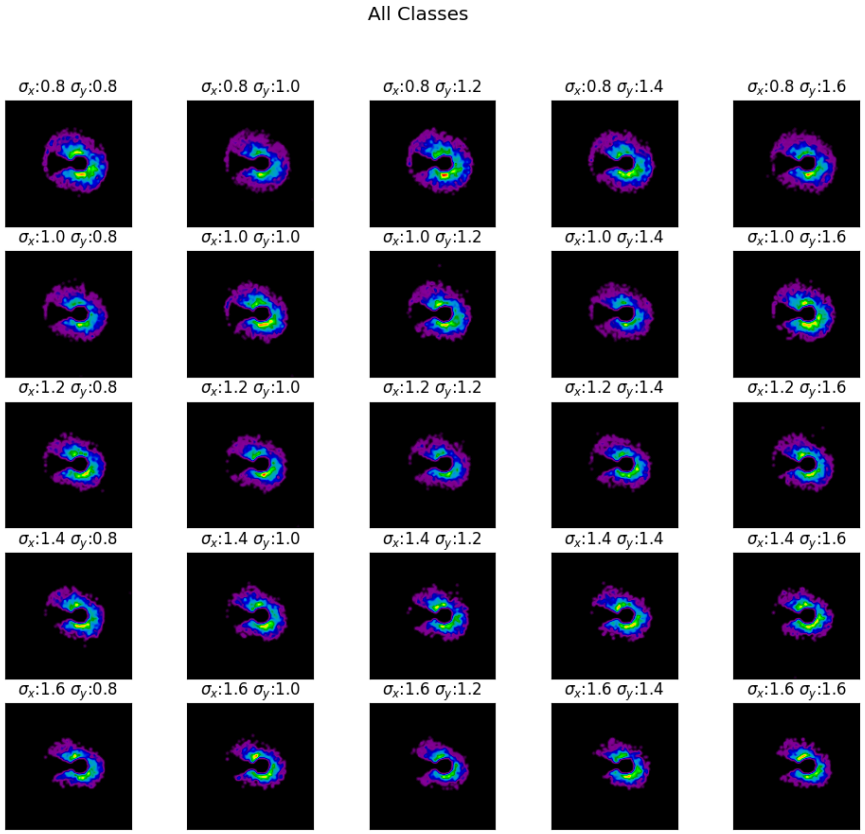


Figure 5.32: Images of all 25 classes (after removing one entry bins)

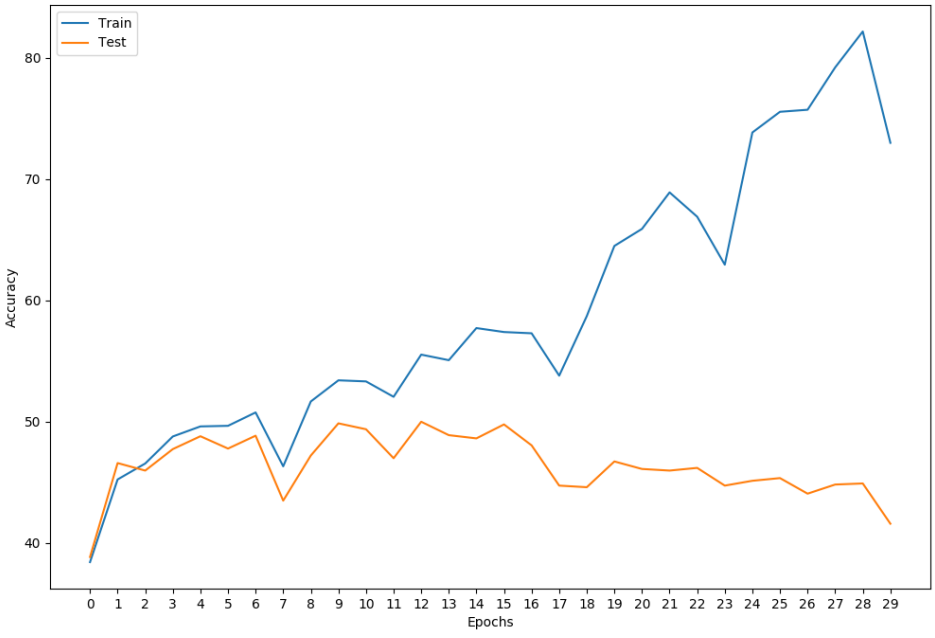


Figure 5.33: Accuracy: Train vs. Test

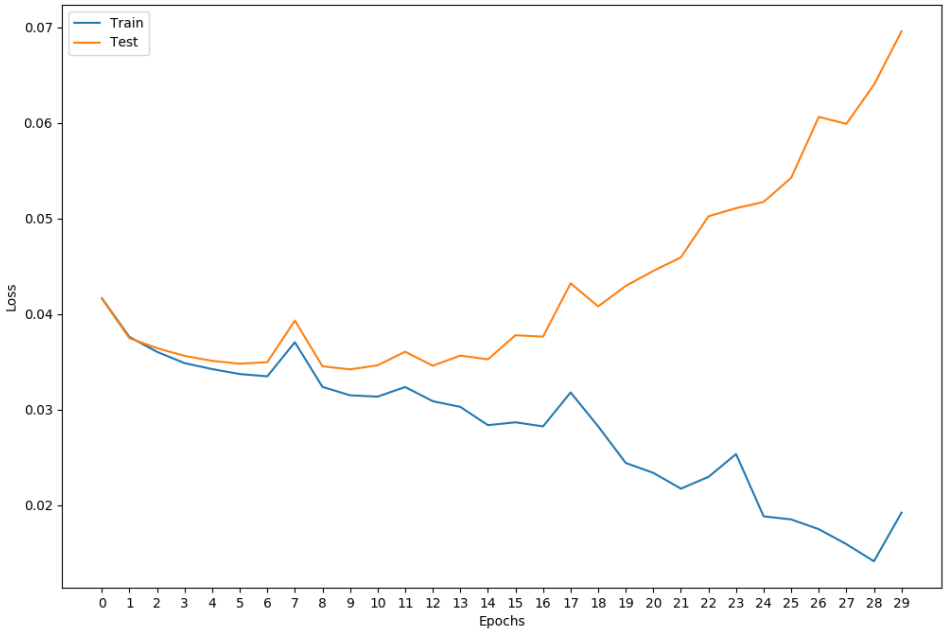


Figure 5.34: Losses: Train vs. Test

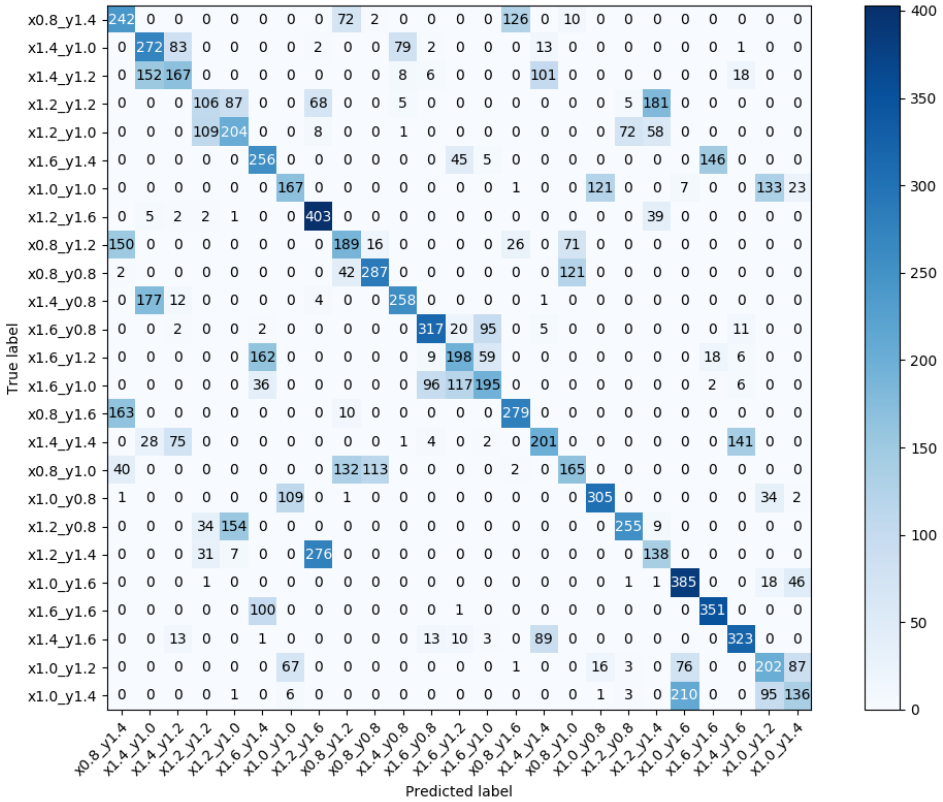


Figure 5.35: Confusion matrix: Train

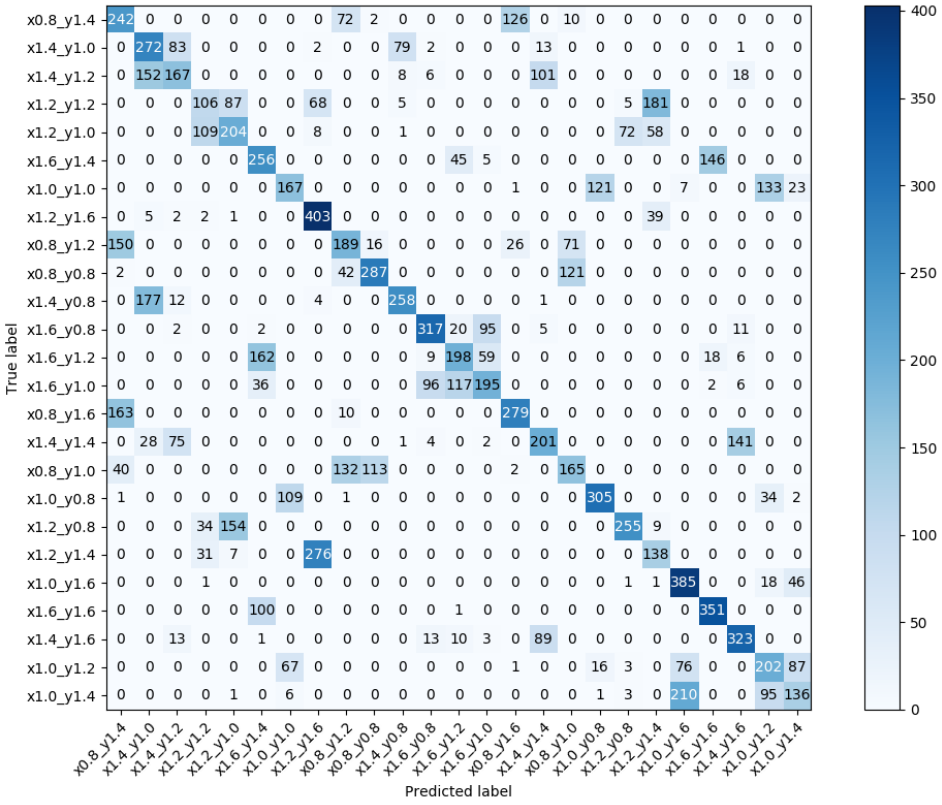


Figure 5.36: Confusion matrix: Test

In order to reduce noise corresponding to the events, the histogram bins with one entry were removed. In absence of this preprocessing step, the training accuracy was very low.

From figure 5.33, it can be seen that over 30 epochs even though the training accuracy was very high, the validation accuracy was low. This means the model is over training. This can be reduced by an increase of model complexity or in this context, an increase of ConvNet layers.

A drawback of this method is that, the amount of dataset required increase by n^2 for each of the n values of σ_x or σ_y. Simulation of very high number of events also require a lot of time. Additionally, the matrix generated is a noisy dataset, because of statistical fluctuations of the ρ or ϕ distributions due to the change of σ_y across each bunch (event). The fluctuation can be reduced by creating each *beam image* from at least 50 bunches. Besides, clever usage of image preprocessing can reduce these noises and thereby increase the training accuracy.

Chapter 6

Conclusion

The parameters used to predict the values of σ_x in this research are based on the approximate equation shown at equation 3.23. In other words, if any of these parameters are taken as a function $f(x, y)$, where x and y correspond to horizontal and vertical beam sizes in this context,

$$f(x,y) \approx f(a,b) + \frac{\partial f}{\partial x}(a,b)(x-a) + \frac{\partial f}{\partial y}(a,b)(y-b) \tag{6.1}$$

where $f(x, y)$ is evaluated about a certain point (a, b), which in this context is (σ_x, σ_y). The violin plots convinces us that the slope due a change of σ_y is much lower than that for the case of σ_x when $f(x, y)$ (Number of entries, ρ_{Mean} etc.), when evaluated at a point with coordinate values equal to the current values if σ_x and σ_y i.e. 729 nm and 7.7 nm respectively. Therefore, the approach of predicting σ_x from *number of entries* only, is justifiable.

Because of this low dependency of ϕ, ρ, number of entries etc. on σ_y values, the statistical fluctuation of these observables when σ_y is changed, is very high for 1 bunch (e.g. figure 4.10). That is why, future analysis to determine σ_y, must be carried out with multiple bunches (at least above 10 bunches per one combination of horizontal and vertical beam).

Besides, it is also seen that regression is a more feasible approach as compared to ConvNets which was used in [2], when it comes to predicting the beam sizes. The fact that, accurate prediction of σ_x has been possible using the observable, number of entries, it can be said that future beam parameters such as transverse beam offset can also be determined accurately using this machine learning approach.

Bibliography

[1] Y. Sato. *Research and development of an interaction-region beam profile monitor for the international linear collider*. 2009. URL: http://epx.phys.tohoku.ac.jp/eeweb/paper/2010_Mthesis_sato.pdf.

[2] Y. Kobayashi. *The application of machine learning to an interaction-point beam profle monitor for the ILC*. 2019. URL: http://epx.phys.tohoku.ac.jp/eeweb/paper/2019_Mthesis_ykoba.pdf.

[3] T. Behnke et al. "The International Linear Collider technical design report - volume 1: Executive Summary". In: *arXiv* (2013), p. 9.

[4] T. Behnke et al. "The International Linear Collider technical design report - volume 1: Executive Summary". In: *arXiv* (2013), p. 12.

[5] T. Behnke et al. "The International Linear Collider technical design report - volume 1: Executive Summary". In: *arXiv* (2013), p. 14.

[6] T. Behnke et al. "The International Linear Collider technical design report - volume 4: Detectors". In: *arXiv* (2013), pp. 181–314. DOI: arXiv:1306.6329.

[7] J. S. Marshall et al. "The Pandora Particle Flow Algorithm". In: *arXiv* (2013). DOI: arXiv:1308.4537.

[8] Daniel Jeans. *Particle Flow Algorithms*. 2018. URL: http://ias.ust.hk/program/shared_doc/2018/201801hep/program/exp/HEP_20180118_0950_Daniel_Jeans.pdf.

[9] T. Behnke et al. "The International Linear Collider technical design report - volume 4: Detectors". In: *arXiv* (2013), p. 186. DOI: arXiv:1306.6329.

[10] T. Behnke et al. "The International Linear Collider technical design report - volume 4: Detectors". In: *arXiv* (2013), p. 200. DOI: arXiv:1306.6329.

[11] T. Behnke et al. "The International Linear Collider technical design report - volume 4: Detectors". In: *arXiv* (2013), p. 241. DOI: arXiv:1306.6329.

[12] Adrian Vogel. *The Coordinate System for LDC Detector Studies*. 2005. URL: https://www.lctpc.org/e11/e72/.

[13] Andrei Seryi et al. "IR Optimization, DID and anti-DID". In: (Feb. 2006). URL: http://inspirehep.net/record/709954/files/slac-pub-11662.pdf.

[14] Akiya Miyamoto Daniel Jeans. *Machine-related backgrounds in ILD*. URL: https://confluence.desy.de/display/ILD/ILD+notes.

[15] T. Tauchi and K. Yokoya. "Nanometer-beam-size measurement during collisions at linear colliders". In: (1995). DOI: https://doi.org/10.1103/PhysRevE.51.6119. URL: https://journals.aps.org/pre/pdf/10.1103/PhysRevE.51.6119.

[16] Werner Herr and B Muratori. "Concept of luminosity". In: (2006). DOI: 10.5170/CERN-2006-002.361. URL: https://cds.cern.ch/record/941318.

[17] Daniel Schulte. "Beam-beam Effects in Linear Colliders". In: *CERN Yellow Reports: School Proceedings* 3.0 (2017), p. 431. ISSN: 2519-805X. URL: https://e-publishing.cern.ch/index.php/CYRSP/article/view/267.

[18] Mauro Pivi. *Beam-Beam Effects in Particle Colliders*. 2011. URL: https://uspas.fnal.gov/materials/11ODU/Beam-Beam.pdf.

[19] Y. Sato et al. "SOI readout ASIC of pair monitor for International Linear Collider". In: (2011). DOI: https://doi.org/10.1016/j.nima.2011.02.063. URL: https://www.sciencedirect.com/science/article/pii/S0168900211004190.

[20] K. Yokoya. *User's Manual of CAIN*. 2003. URL: https://ilc.kek.jp/~yokoya/CAIN/cain235/CainMan235.pdf.

[21] M Petrič et al. "Detector Simulations with DD4hep". In: *Journal of Physics: Conference Series* 898 (Oct. 2017), p. 042015. DOI: 10.1088/1742-6596/898/4/042015. URL: https://doi.org/10.1088%2F1742-6596%2F898%2F4%2F042015.

[22] *MARLIN - Modular Analysis and Reconstruction for the LINear collider*. URL: http://ilcsoft.desy.de/portal/software_packages/marlin/index_%20eng.html.

[23] Frank Gaede et al. "LCIO - A persistency framework for linear collider simulation studies". In: *arXiv.org* (2003). DOI: arXiv:physics/0306114v1.

[24] T. Behnke et al. "The International Linear Collider technical design report - volume 1: Executive Summary". In: *arXiv* (2013), p. 11.

[25] S. Agostinelli et al. "Geant4—a simulation toolkit". In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* (2003). DOI: 10.1016/S0168-9002(03)01368-8.

[26] Tim Bray et al. *Extensible Markup Language*. URL: https://www.w3.org/TR/1998/REC-xml-19980210.html.

[27] F. Gaede et al. *ILD detector models - Overview*. URL: https://github.com/iLCSoft/lcgeo/tree/master/ILD/compact.

[28] Python Software Foundation. *Python Language Reference, version 3.5*. URL: http://www.python.org.

[29] Daniel Jeans. *Scans of the simulated ILD models*. URL: http://research.kek.jp/people/jeans/ctpics/.

[30] *ILCSoft web page*. URL: http://ilcsoft.desy.de/portal.

[31] Pankaj Mehta et al. "A high-bias, low-variance introduction to Machine Learning for physicists". In: *Physics Reports* (2019), pp. 11–12. DOI: 10.1016/j.physrep.2019.03.001.

[32] *Machine Learning for Intelligent Systems*. URL: https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote12.html.

[33] Christopher Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[34] Michael M. Bronstein et al. "Geometric Deep Learning: Going beyond Euclidean data". In: *IEEE Signal Processing Magazine* 34 (4 2017). DOI: 10.1109/MSP.2017.2693418.

[35] *Cross-validation: evaluating estimator performance*. URL: https://scikit-learn.org/stable/modules/cross_validation.html.

[36] *Preprocessing data*. URL: https://scikit-learn.org/stable/modules/preprocessing.html.