NAG Library Function Document
nag_rand_kfold_xyw (g05pvc)

1 Purpose
nag_rand_kfold_xyw (g05pvc) generates training and validation datasets suitable for use in cross-validation or jack-knifing.

2 Specification
#include <nag.h>
#include <nag05.h>

void nag_rand_kfold_xyw (Integer k, Integer fold, Integer n, Integer m,
        Nag_DataByObsOrVar sordx, double x[], Integer pdx, double y[],
        double w[], Integer *nt, Integer state[], NagError *fail)

3 Description
Let \(X_o\) denote a matrix of \(n\) observations on \(m\) variables and \(y_o\) and \(w_o\) each denote a vector of length \(n\). For example, \(X_o\) might represent a matrix of independent variables, \(y_o\) the dependent variable and \(w_o\) the associated weights in a weighted regression.

nag_rand_kfold_xyw (g05pvc) generates a series of training datasets, denoted by the matrix, vector, vector triplet \((X_t, y_t, w_t)\) of \(n_t\) observations, and validation datasets, denoted \((X_v, y_v, w_v)\) with \(n_v\) observations. These training and validation datasets are generated as follows.

Each of the original \(n\) observations is randomly assigned to one of \(K\) equally sized groups or folds. For the \(k\)th sample the validation dataset consists of those observations in group \(k\) and the training dataset consists of all those observations not in group \(k\). Therefore at most \(K\) samples can be generated.

If \(n\) is not divisible by \(K\) then the observations are assigned to groups as evenly as possible, therefore any group will be at most one observation larger or smaller than any other group.

When using \(K = n\) the resulting datasets are suitable for leave-one-out cross-validation, or the training dataset on its own for jack-knifing. When using \(K \neq n\) the resulting datasets are suitable for \(K\)-fold cross-validation. Datasets suitable for reversed cross-validation can be obtained by switching the training and validation datasets, i.e., use the \(k\)th group as the training dataset and the rest of the data as the validation dataset.

One of the initialization functions nag_rand_init_repeatable (g05kfc) (for a repeatable sequence if computed sequentially) or nag_rand_init_nonrepeatable (g05kgc) (for a non-repeatable sequence) must be called prior to the first call to nag_rand_kfold_xyw (g05pvc).

4 References
None.

5 Arguments
1: \(k\) – Integer
   
   On entry: \(K\), the number of folds.
   
   Constraint: \(2 \leq k \leq n\).

2: \(fold\) – Integer
   
   On entry: the number of the fold to return as the validation dataset.
On the first call to nag_rand_kfold_xyw (g05pvc) fold should be set to 1 and then incremented by one at each subsequent call until all K sets of training and validation datasets have been produced. See Section 8 for more details on how a different calling sequence can be used.

Constraint: $1 \leq \text{fold} \leq k$.

3: \textbf{n} – Integer \hspace{1cm} \textit{Input}

On entry: $n$, the number of observations.

Constraint: $n \geq 1$.

4: \textbf{m} – Integer \hspace{1cm} \textit{Input}

On entry: $m$, the number of variables.

Constraint: $m \geq 1$.

5: \textbf{sordx} – Nag_DataByObsOrVar \hspace{1cm} \textit{Input}

On entry: determines how variables are stored in $x$.

Constraint: $\text{sordx} = \text{Nag_DataByVar}$ or $\text{Nag_DataByObs}$.

6: \textbf{x[dim]} – double \hspace{1cm} \textit{Input/Output}

Note: the dimension, $\text{dim}$, of the array $x$ must be at least

$$pdx \times m \quad \text{when} \quad \text{sordx} = \text{Nag_DataByVar};$$
$$pdx \times n \quad \text{when} \quad \text{sordx} = \text{Nag_DataByObs}.$$  

The way the data is stored in $x$ is defined by $\text{sordx}$.

If $\text{sordx} = \text{Nag_DataByVar}$, $x[(j-1) \times \text{pdx} + i - 1]$ contains the $i$th observation for the $j$th variable, for $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, m$.

If $\text{sordx} = \text{Nag_DataByObs}$, $x[(i-1) \times \text{pdx} + j - 1]$ contains the $i$th observation for the $j$th variable, for $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, m$.

On entry: if $\text{fold} = 1$, $x$ must hold $X_o$, the values of $X$ for the original dataset, otherwise, $x$ must not be changed since the last call to nag_rand_kfold_xyw (g05pvc).

On exit: values of $X$ for the training and validation datasets, with $X_t$ held in observations 1 to $nt$ and $X_v$ in observations $nt+1$ to $n$.

7: \textbf{pdx} – Integer \hspace{1cm} \textit{Input}

On entry: the stride separating row elements in the two-dimensional data stored in the array $x$.

Constraints:

- if $\text{sordx} = \text{Nag_DataByObs}$, $\text{pdx} \geq m$;
- otherwise $\text{pdx} \geq n$.

8: \textbf{y[n]} – double \hspace{1cm} \textit{Input/Output}

If the original dataset does not include $y_o$ then $y$ must be set to NULL.

On entry: if $\text{fold} \neq 1$, $y$ must not be changed since the last call to nag_rand_kfold_xyw (g05pvc).

On exit: values of $y$ for the training and validation datasets, with $y_t$ held in elements 1 to $nt$ and $y_v$ in elements $nt+1$ to $n$.

9: \textbf{w[n]} – double \hspace{1cm} \textit{Input/Output}

If the original dataset does not include $w_o$ then $w$ must be set to NULL.

On entry: if $\text{fold} \neq 1$, $w$ must not be changed since the last call to nag_rand_kfold_xyw (g05pvc).
On exit: values of \( w \) for the training and validation datasets, with \( w_t \) held in elements 1 to \( nt \) and \( w_v \) in elements \( nt + 1 \) to \( n \).

10: \( nt \) – Integer *  
Output

On exit: \( nt \), the number of observations in the training dataset.

11: \( state[dim] \) – Integer  
Communication Array

Note: the dimension, \( dim \), of this array is dictated by the requirements of associated functions that must have been previously called. This array MUST be the same array passed as argument \( state \) in the previous call to \( nag_rand_init_repeatable \) (g05kfc) or \( nag_rand_init_nonrepeatable \) (g05kgc).

On entry: contains information on the selected base generator and its current state.
On exit: contains updated information on the selected base generator and its current state.

12: \( fail \) – NagError *  
Input/Output

The NAG error argument (see Section 3.6 in the Essential Introduction).

6 Error Indicators and Warnings

NE_ALLOC_FAIL
Dynamic memory allocation failed.
See Section 3.2.1.2 in the Essential Introduction for further information.

NE_ARRAY_SIZE
On entry, \( pdx = \langle \text{value} \rangle \) and \( m = \langle \text{value} \rangle \).
Constraint: if \( sordx = \text{Nag_DataByObs} \), \( pdx \ge m \).
On entry, \( pdx = \langle \text{value} \rangle \) and \( n = \langle \text{value} \rangle \).
Constraint: if \( sordx = \text{Nag_DataByVar} \), \( pdx \ge n \).

NE_BAD_PARAM
On entry, argument \( \langle \text{value} \rangle \) had an illegal value.

NE_INT
On entry, \( m = \langle \text{value} \rangle \).
Constraint: \( m \ge 1 \).
On entry, \( n = \langle \text{value} \rangle \).
Constraint: \( n \ge 1 \).

NE_INT_2
On entry, \( fold = \langle \text{value} \rangle \) and \( k = \langle \text{value} \rangle \).
Constraint: \( 1 \le fold \le k \).
On entry, \( k = \langle \text{value} \rangle \) and \( n = \langle \text{value} \rangle \).
Constraint: \( 2 \le k \le n \).

NE_INTERNAL_ERROR
An internal error has occurred in this function. Check the function call and any array sizes. If the call is correct then please contact NAG for assistance.
An unexpected error has been triggered by this function. Please contact NAG. See Section 3.6.6 in the Essential Introduction for further information.
On entry, state vector has been corrupted or not initialized.

Your licence key may have expired or may not have been installed correctly. See Section 3.6.5 in the Essential Introduction for further information.

More than 50% of the data did not move when the data was shuffled. \(\text{value}\) of the \(\text{value}\) observations stayed put.

Not applicable.

nag_rand_kfold_xyw (g05pvc) will be computationally more efficient if each observation in \(\mathbf{x}\) is contiguous, that is \(\text{sordx} = \text{Nag_DataByObs}\).

Because of the way nag_rand_kfold_xyw (g05pvc) stores the data you should usually generate the \(K\) training and validation datasets in order, i.e., set \(\text{fold} = 1\) on the first call and increment it by one at each subsequent call. However, there are times when a different calling sequence would be beneficial, for example, when performing different cross-validation analyses on different threads. This is possible, as long as the following is borne in mind:

nag_rand_kfold_xyw (g05pvc) must be called with \(\text{fold} = 1\) first. Other than the first set, you can obtain the training and validation dataset in any order, but for a given \(\mathbf{x}\) you can only obtain each once.

For example, if you have three threads, you would call nag_rand_kfold_xyw (g05pvc) once with \(\text{fold} = 1\). You would then copy the \(\mathbf{x}\) returned onto each thread and generate the remaining \(K - 1\) sets of data by splitting them between the threads. For example, the first thread runs with \(\text{fold} = 2, \ldots, L_1\), the second with \(\text{fold} = L_1 + 1, \ldots, L_2\) and the third with \(\text{fold} = L_2 + 1, \ldots, K\).

This example uses nag_rand_kfold_xyw (g05pvc) to facilitate \(K\)-fold cross-validation.

A set of simulated data is split into 5 training and validation datasets. nag_glm_binomial (g02gbc) is used to fit a logistic regression model to each training dataset and then nag_glm_predict (g02gpc) is used to predict the response for the observations in the validation dataset.

The counts of true and false positives and negatives along with the sensitivity and specificity is then reported.

/* nag_rand_kfold_xyw (g05pvc) Example Program. * * Copyright 2014 Numerical Algorithms Group. * * Mark 25, 2014. */ /* Pre-processor includes */ #include <stdio.h> #include <nag.h> #include <nag_stdlib.h> #include <nagg02.h> #include <nagg05.h>
int main(void)
{
    /* Integer scalar and array declarations */
    Integer fn, fold, fp, i, ip, k, pdx, lstate, m,
    max_nv, n, nn, np, nt, nv, obs_val, pred_val,
    subid, tn, tp, j, pdv, rank, max_iter, print_iter;
    Integer exit_status = 0, lseed = 1;
    Integer *isx = 0, *state = 0;
    Integer seed[1];

    /* NAG structures and types */
    NagError fail;
    Nag_Link link;
    Nag_IncludeMean mean;
    Nag_BaseRNG genid;
    Nag_Distributions errfn;
    Nag_Boolean vfobs;
    Nag_DataByObsOrVar sordx;

    /* Double scalar and array declarations */
    double ex_power, dev, eps, tol, df, scale;
    double *b = 0, *cov = 0, *eta = 0, *pred = 0, *se = 0, *seeta = 0,
    *sepred = 0, *v = 0, *offset = 0, *wt = 0, *x = 0, *y = 0, *t = 0;

    /* Character scalar and array declarations */
    char clink[40], cmean[40], cgenid[40];

    /* Initialise the error structure */
    INIT_FAIL(fail);

    printf("nag_rand_kfold_xyw (g05pvc) Example Program Results\n\n");

    /* Skip heading in data file */
    #ifdef _WIN32
        scanf_s("%[^\n] ");
    #else
        scanf("%[^\n] ");
    #endif

    /* Set variables required by the regression (g02gbc) ... */

    /* Read in the type of link function, whether a mean is required */
    /* and the problem size */
    #ifdef _WIN32
        scanf_s("%39s%39s"NAG_IFMT"%"NAG_IFMT"%[^\n] ",clink, _countof(clink),
        cmean, _countof(cmean), &n, &m);
    #else
        scanf("%39s%39s"NAG_IFMT"%"NAG_IFMT"%[^\n] ",clink, cmean, &n, &m);
    #endif
    link = (Nag_Link) nag_enum_name_to_value(clink);
    mean = (Nag_IncludeMean) nag_enum_name_to_value(cmean);

    /* Set storage order for g05pvc */
    /* (pick the one required by g02gbc and g02gpc) */
    sordx = Nag_DataByObs;

    pdx = m;
    if ( !(x = NAG_ALLOC(pdx*n, double)) ||
        !(y = NAG_ALLOC(n,double)) ||
        !(t = NAG_ALLOC(n,double)) ||
        !(isx = NAG_ALLOC(m,Integer)))
    {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }

    /* This example is not using an offset or weights */
    offset = 0;
    wt = 0;
}
/* Read in data */
for (i = 0; i < n; i++)
{
  for (j = 0; j < m; j++)
  {
    #ifdef _WIN32
      scanf_s("%lf", &x[i * pdx + j]);
    #else
      scanf("%lf", &x[i * pdx + j]);
    #endif
  }
  #ifdef _WIN32
    scanf_s("%lf%lf%*[^\n] ", &y[i], &t[i]);
  #else
    scanf("%lf%lf%*[^\n] ", &y[i], &t[i]);
  #endif
}
/* Read in variable inclusion flags */
for (j = 0; j < m; j++)
{
  #ifdef _WIN32
    scanf_s("%"NAG_IFMT",&isx[j]);
  #else
    scanf("%"NAG_IFMT",&isx[j]);
  #endif
}
/* Read in control parameters for the regression */
#ifdef _WIN32
  scanf_s("%"NAG_IFMT "%lf%lf%"NAG_IFMT "%*[^\n] ", &print_iter, &eps, &tol, &max_iter);
#else
  scanf("%"NAG_IFMT "%lf%lf%"NAG_IFMT "%*[^\n] ", &print_iter, &eps, &tol, &max_iter);
#endif
/* Calculate IP */
for (ip = 0, i = 0; i < m; i++) ip += (isx[i] > 0);
if (mean == Nag_MeanInclude) ip++;
/* ... End of setting variables required by the regression */
/* Set variables required by data sampling routine (g05pvc) ... */
/* Read in the base generator information and seed */
#ifdef _WIN32
  scanf_s("%39s%"NAG_IFMT "%NAG_IFMT "%*[^\n] ", cgenid, _countof(cgenid), &subid, &seed[0]);
#else
  scanf("%39s%"NAG_IFMT "%NAG_IFMT "%*[^\n] ", cgenid, &subid, &seed[0]);
#endif
genid = (Nag_BaseRNG) nag_enum_name_to_value(cgenid);
/* Initial call to g05kfc to get size of STATE array */
lstate = 0;
nag_rand_init_repeatable(genid,subid,seed,lseed,state,&lstate,NAGERR_DEFAULT);
/* Allocate state array */
if (!(state = NAG_ALLOC(lstate, Integer)))
{
  printf("Allocation failure\n");
  exit_status = -1;
  goto END;
}
/* Initialise the generator to a repeatable sequence using g05kfc */
nag_rand_init_repeatable(genid, subid, seed, lseed, state, &lstate,
   NAGERR_DEFAULT);

/* Read in the number of folds */
#ifdef _WIN32
    scanf_s("%"NAG_IFMT"%[\n] ",&k);
#else
    scanf("%"NAG_IFMT"%[\n] ",&k);
#endif
/* ... End of setting variables required by data sampling routine */

/* Set variables required by prediction routine (g02gpc) ... */
/* Regression is performed using g02gbc so error structure is binomial */
errfn = Nag_Binomial;
/* This example does not use the predicted standard errors, so */
/* it doesn’t matter what VFOBS is set to */
vfobs = Nag_FALSE;
/* The error and link being used in the linear model don’t use scale */
ex_power = 0.0;
scale = 0.0;
/* ... End of setting variables required by prediction routine */

/* This is the maximum size for a validation dataset */
max_nv = ((double) n / (double) k) + 0.5;

/* Allocate arrays */

/* Initialise counts */
/* Loop over each fold */
for (fold = 1; fold <= k; fold++)

/* Use g05pvc to split the data into training and validation datasets */
nag_rand_kfold_xyw(k, n, m, sordx, x, pdx, y, t, &nt, state, &fail);
if (fail.code != NE_NOERROR)

/* Calculate the size of the validation dataset */
/* Call g02gbc to fit generalized linear model, with Binomial */
/* errors to training data */
nag_glm_binomial(link, mean, nt, x, pdx, m, isx, ip, y, t, wt,
offset, &dev, &df, b, &rank, se, cov, v, pdv,
tol, max_iter, print_iter, "", eps, &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_glm_binomial (g02gbc).\n%s\n",
            fail.message);
    exit_status = 1;
go_to END;
}
/* Call g02gpc to predict the response for the observations in the */
/* validation dataset */
/* We want to start passing X and T at the (NT+1)th observation, */
/* These start at (i,j)=(nt+1,1), hence the (nt*pdx+0)th element */
/* of X and the nt'th element of T */
/* Call g02gpc to predict the response for the observations in the */
/* validation dataset */
/* Call g02gpc to predict the response for the observations in the */
/* validation dataset */
/* Count the true/false positives/negatives */
for (i = 0; i < nv; i++)
{ 
    obs_val = (Integer) y[nt+i];
    pred_val = (pred[i] >= 0.5 ? 1 : 0);
    if (obs_val)
    {
        /* Positive */
        if (pred_val)
        {
            /* True positive */
            tp++;
        }
        else
        {
            /* False Negative */
            fn++;
        }
    }
    else
    {
        /* Negative */
        if (pred_val)
        {
            /* False positive */
            fp++;
        }
        else
        {
            /* True negative */
            tn++;
        }
    }
}
/* Display results */
np = tp + fn;
nn = fp + tn;
printf("Observed\n");
printf("Predicted | Negative Positive Total\n");
printf("%s %s %s\n",
        "Negative | %5"NAG_IFMT" %5"NAG_IFMT" %5"NAG_IFMT"\n");
printf("%d %d %d\n",
        tn, fn, tn + fn);
if (np != 0)
    {
        printf(" True Positive Rate (Sensitivity): %4.2f\n", (double) tp / (double) np);
    }
else
    {
        printf(" True Positive Rate (Sensitivity): No positives in data\n");
    }
if (nn != 0)
    {
        printf(" True Negative Rate (Specificity): %4.2f\n", (double) tn / (double) nn);
    }
else
    {
        printf(" True Negative Rate (Specificity): No negatives in data\n");
    }

END:

NAG_FREE(isx);
NAG_FREE(state);
NAG_FREE(b);
NAG_FREE(cov);
NAG_FREE(eta);
NAG_FREE(pred);
NAG_FREE(se);
NAG_FREE(seeta);
NAG_FREE(sepred);
NAG_FREE(v);
NAG_FREE(offset);
NAG_FREE(wt);
NAG_FREE(x);
NAG_FREE(y);
NAG_FREE(t);

return(exit_status);

9.2 Program Data

nag_rand_kfold_xyw (g05pvc) Example Program Data

Nag_Logistic Nag_MeanInclude 40 4 :: link, mean, n, m
0.0 -0.1 0.0 1.0 0.0 1.0
0.4 -1.1 1.0 1.0 1.0 1.0
-0.5 0.2 1.0 0.0 0.0 1.0
0.6 1.1 1.0 0.0 0.0 1.0
-0.3 -1.0 1.0 1.0 0.0 1.0
2.8 -1.8 0.0 1.0 0.0 1.0
0.4 -0.7 0.0 1.0 1.0 1.0
-0.4 -0.3 1.0 0.0 1.0 1.0
0.5 -2.6 0.0 0.0 1.0 1.0
-1.6 -0.3 1.0 1.0 0.0 1.0
0.4 0.6 1.0 0.0 0.0 1.0
-1.6 0.0 1.0 1.0 1.0 1.0
0.0 0.4 1.0 1.0 1.0 1.0
-0.1 0.7 1.0 1.0 0.0 1.0
-0.2 1.8 1.0 1.0 0.0 1.0
-0.9 0.7 1.0 1.0 0.0 1.0
-1.1 -0.5 1.0 1.0 0.0 1.0
-0.1 -2.2 1.0 1.0 1.0 1.0
-1.8 -0.5 1.0 1.0 1.0 1.0
-0.8 -0.9 0.0 1.0 1.0 1.0
### 9.3 Program Results

nag_rand_kfold_xyw (g05pvc) Example Program Results

<table>
<thead>
<tr>
<th>Observed</th>
<th>Negative</th>
<th>Positive</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Negative</td>
<td>18</td>
<td>8</td>
<td>26</td>
</tr>
<tr>
<td>Positive</td>
<td>4</td>
<td>10</td>
<td>14</td>
</tr>
<tr>
<td>Total</td>
<td>22</td>
<td>18</td>
<td>40</td>
</tr>
</tbody>
</table>

True Positive Rate (Sensitivity): 0.56
True Negative Rate (Specificity): 0.82