IREGNET on CRAN
R Project for Statistical Computing
GSOC 2019

Aditya Samantaray
International Institute of Information Technology
Bhubaneswar, India
Computer Engineering

aditya.samantaray1@gmail.com
b517003@iiit-bh.ac.in

April 9, 2019

Contents

1 About the Project

- **Project :** IREGNET on CRAN
- **Description :** IREGNET is a package that fits an Accelerated Failure Time(AFT) model on all four types of censored data i.e. none, left, right as well as interval censored. It fits a linear model via elastic net penalized maximum likelihood using coordinate descent and supports gaussian, logistic as well as weibull distributions. The main objective of this year's project is to get iregnet on CRAN with proper testing and documentation.
- **Project Page URL :** `https://github.com/rstats-gsoc/gsoc2019/wiki/iregnet-on-CRAN`

2 About Me

My name is Aditya Samantaray and I'm currently pursuing a B.Tech degree in Computer Engineering from *International Institute of Information Technology*, *Bhubaneswar*, *India*.
As a person, I'm an enthusiastic guy who fell in love with open source and wants to be a part of something great. I spent a lot of time exploring many fields such as network security, dev ops, game development but have finally found solace in machine learning. In my spare time, I try to acquire new skills, learn about new technologies and their functioning.

2.1 Contact Information

- **Name :** Aditya Samantaray
- **Country :** India
- **Postal Address :** Sai Krupa C.H.S., Sector-27, Nerul(East), Navi Mumbai(400706), Maharashtra, India
- **Email :** aditya.samantaray1@gmail.com, b517003@iiit-bh.ac.in
- **LinkedIn :** `https://www.linkedin.com/in/aditya-samantaray/`
- **Github :** `https://github.com/theadityasam`

2.2 Affiliations

- **Institute :** International Institute of Information Technology, Bhubaneswar, India
- **Program :** B.Tech Computer Engineering
- **Course Period :** July, 2017 - May, 2021
- **Contacts to verify:**
  Anjali Mohapatra
  anjali@iiit-bh.ac.in
  Department of Computer Science Engineering, IIIT Bhubaneswar

2.3 Conflicts during the project period

Although currently I do not have any prior commitments for my summers, I have applied for internships in firms as well as institutes in case I'm unable to qualify for this project. I really want to work on open source but I also need to consider my profile and hence need some achievement or experience for this summer.
If I qualify for this project, I will dedicate all of my summer to it with no parallel schedules.

3 Mentor Details

- **Mentor Names :** Toby Dylan Hocking, Anuj Khare
- **Mentor Email ID's:**
  - Toby Dylan Hocking : tdhock5@gmail.com
  - Anuj Khare : khareanuj18@gmail.com

Have you been in touch with the mentors? When and how?
I have indeed contacted the mentors (sent the first mail on 13^{th} March, 2019 via Gmail) when I hit a roadblock during one of my tests and have gotten responses from both of them.
Anuj Khare gave a thumbs up to the tests and encouraged me to work on the proposal. I have described the error that I faced (due to my own mistake) and have also proposed a solution for it in the Test section.
He has also reviewed my proposal and suggested improvements for it.
Toby Dylan Hocking responded to me when I was stuck in one of my tests and helped me give an insight to the NAN's produced issue.

4 Coding Plan and Methods

To ensure that the work on the project proceeds smoothly, we need to break down the objectives of our project and give enough time to documentation and testing.

Main Objectives of the Coding Project
The main objective of this year's GSOC is to push iregnet into the mainstream i.e. on *The Comprehensive R Archive Network*(*CRAN*). Iregnet has already been coded by *Anuj Khare* in GSOC2016. Now it needs proper documentation and testing so that it passes all CRAN checks. We can break down this year's project into sub-parts.

- **Implementing cv.iregnet**
- **Testing the Code on various Datasets**
- **Make iregnet CRAN ready**
  - **Documenting all functions**
  - **Writing tests for the code**
  - **Closing the open issues**
  - **Make sure the package passes all build tests**
  - **Cleanup of code before pushing it to Github**
- **Writing the Vignette**
- **Creating a Reference Manual**

I've extensively studied the iregnet, glmnet as well as the survival package and then made this proposal accordingly. Any changes to improve it are welcome. I believe, I will be ready before the community bonding period to attempt this task.
I've elaborated all the points below:

4.1 Implementing cv.iregnet (PR#54)

cv.iregnet basically performs K-fold cross-validation on the dataset and finds out the optimal λ with the least cross-validation error. cv.iregnet has been worked upon by *Toby Dylan Hocking* and I believe the code is complete. For the cross validaton error metric, Log-Likelihood has been implemented and the function provides two λ selection criterion i.e. min (λ

with the lowest cross-validation error) and `1sd` (λ within one standard deviation of the minimum). The `S3 method` has also been implemented with `predict.cv.iregnet` and `plot.cv.iregnet`.

From the comments on `PR#54`, I realised that the road block is the "NAN's produced" error encountered, even for simple datasets. I did extensive research on why it occured in particular datasets. For *neuroblastomaProcessed*, fixing the error was as simple as removing the columns with zero variance.

```
library(iregnet)
library(testthat)
data("neuroblastomaProcessed",
                 package="penaltyLearning")
X = neuroblastomaProcessed$feature.mat
Y = neuroblastomaProcessed$target.mat
X = X[,apply(X,2,function(a){
        return(var(a)!=0)
        })]
cvresult <- cv.iregnet(X, Y, family = "gaussian",
                                        nfolds = 5L)
```

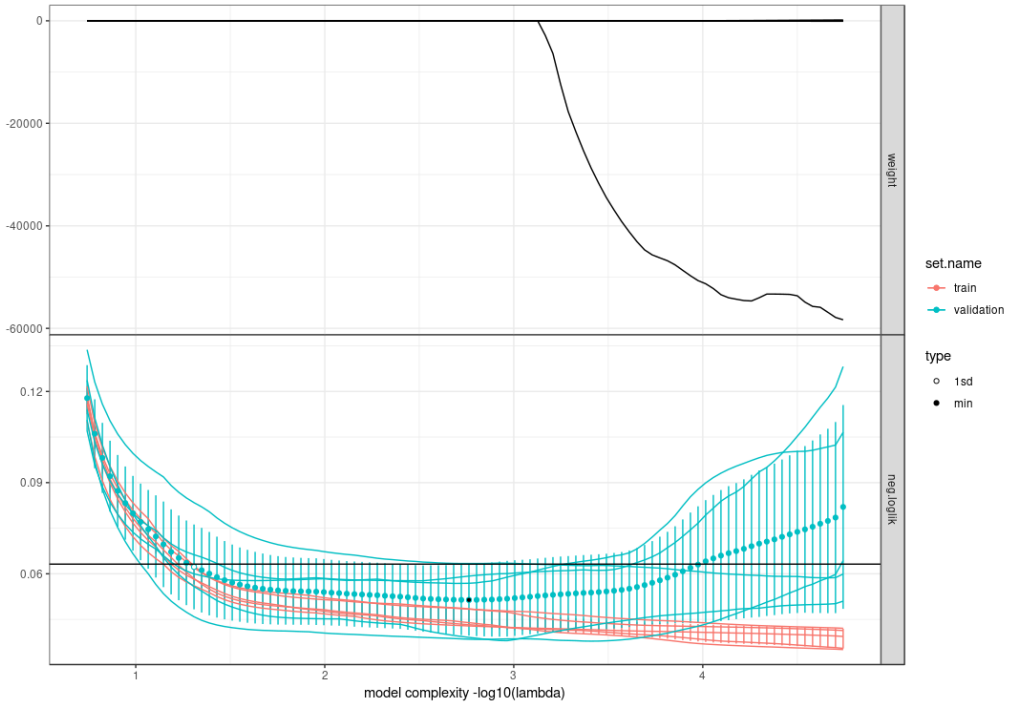

Figure 1: Regularization path plot for neuroblastomaProcessed

The model fits perfectly and also outputs the `min` and `1sd` λ.
But `cv.iregnet` failed for even a very simple dataset as shown like the one below:

```
toy.features <- matrix(rnorm(10), 5, 2)
toy.targets <- rbind( c(1, 2),
                      c(-Inf, 5),
                      c(1, Inf),
                      c(-Inf, 3),
                      c(-Inf, 4))
```

I tried various combinations of target matrices, I noticed that some gave error whereas others didn't. Now, at its core, `cv.iregnet` is splitting the dataset into **K-folds** and then calling `iregnet` K-times and finding out the validation error. Hence, I tried fitting `iregnet` for various types of randomly generated censored datasets like the one's below

```
x <- matrix(rnorm(10), 5, 2)
y <- rbind( c(1, 2),
            c(1, 2),
            c(3, 5),
            c(4, 6),
            c(7, 9))
y <- rbind( c(-Inf, 2),
            c(-Inf, 3),
            c(-Inf, 4),
            c(-Inf, 5),
            c(-Inf, 8))
y <- rbind( c(5, 10),
            c(6, Inf),
            c(9, Inf),
            c(3, 4),
            c(10, Inf))
```

`iregnet` fit on interval censored data but produced the NAN error on left and right censored data. Hence, we can infer that this is what's happening with `cv.iregnet`. As the dataset is split into K-folds, one of the folds is probable to be completely left censored or completely right censored. In our previous example(toy.features and toy.target), the dataset had just 5 observations, hence when we split it into 5 folds, 4 of the folds were either completely left censored or completely right censored. When I increased the sample size in that data, a mix of censored data was produced and hence `cv.iregnet` produced no error in that data while fitting.
Now, the question arises, how was `survreg` able to model that data but

iregnet couldn't?
To understand that, we need to analyze the differences between the two packages.
Iregnet uses a coordinate descent solver. At its core, the **newton method** implemented in the **IRLS solver** is as follows:

$$\beta = \widetilde{\beta} - H^{-1}\widetilde{g} \tag{1}$$

where g is the gradient of the likelihood and H is the **Hessian**, second derivate of loglikelihood. Note that we are trying to invert the Hessian during updation. I believe, this is where the NAN's are produced. For maximising the loglikelihood, a global maximum should exist i.e. the Hessian should be negative definite. Our Hessian might be non invertible and hence the NAN's are produced while trying to invert it.
Survreg also fits an **AFT** model but performs the optimization using the **Newton-Raphson method** which performs the following:

$$\beta_{n+1} = \beta_n - \frac{f(\beta_n)}{f'(\beta_n)} \tag{2}$$

The Newton-Raphson method uses an iterative process to approach one root of a function. The NAN error never occured in survreg as it never had to calculate the Hessian and invert it!

How do we fix it?
We saw that the problem is not exactly with cv.iregnet but with iregnet itself. The problem is more visible in cv.iregnet as it splits the dataset reducing the sample size, making it more prone to the conditions causing the error.
One possible fix that I can think of is implementing Hessian modifications techniques so that the search direction points to the correct optimum, without producing NAN.
We can also segregate the dataset into the left, right and interval censored data and then sample the data into the folds such that no fold is dominated by a single type of censorship. But this method is very naive and will increase the running overhead significantly for larger datasets.
Another method is to implement a first order method for optimization that gets executed when the NAN error occurs and tries to model the data.
I believe I need to discuss with mentors about this extensively and then work on the solution, I plan on discussing about it before the coding period, most probably during the community bondign period.

Also, I believe `cv.iregnet` is feature complete with S3 method and loglikelihood as error evaluation metric. In case the mentors feel that newer features can be implemented to the function, then I'll do the same.

4.2 Testing the Code on various Datasets

The code needs to be tested on various datasets of varying sizes and different types of interval censorships to check if they output the desired result. If the code doesn't work as expected, then I need to go back to the drawing board and debug it and write new code if required. I plan on giving this task ample time, atleast 2 weeks as we need our code to output correct results for different input conditions and types.
Some of the datasets on which the tests will be done are:

- **data(neuroblastomaProcessed)**, package: **penaltyLearning**
- **data(penalty.learning)**, package: **iregnet**
- **data(lung)**, package: **survival**
- **utils::data(MCLcleaned)**, package: **AdapEnetClass**

These are just some of the datasets, many more can be included for tests.

4.3 Make IREGNET CRAN ready

After the code has been tested on various datasets, I'll then work on getting the package ready to be pushed to *CRAN*. *CRAN* provides discoverability, ease of installation and a stamp of authenticity. Hence, the packages need to be well tested and documented before being submitted to the *CRAN* moderators.

- **Documenting all functions** :
  Although most of the functions have been documented, they will be reviewed once again. After documenting the functions, thankfully, the package `roxygen2` makes everything simple for creating the documentation.
  After all the `roxygen` blocks are created, we then run `devtools::document()` which will convert all the `roxygen` blocks to separate `.Rd` file for every function into the `"iregnet/man"` folder. It will also conveniently update the `NAMESPACE` file.

- **Writing tests for the code** :
  In order to verify that all the functions in the package do what they're intended to do, we need to test them on edge cases and error conditions to make sure that they are handled appropriately. Although, `R CMD CHECK` does test the functions with the examples provided in the documentation, for proper tests, I'll be using the `testthat` package for proper testing.
  Note : *CRAN* will run the tests on all major platforms and has a ceiling for test runtimes, hence tests that take over a minute will be handled with `skip_on_cran()` .

- **Closing the open issues** :
  Currently there are 6 open issues. I'll try to close issue

  - **#34** (Comparison with FISTA R code) while writing the tests. As *Toby Dylan Hocking* suggested, I'll add penalty.learning data set as a test data set for iregnet.
  - I'll close issue **#14** (Duplicate columns in covariate matrix) as it won't require any explicit time to be assigned for it.
  - I'll also try to close issue **#17** (Deviance) while implementing **`cv.iregnet`** .

  I'll consider closing rest of the issues after all the main objectives are achieved.

- **Make sure the package passes all build tests** :
  For the build tests, I'll be using **TravisCI** and for code coverage, I'll be using `covr` and **codecove**.

- **Cleanup of code before pushing it to `Github`** :
  After the code passes all the tests, and before pushing the source to `Github` , the code needs to be cleaned. By cleaning, I mean, removing all the unnecessary comments, removing the `roxygen` blocks and adding descriptive comments for the functions. This will make it easier for the newer developers to understand the source of the package. This will be done right before pushing the code and will take no time at all.

4.4 Writing the Vignette

I believe writing the vignette will be the most timetaking part of this project since it needs proper attention to the theory as well as should provide descriptive examples that are drafted and edited properly before being written.

The vignette will be written in **RMarkdown** and built with **knitr**.

Writing the vignette will take a solid 5 weeks of time. Since all pages are different(i.e. some pages will predominantly have figures, and some will have the code), I won't be giving a 5 page per day estimate. Rather I have divided the task into sections that will be written subsequently. The exact dates are mentioned in my timeline and can be referred to over there.

For generating the skeleton of the vignette, we'll be using the function `devtools::use_vignette("iregnet")` and then the following sections will be added.

- **Introduction** :
  An **Accelerated Failure Time** model or the **AFT** model is an alternative to the **Proportional Hazard Model** for generating predictive coefficients in **survival analysis modelling**. It is called failure time because the event of interest is usually death, disease, remission etc. A good **AFT** model often performs better than the corresponding **Cox** model.
  The **AFT** model describes a set of relation between survival probablities and a set of covariates. A standard **AFT** model is defined as:

$$\log(T_i) = \beta_0 + x_i^T \beta + \sigma \epsilon_i \tag{3}$$

  where x_i are the covariates, T_i is the observed time(output), σ is the scaling factor and ϵ_i is a residual term that assumes a specific distribution. In general, an **AFT** model is described as:

$$S(t/x) = S_0(e^{\beta' x} t), t \geqslant 0 \tag{4}$$

  where $S(t/x)$ is the survival function at time t and the $S_0(e^{\beta' x} t)$ is the baseline survival function at time t. The factor $e^{(\beta' x)}$ is the **acceleration factor**. The acceleration factor is the key measure of association obtained in the AFT model. It is a ratio of survival times corresponding to any fixed value of survival time.
  Also, the elastic net penalty is defined as follows:

$$\lambda P_\alpha(\beta) = \lambda(\alpha \sum_{j=1}^{m} \hat{\beta_j}^2 + \frac{(1-\alpha)}{2} \sum_{j=1}^{m} |\hat{\beta}_j|) \tag{5}$$

  For $\alpha = 1$, Lasso is performed and for $\alpha = 0$, Ridge is performed.

Hence the penalized least squares objective function is:

$$M = \sum_{i=1}^{n} \widetilde{w}_i(\widetilde{z}_i - \overline{x}_i^T\beta)^2 + \lambda P_\alpha(\beta) \tag{6}$$

The subderivative of the objective is taken and then three cases of β are obtained.
The coordinate descent algorithm will cycle through each β_j in turn, keeping others constant and then calculate the optimal value $\hat{\beta}_j$. This process is repeated till convergence. This section will be elaborated by refering to the following papers:

- `https://github.com/anujkhare/aft/blob/master/algorithm.pdf` by *Anuj Khare*
- `https://github.com/anujkhare/aft/blob/master/aft.pdf` by *Anuj Kare*
- `https://github.com/tdhock/aft-poster/blob/master/HOCKING-AFT.pdf` by *Toby Dylan Hocking*

As explained in Anuj Khare's "*RegularizedAFTmodelspaper*", the algorithm implemented by coordinate descent can be given as

1. Initialize $\widetilde{\beta}$.
2. Compute $\widetilde{w}_i$ and $\widetilde{z}_i$.
3. Find $\hat{\beta}$ by solving the penalized weighted least square problem defined in (6) using coordinate descent.
4. Set $\widetilde{\beta} = \hat{\beta}$
5. Repeat steps 2-4 until convergence of $\hat{\beta}$

- **Installation of the package** :
  This will include the methods of installation from *CRAN* or building from Github .

- **Quick Start** :
  Over here, I'll describe how to fit a generic model using **iregnet**. For uncensored data example, I'll choose the *Prostate* dataset, for fitting with survival object example, I'll consider the *Ovarian* dataset in the **survival** package and for an example with interval censored data, I'll select the *neuroblastomaProcessed* dataset from **penaltyLearning** package.

- **Fitting AFT Model** :
  Guide on fitting AFT models with different distributions in iregnet

- **Cross Validation** :
  Guide on model selection using cross-validation

- **Speed and Accuracy Comparisions** :
  A subset of this task has been performed in the easy test. The iregnet package will be compared to the glmnet, survreg, icfit, and AdapEnet-Class::WEnetCC.aft packages. For the computation time, I'll use the **microbenchmark** package. An example of the comparision between iregnet and glmnet has been performed in the *Easy Test* section of this proposal.

  For the goodness of fit test, we can perform the **log-rank** test on the models with the chi-square statistic. I'll then compute the AIC values and log-likelihood and compare the performance of the models. The Akaike Information Criterion (AIC) is defined as follows:

$$AIC = -2(log - likelihood) + 2(P + K) \tag{7}$$

  The model with the lower AIC value and a higher corresponding log-likelihood value will be a better fit.
  We can also use the Welch's t-test to test the hypothesis using the `t.test()` function.

If required, more sections will be added to the Vignette. Those sections will be worked upon by me during the buffer periods described in the timeline.

4.5 Creating the Reference Manual

The manual is created by default when `R CMD check` is run. The `DESCRIPTION` file is used to create the PDF reference manual. R internally creates a **Latex** document but deletes it after the PDF is created.
If the manual needs to be edited, `- -no-clean` will be added to the command, which will provide us with the `tex` file which can be then edited at will.

4.6 Development Environment

Following are the tools that I'll be using throughout the coding period. I've tried to cover everything that I might require for the

- **Operating System :** Arch Linux(KDE with i3WM)
- **Coding :** RStudio(for R), VSCode(primarily for $C++$)
- **Version Control :** Git + Github
- **Code Testing/Coverage:** testthat package, Travis-CI; covr package, codecov.io
- **Documentation :** roxygen2 package
- **Vignette :** RMarkdown, knitr package

5 Timeline

I'll keep on studying the iregnet package, understanding its structure before and during the community bonding period.
The coding period spans across 12 weeks, of which I plan on accomplishing the main objectives within 9 weeks and then keep on testing the package on different datasets as well as monitoring the package and fixing the issues if encountered in the next 3 weeks. The number of hours I'll work in a day is as follows:
Weekdays : 7 $Hours$ (9am-12am, 4pm-6pm, 9pm-11am)
Weekends : $5-6$ $Hours$ (Indefinite)
Two hours during weekends will mostly be used for the proposed improvements.
Effectively I'll be giving **minimum 45 hours a week** for this project. Roughly, for all of the weeks, I'll follow the following pattern:

- **Monday-Friday** : Work on the objectives defined below
- **Saturday** : Get the objectives reviewed by the mentor, take feedback and work on improving as per mentor's guidance
- **Sunday** : Buffer period, work on proposed improvement, plan for the next week

Although my next academic year will start in August, it won't be much of a problem as there won't be any exams till the coding period ends, hence it won't affect the work flow. Here's my proposed timeline that I will be following:

- **Before 27^{th} May : Self Study**

  - Familiarize myself with the package/codes
  - Discuss the strategies with the mentors
  - Start working on `cv.iregnet`

- 27^{th} **May -** 2^{nd} **June** :
  - Work on `cv.R`
  - Code the method to fix the "NAN" error

- 3^{rd} **June -** 9^{th} **June**
  - Test iregnet with different datasets
  - Write examples for the functions while testing the functions
  - If errors encountered, correct them

- 10^{th} **June -** 16^{th} **June**
  - Keep on testing the package with different datasets
  - Simultaneously document the functions/ write tests for the functions

- 17^{th} **June -** 23^{rd} **June**
  - Attempt fixing the issues described in the coding plan
  - Create the documentation and manual, edit the DESCRIPTION file

- 24^{th} **June -** 30^{th} **June**
  - Make sure the package passes all build tests on all platforms
  - Check that the package follows all CRAN policies
  - Final check with `R CMD check –as-cran`

- 1^{st} **July -** 7^{th} **July**
  - Start writing the vignette, start with the introduction
  - Explain AFT models
  - Calculate the likelihood/State the optimization objective

- 8^{th} **July -** 14^{th} **July**
  - Explain the elastic net penalty and coordinate descent method

  - Explain how to fit iregnet with different types of distributions

- 15^{th} **July -** 21^{st} **July**
  - Explain the model selection process with cross validation(cv.iregnet)
  - Speed and accuracy comparisions with various packages

- 22^{nd} **July -** 28^{th} **July**
  - Continuation of speed and accuracy comparisions with various packages
  - Finalize everything and submit to CRAN

- 29^{th} **July -** 19^{th} **August**
  - Final commit on Github, code for iregnet will be pushed after cleaning of the code(making it friendlier for aspiring contributors, described in the coding plan)
  - Run the package by installing from CRAN
  - Discuss with mentors about the features that can be added to iregnet
  - Work on adding support for sparse matrices
  - Work on a blog post about the tasks accomplished during the coding period

- 19^{th} **August -** 26^{th} **August**
  - Final week, done with everything! :)
  - Submission of project report

6 Management of Coding Project

While coding the Hessian modifications for `cv.iregnet`, I'll commit the improvements atleast twice a week. I'll keep the mentors updated about all the developments that I perform by mailing them twice a week. It is important to take their feedback before attempting any objectives so that the work proceeds smoothly without any hindrance. While writing the Vignette, I'll keep on committing every two days so that the mentors can see the progress of the Vignette. All of my methods will be transparent and will be strictly according to the timeline.

7 Test Results

I came across the iregnet package on 17^{th} of February and attempted the easy test on 20^{th} of February. I then got caught up with my mid semester examinations and hence attempted the medium test in the first week of March. I encountered an error in my code, wherein for some reason Nan's were produced. I finally completed the medium test on 16^{th} of March. I had learned a lot about iregnet as well as R programming in general within that week and hence when I looked at my easy test again, I found several methodological error, and hence attempted the easy test again. The next day itself, I completed the hard test and had successfully completed all the tests.

7.1 Easy Test

Perform a side-by-side comparison of iregnet and glmnet for a lasso problem with no censored data. Consider the prostate cancer data set, which has no censored data. Use the microbenchmark package to time the iregnet and glmnet functions. Do the two functions return the same result? Which is faster? Plot of time versus data set size. (one plot for rows and one plot for columns)

Old Easy Test
URL : `https://github.com/theadityasam/iregtest/blob/master/old_easy_test.md`
Initially I didn't have much knowledge of the regression libraries and their model fitting procedures. I also didn't have any experience with microbenchmark and hence the easy_test I attempted was pretty erroneous which I realised later. I won't include the old easy test as it adds nothing to this proposal, but it can be viewed by clicking on the given URL.
New Easy Test
URL : `https://github.com/theadityasam/iregtest/blob/master/easy_test.md`
Including the required libraries and the Prostate dataset

```
rm(list = ls())
library(glmnet)
library(iregnet)
library(directlabels)
library(ggplot2)
library(microbenchmark)
library(dplyr)
```

```
data("Prostate", package = "lasso2")
X = as.matrix(Prostate[, c(2:9)])#Feature matrix
#Since, the input to iregnet cannot be a one
#dimensional matrix, replicating column one
#into two
Y = matrix(c(Prostate[, 1],Prostate[, 1]),
    nrow = nrow(Prostate), ncol = 2)#Target matrix
colnames(Y)[c(1,2)] <- "lcalvol"
#Centering the data, not really required
#as the standardize parameter in iregnet is true
#by default
Y <- apply(Y, 2, function(y) y - mean(y))
X <- apply(X, 2, function(x) x - mean(x))
```

Fitting the data with iregnet and glmnet

```
irg <- iregnet(X, Y) #alpha is 1 by default
glm <- glmnet(X, Y[,1])
plot(glm)
plot(irg)
```

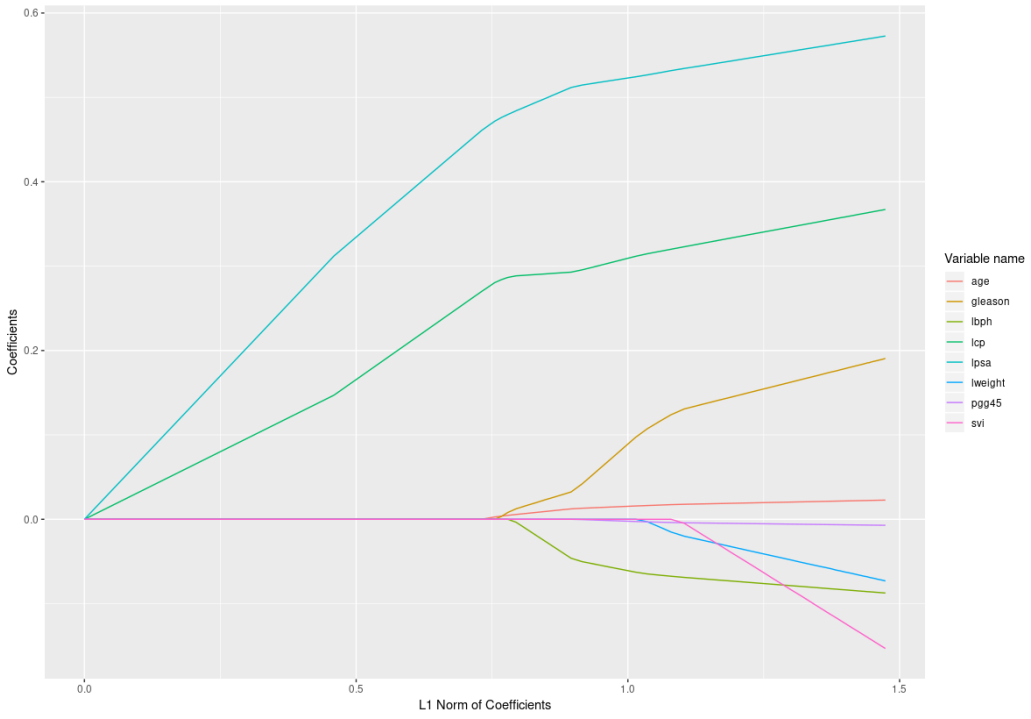

Figure 2: Lasso plot for iregnet

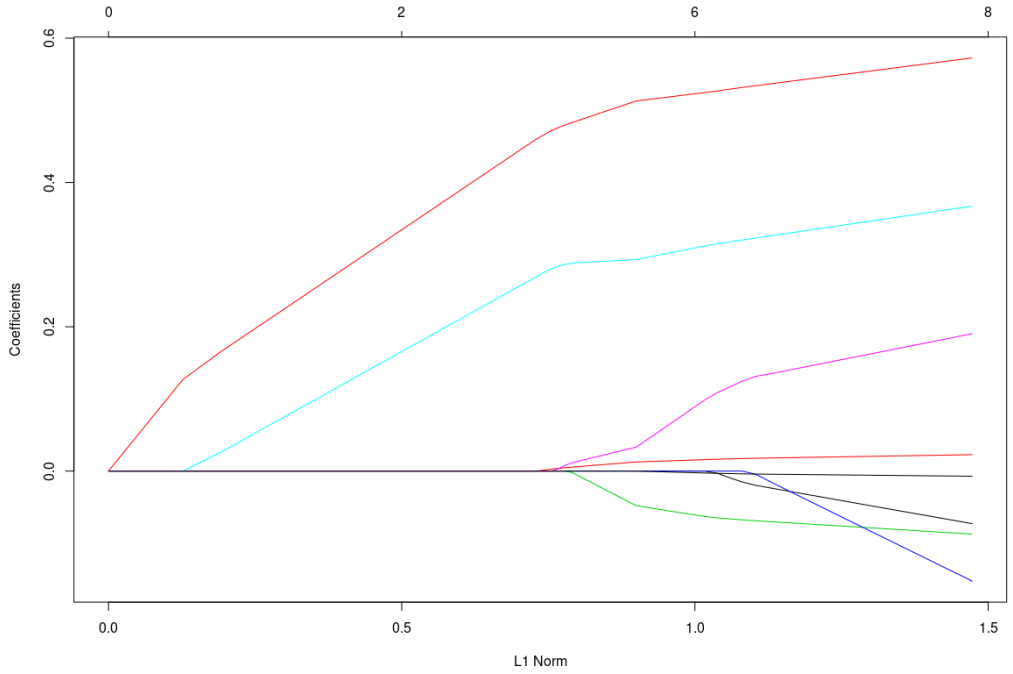

Figure 3: Lasso plot for glmnet

We see that the two plots obtained are almost similar, and will roughly output the same predictions.
Now, we'll try to evaluate the runtimes of both the libraries and compare their plots. For this test, we'll use the microbenchmark package and plot the results.

```
res <- data.frame() #Result data frame
for(i in 30:nrow(X))
{
  evaltime <- microbenchmark(iregnet(X[1:i,], Y[1:i,]),
                glmnet(X[1:i,], Y[1:i,1]), times = 100L)
  res <- bind_rows(res, data.frame(i,
        list(summary(evaltime)[,c('min','mean','max')])))
}
res <- cbind.data.frame(c("IREGNET", "GLMNET"), res)
names(res) <- c("expr", names(res)[2:5])
p <- ggplot(res, aes(x = i))+
  geom_ribbon(aes(ymin = min, ymax = max, fill = expr,
                              group = expr), alpha = 1/2)+
  geom_line(aes(y = mean, group = expr, colour = expr))+
```

```
  ggtitle('Runtime(in milliseconds) vs Dataset Size') +
  xlab('Dataset Size') +
  ylab('Runtime (in milliseconds)')
direct.label(p,"angled.boxes")
```

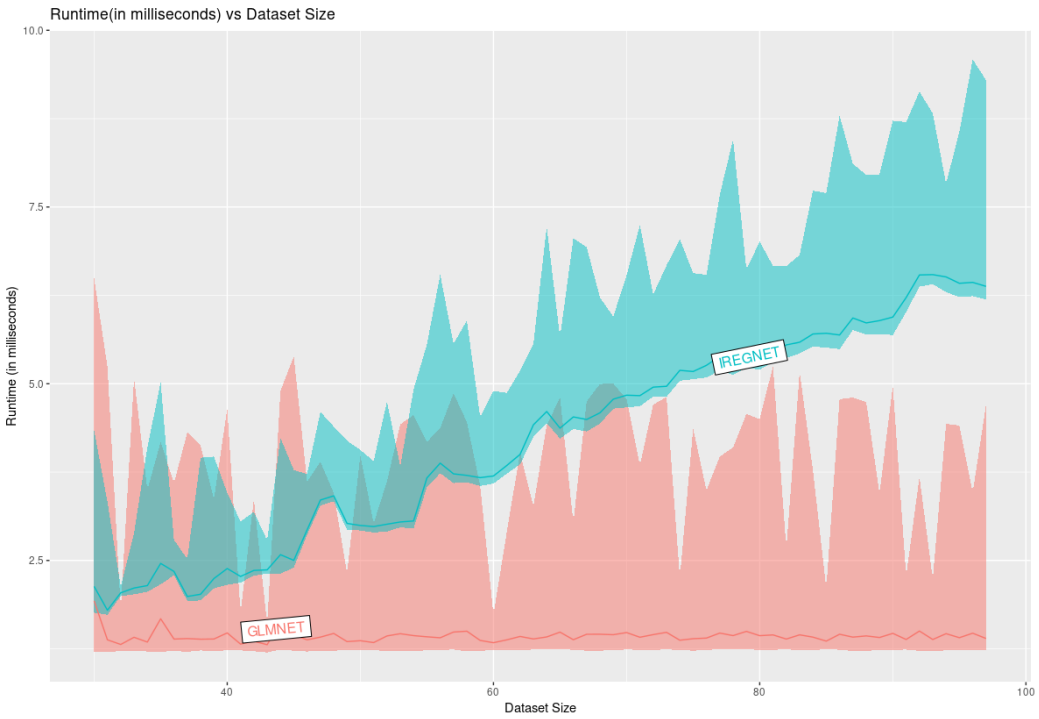

Figure 4: Runtimes Comparision of glmnet and iregnet

We see that glmnet performs significantly better (roughly six times) than iregnet as the dataset size increases

7.2 Medium Test

Use iregnet to fit a model on the ***penaltyLearning::neuroblastomaProcessed*** data.
Including the required library files and the neuroblastomaProcessed dataset and fitting the features with iregnet.

```
rm(list = ls())
library("iregnet")
library("glmnet")
data("neuroblastomaProcessed", package="penaltyLearning")
X = as.matrix(neuroblastomaProcessed$feature.mat)
```

```
Y = as.matrix(neuroblastomaProcessed$target.mat)
#Processing the data, removing the features with
#zero variance
X = X[,apply(X,2,function(x){
  return(var(x)!=0)
})]
#Fitting the model with iregnet
fitireg <- iregnet(X, Y)
plot(fitireg)
```

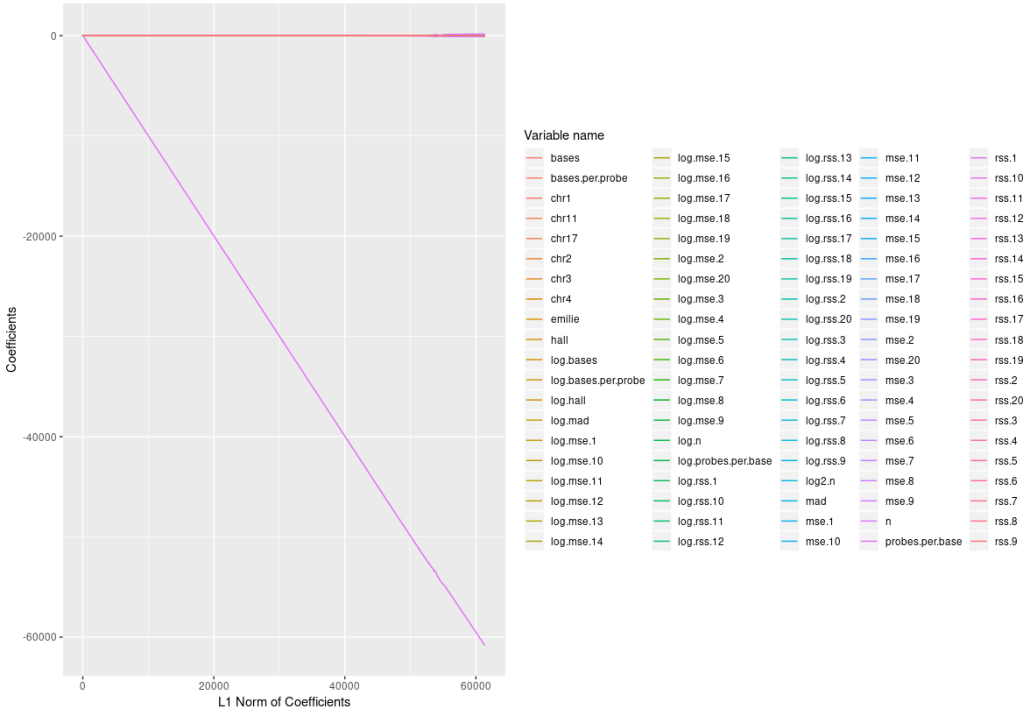

Figure 5: Lasso plot for neuroblastomaProcessed

One of the coefficient's magnitude is way more than the other coefficients, hence this plot is not very effective for evaluation. We remove this coefficient to obtain a cleaner lasso plot for the other coefficients.

```
#Finding the variable with the most negative value
plot(fit$beta[,100])
which.min(fit$beta[,100]
#Plotting
fit$beta <- fit$beta[-c(which.min(fit$beta[,100])),]
plot(fit)
```

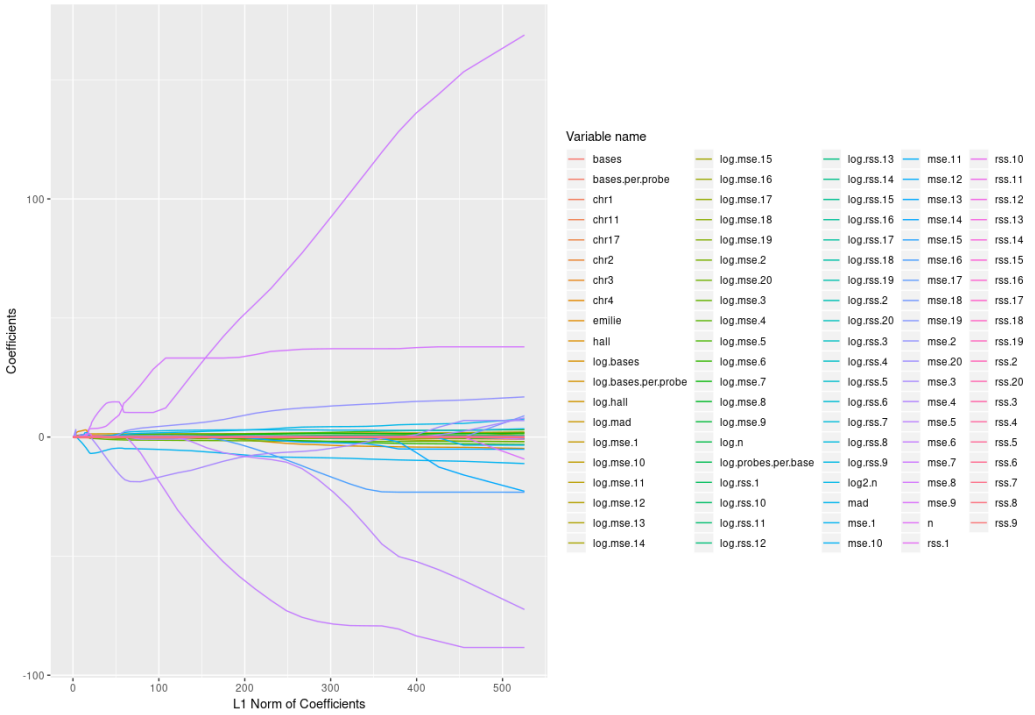

Figure 6: Lasso plot for neuroblastomaProcessed with probes.per.base removed

The plot obtained is:

The feature probes.per.base has a very small variance throughout the data points. Hence it has a very negative coefficient value as compared to others. By removing probes.per.base from the plot. We obtain a much cleaner plot.

7.3 Hard Test

Use 5-fold CV to compare the iregnet model with penaltyLearning::IntervalRegressionCV in terms of test error. For a test error function you can use the number of predicted values which are outside the corresponding label/interval.

```
rm(list = ls())
library(penaltyLearning)
data("neuroblastomaProcessed")
#Creating function errorpercent to return test error
```

```
errorpercent <- function(predicted, output)
{
  len <- length(predicted)
  errors <- 0
  for(i in 1:len)
  {
    if((predicted[i]<output[i,1]) ||
                        (predicted[i]>output[i,2]))
    {
      errors <- errors+1
    }
    else{}
  }
  return (errors/len)
}
```

Fitting the model with `penaltyLearning::IntervalRegressionCV` and calculating the testerror on target

```
fitircv <- with(neuroblastomaProcessed,
        IntervalRegressionCV(feature.mat, target.mat,
        n.folds = 5L, verbose=0))
errorpercent(predict(fitircv, newx =
            neuroblastomaProcessed$target.mat),
            neuroblastomaProcessed$target.mat)
>0.01492101
```

Hence we see that the test error for `penaltyLearning::IntervalRegressionCV` with n.folds = 5 is 1.492101% Now, creating a function that performs n folds CV with Iregnet

```
cv.iregnet.predict <- function(X, Y, nfolds,
                          seed = 1218, lambda = NULL)
{
  #Removing features with zero variance
  X = X[,apply(X,2,function(x){
    return(mean(x)!=0)
  })]
  set.seed(seed)
  index <- sample(1:nrow(X))
  pred <- matrix(0,nrow(X), 1)
  folds <- cut(seq(1,nrow(X)),
                breaks=nfolds,labels=FALSE)
  for (i in seq(nfolds))
```

```
  {
    x.train <- X[-index[folds == i],]
    y.train <- Y[-index[folds == i],]
    x.test <- X[index[folds == i],]
    y.test <- Y[index[folds == i],]
    fit <- iregnet(x.train, y.train)
    temp <- predict(fit, newx = x.test)[,100]
    pred[index[folds == i],1] <- temp
  }
  return (pred)
}
```

Hence, we see that, the test error for `cv.iregnet.predict` is 1.813926%, which is more than the 1.492101% test error obtained with `IntervalRegressionCV` Let's evaluate the two functions over 10 iterations and try to evaluate their errors

```
errireg <- vector()
errircv <- vector()
for(i in 1:10)
{
  set.seed(1240)
  seeds <- sample.int(1000, 10)
  set.seed(seeds[i])
  fitircv <- with(neuroblastomaProcessed,
    IntervalRegressionCV(
    feature.mat, target.mat, n.folds = 5L,
    verbose=0))
  errintreg <- c(errintreg,
            errorpercent(predict(fitircv,
            newx = neuroblastomaProcessed$target.mat),
            neuroblastomaProcessed$target.mat
            )
        )
  predireg <- cv.iregnet.predict(
                    neuroblastomaProcessed$feature.mat,
                    neuroblastomaProcessed$target.mat,
                    5L, seeds[i]
                    )
  errireg <- errorpercent(predireg,
                    neuroblastomaProcessed$target.mat)
}
> mean(errircv)
```

```
[1] 0.01550614
> mean(errireg)
[1] 0.02018724
```

We see that, over 10 iterations, the test error of `IntervalRegressionCV` is 1.550614% whereas the test error of `cv.iregnet.predict` is 2.018724% i.e. on an average, the test error of 5 fold CV iregnet is more than that of `IntervalRegressionCV`

8 Miscellaneous

8.1 Proposed Improvements

Apart from the primary objectives, I propose on working on the following improvements. Note that the following points improve the experience or speed of the current package and do not add any features to it. I will be attempting them during the weekends of the coding period. I'll be dedicating 2 hours every weekend day to these proposals. What I would like to see Not as fast as glmnet, Implementing quasi newton

8.1.1 Optimize iregnet

PR#59 should be merged to the master. The optimization branch was being worked upon by RoverVan for GSOC 2017 and has shown significant increases in the runtime of the optimized iregnet as shown in the blog post by RoverVan on **'iregnetbenchmark'**.

8.1.2 Work on the TODO's

There are seven TODO's in `iregnet.R` and should we worked on before pushing the code on *github*.
I'll be working on them while writing the documentation.

8.1.3 Improve upon the error messages

The error messages can be worked upon. They can be made more specific to help the user debug his code easily. The error messages will be executed with the help of `stopifnot_error()` function.
This task will be done simultaneously with code testing(i.e.while writing examples for the functions).

8.1.4 Support for single columned response variable

In cases where the target matrix is single columned, iregnet function shouldbe able to internally duplicate the column for fitting. The code below can do this, all we need to do is include this code in *iregnet.R*.

```
    y = matrix(c(y,y), nrow = nrow(x), ncol = 2)
```

This will be executed after an if condition check. Implementing this will take no time at all.

8.2 Future Scope

By the end of this gsoc, we'll be having a complete, well tested, well documented iregnet package.
Hence, for future scope of iregnnet, I propose two projects:

- **Implementing the Newton Raphson method for optimization** : The *Newton Raphson* method has quadratic convergence and for perfectly quadratic functions can converge in single iteration. But, it has has some major drawbacks;

  - It requires the calculation of derivates of the function and might fail to converge to a root for a complicated first derivative
  - The function and its derivatives need to be continuous in the range. If not, it might fail to converge
  - The initial guess for X_0 should be accurate

  Hence users can be provided with an optional paramter of optimization algorithm (default will be *coordinate descent*, as it is more reliable than *Newton Raphson* method).

- **Support for sparse matrices** : Support of input sparse matrices from the sparseMatrix class of Matrix package.

- **Support for parallelization** : The coordinate descent algorithm can be parallelised. More details are provided in this link: `https://www.csie.ntu.edu.tw/~cjlin/papers/l1_parallel.pdf`

8.3 Questions you might have for me

This section is more about me and can be completely skipped

Why choose this particular project?

I came across R programming language last year and since then I've grown fond of it, in particular the abstractions that it provides while being versatile for machine learning and data science applications. I read about survival regression during a fun problem on kaggle about survival analysis of the game *PUBG* but hadn't given much thought to its theory.
When I read about the **iregnet** package, I became curious about how the surv regression is performed. Hence, I decided that I want to learn more and contribute to **iregnet**. If I'm selected, my final task will be a medium post on performing survival analysis on *PUBG* dataset using **iregnet**.

Do you consider yourself fit for the project?
I believe I will be able to do it. When interested in something, I dedicate all of my time to it. I'm very interested in being part of this project, contributing to a *CRAN* package is a big deal and I want to do it. Survival analysis really intrigued me and now, I spend all of my time reading about it. If the timeline is followed, the project can be comleted without any issues.

How long have you been contributing to open source? Although, I have used many open source softwares (infact now I rarely use any closed source programs), I haven't really contributed to them. This will be my first project and I plan on working hard to make it successful.

Do you want to work on this just for the sake of GSOC?
I have noticed amongst my peers that GSOC is considered more of a qualification rather than an experience. I am strongly against this notion. For a qualification or something to add in CV, I would've rather applied for internships in firms. The reason why I want to work on this project through GSOC is because it provides me with a very experienced mentor who actively wants to help me grow. I'll have the oppurtunity to be guided by mentors who are pioneers of their respective fields, The ex-gsoc'ers of my college talk about how hard they had to work and that they learnt a lot during their coding period. That's my excuse for this proposal, I'll be able to do what I always wanted to do, be a part of something which will help people and learn a lot in the process.

Will you mentor for any project in the future if given the chance?
Yes! I've stated some future scopes for iregnet and will be working on them after the coding period. If the implementation goes on till next year, it can be made into one of the proposed coding projects for next year.