AN ABSTRACT OF THE THESIS OF

Tanjin Xu for the degree of Master of Science in Computer Science presented on May 31, 2016.

Title: Exploration of Regression Models for Cancer Noncoding Mutation Recurrence

Abstract approved: ______________________________

Stephen A. Ramsey

An important impact of the genome technology revolution will be the elucidation of mechanisms of cancer pathogenesis, leading to improvements in the diagnosis of cancer and the selection of cancer treatment. Integrated with current well-studied massive knowledge and findings about the role of protein-coding mutations in cancer, demystifying the functional role of human "junk" DNA (non-protein-coding DNA) mutations for cancer development and progression is one of the most popular and promising approaches these days to improve our understanding of the complicated cellular mechanisms in cancer. In light of one recent finding that non-protein-coding driver mutations tend to be highly recurrent, in this thesis we explore three different kinds of regression models for predicting non-protein-coding mutation recurrence: generalized linear models, conventional machine learning approaches and deep neural network learning models. We compare the regression model results and find the most accurate model for non-protein-coding mutation recurrence prediction so that we can prioritize somatic mutations based on their predicted recurrence and provide insights for further biological validation and eventually improved cancer therapy.

Exploration of Regression Models for Cancer Noncoding Mutation Recurrence

by

Tanjin Xu

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Presented May 31, 2016
Commencement June 2016

Master of Science thesis of Tanjin Xu presented on May 31, 2016.

APPROVED:

Major Professor, representing Computer Science

Director of the School of Electrical Engineering and Computer Science

Dean of the Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

Tanjin Xu, Author

ACKNOWLEDGEMENTS

I have been quite lucky to have Professor Stephen Ramsey as my supervisor. Steve's professional ability to clearly and cleanly formalize problems and intuitions is unsurpassed in my experience. Working with Steve has fundamentally changed the way that I think about research and working on projects. And also great thanks to Steve for his financial support, inspiration and encouragement for two years in the lab. I strongly believe all that I have learned from him will benefit a lot in my future career.

I would also like to thank all other committee members, Prof. Xiaoli Fern, Prof. David Hendrix and Prof. Hector A. Vergara for their insights, suggestions and their time for my thesis defence.

Thanks to all my colleagues and friends for their support and help, especially thanks to Dr. Jichen Yang, Jun He, Xin Liu, Yao Yao and Zheng Liu for putting up with me throughout my graduate life in U.S..

And finally, I am so grateful to my family. Without continuous strong support and understanding of my parents, I can not step so far. Great thanks to my wife Li Li for her deep love and standing behind me all the time.

TABLE OF CONTENTS

Page

1 Introduction 1

1.1 Background 1

1.2 Cancer Overview 2

1.3 Noncoding Mutations 4

1.4 Mutation Recurrence 5

1.5 Research Objective 6

1.6 Outline 7

2 Literature Review 8

2.1 Driver Mutation Detection 8

2.2 Generalized Linear Regression Models 11

2.2.1 Poisson Regression 12

2.2.2 Negative Binomial Regression 14

2.2.3 Goodness of Fit 15

2.3 Conventional Machine Learning Approaches 16

2.3.1 Random Forest Regression 17

2.3.2 Gradient Boosted Regression Trees 19

2.3.3 Ada Boosting Regression 20

2.4 Deep Learning Approach 22

2.4.1 Neural Network Regression 22

2.4.2 Stacked AutoEncoder Regression 24

2.5 Cross-Validation 27

3 Materials and Methods 28

3.1 Datasets and Feature Selection 29

3.2 Annotation of Noncoding Mutations 34

3.3 Regression Analysis 36

3.3.1 Definition of Mutation Recurrence 37

3.3.2 Generalized Linear Models 37

3.3.3 Ensemble of Decision Trees Models 40

3.3.4 Deep Neural Network 44

3.4 Model Evaluation 48

3.4.1 Explained Variance Score 49

TABLE OF CONTENTS (Continued)

Page

3.4.2 Mean Absolute Error 49
3.4.3 Mean Squared Error 49
3.4.4 Median Absolute Error 50
3.4.5 R^2 score 50

3.5 Overall Procedure 51

4 Results 52

4.1 Recurrence Distribution 53

4.2 GLM Regression Analysis 54

4.3 Ensemble Methods Analysis 57

4.4 Deep Neural Network Regression Analysis 59

4.5 Overall Performance Summary 60

4.6 Feature Importances 61

5 Discussion 63

6 Conclusion 65

Bibliography 66

LIST OF FIGURES

Figure Page

1.1 The prevalence of somatic mutations across human cancer types (reproduced from [1]) 3

1.2 Noncoding mutation frequency in different regions (reproduced from [82] where error bar represents the mean standard error) 5

2.1 Overview of strategies for driver detection [65] 9

2.2 Random Forest Classifier for driver detection (reproduced from [75]) 10

2.3 Poisson and Negative Binomial Distribution 12

2.4 Poisson Regression dispersion [48] 14

2.5 Random Forest Regression Models [33]. As the figures shows, the original input X is bootstrapped sampled to build different decision trees. And in the last step to predict $\hat{y}$ using the weighted averaging strategy. 17

2.6 Feed-Forward Neural Network Model [76]. In the forward phase, from the input layer to the output layer, the activation values of the hidden nodes and the output are calculated, and also the gradients. In the backward phase, from the output layer to the input layer, the weights are updated layer by layer using a backpropagation algorithm. 23

2.7 Stacked Auto Encoder Pretraining [57]. In the example above, train the first hidden layer $h^{(1)}$ (encoder) and drop the output, then use the $h^{(1)}$ as input to train the second encoder $h^{(2)}$ and also drop the output. And in the last train the weights from the second hidden layer to the output layer. All the activation values and weights were memorized and re-updated. 25

2.8 Stacked Auto Encoder Fine-tuning [57]. Put the input layer X, the two pre-trained encoders $h^{(1)}, h^{(2)}$ and the output layer all together, and also reuse the learned weights to fit the input data again. Use the backpropagation based on gradient descent to update the weights. 26

3.1 Example rows from noncoding mutation dataset from the COSMIC database . . 29

3.2 Noncoding mutation and TSS distance 31

3.3 Noncoding mutation and GC-content 33

LIST OF FIGURES (Continued)

Figure Page

3.4 Noncoding intergenic regions 36

3.5 Noncoding mutation recurrence 37

3.6 Neural Network Nonlinear Representation (figure reproduced from [80]) 45

3.7 Stacked AutoEncoder Regression 48

3.8 Overall Regression Analysis Procedures 51

4.1 Noncoding mutation data distribution: (a) the frequency (number of occurrence) of noncoding mutation recurrence; (b) log frequency of overall noncoding mutations; (c) the frequency of noncoding mutations with recurrence less than 100; (d) as a function of the recurrence count, the Poisson model-predicted and empirical log frequency of mutations (plus the log factorial). 53

4.2 MSE of GLM regression models with different sampling rates 55

4.3 GLM regression models performance with different sampling rates 56

4.4 Ensemble regression models test performance with different sampling rates . . 57

4.5 Ensemble regression models training performance with different sampling rates 58

4.6 Deep neural network regression training error with different sampling rates . . 59

4.7 Deep neural network regression test error with different sampling rates 60

4.8 Overall regression models test MSE/MAE with different sampling rates 61

4.9 Variable importances score 62

LIST OF TABLES

Table		Page
3.1	*Twenty-nine* noncoding mutation features for regression	30
4.1	Recurrence variance/mean ratio	54
4.2	GLM regression statistics	55
4.3	Ensemble methods parameters tuning	57
4.4	Neural Network regression parameters tuning	59

LIST OF ALGORITHMS

Algorithm		Page
1	RANDOM FOREST REGRESSION	18
2	GRADIENT BOOSTING METHOD	19
3	ADABOOST.R2 [20]	21
4	TF BINDINGS SITES FEATURES EXTRACTION	32
5	NONCODING MUTATIONS ANNOTATION	35
6	NEURAL NETWORK REGRESSION	46
7	STACKED AUTOENCODER REGRESSION	47

Chapter 1: Introduction

Cancer is the second leading cause of death in the United States [56]. As of 2009, approximately 40% of Americans are expected to develop cancer in their lifetime, and approximately 50% of these individuals are expected to die of disease [66]. Given the worldwide increase in the incidence of cancer and the urgent need to find ways for prevention and cure, cancer research is progressing at a rapid rate toward understanding the mechanisms of the transformation of a normal cell into cancer and eventually spreading to different organs – a process called metastasis [14].

1.1 Background

Today our knowledge of cancer is advancing at an unprecedented rate. It is well established that cancer primarily develops because of somatic alterations in the genome [82]. Many genomic events with direct phenotypic impact have been identified and a subset of critical mutations have been successfully targeted therapeutically [65]; for example, imatinib has been used to target cells expressing the *BCR-ABL* fusion gene in chronic myeloid leukemia [27], and gefitinib has been used to inhibit the epidermal growth factor receptor in lung cancer [58].

However, the more we understand about cancer, the more its complexity becomes apparent [25]. Cancer can take hundreds of different forms depending on the location, cell of origin and spectrum of genomic alterations that promote oncogenesis and affect therapeutic response [83]. On the other hand, cancer patients are extremely heterogeneous in their responses to treatments and disease outcomes [63]. No two individuals' cancers are identical due to the fact that cancer arises from the selection of specific point mutations, structural variants and epigenetic alterations from a large pool of such variation. To date, the vast majority of studies of somatic mutations in cancer have focused on coding regions; however, the protein-coding component of the genome accounts for less than 2% of the total sequence, and there is very little information on how noncoding variation affects cancer development and progression [82].

1.2 Cancer Overview

Cancer is fundamentally a genetic disease in that it results from dysregulation of the gene networks that maintain normal cellular identity, growth and differentiation [12]. And the cause is mainly due to genomic instability which refers to an increased tendency of alteration in the genome during the life cycle of cells [71]. Only a small proportion of cancers are attributed to heritable single-gene disorders (also called germline mutations), usually involving non-synonymous mutations in the coding sequence of protein-coding genes, such as *BRCA1* in familiar breast cancer and *RB1*, which causes familial retinoblastoma [12]. The majority of cancers primarily develops because of somatic alterations in the genome [82]. These alterations include nucleotide substitutions, small insertions and deletions, large chromosomal rearrangements(also called structural variants) and copy number changes [39]. Such alterations often, of course, concomitantly alter genes in a number of ways that may be critical to cancer onset or progression, such as adverse effect on protein-coding and regulatory components of genes, and especially might affect the two important types of genes in cancer - oncogenes which encodes a protein capable of transforming cells in culture or inducing cancer, and tumor-suppressor genes which encode proteins that normally inhibit cell division [35].

Genomic coding mutations have been studied for decades using low-throughput approaches such as targeted gene sequencing or cytogenetic techniques, which have led to the identification of a number of highly recurrent somatic mutations [27, 81]. In the past few years, revolutionary advances in next-generation sequencing (NGS) technologies have enabled Whole-genome Sequencing and the Whole-exome Sequencing with much greater speed and much lower cost. This has engendered the acquisition of massive amounts of cancer genomic data and provided somatic mutation landscapes for better understanding cancer biology and improving cancer diagnosis and therapy [13]. Since many gene alterations might not have been observed before and might not necessarily reside in coding regions of the genome, whole-genome sequencing is increasingly regarded as a promising approach that can find all the variants in a cancer genome. Based on the assumption that changes in gene expression levels impact disease progression, high-throughput short-read sequencing of mRNA (RNA-seq) is increasingly employed to determine whether and how these genetic alterations impact cancer progression [31].

So far, somatic mutations of more than 20 cancer types have been systematically explored. The COSMIC (Catalogue Of Somatic Mutations In Cancer) database [3] contains information on 3.1 million coding mutations and 4.5 million noncoding mutations. In addition, large-scale

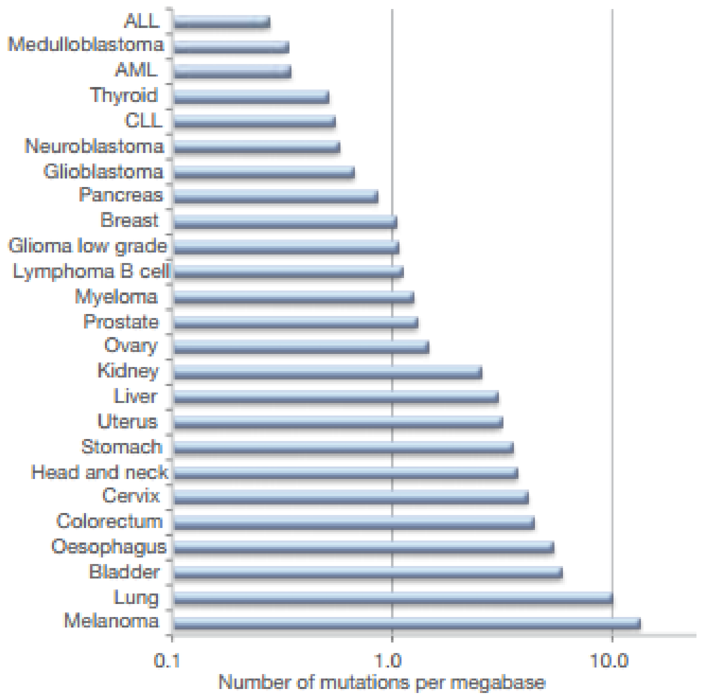


Figure 1.1: The prevalence of somatic mutations across human cancer types (reproduced from [1])

cancer genomics projects such as The Cancer Genome Atlas (TCGA) and the International Cancer Genome Consortium (ICGC [39]) have profiled tens of thousands of cancer samples [82]. As of May 2015, nearly 13 million somatic mutations have been uncovered by ICGC [13].

The mutation rate in cancer genomes varies by 1000-fold among different cancer types [44]. Most solid tumor genomes harbor hundreds of sequence-level genetic alterations. The majority of these alterations are expected to be passenger mutations that do not confer a selective growth advantage, while a few are driver mutations which have a selective growth advantage in tumor cells [81]. Though it is easy to define a "driver gene mutation" in physiologic terms (as one conferring a selective growth advantage), it is more difficult to identify which somatic mutations are drivers and which are passengers. Moreover, it is important to point out that there is a fundamental difference between a driver gene and a driver gene *mutation.* A driver gene is one that contains driver gene mutations. But driver genes may also contain passenger gene mutations [81].

1.3 Noncoding Mutations

Despite extensive study on coding variants, the majority of the genetic component of cancer susceptibility has not yet been linked to individual genes, highlighting significant deficiencies in our understanding of the molecular basis of cancer [12]. A key development in unravelling the complex genetics of cancer may be the shift in focus from looking exclusively at the protein-coding components of the genome to consideration of the role of variation in genomic regulatory elements. The Encyclopedia of DNA Elements (ENCODE) Project has stimulated a dramatic reassessment of the information content of the human genome [82]. Rather than islands of protein-coding genes in a sea of junk DNA, it is increasingly apparent that much of the genome, far more than expected, encodes regulatory information. Indeed the ENCODE project recently concluded that although only $\sim$1.2% of the genome is protein-coding, at least 20% shows biological function and over 80% exhibits biochemical indices of function [12, 17].

Of these non-protein-coding sequences, a large portion contains regulatory elements. Broadly speaking, they consist of *cis*-regulatory regions and noncoding RNAs (ncRNAs). Cis-regulatory regions include promoters and distal elements (enhancers, silencers and insulators), which regulate gene expression following binding by transcription factors (TFs) [42]. It is well known that somatic mutations in noncoding regions are frequent, but their effects are poorly understood [82]. Recent efforts to understand noncoding variation in the human population have shown that disease-associated genomic variation is commonly located in regulatory elements [82]. A notable recent discovery is that recurrent somatic mutations in the promoter region of the gene *TERT* have been found in multiple cancer types including cancers of the central nervous system, bladder, thyroid, and skin [78]; and a more distally located enhancer mutation upstream of *TAL1* in T-cell acute lymphoblastic leukemia [50]. These two examples of driver mutations generate *de novo* binding sites for oncogenic transcription factors. Particularly, the *TERT* promoter mutations create new ETS-like binding sites leading to their binding to the *TERT* promoter and subsequent upregulation of gene expression, while the *TAL1* mutation creates a MYB binding site [75].

Accumulating evidence indicates that *cis*-regulatory element alteration is associated with multiple diseases [51]. This is illustrated by the human gene mutation database (HGMD [74]), which includes more than 3,000 disease-implicated mutations categorized as 'regulatory'. A meta-analysis of $\sim$1,200 GWAS (Genome-Wide Association Study) SNPs showed that more than one third of noncoding variants are likely to be causal for the phenotypic or disease traits

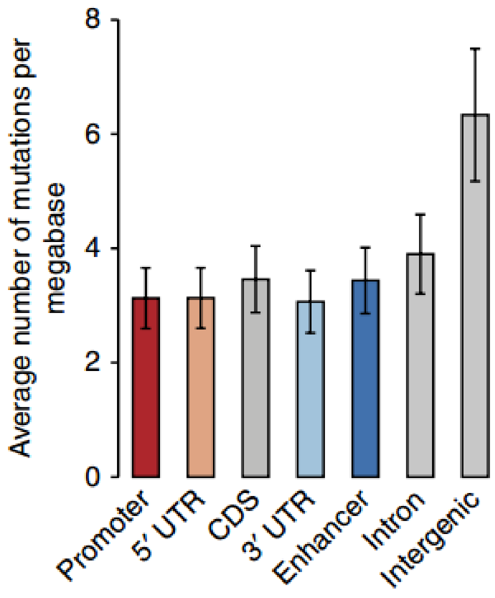


Figure 1.2: Noncoding mutation frequency in different regions (reproduced from [82] where error bar represents the mean standard error)

observed [79]. Of particular relevance is cancer, which has been described as disease of disrupted gene regulation. It is therefore unsurprising that non-coding variations are linked to tumorigenesis [51].

1.4 Mutation Recurrence

As illustrated in the above section, driver mutations are those that grant cells a survival advantage, while the passenger mutations are those that have been acquired at some point during clonal evolution but do not provide a substantial survival advantage [31]. Therefore driver mutations contribute to oncogenesis whereas the majority are passenger mutations accumulated during cancer progression. A major challenge of cancer research is to differentiate these two types of mutations. In the current state-of-art, bioinformatic identification of driver mutations is based on two lines of evidence: detection of signals of positive selection [44], namely the presence of more recurrent mutations than expected by random chance, or prediction of mutations with high functional impact [26]. Analysis of the recurrence of somatic variants from tumour samples in functional elements to identify regions under positive selection is similar to the burden test strategy that is used to associate rare germline variants with complex traits [42].

A case of study [82] of 864 human tumors highlighted recurrent mutations in promoter regions of the *PLEKHS1, WSR74*, and *SHDH* along with those already known to be associated to the *TERT* gene which encodes the catalytic subunit of telomerase. The recurrent mutations found in the *SHDH* promoter have been shown to be associated with reduced gene expression. Tumours in tissues with relatively low rates of self-renewal (including melanomas, urothelial carcinomas and medulloblastomas) tend to exhibit higher frequencies of *TERT* promoter mutations [42]. The high occurrence of these mutations points to their role as driver as opposed to passenger mutations [42].

In general, computational identification of coding drivers is based on mutation recurrence. Computational identification of non-coding drivers is in many ways even more challenging than coding drivers owing to their complex and varied modes of action and our poor understanding of noncoding regions in general. Noncoding mutations are also more abundant than coding ones and thus the key mutations with functional impact have to be distinguished from a larger set of passenger events [42]. However, in light of the recent findings (as illustrated above) about high recurrent mutations in the regulatory noncoding regions including promoters and enhancers and those noncoding mutations were also identified as drivers, it is still promising to apply mutation recurrence approach to distinguish the noncoding driver mutations from the passenger mutations.

1.5 Research Objective

It is well established that the protein-coding "driver" mutations are highly recurrent, i.e., they occur in many tumour samples high frequently and also in many different cancer types [75]. For noncoding mutations, recent studies [42, 52, 75, 82], especially for the finding of *TERT* promoter mutations, suggests that the noncoding "driver" mutations also show high recurrence. Therefore in light of these studies and the growing availability of whole-genome cancer sequencing, in this study, we analyzed noncoding mutations from the COSMIC database. We developed a model of mutation recurrence counts and we explored different kinds of regression models to find the best model to fit the recurrence counts of noncoding mutations. Our rationale was that if successful, this model could be used to predict the potential of a newly discovered somatic mutation to be a driver mutation, and by extension, it could be used to rank novel somatic mutations by their predicted cancer-driving potential.

To achieve this, our study involved three steps. First, we annotated noncoding mutations focusing on the regulatory region in genome, such as $5'$ untranslated regions (UTRs) and intergenic regions including promoters and the enhancers. Second, we extracted the most important genomic features for each noncoding mutation, such as transcription factor binding sites, the distance to the nearest gene transcription start sites, and DNase I hypersensitive sites. Third, we built mutation count response models and we explored and evaluated various linear and non-linear regression models. The details are illustrated in Chapter 3.

1.6 Outline

In the following sections, the content is organized as described below. Chapter 2 summarizes the current state-of-art computational methods for detecting noncoding mutations, and also the regression models used for count data which are discrete integers. Additionally, we analyze the accuracy of traditional statistical machine learning approaches and the most popular deep learning techniques for regression problems, for predicting cancer mutation counts. Chapter 3 illustrates the count models used for regression analysis to predict noncoding mutation recurrence and the computational methods and algorithms. Chapter 4 shows the regression results and in Chapter 5 we discuss the results from our analysis. In Chapter 6, we summarize our analysis and discuss possible follow-on studies.

Chapter 2: Literature Review

Computational identification of driver (vs. passenger) mutations is a challenging task. Below we discuss various methods for recognizing cancer driving mutations in non-coding regions, and also the computational models that can be used for regression analysis of discrete counts data, including both the traditional statistical machine learning methods and the recent popular deep learning approach.

2.1 Driver Mutation Detection

To analyze regulatory mutations on the genome-wide scale and to prioritize candidate driver mutations, several types of information can be integrated. A first class of methods is based on *filtering* all candidate mutations, such as single nucleotide variants (SNV) and small indels, to retain only those that affect "interesting" nucleotides [75]. For example, a method called FunSeq [2] retains mutations that affect "sensitive" genomic positions. Sensitive positions are determined by FunSeq as positions that are significantly infrequently substituted in the normal human population. Other methods like OncoCis [61] and RegulomeDB [7], retain mutations that are located in candidate regulatory regions, as determined by publicly available regulatory data (e.g., from ENCODE). The disadvantage of this approach is that regulatory activity observed in a cancer sample may not correspond to any of the available annotation, particularly when the mutation creates a gain-of-function *cis*-regulatory module (CRM), or in other words, publicly available regulatory annotation information is not always indicative for the function of the CRM in cancer.

A second class of approaches used information about transcription factor (TF) binding site sequence motifs and selects mutations that affect transcription binding sites by scoring the reference and mutated sequences with a position weight matrix (PWM) of a particular TF, assessing the impact of the mutation by the difference of the scores for the reference and mutated sequences [75]. One limitation of these methods is that PWM-scanning methods are notorious for generating high amount of false positive predictions, which can affect the accuracy of PWM-based mutation scoring, yielding excessive amounts of false-positive driver mutations [75].

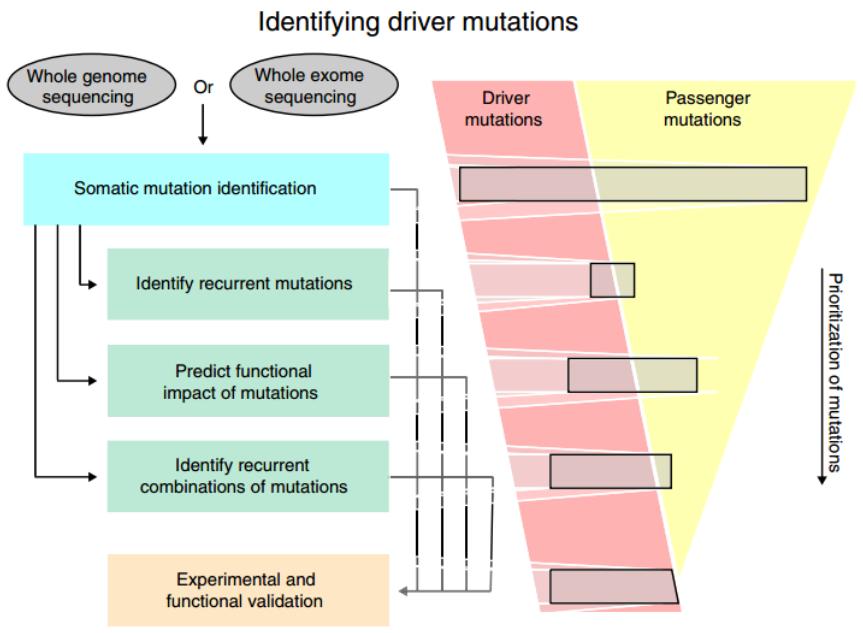

Identifying driver mutations
Whole genome sequencing
Or
Whole exome sequencing
Somatic mutation identification
Identify recurrent mutations
Predict functional impact of mutations
Identify recurrent combinations of mutations
Experimental and functional validation
Driver mutations
Passenger mutations
Prioritization of mutations

Figure 2.1: Overview of strategies for driver detection [65]

The first two classes of identification methods require biological prior knowledge and mostly focus on functional analysis of the noncoding mutations. Some other methods integrate these types of functional analysis with gene expression analysis [22]. Meanwhile there are also a number of computational tools that exist to annotate and prioritize potentially functional non-coding mutations with high impact as listed in Table 2 of the article by Khurana *et al.* [42]. However, the main problem with these methods is their low accuracy and their dependence on biological prior knowledge. Since in this thesis we try to explore regression models to predict noncoding mutation recurrence with the features of well-known or more likely to be correlated with the mutations in the noncoding regions in cancer, we focus more on statistical learning methods and computational algorithms.

One method for identifying non-coding driver mutations involves analyzing the recurrence of somatic mutations from tumour samples in functional elements [22, 52, 82]. Specifically Weinhold *et al.* [82] used a hotspot analysis to identify focal regulatory regions that were significantly recurrently mutated in comparison to a random distribution of mutations across the genome and also used a regional recurrent approach targeting regulatory regions that were mutated more frequently than expected by chance. And another interesting finding by Melton *et al.* [52] is that the detected recurrent mutations in distinct noncoding positions are close to the transcription start site (TSS), suggesting that TSS distance would be an essential feature for predicting recurrent mutations in our regression analysis. Such types of methods that try to identify noncoding elements based on mutation recurrence (that is, those mutations that are undergoing positive selection within their respective tumours) also need to account for genomic mutation rate covariates (similarly to methods used for driver mutation analysis of coding exons [42]).

Another machine-learning approach, employed by Svetlichny *et al.* [75], is to use a Random Forest model for identification of regulatory mutations, as shown in Figure 2.2. They trained TF-target Random Forest classifiers for 45 cancer-related TFs to predict which *cis*-regulatory mutations may have a significant impact on gene regulation by evaluating whether the mutation causes a significant gain or loss in the probability that the CRM is a functional TF target. And the mutations with the top highest PRIME (Predicted Impact of a Mutation in an Enhancer) score were chosen as candidate drivers.

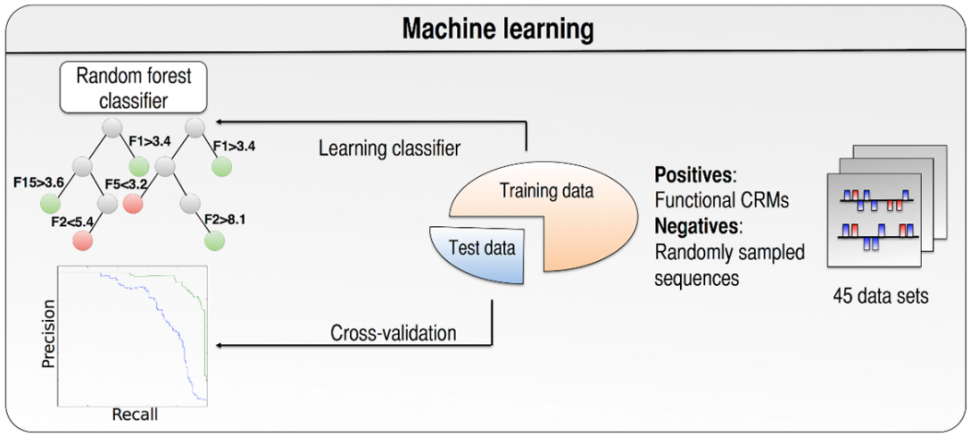


Figure 2.2: Random Forest Classifier for driver detection (reproduced from [75])

The main problem of this approach is that the classification model held an assumption that the mutations change the existing TF binding sites or create new ones and scan the mutated sequence comparing to the reference TF binding sites. However, some mutations may not directly change the existing binding sites or create new binding sites, but they may indirectly impact regulatory elements that affect cancer development and progression. And also the model only targeted 45 specific transcription factors, which may not be helpful for detecting noncoding driver mutations in regulatory regions that are not yet discovered.

In summary, computational methods for noncoding driver mutation detection are at present not fully explored. In this thesis we build a regression model to predict the recurrence of non-coding mutations using state-of-the-art regression algorithms. Since in our regression model, the response variable—the mutation recurrence—is represented by an integer count, in the section below we explore the regression models which are suited for counts data analysis.

2.2 Generalized Linear Regression Models

Regression models are the most popular tool for modelling the relationship between a response variable and a set of predictors, including linear regression models and non-linear regression models. In linear regression, the parameters appear in a linear form as:

$$y_i = \beta_1 x_{i1} + \cdots + \beta_p x_{ip} + \epsilon_i = x_i^T \beta + \epsilon_i, \quad i = 1, \ldots, n, \tag{2.1}$$

and the closed form:

$$y = X^T \beta + \epsilon, \tag{2.2}$$

where y_i represents the response variable, x_i represents the predictor, ϵ_i represents the error term, β represents the coefficient parameters, p denotes the number of predictors (dimension) and ^{T} denotes the transpose. While for the non-linear regression model, there is no requirement of the linear form of the parameters and it usually has the form as:

$$y = f(X, \beta) + \epsilon, \tag{2.3}$$

where X is a vector of p predictors, β is a vector of k parameters, f is some known regression function, and ϵ is an error term whose distribution may or may not be normal.

One of the main assumptions of linear regression model is that the residual errors follow a normal distribution. To meet this assumption when a continuous response is skew, a transformation of the response variable can produce errors that approximate normal. However, in computational biology, it is very common that the response variable y is categorical or discrete, or even limited to non-negative numbers. One obvious example is that the response variable represents the count number. In this case, the linear regression model is not appropriate for the counts data analysis because there are high number of 0's so that it is impossible to transform to normal distribution, and also the linear regression model may predict negative values. Instead, the most widely used models are Poisson regression and Negative Binomial regression.

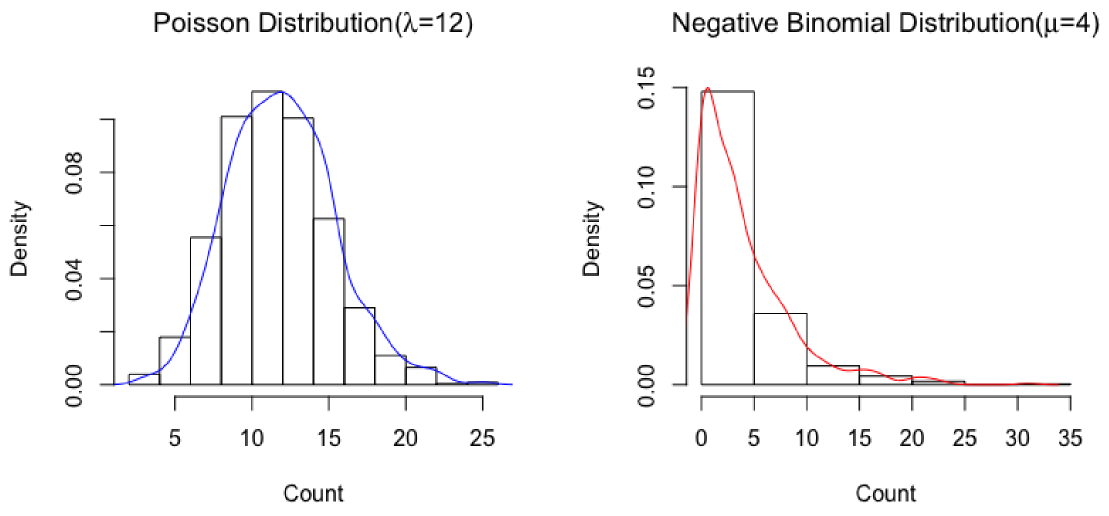


Figure 2.3: Poisson and Negative Binomial Distribution

2.2.1 Poisson Regression

The Poisson distribution is often used for modelling count data. The Poisson regression model specifies that each y_i is drawn from a Poisson population with parameter λ_i, where λ_i is related to the regressors x_i [30]. The primary equation of the model is:

$$Prob(Y = y_i|x_i) = \frac{e^{-\lambda_i}\lambda_i^{y_i}}{y_i!}, \quad \lambda_i > 0, \quad y_i \in \{0, 1, 2, \dots\}. \tag{2.4}$$

The most common formulation for λ_i is the log-linear model:

$$\ln\lambda_i = x_i^T\beta. \tag{2.5}$$

It is easily shown that the expected number of events per period is given by:

$$E[y_i|x_i] = \text{Var}[y_i|x_i] = \lambda_i = e^{x_i^T\beta}, \tag{2.6}$$

so we can obtain the derivative of x_i:

$$\frac{\partial E[y_i|x_i]}{\partial x_i} = \lambda_i\beta. \tag{2.7}$$

With the parameter estimates in hand, this vector can be computed using any data vector desired.

In principle, the Poisson model is simply a nonlinear regression [30]. But it is far easier to estimate the parameters with maximum likelihood techniques. The log-likelihood function is:

$$\ln L = \sum_{i=1}^{n}[-\lambda_i + y_i x_i^T\beta - \ln y_i!\,] \tag{2.8}$$

The likelihood equations are:

$$\frac{\partial \ln L}{\partial \beta} = \sum_{i=1}^{n}(y_i - \lambda_i)x_i = 0. \tag{2.9}$$

And the Hessian is:

$$\frac{\partial^2 \ln L}{\partial\beta\,\partial\beta^T} = -\sum_{i=1}^{n}\lambda_i x_i x_i^T. \tag{2.10}$$

Apparently the log likelihood function is concave as the Hessian is negative definite for all x and β so there is no explicit solution for β. However, the Newton-Raphson iterative method [30] can be used to estimate β and it usually converges rapidly. At the converging value for β, the quantity $[\sum_{i=1}^{n}\hat{\lambda}_i x_i x_i^T]^{-1}$ provides an estimator of the asymptotic covariance matrix for the parameter estimator. Given the estimates, the prediction for observation i is the expectation value $\hat{\lambda}_i = E(Y_i|x_i) = \exp(x_i^T\hat{\beta})$.

The standard univariate Poisson regression model makes two assumptions [84]. First, $p(y|\lambda)$ is the conditional probability function of y given λ, and it must hold that $\lambda > 0$. Second, observation pairs $(y_i, x_i), i = 1, \ldots, n$ are independently distributed [84].

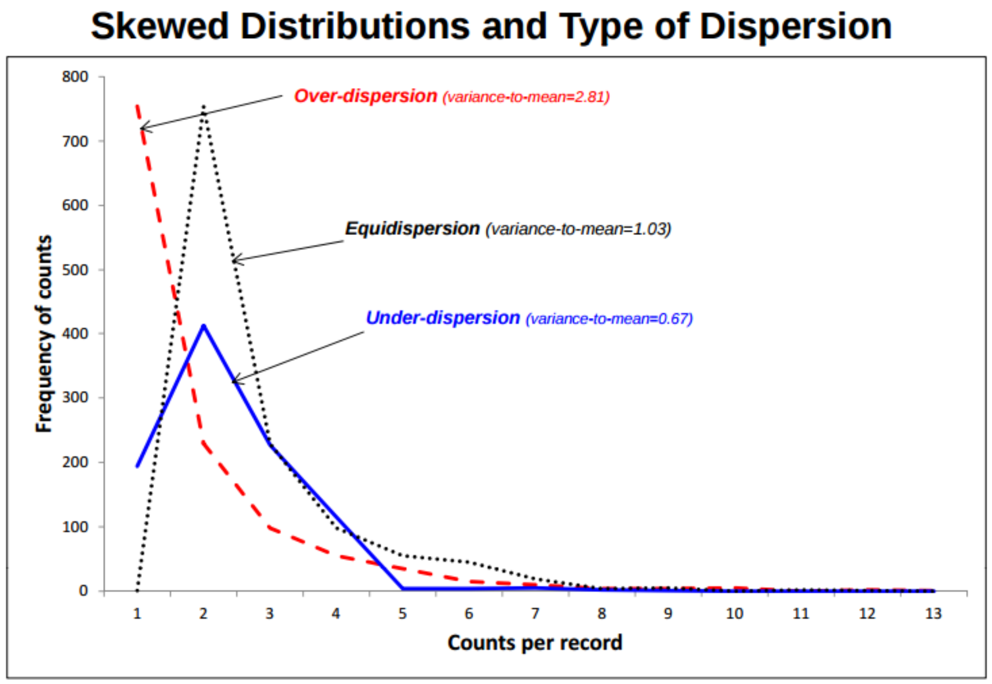


Figure 2.4: Poisson Regression dispersion [48]

2.2.2 Negative Binomial Regression

A Poisson random variable is *equidispersed*, i.e., the variance is equal to the mean. However, in many cases, the actual random variable is *overdispersed*, with variance greater than the mean, due to additional factors that are not accounted for by the input x or the model itself [11]. Poisson regression is ill-suited to model overdispersion because it will bias the mean towards the variance, in order to keep the equidispersion property. One popular regression model for overdispersed count data is based on the Negative Binomial [10].

Negative binomial regression is a type of generalized linear model in which the dependent variable Y is a count of the number of times an event occurs. A convenient parametrization of the negative binomial distribution is given by:

$$p(y) = p(Y = y) = \frac{\Gamma(y + 1/\alpha)}{\Gamma(y+1)\Gamma(1/\alpha)} \left(\frac{1}{1+\alpha\mu}\right)^{1/\alpha} \left(\frac{\alpha\mu}{1+\alpha\mu}\right)^{y}. \tag{2.11}$$

where $\mu > 0$ is the mean of Y and $\alpha > 0$ is the heterogeneity parameter.The traditional negative binomial regression model, designated the NB2 model in [36], is:

$$\ln \mu = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p. \tag{2.12}$$

where the predictor variables $x_1, x_2, \ldots, x_p$ are given, and the population regression coefficients $\beta_0, \beta_1, \beta_2, \ldots, \beta_p$ are to be estimated. Then the distribution formula can be rewritten as:

$$p(y_i) = \frac{\Gamma(y_i + 1/\alpha)}{\Gamma(y_i + 1)\Gamma(1/\alpha)} \left(\frac{1}{1 + \alpha e^{x_i \beta}}\right)^{1/\alpha} \left(\frac{\alpha e^{x_i \beta}}{1 + \alpha e^{x_i \beta}}\right)^{y_i}, \; i = 1, 2, \ldots, n. \tag{2.13}$$

Typically, likelihood maximization is employed to estimate α and β. The likelihood function is:

$$L(\alpha, \beta) = \prod_{i=1}^{n} p(y_i) = \prod_{i=1}^{n} \frac{\Gamma(y_i + 1/\alpha)}{\Gamma(y_i + 1)\Gamma(1/\alpha)} \left(\frac{1}{1 + \alpha e^{x_i \beta}}\right)^{1/\alpha} \left(\frac{\alpha e^{x_i \beta}}{1 + \alpha e^{x_i \beta}}\right)^{y_i}.$$

and the log-likelihood function is

$$\begin{aligned} \ln L(\alpha, \beta) = \sum_{i=1}^{n} \Bigg(& y_i \ln \alpha + y_i (x_i \cdot \beta) - \left(y_i + \frac{1}{\alpha}\right) \ln \left(1 + \alpha e^{x_i \cdot \beta}\right) + \ln \Gamma \left(y_i + \frac{1}{\alpha}\right) \\ & - \ln \Gamma(y_i + 1) - \ln \Gamma \left(\frac{1}{\alpha}\right) \Bigg). \end{aligned} \tag{2.14}$$

The pair (α, β) that maximizes $\ln L(\alpha, \beta)$ will be the maximum likelihood estimates, and the estimated variance-covariance matrix of the estimators is $\sum = -H^{-1}$,where H is the Hessian matrix of the log-likelihood function.

2.2.3 Goodness of Fit

A measure of discrepancy between observed and fitted values is the deviance [10]. For Generalized Linear Models (GLM) including Poisson regression and Negative Binomial regression, the deviance takes the form

$$D(y, \hat{\mu}) = 2\{L(y) - L(\hat{y})\} \tag{2.15}$$

which is twice the difference between the maximum log-likelihood achievable and the log-likelihood of the fitted model. For Poisson regression, it takes the form

$$D_{\text{poisson}} = \sum_{i=1}^{n} \{y_i \log(\frac{y_i}{\hat{y}_i}) - (y_i - \hat{y}_i)\}, \tag{2.16}$$

while for Negative Binomial regression, it takes the form

$$D_{\text{NB2}} = \sum_{i=1}^{n} \left\{ y_i \ln\left(\frac{y_i}{\hat{y}_i}\right) - (y_i + \alpha^{-1}) \ln\left[\ln \frac{y_i + \alpha^{-1}}{\hat{y}_i + \alpha^{-1}}\right] \right\}, \tag{2.17}$$

where $\hat{y}_i$ denotes the fitted values based on the current parameter estimates. The first term is identical to the binomial deviance, representing twice a sum of observed times log of observed fitted. The second term, a sum of differences between observed and fitted values, is usually zero, because the maximum likelihood estimates (MLE) have the property of reproducing marginal totals [67].

For large samples the distribution of the deviance is approximately a chi-squared with $n - p$ degrees of freedom, where n is the number of observations and p is the number of predictors (or parameters). Thus, the deviance can be used directly to test the goodness of fit of the model [67].

An alternative measure of goodness of fit is Pearson's chi-squared statistics, which is defined as

$$\chi_p^2 = \sum_{i=1}^{n} \frac{(y_i - \hat{y}_i)^2}{\hat{y}_i}. \tag{2.18}$$

The numerator is the squared difference between observed and fitted value, and the denominator is the expected (fitted) value. The Pearson statistic has the same form for Poisson and binomial data, namely a 'sum of squared observed minus expected over expected'.

2.3 Conventional Machine Learning Approaches

Machine learning is the study of data-driven methods capable of mimicking, understanding and aiding human and biological information processing tasks [4]. Machine learning methods aim at improving a predictive performance measure by repeated observation of experiences. They may capture hidden information from large databases [28]. The ensemble methods, such as Random

Forest (RF) algorithms [9] and boosting [23], are the most appealing alternatives to analyze count data, and they have been widely applied in GWAS for human diseases [24, 28, 29].

2.3.1 Random Forest Regression

Random Forest can be viewed as a machine learning ensemble algorithm (ensembles of decision trees) and was first proposed by Breiman [9]. It is non-parametric, robust to over-fitting and able to capture complex interaction structures in the data, which may alleviate the problems of analysing genome-wide data. This algorithm constructs many decision trees on bootstrapped samples of the data set, averaging each estimate to make final predictions. This strategy, called bootstrap aggregating (or "bagging") [8], reduces error prediction by a factor of the number of trees. Like decision trees, random forests handle categorical features, extend to the multiclass classification setting, do not require feature scaling, and are able to capture non-linearities and feature interactions.

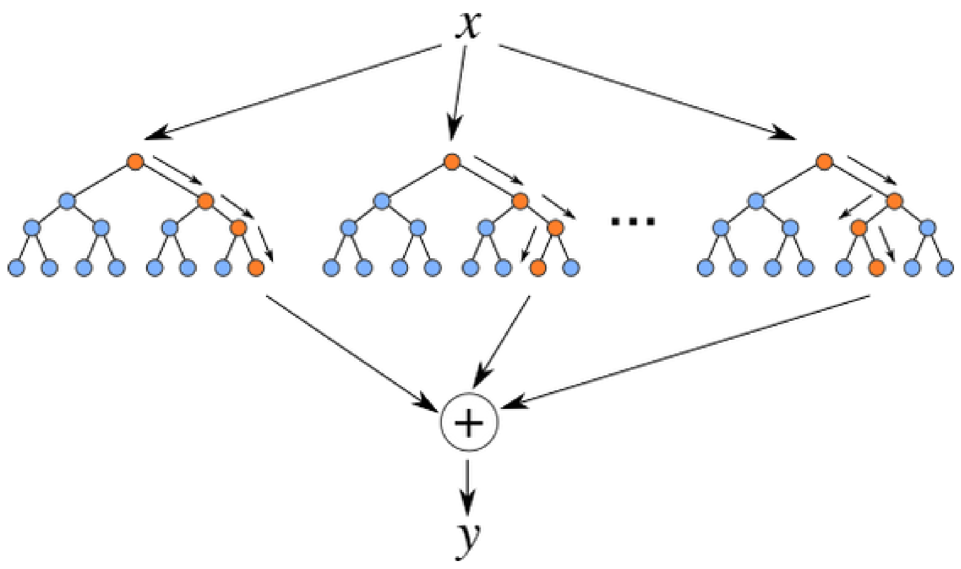


Figure 2.5: Random Forest Regression Models [33]. As the figures shows, the original input X is bootstrapped sampled to build different decision trees. And in the last step to predict $\hat{y}$ using the weighted averaging strategy.

Let $y_{n\times 1}$ be the data vector consisting of discrete observations for the outcome of a given event, i.e., noncoding mutations in this thesis, and $X = \{x_i\}$ where x_i is a $p \times 1$ vector representing p features (predictors), to which T decision trees are built. The ensemble can be described as an additive expansion of the form:

$$y = \mu + \sum_{t=1}^{T} c_t h_t(y; X). \tag{2.19}$$

Each tree $h_t(y; X)$ for $t \in (1, T)$ is distinct from any other in the ensemble as it is constructed from n samples from the original data set selected at random with replacement, and at each ode only a small group of noncoding mutations are randomly selected to create the splitting rule. Each tree is grown to be the largest extent possible until all the terminal nodes are maximally homogeneous. Then, c_t is some shrinkage factor averaging the trees. The trees are independent identically distributed random vectors, each of them casting a unit vote for the most popular outcome of the disease at a given combination of observed mutations.

The random forest regression algorithm is shown in Algorithm 1 (adapted from [40]).

Algorithm 1 RANDOM FOREST REGRESSION

```
input  : All observations $X_{n \times p}$ and $Y_{n \times 1}$ where $n$ is training size and $p$ is input dimension; $B$ is
         the number of tree to build.
output : Predicted values $\hat{Y}_{n \times 1}$ and feature importance scores $I_{p \times 1}$
for $b \leftarrow 1$ to $B$ do
    Draw a bootstrap sample $Z^\star$ of size $N$ from the training data.
    Grow a random-forest tree $T_b$ to the bootstrapped data, by recursively repeating the following
    steps for each terminal node of the tree, until the minimum node size $n_{\min}$ is reached.
        1. Select $m$ variables at random from the $p$ variables.
        2. Pick the best variable/split-join among the $m$.
        3. Split the node into two daughter nodes.
end
Output the ensemble of trees $\{T_b\}_1^B$
Prediction of $x$:$\hat{f}_{\text{rf}}^B(x) = \frac{1}{B} \sum_{b=1}^{B} T_b(x)$.
```

2.3.2 Gradient Boosted Regression Trees

Boosting is one of several classic methods for creating ensemble models, along with bagging and random forests which are based on model averaging. However, boosting is sequential: it is a forward, stage-wise procedure. Gradient boosting [23] is a machine learning technique for regression problems. It builds each regression tree in a step-wise fashion, using a predefined loss function (i.e., mean squared error) to measure the error in each step and correct for it in the next. The predictor is an ensemble of weak prediction models, typically decision trees.

Gradient Tree Boosting or Gradient Boosted Regression Trees (GBRT) is a generalization of boosting to arbitrary differentiable loss functions. GBRT is an accurate and effective procedure that can be used for both regression and classification problems. Gradient Tree Boosting models are used in a variety of areas including Web search and ecology [60]. The generic gradient boosting algorithm is described in Algorithm 2 (derived from [23, 32]). The Gradient Tree

Algorithm 2 GRADIENT BOOSTING METHOD

```
input : Training set $\{(x_i, y_i)\}_{i=1}^n$, a differentiable loss function $L(y, F(x))$, and also number of
        iterations $M$ for training to converge.
Initialize model with a constant value:
Let $F$ be the learning model, $L$ be the loss function, and $\gamma$ be the weighting factor.
    $F_0(x) = \underset{x}{\text{argmin}} \sum_{i=1}^n L(y_i, \gamma)$,
for $m \leftarrow 1$ to $M$ do
    1. Compute so-called pseudo-residuals:
    $r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}$   for $i = 1, ..., n$.
    2. Fit a base learner $h_m(x)$ to pseudo-residuals, for example, train it using the training set
    $\{(x_i, r_{im})\}_{i=1}^n$
    3. Compute multiplier $\gamma_m$ by solving the following one-dimensional optimization problem:
    $\gamma_m = \underset{\gamma}{\text{argmin}} \sum_{i=1}^n L(y_i, F_{m-1}(x_i) + \gamma h_m(x_i))$.
    4. Update the model:
    $F_m(x) = F_{m-1}(x) + \gamma_m h_m(x)$
end
output: $F_M(x)$
```

Boosting algorithm fits a decision tree $h_m(x)$ to pseudo-residuals. Let J be the number of the tree's leaves. The tree partitions the feature vector space into J disjoint regions $R_{1m}, \ldots, R_{Jm}$ and predicts a constant value in each region:

$$h_m(x) = \sum_{j=1}^{J} b_{jm} I(x \in R_{jm}), \tag{2.20}$$

where I is an indicator function, b_{jm} is the value predicted in the region R_{jm} [32]. And each tree's region has a separate optimal value γ_{jm}, so the update rule would be rewritten as:

$$F_m(x) = F_{m-1}(x) + \sum_{j=1}^{J} \gamma_{jm} h_m(x) I(x \in R_{jm}), \tag{2.21}$$

$$\gamma_{jm} = \underset{\gamma}{\text{argmin}} \sum_{x_i \in R_{jm}} L(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)). \tag{2.22}$$

There are some advantages for the gradient boosting tree regression model [60]. It can handle heterogeneous features (mixed data type) and has strong predictive power. Meanwhile it is robust to outliers in output space via robust loss functions. One minor disadvantage is that since the boosting method is inherently sequential, it is not conducive to parallelization.

2.3.3 Ada Boosting Regression

Another popular boosting technique is AdaBoost [70] which can be also used for both classification and regression. AdaBoost is one of the best known boosting methods for classification [59]. In AdaBoost, each training instance receives a weight w_i that is used when learning each hypothesis; this weight indicates the relative importance of each instance and is used in computing the error of a hypothesis on the data set. After each iteration, instances are reweighted, with those instances that are not correctly classified by the last hypothesis receiving larger weights. Thus, as the process continues, learning focuses on those instances that are most difficult to classify.

A number of methods have been proposed for modifying AdaBoost for regression, and one of the methods that have been shown to be generally effective is `AdaBoost.R2` [20] which is also implemented in the function interface `sklearn.ensemble.AdaBoostRegressor` of the popular machine learning toolkit `scikit-learn` [60]. The key to AdaBoost is the reweighting of those instances that are misclassified at each iteration. In regression problems, the output given by a hypothesis h_t for an instance x_i is not correct, but has a real-valued error

$e_i = |y_i - h_t(x_i)|$ that may be arbitrary large. Thus, a method of mapping an error e_i into an adjusted error $e_i^{'}$ is needed in the reweighting formula used by AdaBoost. The `AdaBoost.R2` algorithm is described in Algorithm 3 (adapted from the description in Drucker *et al.* [20]). Compared to gradient boosting which accounts for "shortcomings" by gradients, in AdaBoost,

Algorithm 3 ADABOOST.R2 [20]

```
input  : the labeled target data set $T$ of size $n$, the maximum number of iterations $N$, and a base
         learning algorithm $Learner$. Unless otherwise specified, set the initial weight vector
         $w^1$ such that $w_i^1 = 1/n$ for $1 \leq i \leq n$.
for $t \leftarrow 1$ to $N$ do
    1. Call $Learner$ with the training set $T$ and the distribution $w^t$, and get a hypothesis
       $h_t : X \to \mathbb{R}$.
    2. Calculate the adjusted error $e_i^t$ for each instance:
        (a) let $D_t = \max_{j=1}^{n} |y_i - h_t(x_j)|$
        (b) then $e_i^t = |y_i - h_t(x_i)|/D_t$
    3. Calculate the adjusted error of $h_t$:
        (a) $\epsilon_t = \sum_{i=1}^{n} e_i^t w_i^t$; if $\epsilon_t \geq 0.5$, stop and set $N = t - 1$.
    4. Let $\beta_t = \epsilon_t/(1 - \epsilon_t)$
    5. Update the weight vector:
        (a) $w_i^{t+1} = w_i^t \beta_t^{1-e_i^t}/Z_t$ ($Z_t$ is a normalizing constant)
end
output: the hypothesis:
$h_f(x)$ = the weighted median of $h_t(x)$ for $1 \leq t \leq N$, using $\ln(1/\beta_t)$ as the weight for
hypothesis $h_t$.
```

"shortcomings" are identified by high-weight data points [49]. Since both of the two methods are sequential, and also AdaBoost can use the decision tree as its base learner, all of the advantages and disadvantages of Gradient Boosting Regression Tree also apply to AdaBoost regression.

2.4 Deep Learning Approach

Conventional machine learning algorithms have limitations in processing feature data in raw form, so researchers typically transform the raw form into suitable high-abstraction level features with considerable domain expertise [45]. On the other hand, deep learning, a new class of machine learning algorithms, has emerged recently on the basis of increasing training data set sizes (Big Data), the power of parallel and distributed computing, and breakthroughs in algorithmic efficiency [54]. Deep learning algorithms have overcome the former limitations and are making major advances in diverse fields such as image recognition, speech recognition, and natural language processing. Recently deep learning has been emerged in bioinformatics areas, such as gene expression profiling [19, 21], protein structure prediction [34, 72, 73], and biomedical image processing in cancer [37, 62]. Closer to the study described in this thesis, in recent work, Zhou *et al.* [87] described a convolutional neural network model for predicting noncoding variant effects *de novo* from sequence using. However, their analysis is just based on noncoding population genetic variants in human disease, instead of somatic variants in cancer.

A hallmark of deep learning algorithms is the use of an artificial neural network of multiple nonlinear layers which is usually called as base learner. The key aspect of deep learning is that predictive features are not human-engineered but instead are learned from the data by the algorithm [54]. As one of the representation learning methods, deep learning can learn and discover hierarchical representations of data with increasing level of abstraction [45]. Although currently the majority research of deep learning is about classification problems, applying deep learning techniques in regression problems is still promising if appropriate activation and loss functions are used. The following sections describe the base learner (neural network), and the deep neural network (stacked auto encoder) for regression analysis.

2.4.1 Neural Network Regression

Although neural networks are widely known for use in deep learning and modelling complex problems such as image recognition, they are easily adapted to regression problems. Any class of statistical models can be termed a neural network if they use adaptive weights and can approximate non-linear functions of their input [53]. Thus neural network regression is suited to problems where a more traditional regression model cannot fit a solution.

A feed-forward neural network is a non-parametric statistical model for extracting nonlinear relations in the data [76]. A common neural network model configuration is to place between the input and output variables (also called 'neurons'), a layer of 'hidden neurons' as shown in Figure 2.6. The value of the j_{th} hidden neuron is:

$$y_j = \tanh(\sum_i w_{ij}x_i + b_j), \tag{2.23}$$

where x_i is the i_th input, w_{ij} is the weight parameters and b_j is the bias parameters, and

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \tag{2.24}$$

The output neuron is given by

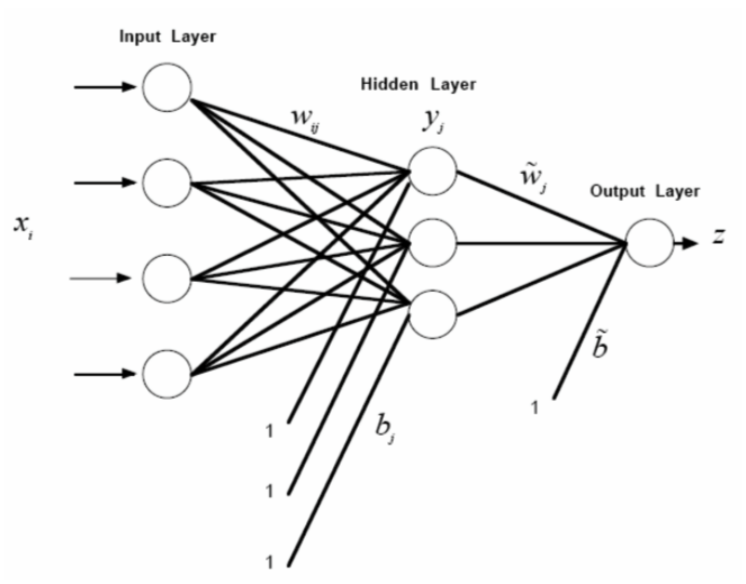


Figure 2.6: Feed-Forward Neural Network Model [76]. In the forward phase, from the input layer to the output layer, the activation values of the hidden nodes and the output are calculated, and also the gradients. In the backward phase, from the output layer to the input layer, the weights are updated layer by layer using a backpropagation algorithm.

$$z = \sum_j \tilde{w_j} y_j + \tilde{b} \tag{2.25}$$

and a cost function

$$J = \frac{1}{n}\sum_{i=1}^{n}(z - z_{obs})^2 \tag{2.26}$$

measures the mean squared error between the model output z and the observed values z_{obs}. The parameters w_{ij}, $\tilde{w_j}$, b_j and $\tilde{b}$ are adjusted as the cost function is minimized using a backpropagation algorithm from layer to layer backwards based on gradient descent optimization.

2.4.2 Stacked AutoEncoder Regression

Unsupervised feature learning methods and deep learning have been widely used for image and audio applications [38, 47]. In these domains, these techniques have shown strong promise in automatically representing the feature space using unlabelled data in order to increase the accuracy of subsequent classification or regression tasks. Using additional properties of the data, these capabilities have been further extended to facilitate learning in very high dimensional feature spaces. For example, by using image characteristics such as locality and stationary of images, Lee [46] proposed a method to scale the unsupervised feature learning and deep learning methods to high dimensional and full-sized images. Similarly Le [38] applied an supervised feature learning method in the context of cancer detection by applying it in the classification of histological image signature and classification of tumour architecture. Recently, deep learning techniques have also been applied in the area of bioinformatics such as in the work of Fakoor [21], who used deep learning to enhance cancer diagnosis and classification by applying a sparse autoencoder method to learn a concise feature representation from unlabelled data.

The autoencoder neural network is an unsupervised feature learning method in which the input is used as the target for the output layer [64]. In this way it learns a function $h_{w,b}(x) \approx x$ that represents an approximation of the input data constructed from a limited number of feature activations represented by the hidden units of the network. The sparse autoencoder is constructed by three layers in the neural network (i.e., input layer, hidden layer, and output layer) in which the hidden layer contains K nodes. The units in the hidden layer force the network to learn a representation of the input with only K hidden unit activations, representing K features. To train the network it uses the backpropagation method to minimize the squared reconstruction

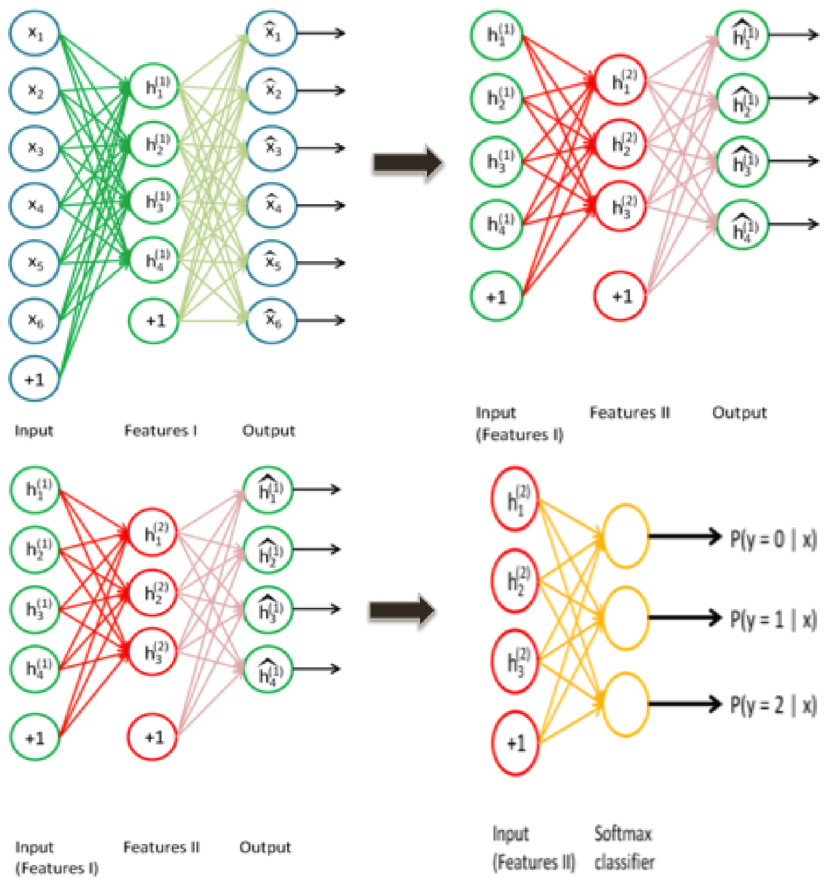


Figure 2.7: Stacked Auto Encoder Pretraining [57]. In the example above, train the first hidden layer $h^{(1)}$ (encoder) and drop the output, then use the $h^{(1)}$ as input to train the second encoder $h^{(2)}$ and also drop the output. And in the last train the weights from the second hidden layer to the output layer. All the activation values and weights were memorized and re-updated.

error with an additional sparsity penalty [16, 64]

$$\min_{b,a} \sum_{i}^{m} \|x_u^{(i)} - \sum_{j}^{K} a_j^{(i)} b_j\|_2^2 + \beta \|a^{(i)}\|_1 \tag{2.27}$$

$$\text{s.t.} \quad \|b_j\|_2 \leq 1, \ \forall j \in \{1, \ldots, s\},$$

where $x_u^{(i)}$ is unlabeled training example, b is basis vector, and a is the vector of activations of

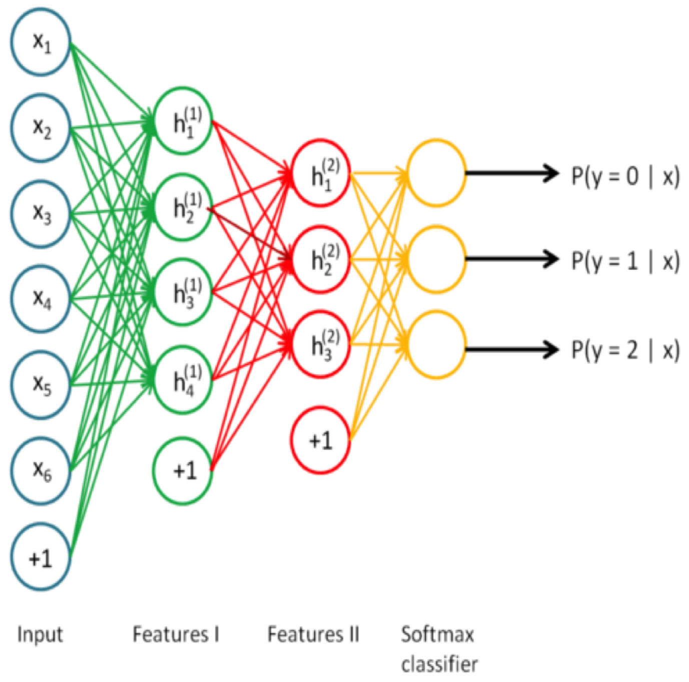


Figure 2.8: Stacked Auto Encoder Fine-tuning [57]. Put the input layer X, the two pretrained encoders $h^{(1)}, h^{(2)}$ and the output layer all together, and also reuse the learned weights to fit the input data again. Use the backpropagation based on gradient descent to update the weights.

the basis [64]. The sparsity penalty included in the form of the L^1-norm of the activation vector, a, here biases the learner towards features, b_j, that allow the data items to be represented using a combination of a small number of these features.

As Figure 2.7 and Figure 2.8 show, there are two phases, one is pretraining which is a supervised learning for feature learning, and the other is the fine-tuning phase. In the pretraining phase, the weights from the input to the first encoder, the weights from the first encoder to the second encoder, and the weights from the second encoder to the output layer are trained one by one. Additionally, during the training, the weights are saved, but the decoder layers and the output are dropped. During the fine-tuning phase, the input layer, the two encoders, and the softmax output layer are all put together, using backpropagation to update the weights. Although the stacked auto encoder (SAE) is popular for classification problems, just as in the case of the neural network, the SAE can be easily modified to be used for regression problems.

2.5 Cross-Validation

Cross-validation is a general method for obtaining unbiased performance estimates for both unsupervised learning and supervised learning methods and for tuning parameters or models for optimal performance. A standard version of cross-validation is k-fold cross-validation, which works this way: the data (i.e., the samples, sometimes called cases) are partitioned into k random subsets of samples, and then the algorithm of interest (learning machine, regression model) is generated or trained on $k - 1$ of the subsets and applied or tested using the samples in the remaining subset. This is done repeatedly over all the k possible arrangements of the subsets into these two groups. At each iteration the measure of model performance (e.g., prediction error for supervised learning and distance matrix for unsupervised clustering) of the algorithm in the test set is computed, leading to k estimates. The final estimate of model performance is the average of these k performance estimates. Assuming that samples are independent, in cross-validation, the algorithm is in every iteration being trained and tested on statistically independent subsets of samples [18].

Chapter 3: Materials and Methods

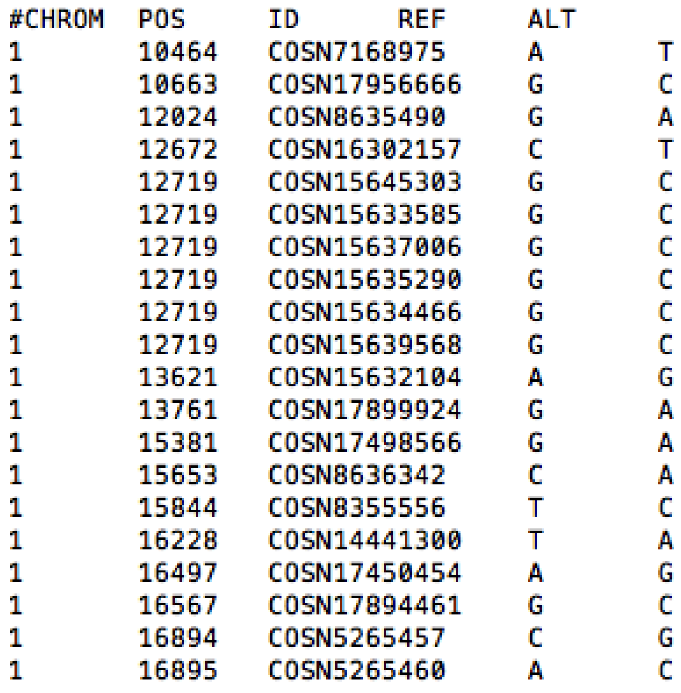
As we discussed in the introduction section 1.2, whole-genome sequencing is becoming less expensive so whole cancer genome analysis is expected to become more feasible and affordable in the near future. This is expected to lead to improvements in cancer diagnosis and targeted (i.e., more precisely tailored to the cancer's unique molecular genetic signature) therapy. With cancer genome sequences in hand, data-driven approaches such as applications of statistics, machine learning and deep learning can be used to uncover new molecular and genetic mechanisms as well as more precisely subtype cancers. This thesis is focused on data-driven approaches, specifically, on the functional analysis of genome somatic variants in cancer (also broadly called as mutations) compared to the genome sequences from normal (non-cancerous) cells from the same individual.

In general, there are two categories of mutations, protein-coding mutations (simply called coding mutations) and non-protein-coding mutations (simply called as noncoding mutations). Coding mutations have been studied for decades and bioinformatic methods for functionally characterizing coding somatic mutations are relatively mature, such as detection of mutations of oncogenes and tumour-suppressor genes, gene expression analysis (up-regulated or down-regulated), and corresponding up-stream and down-stream analysis. However, it is well known that the majority of mutations are passenger mutations which has no impact on tumour cell growth, cancer development and progression, and only very few of them are driver mutations which have real impact. Additionally, mutations are heterogeneous for different individuals and cancers. Given these difficulties, distinguishing the driver mutations from passenger mutations is challenging. Meanwhile, the recent finding that *TERT* promoter mutation (noncoding mutation) appears in multiple cancer types, has spurred interest in identifying driver mutations in noncoding genome regions, which together constitute than 90% of the human genome. Multiple studies have confirmed that the *TERT* promoter is frequently mutated in human cancer samples and also located in the hotspot (one genome segment which has most number of mutations) [52, 75, 78, 82]. Therefore, in this thesis we explore regression models for predicting noncoding mutation recurrence so that the noncoding mutations with highest predicted recurrence can be chosen as candidate "drivers" for further analysis.

In this chapter, first we introduce the noncoding mutation dataset, data processing (i.e., annotation of noncoding mutations and feature extraction). Secondly, we describe how we quantified the recurrence (the response or dependent variable) of a noncoding mutation within its chromosomal location, the features that we used as predictors (the independent variables), and appropriate regression models for noncoding mutation recurrence count data. Lastly, we describe the model evaluation methods that can be used and calculated in all regression models so that we can do further comparative performance analysis.

3.1 Datasets and Feature Selection

We downloaded the latest dataset of noncoding mutations from the COSMIC [3] database. This dataset contains $\sim$10.1 million mutation positions in human genome GRCh37 (also known by its UC Santa Cruz Genome Build identifier, hg19). And each mutation entry provides the chromosome ID, the genome assembly coordinate in that chromosome, the COSMIC mutation ID, the reference nucleotide and the altered (mutated) nucleotide as Figure 3.1 shows: For each muta-



```
#CHROM  POS     ID      REF     ALT
1       10464   COSN7168975     A       T
1       10663   COSN17956666    G       C
1       12024   COSN8635490     G       A
1       12672   COSN16302157    C       T
1       12719   COSN15645303    G       C
1       12719   COSN15633585    G       C
1       12719   COSN15637006    G       C
1       12719   COSN15635290    G       C
1       12719   COSN15634466    G       C
1       12719   COSN15639568    G       C
1       13621   COSN15632104    A       G
1       13761   COSN17899924    G       A
1       15381   COSN17498566    G       A
1       15653   COSN8636342     C       A
1       15844   COSN8355556     T       C
1       16228   COSN14441300    T       A
1       16497   COSN17450454    A       G
1       16567   COSN17894461    G       C
1       16894   COSN5265457     C       G
1       16895   COSN5265460     A       C
```

Figure 3.1: Example rows from noncoding mutation dataset from the COSMIC database

tion, we have to extract biological features (predictors) for regression analysis. With the existing findings in literature which is also discussed in the sections above, the noncoding "driver" mutations tend to more likely appear in regulatory regions which contain abundant transcription factor binding sites, and also they are close to the gene transcription start sites. Therefore from biology perspective, we extracted 29 following features as shown in Table 3.1.

Table 3.1: *Twenty-nine* noncoding mutation features for regression

Feature Names	Data Type	Description
tfbs_cnt, tfbs_max/avg_sc	Integer	Number,Max/Avg score of overall TFBS
atf3_cnt, atf3_max/avg_sc	Integer	Number,Max/Avg score of *ATF3* TFBS
cebpb_cnt, cebpb_max/avg_sc	Integer	Number,Max/Avg score of *CEBPB* TFBS
cebpd_cnt, cebpd_max/avg_sc	Integer	Number,Max/Avg score of *CEBPD* TFBS
creb1_cnt, creb1_max/avg_sc	Integer	Number,Max/Avg score of *CREB1* TFBS
egr1_cnt, egr1_max/avg_sc	Integer	Number,Max/Avg score of *EGR1* TFBS
ets1_cnt, ets1_max/avg_sc	Integer	Number,Max/Avg score of *ETS1* TFBS
maff_cnt, maff_max/avg_ sc	Integer	Number,Max/Avg score of *MAFF* TFBS
dhs_src_cnt	Integer	Number of DNase I Hypersensitive Sites
dhs_max_ sc	Integer	Max score of DHS sites
gerp_sc	Float	Genomic evolutionary rate profiling score
tss_dist	Integer	Distance to nearest transc. start sites
gc_per	Float	The percentage of GC content

As we can see from Table 3.1, most of the features are integer-valued and two of them are floating-point-valued. All of the raw feature data including transcription factor binding sites (TFBS), transcription start sites (TSS), DNase I hypersensitive sites (DHS), and GERP (Genome Evolutionary Rate Profiling) score were in BED [41] format and downloaded from UCSC Genome Browser [68, 69]. When extracting features related to the bindings sites, given a noncoding mutation (chr_i, p_j) where chr_i is chromosome ID and p_j is the genome coordinate of the mutation, and the TF binding sites $\{(chr_i, tsstart_k, tsend_k)\}_{k=1,2,\ldots,m}$ where $tsstart_k, tsend_k$ represent TSS starting and ending coordinate for TSS site k, and k represents the total number of transcription factor binding sites in chromosome i. We then count the matchings:

$$tfbs\_cnt = \sum_{k=1}^{m} I(tsstart_k \leq p_j \leq tsend_k), \tag{3.1}$$

where I is the indicator function. Similarly, we can calculate the maximum peak value max_sc and average peak value avg_sc. We can then apply the same strategy to the specific transcription factors *ATF3, CEBPB, CEBPD, CREB1, EGR1, ETS1* and *MAFF* including the *DHS* features as well. The details are illustrated in Algorithm 4.

As discussed earlier, all other features of counts and maximum/average scores can be calculated using the same or similar algorithm as Algorithm 4. However, for TSS distance, calculating the closest distance from the mutation point to the nearest TSS including both the downstream and upstream directions, it is a little more complicated since the same gene may have multiple different transcripts, different transcripts may have the same start sites and gene transcriptions have two directions.

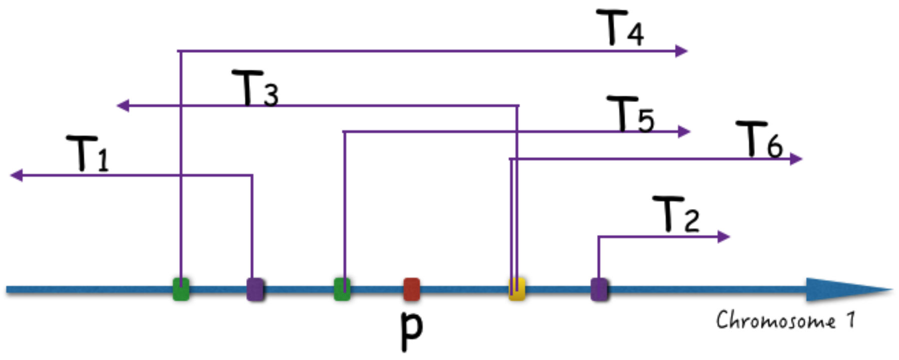


Figure 3.2: Noncoding mutation and TSS distance

For example, in Figure 3.2, p is one noncoding mutation point in chromosome 1, and $T_1, T_2, \ldots, T_6$ are transcription start sites, among which T_1, T_3 have the negative direction, and T_2, T_4, T_5, T_6 have an opposite (positive) direction. And the mutation point p is in the upstream of T_1, T_2, T_6 and in the downstream of T_3, T_4, T_5. Also one exception is that T_3 and T_6 use the same start sites but have different directions. At this point, we prefer the upstream so T_6 would be chosen. Now given the two closest transcription start sites T_i and T_j for the mutation point p, with the condition $cord(T_i) \leq p \leq cord(T_j)$ where $cord$ represents the coordinate of the TSS, we can

Algorithm 4 TF BINDINGS SITES FEATURES EXTRACTION

```
input : List of noncoding mutations ${\{(chr_i, p_j)\}_{j=1,2,...,n}}$ and list of TF binding sites
        ${\{(chr_i, tsstart_k, tsend_k, score_k)\}_{k=1,2,...,m}}$
for $i \in \{1, 2, \ldots, 22, X, Y\}$ do
    Sort the mutation list in chromosome $i$ by $p_j$ in ascending order
    Sort the TFBS list in chromosome $i$ by $tsstart_k$ in ascending order
    $k \leftarrow 1$
    $t \leftarrow 1$
    for $j \leftarrow 1$ to $n$ do
        $tfs\_cnt \leftarrow 0$
        $tfs\_max\_score \leftarrow 0$
        $tfs\_avg\_score \leftarrow 0$
        $tfslist \leftarrow \{\}$
        while $k \leq m$ and $p_j > tsend_k$ do
            $k \leftarrow k + 1$
        end
        if $k > m$ or $(k \leq m$ and $p_j < tsstart_k)$ then
            continue
        end
        // Memorize the current start searching position
        $t \leftarrow k$
        while $t \leq m$ and $p_j \geq tsstart_t$ do
            if $p_j \leq tsend_t$ then
                $tfslist.push\_back((tsstart_t, tssend_t, score_t))$
            end
            $t \leftarrow t + 1$
        end
        $tfs\_cnt \leftarrow length(tfslist)$
        $tfs\_max\_score \leftarrow max\{tf.score \mid tf \in tfslist\}$
        $tfs\_avg\_score \leftarrow avg\{tf.score \mid tf \in tfslist\}$
    end
end
output: List of feature values for each mutation: $\{(p_j, tfs\_cnt, tfs\_max\_score, tfs\_avg\_score)\}$
Running Time: $O(n \log n + m \log m)$
```

define the TSS distance to the mutation point as:

$$dist(p, T_i) = \begin{cases} (p - cord(T_i)) * dir(T_i) & \text{If all TSS have the same direction,} \\ -\left|p - cord(T_i)\right| & \text{Otherwise} \end{cases} \tag{3.2}$$

$$TSSDist(p) = \min\{\left|dist(p, T_i)\right|, \left|dist(p, T_j)\right|\} * sign(p), \tag{3.3}$$

where dir indicates the transcription direction, and $sign$ indicates the direction of TSS which has smaller absolute distance. As shown in Figure 3.2, for mutation p, the calculated TSS distance value which would be used for regression analysis is $dist(p, T_5) = (p - cord(T_5))$. The same strategy as Algorithm 4 can be used to annotate all the mutations with the TSS distance feature and the running time would be $O(n \log n + m \log m)$ where m is the total number of transcription start sites across the whole genome.

For the GERP score feature, we use the existing tool `bigWigAverageOverBed` [41] to extract the value of $gerp\_sc$ for all the input noncoding mutations. If there is no matching found for the mutation, then $gerp\_sc$ would be assigned to zero.

While for the feature gc_per which represents the percent of G/C-content in one fixed window, that is defined as

$$gc\_per = \frac{G + C}{A + T + G + C}, \tag{3.4}$$

where A, T, G, C represents the count number of the nucleotides A,T,G,C respectively.

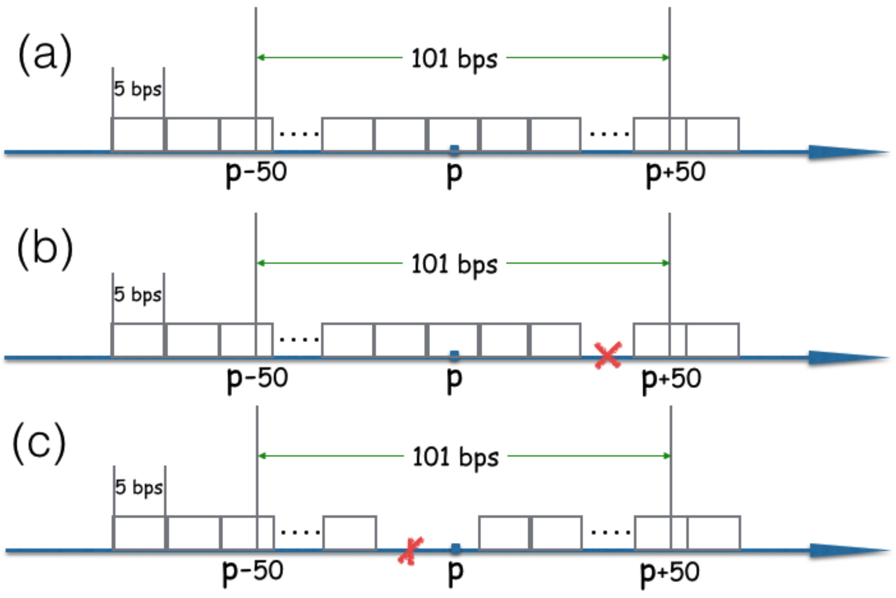


Figure 3.3: Noncoding mutation and GC-content

However, the GC-ratio that we can get from UCSC genome browser in only based on 5 base-pairs (bp) window. For each mutation that is a single nucleotide, it is meaningless to calculate the GC-content of one mutation point. Instead, to be in line with our counting model which is discussed in the section below, we compute the GC-content of a 101 bp window, including 50 bp upstream and 50 bp downstream of the mutation position. And also the downloaded data is incomplete, for some mutations, they are not contained in any 5 bp window or in 101 bp window, for some segments, the information of GC-content is missing. As Figure 3.3 shows, the scenario (a) is the ideal case where the mutation position is located in continuous 5 bp windows while for case (b) and (c) some reference GC-content information is missing. Therefore, we use the approximated GC-content value for the 101 bp window by averaging. For instance, given windows $W = \{w_i : (start, end)\}_{i=1,2,...,n}$ which are intersected or contained by the 101 bp window that are centered at the mutation point p, and $\{gc_i\}_{i=1,2,...,n}$ the GC-content of each 5 bp window, then the approximated GC-content value gc_per is

$$size\left[i\right] = \begin{cases} w_i.end - p + 50 + 1 & i = 1, w_i.start < p - 50 < w_i.end \\ p + 50 - w_i.start + 1 & i = n, w_i.start < p + 50 < w_i.end \\ 5 & otherwise \end{cases} \tag{3.5}$$

$$gc\_per = \frac{\sum_{i=1}^{n} size\left[i\right] * gc_i}{\sum_{i=1}^{n} size\left[i\right]}, \tag{3.6}$$

where $size\left[i\right]$ represents the size of window i; $w_i.start, w_i.end$ represents the start coordinate and the end coordinate respectively, $w_i.end = w_i.start + 4$.

3.2 Annotation of Noncoding Mutations

In general, noncoding regions include promoters, enhancers, 5'-UTRs, 3'-UTRs, introns and intergenic regions as shown in Figure 1.3. For simplicity, in this thesis we only consider the two regions only: 5'-UTR and the intergenic regions which contain most of the regulatory regions. Intergenic region is the DNA segment between two adjacent genes as Figure 3.4 shows.

Therefore we need to filter out those non-relevant mutation points from the noncoding mutation dataset downloaded from UCSC genome browser.

Algorithm 5 NONCODING MUTATIONS ANNOTATION

```
input : List of noncoding mutations ${\{(chr_i, p_j)\}_{j=1,2,...,n}}$ and list of hg19 transcripts:
        ${\{(chr_i, tsstart_k, tsend_k, tscoding_k, 5start_k, 5end_k, 3start_k, 3end_k)\}_{k\in[1,m]}}$

for $i \in \{1, 2, \ldots, 22, X, Y\}$ do
    $tsmap \leftarrow$ build map with key $(tsstart, tsend)$
    for $j \leftarrow 1$ to $n$ do
        $found \leftarrow 0$ ; $label_j \leftarrow null$
        for $u \leftarrow 1$ to $length(tsmap.keys)$ do
            $tsslist \leftarrow tsmap[(tsstart_u, tsend_u)]$
            for $k \leftarrow 1$ to $length(tsslist)$ do
                if $p_j \geq tsstart_k$ and $p_j \leq tsend_k$ then
                    $found \leftarrow 1$
                    if $tscoding_k = true$ then
                        if $5start_k > 0$ and $p_j \geq 5start_k$ and $p_j \leq 5end_k$ then
                            $label_j \leftarrow$ "5utr" ; break
                        else if $3start_k > 0$ and $p_j \geq 3start_k$ and $p_j \leq 3end_k$ then
                            $label_j \leftarrow$ "3utr" ; break
                        else
                            $label_j \leftarrow$ "intron"; continue
                        end
                    else
                        if $length(label) = 0$ then
                            $label_j \leftarrow$ "utr"
                        end
                    end
                end
            end
        end
        if $found = 0$ then
            $label_j \leftarrow$ "intergenic"
        end
    end
end
output: List of labelled mutations: $\{(p_j, label_j)\}$
```

Running Time: $O(nm)$ where m is total number of non-overlapping transcripts

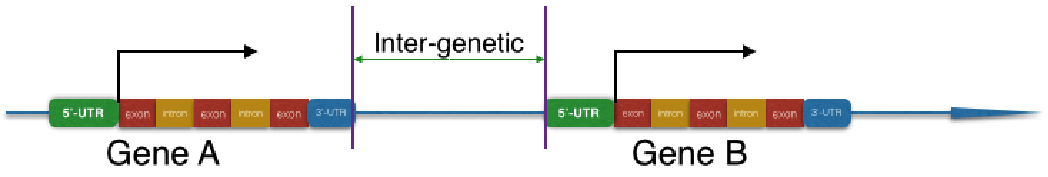


Figure 3.4: Noncoding intergenic regions

The basic idea is to refer to human hg19 transcriptome data which contains detailed gene transcripts information including the coordinates range of 5'-UTR region, 3'-UTR region and the CDS (Coding DNA Sequence) region. For each mutation, compare its coordinate to the transcripts regions $(tsstart, tsend)$, if no matching with any transcription regions is found, then the mutation is inter-genetic, otherwise further compare to the coordinate range of 5'-UTR regions, if one matching (the mutation position is located in the region) is found, then mark the mutation as 5'-UTR mutation. And then the same procedure for 3'-UTR regions. Nevertheless, there are some exceptions since the transcripts region may overlap. First the mutation point may be located in both protein-coding transcripts and non-protein-coding transcripts, in this case, we prefer the protein-coding transcripts first. Second, the mutation may appear in different regions of different protein-coding transcripts but we prefer the UTR (untranslated) region, that is as long as the mutation is located in the UTR region of one coding transcript then we annotate it as UTR mutation irrespective wherever the mutation is located in other transcripts. All the details are illustrated in Algorithm 5.

3.3 Regression Analysis

In this section, we describe our regression models with the extracted features of noncoding mutations that were discussed in the sections above. In the beginning we give the definition of noncoding mutation recurrence and the counting model to get the recurrence values, and then give the algorithm of exploring different computational regression models, which are mainly in three categories: generalized linear model (also known as log-linear model), ensemble of trees, and deep neural network. And then we elucidate the state-of-the-art model evaluation metrics that could be calculated in all regression models so that we can do comparative performance analysis although different models have different loss functions and optimization strategies.

3.3.1 Definition of Mutation Recurrence

In light of existing findings in the literature about noncoding mutations and driver mutations detection, especially the latest paper [82], it can be concluded with high confidence that the noncoding mutations that are located in the hotspot of regulatory region, close to the transcription start sites and highly frequent in tumor samples, are more likely to be the drivers. Therefore for the purpose of regression analysis, we define the noncoding mutation recurrence based on the strategy of 101 bp sliding window that is centered at the mutation point.

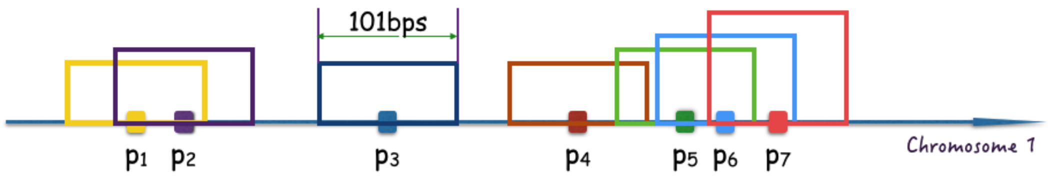


Figure 3.5: Noncoding mutation recurrence

As Figure 3.5 shows, $p_1 \sim p_7$ are mutation points $(p_1 < p_2 < \cdots < p_7)$, and each mutation has a corresponding window w_i of 101 bp, which contains the mutations: $\{p_j \mid p_i - 50 \leq p_j \leq p_i + 50\}$. Now we define the noncoding mutation recurrence within a 101 bp window as

$$\text{recurrence}[i] = \sum_{j=1}^{K} \text{freq}(p_j), \;\; s.t. \; p_i - 50 \leq p_j \leq p_i + 50, \tag{3.7}$$

where freq(p_j) indicates the sample frequency from COSMIC. We can easily calculate the recurrence of all mutations by walking through the mutation list exactly once after the list is sorted therefore the running time is $O(n \log n)$. Now we have the response variable (mutation recurrence) and the predictors (features), so in the next section, we illustrate the proper regression models for the noncoding mutation recurrence data analysis.

3.3.2 Generalized Linear Models

Although linear regression models are popular and powerful for regression analysis in general, they cannot be applied to mutation count data. First, apparently from the definition of recurrence, we can see the noncoding mutation recurrence value is based on the counts of mutation points within the 101 bp window, the value is 1 at least, and most of the values are 1s but only very few mutations have big recurrence value. At this point, it can be concluded that the distribution of the recurrence (discrete positive integers) is not Gaussian and also it is very likely that the error (difference between the predicted value and the ground-truth value) is not normally distributed either, which does not satisfy the error normal distribution assumption of the linear regression models [10]. Second, the features are in different scales, i.e., the counts feature vs. the scores feature, or the distance feature vs. the ratio feature, therefore it is hard to see a linear correlation between the predictors (the independent variables) and the response variable. One alternative is to log-transform the data but it would lose too much valuable information. Instead for noncoding mutation recurrences, we need to consider non-linear regression models.

As discussed in the review section, the well-known regression models for discrete count data analysis are Poisson regression and Negative Binomial regression, which belong to the family of Generalized Linear Models (GLM). The most well-established toolkit in the state-of-the-art for GLM regression is the R package `MASS` [77].

3.3.2.1 Poisson Regression

We split the whole dataset randomly into two parts: one is for training which takes 80%, and the left 20% is for testing:

$$train\_data = [train.X, train.y]$$
$$test\_data = [test.X, test.y],$$

where $train\_data$ and $test\_data$ are data frames, $train.X$ and $test.X$ are the feature vectors, $train.y$ and $test.y$ are vectors of response variables, i.e., noncoding mutation recurrence counts.

For Poisson regression (also called as log-linear regression), we use the GLM functions to build the regression model with training data and then use the learned model to predict with test data:

$$glm.fit.res \leftarrow glm(counts \sim ., data = train\_data, family = poisson(link = log)$$
$$pred.counts \leftarrow predict.glm(glm.fit.res, newdata = test.X, type = \text{``}response\text{''})$$

We use the log-link function for regression:

$$\begin{aligned}
\ln(\mu) &= \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p = \beta^T X \\
\mu &= e^{\beta^T X} \\
P(Y = y|\mu) &= \frac{e^{-\mu}\mu^y}{y!},
\end{aligned}$$

where $\beta = (\beta_0, \beta_1, \ldots, \beta_p)$ is the coefficients vector that the regression model need to learn, and p is the number of predictors (features). The squared loss function and the maximum likelihood estimate optimization strategy are already illustrated in the review section. However, since the Poisson regression model has a strong hypothesis:

$$\begin{aligned}
\mu &= \text{mean}(y) = \text{variance}(y) \\
\text{dispersion} &= \frac{\text{variance}(y)}{\text{mean}(y)},
\end{aligned}$$

that is, the expectation (mean) value of the response variable is equal to its variance, we need to verify the assumption by checking dispersion (ratio of the mean value and the variance value). Meanwhile we also need to check the goodness-of-fit of the regression model which is discussed in the model evaluation section below.

3.3.2.2 Negative Binomial Regression

For counts data, usually it easily gets overdispersed: variance$(y) >$ mean(y), and the experience ratio is approximate to 2 as shown in Figure 2.4 in the review section. If this is the case, a corrected regression model is required, that is where the Negative Binomial regression is used. Similarly as Poisson regression, we use the GLM functions for model learning and prediction in the R package:

$$glm.fit.res \leftarrow glm.nb(counts \sim ., data = train\_data, link = log)$$
$$pred.counts \leftarrow predict.glm(glm.fit.res, newdata = test.X, type = \text{“}response\text{”}).$$

We also use the log-link function for regression:

$$\begin{aligned}
&\ln(\mu) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p = \beta^T X \\
&\mu = e^{\beta^T X} \\
&prob(Y = y|\mu, \alpha) = \frac{\Gamma(y + 1/\alpha)}{\Gamma(y+1)\Gamma(1/\alpha)} \left(\frac{1}{1+\alpha\mu}\right)^{1/\alpha} \left(\frac{\alpha\mu}{1+\alpha\mu}\right)^{y},
\end{aligned}$$

where μ is the mean, α is the heterogeneity parameter which represents the extent of overdispersion and is tuned internally in the regression algorithm, and β is the coefficients parameter that the regression model need to learn. All details about the loss function used in Negative Binomial regression and the maximum likelihood estimates of the parameters were already illustrated in the review section. However, in our model, the response variable recurrence ≥ 1, since there is at least 1 mutation point in each 101 bp sliding window but the Negative Binomial regression would still predict $0s$ so that the model may not converge. Therefore, for simplicity and for Negative Binomial regression only, we adjust the response variable as

$$\begin{aligned}
train.y &\leftarrow train.y - 1 \\
test.y &\leftarrow test.y - 1
\end{aligned}$$

with this approach, the response variable would have new range recurrence ≥ 0. An alternative approach is to use Zero-Truncated Negative Binomial Regression model [36, 55, 67, 86] which excludes zeros and rescales the other probabilities to sum to one, which is implemented in another R package `VGAM` [85] as

$$vglm.fit \leftarrow vglm(counts \sim ., family = posnegbinomial(), data = train\_data)$$

However, for the present thesis, we used the Poisson and negative binomial regression models.

3.3.3 Ensemble of Decision Trees Models

The main advantages of decision trees for regression include: (1) it easily handle irrelevant features through information gain as it always select the best features to split the branches; (2) it is robust against skewed distributions, since it does not make any assumptions regarding the variable's distribution when constructing axis splits; (3) it does not require any assumptions of linearity in the data. Since our features data are highly skewed and in different scales or fields, we can take advantage of decision trees to build the regression model.

However, a notable disadvantage of decision trees is overfitting. Therefore we use additional ensemble methods to avoid that by using the weak learner - decision tree stump as the base learner. Generally the ensemble methods include three categories: bagging, boosting, and random ensembles (also called random forest). In this thesis, we use the algorithms that are widely applied in regression problems analysis, which are random forest regression, gradient boosting regression and AdaBoost regression. All the three models have been discussed in the review section. To our best of our knowledge, the most well-established machine learning toolbox is `scikit-learn` [60], which is also deployed in our regression analysis.

3.3.3.1 Random Forest Regression

The basic idea of random forest is to build trees using fixed structures and random features. Each tree in the ensemble is built from a sample drawn with replacement (i.e., a bootstrap sample) from the training set. In addition, when splitting a node during the construction of the tree, the split that is chosen is no longer the best split among all features. Instead, the split that is picked is the best split among a random subset of the features. As a result of this randomness, the bias of the forest usually slightly increases (with respect to the bias of a single non-random tree) but, due to averaging, its variance also decreases, usually more than compensating for the increase in bias, hence yielding an overall better model. And for prediction, just average the predicted output values of each tree. Usually random forest model is popularly applied in classification problems but it is easily to be modified for regression problems, for example, changing the objective function to MSE (mean squared error). In `scikit-learn` toolkit, the function interface that we use:

$$\begin{aligned} sklearn.ensemble.RandomForestRegressor(&n\_estimators,\\ &max\_features = p,\\ &criterion = \text{“}mse\text{”},\\ &oob\_score = True,\\ &min\_samples\_split,\\ &n\_jobs = 16) \end{aligned}$$

where $n\_estimators$ indicates the number of trees to build, $max\_features$ the number of features to consider when looking for the best split, $criterion$ the objective function used to measure the quality of a split, oob_score whether to use out˙of˙bag samples to estimate the generalization error, $min\_samples\_split$ the minimum number of samples required to split an internal node, and n_jobs the number of threads to run in parallel for both predict and fit. For all other parameters, we just use the default configurations.

The main parameters to be tuned in the Random Forest regression model are the number of trees ($n\_estimators$) and the number of features for the best split ($max\_features$). Having a larger number of trees is better but accuracy would not be expected to significantly improve beyond a critical number of trees, which is in most cases determined empirically. For regression, the default empirical good value for $max\_features$ is the number of features in data [60]. Therefore mainly we tune the parameter $n\_estimators$ using a cross-validation approach as illustrated in the review section.

3.3.3.2 Gradient Boosting Trees Regression

As explained in the review section, boosting is an another ensemble method which learns the model from previous errors and update the weights sequentially. And during the weights update, it takes the gradients of the loss function, in this thesis, we still use the squared error as the loss function as all other regression models. Putting all together, we use the function interface in the `scikit-learn` toolkit:

$$\begin{aligned}
sklearn.ensemble.GradientBoostingRegressor(n\_estimators,\\
max\_depth &= 4,\\
min\_samples\_split &= 2,\\
learning_rate &= 0.1,\\
loss &=' ls',\\
subsample &= 0.5)
\end{aligned}$$

where $n\_estimators$ represents the number of boosting stages to perform; max_depth represents the maximum depth of the individual regression estimators, which limits the number of nodes in the tree; $min\_samples\_split$ represents the minimum number of samples required to split an internal node; $learning\_rate$ shrinks the contribution of each tree; $loss$ is the loss function to be optimized, here we use the default "ls" - least squared error; $subsample$ indicates the fraction of samples to be used for fitting the base learners.

To get best performance of the model, we mainly tune the parameter $n\_estimators$ - the size of the regression tree base learners which defines the level of variable interactions that can be captured by the gradient boosting model as gradient boosting is fairly robust to over-fitting so a large number usually results in better performance. Another main parameter to be tuned is max_depth. And still we use K-fold Cross-Validation to choose the best parameter for the best performance.

3.3.3.3 Decision Trees Regression with AdaBoost

Another widely-used boosting method is AdaBoost which is different from gradient boosting, as AdaBoost does not take gradients to update the coefficient weights, instead it update the weights of the instances according to the error of the current prediction so that subsequent regressors focus more on difficult cases. Namely in each boosting iteration the weights $w_1, w_2, \ldots, w_N$ were applied/updated to each of the training samples. Initially those weights were assigned to $w_i = 1/N$. For each successive iteration, the sample weights are individually modified and the learning algorithm is reapplied to the re-weighted data. At a given step, those training examples that were incorrectly predicted by the boosted model induced at the previous step have their weights increased, whereas the weights are decreased for those that were predicted correctly. As iterations proceed, examples that are difficult to predict receive ever-increasing influence. Each subsequent weak learner is thereby forced to concentrate on the examples that are missed by the previous ones in sequence [60]. Like the other two ensemble methods, we use the `scikit-learn` toolkit:

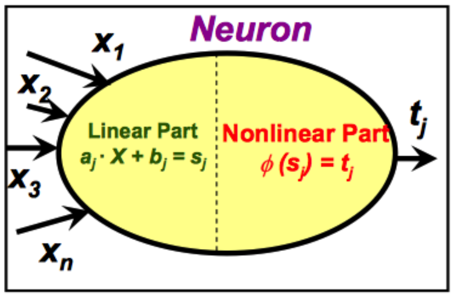
$$
\begin{aligned}
sklearn.ensemble.AdaBoostRegressor(DecisionTreeRegressor(&\\
&criterion =' mse',\\
&max\_features =' auto',\\
&max\_depth = 4,\\
&min\_samples\_split = 2\\
&),\\
&n\_estimators,\\
&learning\_rate = 0.1,\\
&loss =' square'\\
&)
\end{aligned}
$$

where $DecisionTreeRegressor$ is the base learner for regression used in AdaBoost regression model, $criterion$ is the cost function used in decision tree regressor to measure the quality of a split (here we use 'mse' - mean squared error), $max\_features$ is the number of features to consider when looking for the best split (same as random forest), max_depth is the maximum depth of the tree, $min\_samples\_split$ is the minimum number of samples to split an internal node, $n\_estimators$ is the number of estimators at which boosting is terminated, $learning\_rate$ shrinks the contribution of each regressor, and $loss$ is the loss function to use when updating the weights after each boosting regression (here we use squared error which is also used in other models).

Same as the gradient boosting regression algorithm, the main parameter to be tuned for best performance of the AdaBoost regression model is $n\_estimators$, which indicates the number of weak learners (decision trees).

3.3.4 Deep Neural Network

Neural Networks are generic, accurate and convenient mathematical (statistical) models which are able to emulate numerical model components, which are complicated nonlinear input/output relationships [43]. The main advantages of the neural network model class is that it can avoid curse of dimensionality and it is robust with respect to random noise and fault-tolerant. To take advantage of the power of nonlinear representations of neural network, we also build a feed-forward ANN (Artificial Neural Network) as the base learner and then build a deep neural network—a Stacked Auto Encoder—for our regression analysis. Mainly the base learner ANN has three layers: the input layer which represents the input features vector X, the hidden layer which is consist of nonlinear neurons, and the output layer is consist of linear neurons Y as Figure 2.6 shows.



$$t_j = \phi(b_{j0} + \sum_{i=1}^{n} b_{ji} \cdot x_i) =$$

$$= \tanh(b_{j0} + \sum_{i=1}^{n} b_{ji} \cdot x_i)$$

Figure 3.6: Neural Network Nonlinear Representation (figure reproduced from [80])

For our noncoding mutation recurrence regression analysis, the input X is the vector of the extracted genomic features as we discussed above; in the hidden layer, we use the tanh function as the activation function; and for the output layer, we only have one output node that is a linear combination of the activation output of the hidden layer. We use the MSE as the loss function and take the gradients to update the weights backwards using backpropagation algorithm which was already discussed in the review section.

The best-known toolkit for ANN and deep learning is Theano [5, 6] and Keras [15] which is built based on Theano. The main procedure for building Neural Network Regression model using Keras is described in Algorithm 6. In Algorithm 6, the function $Dense$ indicates a fully-connected network, which means that the connections from the input layer nodes to the hidden layer nodes, and from the hidden layer nodes to the output layer nodes, are complete; $model.fit$ is the training function where 10-fold cross-validation ($validation\_split$ = 0.1), while $model.predict$ is a prediction function based on the trained model; $Adam$ is a regularization function used for controlling the learning rate in the training phase; $model.compile$ is a function compiling the network structure with the loss function $mean\_squared\_error$ which is used in our regression analysis.

Compared to the base learner ANN, a deep neural network, which has two or more hid-

Algorithm 6 NEURAL NETWORK REGRESSION

```
input : Training set $X\_train_{n\times p}$ and $Y\_train_{n\times 1}$ where $n$ is training size and $p$ is feature
        dimension; Test set $X\_test_{m\times p}$ and $Y\_test_{m\times 1}$ where $m$ is the test data size. And
        list of parameters to be tuned for best model performance: number of hidden nodes
        $hidden\_sizes$, batch size for batching update in training $batch\_size$, number of itera-
        tions $nb\_epoch$
output: Predicted recurrences $\hat{Y}\_test$

for $hidsize$ in $hidden\_sizes$ do
    $model \leftarrow Sequential()$
    $model.add(Dense(hidsize, input\_dim = p, activation =' tanh'))$
    $model.add(Dense(1, input\_dim = hidsize))$
    $adam \leftarrow Adam(lr = 0.01, beta\_1 = 0.9, beta\_2 = 0.999, epsilon = 1e-08)$
    $model.compile(loss =' mean\_squared\_error', optimizer = adam)$
    $hist \leftarrow model.fit(X\_tain, Y\_train, batch\_size, nb\_epoch, validation\_split = 0.1)$
    $hist\_list.append(hist)$
    $model\_list.append(model)$
end
$j \leftarrow \underset{i}{argmin}\{mean(hist\_list[i].val\_loss)\}_{1\le i\le length(hidden\_sizes)}$
$best.model \leftarrow model\_list[j]$
$\hat{Y}\_test \leftarrow best.model.predict(X\_test)$
return $\hat{Y}\_test$
```

den layers, is more powerful for nonlinear representation of the data, especially in self-learning features. In order to take advantage of the deep learning techniques, we use sparse Stacked AutoEncoder model for our noncoding mutation recurrence regression analysis. The main idea of auto encoder deep neural network is to encode the features to a lower dimension (learned new features) and then to decode to the original input dimension, so that the features can be more compacted and the noise were filtered out. Stacked AutoEncoder is namely a stack of encoders, usually two or more than two. For simplicity, we use two encoders as Figure 3.7 shows:

The SAE network looks similar to a fully-connected two-hidden-layer neural network but they are different. All the weights in SAE were pretrained: weights from the input layer to the first encoders, from the first encoder layer to the second encoder layer, and from the second encoder layer to the output layer. As discussed in the review section, the weights were pretrained using Auto Encoder network structure but the output layer was dropped and the weights were cached. When training the weights from the first encoder to the second encoder, the previous

Algorithm 7 STACKED AUTOENCODER REGRESSION

input : Training set $X\_train_{n\times p}$ and $Y\_train_{n\times 1}$ where n is training size and p is feature dimension; Test set $X\_test_{m\times p}$ and $Y\_test_{m\times 1}$ where m is the test data size. And list of parameters to be tuned for best model performance: pair number of nodes in two hidden layers $\{(num\_hid1, num\_hid2)\}$, batch size for batching update in training $batch\_size$, number of iterations nb_epoch

output: Predicted recurrences $\hat{Y}\_test$

```
for $(i, j)$ in $\{(num\_hid1, num\_hid2)\}$ do
    $encoders \leftarrow \{\}$
    $decoders \leftarrow \{\}$
    $X\_train\_tmp \leftarrow X\_train$
    for $(n\_in, n\_out)$ in $\{(p, i), (i, j)\}$ do
        $model \leftarrow Sequential()$
        $encoder \leftarrow containers.sequential([Dense(n\_out, n\_in, activation =' tanh')])$
        $decoder \leftarrow containers.sequential([Dense(n\_in, n\_out, activation =' tanh')])$
        $model.add(AutoEncoder(encoder, decoder, output\_reconstruction = False))$
        $model.compile(loss =' mse', optimizer =' adam')$
        $model.fit(X\_train\_tmp, X\_train\_tmp, batch\_size, nb\_epoch)$

        $encoders.append(encoder)$
        $decoders.append(decoder)$
        $X\_train\_tmp \leftarrow model.predict(X\_train\_tmp)$
    end
    $full\_ae \leftarrow Sequential()$
    $full\_ae.add(AutoEncoder(encoders, decoders, output\_reconstruction = False)$
    $full\_ae.compile(loss =' mse', optimizer =' adam')$
    $full\_ae.fit(X\_train, X\_train, batch\_size, nb\_epoch)$


    $model \leftarrow Sequential()$
    $model.add(encoders)$
    $model.add(Dense(1, input\_dim = j))$
    $model.compile(loss =' mean\_squared\_error', optimizer =' adam')$
    $hist \leftarrow model.fit(X\_train, Y\_train, batch\_size, nb\_epoch, validation\_split = 0.1)$
    $hist\_list.append(hist)$
    $model\_list.append(model)$
end
$j \leftarrow \underset{i}{argmin}\{mean(hist\_list[i].val\_loss)\}_{1\leq i\leq length(hidden\_sizes)}$
$best.model \leftarrow modle\_list[j]$
$\hat{Y}\_test \leftarrow best.model.predict(X\_test)$
return $\hat{Y}\_test$
```

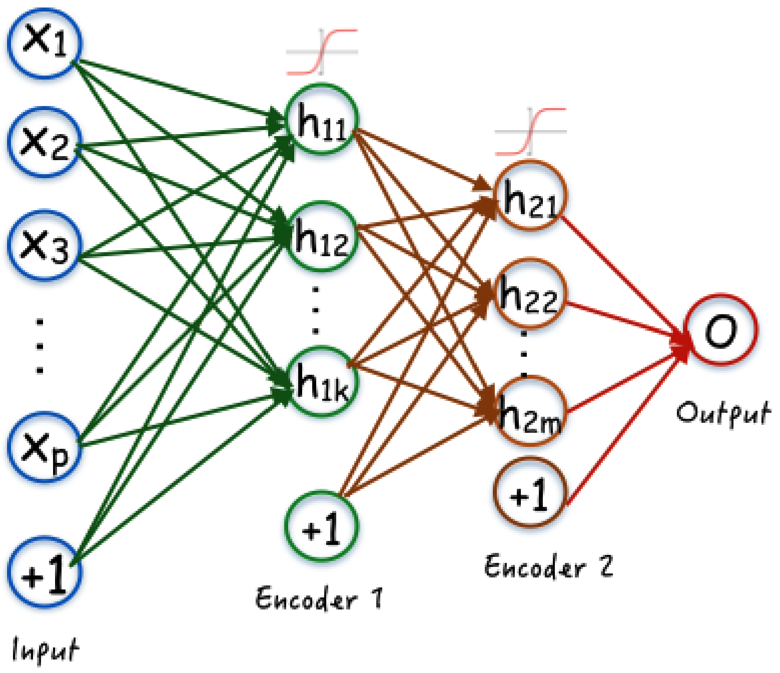


Figure 3.7: Stacked AutoEncoder Regression

learned activation values of the first hidden units were used as the network input, instead of the original input features. Still we use the Python package Keras for SAE implementation. The main procedures are described in Algorithm 7.

To get the best performance of neural network regression and Stacked Auto Encoder regression, the main parameters to be tuned is the number of hidden units and the learning rate used for regularization in the training phase. Since we use auto encoder network structure to compact the input features, the number of hidden units should be less than the input dimensions $p > k > m$ where p is the input feature dimension, k is the first encoder dimension, and m is the second encoder dimension in Figure 3.7.

3.4 Model Evaluation

So far we already talked about six different regression models, including Poisson regression, Negative Binomial regression, random forest regressor, gradient boosting regressor, AdaBoost regressor, Neural Net, and Stacked Auto Encoder. All these models use different optimization strategy, so it is not easy to calculate the deviance and Pearson chi-square statistics among all models which are elucidated in the review section. However, as `scikit-learn` suggests, for regression problems, we can calculate the general metrics used to evaluate the goodness-of-fit of the model: the mean absolute error, the mean squared error, the median absolute error, the R^2 score, and the explained variance score.

3.4.1 Explained Variance Score

If $\hat{y}$ is the estimated target output, y the corresponding (ground-truth) target output, and Var is variance, the square of the standard variation, then the explained variance is estimated as follows,

$$explained\_variance(y, \hat{y}) = 1 - \frac{\text{Var}\{y - \hat{y}\}}{\text{Var}\{y\}}.$$

The best possible score is 1.0 and lower values are worse.

3.4.2 Mean Absolute Error

The performance metric $mean\_absolute\_error$ computes mean absolute error, a risk metric corresponding to the expected value of the absolute error loss or $l1 - norm$ loss.

If $\hat{y_i}$ is the predicted value of the i-th sample, and y_i is the corresponding true value, then the mean absolute error(MAE) estimated over n samples is defined as

$$\text{MAE}(y, \hat{y}) = \frac{1}{n} \sum_{i=0}^{n-1} |y_i - \hat{y_i}|.$$

Apparently lower MAE can achieve better prediction performance.

3.4.3 Mean Squared Error

The mean squared error is a risk metric corresponding to the expected value of the squared (quadratic) error loss.

If $\hat{y_i}$ is the predicted value of the i-th sample, and y_i is the corresponding true value, then the mean squared error (MSE) estimated over n samples is defined as

$$\text{MSE}(y, \hat{y}) = \frac{1}{n} \sum_{i=0}^{n-1} (y_i - \hat{y_i})^2.$$

Just as with MAE, a lower MSE corresponds to better regression performance.

3.4.4 Median Absolute Error

The metric $median\_absolute\_error$ is particularly interesting because it is robust to outliers. The loss is calculated by taking the median of all absolute differences between the target and the prediction.

If $\hat{y_i}$ is the predicted value of the i-th sample, and y_i is the corresponding true value, then the median absolute error (MedAE) estimated over n samples is defined as

$$MedAE(y, \hat{y}) = median(|y_1 - \hat{y_1}|, \ldots, |y_n - \hat{y_n}|).$$

3.4.5 R^2 score

In the context of count regression, the R^2 score is known as the coefficient of determination. It provides a measure of how well future samples are likely to be predicted by the model. Best possible score is 1.0 and it can be negative (because the model can be arbitrarily worse) [60]. A constant model that always predicts the expected value of y, disregarding the input features, would get a R^2 score of zero.

If $\hat{y_i}$ is the predicted value of the i-th sample and y_i is the corresponding true value, then the score R^2 estimated over n samples is defined as

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=0}^{n-1} (y_i - \hat{y_i})^2}{\sum_{i=0}^{n-1} (y_i - \bar{y})^2},$$

where $\bar{y} = \frac{1}{n} \sum_{i=0}^{n-1} y_i$. Higher R^2 score indicates better prediction performance.

3.5 Overall Procedure

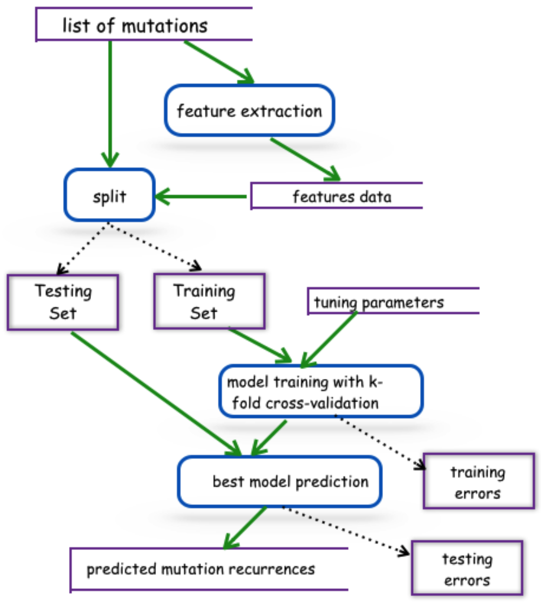


Figure 3.8: Overall Regression Analysis Procedures

Now putting it all together, we used three categories of regression methods for our noncoding mutation recurrence regression analysis, including the generalized linear models, the ensemble method of decision trees, and the deep neural networks. For each noncoding mutation, we constructed the features that are likely to be correlated with the mutation frequency in cancer. We then randomly split the set of samples (mutations) into two parts: one set of mutations is for model training and the other is for testing. Since all machine learning methods have to be tuned with different model parameters (e.g., the number of decision trees, the number of iterations, and the learning rate) to get best performance, we used 5-fold cross-validation to tune each model. We then measured the accuracy of the tuned model's predictions on the test data set, and compared the models' test-set prediction accuracies on identical test sets, using multiple accuracy measures.

Chapter 4: Results

4.1 Recurrence Distribution

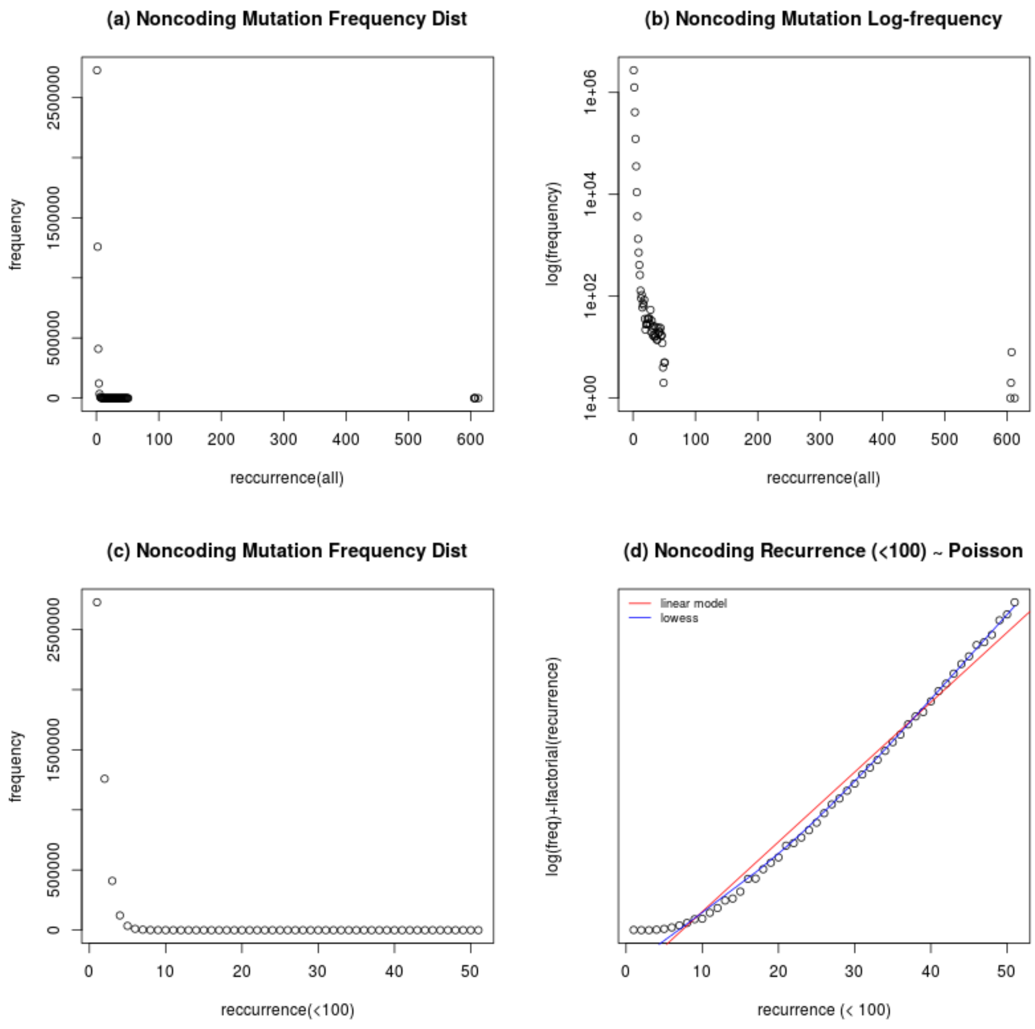


Figure 4.1: Noncoding mutation data distribution: (a) the frequency (number of occurrence) of noncoding mutation recurrence; (b) log frequency of overall noncoding mutations; (c) the frequency of noncoding mutations with recurrence less than 100; (d) as a function of the recurrence count, the Poisson model-predicted and empirical log frequency of mutations (plus the log factorial).

As we can see from the Figure 4.1, in the regression model, the response variable, i.e., the non-coding mutation recurrence, is not normally or uniformly distributed. Instead, the distribution is very close to Poisson as shown in (c) and (d): small recurrence has extremely high frequency while large recurrence has quite low frequency (i.e., equal to 1). From the Poisson distribution formula:

$$p(y) = \frac{e^{-\lambda}\lambda^{y}}{y!} \Rightarrow$$
$$\ln(p(y)) + \ln(y!\,) = -\lambda + y\ln(\lambda),$$

so the correlation between $\ln(p(y)) + \ln(y!\,)$ and y (mutation recurrence) is linear. However, when looking at mutations with recurrence less than 100, the linear correlation is not quite strong so it may have over-dispersion, as Figure 2.3 shows. To quantify this, we can examine the ratio between the mean and the variance of the response variable:

Table 4.1: Recurrence variance/mean ratio

sampling	mean	variance	ratio
1.0	1.596	1.919	1.203
0.3	1.599	2.335	1.460
0.4	1.595	1.861	1.167
0.5	1.595	1.885	1.181
0.6	1.597	2.305	1.444
0.7	1.596	1.901	1.191
0.8	1.597	2.234	1.399

Therefore, the response variable variance is a bit more than the mean so that the data may be not be well-fit by the Poisson regression model which has the assumption that the mean is equal to the variance. Instead a overdispersed version model is needed such as Quasi Poisson regression or Negative Binomial regression, which are discussed in the section below.

4.2 GLM Regression Analysis

In this section, we present the results from applying the three GLM regression models to the mutation recurrence count prediction problem: Poisson regression, Quasi-Poisson regression and

Negative Binomial regression by evaluating the goodness-of-fit of the three models by comparing the deviance and the Pearson's chi-square statistics which are illustrated in the review section 2.2.3.

Table 4.2: GLM regression statistics

metric	poisson	quasi_poisson	neg_binomial
$null\_deviance$	507,429	507,429	1,017,382
$residual\_deviance$	506,095	506,095	1,015,060
$deviance\_improve$	0.26%	0.26%	0.23%
chi_square	1,492,645	1,492,645	3,639,893

From Table 4.2 we can see the overdispersed version of Poisson regression model - Quasi-Poisson regression does not have improvement. From the deviance improvement compared to the null hypothesis, the Poisson model performs a little better than the Negative Binomial model. Now we look at the mean squared error as Figure 4.2 shows:

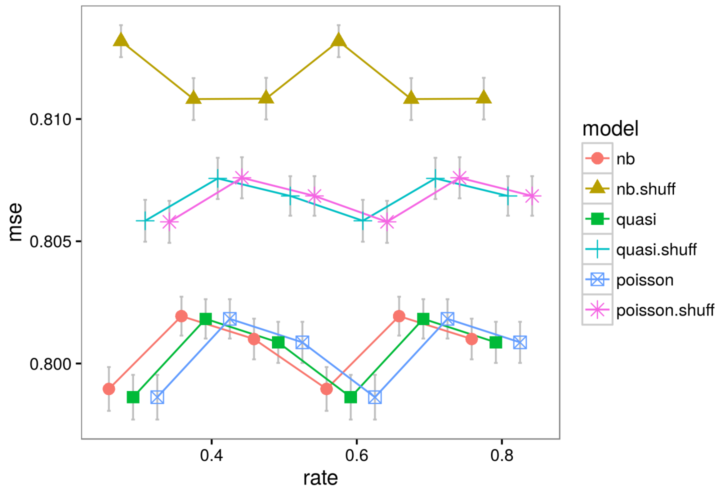


Figure 4.2: MSE of GLM regression models with different sampling rates

where the X axis is the sampling rate, each MSE value is averaged over multiple samplings of each sampling rate, and the error bar represents the standard error of the MSE. And in the figure, the above three lines (*nb.shuff,quasi.shuff,poisson.shuff*) represent the MSE of the GLM models against the shuffled features, namely the MSE of prediction on the random features' values, and the below three lines represent the MSE of the GLM models against the real features data in the testing set. As expected we can see the MSE of the shuffled data is larger than the MSE of the real data, which means the three models have prediction power for the noncoding mutation recurrences. Furthermore, the Poisson regression model performs a little better than the Negative Binomial model as it has lower MSE.

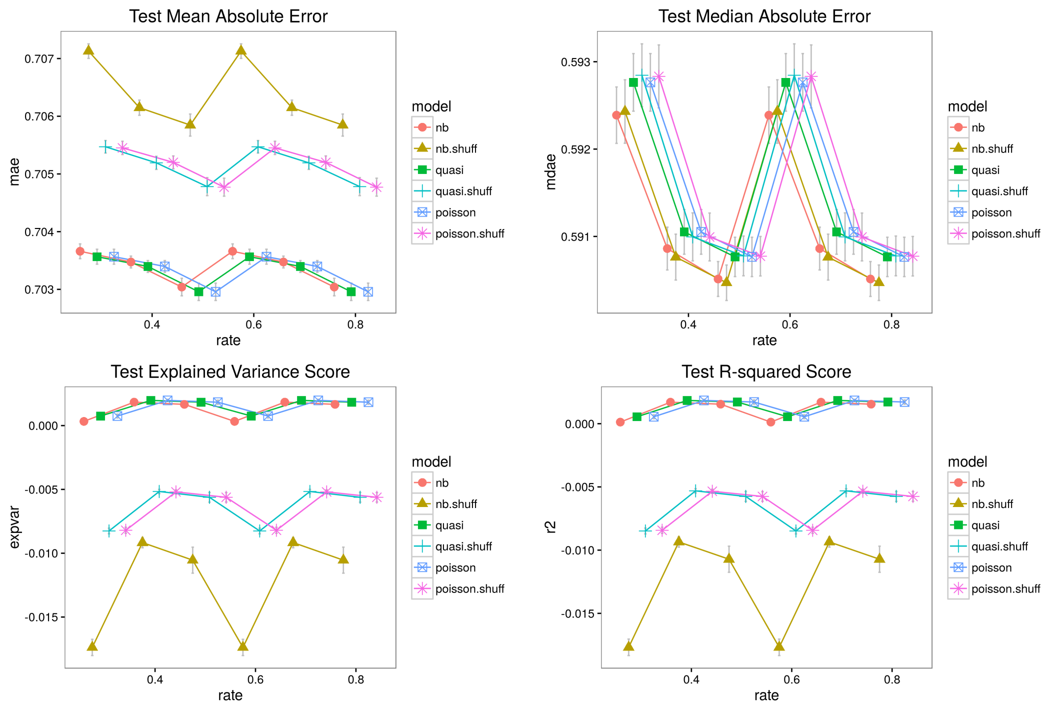


Figure 4.3: GLM regression models performance with different sampling rates

Also in Figure 4.3 (where the error bar indicates the standard error of the mean), all of the three models have positive explained variance scores and R^2 scores for the real features data while all of the score values for the shuffled features data are negative, which is consistent with the MSE results. Among the three models, the Poisson models have lower MAE and higher explained variance score and R^2 scores than the Negative Binomial model.

To sum up, with regression analysis of three GLM models, the Poisson models perform a little better than the Negative Binomial model with lower MSE/MAE and higher explained variance scores and R^2 scores.

4.3 Ensemble Methods Analysis

In this section we analyze regression results from applying three ensemble methods: Random Forest, gradient boosting trees, and decision trees with AdaBoost. The table below shows parameters to be tuned for each model to get best performance.

Table 4.3: Ensemble methods parameters tuning

model	parameter	values
Random Forest	Number of trees	$\{100, 300, 500, 700\}$
Gradient Boosting Trees	Number of boosting iterations	$\{50, 100, 200, 400\}$
Decision Trees with AdaBoost	Number of trees	$\{50, 100, 200, 300\}$

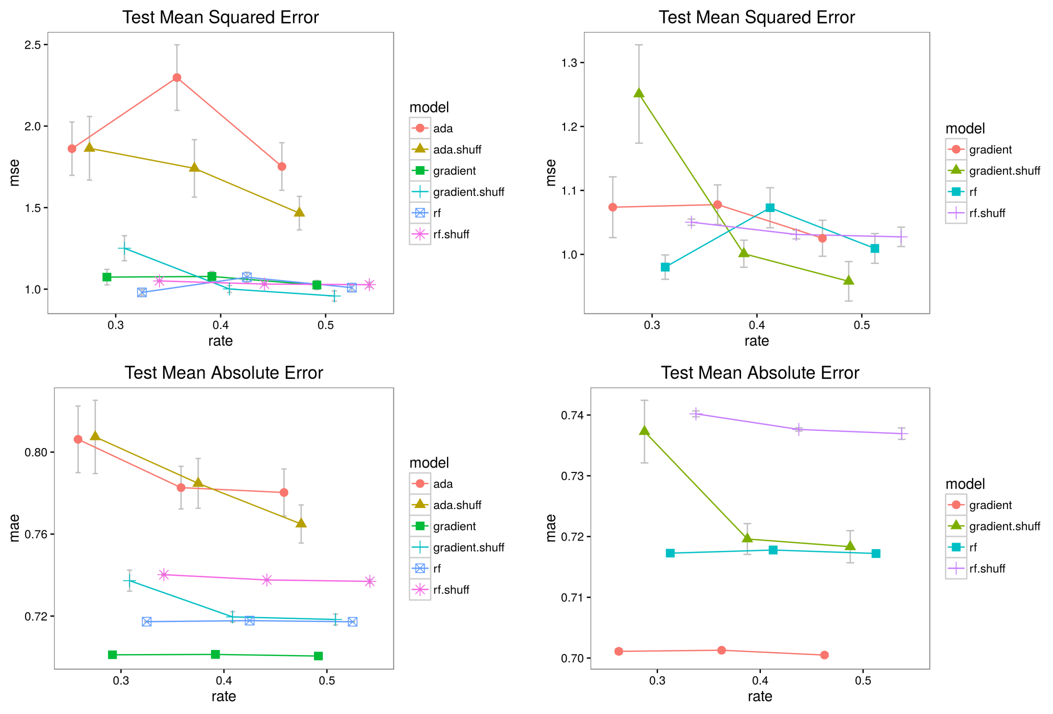


Figure 4.4: Ensemble regression models test performance with different sampling rates

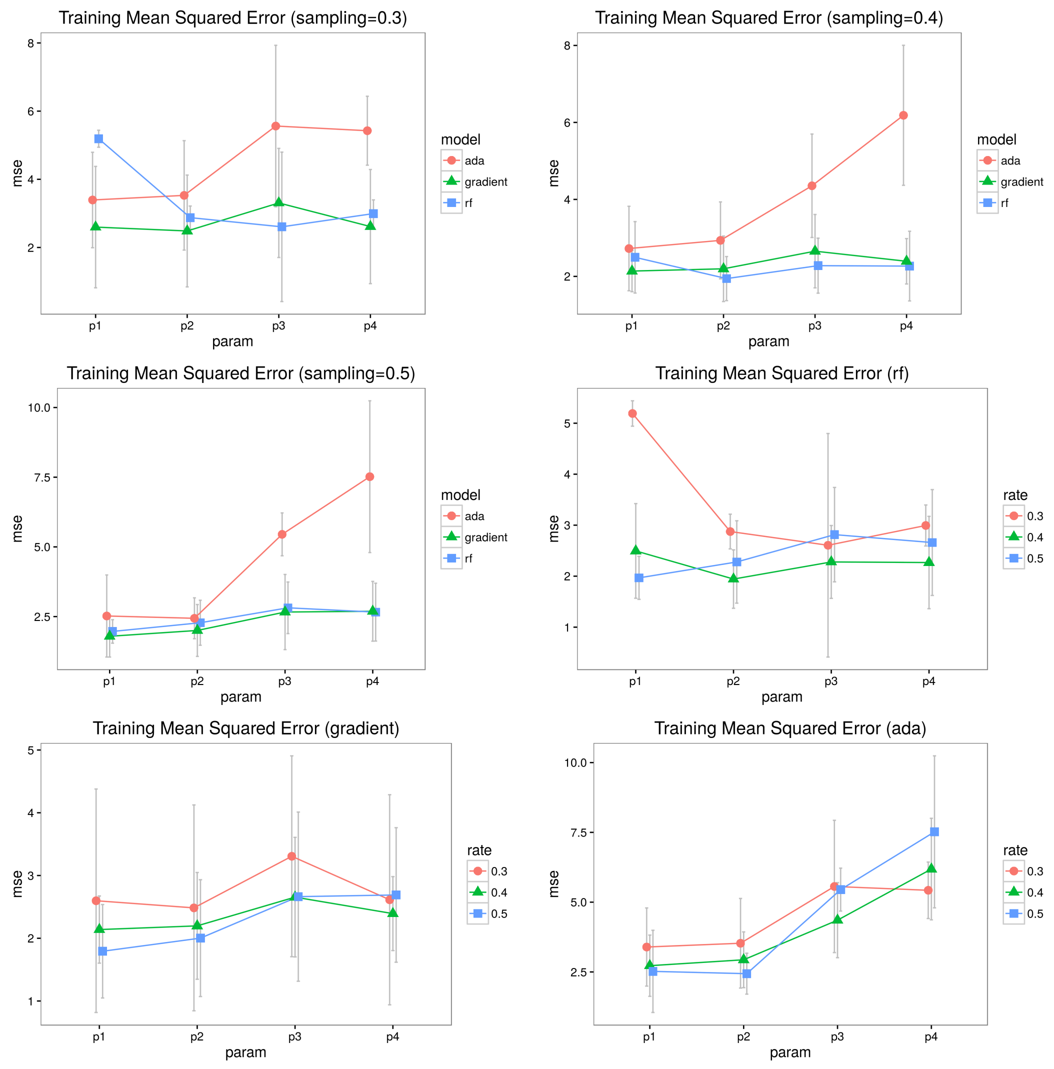


Figure 4.5: Ensemble regression models training performance with different sampling rates

And the corresponding training MSE over different parameters is shown in Figure 4.5 where the error bars represent the standard error of the mean value.

Obviously among the three ensemble regression models, the decision tree regression with AdaBoost performs the worst for all sizes of data since the error of the random features (*ada.shuff*) is less than the error of real features. From the test mean squared error, the other two models gradient boosting and random forest perform relatively poorly since there is no big error difference between prediction on the real feature data (*rf, gradient*) and prediction on the random feature data (*rf.shuff, gradient.shuff*).

From the training result which is shown in Figure 4.5 where the error bars represent the standard deviation, tuning the number of boosting iterations did not help to get best AdaBoost regression model which basically does not work well for our noncoding mutation recurrence data. Increasing number of decision trees could get lower MSE/MAE for both gradient boosting and random forest but when the trees number arrives at some extent, the error would increase. However the gradient boosting tree and random forest regression models may still have some predictive power if more data is ingested.

4.4 Deep Neural Network Regression Analysis

In this section, we discuss about the regression result of regression models based on neural network structures, including the one-hidden layer artificial neural network and Stacked AutoEncoder (SAE) which is based on ANN. To get best performance of network regression model, the main tuning parameter is the size (number of nodes) of the hidden layers.

Table 4.4: Neural Network regression parameters tuning

model	parameter	values
ANN	one-hidden layer	$\{2, 3, 5, 7, 9, 10, 12\}$
SAE	two-layer encoders	$\{(12, 10), (10, 8), (12, 5), (10, 5), (8, 4), (6, 3)\}$

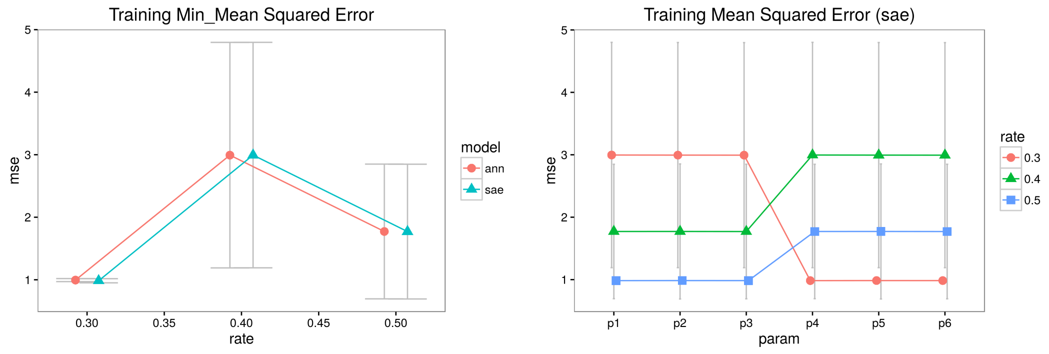


Figure 4.6: Deep neural network regression training error with different sampling rates

As Figure 4.6 shows, *Min_Mean_Squared Error* represents the minimum MSE with each sampling rate over different hidden layer size configuration. One interesting finding is that the ANN and the SAE have approximately equal error even the difference can be ignored. One possible reason is that the feature space is too small (as eventually we only have 13 features after applying filtering strategy). However, most importantly from the test MSE of SAE, with increasing size of training data, the error went down deeply. This is consistent with the beauty of deep learning techniques - more data, then the model is more powerful.

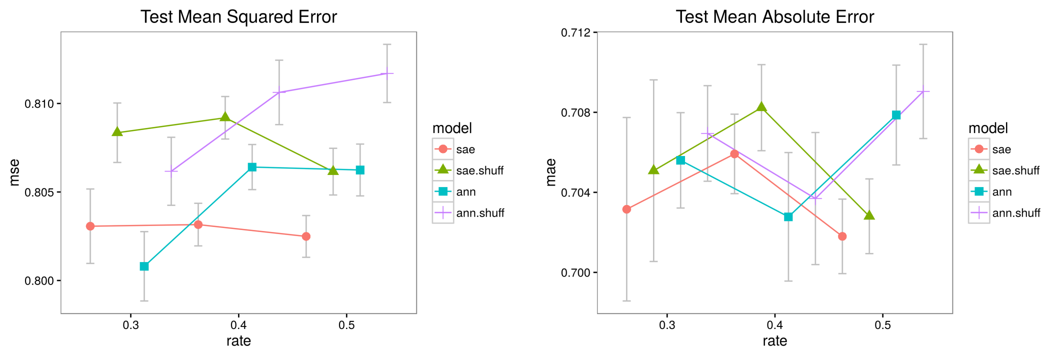


Figure 4.7: Deep neural network regression test error with different sampling rates

Although we did not see much difference between ANN and SAE from the training error, from the test error as Figure 4.7 shows, the stacked AutoEncoder performed a bit better than the simple base learner - *one-hidden-layer* ANN.

4.5 Overall Performance Summary

Now put all test performance data together to see the differences across all regression methods used in this thesis. As Figure 4.8 shows where the error bars represent the standard error of the mean value, the decision tree with AdaBoost regression is the worst model for the noncoding mutation recurrence data. As we already discussed, AdaBoost regression even performed worse on the real feature data compared to the shuffled (random) feature data, which suggests AdaBoost may not be good for the mutation recurrence data.

Instead, the other four regression models: Poisson, Negative Binomial, ANN, SAE perform the best and their MSE is very close. Comparatively Poisson regression model performs a bit better than the other three models when looking at the MSE only although the difference is quite small. When looking at the MAE, the SAE regression looks a little better than the others when the data volume is larger.

For the left two ensemble methods, random forest and gradient boosting regression, the overall performance is better than AdaBoost, but they are still weak predictors as we already discussed, the error difference between the prediction on the real feature data and the random feature data is not big.

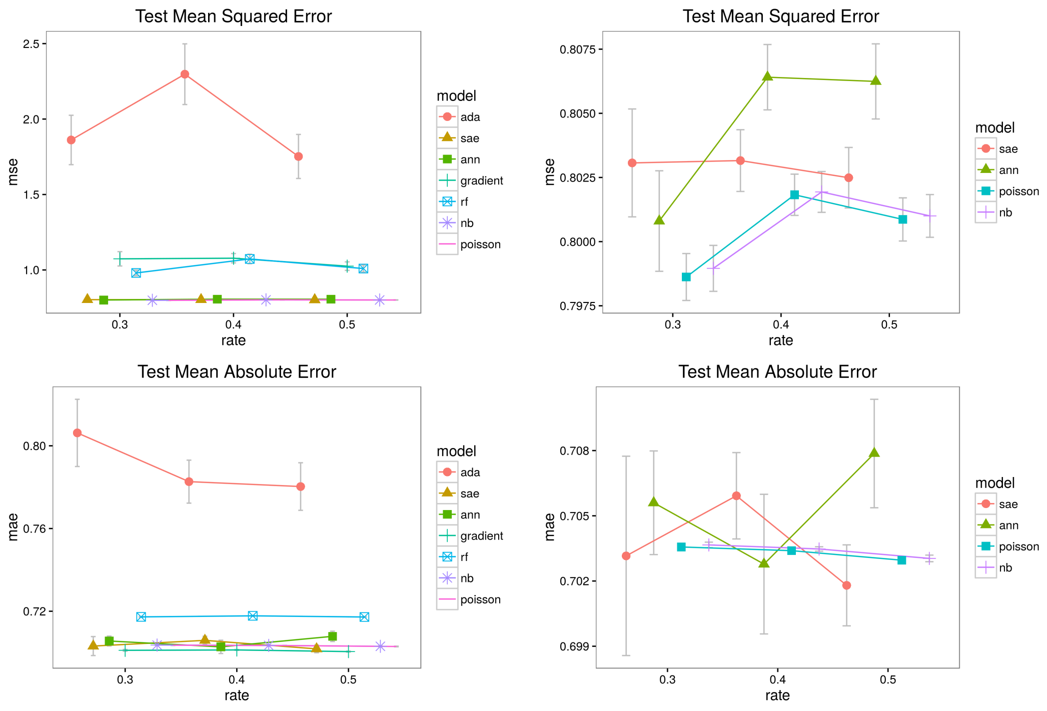


Figure 4.8: Overall regression models test MSE/MAE with different sampling rates

4.6 Feature Importances

Now we look at the feature importances predicted by the top 2 best regression models for our data - Poisson regression and Negative Binomial regression. As Figure 4.9 shows where error bars represent the standard error of the mean, the top 5 most important features are $tfbs\_count$, dhs_max_score, dhs_src_count, tss_dist, and $gerp\_score$ which are consistent with the recent literature findings that the "driver" noncoding mutations most likely to be located in the regula-

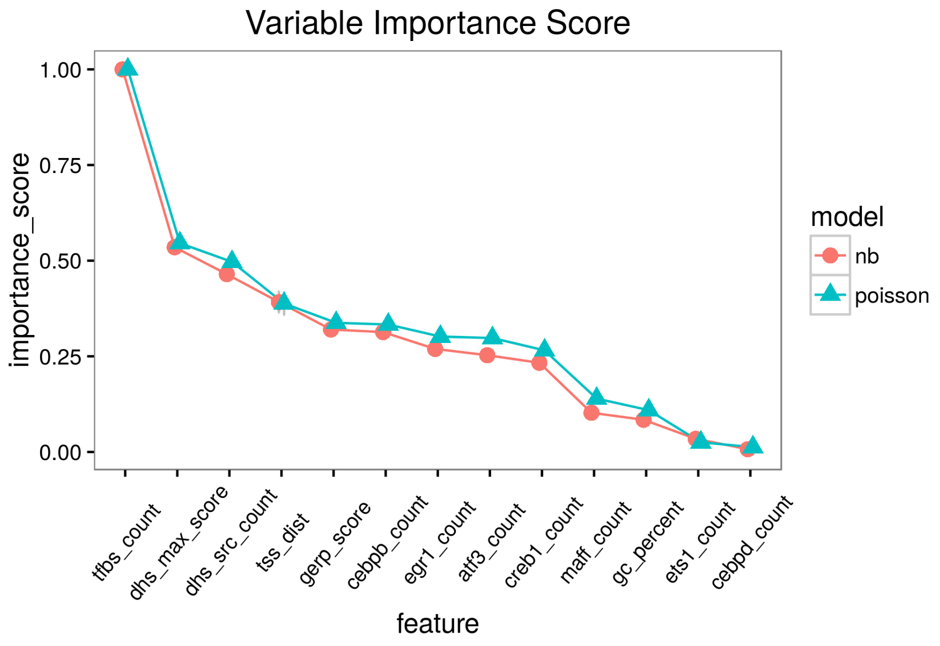


Figure 4.9: Variable importances score

tory region and close to the transcription start site. And also both Poisson regression model and negative binomial regression model predict the same order of feature importances.

Chapter 5: Discussion

We have explored three different kinds of regression models for predicting local mutation recurrence counts for noncoding somatic mutations and compared the models by evaluating the models' mean_squared_error, men_absolute_error, explained_variance_score, and R^2_score. We found that the generalized linear models like Poisson regression and negative binomial regression perform better than other models and the GLM models are more stable over different sampling size of data. And we also found deep neural network regression looks promising as its test-set error is very close to those of the GLM models. However, there are several interesting questions that remain to be answered about this problem.

Is noncoding mutation data sufficient for learning? We have around 4.5 million noncoding mutations in total (80% for training and validation, and 20% for testing). From the volumn size, the answer should be yes according to machine-learning theory since we only have 13 features in total (i.e., $n \gg p$). However, looking at the mutation recurrence (response variable) distribution, it is mainly focused on small regions of numbers. The whole range of the recurrence variable is between 1 and 617, but most ($> 90\%$) values are less than 50 especially the value between 50 or 600 are missing, as can be seen in Figure 3.1. However, we can not just remove or exclude those "rare" mutations with high frequency (> 600) because those mutations contain the most popular non-coding driver mutations like the promoter mutation of the *TERT* gene. Meanwhile the number of features is small, and likely does not include all possible genomic and epigenomic correlates of mutation recurrence. Therefore, in the future we may need to extract more genomic features for better regression performance.

On the other hand, even for the best model—Poisson regression—the absolute prediction power is still not strong since the difference between the prediction error on the real feature data and the error on the shuffled (random) feature data is small, especially the correlation metrics (explained_variance_score and R^2_score) between the ground-truth values and the predicted values are not good as expected. Generally for these two values, the larger the better. And if the value is close to zero, it indicates the model always predicts the mean value of the response variable while the minus value means the model is very bad for the data. Although the positive explained_variance_score and R^2_score of the Poisson regression model are positive for the test-set data, they are still less than 0.5. However, for GLM regression, the goodness-of-fit of the model is usually evaluated by the deviance and the chi-squared test, which look good for our model as the deviance is decreased by 0.26% compared to the null hypothesis model where all the coefficients are one.

Additionally, we observed that the local mutation recurrence count statistic that we have defined is overdispersed, with a dispersion value of $\sim$1.3. In this case, the Negative Binomial (NB) regression model would be expected to achieve better prediction accuracy. For the noncoding mutation recurrence data, the NB model performance is very close to the Poisson model, with the latter being slightly better. One possible reason is that all mutation recurrence is at least 1 while the NB model starts from 0 which is the most frequent. In future work, this conjecture could be tested by applying a Zero-Truncated Negative Binomial regression as we already discussed in the previous sections.

Chapter 6: Conclusion

In this thesis we describe our investigation of a regression-based approach for noncoding driver mutation detection in cancer. We implemented and compared multiple models for predicting noncoding mutation recurrence under the principle that mutations with higher recurrence frequencies are more likely to be driver mutations. Our model applies to somatic mutations in noncoding genome, which includes intergenic regions and the 5' regulatory regions for genes. For each mutation, we extracted 29 genomic features, of which we used 13 features for regression analysis. We used a sliding window approach to compute the local mutation recurrence count, which in turn constituted the dependent variable for the regression. We fit the combined feature and count data matrix with seven different regression models in three categories (generalized linear models, ensemble decision trees, and the deep neural network): Poisson regression, Negative Binomial regression, Random Forest regression, Gradient Boosting tree regression, the decision tree regression with AdaBoost, the artificial neural network regression and stacked AutoEncoder regression. We compared these models by evaluating their mean_squared_error, mean_absolute_error, explained_variance_score, and R^2_score.

From comparing the regression models' prediction performance, we reached the following conclusions. First, the Poisson and Negative Binomial regression models perform the best for the noncoding mutation recurrence data, which is consistent with the existing research findings of counts data regression analysis in ecology and economics areas where the counts data is prevalent and the Poisson and Negative Binomial models are widely applied. Second, the neural network regression models, namely feed-forward neural network and stacked AutoEncoder, are very promising for noncoding mutation recurrence regression analysis. The performance achieved by ANN and SAE is very close to Poisson even only small part of data is applied. It is well-known that for deep neural network training, more data would be expected to further improve performance. Third, conventional machine learning regression models may not be well-suited to problems where the dependent variable is an integer count. As we defined, the noncoding mutation recurrence is the count number of mutations within the 101 bp window centered at one mutation. Among the models we studied in this thesis, including Random Forest regression, Gradient Boosting Trees regression, and the Decision Tree regression with AdaBoost, the

AdaBoost performs the worst and almost has no predictive power on this regression problem. The Random Forest model and the gradient boosting model still have some predictive power. Whether they could achieve better performance with additional tuning is an open question.

Last but not the least, the feature (variable) importances predicted by the regression models is consistent with the current understanding of genomic and epigenomic covariates for recurrent noncoding mutations, that the "hot" (more likely to be driver) noncoding mutations are located in the gene regulatory region and close to the transcription start sites. From the rank of the predicted feature importances, the feature $tfbs\_count$ (number of binding sites of transcription factors within the window centered at the mutation) has the highest score and the feature tss_dist (distance to the closest transcription start sites) is also on the top, which suggests these two features are very important for noncoding mutation driver detection.

To sum up, GLM regression models and deep neural network models are two promising—and highly distinct—model classes for noncoding mutation recurrence prediction. With more features incorporated in the near future, better performance can be achieved then the regression model that we learned can be very helpful for noncoding driver mutations detection.

Bibliography

[1] Ludmil B Alexandrov, Serena Nik-Zainal, David C Wedge, Samuel AJR Aparicio, Sam Behjati, Andrew V Biankin, Graham R Bignell, Niccolò Bolli, Ake Borg, Anne-Lise Børresen-Dale, et al. Signatures of mutational processes in human cancer. *Nature*, 500(7463):415–421, 2013.

[2] Orli Bahcall. Funseq for cancer genomics. *Nature genetics*, 45(11):1273–1273, 2013.

[3] S Bamford, E Dawson, Simon Forbes, J Clements, R Pettett, A Dogan, A Flanagan, J Teague, P Andrew Futreal, MR Stratton, et al. The cosmic (catalogue of somatic mutations in cancer) database and website. *British journal of cancer*, 91(2):355–358, 2004.

[4] David Barber. *Bayesian reasoning and machine learning*. Cambridge University Press, 2012.

[5] Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, James Bergstra, Ian J. Goodfellow, Arnaud Bergeron, Nicolas Bouchard, and Yoshua Bengio. Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.

[6] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, June 2010. Oral Presentation.

[7] Alan P Boyle, Eurie L Hong, Manoj Hariharan, Yong Cheng, Marc A Schaub, Maya Kasowski, Konrad J Karczewski, Julie Park, Benjamin C Hitz, Shuai Weng, et al. Annotation of functional variation in personal genomes using regulomedb. *Genome research*, 22(9):1790–1797, 2012.

[8] Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

[9] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[10] A Colin Cameron and Pravin K Trivedi. *Regression analysis of count data*, volume 53. Cambridge university press, 2013.

[11] Antoni B Chan and Nuno Vasconcelos. Bayesian poisson regression for crowd counting. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 545–551. IEEE, 2009.

[12] SW Cheetham, F Gruhl, JS Mattick, and ME Dinger. Long noncoding rnas and the genetics of cancer. *British journal of cancer*, 108(12):2419–2425, 2013.

[13] Feixiong Cheng, Junfei Zhao, and Zhongming Zhao. Advances in computational approaches for prioritizing driver mutations and significantly mutated genes in cancer genomes. *Briefings in bioinformatics*, page bbv068, 2015.

[14] Arul M Chinnaiyan and Nallasivam Palanisamy. Chromosomal aberrations in solid tumors. *Prog Mol Biol Transl Sci*, 95:55–94, 2010.

[15] Franois Chollet. keras. https://github.com/fchollet/keras, 2015.

[16] Adam Coates, Andrew Y Ng, and Honglak Lee. An analysis of single-layer networks in unsupervised feature learning. In *International conference on artificial intelligence and statistics*, pages 215–223, 2011.

[17] ENCODE Project Consortium et al. An integrated encyclopedia of dna elements in the human genome. *Nature*, 489(7414):57–74, 2012.

[18] Abhijit Dasgupta, Yan V Sun, Inke R König, Joan E Bailey-Wilson, and James D Malley. Brief review of regression-based and machine learning methods in genetic epidemiology: the genetic analysis workshop 17 experience. *Genetic epidemiology*, 35(S1):S5–S11, 2011.

[19] Olgert Denas and James Taylor. Deep modeling of gene expression regulation in an erythropoiesis model. In *Representation Learning, ICML Workshop*. Citeseer, 2013.

[20] Harris Drucker. Improving regressors using boosting techniques. In *ICML*, volume 97, pages 107–115, 1997.

[21] Rasool Fakoor, Faisal Ladhak, Azade Nazi, and Manfred Huber. Using deep learning to enhance cancer diagnosis and classification. In *Proceedings of the ICML Workshop on the Role of Machine Learning in Transforming Healthcare. Atlanta, Georgia: JMLR: W&CP*, 2013.

[22] Nils J Fredriksson, Lars Ny, Jonas A Nilsson, and Erik Larsson. Systematic analysis of noncoding somatic mutations and gene expression alterations across 14 tumor types. *Nature genetics*, 46(12):1258–1263, 2014.

[23] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[24] Manuel García-Magariños, Inaki López-de Ullibarri, Ricardo Cao, and Antonio Salas. Evaluating the ability of tree-based methods and logistic regression for the detection of snp-snp interaction. *Annals of human genetics*, 73(3):360–369, 2009.

[25] Andrew J Gentles and Daniel Gallahan. Systems biology: confronting the complexity of cancer. *Cancer research*, 71(18):5961–5964, 2011.

[26] Florian Gnad, Albion Baucom, Kiran Mukhyala, Gerard Manning, and Zemin Zhang. Assessment of computational methods for predicting the effects of missense mutations in human cancers. *BMC genomics*, 14(Suppl 3):S7, 2013.

[27] John M Goldman and Junia V Melo. Chronic myeloid leukemia?advances in biology and new approaches to treatment. *New England Journal of Medicine*, 349(15):1451–1464, 2003.

[28] Oscar González-Recio and Selma Forni. Genome-wide prediction of discrete traits using bayesian regressions and machine learning. *Genet. Sel. Evol*, 43(7):21329522, 2011.

[29] Oscar González-Recio, Kent A Weigel, Daniel Gianola, Hugo Naya, and Guilherme JM Rosa. L2-boosting algorithm applied to high-dimensional problems in genomic selection. *Genetics research*, 92(03):227–237, 2010.

[30] William H Greene. *Econometric analysis*. Pearson Education India, 2003.

[31] Adrian D Haimovich. Methods, challenges, and promise of next-generation sequencing in cancer biology. *Yale J Biol Med*, 84(4):439–46, 2011.

[32] T Hastie, R Tibshirani, and J Friedman. The elements of statistical learning 2nd edition, 2009.

[33] Hatena. Random forest models., 2016.

[34] Rhys Heffernan, Kuldip Paliwal, James Lyons, Abdollah Dehzangi, Alok Sharma, Jihua Wang, Abdul Sattar, Yuedong Yang, and Yaoqi Zhou. Improving prediction of secondary structure, local backbone angles, and solvent accessible surface area of proteins by iterative deep learning. *Scientific reports*, 5, 2015.

[35] Momna Hejmadi. *Introduction to cancer biology*. Bookboon, 2010.

[36] Joseph M Hilbe. *Negative binomial regression*. Cambridge University Press, 2011.

[37] Kai-Lung Hua, Che-Hao Hsu, Shintami Chusnul Hidayati, Wen-Huang Cheng, and Yu-Jen Chen. Computer-aided classification of lung nodules on computed tomography images via deep learning technique. *OncoTargets and therapy*, 8, 2015.

[38] Gary B Huang, Honglak Lee, and Erik Learned-Miller. Learning hierarchical representations for face verification with convolutional deep belief networks. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 2518–2525. IEEE, 2012.

[39] Thomas J Hudson, Warwick Anderson, Axel Aretz, Anna D Barker, Cindy Bell, Rosa R Bernabé, MK Bhan, Fabien Calvo, Iiro Eerola, Daniela S Gerhard, et al. International network of cancer genome projects. *Nature*, 464(7291):993–998, 2010.

[40] Markus Kalisch. Applied multivariate statistics., 2012.

[41] W James Kent, Ann S Zweig, G Barber, Angie S Hinrichs, and Donna Karolchik. Bigwig and bigbed: enabling browsing of large distributed datasets. *Bioinformatics*, 26(17):2204–2207, 2010.

[42] Ekta Khurana, Yao Fu, Dimple Chakravarty, Francesca Demichelis, Mark A Rubin, and Mark Gerstein. Role of non-coding sequence variants in cancer. *Nature Reviews Genetics*, 17(2):93–108, 2016.

[43] VM Krasnopolsky, LC Breaker, and WH Gemmill. A neural network as a nonlinear transfer function model for retrieving surface wind speeds from the special sensor microwave imager. *Journal of Geophysical Research: Oceans*, 100(C6):11033–11045, 1995.

[44] Michael S Lawrence, Petar Stojanov, Paz Polak, Gregory V Kryukov, Kristian Cibulskis, Andrey Sivachenko, Scott L Carter, Chip Stewart, Craig H Mermel, Steven A Roberts, et al. Mutational heterogeneity in cancer and the search for new cancer-associated genes. *Nature*, 499(7457):214–218, 2013.

[45] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[46] Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 609–616. ACM, 2009.

[47] Honglak Lee, Peter Pham, Yan Largman, and Andrew Y Ng. Unsupervised feature learning for audio classification using convolutional deep belief networks. In *Advances in neural information processing systems*, pages 1096–1104, 2009.

[48] Ned Levine and II CrimeStat. A spatial statistics program for the analysis of crime incident locations. *Ned Levine and Associates, Houston, TX, and the National Institute of Justice, Washington, DC*, 2002.

[49] Cheng Li. A gentle introduction to gradient boosting., 2014.

[50] Marc R Mansour, Brian J Abraham, Lars Anders, Alla Berezovskaya, Alejandro Gutierrez, Adam D Durbin, Julia Etchin, Lee Lawton, Stephen E Sallan, Lewis B Silverman, et al. An oncogenic super-enhancer formed through somatic mutation of a noncoding intergenic element. *Science*, 346(6215):1373–1377, 2014.

[51] Anthony Mathelier, Wenqiang Shi, and Wyeth W Wasserman. Identification of altered cis-regulatory elements in human disease. *Trends in Genetics*, 31(2):67–76, 2015.

[52] Collin Melton, Jason A Reuter, Damek V Spacek, and Michael Snyder. Recurrent somatic mutations in regulatory regions of human cancer genomes. *Nature genetics*, 47(7):710–716, 2015.

[53] Microsoft. Neural network regression., 2016.

[54] Seonwoo Min, Byunghan Lee, and Sungroh Yoon. Deep learning in bioinformatics. *arXiv preprint arXiv:1603.06430*, 2016.

[55] John Mullahy. Specification and testing of some modified count data models. *Journal of econometrics*, 33(3):341–365, 1986.

[56] Sherry L Murphy, Jiaquan Xu, and Kenneth D Kochanek. Deaths: preliminary data for 2010. *National vital statistics reports: from the Centers for Disease Control and Prevention, National Center for Health Statistics, National Vital Statistics System*, 60(4):1–52, 2012.

[57] Andrew Ng, Jiquan Ngiam, Chuan Yu Foo, Yifan Mai, and Caroline Suen. Ufldl tutorial, 2012.

[58] J Guillermo Paez, Pasi A Jänne, Jeffrey C Lee, Sean Tracy, Heidi Greulich, Stacey Gabriel, Paula Herman, Frederic J Kaye, Neal Lindeman, Titus J Boggon, et al. Egfr mutations in lung cancer: correlation with clinical response to gefitinib therapy. *Science*, 304(5676):1497–1500, 2004.

[59] David Pardoe and Peter Stone. Boosting for regression transfer. In *Proceedings of the 27th international conference on Machine learning (ICML-10)*, pages 863–870, 2010.

[60] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[61] Dilmi Perera, Diego Chacon, Julie AI Thoms, Rebecca C Poulos, Adam Shlien, Dominik Beck, Peter J Campbell, John E Pimanda, and Jason WH Wong. Oncocis: annotation of cis-regulatory mutations in cancer. *Genome biology*, 15(10):1–14, 2014.

[62] Sergey M Plis, Devon R Hjelm, Ruslan Salakhutdinov, and Vince D Calhoun. Deep learning for neuroimaging: a validation study. *arXiv preprint arXiv:1312.5847*, 2013.

[63] Eduard Porta-Pardo, Luz Garcia-Alonso, Thomas Hrabe, Joaquin Dopazo, and Adam Godzik. A pan-cancer catalogue of cancer driver protein interaction interfaces. *PLoS Comput Biol*, 11(10):e1004518, 2015.

[64] Rajat Raina, Alexis Battle, Honglak Lee, Benjamin Packer, and Andrew Y Ng. Self-taught learning: transfer learning from unlabeled data. In *Proceedings of the 24th international conference on Machine learning*, pages 759–766. ACM, 2007.

[65] Benjamin J Raphael, Jason R Dobson, Layla Oesper, and Fabio Vandin. Identifying driver mutations in sequenced cancer genomes: computational approaches to enable precision medicine. *Genome Med*, 6(5), 2014.

[66] Lynn AG Ries, D Harkins, M Krapcho, Angela Mariotto, BA Miller, Eric J Feuer, Limin X Clegg, MP Eisner, Marie-Josèphe Horner, Nadia Howlader, et al. Seer cancer statistics review, 1975-2003. 2006.

[67] G Rodríguez. Lecture notes on generalized linear models., 2007.

[68] Kate R Rosenbloom, Joel Armstrong, Galt P Barber, Jonathan Casper, Hiram Clawson, Mark Diekhans, Timothy R Dreszer, Pauline A Fujita, Luvina Guruvadoo, Maximilian Haeussler, et al. The ucsc genome browser database: 2015 update. *Nucleic acids research*, 43(D1):D670–D681, 2015.

[69] Kate R Rosenbloom, Timothy R Dreszer, Jeffrey C Long, Venkat S Malladi, Cricket A Sloan, Brian J Raney, Melissa S Cline, Donna Karolchik, Galt P Barber, Hiram Clawson, et al. Encode whole-genome data in the ucsc genome browser: update 2012. *Nucleic acids research*, page gkr1012, 2011.

[70] Robert E Schapire, Yoav Freund, Peter Bartlett, and Wee Sun Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. *Annals of statistics*, pages 1651–1686, 1998.

[71] Zhiyuan Shen. Genomic instability and cancer: an introduction. *Journal of molecular cell biology*, 3(1):1–3, 2011.

[72] Søren Kaae Sønderby and Ole Winther. Protein secondary structure prediction with long short term memory networks. *arXiv preprint arXiv:1412.7828*, 2014.

[73] Matt Spencer, Jesse Eickholt, and Jianlin Cheng. A deep learning network approach to ab initio protein secondary structure prediction. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, 12(1):103–112, 2015.

[74] Peter D Stenson, Matthew Mort, Edward V Ball, Katy Howells, Andrew D Phillips, NS Thomas, David N Cooper, et al. The human gene mutation database: 2008 update. *Genome Med*, 1(1):13, 2009.

[75] Dmitry Svetlichnyy, Hana Imrichova, Mark Fiers, Zeynep Kalender Atak, and Stein Aerts. Identification of high-impact cis-regulatory mutations using transcription factor specific random forest models. *PLoS Comput Biol*, 11(11):e1004590, 2015.

[76] Youmin Tang. Geophysical data analysis., 2015.

[77] W Venables and B Ripley. Modern applied statistics using s, 2002.

[78] João Vinagre, Ana Almeida, Helena Pópulo, Rui Batista, Joana Lyra, Vasco Pinto, Ricardo Coelho, Ricardo Celestino, Hugo Prazeres, Luis Lima, et al. Frequency of tert promoter mutations in human cancers. *Nature communications*, 4, 2013.

[79] Axel Visel, Edward M Rubin, and Len A Pennacchio. Genomic views of distant-acting enhancers. *Nature*, 461(7261):199–205, 2009.

[80] V.Krasnopolsky. Nonlinear statistics and nns, 2006.

[81] Bert Vogelstein, Nickolas Papadopoulos, Victor E Velculescu, Shibin Zhou, Luis A Diaz, and Kenneth W Kinzler. Cancer genome landscapes. *science*, 339(6127):1546–1558, 2013.

[82] Nils Weinhold, Anders Jacobsen, Nikolaus Schultz, Chris Sander, and William Lee. Genome-wide analysis of noncoding regulatory mutations in cancer. *Nature genetics*, 46(11):1160–1165, 2014.

[83] John N Weinstein, Eric A Collisson, Gordon B Mills, Kenna R Mills Shaw, Brad A Ozenberger, Kyle Ellrott, Ilya Shmulevich, Chris Sander, Joshua M Stuart, Cancer Genome Atlas Research Network, et al. The cancer genome atlas pan-cancer analysis project. *Nature genetics*, 45(10):1113–1120, 2013.

[84] Rainer Winkelmann. *Econometric analysis of count data*. Springer Science & Business Media, 2013.

[85] Thomas W Yee. Vector generalized linear and additive models. 2007.

[86] Achim Zeileis, Christian Kleiber, and Simon Jackman. Regression models for count data in r. 2007.

[87] Jian Zhou and Olga G Troyanskaya. Predicting effects of noncoding variants with deep learning-based sequence model. *Nature methods*, 12(10):931–934, 2015.