NTNU
Norwegian University of
Science and Technology

Department of Mathematical Sciences

Examination paper for

ST2304 Statistical modelling for biologists and biotechologists

Academic contact during examination: Bob O'Hara

Phone: +47 915 54 416

Examination date:

Examination time (from–to):

Permitted examination support material: C: One yellow A4 sheet with your own handwritten notes (stamped by the Department of Mathematical Sciences), Tabeller og formler i statistikk (Tapir forlag, Fagbokforlaget), Matematiske formelsamling (K. Rottmann), specified calculator.

Other information:
All answers must be justified, and relevant calculations provided. Help pages for some R functions you may need are enclosed.

Language: English

Number of pages: 8

Number of pages enclosed: 5

Checked by:

Informasjon om trykking av eksamensoppgave	
Originalen er:	
1-sidig ☐	**2-sidig** ☒
sort/hvit ☒	**farger** ☐
skal ha flervalgskjema ☐	

Date Signature

Problem 1 Lazy Mole-rats

Mole-rats are small mammals native to Africa, with a high level of social organisation. Within their social structure different individuals play different roles, called castes. There are two castes: workers and lazy. The data below were collected by researchers to ask whether the lazy caste really is lazy. They collected data for 35 mole-rats, measuring the amount of energy individuals used, body size (because larger mole-rats will use more energy), and caste.

	caste	mass	energy
1	worker	3.85	3.69
2	worker	3.99	3.69
3	worker	4.11	3.69
33	lazy	5.00	4.53
34	lazy	4.88	4.62
35	lazy	4.81	4.49

Table 1: Data for seven mole rats: caste, log(mass (g)), and log(energy use (kJ day^{-1}))

Seven rows of the data are shown in Table 1, and the full data are plotted in Figure 1. The body mass and energy expenditure are log-transformed, using a **natural** log.

The first model just uses caste to explain energy use, see Figure 2.

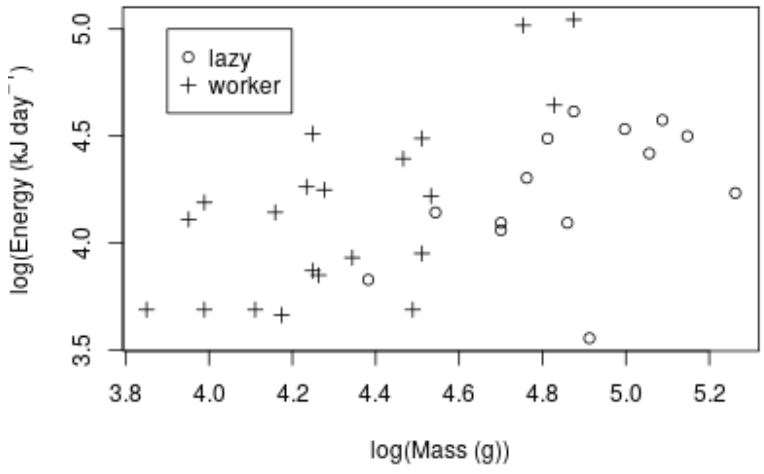

Figure 1: Measured energy use for 35 mole-rats from two castes.

```
> WorkerEnergy <- MoleRatData$energy[MoleRatData$caste=="worker"]
> LazyEnergy <- MoleRatData$energy[MoleRatData$caste=="lazy"]
> t.test(x=WorkerEnergy, y=LazyEnergy,
+        var.equal=TRUE)

        Two Sample t-test

data:  WorkerEnergy and LazyEnergy
t = -0.68391, df = 33, p-value = 0.4988
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.3538228  0.1757910
sample estimates:
mean of x mean of y
 4.156994  4.246010
```

Figure 2: R code and summary from a t-test for the effect of caste on energy use by mole-rats.

a) What specific hypothesis does this analysis test?

b) What can you conclude from this test? Explain how you reach your conclusion.

As larger animals tend to use more energy, there was a concern that body size might also be having an effect. So it was also added to the model, see Figure 3

c) Write the mathematical equation for the model fitted with this code

d) How does this model differ to those from the t-test?

e) What statistical conclusions can you draw from these results?

The model was extended to check if it described the data well. The new model (model2 in Figure 4) and a comparison with the previous model (model1 in Figure 3) are given in Figure 4.

f) What specific hypothesis does this ANOVA analysis test?

g) What can you conclude from this analysis? Explain how you reach your conclusion.

The quality of the model was also be checked: some plots relating to model fit of model1 (fitted with the code in Figure 3) are presented in Figure 5.

```
> model1 <- lm(energy ~ caste + mass, data = MoleRatData)
> summary(model1)

Call:
lm(formula = energy ~ caste + mass, data = MoleRatData)

Residuals:
     Min       1Q   Median       3Q      Max
-0.73388 -0.19371  0.01317  0.17578  0.47673

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -0.09687    0.94230  -0.103   0.9188
casteworker  0.39334    0.14611   2.692   0.0112 *
mass         0.89282    0.19303   4.625 5.89e-05 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.2966 on 32 degrees of freedom
Multiple R-squared:  0.409,        Adjusted R-squared:  0.3721
F-statistic: 11.07 on 2 and 32 DF,  p-value: 0.0002213
```

Figure 3: R code and summary for a linear model for the effect of body size and caste on energy use by mole-rats.

h) What are the assumptions of a regression model like model1?

i) How good is the model fit for model1? Explain your answer in the context of assumptions of the model.

j) What can you conclude biologically about the association between log energy expenditure and log mass from the results of these analyses?

```
> model2 <- lm(energy ~ caste * mass, data = MoleRatData)
> anova(model1, model2)
Analysis of Variance Table

Model 1: energy ~ caste + mass
Model 2: energy ~ caste * mass
  Res.Df    RSS Df Sum of Sq      F Pr(>F)
1     32 2.8145
2     31 2.7249  1  0.089557 1.0188 0.3206
```

Figure 4: R code and anova for two linear models for the effect of body size and caste on energy use by mole-rats.

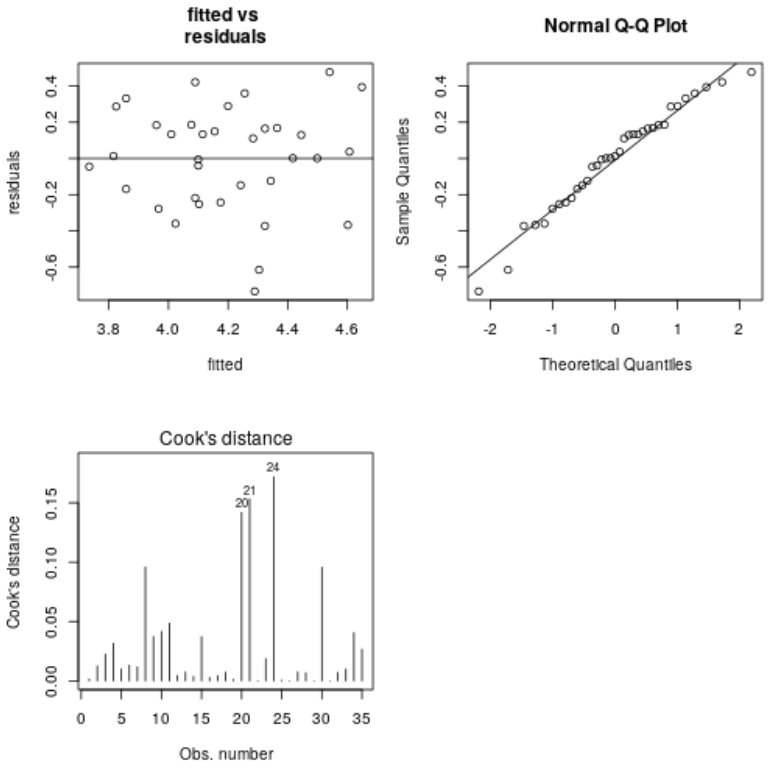

Figure 5: Graphs for checking model of energy use of two castes of mole-rat.

Problem 2 Dragon Breeds

Biologists investigating the genetics of dragon morphology believe that the difference between Norwegian Ridgebacks and Swedish Short-snouts is controlled by several genetic loci (roughly, these are genes). Some researchers sampled dragon DNA from 100 dragons, and genotyped them at 10 loci thought to affect morphology. There are no intermediates, so the phenotype (Norwegian Ridgeback or Swedish Short-snout) is easy to see. Dragons are diploid (i.e. they have two copies of each locus), and each locus has 2 alleles ("long" and "short"), so it can have 0, 1, or 2 long alleles.

The purpose of the study was to see which loci might affect morphology. This can be done using a logistic regression (i.e. a binomial GLM with a logit link function). The response is whether the dragon is a Norwegian Ridgeback, and the explanatory variables are the number of "long" alleles at each locus. We treat the number of long alleles as if it was continuous, so it is purely the number of long alleles that matters (genetically we are assuming that long alleles are co-dominant).

The first model that was fitted included all of the loci. A summary from this initial model is given in Figure 6.

a) which locus has the largest estimated effect on whether a dragon is a Norwegian Ridgeback?

b) what is the size of that effect?

c) what is the (approximate) 95% confidence interval for this effect?

The reasearchers decided to carry out model selection

d) why might this be a good idea here, as opposed to just using the analysis above?

e) what form of model selection is this - exploratory or confirmatory?

f) suggest a situation where the other form of model selection would be preferred (you can invent one involving dragons)

The researchers used full subset selection to find the best model, by fitting every model with just main effects and comparing them with AIC. The code and summary of the final model are in Figure 7.

```
> formula <- paste0("Trait ~ ",
       paste(names(Dragons)[grep("Locus", names(Dragons))],
       collapse=" + "))
> mod <- glm(formula, family="binomial",data=Dragons)
> summary(mod)

Call:
  glm(formula = formula, family = "binomial", data = Dragons)

Deviance Residuals:
  Min       1Q   Median       3Q      Max
-1.4756  -0.7866  -0.5750   0.9463   2.3460

Coefficients:
  Estimate Std. Error z value Pr(>|z|)
(Intercept) -0.68252    0.88612  -0.770  0.44116
Locus1      -0.05836    0.48238  -0.121  0.90371
Locus2      -1.36223    0.50413  -2.702  0.00689 **
Locus3      -0.18175    0.48912  -0.372  0.71020
Locus4      -0.60753    0.48307  -1.258  0.20852
Locus5       0.15452    0.51132   0.302  0.76250
Locus6       0.46858    0.48153   0.973  0.33050
Locus7       0.37802    0.50762   0.745  0.45646
Locus8      -0.22972    0.50103  -0.458  0.64659
Locus9       0.82950    0.54645   1.518  0.12902
Locus10      0.19608    0.49874   0.393  0.69421
---
  Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

Null deviance: 123.82  on 99  degrees of freedom
Residual deviance: 107.76  on 89  degrees of freedom
AIC: 129.76

Number of Fisher Scoring iterations: 4
```

Figure 6: R code and summary from a logistic regression for the estimated effect of 10 loci on being a Norwegian ridgback. Note that the first line just writes the formula for the model.

```
> library(bestglm)
> AllSubsetsAIC <- bestglm(Xy=Dragons, IC="AIC", family= binomial)
Morgan-Tatar search since family is non-gaussian.
>
  > summary(AllSubsetsAIC$BestModel)

Call:
  glm(formula = y ~ ., family = family, data = Xi, weights = weights)

Deviance Residuals:
  Min       1Q   Median       3Q      Max
-1.2494  -0.7646  -0.5285   1.1071   2.0188

Coefficients:
  Estimate Std. Error z value Pr(>|z|)
(Intercept)  -0.6503     0.4315  -1.507  0.13179
Locus2       -1.2478     0.4693  -2.659  0.00785 **
Locus9        0.8179     0.4741   1.725  0.08453 .
---
  Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

Null deviance: 123.82  on 99  degrees of freedom
Residual deviance: 111.29  on 97  degrees of freedom
AIC: 117.29

Number of Fisher Scoring iterations: 3
```

Figure 7: R code and summary of model selection for a logistic regression for the estimated effect of up to 10 loci on being a Norwegian Ridgback. Note that the first line just writes the formula for the model

g) what alternative is there to using AIC, and why might one be preferred to the other?

This second model can be used to calculate the probabilities that a dragon is a Norwegian Ridgeback, based on its genetics (remember: the inverse logit function is $p = \frac{e^{\mu}}{1+e^{\mu}}$).

h) Write down the final model selected in Figure 7.

i) If we have a dragon that is homozygous for "long" alleles (i.e. it has 2 long alleles at each locus), what is the probability that it will be a Norwegian Ridgeback?

j) If we have a dragon that is homozygous for "short" alleles (i.e. it has 0 long alleles at each locus), what is the probability that it will be a Norwegian Ridgeback?

k) From these predictions, how important do you think these loci are for determining whether a dragon is a Norwegian Ridgebacks or a Swedish Short-snout?

One possible problem with this analysis is that it assumes co-dominance, so that the effect of the long alleles are additive, i.e. the difference between 0 and 1 alleles is the same as the difference between 1 and 2 (on the logit scale).

l) How could we change the model if we did not think the effects would be additive? (think about this for a single locus)

`lm` *Fitting Linear Models*

Description

`lm` is used to fit linear models. It can be used to carry out regression, single stratum analysis of variance and analysis of covariance (although `aov` may provide a more convenient interface for these).

Usage

```
lm(formula, data, subset, weights, na.action,
   method = "qr", model = TRUE, x = FALSE, y = FALSE, qr = TRUE,
   singular.ok = TRUE, contrasts = NULL, offset, ...)
```

Arguments

`formula`	an object of class `"formula"` (or one that can be coerced to that class): a symbolic description of the model to be fitted. The details of model specification are given under 'Details'.
`data`	an optional data frame, list or environment (or object coercible by `as.data.frame` to a data frame) containing the variables in the model. If not found in `data`, the variables are taken from `environment(formula)`, typically the environment from which `lm` is called.
`subset`	an optional vector specifying a subset of observations to be used in the fitting process.
`weights`	an optional vector of weights to be used in the fitting process. Should be `NULL` or a numeric vector. If non-NULL, weighted least squares is used with weights `weights` (that is, minimizing `sum(w*e^2)`); otherwise ordinary least squares is used. See also 'Details',
`na.action`	a function which indicates what should happen when the data contain `NA`s. The default is set by the `na.action` setting of `options`, and is `na.fail` if that is unset. The 'factory-fresh' default is `na.omit`. Another possible value is `NULL`, no action. Value `na.exclude` can be useful.
`method`	the method to be used; for fitting, currently only `method = "qr"` is supported; `method = "model.frame"` returns the model frame (the same as with `model = TRUE`, see below).
`model, x, y, qr`	logicals. If `TRUE` the corresponding components of the fit (the model frame, the model matrix, the response, the QR decomposition) are returned.
`singular.ok`	logical. If `FALSE` (the default in S but not in R) a singular fit is an error.
`contrasts`	an optional list. See the `contrasts.arg` of `model.matrix.default`.
`offset`	this can be used to specify an *a priori* known component to be included in the linear predictor during fitting. This should be `NULL` or a numeric vector of length equal to the number of cases. One or more `offset` terms can be included in the formula instead or as well, and if more than one are specified their sum is used. See `model.offset`.
`...`	additional arguments to be passed to the low level regression fitting functions (see below).

Details

Models for `lm` are specified symbolically. A typical model has the form `response ~ terms` where `response` is the (numeric) response vector and `terms` is a series of terms which specifies a linear predictor for `response`. A terms specification of the form `first + second` indicates all the terms in `first` together with all the terms in `second` with duplicates removed. A specification of the form `first:second` indicates the set of terms obtained by taking the interactions of all terms in `first` with all terms in `second`. The specification `first*second` indicates the *cross* of `first` and `second`. This is the same as `first + second + first:second`.

If the formula includes an `offset`, this is evaluated and subtracted from the response.

If `response` is a matrix a linear model is fitted separately by least-squares to each column of the matrix.

See `model.matrix` for some further details. The terms in the formula will be re-ordered so that main effects come first, followed by the interactions, all second-order, all third-order and so on: to avoid this pass a `terms` object as the formula (see `aov` and `demo(glm.vr)` for an example).

A formula has an implied intercept term. To remove this use either `y ~ x - 1` or `y ~ 0 + x`. See `formula` for more details of allowed formulae.

Non-NULL `weights` can be used to indicate that different observations have different variances (with the values in `weights` being inversely proportional to the variances); or equivalently, when the elements of `weights` are positive integers w_i, that each response y_i is the mean of w_i unit-weight observations (including the case that there are w_i observations equal to y_i and the data have been summarized).

`lm` calls the lower level functions `lm.fit`, etc, see below, for the actual numerical computations. For programming only, you may consider doing likewise.

All of `weights`, `subset` and `offset` are evaluated in the same way as variables in `formula`, that is first in `data` and then in the environment of `formula`.

Value

`lm` returns an object of `class` `"lm"` or for multiple responses of class `c("mlm", "lm")`.

The functions `summary` and `anova` are used to obtain and print a summary and analysis of variance table of the results. The generic accessor functions `coefficients`, `effects`, `fitted.values` and `residuals` extract various useful features of the value returned by `lm`.

An object of class `"lm"` is a list containing at least the following components:

`coefficients`	a named vector of coefficients
`residuals`	the residuals, that is response minus fitted values.
`fitted.values`	the fitted mean values.
`rank`	the numeric rank of the fitted linear model.
`weights`	(only for weighted fits) the specified weights.
`df.residual`	the residual degrees of freedom.
`call`	the matched call.
`terms`	the `terms` object used.
`contrasts`	(only where relevant) the contrasts used.
`xlevels`	(only where relevant) a record of the levels of the factors used in fitting.
`offset`	the offset used (missing if none were used).
`y`	if requested, the response used.
`x`	if requested, the model matrix used.
`model`	if requested (the default), the model frame used.
`na.action`	(where relevant) information returned by `model.frame` on the special handling of `NA`s.

In addition, non-null fits will have components `assign`, `effects` and (unless not requested) `qr` relating to the linear fit, for use by extractor functions such as `summary` and `effects`.

Using time series

Considerable care is needed when using `lm` with time series.

Unless `na.action = NULL`, the time series attributes are stripped from the variables before the regression is done. (This is necessary as omitting `NA`s would invalidate the time series attributes, and if `NA`s are omitted in the middle of the series the result would no longer be a regular time series.)

Even if the time series attributes are retained, they are not used to line up series, so that the time shift of a lagged or differenced regressor would be ignored. It is good practice to prepare a `data` argument by `ts.intersect(..., dframe = TRUE)`, then apply a suitable `na.action` to that data frame and call `lm` with `na.action = NULL` so that residuals and fitted values are time series.

Note

Offsets specified by `offset` will not be included in predictions by `predict.lm`, whereas those specified by an offset term in the formula will be.

Author(s)

The design was inspired by the S function of the same name described in Chambers (1992). The implementation of model formula by Ross Ihaka was based on Wilkinson & Rogers (1973).

References

Chambers, J. M. (1992) *Linear models.* Chapter 4 of *Statistical Models in S* eds J. M. Chambers and T. J. Hastie, Wadsworth & Brooks/Cole.

Wilkinson, G. N. and Rogers, C. E. (1973) Symbolic descriptions of factorial models for analysis of variance. *Applied Statistics*, **22**, 392–9.

See Also

`summary.lm` for summaries and `anova.lm` for the ANOVA table; `aov` for a different interface.

The generic functions `coef`, `effects`, `residuals`, `fitted`, `vcov`.

`predict.lm` (via `predict`) for prediction, including confidence and prediction intervals; `confint` for confidence intervals of *parameters*.

`lm.influence` for regression diagnostics, and `glm` for **generalized** linear models.

The underlying low level functions, `lm.fit` for plain, and `lm.wfit` for weighted regression fitting.

More `lm()` examples are available e.g., in `anscombe`, `attitude`, `freeny`, `LifeCycleSavings`, `longley`, `stackloss`, `swiss`.

`biglm` in package **biglm** for an alternative way to fit linear models to large datasets (especially those with many cases).

Examples

```
require(graphics)

## Annette Dobson (1990) "An Introduction to Generalized Linear Models".
## Page 9: Plant Weight Data.
ctl <- c(4.17,5.58,5.18,6.11,4.50,4.61,5.17,4.53,5.33,5.14)
trt <- c(4.81,4.17,4.41,3.59,5.87,3.83,6.03,4.89,4.32,4.69)
group <- gl(2, 10, 20, labels = c("Ctl","Trt"))
weight <- c(ctl, trt)
lm.D9 <- lm(weight ~ group)
lm.D90 <- lm(weight ~ group - 1) # omitting intercept

anova(lm.D9)
summary(lm.D90)

opar <- par(mfrow = c(2,2), oma = c(0, 0, 1.1, 0))
plot(lm.D9, las = 1)      # Residuals, Fitted, ...
par(opar)

### less simple examples in "See Also" above
```

`glm` *Fitting Generalized Linear Models*

Description

`glm` is used to fit generalized linear models, specified by giving a symbolic description of the linear predictor and a description of the error distribution.

Usage

```
glm(formula, family = gaussian, data, weights, subset,
    na.action, start = NULL, etastart, mustart, offset,
    control = list(...), model = TRUE, method = "glm.fit",
    x = FALSE, y = TRUE, contrasts = NULL, ...)

glm.fit(x, y, weights = rep(1, nobs),
        start = NULL, etastart = NULL, mustart = NULL,
        offset = rep(0, nobs), family = gaussian(),
        control = list(), intercept = TRUE)

## S3 method for class 'glm'
weights(object, type = c("prior", "working"), ...)
```

Arguments

`formula`	an object of class `"formula"` (or one that can be coerced to that class): a symbolic description of the model to be fitted. The details of model specification are given under 'Details'.
`family`	a description of the error distribution and link function to be used in the model. For `glm` this can be a character string naming a family function, a family function or the result of a call to a family function. For `glm.fit` only the third option is supported. (See `family` for details of family functions.)
`data`	an optional data frame, list or environment (or object coercible by `as.data.frame` to a data frame) containing the variables in the model. If not found in `data`, the variables are taken from `environment(formula)`, typically the environment from which `glm` is called.
`weights`	an optional vector of 'prior weights' to be used in the fitting process. Should be `NULL` or a numeric vector.
`subset`	an optional vector specifying a subset of observations to be used in the fitting process.
`na.action`	a function which indicates what should happen when the data contain `NA`s. The default is set by the `na.action` setting of `options`, and is `na.fail` if that is unset. The 'factory-fresh' default is `na.omit`. Another possible value is `NULL`, no action. Value `na.exclude` can be useful.
`start`	starting values for the parameters in the linear predictor.
`etastart`	starting values for the linear predictor.
`mustart`	starting values for the vector of means.
`offset`	this can be used to specify an *a priori* known component to be included in the linear predictor during fitting. This should be `NULL` or a numeric vector of length equal to the number of cases. One or more `offset` terms can be included in the formula instead or as well, and if more than one is specified their sum is used. See `model.offset`.
`control`	a list of parameters for controlling the fitting process. For `glm.fit` this is passed to `glm.control`.
`model`	a logical value indicating whether *model frame* should be included as a component of the returned value.
`method`	the method to be used in fitting the model. The default method `"glm.fit"` uses iteratively reweighted least squares (IWLS): the alternative `"model.frame"` returns the model frame and does no fitting. User-supplied fitting functions can be supplied either as a function or a character string naming a function, with a function which takes the same arguments as `glm.fit`. If specified as a character string it is looked up from within the **stats** namespace.
`x, y`	For `glm`: logical values indicating whether the response vector and model matrix used in the fitting process should be returned as components of the returned value. For `glm.fit`: `x` is a design matrix of dimension `n * p`, and `y` is a vector of observations of length `n`.
`contrasts`	an optional list. See the `contrasts.arg` of `model.matrix.default`.
`intercept`	logical. Should an intercept be included in the *null* model?
`object`	an object inheriting from class `"glm"`.
`type`	character, partial matching allowed. Type of weights to extract from the fitted model object. Can be abbreviated.
`...`	For `glm`: arguments to be used to form the default `control` argument if it is not supplied directly. For `weights`: further arguments passed to or from other methods.

Details

A typical predictor has the form `response ~ terms` where `response` is the (numeric) response vector and `terms` is a series of terms which specifies a linear predictor for `response`. For `binomial` and `quasibinomial` families the response can also be specified as a `factor` (when the first level denotes failure and all others success) or as a two-column matrix with the columns giving the numbers of successes and failures. A terms specification of the form `first + second` indicates all the terms in `first` together with all the terms in `second` with any duplicates removed.

A specification of the form `first:second` indicates the set of terms obtained by taking the interactions of all terms in `first` with all terms in `second`. The specification `first*second` indicates the *cross* of `first` and `second`. This is the same as `first + second + first:second`.

The terms in the formula will be re-ordered so that main effects come first, followed by the interactions, all second-order, all third-order and so on: to avoid this pass a `terms` object as the formula.

Non-`NULL` `weights` can be used to indicate that different observations have different dispersions (with the values in `weights` being inversely proportional to the dispersions); or equivalently, when the elements of `weights` are positive integers w_i, that each response y_i is the mean of w_i unit-weight observations. For a binomial GLM prior weights are used to give the number of trials when the response is the proportion of successes: they would rarely be used for a Poisson GLM.

`glm.fit` is the workhorse function: it is not normally called directly but can be more efficient where the response vector, design matrix and family have already been calculated.

If more than one of `etastart`, `start` and `mustart` is specified, the first in the list will be used. It is often advisable to supply starting values for a `quasi` family, and also for families with unusual links such as `gaussian("log")`.

All of `weights`, `subset`, `offset`, `etastart` and `mustart` are evaluated in the same way as variables in `formula`, that is first in `data` and then in the environment of `formula`.

For the background to warning messages about 'fitted probabilities numerically 0 or 1 occurred' for binomial GLMs, see Venables & Ripley (2002, pp. 197–8).

Value

`glm` returns an object of class inheriting from `"glm"` which inherits from the class `"lm"`. See later in this section. If a non-standard `method` is used, the object will also inherit from the class (if any) returned by that function.

The function `summary` (i.e., `summary.glm`) can be used to obtain or print a summary of the results and the function `anova` (i.e., `anova.glm`) to produce an analysis of variance table.

The generic accessor functions `coefficients`, `effects`, `fitted.values` and `residuals` can be used to extract various useful features of the value returned by `glm`.

`weights` extracts a vector of weights, one for each case in the fit (after subsetting and `na.action`).

An object of class `"glm"` is a list containing at least the following components:

`coefficients`	a named vector of coefficients
`residuals`	the *working* residuals, that is the residuals in the final iteration of the IWLS fit. Since cases with zero weights are omitted, their working residuals are `NA`.
`fitted.values`	the fitted mean values, obtained by transforming the linear predictors by the inverse of the link function.
`rank`	the numeric rank of the fitted linear model.
`family`	the `family` object used.
`linear.predictors`	the linear fit on link scale.
`deviance`	up to a constant, minus twice the maximized log-likelihood. Where sensible, the constant is chosen so that a saturated model has deviance zero.
`aic`	A version of Akaike's *An Information Criterion*, minus twice the maximized log-likelihood plus twice the number of parameters, computed by the `aic` component of the family. For binomial and Poison families the dispersion is fixed at one and the number of parameters is the number of coefficients. For gaussian, Gamma and inverse gaussian families the dispersion is estimated from the residual deviance, and the number of parameters is the number of coefficients plus one. For a gaussian family the MLE of the dispersion is used so this is a valid value of AIC, but for Gamma and inverse gaussian families it is not. For families fitted by quasi-likelihood the value is `NA`.
`null.deviance`	The deviance for the null model, comparable with `deviance`. The null model will include the offset, and an intercept if there is one in the model. Note that this will be incorrect if the link function depends on the data other than through the fitted mean: specify a zero offset to force a correct calculation.
`iter`	the number of iterations of IWLS used.
`weights`	the *working* weights, that is the weights in the final iteration of the IWLS fit.
`prior.weights`	the weights initially supplied, a vector of `1`s if none were.
`df.residual`	the residual degrees of freedom.
`df.null`	the residual degrees of freedom for the null model.
`y`	if requested (the default) the `y` vector used. (It is a vector even for a binomial model.)
`x`	if requested, the model matrix.
`model`	if requested (the default), the model frame.
`converged`	logical. Was the IWLS algorithm judged to have converged?
`boundary`	logical. Is the fitted value on the boundary of the attainable values?
`call`	the matched call.
`formula`	the formula supplied.
`terms`	the `terms` object used.
`data`	the `data` argument.
`offset`	the offset vector used.
`control`	the value of the `control` argument used.
`method`	the name of the fitter function used, currently always `"glm.fit"`.
`contrasts`	(where relevant) the contrasts used.
`xlevels`	(where relevant) a record of the levels of the factors used in fitting.
`na.action`	(where relevant) information returned by `model.frame` on the special handling of `NA`s.

In addition, non-empty fits will have components `qr`, `R` and `effects` relating to the final weighted linear fit.

Objects of class `"glm"` are normally of class `c("glm", "lm")`, that is inherit from class `"lm"`, and well-designed methods for class `"lm"` will be applied to the weighted linear model at the final iteration of IWLS. However, care is needed, as extractor functions for class `"glm"` such as `residuals` and `weights` do **not** just pick out the component of the fit with the same name.

If a `binomial` `glm` model was specified by giving a two-column response, the weights returned by `prior.weights` are the total numbers of cases (factored by the supplied case weights) and the component `y` of the result is the proportion of successes.

Fitting functions

The argument `method` serves two purposes. One is to allow the model frame to be recreated with no fitting. The other is to allow the default fitting function `glm.fit` to be replaced by a function which takes the same arguments and uses a different fitting algorithm. If `glm.fit` is supplied as a character string it is used to search for a function of that name, starting in the **stats** namespace.

The class of the object return by the fitter (if any) will be prepended to the class returned by `glm`.

Author(s)

The original R implementation of `glm` was written by Simon Davies working for Ross Ihaka at the University of Auckland, but has since been extensively re-written by members of the R Core team.

The design was inspired by the S function of the same name described in Hastie & Pregibon (1992).

References

Dobson, A. J. (1990) *An Introduction to Generalized Linear Models.* London: Chapman and Hall.

Hastie, T. J. and Pregibon, D. (1992) *Generalized linear models.* Chapter 6 of *Statistical Models in S* eds J. M. Chambers and T. J. Hastie, Wadsworth & Brooks/Cole.

McCullagh P. and Nelder, J. A. (1989) *Generalized Linear Models.* London: Chapman and Hall.

Venables, W. N. and Ripley, B. D. (2002) *Modern Applied Statistics with S.* New York: Springer.

See Also

anova.glm, summary.glm, etc. for glm methods, and the generic functions anova, summary, effects fitted.values, and residuals.

lm for non-generalized *linear* models (which SAS calls GLMs, for 'general' linear models).

loglin and loglm (package **MASS**) for fitting log-linear models (which binomial and Poisson GLMs are) to contingency tables.

bigglm in package **biglm** for an alternative way to fit GLMs to large datasets (especially those with many cases).

esoph, infert and predict.glm have examples of fitting binomial glms.

Examples

```
## Dobson (1990) Page 93: Randomized Controlled Trial :
counts <- c(18,17,15,20,10,20,25,13,12)
outcome <- gl(3,1,9)
treatment <- gl(3,3)
print(d.AD <- data.frame(treatment, outcome, counts))
glm.D93 <- glm(counts ~ outcome + treatment, family = poisson())
anova(glm.D93)
summary(glm.D93)

## an example with offsets from Venables & Ripley (2002, p.189)
utils::data(anorexia, package = "MASS")

anorex.1 <- glm(Postwt ~ Prewt + Treat + offset(Prewt),
                family = gaussian, data = anorexia)
summary(anorex.1)


# A Gamma example, from McCullagh & Nelder (1989, pp. 300-2)
clotting <- data.frame(
    u = c(5,10,15,20,30,40,60,80,100),
    lot1 = c(118,58,42,35,27,25,21,19,18),
    lot2 = c(69,35,26,21,18,16,13,12,12))
summary(glm(lot1 ~ log(u), data = clotting, family = Gamma))
summary(glm(lot2 ~ log(u), data = clotting, family = Gamma))

## Not run:
## for an example of the use of a terms object as a formula
demo(glm.vr)

## End(Not run)
```

bestglm package:bestglm R Documentation

Best Subset GLM using Information Criterion or Cross-Validation

Description:

Best subset selection using 'leaps' algorithm (Furnival and Wilson, 1974) or complete enumeration (Morgan and Tatar, 1972). Complete enumeration is used for the non-Gaussian and for the cas where the input matrix contains factor variables with more than 1 levels. The best fit may be found using the information criterio IC: AIC, BIC, EBIC, or BICq. Alternatively, with IC='CV' various types of cross-validation may be used.

Usage:

```
bestglm(Xy, family = gaussian, IC = "BIC", t = "default",
 CVArgs = "default", qLevel = 0.99, TopModels = 5,
 method = "exhaustive", intercept = TRUE, weights = NULL,
 nvmax = "default", RequireFullEnumerationQ = FALSE, ...)
```

Arguments:

```
      Xy: Dataframe containing the design matrix X and the output
          variable y. All columns must be named.

  family: One of the glm distribution functions. The glm function is
          not used in the Gaussian case. Instead for efficiency eithe
          'leaps' is used or when factor variables are present with
          more than 2 levels, 'lm' may be used.

      IC: Information criteria to use: "AIC", "BIC", "BICg", "BICq",
          "LOOCV", "CV".

       t: adjustable parameter for BICg, BICq or CV. For BICg, defaul
          is g=t=1.  For BICq, default is q=t=0.25.  For CV, default
          the delete-d method with d=ceil(n(1-1/(log n - 1))) and
          REP=t=1000. The default value of the parameter may be chang
          by changing t.

  CVArgs: Used when IC is set to 'CV'.  The default is use the delete-
          algorithm with d=ceil(n(1-1/(log n - 1))) and t=100
          repetitions. Note that the number of repetitions can be
          changed using t.  More generally, CVArgs is a list with 3
          named components: Method, K, REP, where Method is one of
          \"HTF\", \"DH\", \"d\" corresponding to using the functions
          CVHTM (Hastie et al., 2009, K-fold CV), CVDH (adjusted K-fol
          CV, Davison and Hartigan, 1997) and CVd (delete-d CV with
          random subsamples, Shao, 1997).

  qLevel: the alpha level for determining interval for best q. Larger
          alpha's result in larger intervals.

TopModels: Finds the best 'TopModels' models.

  method: Method used in leaps algorithm for searching for the best
          subset.

intercept: Default TRUE means the intercept term is always included.
          If set to FALSE, no intercept term is included.  If you want
          only include the intercept term when it is signficant then
          set IncludeInterceptQ=FALSE and include a column of 1's in
          the design matrix.

 weights: weights

   nvmax: maximum number of independent variables allowed. By default,
          all variables

RequireFullEnumerationQ: Use exhaustive search algorithm instead of
          'leaps'

     ...: Optional arguments which are passed to 'lm' or 'glm'
```

Details:

In the Gaussian case, the loglikelihood may be written logL = -(n/2) log (RSS/n), where RSS is the residual sum-of-squares and is the number of observations. When the function 'glm' is used, the log-likelihood, logL, is obtained using 'logLik'. The penalty for EBIC and BICq depends on the tuning parameter argument, 't'. The argument 't' also controls the number of replications used when the delete-d CV is used as default. In this case, the parameter d is chosen using the formula recommended by Shao (1997). See 'CVd' for more details.

In the binomial GLM, nonlogistic, case the last two columns of Xy are the counts of 'success' and 'failures'.

Cross-validation may also be used to select the best subset. When cross-validation is used, the best models of size k according to the log-likelihood are compared for k=0,1,...,p, where p is the number of inputs. Cross-validation is not available when there ar categorical variables since in this case it is likely that the training sample may not contain all levels and in this case we can't predict the response in the validation sample. In the case of GLM, the \"DH\" method for CV is not available.

Usually it is a good idea to keep the intercept term even if it i not significant. See discussion in vignette.

Cross-validation is not available for models with no intercept term or when 'force.in' is non-null or when 'nvmax' is set to les than the full number of independent variables.

Please see the package vignette for more details and examples.

Value:

A list with class attribute 'bestglm' and named components:

```
BestModel : An lm-object representing the best fitted regression.

   Title: A brief title describing the algorithm used: CV(K=K),
          CVadj(K=K), CVd(d=K).  The range of q for an equivalent BIC
          model is given.

Subsets : The best subsets of size, k=0,1,...,p are indicated as well
          the value of the log-likelihood and information criterion fo
          each best subset.  In the case of categorical variables with
          more than 2 levels, the degrees of freedom are also shown.

  qTable: Table showing range of q for choosing each possible subset
          size. Assuming intercept=TRUE, k=1 corresponds to model with
          only an intercept term and k=p+1, where p is the number of
          input variables, corresponds to including all variables.

   Bestq: Optimal q

ModelReport: A list with components: NullModel, LEAPSQ, glmQ,
          gaussianQ, NumDF, CategoricalQ, Bestk.
```

```
BestModels: Variables in the 'TopModels' best list
     Methods function 'print.bestglm' and 'summary.bestglm' are
     provided.

Author(s):

     C. Xu and A.I. McLeod

References:

     Xu, C. and McLeod, A.I. (2009). Bayesian Information Criterion
     with Bernouilli Prior.

     Chen, J. and Chen, Z. (2008). Extended Bayesian Information
     Criteria for Model Selection with Large Model Space. Biometrika
     2008 95: 759-771.

     Furnival, G.M. and Wilson, R. W. (1974).  Regressions by Leaps a
     Bounds.  Technometrics, 16, 499-511.

     Morgan, J. A.  and Tatar, J. F. (1972).  Calculation of the
     Residual Sum of Squares for All Possible Regressions.
     Technometrics 14, 317-325.

     Miller, A. J. (2002), Subset Selection in Regression, 2nd Ed.
     London, Chapman and Hall.

     Shao, Jun (1997). An Asymptotic Theory for Linear Model Selectio
     Statistica Sinica 7, 221-264.

See Also:

     'glm', 'lm', 'leaps' 'CVHTF', 'CVDH', 'CVd'

Examples:

     #Example 1.
     #White noise test.
     set.seed(123321123)
     p<-25   #number of inputs
     n<-100  #number of observations
     X<-matrix(rnorm(n*p), ncol=p)
     y<-rnorm(n)
     Xy<-as.data.frame(cbind(X,y))

     names(Xy)<-c(paste("X",1:p,sep=""),"y")
     bestAIC <- bestglm(Xy, IC="AIC")
     bestBIC <- bestglm(Xy, IC="BIC")
     bestEBIC <- bestglm(Xy, IC="BICg")
     bestBICq <- bestglm(Xy, IC="BICq")
     NAIC <- length(coef(bestAIC$BestModel))-1
     NBIC <- length(coef(bestBIC$BestModel))-1
     NEBIC <- length(coef(bestEBIC$BestModel))-1
     NBICq <- length(coef(bestBICq$BestModel))-1
     ans<-c(NAIC, NBIC, NEBIC, NBICq)
     names(ans)<-c("AIC", "BIC", "BICg", "BICq")
     ans
     # AIC  BIC EBIC BICq
     #   3    1    0    0

     #Example 2. bestglm with BICq
     #Find best model. Default is BICq with q=0.25
     data(znuclear) #standardized data.
     #Rest of examples assume this dataset is loaded.
     out<-bestglm(znuclear, IC="BICq")
     out
     #The optimal range for q
     out$Bestq
     #The possible models that can be chosen
     out$qTable
     #The best models for each subset size
     out$Subsets
     #The overall best models
     out$BestModels
     #
     #Example 3. Normal probability plot, residuals, best model
     ans<-bestglm(znuclear, IC="BICq")
     e<-resid(ans$BestModel)
     qqnorm(e, ylab="residuals, best model")
     #
     #To save time, none of the remaining examples are run
     ## Not run:

     #Example 4. bestglm, using EBIC, g=1
     bestglm(znuclear, IC="BICg")
     #EBIC with g=0.5
     bestglm(znuclear, IC="BICg", t=0.5)
     #
     #Example 5. bestglm, CV
     data(zprostate)
     train<-(zprostate[zprostate[,10],])[,-10]
     #the default CV method takes too long, set t=10 to do only
     # 10 replications instead of the recommended 1000
     bestglm(train, IC="CV", t=10)
     bestglm(train, IC="CV", CVArgs=list(Method="HTF", K=10, REP=1))
     #Compare with DH Algorithm. Normally set REP=100 is recommended.
     bestglm(train, IC="CV", CVArgs=list(Method="DH", K=10, REP=1))
     #Compare LOOCV
     bestglm(train, IC="LOOCV")
     #
     #Example 6. Optimal q for manpower dataset
     data(manpower)
     out<-bestglm(manpower)
     out$Bestq
     #
     #Example 7. Factors with more than 2 levels
     data(AirQuality)
     bestglm(AirQuality)
     #
     #Example 8. Logistic regression
     data(SAheart)
     bestglm(SAheart, IC="BIC", family=binomial)
     #BIC agrees with backward stepwise approach
     out<-glm(chd~., data=SAheart, family=binomial)
     step(out, k=log(nrow(SAheart)))
     #but BICq with q=0.25
     bestglm(SAheart, IC="BICq", t=0.25, family=binomial)
     #
     #Cross-validation with glm
     #make reproducible results
     set.seed(33997711)
     #takes about 15 seconds and selects 5 variables
     bestglm(SAheart, IC="CV", family=binomial)
     #about 6 seconds and selects 2 variables
     bestglm(SAheart, IC="CV", CVArgs=list(Method="HTF", K=10, REP=1),
             family=binomial)
     #Will produce an error -- NA
     bestglm(SAheart, IC="CV", CVArgs=list(Method="DH", K=10, REP=1),
             family=binomial)
     bestglm(SAheart, IC="LOOCV", family=binomial)
     #
     #Example 9. Model with no intercept term
     X<-matrix(rnorm(200*3), ncol=3)

     b<-c(0, 1.5, 0)
     y<-X%*%b + rnorm(40)
     Xy<-data.frame(as.matrix.data.frame(X), y=y)
     bestglm(Xy, intercept=FALSE)
     ## End(Not run)
```

```
t.test                  package:stats                  R Documentatio

Student's t-Test

Description:

     Performs one and two sample t-tests on vectors of data.

Usage:

     t.test(x, ...)

     ## Default S3 method:
     t.test(x, y = NULL,
            alternative = c("two.sided", "less", "greater"),
            mu = 0, paired = FALSE, var.equal = FALSE,
            conf.level = 0.95, ...)

     ## S3 method for class 'formula'
     t.test(formula, data, subset, na.action, ...)

Arguments:

       x: a (non-empty) numeric vector of data values.

       y: an optional (non-empty) numeric vector of data values.

alternative: a character string specifying the alternative hypothesis
          must be one of '"two.sided"' (default), '"greater"' or
          '"less"'.  You can specify just the initial letter.

      mu: a number indicating the true value of the mean (or differen
          in means if you are performing a two sample test).

  paired: a logical indicating whether you want a paired t-test.

var.equal: a logical variable indicating whether to treat the two
          variances as being equal. If 'TRUE' then the pooled varianc
          is used to estimate the variance otherwise the Welch (or
          Satterthwaite) approximation to the degrees of freedom is
          used.

conf.level: confidence level of the interval.



 formula: a formula of the form 'lhs ~ rhs' where 'lhs' is a numeric
          variable giving the data values and 'rhs' a factor with two
          levels giving the corresponding groups.

    data: an optional matrix or data frame (or similar: see
          'model.frame') containing the variables in the formula
          'formula'.  By default the variables are taken from
          'environment(formula)'.

  subset: an optional vector specifying a subset of observations to b
          used.

na.action: a function which indicates what should happen when the dat
          contain 'NA's.  Defaults to 'getOption("na.action")'.

     ...: further arguments to be passed to or from methods.

Details:

     The formula interface is only applicable for the 2-sample tests.

     'alternative = "greater"' is the alternative that 'x' has a larg
     mean than 'y'.

     If 'paired' is 'TRUE' then both 'x' and 'y' must be specified an
     they must be the same length.  Missing values are silently remov
     (in pairs if 'paired' is 'TRUE').  If 'var.equal' is 'TRUE' then
     the pooled estimate of the variance is used.  By default, if
     'var.equal' is 'FALSE' then the variance is estimated separately
     for both groups and the Welch modification to the degrees of
     freedom is used.

     If the input data are effectively constant (compared to the larg
     of the two means) an error is generated.

Value:

     A list with class '"htest"' containing the following components:

statistic: the value of the t-statistic.

parameter: the degrees of freedom for the t-statistic.

 p.value: the p-value for the test.

conf.int: a confidence interval for the mean appropriate to the
          specified alternative hypothesis.

estimate: the estimated mean or difference in means depending on
          whether it was a one-sample test or a two-sample test.

null.value: the specified hypothesized value of the mean or mean
          difference depending on whether it was a one-sample test or
          two-sample test.

alternative: a character string describing the alternative hypothesis

  method: a character string indicating what type of t-test was
          performed.

data.name: a character string giving the name(s) of the data.

See Also:

     'prop.test'

Examples:

     require(graphics)

     t.test(1:10, y = c(7:20))      # P = .00001855
     t.test(1:10, y = c(7:20, 200)) # P = .1245    -- NOT significant

     ## Classical example: Student's sleep data
     plot(extra ~ group, data = sleep)
     ## Traditional interface
     with(sleep, t.test(extra[group == 1], extra[group == 2]))
     ## Formula interface
     t.test(extra ~ group, data = sleep)
```