Using Human Variation to Estimate Exonic Intolerance

Supervisor: Andrew Allen
Student: Shuaiqi Zhang

April 19, 2016

1 Introduction

The common variation is a variant can be found in most human populations, and a functional variation indicates some single nucleotide polymorphisms could manifest a Mendelian disease. Ranking human genes based on their tolerance to functional genetic variation can greatly facilitate patient genome interpretation. With this rank, we are able to select the genes having less tolerance to common function mutation, and thus predict if a common functional mutation will become less or even disappear in population due to the nature selection pressure. There are many approaches available attempting to prioritize mutations in terms of their prior probabilities of conferring risk of disease. Petrovski et al. (2013) [1] developed a gene-level assessment. It is a framework that ranks protein-coding genes based on their intolerance to functional variation, by comparing the overall number of observed variants in a gene to the observed common functional variants. The intolerance score itself is a measure of the deviation from this prediction. Specifically, the score is the standard residual of the simple linear regression between exonic common functional mutation and exonic variate sites, and it is termed as Residual Variation Intolerance Score(RVIS). The positive residual indicates this gene has more tolerance to the common functional mutation, while the negative residual indicates this gene has less tolerance. This approach is proved successful in prioritizing genes most likely to result in Mendelian disease. However, using the gene as the unit of analysis fails to represent the reality that these mutations can often cluster in particular parts of genes.

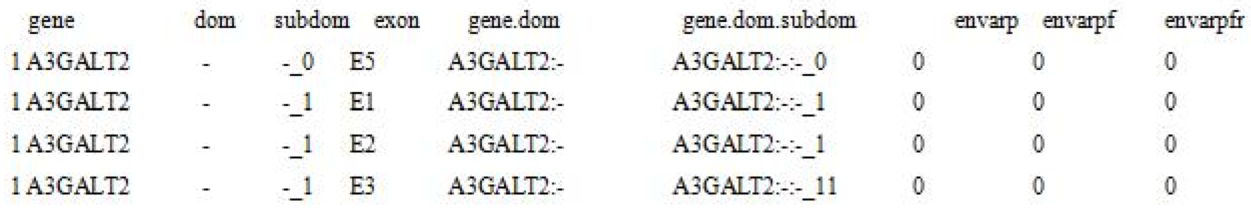
Ayal et al.(2016)[2] improved this method by evaluating the intolerance of gentic sub-regions using two biological sub-region classifications. In other words, the gene are divided into sub-regions and rank the resulting sub-regions by their intolerance to functional variation. Using the same approach, they calculated RVIS for the sub-region under each gene, and it is termed as subRVIS. It is shown that domain subRVIS, exon subRVIS are all significantly correlated with the presence or absence of pathogenic mutations within their corresponding regions. However, this model doesn't translate as well to some certain genes since some exons are identified as tolerant while they are in an intolerant gene. The reason for this is that when calculating the exon subRVIS, the information of genes are not shared to exons.

Here I develop a new score system, which is derived from the hierarchical models with structures of gene and exon, so that the genetic and exonic information can be included and shared in the score system. The result shows that the system doesn't lead to a situation where some exons in a intolerant gene are identified as tolerant, and thus it can provide better identification of intolerant exons.

2 Data

The data is based on ESP6000, which provides information for variants based on European American (EA) and African American (AA). The g "missense", "stop-gained", "missense-near-splice", "stop-lost", "splice-5", "splice-3", "stop-gained-near-splice", and "stop-lost-near-splice" as qualifying functional variates, while "coding-synonymous" and "coding-synonymous-near-splice" as "non-functional" variants. The threshold divides common and rare as 0.1% minor allele frequency (MAF). Y is the total number of common(MAF>0.1%) "functional" missense and "truncating" SNVs (including splice and nonsense) and let X be the total number of variants (including synonymous and "non-functional" missense mutations, regardless of frequency in the population) observed within a gene. [1]

The data set contains the name of the gene, domain, sub-domain and exon respectively, as well as the combination of gene, domain, sub-domain. In addition, it contains the number of variants (envarp) sites, the total number of functional mutation (envarpf), and the number of rare functional mutations(envarpfr). Below is a subset



	gene	dom	subdom	exon	gene.dom	gene.dom.subdom	envarp	envarpf	envarpfr
1	A3GALT2	-	-_0	E5	A3GALT2:-	A3GALT2:-:-_0	0	0	0
1	A3GALT2	-	-_1	E1	A3GALT2:-	A3GALT2:-:-_1	0	0	0
1	A3GALT2	-	-_1	E2	A3GALT2:-	A3GALT2:-:-_1	0	0	0
1	A3GALT2	-	-_1	E3	A3GALT2:-	A3GALT2:-:-_11	0	0	0

Specifically, the observed common functional mutation is the difference between the total number of functional mutation and the rare functional mutations. There are 18,596 different genes, 37,427 domains under gene level, 67,414 sub-domains under gene and domain levels, 233,740 exons under gene level in total.

3 Aims and Method

3.1 Aim 1: Producing models which contains information of gene and exon.

The hierarchical model is used to establish the relationship between variate sites(X) and common functional mutation(Y). As shown in Figure 1, the data set presents a nested structure, and hierarchical models provide a way to organize the lowest-level units into a hierarchy of successively higher-level units.

The model can be sectioned in the equations below:

$log(y_{ij} + 0.01) \sim N(\mu + \beta x_{ij} + \alpha_{[i]} + \alpha_{[ij]}, \sigma^2)$,where $\alpha_{[i]}$ and $\alpha_{[ij]}$ are the random intercepts at gene and exon level respectively and,
$\alpha_{[i]} \sim N(0, \sigma_\alpha)$,
$\alpha_{[ij]} \sim N(0, \sigma_{[i]})$, where

i =1,2,3....18597, i is the index of gene
j =1,2,3.....233740, j is the index of exon

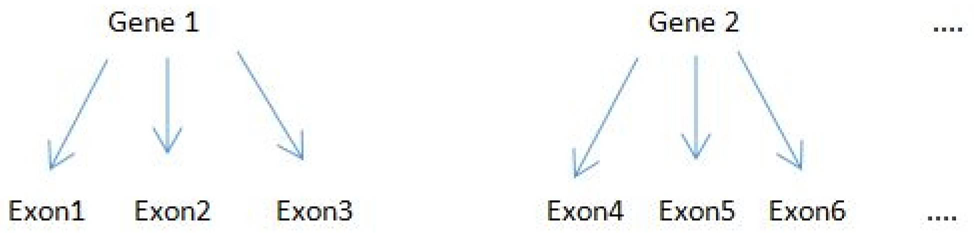


Figure 1: Data structure

y_{ij} is set to have log-normal distribution, since y_{ij} is count, and hence y_{ij} has constraint that it must be larger than 0. Taking log on y_{ij} expand the response variable to the whole real line, which make linear model reasonable.

Given that some exons have no common functional mutation ($y_{ij} = 0$), we could add a small number (0.01) to it before taking log.

For the hierarchical model, we have following assumptions:
1. Linearity and additivity of the relationship between dependent and independent variables.
2. Statistical independence of the errors.
3. Homoscedasticity of the errors.
4. Normality of the error distribution.

3.2 Aim 2: Deriving the score system to represent the exonic intolerance to the common functional mutation

If the exons has little variation in common functional mutation, I shrink the α_{ij} to 0 and use α_i as a common score for the exons. For example, ABCD3 is a "flat" gene since it has no common functional mutation in its 4 exons, then α_i is the common score for those 4 exons. And for those "non-flat" gene, $\alpha_{ij} + \alpha_j$ is the individual score for each exons. In order to do this, I shrink σ_is, so that α_{ij} can be shrinked automatically.

Various prior distributions have been suggested for scale parameters in hierarchical models. Andrew Gelman pointed out in Bayesian Analysis (2006)[3] that if variance of the random intercept has a weakly-informative prior inverse-gamma(ϵ, ϵ), then the conditional posterior distribution of this variance is also inverse-gamma. Therefore, the inverse-gamma is a conjugate prior of σ_i^2. Moreover, as ϵ getting bigger, the estimation of parameter rely more on the data set, while as ϵ getting smaller, there are more shrinkage on the variance. However, a difficulty of this prior distribution is that in the limit of $\epsilon \to 0$, it yields an improper posterior density, and thus ϵ must be set to a reasonable value. Unfortunately, for datasets in which low values of variances are possible, inferences become very sensitive to ϵ and the prior distribution hardly looks uninformative.

Because of this, I assign different values (0.05,0.1,0.5,1,10) on ϵ, so I can choose the best one. Moreover, I add a hyper prior on ϵ, which is $Unif(0, 10)$ since the hyper prior allows the prior of $\sigma_{[i]}^2$ to be modeled as a mixture of inverse gammas instead of forcing the prior to follow a given inverse gamma.

For other parameters of less interests, I choose the prior to be as flat as possible, such that the data set can decide the posterior distributions of these parameters. Particularly, the prior for μ and β is $N(0, 2500)$, for σ is $Unif(0, 20)$, and for σ_α is $Unif(0, 10)$.

The trace plot and Potential Scale Reduction Factor $\hat{R}$ is used to check the MCMC convergence for multiple chains. In particular, the trace plot tells if the chain has not yet converged to its stationary distribution, and it provides visual inspection on how well each chain is mixing. A chain might have reached stationarity if the distribution of points is not changing as the chain progresses. A chain that mixed well will traverses its posterior space rapidly, and it can jumpy from one remote region of the posterior to another in relatively few steps. The other method to check convergence is Potential Scale Reduction Factor $\hat{R}$. Especially, $\hat{R} = \sqrt{\frac{V\hat{ar}(\theta)}{W}}$, where $V\hat{ar}(\theta) = (1 - \frac{1}{n})W + \frac{1}{n}B$, n = number of iterations, W = within chain variance and B = between chain variance. As $n \to \infty$, the $V\hat{ar}(\theta)$ tends to be $(1 - \frac{1}{n})W$. Therefore, $\hat{R} = 1$ indicates a well mixed chain.

3.3 Algorithm and Computational efficiency

Stan is used to get the estimation for hierarchical models. Users specify log density functions in Stans probabilistic programming language and get full Bayesian statistical inference with Markov Chain Monte Carlo(MCMC) sampling, approximate Bayesian inference with variational inference and penalized maximum likelihood estimation with optimization.

Markov Chain Monte Carlo(MCMC) technique is a practical methods for high dimensional model analysis. It is a class of algorithms for sampling from a probability distribution based on constructing a Markov chain that has the desired distribution as its equilibrium distribution. Stan uses Hamiltonian Monte Carlo algorithm(HMC). Each iteration of the HMC has two steps. The first changes only the momentum(p); the second may change both position(q) and momentum(p). Both steps leave the canonical joint distribution of (q,p) in variant, and hence their combination also leaves the distribution invariant.

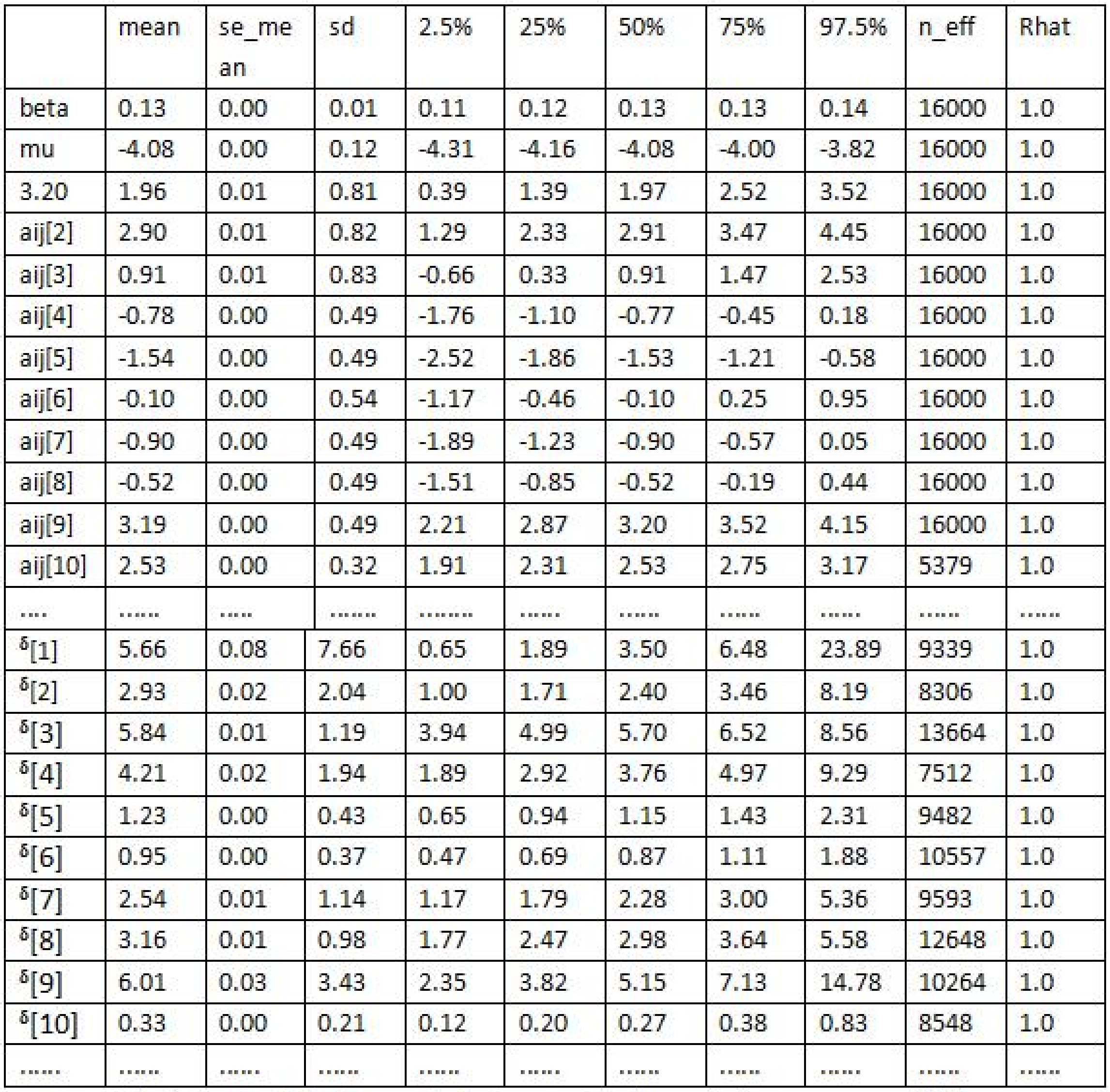
In the first step, new values for the momentum variables are randomly drawn from Gaussian distribution, which are independent of the current value of the position variables. In the second step, a Metropolis update with Hamiltonian dynamics is performed to propose a new state. Starting with the current state(q,p), Hamiltonian dynamics is simulated for L steps using reversible method that preserve volume, with a stepsize a. The momentum variables at the end of this L step are then rejected, giving a proposed state($\star p, \star q$). This proposed state is accepted as the next state of the Markov chain. If the proposed state is accepted, the next state is the same as the current state.

The advantages of MCMC are first, it is applicable even when we cant directly draw samples, second, it works for complicated distributions in high-dimensional spaces, even when we dont know where the regions of high probability are. Last but not least, it is relatively easy to implement and fairly reliable.

However, MCMC technique is notoriously compute intensive, with some analyses requiring weeks of CPU time on powerful computers. Therefore, the use of parallel computing technology is necessary, especially when the sample size is huge, such that multiple chains and calculations are carried out simultaneously.

4 Result

I use 2000 exons to get the primary result. There are 4 chains in HMC algorithm, each with 20000 iterations, 16000 warm ups, and 4000 post-warmup draws. I use parallel computing so that 4 chains can be carried out simultaneously. Rstan output provides mean estimator, four quantiles, neff and Potential Scale Reduction Factor $\hat{R}$ for each parameter. neff is a crude measure of effective sample size. The result shows that 20000 iterations can make 4 chains converge since the Potential Scale Reduction Factor $\hat{R} \approx 1$. Table 1 is a short cut of the result for the model using Inverse-Gamma(1,1) as a prior on $\sigma_{[i]}$



	mean	se_me an	sd	2.5%	25%	50%	75%	97.5%	n_eff	Rhat
beta	0.13	0.00	0.01	0.11	0.12	0.13	0.13	0.14	16000	1.0
mu	-4.08	0.00	0.12	-4.31	-4.16	-4.08	-4.00	-3.82	16000	1.0
3.20	1.96	0.01	0.81	0.39	1.39	1.97	2.52	3.52	16000	1.0
aij[2]	2.90	0.01	0.82	1.29	2.33	2.91	3.47	4.45	16000	1.0
aij[3]	0.91	0.01	0.83	-0.66	0.33	0.91	1.47	2.53	16000	1.0
aij[4]	-0.78	0.00	0.49	-1.76	-1.10	-0.77	-0.45	0.18	16000	1.0
aij[5]	-1.54	0.00	0.49	-2.52	-1.86	-1.53	-1.21	-0.58	16000	1.0
aij[6]	-0.10	0.00	0.54	-1.17	-0.46	-0.10	0.25	0.95	16000	1.0
aij[7]	-0.90	0.00	0.49	-1.89	-1.23	-0.90	-0.57	0.05	16000	1.0
aij[8]	-0.52	0.00	0.49	-1.51	-0.85	-0.52	-0.19	0.44	16000	1.0
aij[9]	3.19	0.00	0.49	2.21	2.87	3.20	3.52	4.15	16000	1.0
aij[10]	2.53	0.00	0.32	1.91	2.31	2.53	2.75	3.17	5379	1.0
....	......	.....	....	....	......	......	......	......	......	......
δ[1]	5.66	0.08	7.66	0.65	1.89	3.50	6.48	23.89	9339	1.0
δ[2]	2.93	0.02	2.04	1.00	1.71	2.40	3.46	8.19	8306	1.0
δ[3]	5.84	0.01	1.19	3.94	4.99	5.70	6.52	8.56	13664	1.0
δ[4]	4.21	0.02	1.94	1.89	2.92	3.76	4.97	9.29	7512	1.0
δ[5]	1.23	0.00	0.43	0.65	0.94	1.15	1.43	2.31	9482	1.0
δ[6]	0.95	0.00	0.37	0.47	0.69	0.87	1.11	1.88	10557	1.0
δ[7]	2.54	0.01	1.14	1.17	1.79	2.28	3.00	5.36	9593	1.0
δ[8]	3.16	0.01	0.98	1.77	2.47	2.98	3.64	5.58	12648	1.0
δ[9]	6.01	0.03	3.43	2.35	3.82	5.15	7.13	14.78	10264	1.0
δ[10]	0.33	0.00	0.21	0.12	0.20	0.27	0.38	0.83	8548	1.0
......	......	......	......	......	......	......	......	......	......	......

Table 1: Short cut for the result of rstan model when using inverse-gamma(1,1)

.

The Inverse-gamma prior yields shrinkage on σ_i^2. In particular, the smaller the scale value ϵ is, the more shrinkage it can produce. And this influence the posterior distribution of α_{ij}. Specifically, different ϵ values yield a shift and scale in posterior densities. Figure 2 shows the difference in posterior of α_{ij} for ANKRD45 exon 3.

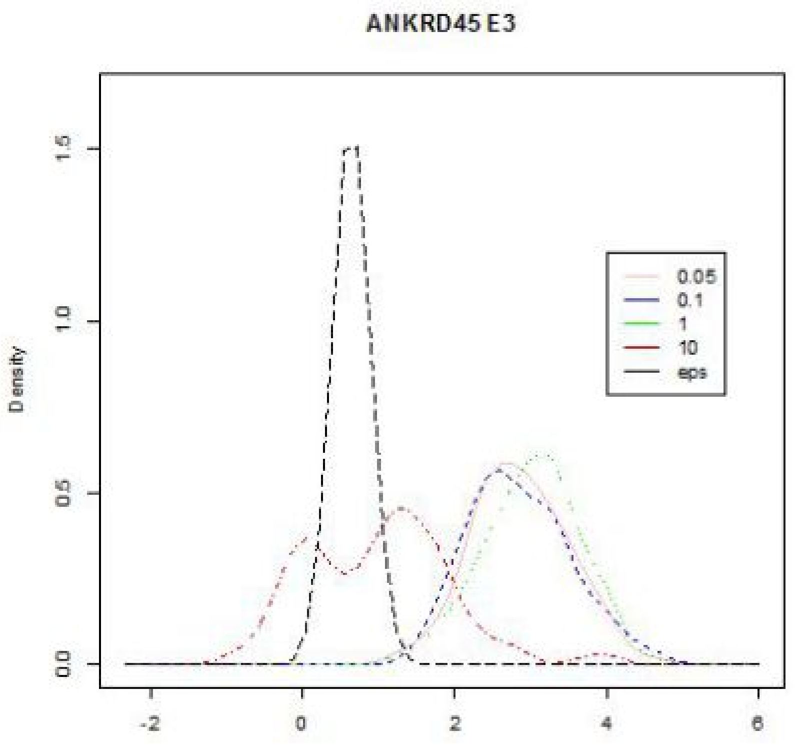


Figure 2: Density plot of exonic intercept for exon 3 in gene ANKRD45

In this project, I choose the inverse-gamma with hyper prior on ϵ as the desired prior, since we have little knowledge about the data, and the hyper prior allows the prior of σ_{ij}^2 to be a mixture of inverse gamma. Moreover, it needs a moderate computation time. Meanwhile, this prior produces an Inverse-Gamma posterior. Take $\sigma_{[76]}^2$ as example(Figure 3). The posterior density plot has a long left tail with a narrow scale.

When comparing the result with the data set, we can confirm that the model and prior produce a reasonable result. I set one as the cutoff, and those genes with $\sigma_i^2 \leq 1$ are pulled out. There are no common functional mutation in the exons in those genes, which means there is little variation in exons. Take ABCD3 as an example. The result shows the ABCD3 has a small variance, which is consistent with data set since it has no common functional muta-

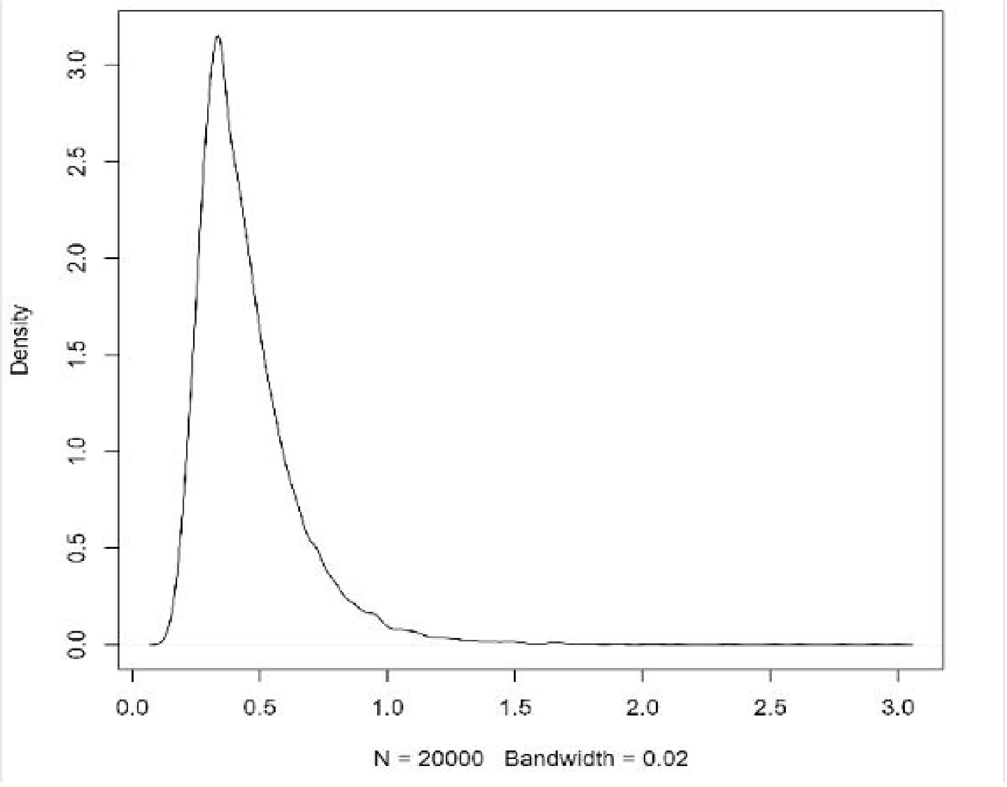
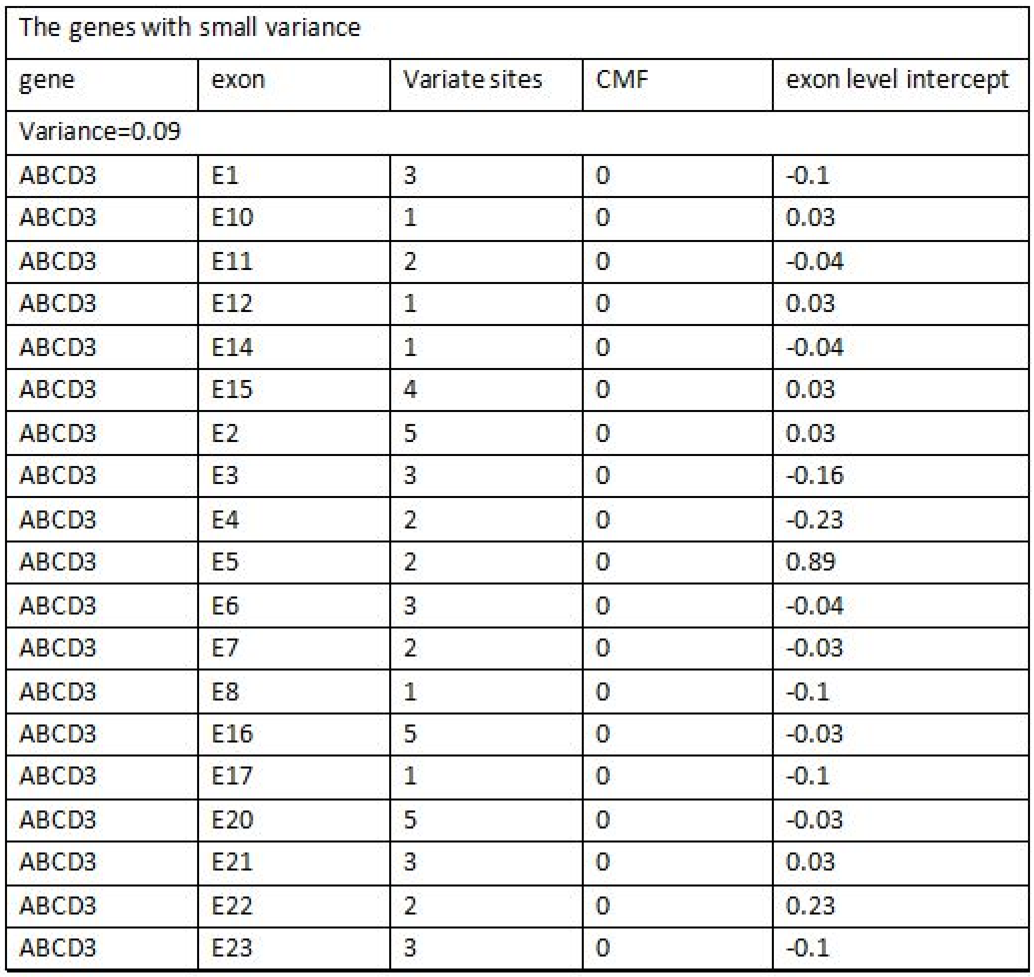


Figure 3: Density plot for $\sigma^2_{[76]}$ of gene ABCD3

tion in its exons. In this case, I can force α_{ij} in ABCD3 to be 0, and use $\alpha_i = -0.06$ to represent those exons.

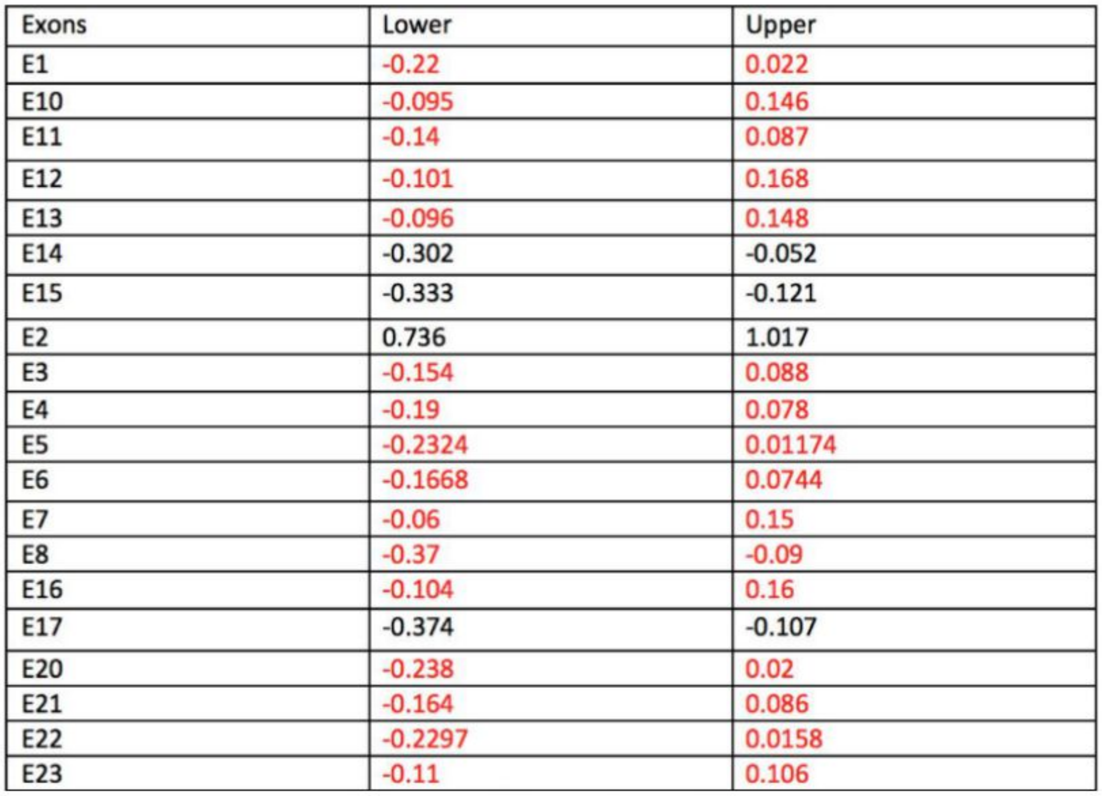


The genes with small variance				
gene	exon	Variate sites	CMF	exon level intercept
Variance=0.09				
ABCD3	E1	3	0	-0.1
ABCD3	E10	1	0	0.03
ABCD3	E11	2	0	-0.04
ABCD3	E12	1	0	0.03
ABCD3	E14	1	0	-0.04
ABCD3	E15	4	0	0.03
ABCD3	E2	5	0	0.03
ABCD3	E3	3	0	-0.16
ABCD3	E4	2	0	-0.23
ABCD3	E5	2	0	0.89
ABCD3	E6	3	0	-0.04
ABCD3	E7	2	0	-0.03
ABCD3	E8	1	0	-0.1
ABCD3	E16	5	0	-0.03
ABCD3	E17	1	0	-0.1
ABCD3	E20	5	0	-0.03
ABCD3	E21	3	0	0.03
ABCD3	E22	2	0	0.23
ABCD3	E23	3	0	-0.1

Table 3: The information of an example of gene with samll vairance

When looking into the 95% credible interval for α_{ij} in ABCD3, some intervals don't contain 0, which means there is variation in α_{ij}s.

Last but not least, the approach solves the problem that some exons are identified as tolerant while they are in an intolerant gene. For example, gene GNB1 and ABCC9 are known disease-causing genes. GNB1 has 7 exons, but none of them has common functional mutation. ABCC9 has 35 exons, only E27 and E17 has one common functional mutations. Therefore, these two genes and their exons are intolerant to the mutations. However, GNB1 Exon



Exons	Lower	Upper
E1	-0.22	0.022
E10	-0.095	0.146
E11	-0.14	0.087
E12	-0.101	0.168
E13	-0.096	0.148
E14	-0.302	-0.052
E15	-0.333	-0.121
E2	0.736	1.017
E3	-0.154	0.088
E4	-0.19	0.078
E5	-0.2324	0.01174
E6	-0.1668	0.0744
E7	-0.06	0.15
E8	-0.37	-0.09
E16	-0.104	0.16
E17	-0.374	-0.107
E20	-0.238	0.02
E21	-0.164	0.086
E22	-0.2297	0.0158
E23	-0.11	0.106

Table 4: Credible intervals for exonic intercept in gene ABCD3

4 and ABCC9 Exon 27 has high positive subRVIS, which indicates they are tolerant. Nevertheless, this situation doesn't happen in my model. GNB1 and ABCC9 has small σ_is, so I can use α_is to represent the exons in those two genes. Moreover, the α_i for those two genes are negative(-0.06,-0.07), and hence these two genes and their exons are intolerant to the common functional mutation. The 95% credible interval for α_{ij} of GNB1 Exon4 and ABCC9 Exon27 are (-0.064,0.226) and (-0.165,0.1012) respectively. But, the credible interval shows there is variation among α_{ij}s in ABCC9.

5 Conclusion

The primary result shows that the hierarchical model can produce a better score system than RVIS and subRVIS. It contains the genetic and exonic information, and allow different levels share information with each other. Therefore, the score can work better in selecting intolerance genes.

The next step is fitting the whole data set with this hierarchical model, and the prior for σ_i^2 is inverse-gamma(ϵ, ϵ), with hyper prior $Unif(0, 10)$ on ϵ. Since the credible intervals for α_{ij} show that there is variation among exonic intercepts. Thus, we can't force α_{ij} to be 0, instead, we will get the sampling distribution for $\alpha_i + \alpha_{ij}$, and use the mean or mode of this sample distribution as the score. In order to o assess whether the score can discriminate genes that do and do not cause disease, we need to compared the scores for genes causing different kinds of Mendelian diseases. We have six-lists reflecting different contexts: OMIM genes, haploinsufficiency, dominant-negative, de novo disease causing, recessive, and non-disease gene list. The logistic regression will be used to assess the relationship between the score and different disease-causing genes. Moreover, the ROC curve will be used to illustrate the capacity of the score to predict the OMIM gene lists.

The score derived from the hierarchical model improves the existed score in selecting intolerant genes, and it has obvious implications for human disease gene discovery. With this score system, we are able to select the mutation which will become less or disappear in population under the nature selection pressure.

References

[1] and Goldstein, D. *The intolerance to functional genetic varia Petrovski, S., Wang, Q., Heinzen, E., Allen, A., and Goldstein, D. Genic Intolerance to Functional Variation and the Interpretation of Personal Genomes. PLOS Genetics PLoS Genet,9(8). 2014*

[2] Ayal G.Petrovski, S., Wang, Q., Heinzen, E., Allen, A.,tion of protein domains predicts the localization of pathogenic mutations within genes. Genome Biology, 17:9. 2016

[3] Gelman,A. *Prior Distribution for Variance Parameters in Hierarchical Models.* Bayesian Analysis, 2006

[4] Hox,J *Multilevel Analysis: Techniques and Applications.* Mahwah, NJ: Lawrence Erlbaum Association,2002

6 Appendix

Code for the project:

6.1 Data Management

```
colnames(table)<- c("chr", "gene", "dom", "subdom", "exon",
                    "gene.dom",
                    "gene.dom.subdom",
                    "envarp",      # pass
                    "envarpf",     # pass functional
                    "envarpfr",    # pass functional rare
                    "emutr")       # mutation rate

table<-within(table,envarpfc<-envarpf-envarpfr)#y
table<-within(table,gene<-factor(gene))
table<-within(table,gene.dom<-factor(gene.dom))
table<-within(table,gene.dom.subdom<-factor(gene.dom.subdom))
table<-table[which(table$envarp!=0), ]
table$x=scale(table$envarp)
table<-table[1:1000,]
```

```
#get rid of genes with only one exon
gene<-as.numeric(table$gene) #list
genelevel<-length(unique(gene)) #number
indexg<-match(gene, unique(gene))  #list
leng=c()
len=1
for  ( i in 1:(length(indexg)-1)){
        if (indexg[i+1]==indexg[i]){
                len=len+1
           }
        if (indexg[i+1]!=indexg[i]){
                leng=c(leng,len)
                len=1
        }
}
leng=c(leng,len)
delet=which(indexg%in%which(leng%in%1))
#get the index of exon in gene which only contains one exon
table=table[-delet,]
# delet the row of genes contains only one exons


#for the use of counting number of gene
sumenvarp<-aggregate(table$envarp,
                    by=list(Category=table$gene),
                    FUN=sum)
sumenvarpfc<-aggregate(table$envarpfc,
                      by=list(Category=table$gene),
                      FUN=sum)[,2]
table1<-data.frame(cbind(sumenvarp,sumenvarpfc))
colnames(table1)<-c("gene","sumenvarp","sumenvarpfc")
```

6.2 Rstan code for the model

```
hiernormalinvg<-"
data{ #get the data set
int<lower=0> N;   # number of exon level
int<lower=0> J;    # number of gene level
int <lower=1,upper=J> gene[N];  #index of gene
int <lower=1,upper=N> exon[N];  #index of exon
vector[N] xij;    #x at exon level
vector[N] yij; #y at exon level
}
parameters{ #specify the parameter we want to know
real beta;  #common slope for the exon level
real mu;       #common intercept for the exon level
vector[N] aij; #random intercept for the exon level
real <lower=0> sigma_aj2[J];# variance of intercept at exon level
vector[J] aj; #random intercept for the gene level
real <lower=0> sigma_a;  #variance of intercept at gene level
real <lower=0> sigma; #variance of yij

}
transformed parameters{ #specify the model we will use

}
model { #give the prior distribution
   vector[N] lambdaij; #exon level model
    for (i in 1:N)
          lambdaij[i] <- mu+beta*xij[i]+aij[exon[i]]+aj[gene[i]];
   beta ~normal(0,50);
  mu~normal(0,50);
   sigma ~ uniform(0, 20);
   sigma_a ~uniform(0,10);
   sigma_aj2 ~inv_gamma(10,10);
   aj ~ normal(0, sigma_a);
   for (i in 1:N)
        aij[i]~ normal(0,sqrt(sigma_aj2[gene[i]]));
    yij ~ normal(lambdaij,sigma);
  }
 "
library("rstan")
J<-dim(table1)[1] #gene number
N<-dim(table)[1]  #exon number
xij=c(table$envarp)
```

```
yij=log(table$envarpfc+0.01)

gene<-as.numeric(table$gene) #list
genelevel<-length(unique(gene)) #number
indexg<-match(gene, unique(gene))  #list
exon<-c(1:length(table$envarpfc))
M1_table<-list(N=N, J=J, xij=xij,
yij=yij,gene=indexg, exon=exon)
control=list(adapt_delta=0.99,max_treedepth=12)
fitinv<-stan(model_code=hiernormalinvg,
            data=M1_table,iter=40000,
            warmup=35000,chains=4) #10,000 samplings
print(fitinv)

answer<-extract(fitinv,permuted=TRUE)
print(answer)


plotdes<-function(J,N){
        pdf(file = "inverse gamma (10,10)
        prior variance density plot.pdf")
        for (i in 1:J){
                plot(density(answer$sigma_aj2[,i]),
                main=c("density plot of exon-level variance",i))
                }
        for (i in 1:J){
        plot(density(answer$aj[,i]),
        main=c("density plot of gene-level intercept",i))
                }
        for (j in 1:N){
                plot(density(answer$aij[,j]),
            main=c("density plot of exon-level intercept",j))
                }
     plot(density(answer$beta),main="density plot of beta")
         plot(density(answer$mu),main="density plot of mu")
         plot(density(answer$sigma_a),main="density plot of sigma_a")
         plot(density(answer$sigma),main="density plot of sigma")
```

```
    dev.off()

}
plotdes(J,N)
```

6.3 Get the information for special exons

```
which(indexg%in%35)
which(indexg%in%44)
which(indexg%in%53)
which(indexg%in%63)
which(indexg%in%71)
which(indexg%in%73)
which(indexg%in%78)
which(indexg%in%82)
table[which(indexg%in%63),]
table[which(indexg%in%73),]
table[which(indexg%in%35),]
table[which(indexg%in%71),]
```

6.4 Posterior comparison for $\alpha_{[ij]}$

```
setwd("/Users/shuaiqizhang/Desktop/special/inversegamma(0.05,0.05)")
#large GENE 63
g005818<-read.table("818819.txt")$X..818.
g005819<-read.table("818819.txt")$X..819.
#large gene 73
g005893<-read.table("893894.txt")$X..893.
g005894<-read.table("893894.txt")$X..894.
#samll gene 35
g005483<-read.table("483.txt")$X..483.
g005484<-read.table("484.txt")$X..484.
#samll gene 73
g005876<-read.table("876877.txt")$X..876.
g005877<-read.table("876877.txt")$X..877.
```

```
setwd("/Users/shuaiqizhang/Desktop/special/inversegamma(0.1,0.1)")
#large GENE 63
g01818<-read.table("818819.txt")$X..818.
g01819<-read.table("818819.txt")$X..819.
#large gene 73
g01893<-read.table("893894.txt")$X..893.
g01894<-read.table("893894.txt")$X..894.
#samll gene 35
g01483<-read.table("483.txt")$X..483.
g01484<-read.table("484.txt")$X..484.
#samll gene 73
g01876<-read.table("876877.txt")$X..876.
g01877<-read.table("876877.txt")$X..877.



setwd("/Users/shuaiqizhang/Desktop/special/inversegamma(1,1)")
#large GENE 63
g1818<-read.table("818819.txt")$X..818.
g1819<-read.table("818819.txt")$X..819.
#large gene 73
g1893<-read.table("893894.txt")$X..893.
g1894<-read.table("893894.txt")$X..894.
#samll gene 35
g1483<-read.table("483484.txt")$X..483.
g1484<-read.table("483484.txt")$X..484.
#samll gene 71
g1876<-read.table("876877.txt")$X..876.
g1877<-read.table("876877.txt")$X..877.


setwd("/Users/shuaiqizhang/Desktop/special/inversegamma(10,10)")
#large GENE 63
g10818<-read.table("818819.txt")$X..818.
g10819<-read.table("818819.txt")$X..819.
#large gene 73
g10893<-read.table("893894.txt")$X..893.
g10894<-read.table("893894.txt")$X..894.
```

```
#samll gene 35
g10483<-read.table("483484.txt")$X..483.
g10484<-read.table("483484.txt")$X..484.
#samll gene 73
g10876<-read.table("876.txt")$X..876.
g10877<-read.table("877.txt")$X..877.


setwd("/Users/shuaiqizhang/Desktop/special/inversegamma(eps,eps)")
#large GENE 63
ge818<-read.table("818819.txt")$X..818.
ge819<-read.table("818819.txt")$X..819.
#large gene 73
ge893<-read.table("893894.txt")$X..893.
ge894<-read.table("893894.txt")$X..894.
#samll gene 35
ge483<-read.table("483484.txt")$X..483.
ge484<-read.table("483484.txt")$X..484.
#samll gene 71
ge876<-read.table("876.txt")$X..876.
ge877<-read.table("877.txt")$X..877.

library(sm)
#plot the density
s1<-rep(1,100)
s2<-rep(2,100)
s3<-rep(3,100)
s4<-rep(4,100)
s5<-rep(5,100)
group=c(s1,s2,s3,s4,s5)
exon483<-c(g005483,g01483,g1483,g10483,ge483) #gene 35 small
exon484<-c(g005484,g01484,g1484,g10484,ge484)
exon876<-c(g005876,g01876,g1876,g10876,ge876)  #gene 71 small
exon877<-c(g005877,g01877,g1877,g10877,ge877)

exon818<-c(g005818,g01818,g1818,g10818,ge818)    #gene 63 large
exon819<-c(g005819,g01819,g1819,g10819,ge819)
exon893<-c(g005893,g01893,g1893,g10893,ge893)  #gene 73 large
```

```
exon894<-c(g005894,g01894,g1894,g10894,ge894)

#large GENE 63
col=c("pink","blue","green","red","black")
sm.density.compare(exon818,group,col=col)
title(main="AMY1C  E6")
legend(5,1.2,c("0.05","0.1","1","10","eps"),col=col,
        lty=c(1,1,1,1,1))
kruskal.test(list(g005818,g01818,g1818,g10818,ge818))#*

sm.density.compare(exon819,group,col=col)
legend(5,1.2,c("0.05","0.1","1","10","eps"),col=col,
        lty=c(1,1,1,1,1))
title(main="AMY1C  E7")
kruskal.test(list(g005819,g01819,g1819,g10819,ge819))#*

#large gene 73
sm.density.compare(exon893,group,col=col)
legend(4,1.2,c("0.05","0.1","1","10","eps"),col=col,
       lty=c(1,1,1,1,1))
title(main="ANKRD45 E3")
kruskal.test(list(g005893,g01893,g1893,g10893,ge893))#*
ks.test(ge893,  g01893,exact=FALSE)#*

sm.density.compare(exon894,group,col=col)
legend(4,1.2,c("0.05","0.1","1","10","eps"),col=col,
        lty=c(1,1,1,1,1))
title(main="ANKRD45 E4")
kruskal.test(list(g005894,g01894,g1894,g10894,ge894))#*
ks.test(ge894,  g01894,exact=FALSE)#*

#small  gene 35
sm.density.compare(exon483,group,col=col)
legend(1.5,4,c("0.05","0.1","1","10","eps"),col=col,
        lty=c(1,1,1,1,1))
title(main="AGO3 E1")
kruskal.test(list(g005483,g01483,g1483,g10483,ge483))#*
ks.test(g005483,  g01483,exact=FALSE)#*
ks.test(g005483, g1483,exact=FALSE)#*
ks.test(g005483, g10483,exact=FALSE)#*
ks.test(g005483, ge483,exact=FALSE)#*
ks.test(g01483, g1483,exact=FALSE)#*
ks.test(g01483, g10483,exact=FALSE)#*
ks.test(g01483, ge483,exact=FALSE)#*
ks.test(g1483, g10483,exact=FALSE)#*
ks.test(g1483, ge483,exact=FALSE)#*
ks.test(g10483, ge483,exact=FALSE)#*

sm.density.compare(exon484,group,col=col)
legend(1.5,4,c("0.05","0.1","1","10","eps"),col=col,
       lty=c(1,1,1,1,1))
title(main="AGO3 E2")
kruskal.test(list(g005484,g01484,g1484,g10484,ge484))#*

#small gene 71
sm.density.compare(exon876,group,col=col)
legend(1.5,1.2,c("0.05","0.1","1","10","eps"),col=col,
       lty=c(1,1,1,1,1))
title(main="ANKRD34A E1")
kruskal.test(list(g005876,g01876,g1876,g10876,ge876))

sm.density.compare(exon877,group,col=col)
legend(1.5,1.2,c("0.05","0.1","1","10","eps"),col=col,
       lty=c(1,1,1,1,1))
title(main="ANKRD34A E1*")
kruskal.test(list(g005877,g01877,g1877,g10877,ge877))#*
```