

# Package ‘allan’

January 2, 2012

**Type** Package

**Title** Automated Large Linear Analysis Node

**Version** 1.01

**Date** 2010-07-23

**Author** Alan Lee

**Maintainer** Alan Lee <alanlee@stanfordalumni.org>

**Description** Automated fitting of linear regression models and a stepwise routine

**License** GPL

**LazyLoad** yes

**Depends** biglm

**Repository** CRAN

**Date/Publication** 2010-07-29 09:13:15

## R topics documented:

allan-package . . . . .	2
allanVarSelect . . . . .	3
fitvbiglm . . . . .	5
getbestchunksize . . . . .	6
predictvbiglm . . . . .	7
readinbigdata . . . . .	10

<b>Index</b>	<b>13</b>
--------------	-----------

allan-package

*Automated Large Linear Analysis Node*

---

**Description**

Builds on biglm package to automate fitting of linear models with data that do not fit into R memory. Also includes a forward step variable selection routine. None of the functions are bounded by the size of the training or validation datasets. Requires biglm package.

**Details**

Package: allan  
Type: Package  
Version: 1.0  
Date: 2010-06-15  
License: What license is it under?  
LazyLoad: yes

The main functions that the user will use are fitvbiglm, predictvbiglm, and allanVarSelect. The fitvbiglm function fits a biglm object to a specified training dataset of any size. The predictvbiglm function returns the AIC, BIC, and MSE of a linear model on a specified validation dataset. The allanVarselect function takes a biglm object and performs forward stepwise variable selection and returns the fitted model.

**Author(s)**

Alan Lee <alanlee@stanfordalumni.org>

Maintainer: Alan Lee <alanlee@stanfordalumni.org>

**References**

biglm package.

**Examples**

```
#Get external data. For your own data skip this next line and replace all
#instance of SampleData with "YourFile.csv".
SampleData=system.file("extdata","SampleDataFile.csv", package = "allan")

#get column names of dataset
columnnames<-names(read.csv(SampleData, nrows=2,header=TRUE))

#use the readinbigdata function to grab a small portion of the data
datafunction<-readinbigdata(SampleData,chunksize=1000,col.names=columnnames)

#initialize dataset and set to first line with TRUE
```

```

datafunction(TRUE)

#assign a small chunk to a dataset to create a biglm object
smallchunk<-datafunction(FALSE)

#fit a biglm object with all variables being considered for model
bigmodel <- biglm(PurePremium ~ cont1 + cont2 + cont3 + cont4 + cont5,data=smallchunk,weights=~cont0)

#perform var selection and look for best 3 variables using MSE as a metric. You
#should use a different file for validation but for simplicity here we use same.
bestmodel<-allanVarSelect(bigmodel,SampleData,SampleData,NumOfSteps=3,criteria="MSE",silent=FALSE)

#just for fun, fit the full model again
bestmodelagain<-fitvbiglm(bigmodel,SampleData)

```

---

allanVarSelect	<i>Memory Unlimited Forward Stepwise Variable Selection for Linear Models</i>
----------------	---

---

## Description

The function performs forward stepwise variable selection for linear models on any sized dataset, even if it does not fit into R memory. AIC, BIC, and MSE are the available criteria for variable selection. The variable that minimizes these metrics is selected each step until the specified number of variables are entered into the model. The selection starts with a NULL model and adds variables.

## Usage

```
allanVarSelect(BaseModel, TrnDataSetFile, ValDataSetFile, ResponseCol = 1, NumOfSteps = 10, criteria =
```

## Arguments

BaseModel	A biglm object that has a formula that specifies the full model with all variables being considered for selection. ie. $y \sim x_1 + x_2 + x_3 + \dots$ etc. In order to get a biglm object to pass, you will need to create a biglm model on a small subsection of the dataset if the dataset cannot fit into R memory. Note: Offsets should be specified with an offset option instead of included in the model formula. Otherwise an error may result.
TrnDataSetFile	The training dataset that the BaseModel will be trained on. Unlimited by size.
ValDataSetFile	The validation dataset that the BaseModel will be validated on. AIC, BIC, and MSE will be calculated from this dataset to select variables. Unlimited by size.
ResponseCol	The column that the y or response variable is in in the dataset. Training, validation, as well as the smaller data chunk that the passed biglm object was initially fit on must all have the same format ie. same variables and columns.
NumOfSteps	Number of variables to enter into the final fitted model.

criteria	criteria for variable selection. "AIC", "BIC", or "MSE" can be chosen
currentchunksize	See documentation for <code>getbestchunksize</code> .
silent	Boolean. Suppresses unnecessary output to screen if <code>silent=TRUE</code> .
MemoryAllowed	See function <code>getbestchunksize</code> for argument description.
TestedRows	See function <code>getbestchunksize</code> for argument description.
AdjFactor	See function <code>getbestchunksize</code> for argument description.

### Value

Returns the final fitted `biglm` object with the final number of variables specified. The selection statistics is saved in the object under `$SelectionSummary`.

### Note

Offsets should be specified with the `offset` option and not placed in the model formula to avoid errors.

### Author(s)

Alan Lee <[alanlee@stanfordalumni.org](mailto:alanlee@stanfordalumni.org)>

### Examples

```
#Get external data. For your own data skip this next line and replace all
#instance of SampleData with "YourFile.csv".
SampleData=system.file("extdata","SampleDataFile.csv", package = "allan")

#fit smaller data to biglm object
columnnames<-names(read.csv(SampleData, nrows=2,header=TRUE))
datafeed<-readinbigdata(SampleData,chunksize=1000,col.names=columnnames)
datafeed(TRUE)
firstchunk<-datafeed(FALSE)

#create a biglm model from the small chunk with all variables that will be considered
#for variable selection.
bigmodel <- biglm(PurePremium ~ cont1 + cont2 + cont3 + cont4 + cont5,data=firstchunk,weights=~cont0)

#now run variable selection
FinalModel<-allanVarSelect(bigmodel,SampleData,SampleData,NumOfSteps=2,criteria="MSE",silent=FALSE)
```

**Description**

Fits a biglm object on any sized dataset. Automatically chunks up the data and returns a fitted biglm object on the entire dataset.

**Usage**

```
fitvbiglm(BaseModel, filename, currentchunksize = -1, silent = TRUE, MemoryAllowed = 0.5, TestedRows = 1)
```

**Arguments**

BaseModel	BaseModel is a biglm object. Must have a formula in the biglm object that specifies the model ie. $y \sim x_1 + x_2$ etc.
filename	Name of the training set file
currentchunksize	Allows user to specify the size of chunking. default is -1 for automatically determining the size by use of getbestchunksize function
silent	specify as TRUE to suppress all nonimportant messages by the function
MemoryAllowed	See function getbestchunksize for argument description.
TestedRows	See function getbestchunksize for argument description.
AdjFactor	See function getbestchunksize for argument description.

**Value**

Returns a fitted biglm object.

**Author(s)**

Alan Lee <alanlee@stanfordalumni.org>

**Examples**

```
#Get external data. For your own data skip this next line and replace all
#instance of SampleData with "YourFile.csv".
SampleData=system.file("extdata","SampleDataFile.csv", package = "allan")

#get smaller chunk of data to fit initial model
columnnames<-names(read.csv(SampleData, nrows=2,header=TRUE))
datafeed<-readinbigdata(SampleData,chunksize=1000,col.names=columnnames)
datafeed(TRUE)
firstchunk<-datafeed(FALSE)

#create a biglm model from the small chunk with all variables that will be considered
#for variable selection.
```

```
bigmodel <- biglm(PurePremium ~ cont1 + cont2 + cont3 + cont4 + cont5,data=firstchunk,weights=~cont0)

#now fit the model on the humongous dataset
finalbigmodel<-fitvbiglm(bigmodel,SampleData)
```

---

getbestchunksize      *Calculate Optimal Size of Each Chunk of Data when Fitting Models*

---

### Description

Reads in a small portion of the data and measures the amount of memory the portion occupies in R and then calculates the best size for each chunk based on available memory and additional overhead needed for calculations.

### Usage

```
getbestchunksize(filename, MemoryAllowed = 0.5, TestedRows = 1000, AdjFactor = 0.095, silent = TRUE)
```

### Arguments

filename	The name of the file being chunked
MemoryAllowed	The maximum amount of memory,in gigabytes, that you want allowed by the R process on the current system or OS. The recommend setting is 0.5-1.0 Gb. Please see the CRAN website for inherent limits to memory on various versions of R.
TestedRows	Number of rows to read in for determining optimal chunksize. On thousand is set by default
AdjFactor	Adjustment factor to account for overhead of processes during fitting. Increase factor to increase the memory used. By default, the factor is 0.095.
silent	Set silent=TRUE to suppress most messages from function.

### Value

Returns the optimal chunksize as the number of lines to read each iteration.

### Author(s)

Alan Lee <alanlee@stanfordalumni.org>

**Examples**

```

#Get external data. For your own data skip this next line and replace all
#instance of SampleData with "YourFile.csv".
SampleData=system.file("extdata","SampleDataFile.csv", package = "allan")

#To get optimal chunksize for up to 1 Gb of allowable ram use for R while
#testing memory use by reading 1000 rows of current dataset and suppressing
#some output.
currentchunksize<-getbestchunksize(SampleData,MemoryAllowed=1 ,TestedRows=1000,silent=FALSE)

## The function is currently defined as
getbestchunksize<-function(filename,MemoryAllowed=0.5,TestedRows=1000,AdjFactor=0.095,silent=TRUE){
#Function that tests data size and adjusts memory for best chunking of large dataset
#This is done by reading in a number of rows(1000 by default)and then measuring the size of the memory
#used. Memory allwed is specified in Gb. The adjfactor is a factor used to adjust memory for overhead
#in the biglm fitting functions.

#get column names
columnnames<-names(read.csv(filename, nrows=2,header=TRUE))
#read in rows and test size
datapreview<-read.csv(filename, nrows=TestedRows,header=TRUE)
datamemsize<-object.size(datapreview)
optimalchunksize=floor(((MemoryAllowed*1000000000)/datamemsize[1])*TestedRows *AdjFactor)
if (silent!=TRUE){
print("Total memory usage for 1000 lines:")
print(datamemsize)
print("Chunksize for dataframe after adjustment factor:")
print(optimalchunksize)
}
return(optimalchunksize)
}

```

---

predictvbiglm

*Generates Predictions and Fitting Statistics for biglm object*


---

**Description**

Generates prediction statistics on a validation file and returns data frame of results. For actual predictions, use the biglm predict function.

**Usage**

```
predictvbiglm(BaseModel, ValFileName, currentchunksize = -1, ResponseCol = 1, silent = TRUE, MemoryAll
```

**Arguments**

BaseModel	BaseModel is a biglm object. Must have a formula in the biglm object that specifies the model ie. $y \sim x_1 + x_2$ etc.
ValFileName	Validation filename. Can be any size. This is what is used to calculate AIC,BIC, or MSE and other statistics.
currentchunksize	Allows user to specify the size of chunking. default is -1 for automatically determining the size by use of getbestchunksize function
ResponseCol	The column that the y or response variable is in in the dataset. Training, validation, as well as the smaller data chunk that the passed biglm object was initially fit on must all have the same format ie. same variables and columns.
silent	specify as TRUE to suppress all nonimportant messages by the function
MemoryAllowed	See function getbestchunksize for argument description.
TestedRows	See function getbestchunksize for argument description.
AdjFactor	See function getbestchunksize for argument description.

**Value**

Returns a data frame that contains the AIC,BIC, and MSE of the model on the specified validation or ValFileName file.

**Author(s)**

Alan Lee <alanlee@stanfordalumni.org>

**Examples**

```
#Get external data. For your own data skip this next line and replace all
#instance of SampleData with "YourFile.csv".
SampleData=system.file("extdata","SampleDataFile.csv", package = "allan")

#get smaller chunk of data to fit initial model
columnnames<-names(read.csv(SampleData, nrows=2,header=TRUE))
datafeed<-readinbigdata(SampleData,chunksize=1000,col.names=columnnames)
datafeed(TRUE)
firstchunk<-datafeed(FALSE)

#create a biglm model from the small chunk with all variables that will be considered
#for variable selection.
bigmodel <- biglm(PurePremium ~ cont1 + cont2 + cont3 + cont4 + cont5,data=firstchunk,weights=~cont0)

#now fit the model on the humongous dataset
finalbigmodel<-fitvbiglm(bigmodel,SampleData)

#now use predictvbiglm to get metric results
metricresults<-predictvbiglm(finalbigmodel,SampleData,ResponseCol=1)

## The function is currently defined as
```

```

predictvbiglm<-function(BaseModel,ValFileName,currentchunksize=-1,ResponseCol=1,silent=TRUE,MemoryAllowed=0.5,
#This function takes a biglm object and makes predictions as well as returns fit statistics

#get optimal chunksize if not specified
if (currentchunksize==-1){
currentchunksize<-getbestchunksize(ValFileName,MemoryAllowed=MemoryAllowed,TestedRows=TestedRows,AdjFactor=Adj

}

#get datafeed
columnnames<-names(read.csv(ValFileName, nrows=2,header=TRUE))
datafeed<-readinbigdata(ValFileName,chunksize=currentchunksize,col.names=columnnames)

#initialize running total variables
ObsVec<-NULL
RSSVec<-NULL
VarianceVec<-NULL
MeanVec<-NULL
Var1Vec<-NULL
YValues<-NULL
WeightVec<-NULL

#fit first iteration
#use data grabbing function and initialize for first iteration
datafeed(TRUE)
#use predict for biglm to get predictions
CurrentDataSet<-datafeed(FALSE)
while (!is.null(CurrentDataSet)){
Predictions<-predict(BaseModel,CurrentDataSet)

#set current number of observations in current iteration
CurrentNumObs=length(Predictions)

#assign weight vector depending on whether weights have been specified
#weights not assigned
if (is.null(BaseModel$weights)){
#assign weights as all same
weightvector<-as.vector(matrix(1,CurrentNumObs,1))
}
#weights assigned
else{
#parse name of weights
weightname<-substr(BaseModel$weights,1,100)[2]
#assign to weights
weightvector<-as.vector(eval(parse(text=paste("CurrentDataSet", "$", weightname, sep=""))))
}

#calculate mean, variance, and RSS for current chunk
CurrentMean=weighted.mean(CurrentDataSet[,ResponseCol],weightvector)
CurrentVariance=sum(((CurrentDataSet[,ResponseCol]-CurrentMean)^2)*weightvector)/sum(weightvector)
CurrentRSS= sum(((Predictions-CurrentDataSet[,ResponseCol])^2)*weightvector)

```

```

#store data for all chunks in vector for final calculation
RSSVec<-c(RSSVec,CurrentRSS)
MeanVec<-c(MeanVec,CurrentMean)
VarianceVec<-c(VarianceVec,CurrentVariance)
ObsVec<-c(ObsVec,CurrentNumObs)
WeightVec<-c(WeightVec,weightvector)
#for debugging save y values
#YValues<-c(YValues,CurrentDataSet[,ResponseCol])

#increment dataset feed connection to next chunk
CurrentDataSet<-datafeed(FALSE)
}

#calculate total RSS mean and variance
TotalRSS<-sum(RSSVec)
TotalMean<-weighted.mean(MeanVec,ObsVec)
TotalObs<-sum(ObsVec)
MSEValue<-TotalRSS/sum(WeightVec)

MeanVariance<-sum((MeanVec)^2*(ObsVec/TotalObs))-TotalMean^2
TotalVariance<-sum((VarianceVec*ObsVec)/TotalObs)+MeanVariance
#Calculate degrees of freedom: add all variables plus intercept if necessary
NumParameters=attr(BaseModel$terms,"intercept")+length(attr(BaseModel$terms,"term.labels"))

#Calculate likelihood and resulting measures
LogLikelihood= (-TotalObs/2)*(log(2*pi*(TotalVariance^2)))-(1/2)*TotalRSS/(TotalVariance^2)
AICValue=2*NumParameters-2*LogLikelihood
BICValue=-2*LogLikelihood+NumParameters*log(TotalObs)
#data frame with values
metricvalues<-data.frame(AICValue,BICValue,MSEValue)
#return metric for measuring model
return(metricvalues)

}

```

---

readinbigdata

---

*Create a Connection to Very Large Datasets for Linear Model Fitting*


---

## Description

This function opens a connection to a dataset. It is typically used when the dataset used for fitting is too large to reside in R memory. Does not necessarily need to be used by end-user.

## Usage

```
readinbigdata(filename, chunksize, ...)
```

**Arguments**

filename	The name of the file that is being connected to.
chunksize	The size of each chunk that will be read into memory at a time when fitting models. Default is -1 which means that the chunk will be determined by the function <code>getbestchunksize</code> .
...	Primarily used to pass <code>col.names</code> option to internal functions.

**Details**

This function does not need to be called by end-user. It is called everytime a model needs to be fit by the main workhorse fitting routines.

**Value**

Returns a function that passes the next set of lines of data when the argument is FALSE and resets to the beginning of the file when the Argument is TRUE. The function passed also give NULL when the end of the file has been reached.

**Author(s)**

Alan Lee <alanlee@stanfordalumni.org>

**References**

Most of the function was taken from the example given in the `biglm` package.

**Examples**

```
#Get external data. For your own data skip this next line and replace all
#instance of SampleData with "YourFile.csv".
SampleData=system.file("extdata","SampleDataFile.csv", package = "allan")

datafeed<-readinbigdata(SampleData,chunksize=1000,col.names=columnnames)

## The function is currently defined as
readinbigdata<-function(filename, chunksize,...){
#This function sets a connection to the large dataset with the specified chunksize.
#The return value is either the next chunk of data or NULL if there is no additional data left.
#Additionally if a reset=TRUE flag is passed, then the data stream goes back to the beginning.
#This was originally done to accommodate the bigglm data function option
#Taken mostly from help from biglm package

#initialize connection
conn<-NULL
function(reset){
if(reset){
if(!is.null(conn)) close(conn)
conn<<-file (description=filename, open="r")
#print("new connection open")
} else{
```

```
#make sure header isn't read for other cases and assign next block of data
rval<-read.csv(conn, nrows=chunksize,header=FALSE,skip=1,...)

#print("rows processed")
#print(dim(rval))

if (nrow(rval)==0) {
  close(conn)
  conn<-NULL
  rval<-NULL
  #print("end of file reached")
}
#print(reset)
#print(dim(rval))
return(rval)
}
}
}
```

# Index

- \*Topic **\textasciitildekw2**
  - allanVarSelect, 3
  - getbestchunksize, 6
  - predictvbiglm, 7
- \*Topic **chunksize**
  - getbestchunksize, 6
- \*Topic **chunk**
  - getbestchunksize, 6
  - readinbigdata, 10
- \*Topic **fit**
  - fitvbiglm, 5
- \*Topic **large**
  - allan-package, 2
- \*Topic **linear**
  - allan-package, 2
  - allanVarSelect, 3
  - fitvbiglm, 5
  - getbestchunksize, 6
  - predictvbiglm, 7
  - readinbigdata, 10
- \*Topic **memory**
  - allan-package, 2
  - allanVarSelect, 3
  - fitvbiglm, 5
  - getbestchunksize, 6
  - predictvbiglm, 7
  - readinbigdata, 10
- \*Topic **package**
  - allan-package, 2
- \*Topic **predict**
  - predictvbiglm, 7
- \*Topic **regression**
  - allan-package, 2
  - allanVarSelect, 3
  - fitvbiglm, 5
  - getbestchunksize, 6
  - predictvbiglm, 7
  - readinbigdata, 10
- \*Topic **selection**
  - allan-package, 2
- \*Topic **stepwise**
  - allanVarSelect, 3
- \*Topic **unbounded**
  - allan-package, 2
- allan (allan-package), 2
- allan-package, 2
- allanVarSelect, 3
- fitvbiglm, 5
- getbestchunksize, 6
- predictvbiglm, 7
- readinbigdata, 10