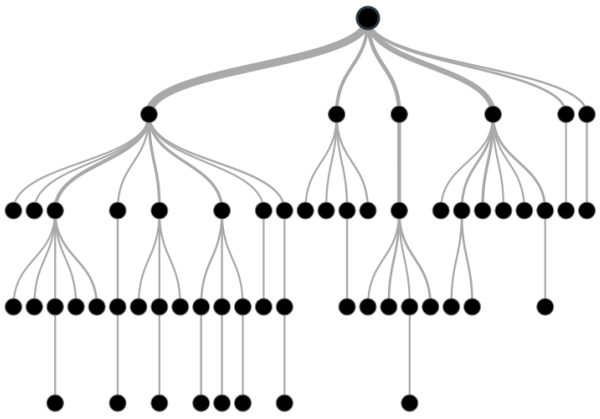
ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
Εθνικό και Καποδιστριακό
Πανεπιστήμιο Αθηνών

National Kapodistrian University of Athens
Department of Biology
MSc Bioinformatics and Computational Biology

MSc Thesis:

Machine Learning Techniques and Cell Irradiation

Dimitris Papakonstantinou



Advisory committee: **A.Georgakilas**, A.Chatziioannou, I.Trougakos

Athens, October 2019

Abstract

This is study on how Machine Learning (ML) techniques can be used to predict the outcome of ionizing radiation on cells. The problem consists of predicting the quantities of Relative Biological Effectiveness (RBE) and the α and β coefficients of the quadratic model. The quantities to be predicted are continuous, so in essence it is a multi-variable regression problem. The RBE quantity in our case, signifies, what effect the radiation has on cell survival. The α and β coefficients, represent the linear and quadratic contributions to cell death respectively. There are 3 different ML algorithms used, using 2 separate datasets. The algorithms used were, Gradient Boosting Decision Trees (GBDT), the Random Forest Regression (RF) and Support Vector Regression (SVR). Two implementations of the GDBT were used. As a further trial, a voting regression (VR) is implemented across all previous algorithms, that predicts the dependent variables based on a consensus method. The algorithms employed were of radically different design and approach. The aim was to combine different techniques and show that the combined model fairs better in predictive performance and generalization. The results show that the CatBoost implementation of the GBDT algorithm fairs quite better in generalizing and predicting. Our datasets consists of mainly HZE irradiation features, i.e they contain cell specific data like cell line, cell phase and tumorous state, as well as radiation features like LET, specific energy and ion species. The Datasets are quite small, compiled by different methods and generally cannot been seen as a black box. It is shown that the datasets are somewhat noisy and contain multi-collinearities The whole work is meant to be a showcase of the usefulness of ensemble and consensus techniques in predicting the aforementioned quantities. A bigger, more cohesive and consistent dataset is required, in order to predict, prior to a radiotherapy treatment, the outcome in a more robust and less arbitrary way. A more in-depth feature analysis is required, to assess which features are essential to the predictions. To make our estimators intended for a more general use in the scope of radiobiology, further integration with data from photon irradiation data are needed.

Acknowledgements

I would like to thank, Alexandros Georgakilas for his guidance through the whole project. Costas Siettos for his insightful recommendations that made this work more substantiated. Zacharenia Nikitaki for her help throughout the initial stages. Adriani Nikolakopoulou and Thodoris Papakonstantinou for their insights on the statistical and information theory argumentation. V.Kotsaris for his help in editing.

Contents

Chapter 1

Introduction

This thesis is a study on the prediction of the effect that radiation has on the survival of cells. We will employ various machine learning techniques to achieve this. This introduction is meant to give the context of the biological and physical details of the relation of ionizing radiation and cell survival, as well as the most common ways of modeling and quantifying the phenomenon, i.e. the Linear Quadratic model and the Relative Biological Effectiveness.

1.1 The Linear-Quadratic model

The LQ model, proposed by Douglas and Fowler in 1972 [1], is related to the characteristic shape of the log of percentage of cells that survive from a population in the y-axis in relation to the radiation dose in Gy in the x-axis (fig 1.1). Cell survival curves are plotted on a log-linear scale. With the increase of the dose, the curves bends over a region of several Gy. This region is often described to as the shoulder of the survival curve. At very high doses, the curve tends to straiten out again [2].

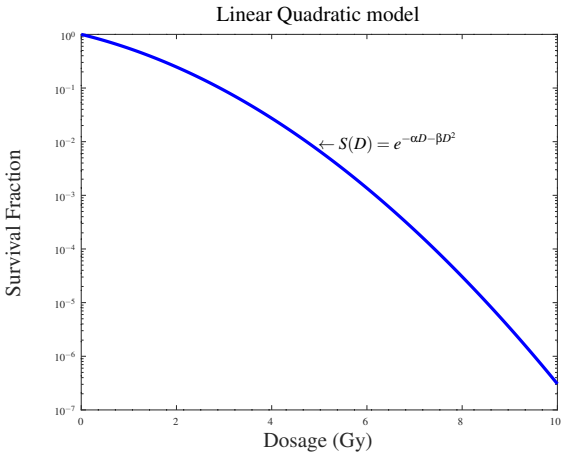


Figure 1.1: The LQ model

The LQ model is one among many that at various times have been proposed to represent the relation of cell survival and dose absorption. Hall and Giaccia et al [3] have made a thorough review of the various models. LQ model is among the most prevalent models in this field, and in essence, is seated on the assumption that there are two coefficients that contribute to cell killing: the first α has to do with dosage D and the second β with the square of the dosage D^2. This relation is expressed in the following exponential equation:

$$S(D) = e^{-(\alpha D + \beta D^2)}$$

where S is the fraction of cells that have survive dose D.The α and β parameters represent the linear and quadratic contributions to cell death respectively. LQ's convenience lies in the fact that it has only two parameters and it's not computationally demanding. There is also a biophysical importance and interpretation to these parameters. Single hits of radiation (one particle) may cause cell death by inducing breaks on two adjacent chromosomes (αD parameter). When two or more particles induce two chromosomal breaks, the injure becomes cumulative. The mechanism behind cell death due to irradiation is believed to be related to DNA double strand breaks [4]. The probability of this event, is proportional to the square of the dose, and this is what β parameter represents.

1.2 Relative Biological Effectiveness

In Radiobiology, RBE represents the ratio of the effects of two types of ionizing radiation on an organism, given the amount energy absorbed is the same. RBE is specific to the particles, the energies involved, and the biological effects that are being studied. The relative biological effectiveness for radiation of type R on a tissue is defined as the ratio

$$RBE = \frac{D_x}{D_l}$$

Where D_x is a reference dose of type X e.g. background radiation, and D_l is the absorbed dose that has the same biological effect, e.g. cell death. Different types of radiation will vary in their effectiveness.

Deriving an RBE value from the LQ model can be done with the assumption that, we define the biological effect to be the survival of a certain fraction of the population of cells. Commonly is used the quantity $RBE_{0.1}$. This basically denotes the ratio of the two types of radiations at a point where both the studied dose and the reference cause 90% of the cell dying. Given the LQ's model's equation, follows:

$$0.1 = e^{-(\alpha_x D_x + \beta_x D_x^2)}$$

$$0.1 = e^{-(\alpha_l D_l + \beta_l D_l^2)}$$

$$ln(0.1) = -\alpha_x D_x - \beta_x D_x^2$$

$$ln(0.1) = -\alpha_l D_l - \beta_l D_l^2$$

$$\beta_x D_x^2 + \alpha_x D_x + ln(0.1) = 0$$

$$\beta_l D_l^2 + \alpha_l D_l + ln(0.1) = 0$$

$$D_x = \frac{-\alpha_x + \sqrt{\alpha_x^2 - 4\beta_x ln(0.1)}}{2\beta_x}$$

$$D_l = \frac{-\alpha_l + \sqrt{\alpha_l^2 - 4\beta_l ln(0.1)}}{2\beta_l}$$

$$RBE_{0.1} = \frac{2\beta_l(-\alpha_x + \sqrt{\alpha_x^2 - 4\beta_x ln(0.1)})}{2\beta_x(-\alpha_l + \sqrt{\alpha_l^2 - 4\beta_l ln(0.1)})}$$

1.3 Linear Energy Transfer & ionizing radiation

Linear Transfer Energy (LET), is a quantity that signifies the amount of energy that ionizing radiation conveys to a material per unit distance. It describes the transfer of energy of radiation to matter. LET is dependent both from the nature of the matter irradiated, and the type of the radiation and it is strictly positive. High LET value, denotes quick attenuation of the radiation inside the material, greater protection and lower radiation penetration. Energy transfer causes damage to structure of the material, near the radiation track the ionizing particle takes in the material. In biological systems, such microscopic failures increase the chance of a large scale failure like DNA double strand breaks and cell apoptosis. LET can account for the fact that radiation damage is not proportional to the absorbed dose. The SI unit for LET is the newton, although it is typically measured in $KeV/\mu m$ or $MeV/\mu m$.

The impacting ionizing particles produce secondary electrons, if they themselves have high enough energy to cause ionization, are referred to as delta rays [5]. These secondary interactions happen outside the vicinity of the primary particle track, and therefore many studies exclude the consideration of delta rays with energies larger than a certain value Δ. This energy limit is meant to exclude secondary electrons that carry energy far from the track of the primary particle. It's an approximation that doesn't take into account the directional distribution of the secondary electrons energy transfer and their non-linear path, but it serves in simplifying analytic evaluation [6]. Thus the restricted linear energy transfer is defined:

$$L_\Delta = \frac{dE_\Delta}{dx}$$

where dE_Δ is the energy loss of the particle due to electronic collisions while traversing a distance dx, excluding all secondary electrons with energies larger than Δ. The higher the Δ value we set, the closest the LET and restricted LET become, because there are fewer electrons with energy higher than Δ.

LET is mostly suited for monoenergetic ions i.e. protons, alpha particles, and heavier nuclei called HZE ions present in cosmic rays or produced by accelerators. These particles frequently cause direct ionization within a narrow diameter diameter around a more or less straight track, we can therefore say that continuous deceleration occurs. During deceleration the particle cross section changes, and this in turn modifies their LET, which increases up to a Bragg Peak where there is a thermal equilibrium with the material. At equilibrium the incident particle either comes to rest or is absorbed, at which point LET is undefined.

Since LET is varying along the path length of the incident particle, the average value of LET is used to represent the spread. In dosimetry the common practice is to weight the average by the absorbed dose. The averages are not sufficiently separated for heavy particles with hight LET, in other types of radiation like beta particles the difference becomes important [6].

Another source of ionizing radiation are Beta particles. which are electrons produced during nuclear decay. Their low mass results in a stronger scattering by nuclei, than heavier particles. This is the reason that incident electron tracks are non-linear. Apart from delta rays beta particles also produce bremsstrahlung photons. Gamma rays can also cause ionization, although photon absorption cannot be described by LET. Photon interacts in three ways with matter: photoelectric effect, Compton effect or pair production. In biological systems, the Compton effect is the most probable interaction of gamma rays or high energy X-rays. Photon energy transfer is best described by an exponential relation

$$I(x) = I_0 \cdot e^{-\mu x}$$

where the absorption is described by the coefficient μ or the half value thickness. The meaning of LET in the said circumstances is related to the secondary Compton electrons produced by the primary photons and are the main contributor of ionization. Photon radiation attenuation has little to do with gamma LET, but it could have a relation to the defects in the microstructure in the absorber. Photon scattering produces a spectrum of secondary electrons with varying LET values and tracks, so the "gamma LET" is an averaging value.

Regarding the relation of LET and RBE

Interestingly, after a given point, higher LET results in decreased relative effectiveness. Such behavior reflects the decreased efficiency of higher LET radiation damage. The key mechanism of biologic damage, double strand DNA breakage, is maximal at a LET of 100 keV/um Examples of 100 keV/um radiation include 300 KeV neutrons, low energy protons, and low energy alpha particles.

For this average LET, spacing of ionizing events equals the diameter of a DNA double helix. Greater LET energies may have closer spacing of ionizations, but such additional path ionizations do not yield additional double strand breaks.

1.4 Particle irradiation

Whereas photon radiation decreases while traversing matter according to the negative exponential law we described above, a charged particle exhibits a different absorption pattern. When a fast charged particle traverses a material, it ionizes the atoms along its track, depositing energy. The energy deposition as expressed by LET is not constant, it increases along with the cross section as particle's energy decreases as shown in fig 1.2. The charged particle's energy loss is inversely proportional to square of its velocity, that's the reason, that the LET curve gradually increases up to a peak, referred to as Bragg peak.

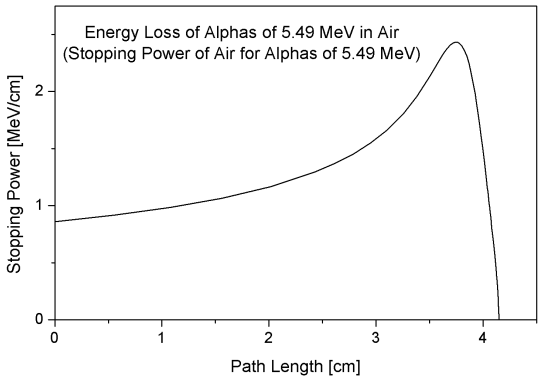


Figure 1.2: Bragg Peak

1.4.1 irradiation modalities

The sharpness of a monoenergetic ion beam Bragg Peak, indicates that most of the energy of the beam is deposited in a narrow depth of only a few millimeters. This has a major consequence in Radio Therapy, since greater distances are warranted in order to encompass the tumor. This is the reason that several techniques are applied [7] in order to produce a spread out Bragg peak (SOBP), namely using particles with somewhat different energies(fig 1.3). Various energies are used with an appropriate weighting to produce a flat SOBP. This property can allow normal tissue to avoid radiation damage.

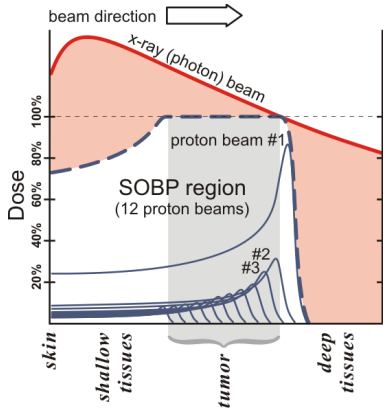


Figure 1.3: Spread Out Bragg Peak

1.5 Cell line

Radiation affects cells in multiple ways including cell damage, apoptosis, mutations and chromosome aberrations. Cells exhibit varying responses to low doses of radiation which cannot be extrapolated accurately from high dose responses with models like the linear no-threshold model [8]. Cells appear to have in many degrees, the capacity to an array of radioadaptive respones to low dose radiation, which can be lead to low-dose hyper-radiosensitivity (HRS) and increased radioresistance (IRR). Radioadaptive response can protect against many challenge. Two mechanisms have been suggested: an intracellular and an inter-cellular one. The first involves cellular changes as a results of radiation. The latter induces a protected state to cells that are yet to be impacted from radiation. Radioadaptation is observed in response to both low and high LET, and its exact shape is highly variable, depending, among other factors on, the cell line [9].

The mechanisms involved in radioadaptation are far from being fully understood. Intracellular response relates to multiple pathways being activated like the temporary suppression and induction of genes. This state is referred to as "memory" because it's the consequence of a previous radiation exposure that leads to protection from futures challenges [10], notable repair mechanisms in mammalian cells are homologous recombination and nonhomologous end-joining. Intercellular response revolves around ways that an irradiated cell emits signals, inducing an adaptation state in other cells that have not yet been impacted [11]. Induced adaptation can occur either with gap junctions to neighboring cell, or with diffusible factors to cells farther away from the impact of the radiation [12].

1.6 Cell phase

In order to maintain the integrity of the DNA after irradiation, multiple repair mechanisms are involved as discussed above [13]. Among the most significant mechanisms is cell cycle regulation which determines radiosensitivity [14]. With the onset of radiation cell cycle checkpoints are activated. This signifies either the initiates a process that start with signals that will activate either temporary checkpoints which induce genetic repair or irreversible growth arrest and necrosis. An activation checkpoint example that forms an integrated response involves sensor (RAD, BRCA, NBS1), transducer (ATM, CHK), and effector (p53, p21, CDK) genes. The tumor suppressor gene p53 plays a key role in this example [15], as it coordinates DNA repair with the progression of the cell cycle and apoptosis [16]. Moreover, p53 mediates the two major cellular checkpoints that relate to DNA damage, one at the G(1)-S transition and the other at the G(2)-M transition, with the influence on the former process being more direct and significant. Cell cycle phase also determines a cell's radiosensitivity, with the G(2)-M phase being the most radiosensitive, followed by the G(1) phase, and the latter part of the S phase being the least sensitive. All these exerted influence on the way radiotherapy is applied, i.e therapy should be synchronized with the most radiosensitive cell phase [17].

1.7 Tumor cells

Radiation therapy aims fatally impact to cancer cells through irreversible DNA damage. It has been shown that tumor DNA is more prone to damage either double or single strand breaks and repairs more slowly [18–21]. Furthermore, expression of some genes involved in DNA repair and cell death invoking, decrease the resistance to ionizing radiation in fast doubling tumor cells, while having the reverse effect in slow doubling tumor cells [22]. Therefore, radiation will have a varying effect in the cells. Several studies show that many signaling pathways involving DNA repair show a level of redundancy in non cancerous cells [23]. Due to multiple mutations a tumor DNA suffers there is a loss of this redundancy [22]. A commonly mutated gene in tumors is the p53, which is a transcription factor contributing to cell death and cell cycle arrest and DNA repair. To complicate matters, DNA repair mechanisms in cancer cells can halt DNA damage and lead to an increase in resistance to ionizing radiation [22]. Furthermore, cancer cells, may exhibit adaptation to ionizing radiation which is most harming to fractionation radiotherapy [24]. This

happens, by triggering adaptive cellular responses in tumor cells. In this context, a major clinical concern is the upregulating of signaling pathways such as the PI3K-AKT-mTOR signaling pathway, that renders a cancer cell more radioresistant and able to evade apoptosis [25, 26].

Chapter 2

Methods

2.1 A brief Intro

Our main effort will be the use of several Machine Learning techniques for ascertaining the effect of ioninzing radiation on cell survival. The computational problem would be described as a **multivariate reggression**, in the sense that, we are trying to predict a continuous quantity with more than one independent variables. Our reasoning is that, combining predictive models can under circumstances, provide a better predictive model. We have compiled two datasets, in the first, the radiation effect is expressed through the coefficients of the quadratic model (α_l,β_l). In the second dataset, the radiation effect on cell survival is expressed through the relative biological effectiveness (RBE) quantity.

2.2 Datasets

We could, restrict ourselves to only one dataset, namely, the one that contains the RBE, because we could derive the respective RBE values from the coefficients of the quadratic model. We considered, on the other hand, that in that case, we would loose the biophysical importance of the information provided by the coefficients of the quadratic model. So, we ended up, constructing a bigger RBE dataset that contains the RBE as the dependent variable, and has data originating both from RBE studies and quadratic model ones (substituting the quadratic coefficients with RBE). We also have a pure quadratic coefficient dataset, whose data were taken from the Particle Irradiation Data Ensemble (PIDE) dataset provided by the GSII in Germany [27]. So we have two almost identical datasets with the RBE dataset consisting with 1000 datapoints and the quadratic dataset with 855 datapoints.

The datasets are compilations of cell survival studies. Each study reports its statistical errors, which indicate a lower limit to the uncertainty of the data. There is not a unique established way of reporting error in survival studies, that is why a broader uncertainty analysis is needed as both statistical and systemic errors should be considered. The overall uncertainty could be determined by compiling larger datasets from experimental data from different laboratories, obtained under similar conditions. From such a dataset, the scatter of the data would lead to an empirical measure of the uncertainty. This poses a problem to the quality of our datasets, and hints to the limitations of the ML models we will build on them.

Samples of the datasets

- RBE dataset sample:

Cell	**Type**	**Phase**	**Genl**	**Ion**	**Charge**	**Irmods**	**LET**	**E**	*RBE*
V79	n	G0	5.6	4He	2	s	2.34	133.4	1.0118
T1	n	a	5.6	20Ne	10	m	71.0	102.5	1.9875
HE20	n	G1/G0	6.0	20Ne	10	m	63.0	120.3	2.2633

- Quadratic dataset sample:

Cell	**Type**	**Phase**	**Genl**	**Ion**	**Charge**	**Irmods**	**LET**	**E**	α_l	β_l
V79	n	G0	5.6	4He	2	s	2.34	133.4	0.06	0.04
T1	n	a	6	12C	6	s	12.16	319	0.323	0.0556
HE20	n	G1/G0	6	20Ne	10	m	63	120.3	1.386	0.104

Description of inputs

Var_Name	**Description**	**categorical\numeric**
Cell	Name of cell line	categorical
Type	Tumor cells (t) or normal cells (n)	categorical
Phase	Cell cycle phase (phases are given explicitly, or 'a' for 'asynchronous')	categorical
Genl	Genomic length of diploid cells (in 10 bp, 5.6 for rodent and 6 for human cells)	numeric
Ion	Ion species	categorical
Charge	charge of ions	numeric
Irrmods	Irradiation modalities: monoenergetic (m) or spread out Bragg peak (s)	categorical
LET	Linear energy transfer in water (in keV/ m, for irradiation in spread out Bragg peak dose mean or track averaged LET)	numeric
E	Specific energy of ions (in MeV/u), evaluated at the target	numeric

2.3 Algorithms

We will address the problem using several separate algorithms, the gradient boosting decision trees (GDBT), the Support Vector Regression (SVR) and the Random Forest Regression (RF) from the SciKit library, will be compared with another Gradient Boosting Decision Tree algorithm from CatBoost [28] (referenced as Catboost from now on). Our goal is to inquire whether ensemble techniques, employing multiple algorithms can perform and generalize better than individual algorithms. For this reason we implemented a Voting Regression algorithm that merges two or more estimator models to produce a more robust one. In our cases we chose another implementation of the GBDT,SVR and the RF algorithms will form the constituent models of the Voting Regression algorithm.

A side effort is demonstrating the usefulness of the Catboost Gradient Boosting Decision Tree algorithm, while the other models, will play a benchmark role that will corroborate our results. We choose these algorithms for various reasons. Firstly, as you can see the datasets are relatively small for Machine Learning standards, and the use of neural network architecture requires orders of magnitude more data to realize its potential. Secondly, CatBoost offers certain advantages that will be mentioned bellow.

2.3.1 Gradient Boosting Desicion Trees

Gradient boosting Desicion Trees [29] is an ensemble machine learning technique for regression and classification problems. The prediction model is produced by a set (ensemble) of weak predictors, in our case decision trees, that are iteratively optimized by minimizing an arbitrary differentiable loss function. That is by changing the parameters of the weak predictors in the direction of the negative gradient of the loss function.

the algorithm:

Input: training set $\{(x_i, y_i)\}_{i=1}^{n}$, a differentiable loss function $L(y, F(x))$, number of iterations M.
Algorithm:

1. Initialize model with a constant value:

$$F_0 = \underset{\gamma}{\operatorname{argmax}} \sum_{i=1}^{n} L(y_i, \gamma).$$

2. *For* $m = 1$ *to* M :

   (a) Compute so-called pseudo-residuals:

$$r_{im} = -[\frac{\theta L(y_i, F(x_i))}{\theta F(x_i)}]_{F(x)=F_{m-1}(x)} \ for\ i\ =\ 1, ..., n.$$

   (b) Fit a base learner (e.g. tree) $h_m(x)$ to pseudo-residuals, i.e. train it using the training set $(x_i, r_{im})_{i=1}^{n}$

   (c) Compute multiplier γ_m by solving the following one-dimensional optimization problem:

$$\gamma_m = \underset{\gamma}{\operatorname{argmax}} \sum_{i=1}^{n} L(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)$$

   (d) Update the model:
   $F_m(x) = F_{m-1}(x) + \gamma_m h_m(x)$.

3. Output $F_M(x)$.

Figure 2.1: Gradient Boosting Algorithm

GBDT build trees one at a time, where each new tree helps to correct errors made by previously trained tree. The total expressiveness of the model increases with every tree added. The main parameters that characterize a GBDT model are the number of the trees, the maximum depth of the trees and the learning rate. Generally, the trees in GDBT models are relatively swallow. Individual trees are built in sequence, that is why training takes longer. However benchmark results have shown GBDT are better learners than other models like Random Forests.

Other significant advantages of the GBDT algorithms are that, GBDT are not affected by feature multicollinearity (statistical coupling of two or more features) like linear regression, and are applicable even in a case where the features are greater in number than our datapoints. The latter part is of great use to us because our datasets contain a great deal of categorical features, which (especially the cell name feature) exhibity high cardinality, and one-hot encoding them, produces the dataset, that we actually fit out model on, which contains a far greater number of features.

2.3.2 Support Vector Regression

Support Vector Reggression is very similar to SVMs (support vector machines) [30] which is a well established technique for classification problems. So the explanation of the workings of the SVR as a method is

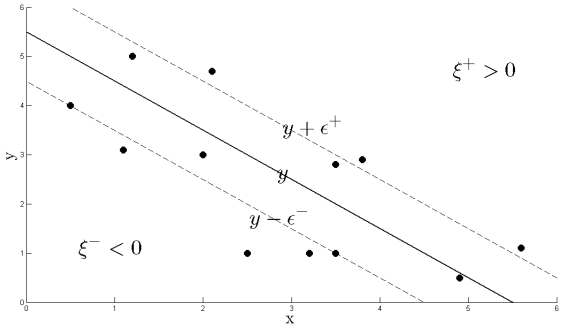


Figure 2.2: Support Vector Machines Algorithm

almost identical to SVMs.

An SVR model is a representation of the examples as points in space(fig 2.2), mapped so that the examples are divided by a clear gap that is as wide as possible from a learned regression line $y_i = \mathbf{w} \cdot \mathbf{x} + b$. Output variables which are outside the $|t_i - y_i| < \epsilon$ area are given one of two variable penalties depending whether they lie above (ξ^+) or below (ξ^-). We define another parameter C that controls the trade-off between the penalty ξ and the size ϵ of the margin area. The regression problem is translated to minimizing the SMR error function, written as:

$$C \sum_{i=1}^{L} (\xi_i^+ + \xi_i^-) + \frac{1}{2} \|\mathbf{w}\|^2$$

Through which we calculate $\mathbf{w}$ and b. More information here

A main feature of the SVRs is that due to the nature of Convex Optimization, the solution is guaranteed to be the global minimum not a local minimum. SVR is useful for both Linearly Separable(hard margin) and Non-linearly Separable(soft margin) data. In terms of parameters that can be tweaked, its essentially only the variable C which multiplies the slack variables. Features are mapped in a implicit way using simple dot products. On the other hand, SVR in its simplest form cannot return a probabilistic confidence value like logistic regression does, lacking in explanatory power.

2.3.3 Random Forest Regression

A Random Forest (RF) is a decision tree ensemble technique capable of performing both regression and classification tasks [31]. Decision trees models that grow deep trees tend to overfit on the data. RFs combat this, by using the Bootstrap Aggregation technique, commonly known as bagging. Bagging, in the Random Forest method, involves training each decision tree on a different data sample where sampling is done with replacement. The basic idea behind this is to combine multiple decision trees in determining the final output rather than relying on individual decision trees. Due to the randomness of the sampling, the technique is less likely to overfit on the training data. Bagging is performed many times and each time a regression tree f_b is fitted on a random sample of a dataset. The actual prediction of an unseen sample x' is the averaging of the individual trees predictions:

$$\hat{f} = \frac{1}{B} \sum_{b=1}^{B} f_b(x')$$

Bootstraping leads to better predictive capacity of the model because it decreases the variance of the model without increasing the bias. This is because whereas a single tree is sensitive to the noise in the dataset, the average of many trees is not, as long they are not correlated. Training many trees on the same dataset, will produce strongly correlated trees and sometimes the same trees. Bagging aims to de-correlate the trees by training them on different datasets. Another way to produce non correlated trees in RFs is the random sampling of the features of the dataset. If one or more of the features are the main contributors of the prediction, then these features will be selected in many of the individual trees, leading them to become correlated. RFs are a powerful and accurate method, and also exhibit good performance on many problems including non linear ones.

2.3.4 Voting Regression

A voting regressor (VR) is an ensemble meta-estimator that fits base regressors each on the whole dataset. Based on the individual predictions, it then decides for a final prediction. There several varieties of decision making in the voting framework algorithm depending on the voting mechanism. These include, hard voting, weighted majority vote and soft voting.

- Hard voting

  Hard voting is the most straightforward way of making a decision on the base regressor's decisions. In hard voting, we predict the class label $\hat{y}$ through a majority vote of each classifier C_j:

  $$\hat{y} = mode\{C_1(x), C_2(x), ..., C_m(x)\}$$

  If we assume that there are three base classifiers/regressors:

  classifier 1 $\rightarrow$ class 0

  classifier 2 $\rightarrow$ class 0

  classifier 3 $\rightarrow$ class 1

  $$\hat{y} = mode\{0, 0, 1\} = 0$$

  Majority vote implies that the sample is classified as class 0

- Weighted majority vote

  We can also compute a weighted majority vote, if we assign a weight w_j to each classifier C_j:

  $$\hat{y} = \underset{i}{\text{argmax}} \sum_{j=1}^{m} w_j \chi_A(C_j(x) = i)$$

  where χ_A is the characteristic function $[C_j(x) = i \in A]$, and A is the set of unique class labels. Based on the previous example:

  classifier 1 $\rightarrow$ class 0

  classifier 2 $\rightarrow$ class 0

  classifier 3 $\rightarrow$ class 1

  assuming weight values as {0.2,0.2,0.6} would output a prediction $\hat{y} = 1$.

- Soft Voting

Here, we predict the class labels. based on the probabilities each classifier produces for a given sample.

$$\hat{y} = \underset{i}{\operatorname{argmax}} \sum_{j=1}^{m} w_j p_{ij}$$

where w are the weights assigned to the classifiers. Based on the previous example:

$C_1(x) \rightarrow [0.9, 0.1]$
$C_2(x) \rightarrow [0.8, 0.2]$
$C_3(x) \rightarrow [0.4, 0.6]$

assuming the weights follow a uniform distribution, we compute the average probabilities:

$$p(i_0|\mathbf{x}) = \frac{0.9 + 0.8 + 0.4}{3} = 0.7$$

$$p(i_1|\mathbf{x}) = \frac{0.1 + 0.2 + 0.6}{3} = 0.3$$

$$\hat{y} = \underset{i}{\operatorname{argmax}}[p(i_0|\mathbf{x}), p(i_1|\mathbf{x})] = 1$$

In our case, we used as "constituent" estimators SciKit's Gradient Boosting Decision Trees, Random Forests Regression and Support Vector Regression implementations mainly because, the VR is provided by SciKit, which offers a nice integration of these models to the VR, and we could also perform an optimized parameter search for our models with SciKit library, instead of tweaking each parameter of the model by hand.

At this point, the models are saved, and we can make actual predictions. The program, handles user input one row at a time but it could have also a bulk predict parameter. The GDBT, RF and SVR models contribute to averaged soft voting prediction, as to gain improved generalization. In essence we have built a combined model, consisting of three models of radically different approaches, that ensures some robustness and optimization of the final model.

2.3.5 Catboost Gradient Boosting Decision Trees

Catboost algorithm is a modification on the classic GBDT algorithm in two respects. The first is the way the actual gradient descent is implemented. The developers of the said algorithm [28], state that a problem in all other GBDT models is the prediction shift caused by the target leakage. This means that t_{th} base estimator

$$h^t = \underset{h \in H}{\operatorname{argmin}} \mathbb{E}(-g^t(\mathbf{x}, y) - h(\mathbf{x})^2)$$

where $g^t(\mathbf{x}, t)$ is the gradient of the the loss function of the previous estimator, and $(\mathbf{x}, y)$ is a test example sampled from the dataset of size n, is normally approximated with

$$h^t = \underset{h \in H}{\operatorname{argmin}} \frac{1}{n} \sum_{k=1}^{n} (-g^t(\mathbf{x_k}, y_k) - h(\mathbf{x_k})^2)$$

where $(\mathbf{x_k}, y_k)$ is part of the training data set. This leads to a shift from the distribution

$$g^t(\mathbf{x}, y)|\mathbf{x}$$

to the distribution

$$g^t(\mathbf{x_k}, y_k)|\mathbf{x_k}$$

which in turn, offers a biased solution for h^t and affects the generalization of the trained estimator. To address this problem, another kind of gradient boosting was introduced called Ordered Boosting. In essence, if we want to train the F^t estimator unshifted, we have to train F^{t-1} without the example x_k. This is achieved by taking random permutations of the training samples and building supporting models $M_1 \dots M_n$ where the model M_j is trained using the first j samples in a given permutation. At every step, to get the estimator F^j we use the supporting model M^{j-1}. The overall training process is computationally more taxing than the plain GBDT algorithm, but it fixes the overfitting problem that GBDT models exhibit, especially in smaller dataset like the ones we study here.

Another key feature of the Catboost algorithm is the encoding of categorical variables in the dataset, using target statistics. A common practice to deal with a categorical feature i is to replace the category x_k^i of the k-th training example with one number equal to s certain target statistic(TS) $\hat{x}_k^i$, this denotes the expected target y conditioned by the category $\hat{x}_k^i \approx \mathbb{E}(y|x^i = x_k^i)$. Catboost uses the Ordered TS strategy. Again the training examples go through permutations to avoid the reliance of the TS to the observed history.

2.4 Grid Search & Cross Validation

In order to fine tune our models, we implemented a thorough search for the appropriate hyper-parameters for each model. Trying blindly to find the parameters for each model that optimizes its predictive performance is not the best practice. To overcome this, we implement a technique called grid search with cross validation [32]. Hyper-parameters are parameters that are user-defined and are not directly learnt by the estimator. Typical examples of hyper-parameters are the *kernel* and *gamma* for SVMs and *number of estimators* for decision trees. The Grid consists of the various parameters that we experiment on, creating a hyper-parameter space, over which the algorithm is trained and its cross validation score provides an evaluation of the hyper-parameters used.

When evaluating hyper-parameters for estimators, such as the C setting that must be manually set for an SVM, this is both laborious and prone to overfit on the test set because the parameters can be tuned until the model performs best. The, knowledge about the test set can leak into the model making the evaluation metrics irrelevant to how well the model generalizes. To address this problem, yet another part of the dataset can be held out as a "validation set": training takes place on the training set, after which, evaluation is done on the validation set, and if the experiment appears to be a success, the final evaluation can be done on the test set.

Understandably, partitioning the available data into more sets, is detrimental to the learning procedure because we reduce the number of samples, and the results can depend on a particular random choice for the pair of (train, validation) sets. Here, steps in a technique called cross-validation (CV for short). We still hold out a test set for final evaluation, but we don't permanently hold out a validation set when doing CV. In the basic approach, called k-fold CV, we split the training set into k smaller sets of equal size. K-fold Cross Validation consists of the following steps:

- A model is trained using $k - 1$ of the folds as training data;
- The resulting model is validated on the remaining part of the data (i.e., it is used as a test set to compute a performance measure such as accuracy).

We perform the previous 2 steps k times and the overall performance is measured as the average performance computed by its fold, this is called k-fold cross-validation score. This approach can be computationally taxing. However CV does not waste data (as is the case when fixing an arbitrary validation set), which is a major advantage in problems such as inverse inference where the number of samples is very small, which is exactly our case.

2.5 Confidence Intervals

The same process that gives the optimal parameters of a model, produces the confidence intervals of the model. In fact, during the grid search and cross-validation procedures, a distribution of the error emerges. We can take the mean of this distribution as the most representative metric of the error of the model, and can use the standard deviation of the distribution to calculate the intervals in which the mean error will be more likely to lie in. Confidence intervals, indicate how certain we are for the value of the mean of a sample. In the results section, confidence intervals are shown for each model. This makes a useful insight for how dispersed is the error of the model. The length of the interval is taken to be a standard deviation, because we cannot infer that the cross-validation scores follow a normal distribution and translate the standard distribution to a percentile, the same approach is followed bellow with the prediction intervals.

2.6 Prediction Intervals

It would be inadequate and not so informative to output a prediction, without an uncertainty attached to it. A prediction interval signifies, where the next value sampled is expected to lie in. The distinction between prediction and confidence intervals, is that prediction intervals informs on the distribution of values and not on the uncertainty of the population mean. We assume that the data sampled are gaussian and its features independent. For each model, a model is trained to learn the differences between the true values and the predicted ones. In that sense, this model (called error model) produces the distribution of errors of the initial model, and given this distribution, it is feasible to compute the standard deviation. This approach is called the Distribution Estimator approach. It is a very generic process applicable to any model, which is desirable for our task given that we handle a multitude of models. The assumption that the data sampled follow a normal distribution is not fulfilled in our case by any means. In fact a Shapiro-Wilks test [33] was performed on the sample data and the result was 0.009, which denies any possibility of a gaussian distributed sample. In these cases a more advanced statistical analysis of the prediction intervals is required falling in the more general category of exponential family of distributions analysis, to have a more mathematically rigid methodology.

Nevertheless, the Distribution Estimator results seem to be satisfactory in respect to the usual methods applied to obtain prediction intervals, like quantile regression, that is available for Gradient Boosting. The program outputs its prediction and the prediction intervals associated with it. In our case, the intervals of the models cannot be quantified in a percentile way, because the distribution is not normal, although the measure of the intervals is the length of a standard deviation above and below the predicted value.

An exception to the above rule is the way the prediction intervals of the Catboost models are acquired. A more straightforward approach was used, simply because given that Catboost is not part of the Voting model,it is possible to extract the intervals in their native way and not implement a workaround as the above method. Here the prediction is made using the quantile loss function with an alpha = 0.5, which means that, the model predicts the value where half the predictions are below and the other half above it. Similarly, we predict the values for the lower limit, i.e. 5% of the values are below the lower limit, and the upper limit respectively where 95% of the values are below it. Hence the prediction has two limits that account for a 90% interval. We cannot directly compare the intervals between catboost and the rest models.

We can summarize the whole model building process with the flow chart below(fig 2.3):

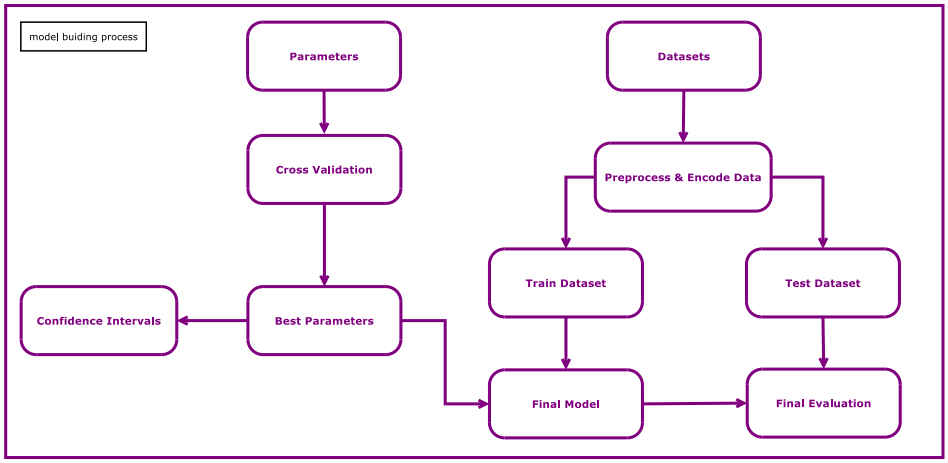


Figure 2.3: Model Building Flowchart

2.7 Program Pipeline & Data Flow

Performing predictions and outputing intervals, involves several stages:

2.7.1 Building the models

We start by handling the data, where a main concern is to transform the categorical features to numerical ones. Due to high cardinality of the features i.e. (high number of discrete categories in relation to dataset) we used the One Hot Encoding technique, which substitutes each categorical feature column, with a number of numerical columns that can hold only 0 or 1 as value. Based on the number of categories in a categorical column, the process makes equal number of numerical columns.

height	gender	age
180	male	40
168	female	56
192	female	85

→

height	gender_female	gender_male	age
180	0	1	40
168	1	0	56
192	1	0	85

The encoding of the categorical features produces a dataset that is specific to our dataset, and this forces us to save the state of the encoder of each dataset and apply it for our predictions.

As a next step we split our datasets to train and test subsets. The train dataset is the actual dataset that gets fitted by our model, while the test subset serves as a set that the model doesn't have knowledge of, and is iteratively evaluated against, to prevent overfitting behavior, and improve generalization of the model.

As a final step we, fit each model and save it, afterwards we get crucial explanatory quantities (feature importances, interactions, training and learning assessments) and plot them. This is the moment where we refine our models and tweak the model parameters to get satisfactory results.

Handling user input data

As a final point, we have taken into account that, in order to make the model even more general in its use, the user may want to use our application but lacks some of the features that are present in our dataset. In

that case, we employ the **K-nearest neighbour** [34] technique so that we can impute the missing features from our dataset. The technique consists of the simple notion that we transform the features of our dataset to vectors. Each vector contains the feature values of each row in the dataset and these correspond to points in the feature space. We use a metric to determine the distance between these points, and accordingly we can deduce the feature value that is missing from a row in our dataset, minimizing this distance. We do this knowingly that prediction has an extra bias originating from our dataset and in this case the results should be taken with a grain of salt.

We give again the respective flowchart(fig 2.4), for the whole prediction process:

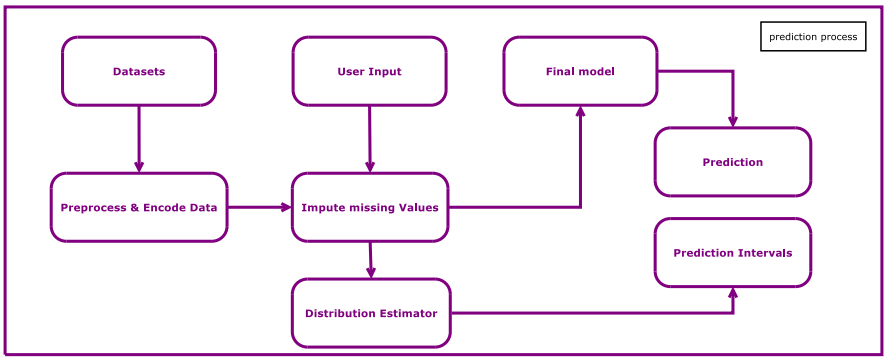


Figure 2.4: Prediction Flowchart

Chapter 3

Results

The evaluation of the implementation of the aforementioned algorithms is not a trivial task because a regression learning model can have many attributes and can be evaluated along different aspects. Here, are presented some basic evaluation aspects like the confidence intervals, the predictive performance and the learning curves of the models. We will also share deeper insights to our models like feature importances, interactions, and training\test curves.

3.1 Model Evaluation & Predictive Performance

Our main effort is to provide more insight to the performance and general model evaluation of the models that are involved. The results shown(fig. 3.1) are products of the Cross Validation process that was followed. The Mean Absolute Error (MAE) scores have the advantage of having the same dimension as the value we are trying to predict with its magnitude being (if nothing goes terribly wrong) relative to our the dependent variable. The table also shows the r^2 scores of the regression models we use the r^2. The r^2 indicates what proportion of the variance of the dependent variable is actually determined by the independent variables, in general terms, it indicates the expressive strength of our model, and it ranges from 0 to 1. As an example, an r^2 value of zero would indicate a model that is constant, predicting always the same values, irrespective of the changing input. The table also shows the confidence intervals for each model which is equivalent to one standard deviation. We consider the mentioned metrics to be a good indicators for the predictive capacity of a model, its propensity to overfit and its overall assessment.

A cross validation process as was described, trains and tests the models several times using different subsets of our data each time and that is why it can give as pretty much accurate picture of how our models behave. The mean absolute errors and r^2 acquired are mean values of the errors and r^2 values produced in the Cross Validation process. The Confidence intervals are the standard deviation of the distribution of errors produced from each model trained during Cross Validation.

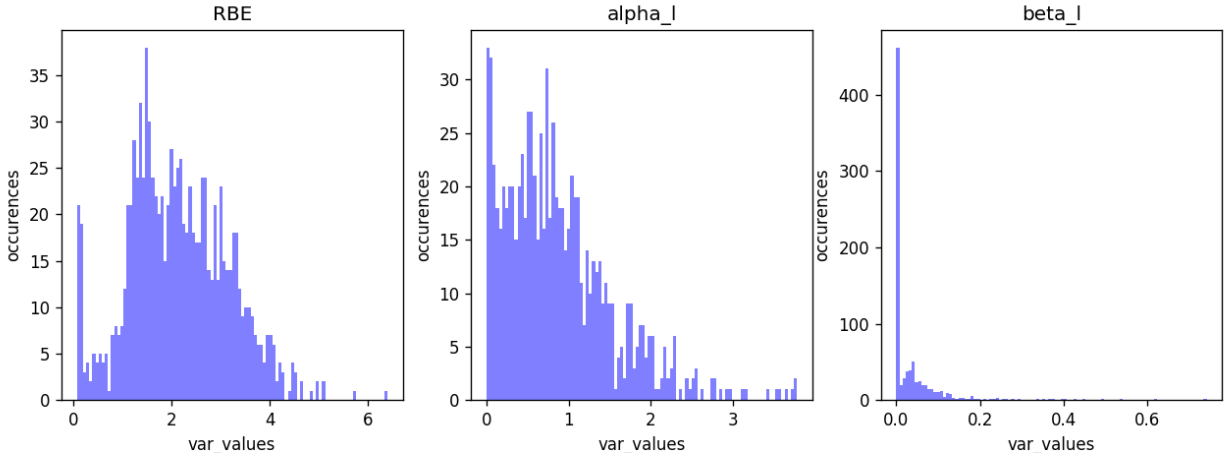
Model Evaluation

	RBE			alpha			beta		
model	MAE	±CI	r^2	MAE	±CI	r^2	MAE	±CI	r^2
Voting Regression	0.42	0.07	0.65	0.35	0.05	0.37	0.04	0.02	-0.41
GradientBoosting Regression	0.38	0.08	0.67	0.32	0.10	0.43	0.05	0.02	-1.34
RandomForest	0.38	0.09	0.67	0.34	0.08	0.42	0.04	0.02	-0.59
SVM Regression	0.59	0.15	0.36	0.47	0.10	-0.24	0.04	0.02	-0.07
CatBoost	0.32	0.03	0.78	0.20	0.03	0.75	0.03	0.01	0.36

Figure 3.1: Cross Validation scores and Confidence intervals

It is obvious that all the models do not fair similarly, except perhaps for the SV regression, which is somewhat worse than the other models in the two of the three cases. Another noteworthy fact is that the Voting regression retains the best performance of its constituents if not slightly a better one. The most noteworthy aspect is the better performance of the CatBoost models, in all the cases.

An obvious problem lies mainly with the prediction of the β_l and to a lesser degree of the α_l quadratic coefficients. The main issue is the negative r^2 values for the β, and the small r^2 values for the α. This indicates that the errors in predictions are large compared to the scale of the values we are trying to predict. In the case of the negative values this so true as much so, that the mean of the data becomes a better predictor than our data-fitted model. We think that this is data related issue, because as you can see from the figure below, the β_l coefficient has a very skewed distribution with the majority of the values being zero or very close to zero, making difficult to predict. A shapiro-wilk test [33] was performed to check for the normality and the results, corroborate what is obvious from the graphs below(fig. 3.2). Shapiro statistic ranges from 0 to 1. The statistic tests the null hypothesis that a sample is came from a normal distribution:



Variable	Shapiro-Wilks	p-value
RBE	0.9886,	5.5048e-07
α	0.9130	1.0432e-21
β	0.5395	1.9253e-42

Figure 3.2: Distribution of dependent variables in datasets & Shapiro-Wilks normality Test

3.2 Prediction results

The actual prediction of the program is done after we have run our models and we execute main.py file, the user is requested for input regarding the radiation and the cell that is irradiated. The result is of the form:

user inputs
cell:V79
tumor or normal cell (t/n):t
phase:G0
genomic length (in millions bp):5.6
ion:4He
charge:2
irmods:s
LET:30
E:120

Prediction Example:

	RBE			alpha			beta		
model	lower	pred	upper	lower	pred	upper	lower	pred	upper
CatBoost Regression	0.751	1.231	1.919	0.086	0.129	0.452	0.005	0.038	0.054
Support Vector Regression	1.154	1.515	1.877	0.393	0.598	0.803	0.059	0.067	0.074
Gradient Boosting Regression	0.853	1.168	1.483	-0.177	-0.011	0.353	0.001	0.037	0.074
Random Forest Regression	1.106	1.432	1.758	0.087	0.403	0.719	-0.034	0.032	0.099
Voting Regression	1.283	1.371	1.459	0.151	0.496	0.84	-0.008	0.032	0.072

Figure 3.3: Prediction results synopsis

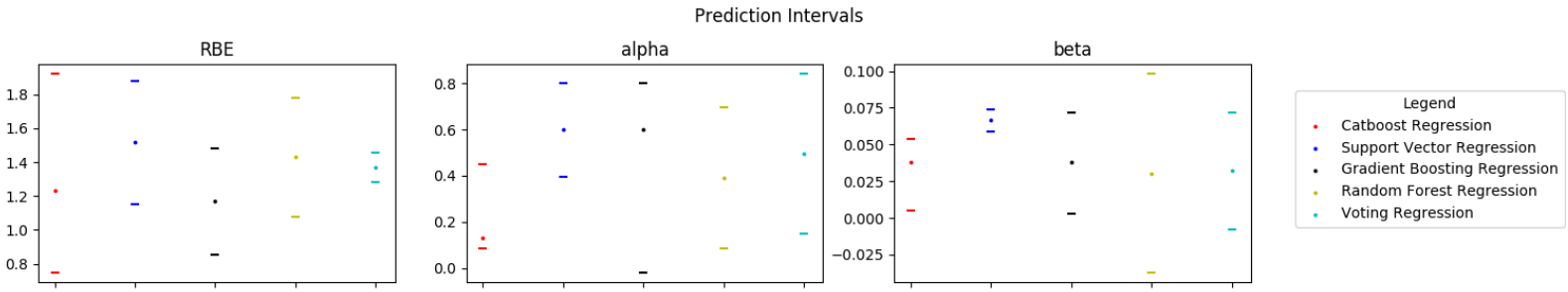


Figure 3.4: Showcase of prediction results and their intervals

3.2.1 Train\Test Performance

Here we will demonstrate how well our models predict values from the test set that was kept out of their training. This offers an objective way of quantifying the predictive performance of our estimators. The plots illustrate the predicted values against the true values.

CatBoost:

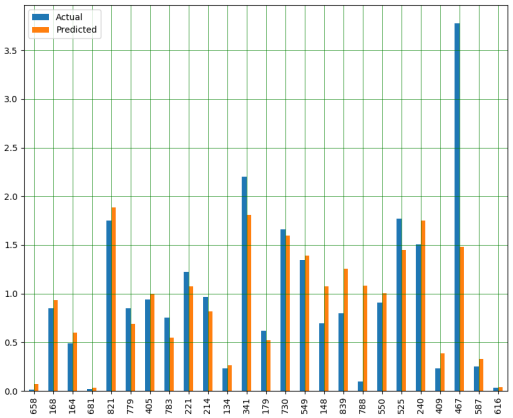


Figure 3.5: α_l

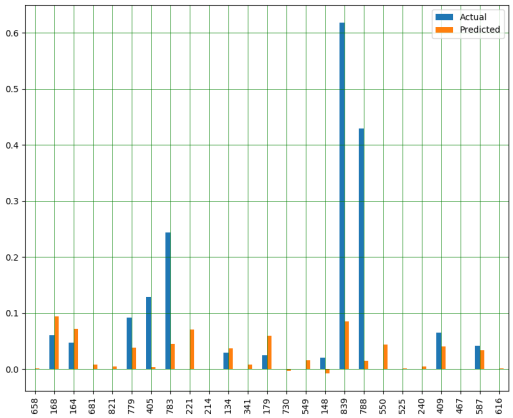


Figure 3.6: β_l

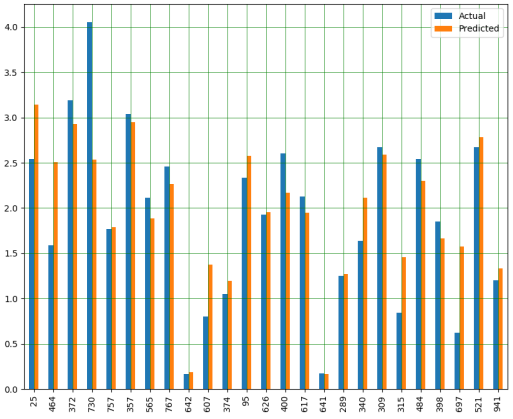


Figure 3.7: RBE

Support Vector Regression:

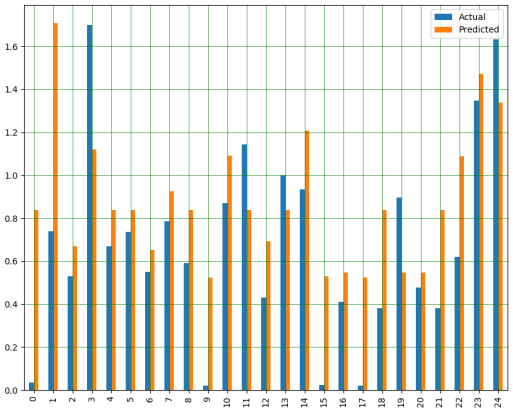


Figure 3.8: α_l

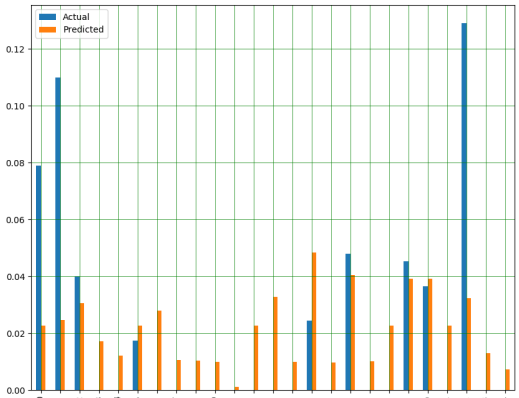


Figure 3.9: β_l

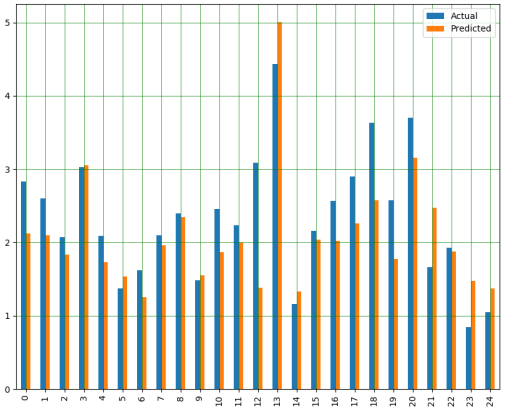


Figure 3.10: RBE

Gradient Boosting Regression:

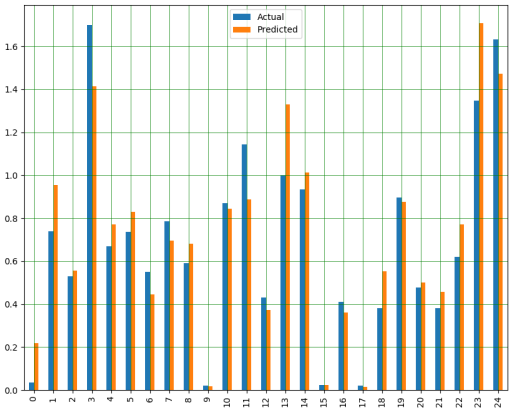


Figure 3.11: α_l

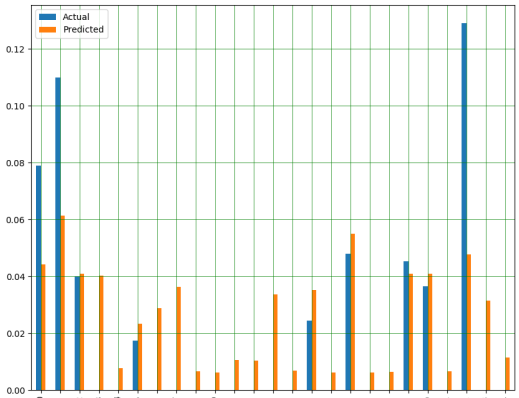


Figure 3.12: β_l

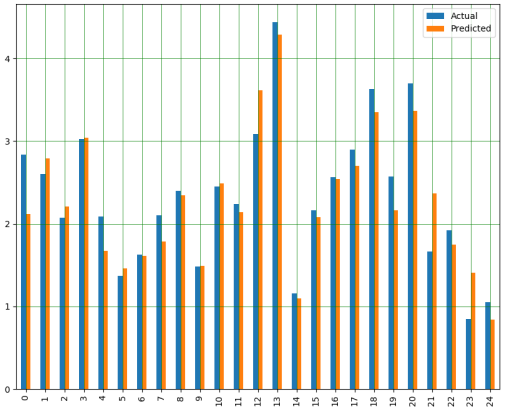


Figure 3.13: RBE

Random Forest Regression:

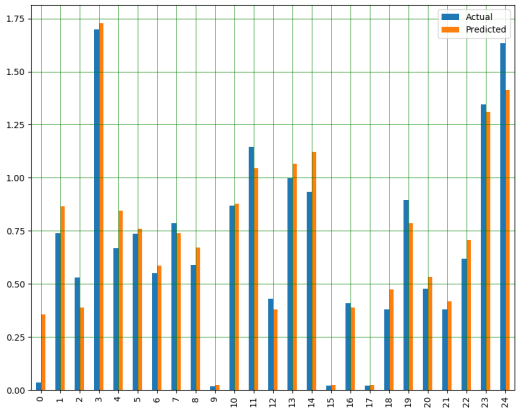


Figure 3.14: α_l

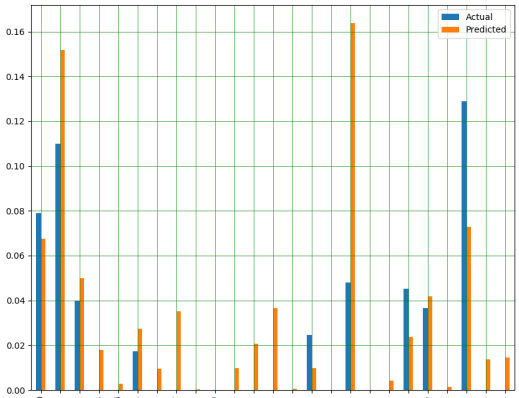


Figure 3.15: β_l

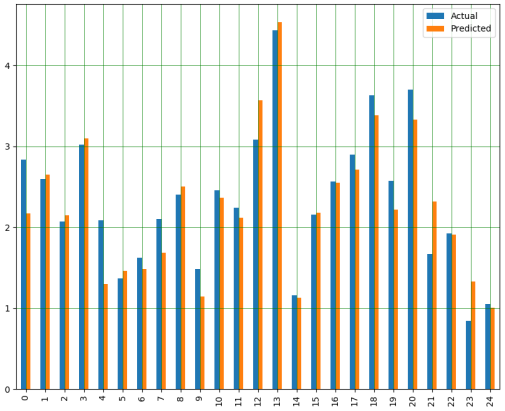


Figure 3.16: RBE

Voting Regression:

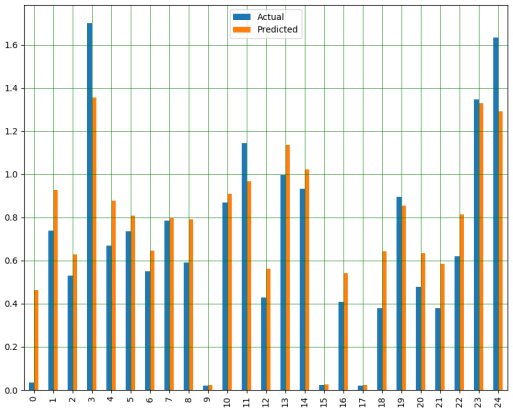


Figure 3.17: α_l

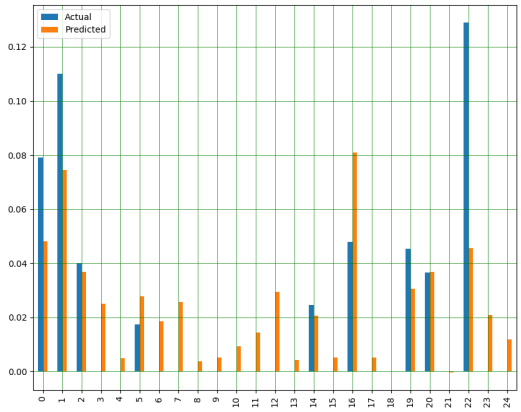


Figure 3.18: β_l

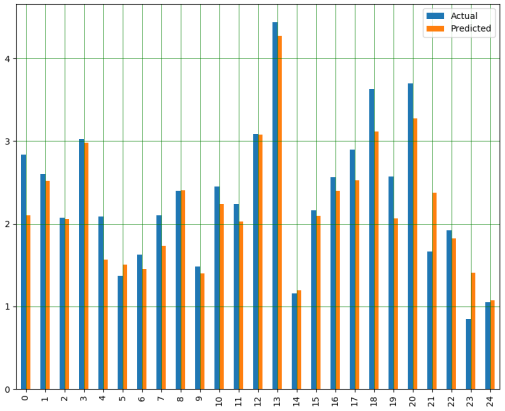


Figure 3.19: RBE

3.3 Learning Performance

Here we will examine how our algorithms learning behaves. There is no single method to quantify how well a predictive algorithm learns during its learning process. That's because each learning procedure can be radically differently from every other. Gradient Boosting algorithms learn iteratively, while SVM and Random Forests don't. Basically, what we want, is to show how an algorithm learns in relation to the evolution of its inner mechanism. In order to do this we will use multiple metrics, which by itself doesn't allow for an out of the box comparison of the Machine Learning techniques used, in terms of evaluating the evolution of learning. We will be based on common metrics such as the root mean squared error ($RMSE$) and the explained variance score, which accounts for the percentage of the variance in the features that is explainable from the model. The procedure consists of either plotting the cross-validation score (in our case

explained variance) of the sum total of the datasets or by plotting the error of the training against the test set. The latter process will be used for the CatBoost algorithm and the former for all the rest algorithms.

3.3.1 CatBoost

Here we demonstrate the Learning Curves of the CatBoost algorithm:

CatBoost Learning Curves:

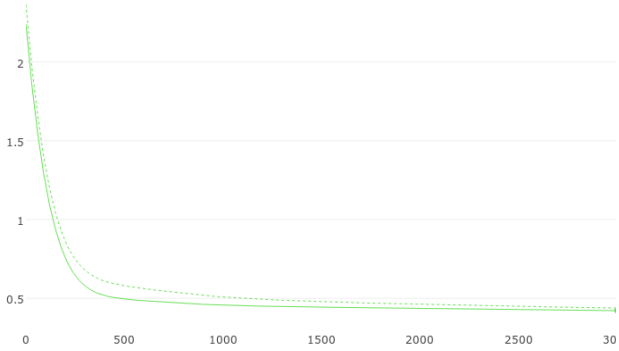


Figure 3.20: CatBoost Learning Curve for the RBE

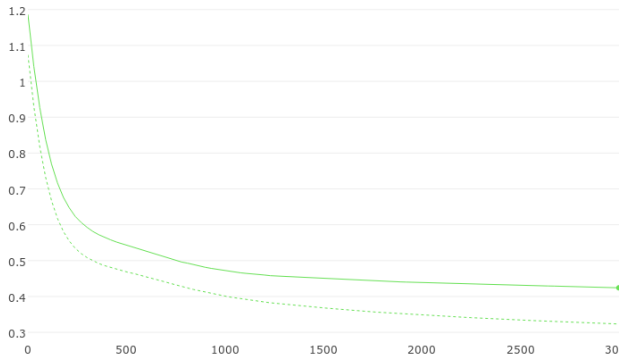


Figure 3.21: CatBoost Learning Curve for the α coefficient

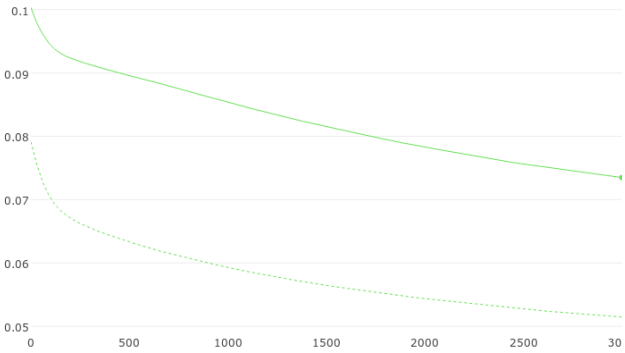


Figure 3.22: CatBoost Learning Curve for the β coefficient

As we can see(fig. 3.20-22) the CatBoost performs very well. On the x-axis are the number of iterations that CatBoost performs, and on the y-axis the $RMSE$ error. The solid line represents the error on the test set which the model has not seen, and the dotted one the learning error. CatBoost seems to do exquisitely good predicting the RBE value, but on the same time fairs pretty well on predicting the coefficients of the quadratic model. The model was trained using 3000 iterations, a learning rate of 0.01, the maximum depth of the decision trees was 10. The algorithm implements a best model usage technique, where the actual model that is used is the on with the lowest error and not the last one trained. This way overfitting is averted.

The $RMSE$ score is comparable to the size of the respective values in our datasets, so its absolute value is not meaningful, on the contrary we have to compare it to the median size of the predicted value. For the RBE, we have to see that both learning error and cross-validation scores settle below 0.5 and given that the RBE ranges between 1 and 4, the error seems to be satisfactory. For the α and β coefficients the case is different. The model trained to predict the α coefficient seems to overfit after 1000 iterations and the testing error increases and overall has a large error relative to the median α on the dataset. The model for the β coefficient seems to lower its errors overtime but it is still large relative to the median β in the dataset.

3.3.2 Learning Curves

We will employ learning curves to demonstrate certain aspects of the learning procedure of the algorithms. To caclculate the true value of our target, we use different methods like Gradient Boosting Decision Tress and Support Vector Regression. Our guess is that the relationship between our dataset features and our target is a complicated one and far from being linear. As quality measures for our model we can introduce the notions of bias and variance, of our model's estimation and and the true values. We could conceptualize a low bias model like in the figure below.

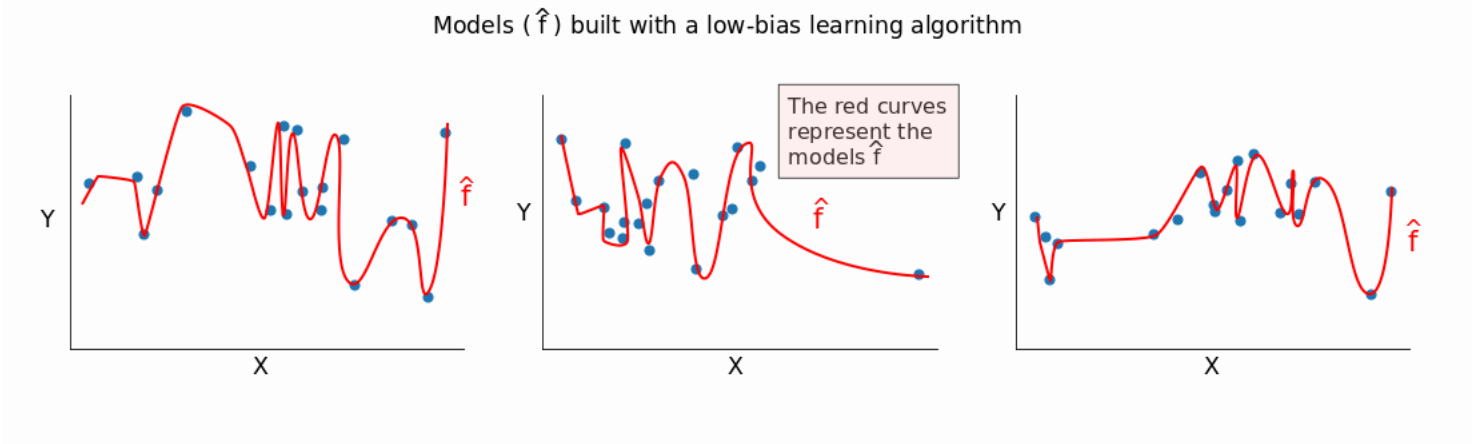


Figure 3.23: Low bias example models

A model with low bias has the capacity to capture even the most minute differences, our model is varies significantly as we change the dataset. There is an inverse relationship between variance and bias. Inversely, a model with high bias, doesn't fit the data well, but the model isn't as significantly variable to changes in the dataset, a low variance model is simpler one that cannot capture non linear feature relationships.

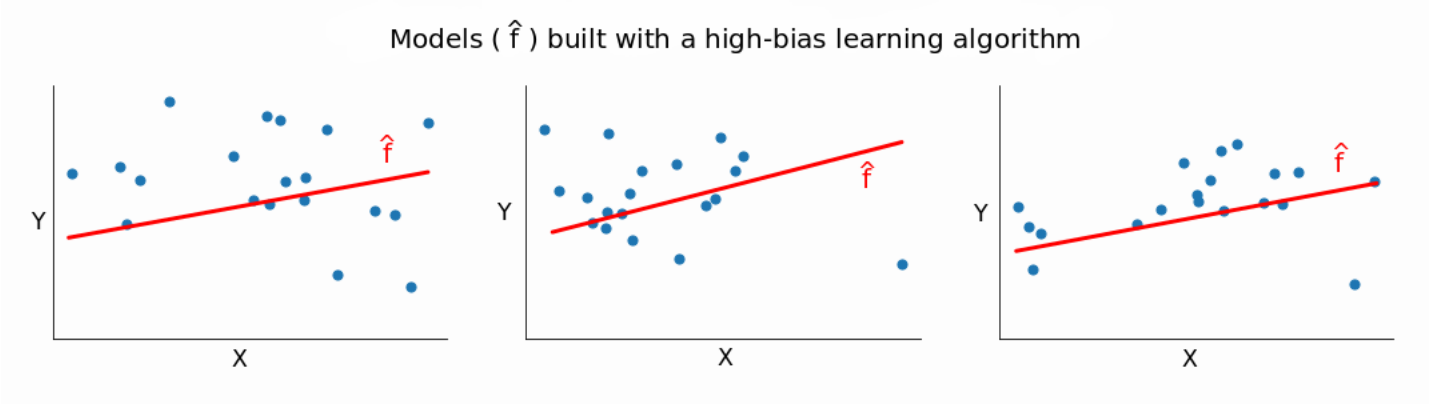


Figure 3.24: High bias example models

Ideally, we want a low variance and low bias model. It's clear that variance and bias add to the overall error of out model. As we don't want a high bias model that's too simplistic for our needs and fits poorly on our data, similatly we don't want a high variance model, that fits perfectly on our data, capturing all the noise that's inherent into that dataset. This noise capturing, overly complex model, also performs poorly on out-of-the-sample data. In practice, we a seek a point of an honorable compromise, between bias and variance, like in the figure below.

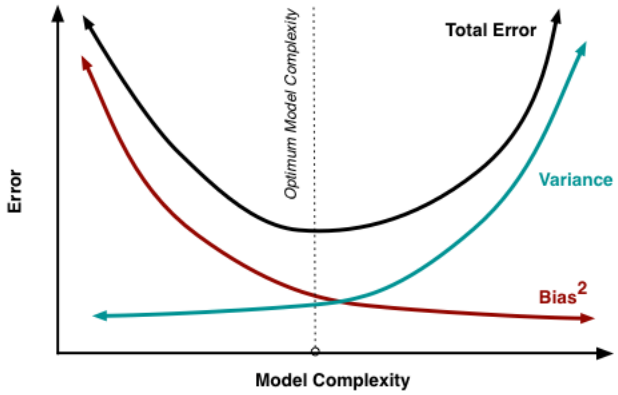


Figure 3.25: Complexity - error trade off

We can illustrate this behavior if our models, using learning curves. We train our models on, gradually larger and larger parts of our data and monitor how the errors on the training set and on the validation set evolves, i.e., we have two curves that show the training and validation errors depending on the training size. The validation error, is not to be mistaken with the cross-validation mean error of the training samples, which we use to determine the training error. The validation error is the error, of the model in predicting the target value of the test set that we have set aside from the beginning. If the training error is relatively low, it means that the model fits the training data very well, which signifies a low bias model. High validation errors indicate a bias problem but don't give the directionality of the bias. Variance is indicated by the gap between the two curves. The smaller the gap, the smaller the variance. The evolution of the curves can importantly indicate also if further training samples can improve our model.

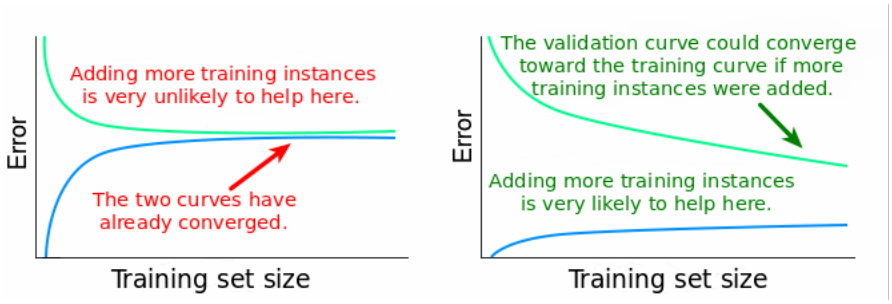


Figure 3.26: High bias example

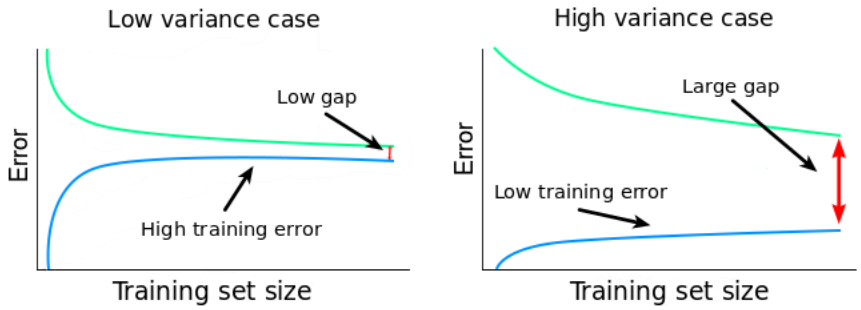


Figure 3.27: Variance and gap size, example

3.3.3 Random Forest

Here we will demonstrate the learning behavior of the Random Forest algorithm using learning curves. Here we take a different approach from the CatBoost algorithm. We will use the training and validation scores using the learning_curve functions provided by SciKit which performs a cross-validation to output the training error. As we can in the figures below.

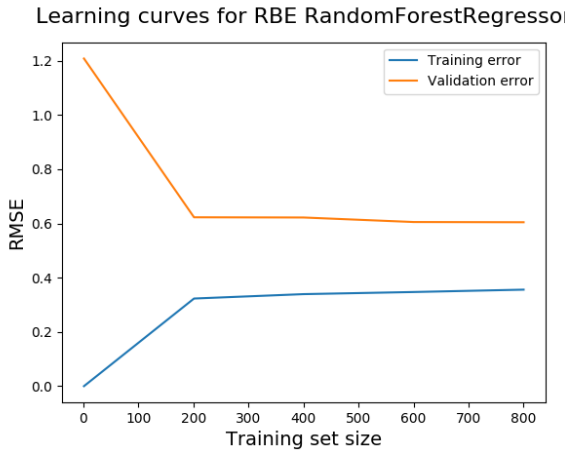


Figure 3.28: RBE

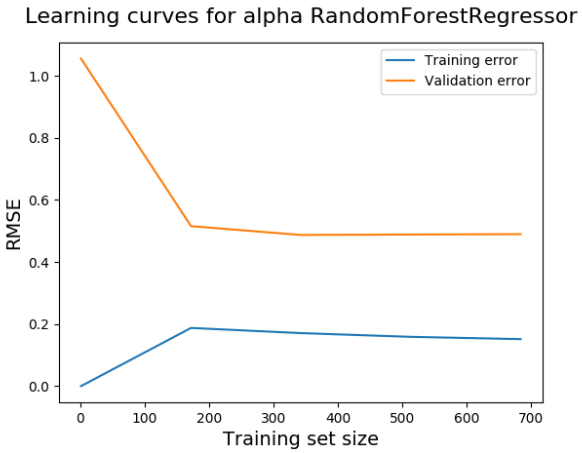


Figure 3.29: α

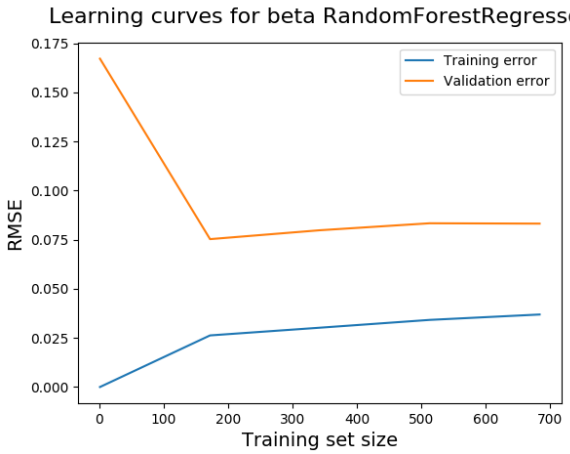


Figure 3.30: β

The RandomForest regressor which predicts the RBE, learning curves form a plateau at relatively low errors with a big gap which indicates a high variance. The plateau indicates that more data won't have a significant effect on our model(fig. 3.23). RF for the α coefficient demonstrates a similar picture, with the RBE model. We can see, that the variance is greater and more training samples would improve in a small way our model(fig. 3.24). RF for the β coefficient demonstrate a quite different picture. The model does not fair adequately. The gap increases with the train size, as is the case for the validation error. Given that β coefficient has the a distribution far from being characterized as normal, it's natural that we conclude that this is data related issue(fig. 3.25).

3.3.4 Gradient Boosting Decision Trees

Similarly, we will analyze the behavior of the GBDT models using learing curves.

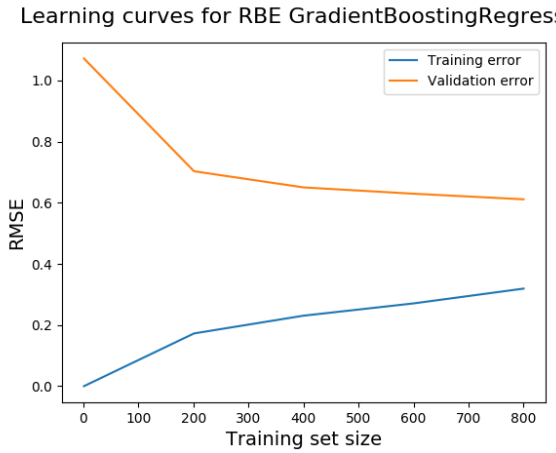


Figure 3.31: RBE

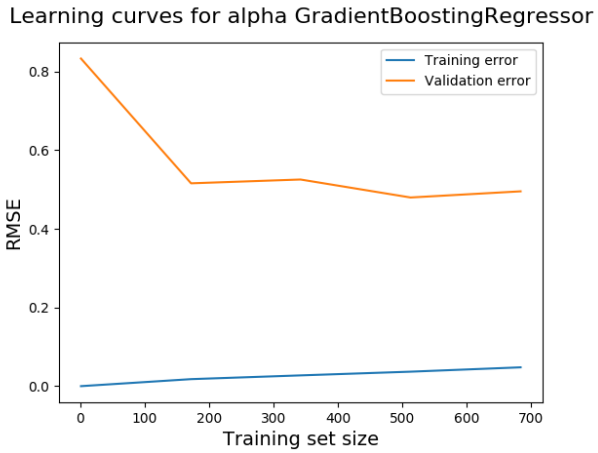


Figure 3.32: α

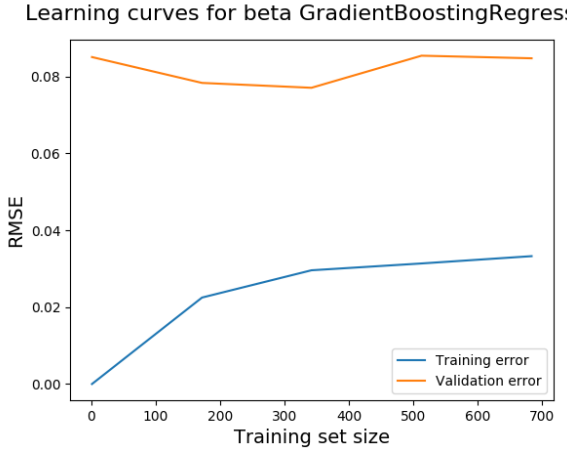


Figure 3.33: β

The Gradient Boosting regressor for the RBE, has also a good fit to the data indicated by the low training error. A bigger dataset seems to improve the model, as the gap would further decrease, as we can see from the decreasing gap (fig. 3.31). The Gradient Boosting regressor for the α coefficient has a similar picture with the RBE, albeit with a higher variance (fig. 3.32). The GBDT model for the β coefficient learning curve shows a different picture from the respective RandomForest model. The training error seems to increase more slowly, and the variance seems to decrease. Further data would decrease the variance of the model and increase its bias(fig. 3.33).

3.3.5 Support Vector Regression

The Support Vector Regression model for the RBE shows an altogether different behavior from the previous model. We will analyze this behavior using learning curves. All the model behave essentially in the same manner so, we will comment on them collectively. The SVR models demonstrate (fig. 3.34-36) higher training errors, which remains high in relation to the train size. This indicates a higher bias than the other models and overall, a more poor fit to the data. The variance in all the models is practically non existent, which

points to more simplistic models.

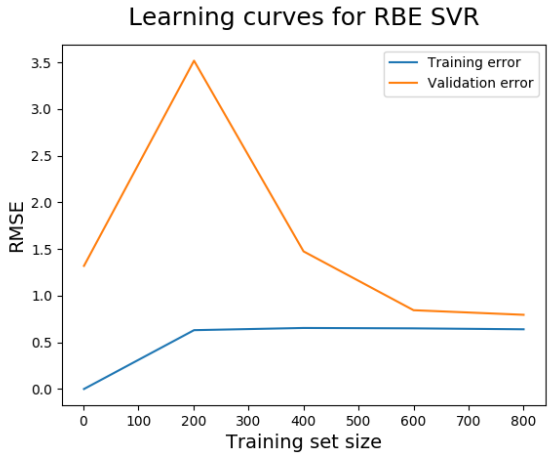


Figure 3.34: RBE

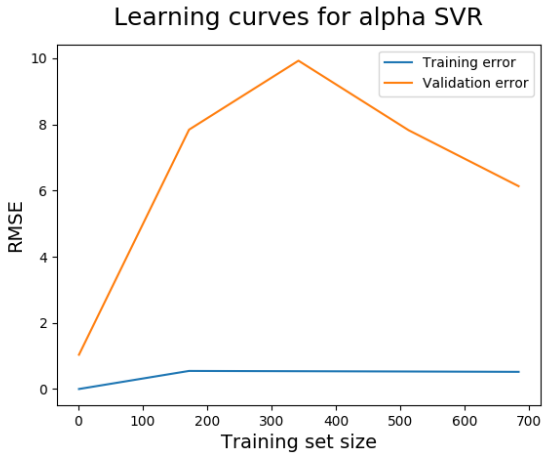


Figure 3.35: α

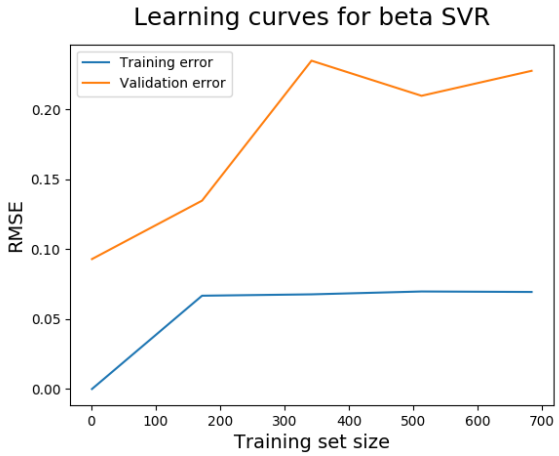


Figure 3.36: β

3.3.6 Voting Regression

Voting Regression, demonstrates a good compromise between the previous models, as is shown by the learning curves below(fig. 3.37-39). Overall the training errors are kept at low levels that indicate a good fit, while the variance is also very low but existent.

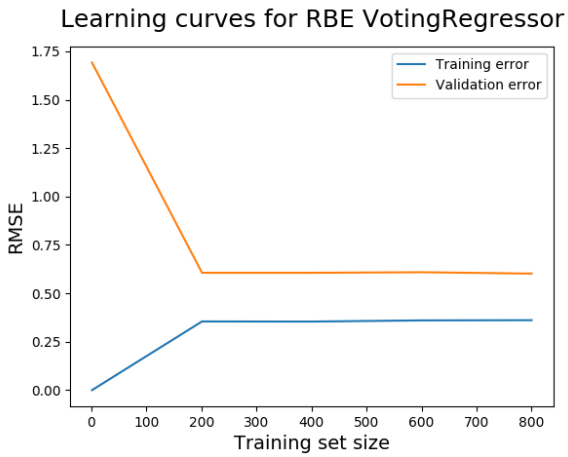


Figure 3.37: RBE

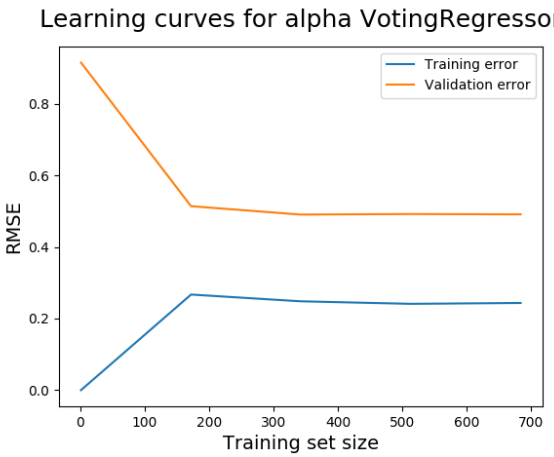


Figure 3.38: α

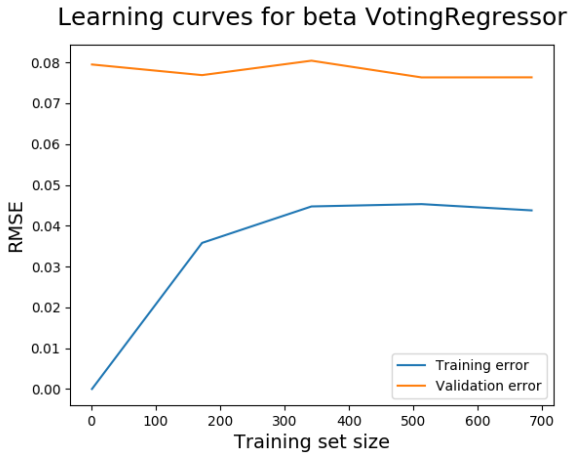


Figure 3.39: β

Voting Regression for the RBE shows an overall improvement in relation to the previous models. This is a low bias and low variance model, as both validation errors and training errors are relatively low and the gap between them is quite small. Further training samples don't seem to improve on the model. Voting Regression for the α coefficient shows a similar picture with the RBE model albeit with higher variance. The same applies for the β coefficient, even though, here we can see that more training samples would improve the model. That is, if the additional data do not contain in such a frequency zero β values.

3.4 Feature Analysis

In this section we will have a deeper look in our data, exploring how they are distributed, how important are to our models predictions and if there is any interaction among them. To save space, we will deal with only the *RBE* dataset as both datasets are quite similar.

3.4.1 Frequencies

Initially, we will take a look how our features are distributed in our dataset. As we can see(fig. 3.40) the majority of the irradiated cells are V79 cells from the lung of the chinese hamster.

- Cell

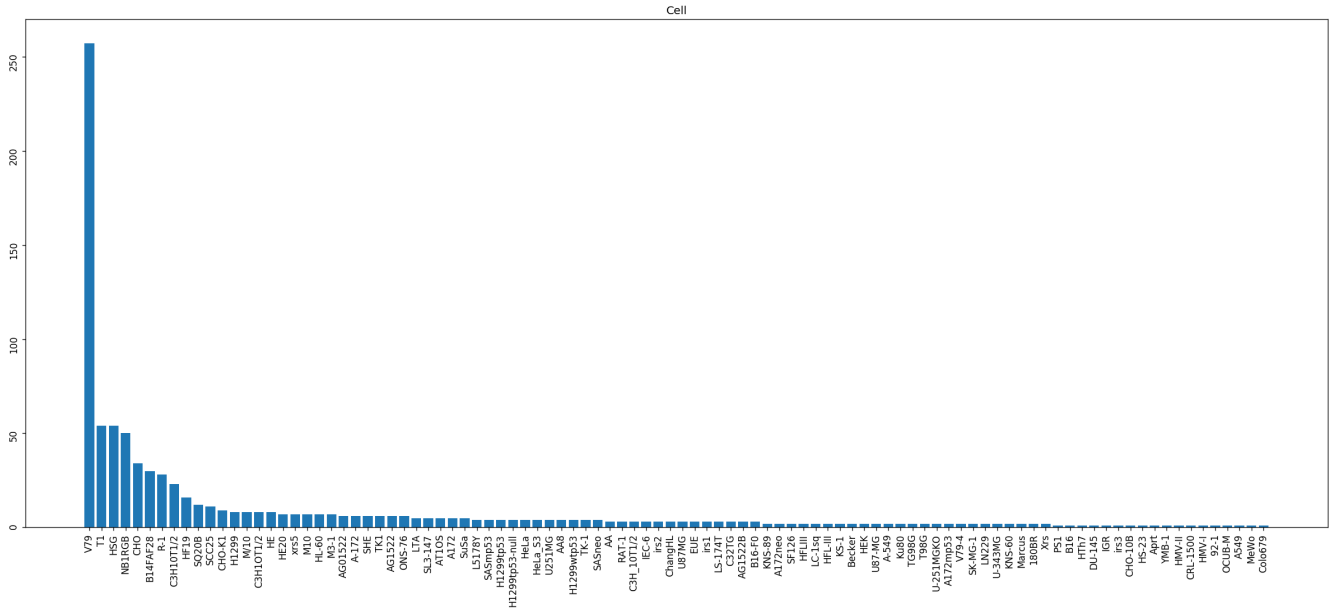


Figure 3.40: Cell line occurrence

- Type,Phase, Genomic Length

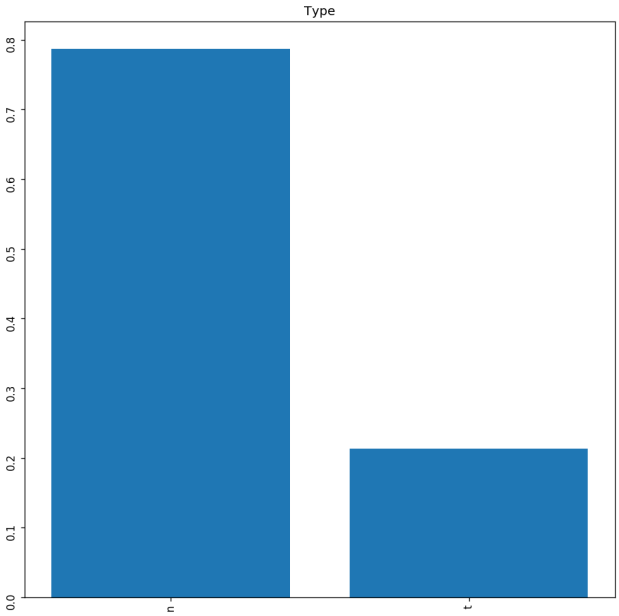


Figure 3.41: Tumor\Normal cell occurrence

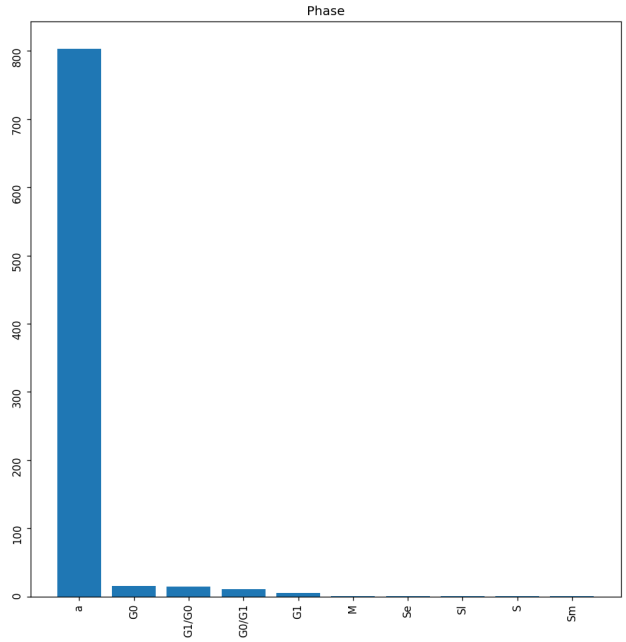


Figure 3.42: Phase occurrence

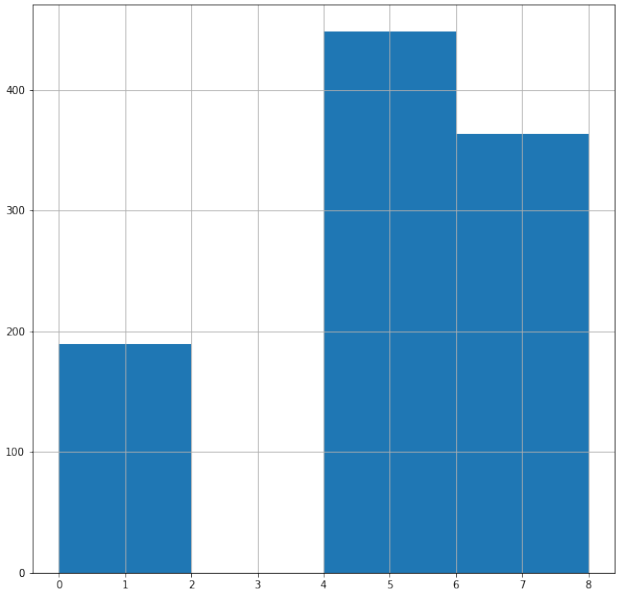


Figure 3.43: Genomic length occurrence

- Ion, Charge, Irmods

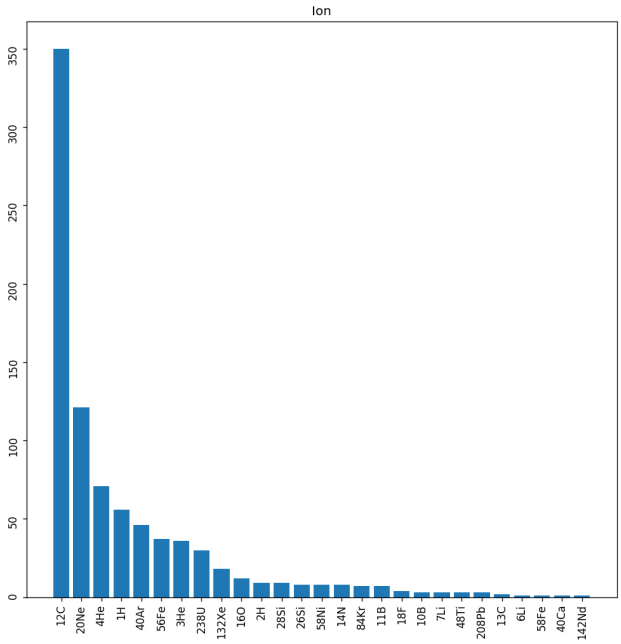


Figure 3.44: Ion occurence

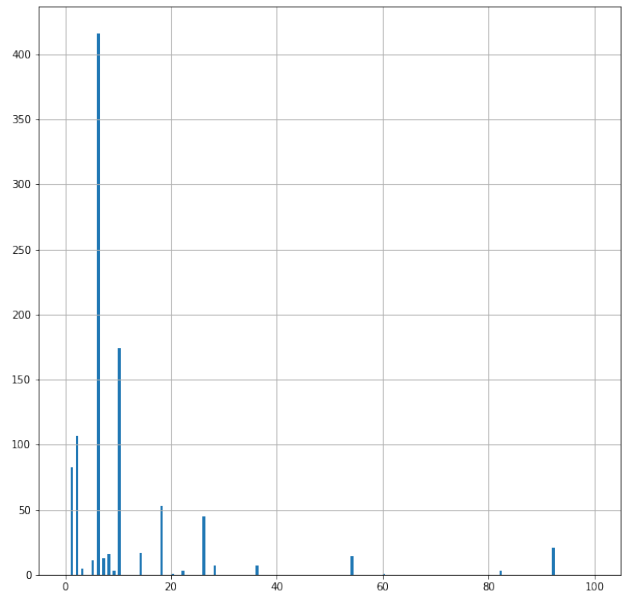


Figure 3.45: Charge occurrence

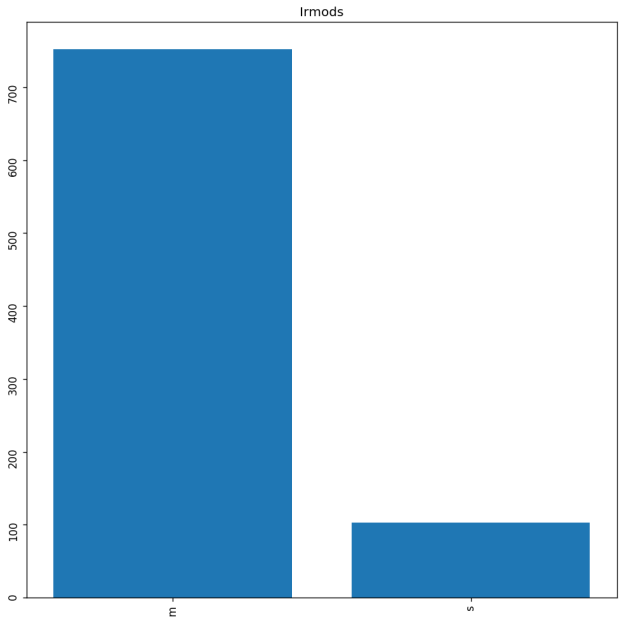


Figure 3.46: Irmods occurrence

- LET, E

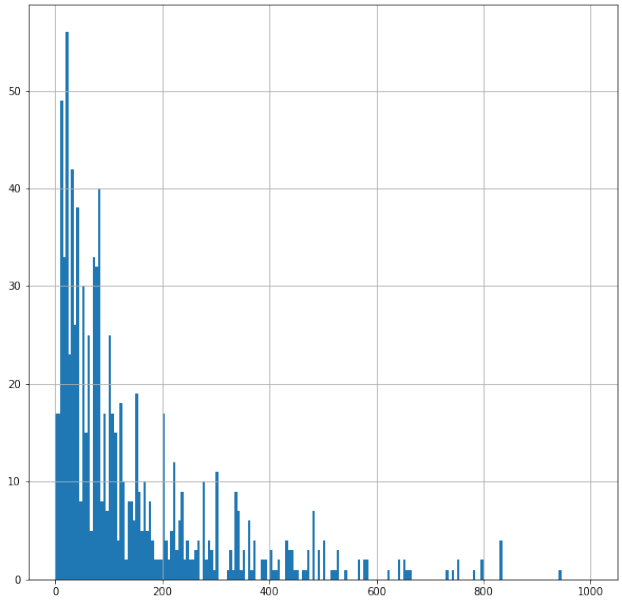


Figure 3.47: LET frequency

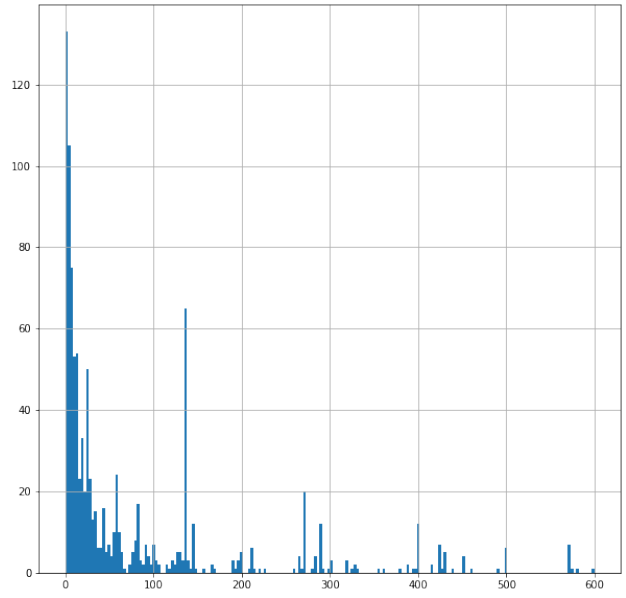


Figure 3.48: Specific Energy frequency

3.4.2 Importances

Using each algorithm, we can make an assessment of how big a contribution, every feature makes on the final prediction. This in Machine Learning lingo is called importance. Feature Importance, is calculated, by removing the feature and measuring the impact of its removal on the prediction. Here, we demonstrate, the importances of the features as calculated by the CatBoost, because all the algorithm show similar results. There are 3 plots(fig. 3.49-51), corresponding to the 3 CatBoost models for the RBE,α,β. All the models don't seem to agree on the distribution of the importances, which underlines the fact that the different predicted quantities arise from different biophysical mechanisms.

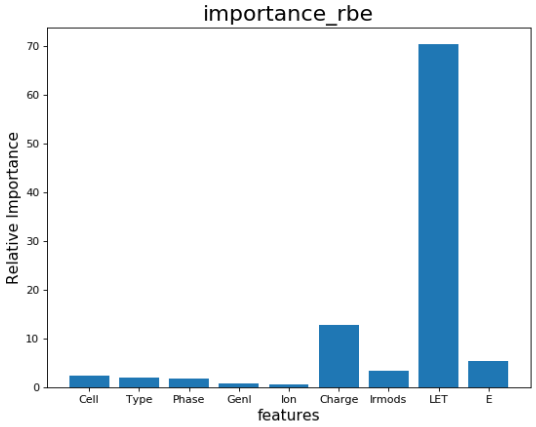


Figure 3.49: RBE importances

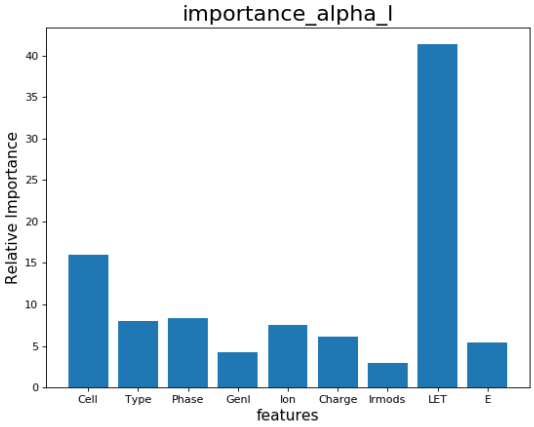


Figure 3.50: α model importances

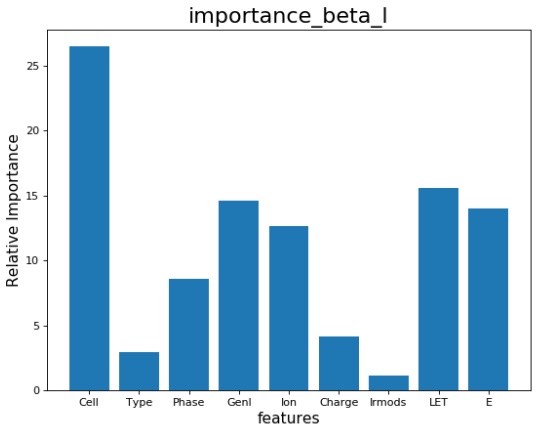


Figure 3.51: β model importances

In all the models fitted to predict the RBE value, it was evident that the value of LET is by far the most important one. The same applies for the α coefficient, we can see the strong LET importance to the model that predicts the α coefficient, as well the greater importance of the Cell line and Cell type features. The picture gets more interesting regarding the β coefficient. It is shown how strongly the Cell line affects the prediction of the β coefficient, which signifies a certain biophysical meaning or a data related problem, this is maybe due to the fact that in many of the irradiation experiments were conducted using radiosenitive cell lines. At any rate this requires a deeper investigation.

3.4.3 Interactions

An important interpretation metric for the models is the *feature interaction strength*. Interaction strength denotes how important to the prediction is the interaction of two features in our sample data. In a non linear model, the model cannot be reduced to the contribution of its individual features, because the effect one feature has on the prediction, depends on the value of another feature. To conceptualize the interaction notion in a Machine learning context, a model with only two features could be decomposed to four terms, two

terms for each feature, a constant term (bias), and a term for the interaction of the two features. The strength of the interaction of the two features is the difference in the prediction with and without this interaction term. In other words, it can be stated that the interaction strength is a measure of how important is the work of two features in tandem. Interaction strength is determined by measuring how much the variation of features depends on the interaction of the features. This metric is called H-statistic, introduced by Friedman and Popescu [35]. Interaction strength can give us a deeper insight into our data, uncovering correlation between features and cases were features contribute cooperatively to result. This augments the explanatory value of a model that appears as a complex, black box. Fortunately this is implement by the CatBoost library, which offers a way to show how strongly the feature interact. It whould be very useful to have a handy way for our SciKit model to calculate the features interactions to corroborate our results. In CatBoost it is implemented as :

$$feauture\_strength = \sum_{leafs} |\sum_{f_1=f_2} Leaf\_Value - \sum_{f_1!=f_2} Leaf\_Value|$$

The meaning of the above equation in the context of decision trees is the following: The value of the strength of the interaction of features f_1 and f_2 is the sum of absolute differences between the leaves of the tree containing the interaction $f_1 : f_2$ on the one hand, and on the other, the leaves not containing of features $f_1 : f_2$.

Feature interactions are shown in plots that compare the features in pairs and form a triangular checkerboard. We have three plots(fig. 3.52-54), one for each CatBoost model trained to predict the RBE,α,β values:

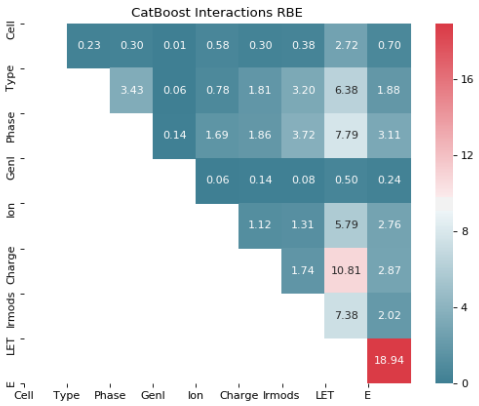


Figure 3.52: RBE model interactions

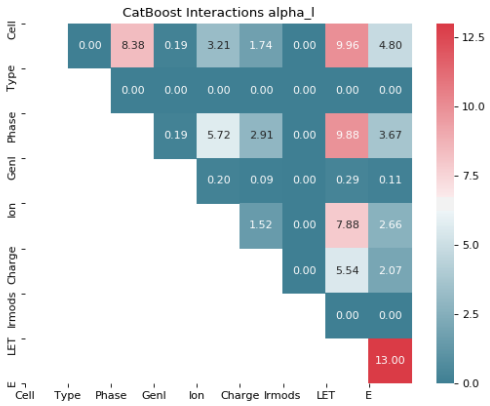


Figure 3.53: α model interactions

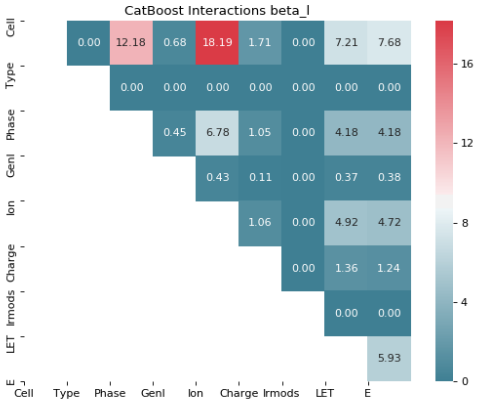


Figure 3.54: β model interactions

Firstly for the RBE CatBoost model, there are two visible findings. The one is that LET and specific energy interact very strongly and their interaction plays an important role in the RBE value. The next important thing is the interaction between LET and the charge of the particle. This may indicate that the charge as feature of the ion species utilized in the cell irradiation is the most important factor and not other attributes of the ion. As we can see LET interacts strongly with almost every feature, which is a further indicator of its importance. For the *alpha* model, there are some interesting facts to note. Similarly to the RBE model LET interacts strongly with almost all other features, with most important interaction being with the E value, additionally, LET interacts strongly with the Cell line and decreasingly with the cell phase and the ion species. There is also a strong interaction between Cell line and Cell Phase. These three factors indicate that LET by itself is not enough to account for the change in the α coefficient, and we should take into account not only how the features interact with LET to determine the value of α. Regarding the β coefficient, we should have in mind the low quality of the data samples that we have, which limits the value of our findings. That being said, there are also seemingly important things to state for the β CatBoost model interactions. Cell line interacts strongly with the ion species and the Cell phase. This is telling, of the biophysical meaning of the β coefficient in the quadratic model, which is believed to be proportional to

the probability of a double strand break event, and the appropriate cell line, phase and ion species may play a significant cooperative role to this happening.

Chapter 4

Discussion

The work done, was not the result of a premeditated plan in its entirety of technical details, rather, it is an ongoing work in progress, with a heavy trial and error element, in it. This doesn't mean that some significant results haven't come up, or that the goals were not clearly defined. The use of artificial intelligence in the radiotherapy domain is fairly young [36–39]. Similar techniques, have been used, but not on the more general question of the effects radiation has on the survival of cells, despite its obvious applications on cancer treatment. The main effort here, was to study various ML techniques to address this question and maybe come up with a more suitable approach on predicting the effects of radiation, especially of heavy ions. In this respect, many aspects of the problem were illustrated, and many possible routes to addressing it are becoming apparent.

Our main goal, was to shed light to the effects of radiation on various types of cells. It was desirable for our tool to be as generic as we could make it. There are two groups of input features in predicting irradiation effects. The first is relevant to the attributes of the radiation itself, which can be described by a seemingly endless number of features, and it is up to the researcher to decide the most crucial ones. The second group of features is relevant to the various attributes of the target cell that is irradiated, and again, it is important to choose from a multitude of attributes. The input features chosen here, were the most studied in the literature, which doesn't mean that each feature plays an equally important role in predicting the values that this study seeks to predict. This means, that a crucial step of feature selection should be added in the workflow of our program, to identify the most important features, and isolate the noise, non crucial features add to the data. This is the trade-off that was necessary to make given the scope of the study, i.e. the general use of the tools produced and the compilation of vast amounts of noisy data, that would be difficult to train on, and lead to overly big and complex models. That is why, at least at this stage, the work was preoccupied studying heavy ion irradiation.

4.1 Conclusions

"All models are wrong, but some are useful" - George Box

The magnitude and quality of our data does not allow for determining a clear cut winner regarding the models trained, in predicting the values assigned to them, although we have to mention from the start that the CatBoost Regression seems to fair better in all fields. An important feature that should be obvious after this study, is that each model has its propensities to be suitable for different problems. Gradient boosting decision trees and Random Forests are called ensemble methods, in the sense that they are comprised of many weak base estimators and iteratively, they build more complex models that can cope with non linear relations between dependent and independent variables. In the Machine Learning world, it has been accepted, that these methods show promise, and have over the last years been increasing in usage, in many aspects

of academic research and industry applications [40–43]. The notable difference between especially the two gradient boosting models can be attributed to characteristics of the CatBoost algorithm, namely its dealing with target leakage, making it suitable for smaller datasets and its handling of categorical variables with target statistics, that were discussed in the methods section. This, on the other hand, comes at the expense of memory,disk usage and training time, which CatBoost models seems to suffer the most.

Support Vector Regression seems to do better in predicting non normal distributions. This is especially true for predicting the β coefficient, which as we saw previously, its sampling is far from a gaussian distribution. This is maybe due to the fact that SVM performs the so called kernel trick, which transforms the data points, mapping them to a kernel function, which making the data points which may not be linearly separable, easier to classify.

Another noteworthy fact is that, the RBE value is much more successfully predicted as is shown in the test\train graphs. This is an interesting fact, because it hints to the importance of the quality of the data in Machine Learning. The majority of the literature that is related to cell survival studies uses the RBE, while α and β coefficients in the PIDE dataset were values extrapolated from the RBE studies. Furthermore, as we saw above, the RBE distribution of values in the dataset is much more natural, spanning across a wide margin and exhibiting a center peak, much like a normal distribution. This fact, points to a less complex model required to predict the RBE value, as opposed to the α and β quadratic model coefficients.

Another important conclusion is the fact that no major gains were maid with the implementation of the Voting Regression as an ensemble method. This illuminates the fact that model complexity does not by itself, ensures a model of better quality. The constituent models of Voting Regression, SciKit's Random Forests and Gradient Boosting Decision Trees, are already complex tree like complex models, and at this stage, we can only state, that the gains maid in combining them in the Voting ensemble results were marginal.

4.2 Usefulness

The usefulness of having a tool that can predict the effects of irradiation on cell survival, becomes more obvious in the domain of radiotherapy, where patients are treated with radiation to inflict cell death to tumorous cells, however this kind of treatment can be detrimental to healthy tissue. Two factors have to be considered by the medical personnel, i.e. the effect the proposed dosage will have on the tumor and the effect on the normal. A predictive tool that could before the actual irradiation, ascertain the effects on the patient, maximizing the damage on the tumor and minimizing the damage on the healthy tissue, would be very useful as a first step in determining these effects.

Similarly for medical reasons, it would be very useful to have a more general use prediction tool that could serve as a prevention tool, that could give hints to health hazards that various types of radiation would pose if we exposed an organism to it. This requires a broadening of the data sampled, to include types of radiation other that ions, like X-rays and UV light.

Another useful aspect of our work is the ability to have deeper insights into the biological and physical mechanisms involved in cell irradiation, through the model building process. This was not in the original scope of our study but emerged as a side effect of it. As it was mentioned above, there many interesting relations in our data that can be illuminated only from training a predictive model. Firstly, is the feature importances that indicate how important is to the prediction a particular feature, which should guide as to form relations of cause and effect in the phenomenon studied. An important finding was the β parameter of the quadratic model is strongly influenced by the cell line. The other finding which was expected, is the fact that the value of LET drives the prediction in the other tow cases, which again shows the causal correlation. Secondly, there the feature interactions. These are useful insight tools that tell us how strongly

two features work in a cooperative way to produce the studied value. Here we found that in respect to RBE value, LET and Specific Energy (E) strongly interact, as well as the charge of ions and LET. In respect to the α quadratic parameter, it was shown that, there are multiple strong interactions, between LET and E, between LET and Cell line, between LET and Cell phase and lastly between LET and ion species. In respect to the β quadratic parameter two significant interactions emerged, the first between Ion species and Cell line and between Cell phase and Cell line.

4.3 Data

Every model can get as good as the data it was trained on will allow it. There is low availability of the sort of data that we needed to implement our models. Data from particle irradiation that are curated and interpretable are surprisingly hard to find. The PIDE dataset was and still is the only resource of such data. This comes at big surprise, given the growing popularity of various types of ion Radiotherapy, over the last decades, and the growing number of particle accelerators.

There is also the issue of the uncertainty of the PIDE datasets which was not addressed by the creators of the database, which originates from the fact that the datasets are compilation of cell survival experiments which were conducted under different conditions and report their errors in different manners. Furthermore, the PIDE dataset is in a big part not a dataset depicting experimental data directly. In a big part, that was no stated in their paper many of the quantities in the dataset were calculated, converted and derived from other quantities. Typically, the experimentators gave either the LET or the E value, so the missing value was calculated with the 'ATIMA' program, however, outside the Bethe-Bloch region, uncertainties of the program increases. In respect to LQ parameters, there is the problem of the difference in how the studies in the dataset perform the fit of the LQ model to their survival data, namely some perform an error weighted fit while other an unweighted fit. Other authors do not give LQ parameters at all, in which case the parameters were extracted by the PIDE developers by fitting the LQ model to the given survival data study. All these points to the fact that the authors of the PIDE dataset urged not to use PIDE as a 'black box', which really should remind as that our work is experimental and should not be taken in face value.

4.4 Next Steps

An obvious next step is corroborating our research with more data. In respect to the consistency of the data, we should strive to find a coherent way in dealing with the inherent inconsistencies in the literature mentioned above. An additional next step is making our prediction tools more generic. What is needed is for our tool to be able to deal with more modes of radiation than ion beams. To achieve this, more data are needed, encompassing photon irradiation studies. The problem lies with the fact is that the data exists uncurrated buried inside the immense radiobiology literature. A first step is to use ML technique like data mining [44] (Natural Language Processing, Network Analysis, Clustering algorithms) to have access to more relevant information that normally gets annotated and curated manually as a last stage. As a last step we will deploy our models as a web application granting access and feedback from the public interested.

Chapter 5

Code

The program was developed in python 3.7.4. The required packages for building and running it are:

```
joblib==0.13.2
matplotlib==3.1.1
numpy==1.16.4
seaborn==0.9.0
pandas==0.24.2
catboost==0.15.2
shap==0.29.3
scikit_learn==0.21.2
```

The code is available in GitHub and is deployed as a web app at http://grainofsalt.appspot.com/

5.1 catboost_model.py

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from catboost import CatBoostRegressor, FeaturesData, Pool
from sklearn.model_selection import train_test_split
from matplotlib.ticker import PercentFormatter
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
import shap
import sys
import tools
# Import Datasets for quadratic model
data=pd.read_csv('./data/pide.csv',usecols = range(3,16))
data_alpha = data.drop(['alpha_x','beta_x','beta_l'],axis=1)
data_beta = data.drop(['alpha_x','beta_x','alpha_l'],axis=1)

# Import Datasets for RBE model
data_rbe=pd.read_csv('./data/data_rbe.csv')


#slit to train test datasets
splitted_rbe = tools.split(data_rbe,'RBE')
splitted_alpha = tools.split(data_alpha,'alpha_l')
splitted_beta = tools.split(data_beta,'beta_l')

# plot the frequencies of categorical variables
tools.categorical_frequencies(data)

# build catboost model
def learn(data,X,y,X_test,y_test,output):
    categorical_features_indices = np.where(X.dtypes == np.object)[0]
    data=Pool(X,y,cat_features=categorical_features_indices)
```

```
    model=CatBoostRegressor(iterations=3000, depth=10, learning_rate=0.01, l2_leaf_reg=100,
      loss_function='Quantile:alpha=0.5',eval_metric='RMSE',metric_period=30)
    model.fit(X, y,cat_features=categorical_features_indices,eval_set=(X_test, y_test),plot=True)
    modelL = CatBoostRegressor(iterations=3000, depth=10, learning_rate=0.01, l2_leaf_reg=100,
      loss_function='Quantile:alpha=0.05',metric_period=30)
    modelL.fit(X, y,cat_features=categorical_features_indices,eval_set=(X_test, y_test),plot=True)
    modelU = CatBoostRegressor(iterations=3000, depth=10, learning_rate=0.01, l2_leaf_reg=100,
      loss_function='Quantile:alpha=0.95',metric_period=30)
    modelU.fit(X, y,cat_features=categorical_features_indices,eval_set=(X_test, y_test),plot=True)
    return modelL, model, modelU

# Get predicted classes
modelL_rbe,model_rbe,modelU_rbe  = learn(data_rbe,splitted_rbe[0],splitted_rbe[1],splitted_rbe[2],splitted_rbe[3],['RBE'])
modelL_alpha,model_alpha,modelU_alpha = learn(data_alpha,splitted_alpha[0],splitted_alpha[1],splitted_alpha[2],splitted_alpha[3],['alpha_l'])
modelL_beta,model_beta,modelU_beta  = learn(data_beta,splitted_beta[0],splitted_beta[1],splitted_beta[2],splitted_beta[3],['beta_l'])


#Catboost Predictions
preds_rbe=model_rbe.predict(splitted_rbe[2])
preds_alpha=model_alpha.predict(splitted_alpha[2])
preds_beta=model_beta.predict(splitted_beta[2])

a = tools.scores(splitted_rbe[3],preds_rbe,'catboost_rbe')
b = tools.scores(splitted_alpha[3],preds_alpha,'catboost_alpha')
c = tools.scores(splitted_beta[3],preds_beta,'catboost_beta')
catboost_results = [a[0],a[1],b[0],b[1],c[0],c[1]]
tools.write_results(catboost_results,'Catboost')

# save catboost models
CatBoostRegressor.save_model(modelL_alpha,'./models/catboost_alpha_L.sav')
CatBoostRegressor.save_model(model_alpha,'./models/catboost_alpha.sav')
CatBoostRegressor.save_model(modelU_alpha,'./models/catboost_alpha_U.sav')
CatBoostRegressor.save_model(modelL_beta,'./models/catboost_beta_L.sav')
CatBoostRegressor.save_model(model_beta,'./models/catboost_beta.sav')
CatBoostRegressor.save_model(modelU_beta,'./models/catboost_beta_U.sav')
CatBoostRegressor.save_model(modelL_rbe,'./models/catboost_rbe_L.sav')
CatBoostRegressor.save_model(model_rbe,'./models/catboost_rbe.sav')
CatBoostRegressor.save_model(modelU_rbe,'./models/catboost_rbe_U.sav')

# importances
def importances(data, model,text):
    importance = model.get_feature_importance()
    fig, axs = plt.subplots(1, 1, figsize=(9, 9), sharey=True)
    names = list(data)
    axs.bar(names, importance)
    fig.suptitle('importance_'+text+'_plot')
    plt.savefig('./plots/importance_'+text+'.png')
    plt.clf()
    plt.close()

importances(data_rbe.drop(['RBE'],axis=1), model_rbe,'rbe')
importances(data_alpha.drop(['alpha_l'],axis=1), model_alpha,'alpha_l')
importances(data_beta.drop(['beta_l'],axis=1), model_beta,'beta_l')

# plot in 2d grid graph the interactions between the input variables based on the model
def interactions(model,data,output):
    X=data.drop(output,axis=1)
    interactions = model.get_feature_importance(type='Interaction',thread_count=8,verbose=True)
    firsts=[item[0] for item in interactions]
    seconds=[item[1] for item in interactions]
    m=int(max(max(firsts),max(seconds)))
    corr=pd.DataFrame(index=range(m+1),columns=range(m+1))
    for item in interactions:
        corr.at[int(item[0]),int(item[1])]=item[2]
    corr=corr.fillna(0)
    colrows=list(X)
    corr.columns=colrows
    corr.index=colrows
    plt.figure(num=None, figsize=(8, 6), dpi=80, facecolor='w', edgecolor='k')
    plt.title('CatBoost Interactions '+output[0])
    dropSelf = np.zeros_like(corr)
    dropSelf[np.tril_indices_from(dropSelf)] = True
    # Generate Color Map
    colormap = sns.diverging_palette(220, 10, as_cmap=True)
    # Generate Heat Map, allow annotations and place floats in map
    sns.heatmap(corr, cmap=colormap, annot=True, fmt=".2f", mask=dropSelf)
    # Apply xticks
```

```
    plt.xticks(range(len(corr.columns)), corr.columns);
    # Apply yticks
    plt.yticks(range(len(corr.columns)), corr.columns)
    # save plot
    plt.savefig('./plots/interactions_'+output[0]+'.png')
    plt.clf()
    plt.close()

interactions(model_rbe,data_rbe,['RBE'])
interactions(model_alpha,data_alpha,['alpha_l'])
interactions(model_beta,data_beta,['beta_l'])

# Shap values
def shap_function(model,data,X,y,output):
    # X = data.drop(output,axis=1)
    # y = data[output]
    shap.initjs()
    categorical_features_indices = np.where(X.dtypes == np.object)[0]
    explainer = shap.TreeExplainer(model)
    shap_values = explainer.shap_values(Pool(X, y, cat_features=categorical_features_indices))
    shap.force_plot(explainer.expected_value, shap_values[0,:], X.iloc[0,:])
    shap.summary_plot(shap_values, X, show=False)
    # shap.dependence_plot("LET", shap_values['LET'], X['LET'])
    plt.savefig('./plots/shap_values_'+output+'.png')
    plt.clf()

shap_function(model_rbe,data_rbe,splitted_rbe[0],splitted_rbe[1],'RBE')
shap_function(model_alpha,data_alpha,splitted_alpha[0],splitted_alpha[1],'alpha_l')
shap_function(model_beta,data_alpha,splitted_beta[0],splitted_beta[1],'beta_l')

Calculate statitistics for each model
model_rbe.calc_feature_statistics(splitted_rbe[0],
                                  splitted_rbe[1],
                                  plot=True,
                                  plot_file='./plots/statistics_RBE.html')
model_rbe.calc_feature_statistics(splitted_alpha[0],
                                  splitted_alpha[1],
                                  plot=True,
                                  plot_file='./plots/statistics_alpha.html')
model_rbe.calc_feature_statistics(splitted_rbe[0],
                                  splitted_rbe[1],
                                  plot=True,
                                  plot_file='./plots/statistics_beta.html')
```

5.2 parameters_grid.py

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.ensemble import VotingRegressor,RandomForestRegressor,GradientBoostingRegressor
from sklearn.preprocessing import OneHotEncoder
from sklearn.svm import SVR
import tools
from sklearn.model_selection import RandomizedSearchCV,GridSearchCV
import sys
from sklearn.model_selection import cross_val_score
# get the data from csv file
data=pd.read_csv('./data/pide.csv',usecols = range(3,16)).drop(['alpha_x','beta_x'],axis=1)
data_rbe = pd.read_csv('./data/data_rbe.csv')


def prepare_data(data,variables):
    data = pd.get_dummies(data,drop_first=True)
    x = data.drop(variables,axis=1)
    y = data[variables]
    return x,y

# prepare the data
rbe = prepare_data(data_rbe,'RBE')
alpha = prepare_data(data.drop(['beta_l'],axis=1),'alpha_l')
beta = prepare_data(data.drop(['alpha_l'],axis=1),'beta_l')

# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 100)]
```

```
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]# Create the random grid
learning_rate = [np.around(x, decimals=3) for x in np.linspace(0.001, 0.1, num = 9)]
max_depth_gbr = [int(x) for x in range(3,11)]

random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap
               }

# Training classifiers
# Use the random grid to search for best hyperparameters
# First create the base model to tune

rf = RandomForestRegressor()

# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = rf,
                               param_distributions = random_grid,
                               n_iter = 100,
                               cv = 3,
                               verbose=2,
                               random_state=42,
                               n_jobs = -1)
# Fit the random search model
rf_random.fit(rbe[0],rbe[1])
print(rf_random.best_params_)

gbr = GradientBoostingRegressor(random_state=1, n_estimators=10)



gbr_grid={'n_estimators':n_estimators,
            'learning_rate': learning_rate,
            'max_depth':max_depth_gbr,
            'min_samples_leaf':min_samples_leaf,
            # 'max_features':[1.0],
           }
gbr_random = RandomizedSearchCV(estimator = gbr,
           param_distributions = gbr_grid,
           n_iter = 10,
           cv = 3,
           verbose=2,
           random_state=42,
           n_jobs = -1)
gbr_random.fit(rbe[0],rbe[1])
print(gbr_random.best_params_)

parameters = {'C':[1, 10, 100, 1000,10000],'epsilon':[0.1,0.2,0.5,0.3,0.05]}
svr = SVR(gamma=0.0001, kernel='rbf', C=1,epsilon=0.1)
clf =GridSearchCV(svr, parameters,cv=5,n_jobs=7,verbose=10)
clf.fit(rbe[0],rbe[1])
print(clf.best_params_)


def evalulate_model(model,parameter_dict,X,y,dataset_name):
    the_scores = np.sqrt(- cross_val_score(model, X, y, cv=10, n_jobs=7,scoring='neg_mean_squared_error'))
    print("RMSE: %0.2f (+/- %0.2f)" % (the_scores.mean(), the_scores.std() * 2))
    length = len(parameter_dict)
    fig ,axs = plt.subplots(1,length)
    locs, labels =plt.yticks()
    fig.suptitle(type(model).__name__ +' Parameter evalutation '+ dataset_name )
    # Do the plotting
```

```
    for key,ax in zip(parameter_dict,axs):
        scores = list()
        scores_std = list()
        for item in  parameter_dict[key]:
            param={key:item}
            model.set_params(**param)
            this_scores = np.sqrt(- cross_val_score(model, X, y, cv=10, n_jobs=7,scoring='neg_mean_squared_error'))
            scores.append(np.mean(this_scores))
            scores_std.append(np.std(this_scores))
        # ax.title.set_text(str(key) + ' evalutation')
        ax.semilogx(parameter_dict[key], scores)
        ax.semilogx(parameter_dict[key], np.array(scores) + np.array(scores_std), 'b--')
        ax.semilogx(parameter_dict[key], np.array(scores) - np.array(scores_std), 'b--')
        ax.set_yticks(locs, list(map(lambda x: "%g" % x, locs)))
        ax.set_ylabel('CV score')
        ax.set_xlabel(str(key))
        ymin = np.min(scores)
        xmin = scores[np.argmin(ymin)]
        text= "Minimum for "+key+"={:.4f}, RMSE={:.4f}".format(xmin, ymin)
        bbox_props = dict(boxstyle="square,pad=0.3", fc="w", ec="k", lw=0.72)
        kw = dict(xycoords='data',textcoords="axes fraction", bbox=bbox_props, ha="right", va="top")
        ax.annotate(text, xy=(xmin, ymin), xytext=(0.94,0.96), **kw)
    plt.show()
svr_parameters = {'C':np.logspace(-10,2,10),'gamma':np.logspace(-10,2,10),'epsilon':np.linspace(start=0,stop=1,num=10)}
evalulate_model(svr,svr_parameters,beta[0],beta[1],'beta')
evalulate_model(gbr,gbr_grid,rbe[0],rbe[1])
evalulate_model(rf,random_grid,rbe[0],rbe[1])
```

5.3 support_vector_regression.py

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
from sklearn.svm import SVR
import matplotlib.pyplot as plt
import joblib
import tools
import sys

# One Hot Encode the categorical data and dump the encoders to disk
def dump_encoder(data_df,name):
    enc = OneHotEncoder(handle_unknown='error')
    categorical = data_df.select_dtypes(include=[object])
    numerical = data_df.drop(categorical,axis=1)
    enc.fit(categorical)
    categorical_enc = enc.transform(categorical)
    categorical_enc_df = pd.DataFrame(categorical_enc.toarray())
    # Export the One Hot Encoder
    if name == 'quadratic' :
        joblib.dump(enc, './models/OneHotEncoder_quadratic.pkl')
        return None
    elif name == 'RBE':
        joblib.dump(enc, './models/OneHotEncoder_rbe.pkl')
        return None
    else:
        return None

def svm_regression(a,name,C,e):
    svr = SVR(gamma='auto',C=C,epsilon=e)
    model = svr.fit(a[0],a[1])
    joblib.dump(model, './models/'+name+'_model.sav')
    preds = model.predict(a[2])
    return tools.scores(a[3],preds,name)

# get the data from csv file
data=pd.read_csv('./data/pide.csv',usecols = range(3,16)).drop(['alpha_x','beta_x'],axis=1)
data_rbe = pd.read_csv('./data/data_rbe.csv')

dump_encoder(data_rbe,'RBE')
dump_encoder(data,'quadratic')

rbe = tools.preprocess_data(data_rbe,['RBE'])
```

```
quadratic_alpha = tools.preprocess_data(data.drop(['beta_l'],axis=1),['alpha_l'])
quadratic_beta = tools.preprocess_data(data.drop(['alpha_l'],axis=1),['beta_l'])

svm_rbe_results=svm_regression(rbe,'svr_rbe',1,0.5)
svm_alpha_results=svm_regression(quadratic_alpha,'svr_alpha',1,0.5)
svm_beta_results=svm_regression(quadratic_beta,'svr_beta',0.1,0.01)
# # 100000
svr_results = [svm_rbe_results[0],
               svm_rbe_results[1],
               svm_alpha_results[0],
               svm_alpha_results[1],
               svm_beta_results[0],
               svm_beta_results[1]]
tools.write_results(svr_results,'SVR')
```

5.4 models_interpretation.py

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.svm import SVR
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import VotingRegressor
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import train_test_split
import tools
import joblib
from catboost import CatBoostRegressor

def prepare_data(data,variables):
    data = pd.get_dummies(data,drop_first=True)
    x = data.drop(variables,axis=1)
    y = data[variables]
    return x,y

#  Split in train and test subset for validation 2nd argument :
def preprocess_data(data,variables):
    data = pd.get_dummies(data,drop_first=True)
    train,test=train_test_split(data ,train_size=0.9,test_size=0.1, random_state=12)

    # split dataset to input and output in train dataset
    x = train.drop(variables,axis=1).values
    y = train[variables].values.ravel()

    # split dataset to input and output in test dataset
    x_test =test.drop(variables,axis=1).values
    y_test = test[variables].values.ravel()

    return x,y,x_test,y_test,

# get the data from csv file
data=pd.read_csv('./data/pide.csv',usecols = range(3,16)).drop(['alpha_x','beta_x'],axis=1)
data_rbe = pd.read_csv('./data/data_rbe.csv')

# prepare the data
rbe = prepare_data(data_rbe,'RBE')
alpha = prepare_data(data.drop(['beta_l'],axis=1),'alpha_l')
beta = prepare_data(data.drop(['alpha_l'],axis=1),'beta_l')


# # declare the RBE regressors
rf_rbe = RandomForestRegressor(n_estimators= 1600, min_samples_split= 5, min_samples_leaf = 1
  ,max_features = 'auto', max_depth = 90, bootstrap = True)
svr_rbe = SVR(gamma='auto',C=1,epsilon=0.5)
gbr_rbe = GradientBoostingRegressor(random_state=1, n_estimators=600, min_samples_leaf = 2
  , max_features = 1.0,max_depth = 4,learning_rate = 0.075)
vr_rbe = VotingRegressor(estimators=[('gb', gbr_rbe), ('rf', rf_rbe),('svr',svr_rbe)])
#
#
# # declare the alpha regressors
rf_alpha = RandomForestRegressor(n_estimators = 1600, min_samples_split= 5, min_samples_leaf = 1,
  max_features = 'auto', max_depth = 90, bootstrap =True)
svr_alpha = SVR(gamma='auto',C=1,epsilon=0.5)
gbr_alpha = GradientBoostingRegressor(n_estimators= 1800, min_samples_leaf = 1, max_features = 1.0,
```

```
  max_depth = 4, learning_rate = 0.063)
vr_alpha = VotingRegressor(estimators=[('gb', gbr_alpha), ('rf', rf_alpha),('svr',svr_alpha)])

# declare the beta regressors
rf_beta = RandomForestRegressor(n_estimators=1600, min_samples_split=5, min_samples_leaf=1,
  max_features='auto', max_depth=90, bootstrap=True)
svr_beta = SVR(gamma='auto',C=0.1,epsilon=0.01)
gbr_beta = GradientBoostingRegressor(n_estimators=200, min_samples_leaf=1, max_features=1.0,
  max_depth = 6, learning_rate=0.013)
vr_beta = VotingRegressor(estimators=[('gb', gbr_beta), ('rf', rf_beta),('svr',svr_beta)])

# fit the rbe regressors
rf_rbe = rf_rbe.fit(rbe[0], rbe[1])
svr_rbe = svr_rbe.fit(rbe[0], rbe[1])
gbr_rbe = gbr_rbe.fit(rbe[0], rbe[1])
vr_rbe =vr_rbe.fit(rbe[0], rbe[1])

# fit the alpha regressors
rf_alpha = rf_alpha.fit(alpha[0],alpha[1])
svr_alpha = svr_alpha.fit(alpha[0],alpha[1])
gbr_alpha = gbr_alpha.fit(alpha[0],alpha[1])
vr_alpha = vr_alpha.fit(alpha[0],alpha[1])

# fit the beta regressors
rf_beta = rf_beta.fit(beta[0],beta[1])
svr_beta = svr_beta.fit(beta[0],beta[1])
gbr_beta = gbr_beta.fit(beta[0],beta[1])
vr_beta = vr_beta.fit(beta[0],beta[1])

# plot feature importances
model_name='Random_Forest'
tools.plot_importances(rf_rbe,rbe[0],data_rbe,'RBE',model_name)
tools.plot_importances(rf_alpha,alpha[0],data.drop(['beta_l'],axis=1),'alpha_l',model_name)
tools.plot_importances(rf_beta,beta[0],data.drop(['alpha_l'],axis=1),'beta_l',model_name)

model_name='Gradient_Boosting_Regression'
tools.plot_importances(gbr_rbe,rbe[0],data_rbe,'RBE',model_name)
tools.plot_importances(gbr_alpha,alpha[0],data.drop(['beta_l'],axis=1),'alpha_l',model_name)
tools.plot_importances(gbr_beta,beta[0],data.drop(['alpha_l'],axis=1),'beta_l',model_name)

# Plot Learning Curves
train_sizes_rbe = [int(round(x)) for x in np.linspace(1,800,5)]
train_sizes_quadratic = [int(round(x)) for x  in np.linspace(1,684,5)]

# Random Forest Regressors
tools.plot_learning_curve(rf_rbe,data_rbe,rbe[0],rbe[1],train_sizes_rbe,5,'RBE')
tools.plot_learning_curve(rf_beta,data,beta[0],beta[1],train_sizes_quadratic,5,'beta')
tools.plot_learning_curve(rf_alpha,data,alpha[0],alpha[1],train_sizes_quadratic,5,'alpha')

# Support Vector Regressors
tools.plot_learning_curve(svr_rbe,data_rbe,rbe[0],rbe[1],train_sizes_rbe,5,'RBE')
tools.plot_learning_curve(svr_alpha,data,alpha[0],alpha[1],train_sizes_quadratic,5,'alpha')
tools.plot_learning_curve(svr_beta,data,beta[0],beta[1],train_sizes_quadratic,5,'beta')

# Gradient Boosting Regressors
tools.plot_learning_curve(gbr_rbe,data_rbe,rbe[0],rbe[1],train_sizes_rbe,5,'RBE')
tools.plot_learning_curve(gbr_alpha,data,alpha[0],alpha[1],train_sizes_quadratic,5,'alpha')
tools.plot_learning_curve(gbr_beta,data,beta[0],beta[1],train_sizes_quadratic,5,'beta')

# Voting Regressors
tools.plot_learning_curve(vr_rbe,data_rbe,rbe[0],rbe[1],train_sizes_rbe,5,'RBE')
tools.plot_learning_curve(vr_alpha,data,alpha[0],alpha[1],train_sizes_quadratic,5,'alpha')
tools.plot_learning_curve(vr_beta,data,beta[0],beta[1],train_sizes_quadratic,5,'beta')


from catboost import Pool, cv

def catboost_eval(X,y,metric):
    cv_data = X
    labels = y
    cat_features =np.where(X.dtypes == np.object)[0]
    cv_dataset = Pool(data=cv_data,
                      label=labels,
                      cat_features=cat_features)
    params = {"iterations": 1000,
              "depth": 8,
              'learning_rate':0.01,
```

```
            "loss_function": "Quantile:alpha=0.5",
            "verbose": True,
            'target_border':0.5,
            'metric_period':30,
            'eval_metric':metric}
    scores = cv(cv_dataset,
              params,
              as_pandas=True,
              fold_count=10)
    return scores

rbe_r2 = catboost_eval(data_rbe.drop(['RBE'],axis=1),data_rbe['RBE'],'R2')['test-R2-mean'].max()
alpha_r2 = catboost_eval(data.drop(['alpha_l'],axis=1),data['alpha_l'],'R2')['test-R2-mean'].max()
beta_r2 = catboost_eval(data.drop(['beta_l'],axis=1),data['beta_l'],'R2')['test-R2-mean'].max()
print(rbe_r2,alpha_r2,beta_r2)

rbe_ci = catboost_eval(data_rbe.drop(['RBE'],axis=1),data_rbe['RBE'],'MAE')['test-MAE-std'].min()
alpha_ci = catboost_eval(data.drop(['alpha_l'],axis=1),data['alpha_l'],'MAE')['test-MAE-std'].min()
beta_ci = catboost_eval(data.drop(['beta_l'],axis=1),data['beta_l'],'MAE')['test-MAE-std'].min()
print(rbe_ci,alpha_ci,beta_ci)

enc_data = pd.get_dummies(data,drop_first=True)
enc_data_rbe = pd.get_dummies(data_rbe,drop_first=True)

from scipy import stats
print(stats.shapiro(enc_data_rbe.values))
print(stats.shapiro(alpha[1].values))
print(stats.shapiro(beta[1].values))
```

5.5 tools.py

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn.model_selection import learning_curve
from sklearn.model_selection import cross_val_score
import os
import joblib
import re
# plot frequencies
def categorical_frequencies(datastruct):
    categorical = np.where(datastruct.dtypes == np.object)[0]
    fig,axes=plt.subplots(2,3,figsize=(15,15),dpi=120, facecolor='w', edgecolor='k')
    fig.suptitle('Categorical Features Frequency')
    fig.canvas.set_window_title('Categorical Features Frequency')
    i=1
    for ax , index in zip(axes.flat, categorical):
        column=datastruct.iloc[:,index].value_counts(normalize=True)
        ax.bar(column.index,height=column.values)
        # ax = column.plot(kind='bar',color='blue')
        ax.set_title(datastruct.columns.values[index])
        ax.tick_params(rotation=90)
        i=i+1
    # fig.delaxes(axes[1][2])
    plt.savefig('./plots/frequencies.png')
    plt.clf()

def split(data,output):
    train,test=train_test_split(data ,train_size=0.9,test_size=0.1, random_state=12)
    X = train.drop(output, axis=1)
    y = train[output]
    X_test = test.drop(output,axis=1)
    y_test = test[output]
    return [X,y,X_test,y_test]

def scores(test,preds,name):
    mse = mean_squared_error(test,preds)
    print(f'MSE ({name}): {mse}')
    r2 = r2_score(test,preds)
    print(f'R^2 ({name}): {r2}')
    plot_accuracy(test,preds,name)
    return mse,r2
```

```
# prepare data with no splitting
def prepare_data(data,variables):
    data = pd.get_dummies(data,drop_first=True)
    x = data.drop(variables,axis=1)
    y = data[variables]
    return x,y

def cv_scores(X,y,model):
    cv_scores = -cross_val_score(model, X, y, cv=10, n_jobs=7,scoring='neg_mean_absolute_error')
    r2_scores = cross_val_score(model, X, y, cv=10, n_jobs=7,scoring='r2')
    return cv_scores.mean(),cv_scores.std(),r2_scores.mean()

# self explanatory
def plot_accuracy(actual,preds,name):
    df = pd.DataFrame({'Actual':actual,'Predicted':preds})
    df1 = df.head(25)
    df1.plot(kind='bar',figsize=(10,8))
    # plt.title('Actual vs Prediction for '+name)
    plt.grid(which='major', linestyle='-', linewidth='0.5', color='green')
    plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
    plt.savefig('./plots/'+name+'.png')
    plt.close()

def write_results(input,model):
    input.insert(0,model)
    headers = ['model','RBE_mse','RBE_r^2','alpha_mse','alpha_r^2','beta_mse','beta_r^2']
    df = pd.DataFrame([input], columns = headers)
    if not os.path.isfile('./results.csv'):
        df.to_csv('./results.csv',index=False)
    else :
        current = pd.read_csv('./results.csv')
        updated = pd.concat([current,df])
        print(updated)
        updated.to_csv('./results.csv',index=False)
    return None

def write_confidence(input,model):
    rounded_input = ['%.2f' % elem for elem in input]
    rounded_input.insert(0,model)
    headers = ['model','RBE_mae','RBE_CI +/-','RBE_r^2','alpha_mae','alpha_CI +/-','alpha_r^2','beta_mae','beta_CI +/-','beta_r^2']
    df = pd.DataFrame([rounded_input], columns = headers)
    if not os.path.isfile('./confidence_intervals.csv'):
        df.to_csv('./confidence_intervals.csv',index=False)
    else :
        current = pd.read_csv('./confidence_intervals.csv')
        updated = pd.concat([current,df])
        print(updated)
        updated.to_csv('./confidence_intervals.csv',index=False)
    return None

# Perform the encoding of the dataset
def encode(data_df,name):
    if name == 'RBE':
        enc = joblib.load('./models/OneHotEncoder_rbe.pkl')
    else:
        enc = joblib.load('./models/OneHotEncoder_quadratic.pkl')
    categorical = data_df.select_dtypes(include=[object])
    numerical = data_df.drop(categorical,axis=1)
    categorical_enc = enc.transform(categorical).toarray()
    categorical_enc_df = pd.DataFrame(categorical_enc)
    return pd.concat([numerical,categorical_enc_df],axis=1, join='inner')

#  Split in train and test subset for validation 2nd argument :
def preprocess_data(data,variables):
    train,test=train_test_split(encode(data,variables[0]) ,train_size=0.9,test_size=0.1, random_state=12)

    # split dataset to input and output in train dataset
    x = train.drop(variables,axis=1).values
    y = train[variables].values.ravel()

    # split dataset to input and output in test dataset
    x_test =test.drop(variables,axis=1).values
    y_test = test[variables].values.ravel()

    return x,y,x_test,y_test,train
```

```
def plot_learning_curve(estimator, data, features, target, train_sizes, cv,target_name):
    train_sizes, train_scores, validation_scores = learning_curve(
    estimator, features, target, train_sizes =
    train_sizes,
    cv = cv, scoring = 'neg_mean_squared_error',shuffle = True)
    train_scores_mean = np.sqrt(-train_scores.mean(axis = 1))
    validation_scores_mean = np.sqrt(-validation_scores.mean(axis = 1))

    plt.plot(train_sizes, train_scores_mean, label = 'Training error')
    plt.plot(train_sizes, validation_scores_mean, label = 'Validation error')

    plt.ylabel('RMSE', fontsize = 14)
    plt.xlabel('Training set size', fontsize = 14)
    title = 'Learning curves for ' +target_name+' '+ str(estimator).split('(')[0]
    plt.title(title, fontsize = 16, y = 1.03)
    plt.legend()
    plt.savefig('./plots/Learning_Curve_'+str(estimator).split('(')[0]+'_'+target_name)
    plt.close()

def plot_importances(model,X,data,dropped,model_name):
    importances = model.feature_importances_
    indices = np.argsort(importances)
    features = X.columns.values
    columns = data.drop([dropped],axis=1).columns.values
    dict ={}
    for column in columns:
        dict[column] = 0
    for column in features:
        index = np.where(features == column)[0][0]
        x = re.split("_",column)
        if x[0]  in columns:
            dict[x[0]] += importances[index]
    features = []
    importances = []
    for key,value in dict.items():
        features.append(key)
        importances.append(value)
    # plt.title('Feature_Importances_'+model_name+'_'+dropped)
    plt.bar(features,importances)
    plt.xlabel('Relative Importance')
    plt.savefig('./plots/'+model_name+'_importances_'+dropped)
    plt.close()
```

5.6 main.py

```
import joblib
import pandas as pd
import numpy as np
from sklearn.neighbors import NearestNeighbors
from catboost import CatBoostRegressor
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import sys
import tools

# Get all models as a tuple
def load_models():

    # svr models
    svr_rbe_model = joblib.load('./models/svr_rbe_model.sav')
    svr_alpha_model = joblib.load('./models/svr_alpha_model.sav')
    svr_beta_model = joblib.load('./models/svr_beta_model.sav')
    srv_models = (svr_rbe_model,svr_alpha_model,svr_beta_model)

    # gbr models
    gbr_rbe_model = joblib.load('./models/gbr_rbe_model.sav')
    gbr_alpha_model = joblib.load('./models/gbr_alpha_model.sav')
    gbr_beta_model = joblib.load('./models/gbr_beta_model.sav')
    gbr_models = (gbr_rbe_model,gbr_alpha_model,gbr_beta_model)

    # rf models
    rf_rbe_model = joblib.load('./models/rf_rbe_model.sav')
    rf_alpha_model = joblib.load('./models/rf_alpha_model.sav')
    rf_beta_model = joblib.load('./models/rf_beta_model.sav')
```

```
    rf_models = (rf_rbe_model,rf_alpha_model,rf_beta_model)

    # vr models
    vr_rbe_model=joblib.load('./models/vr_rbe_model.sav')
    vr_alpha_model=joblib.load('./models/vr_alpha_model.sav')
    vr_beta_model=joblib.load('./models/vr_beta_model.sav')
    vr_models = (vr_rbe_model,vr_alpha_model,vr_beta_model)
    return srv_models,gbr_models,rf_models,vr_models

# impute missing data
def impute(lame_input,model):
    if model == 'rbe':
        dataset = pd.read_csv('./data/data_rbe.csv')
        x = dataset.drop('RBE',axis=1)
    else:
        dataset = pd.read_csv('./data/pide.csv',usecols = range(3,16))
        x = dataset.drop(['alpha_l','alpha_x','beta_l','beta_x'],axis=1)

    headers = x.columns.values
    categorical = dataset.select_dtypes(include=[object])
    catlist = [x.columns.get_loc(c) for c in categorical.columns if c in categorical]
    for index in range(len(lame_input)):
        if  not np.isin(lame_input[index], dataset.iloc[:,index].values):
            if index in catlist:
                lame_input[index] = np.nan
            else:
                lame_input[index]= -1
    a = [z for z,y in enumerate(lame_input) if y == -1 or str(y)=='nan']
    input_df = pd.DataFrame(np.array([lame_input]),columns=headers)
    if len(a)<1:
        return lame_input
    else:
        output = lame_input
        # catlist = [x.columns.get_loc(c) for c in categorical.columns if c in categorical]
        for column in input_df:
            if column not in list(categorical):
                input_df[column] = pd.to_numeric(input_df[column],errors='coerce')

        incomplete = x.append(input_df,ignore_index=True)
        incomplete_dm = pd.get_dummies(incomplete, prefix = catlist, drop_first=True)
        neigh = NearestNeighbors(n_neighbors=2)
        neigh.fit(incomplete_dm.values)
        j=0
        while j < len(lame_input):
            if str(lame_input[j]) == 'nan':
                added = incomplete_dm.loc[incomplete_dm[str(j)+'_nan'] == 1].index.values[0]
            if lame_input[j] == -1:
                col = list(incomplete.columns)[j]
                added = incomplete[incomplete[col]== -1.0].index.values[0]
            if str(lame_input[j]) == 'nan' or lame_input[j] == -1:
                index = neigh.kneighbors([incomplete_dm.iloc[added]], return_distance=False)[0][1]
                values = incomplete.iloc[index].values
                for i, (l,v) in enumerate(zip(lame_input,values)):
                    if i not in a:
                        output[i]=l
                    else:
                        output[i]=v
                break
            j += 1
        return output

def encode_input(data_array,model):
    # get the data from csv file
    data=pd.read_csv('./data/pide.csv',usecols = range(3,16))
    headers = list(data.iloc[:,0:9])

    # Make a DataFrame from the input
    input_df = pd.DataFrame(np.array([data_array]),columns=headers)
    categorical = data.select_dtypes(include=[object])

    # assign correct data type to input variables
    for column in input_df:
        if column not in list(categorical):
            input_df[column] = pd.to_numeric(input_df[column])
    if model == 'rbe':
        enc = joblib.load('./models/OneHotEncoder_rbe.pkl')
    else:
```

```
        enc = joblib.load('./models/OneHotEncoder_quadratic.pkl')
    categorical = input_df.select_dtypes(include=[object])
    numerical = input_df.drop(categorical,axis=1)
    categorical_enc = enc.transform(categorical).toarray()
    categorical_enc_df = pd.DataFrame(categorical_enc)
    return pd.concat([numerical,categorical_enc_df],axis=1, join='inner')

def interval(model, data, enconded_input):
    # INTERVALS
    base_prediction = model.predict(data[2])
    # compute the prediction error vector on the validation set
    validation_error = (base_prediction - data[3])
    model.fit(data[2], validation_error)
    st_dev = np.abs(model.predict(enconded_input))**0.5
    return st_dev[0]

def catboost_results(input,output):
    cat1 = CatBoostRegressor()
    cat2 = CatBoostRegressor()
    cat3 = CatBoostRegressor()
    if output is 'RBE':
        modelL_rbe = cat1.load_model('./models/catboost_rbe_L.sav')
        model_rbe = cat2.load_model('./models/catboost_rbe.sav')
        modelU_rbe = cat3.load_model('./models/catboost_rbe_U.sav')
        return modelL_rbe.predict(input), model_rbe.predict(input), modelU_rbe.predict(input)
    if output is 'alpha':
        modelL_alpha = cat1.load_model('./models/catboost_alpha_L.sav')
        model_alpha = cat2.load_model('./models/catboost_alpha.sav')
        modelU_alpha = cat3.load_model('./models/catboost_alpha_U.sav')
        return modelL_alpha.predict(input), model_alpha.predict(input), modelU_alpha.predict(input)
    if output is 'beta':
        modelL_beta = cat1.load_model('./models/catboost_beta_L.sav')
        model_beta = cat2.load_model('./models/catboost_beta.sav')
        modelU_beta = cat3.load_model('./models/catboost_beta_U.sav')
        return modelL_beta.predict(input), model_beta.predict(input), modelU_beta.predict(input)

def predictions(input,models):

    data=pd.read_csv('./data/pide.csv',usecols = range(3,16)).drop(['alpha_x','beta_x'],axis=1)
    data_rbe = pd.read_csv('./data/data_rbe.csv')
    data_alpha = data.drop(['beta_l'],axis=1)
    data_beta = data.drop(['alpha_l'],axis=1)

    rbe = tools.preprocess_data(data_rbe,['RBE'])
    quadratic_alpha = tools.preprocess_data(data.drop(['beta_l'],axis=1),['alpha_l'])
    quadratic_beta = tools.preprocess_data(data.drop(['alpha_l'],axis=1),['beta_l'])


    rbe_input =  impute(input,'rbe')
    encoded_rbe_input = encode_input(input,'rbe').values
    quadratic_input = impute(input,'quadratic')
    encoded_quad_input = encode_input(input,'quadratic').values

    # predictions

    # RBE
    pred_svr_rbe = models[0][0].predict(encoded_rbe_input)[0]
    pred_gbr_rbe = models[1][0].predict(encoded_rbe_input)[0]
    pred_rf_rbe = models[2][0].predict(encoded_rbe_input)[0]
    pred_vr_rbe = models[3][0].predict(encoded_rbe_input)[0]

    # alpha
    pred_svr_alpha = models[0][1].predict(encoded_quad_input)[0]
    pred_gbr_alpha = models[1][1].predict(encoded_quad_input)[0]
    pred_rf_alpha = models[2][1].predict(encoded_quad_input)[0]
    pred_vr_alpha = models[3][1].predict(encoded_quad_input)[0]

    # beta
    pred_svr_beta = models[0][2].predict(encoded_quad_input)[0]
    pred_gbr_beta = models[1][2].predict(encoded_quad_input)[0]
    pred_rf_beta = models[2][2].predict(encoded_quad_input)[0]
    pred_vr_beta = models[3][2].predict(encoded_quad_input)[0]

    # SVR intervals
    svr_rbe_std=interval(models[0][0],rbe,encoded_rbe_input)
    svr_alpha_std = interval(models[0][1],quadratic_alpha,encoded_quad_input)
    svr_beta_std = interval(models[0][2],quadratic_beta,encoded_quad_input)
```

```
    # gbr intervals
    gbr_rbe_std=interval(models[1][0],rbe,encoded_rbe_input)
    gbr_alpha_std = interval(models[1][1],quadratic_alpha,encoded_quad_input)
    gbr_beta_std = interval(models[1][2],quadratic_beta,encoded_quad_input)

    # rf intervals
    rf_rbe_std=interval(models[2][0],rbe,encoded_rbe_input)
    rf_alpha_std = interval(models[2][1],quadratic_alpha,encoded_quad_input)
    rf_beta_std = interval(models[2][2],quadratic_beta,encoded_quad_input)

    # VR intervals
    vr_rbe_std = interval(models[3][0],rbe,encoded_rbe_input)
    vr_alpha_std = interval(models[3][1],quadratic_alpha,encoded_quad_input)
    vr_beta_std = interval(models[3][2],quadratic_beta,encoded_quad_input)


    # catboost results
    catboost_res_rbe = catboost_results(input,'RBE')
    catboost_res_alpha = catboost_results(input,'alpha')
    catboost_res_beta = catboost_results(input,'beta')

    headers = ['model','RBE','alpha','beta']
    dict = {'model':['Catboost Regression',
                     'Support Vector Regression',
                     'Gradient Boosting Regression',
                     'Random Forest Regression',
                     'Voting Regression'],

    'RBE':[
    [catboost_res_rbe[0],catboost_res_rbe[1],catboost_res_rbe[2]],
    [pred_svr_rbe - svr_rbe_std,pred_svr_rbe,pred_svr_rbe + svr_rbe_std],
    [pred_gbr_rbe - gbr_rbe_std,pred_gbr_rbe,pred_gbr_rbe + gbr_rbe_std],
    [pred_rf_rbe - rf_rbe_std,pred_rf_rbe,pred_rf_rbe + rf_rbe_std],
    [pred_vr_rbe - vr_rbe_std,pred_vr_rbe,pred_vr_rbe + vr_rbe_std]],
    'alpha':[[catboost_res_alpha[0],catboost_res_alpha[1],catboost_res_alpha[2]],
    [pred_svr_alpha - svr_alpha_std,pred_svr_alpha,pred_svr_alpha + svr_alpha_std],
    [pred_gbr_alpha - svr_alpha_std,pred_svr_alpha,pred_svr_alpha + svr_alpha_std],
    [pred_rf_alpha - rf_alpha_std,pred_rf_alpha,pred_rf_alpha + rf_alpha_std],
    [pred_vr_alpha - vr_alpha_std,pred_vr_alpha,pred_vr_alpha + vr_alpha_std]],
    'beta':[[catboost_res_beta[0],catboost_res_beta[1],catboost_res_beta[2]],
    [pred_svr_beta - svr_beta_std,pred_svr_beta,pred_svr_beta + svr_beta_std],
    [pred_gbr_beta - gbr_beta_std,pred_gbr_beta,pred_gbr_beta + gbr_beta_std],
    [pred_rf_beta - rf_beta_std,pred_rf_beta,pred_rf_beta + rf_beta_std],
    [pred_vr_beta - vr_beta_std ,pred_vr_beta,pred_vr_beta + vr_beta_std ]]}
    for key, value in dict.items():
        if key is not 'model':
            for v in value:
                v[:] = [r.astype(float).round(3) for r in v ]
                # v[:] = [0 if i <0 else i for i in v ]
    df = pd.DataFrame(dict, columns = headers)
    return df


# cell = input('cell:')
# type = input('tumor or normal cell (t/n):')
# phase = input('phase:')
# genl = input('genomic length (in bp):')
# ion = input('ion:')
# charge = input('charge:')
# irmods = input('irmods:')
# let = input('LET:')
# E =input('E:')

input =['','','','','',2,'s',15,134]

models = load_models()
results = predictions(input,models)
names = results['model'].values

rbe = results['RBE'].values
rbe_mid=[]
rbe_l=[]
rbe_u=[]
for l in rbe:
    rbe_mid.append(l[1])
    rbe_l.append(l[0])
```

```
    rbe_u.append(l[2])

alpha = results['alpha'].values
alpha_mid=[]
alpha_l=[]
alpha_u=[]
for l in alpha:
    alpha_mid.append(l[1])
    alpha_l.append(l[0])
    alpha_u.append(l[2])

beta = results['beta'].values
beta_mid=[]
beta_l=[]
beta_u=[]
for l in beta:
    beta_mid.append(l[1])
    beta_l.append(l[0])
    beta_u.append(l[2])

fig, axs = plt.subplots(1, 3, figsize=(15, 3))
# colors = {'Catboost Regression':'r',
            'Support Vector Regression':'k',
            'Gradient Boosting Regression':'g',
            'Random Forest Regression':'b',
            'Voting Regression':'y'}
# axs[0].set_prop_cycle(
colors=['r','b','k','y','c']
dots=[]
for name,mid,low,up,color in zip(names,rbe_mid,rbe_l,rbe_u,colors):
    dot = axs[0].scatter(name,mid, c=str(color),s=3,label=name)
    axs[0].scatter(name,low, c=str(color),marker='_')
    axs[0].scatter(name,up, c=str(color),marker='_')
    plt.setp(axs[0].get_xticklabels(), visible=False)
    dots.append(dot)
fig.legend(
           loc="center right",   # Position of legend
           borderaxespad=0.1,    # Small spacing around legend box
           title="Legend"  # Title for the legend
           )
for name,mid,low,up,color in zip(names,alpha_mid,alpha_l,alpha_u,colors):
    axs[1].scatter(name,mid, c=str(color),s=3)
    axs[1].scatter(name,low, c=str(color),marker='_')
    axs[1].scatter(name,up, c=str(color),marker='_')
    plt.setp(axs[1].get_xticklabels(), visible=False)
for name,mid,low,up,color in zip(names,beta_mid,beta_l,beta_u,colors):
    axs[2].scatter(name,mid, c=str(color),s=3)
    axs[2].scatter(name,low, c=str(color),marker='_')
    axs[2].scatter(name,up, c=str(color),marker='_')
    plt.setp(axs[2].get_xticklabels(), visible=False)
fig.subplots_adjust(left=0.03,right=0.8,top=0.8)
fig.suptitle('Prediction Intervals')
axs[0].set_title("RBE")
axs[1].set_title("alpha")
axs[2].set_title("beta")
plt.show()
```

Bibliography

[1] "The effect of multiple small doses of x rays on skin reactions in the mouse and a basic interpretation.," 1976.

[2] H. Pass, D. Ball, and G. Scagliotti, *IASLC Thoracic Oncology E-Book*. Elsevier Health Sciences, 2017.

[3] E. Hall and A. Giaccia, *Radiobiology for the Radiologist*. Lippincott Williams & Wilkins, 2006.

[4] S.-F. D. Obe G., Johannes C., "Dna double-strand breaks induced by sparsely ionizing radiation and endonucleases as critical lesions foe cell death, chromosomal aberrations, mutations and oncogenic transformation.," 1992.

[5] The contibutors of Encyclopedia Britannica, *Delta Ray*. Encyclopedia Britannica,inc, 2019.

[6] ICRU, "Report 16," *Journal of the International Commission on Radiation Units and Measurements*, vol. os9, pp. NP–NP, 04 2016.

[7] E. A. Blakely, F. Q. Ngo, S. B. Curtis, and C. A. Tobias, "Heavy-ion radiobiology: Cellular studies," in *Advances in radiation biology*, vol. 11, pp. 295–389, Elsevier, 1984.

[8] R. L. Kathren, "Historical development of the linear nonthreshold dose-response model as applied to radiation," *Pierce L. Rev.*, vol. 1, p. 5, 2002.

[9] S. Tapio and V. Jacob, "Radioadaptive response revisited," *Radiation and environmental biophysics*, vol. 46, no. 1, pp. 1–12, 2007.

[10] D. Wodarz, R. Sorace, and N. L. Komarova, "Dynamics of cellular responses to radiation," *PLoS computational biology*, vol. 10, no. 4, p. e1003513, 2014.

[11] A. R. Snyder, "Review of radiation-induced bystander effects," *Human & experimental toxicology*, vol. 23, no. 2, pp. 87–89, 2004.

[12] N. Hamada, H. Matsumoto, T. Hara, and Y. Kobayashi, "Intercellular and intracellular signaling pathways mediating ionizing radiation-induced bystander effects," *Journal of radiation research*, pp. 0702230007–0702230007, 2007.

[13] T. M. Pawlik and K. Keyomarsi, "Role of cell cycle in mediating sensitivity to radiotherapy," *International Journal of Radiation Oncology* Biology* Physics*, vol. 59, no. 4, pp. 928–942, 2004.

[14] M. Jung and A. Dritschilo, "Signal transduction and cellular responses to ionizing radiation," in *Seminars in radiation oncology*, vol. 6, pp. 268–272, Elsevier, 1996.

[15] E. Yonish-Rouach, D. Grunwald, S. Wilder, A. Kimchi, E. May, J. Lawrence, P. May, and M. Oren, "p53-mediated cell death: relationship to cell cycle control.," *Molecular and Cellular Biology*, vol. 13, no. 3, pp. 1415–1423, 1993.

[16] A. J. Wagner, J. M. Kokontis, and N. Hay, "Myc-mediated apoptosis requires wild-type p53 in a manner independent of cell cycle arrest and the ability of p53 to induce p21waf1/cip1.," *Genes & Development*, vol. 8, no. 23, pp. 2817–2830, 1994.

[17] T. Samuel, H. O. Weber, and J. O. Funk, "Linking dna damage to cell cycle checkpoints," *Cell cycle*, vol. 1, no. 3, pp. 161–167, 2002.

[18] R. Parshad, K. K. Sanford, and G. M. Jones, "Chromatid damage after g2 phase x-irradiation of cells from cancer-prone individuals implicates deficiency in dna repair," *Proceedings of the National Academy of Sciences*, vol. 80, no. 18, pp. 5612–5616, 1983.

[19] M. Shahidi, H. Mozdarani, and P. E. Bryant, "Radiation sensitivity of leukocytes from healthy individuals and breast cancer patients as measured by the alkaline and neutral comet assay," *Cancer letters*, vol. 257, no. 2, pp. 263–273, 2007.

[20] M. Shahidi, H. Mozdarani, and W. U. Mueller, "Radiosensitivity and repair kinetics of gamma-irradiated leukocytes from sporadic prostate cancer patients and healthy individuals assessed by alkaline comet assay," *Iranian biomedical journal*, vol. 14, no. 3, p. 67, 2010.

[21] A. Mohseni-Meybodi, H. Mozdarani, and S. Mozdarani, "Dna damage and repair of leukocytes from fanconi anaemia patients, carriers and healthy individuals as measured by the alkaline comet assay," *Mutagenesis*, vol. 24, no. 1, pp. 67–73, 2008.

[22] T. J. Jorgensen, "Enhancing radiosensitivity: targeting the dna repair pathways," *Cancer biology & therapy*, vol. 8, no. 8, pp. 665–670, 2009.

[23] E. J. Moding, M. B. Kastan, and D. G. Kirsch, "Strategies for optimizing the response of cancer and normal tissues to radiation," *Nature reviews Drug discovery*, vol. 12, no. 7, p. 526, 2013.

[24] M. Toulany and H. P. Rodemann, "Potential of akt mediated dna repair in radioresistance of solid tumors overexpressing erbb-pi3k-akt pathway," *Translational Cancer Research*, vol. 2, no. 3, pp. 190–202, 2013.

[25] J. A. Engelman, "Targeting pi3k signalling in cancer: opportunities, challenges and limitations," *Nature Reviews Cancer*, vol. 9, no. 8, p. 550, 2009.

[26] L. Liu, X.-D. Zhu, W.-Q. Wang, Y. Shen, Y. Qin, Z.-G. Ren, H.-C. Sun, and Z.-Y. Tang, "Activation of β-catenin by hypoxia in hepatocellular carcinoma contributes to enhanced metastatic potential and poor prognosis," *Clinical cancer research*, vol. 16, no. 10, pp. 2740–2750, 2010.

[27] T. Friedrich, U. Scholz, T. ElsäSser, M. Durante, and M. Scholz, "Systematic analysis of rbe and related quantities using a database of cell survival experiments with ion beam irradiation," *Journal of radiation research*, vol. 54, no. 3, pp. 494–514, 2012.

[28] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin, "Catboost: unbiased boosting with categorical features," in *Advances in Neural Information Processing Systems*, pp. 6638–6648, 2018.

[29] J. H. Friedman, "Stochastic gradient boosting," *Computational statistics & data analysis*, vol. 38, no. 4, pp. 367–378, 2002.

[30] E. Osuna, R. Freund, and F. Girosi, "An improved training algorithm for support vector machines," in *Neural networks for signal processing VII. Proceedings of the 1997 IEEE signal processing society workshop*, pp. 276–285, IEEE, 1997.

[31] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[32] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection*, pp. 1137–1143, Morgan Kaufmann, 1995.

[33] S. S. Shapiro and M. B. Wilk, "An analysis of variance test for normality (complete samples)," *Biometrika*, vol. 52, no. 3/4, pp. 591–611, 1965.

[34] N. S. Altman, "An introduction to kernel and nearest-neighbor nonparametric regression," *The American Statistician*, vol. 46, no. 3, pp. 175–185, 1992.

[35] J. H. Friedman, B. E. Popescu, *et al.*, "Predictive learning via rule ensembles," *The Annals of Applied Statistics*, vol. 2, no. 3, pp. 916–954, 2008.

[36] S. Chen, S. Zhou, F.-F. Yin, L. B. Marks, and S. K. Das, "Investigation of the support vector machine algorithm to predict lung radiation-induced pneumonitis," *Medical physics*, vol. 34, no. 10, pp. 3808–3814, 2007.

[37] J. A. Cruz and D. S. Wishart, "Applications of machine learning in cancer prediction and prognosis," *Cancer informatics*, vol. 2, p. 117693510600200030, 2006.

[38] G. Valdes, T. D. Solberg, M. Heskel, L. Ungar, and C. B. Simone II, "Using machine learning to predict radiation pneumonitis in patients with stage i non-small cell lung cancer treated with stereotactic body radiation therapy," *Physics in Medicine & Biology*, vol. 61, no. 16, p. 6105, 2016.

[39] K. Jayasurya, G. Fung, S. Yu, C. Dehing-Oberije, D. De Ruysscher, A. Hope, W. De Neve, Y. Lievens, P. Lambin, and A. Dekker, "Comparison of bayesian network and support vector machine models for two-year survival prediction in lung cancer patients treated with radiotherapy," *Medical physics*, vol. 37, no. 4, pp. 1401–1407, 2010.

[40] R. Caruana and A. Niculescu-Mizil, "An empirical comparison of supervised learning algorithms," in *Proceedings of the 23rd international conference on Machine learning*, pp. 161–168, ACM, 2006.

[41] B. P. Roe, H.-J. Yang, J. Zhu, Y. Liu, I. Stancu, and G. McGregor, "Boosted decision trees as an alternative to artificial neural networks for particle identification," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 543, no. 2-3, pp. 577–584, 2005.

[42] Q. Wu, C. J. Burges, K. M. Svore, and J. Gao, "Adapting boosting for information retrieval measures," *Information Retrieval*, vol. 13, no. 3, pp. 254–270, 2010.

[43] Y. Zhang and A. Haghani, "A gradient boosting method to improve travel time prediction," *Transportation Research Part C: Emerging Technologies*, vol. 58, pp. 308–324, 2015.

[44] E. Pons, L. M. Braun, M. M. Hunink, and J. A. Kors, "Natural language processing in radiology: a systematic review," *Radiology*, vol. 279, no. 2, pp. 329–343, 2016.