An Introduction to the VDRA package

Thomas Kent

June 11, 2020

1 Background

In medical research, it is common for researchers to need to gather data from multiple sources. However, due to privacy concerns (e.g. HIPAA) or the propitiatory nature of the data, one or more of the data partners may not be in a position to share the data. Various methods have been proposed in the literature, and some R packages have been written (`distcomp` and `ppmHR`), which allow analyses to be performed on distributed data in a secure setting. That is, the analyses are performed as if the data were aggregated, but in reality each data partner maintains control over their own data and only shares high level statistics in such a way that the original data cannot be deduced by the other data partners (references).

Each of these methods assumes that the data is horizontally partitioned. That is, each data partner has the same response variable and covariates for distinct cohort of observations. What makes the approach in this package novel is that we assume that the data is vertically partitioned. That is, each data partner holds a set of unique covariates for the same cohort of observations. (One way to think about this is with horizontally partitioned data, the observation matrix is partitioned by horizontal lines and in vertically partitioned data, the observation matrix is partitioned by vertical lines.)

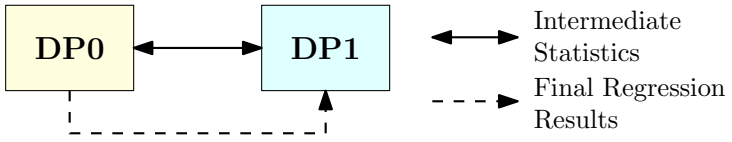
In this package, we implement three different protocols for vertical distributed regression analysis (VDRA) in a secure setting. They are called 2-party, 2^T-party, and k^T-party. Each of these protocols can be used to perform linear, logistic, and Cox regression. The purpose of this vignette is to demonstrate how to use this package, so we will not go into details of the protocols and algorithms here. However, specific information on the algorithms and a discussion of their security can be found in *Distributed Regression Analysis on Vertically Partitioned Data*, Q. Her, T. Kent, Y. Samizo, A. Slackovic, S. Toh, Y. Vilk, in preparation. **(references to our other papers and to JOSS paper)**

1.1 Dependencies

All efforts were made so that this package would run in Base-R. However, there are select cases when using functions from the `survival` package is possible and more efficient than using our own functions, and if the `survival` package is installed, those functions will be used.

1.2 2-party VDRA

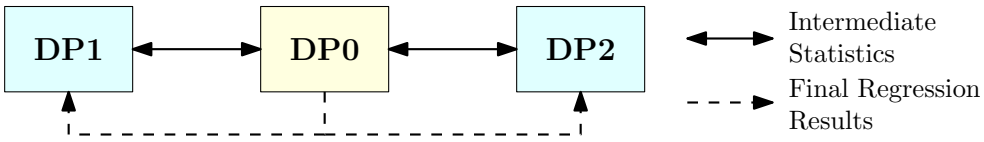
With 2-party VDRA, there are only two data holders, DP0 and DP1 (short for "data partner 0" and "data partner 1"). This protocol allows DP0 and DP1 to communicate directly with each other, but they never share patient level data with each other nor information which would allow the other party to deduce patient level information. DP0 and DP1 share intermediate statistics which each other multiple times in an iterative manner until the final regression results are computed by DP0 and shared with DP1. A pictorial representation of the data flow between the parties is shown below.



The name DP0 indicates which party is acting as the analysis center. This is the only protocol where the analysis center also provides data. For the next For the next two protocols, the analysis center does not provide any data.

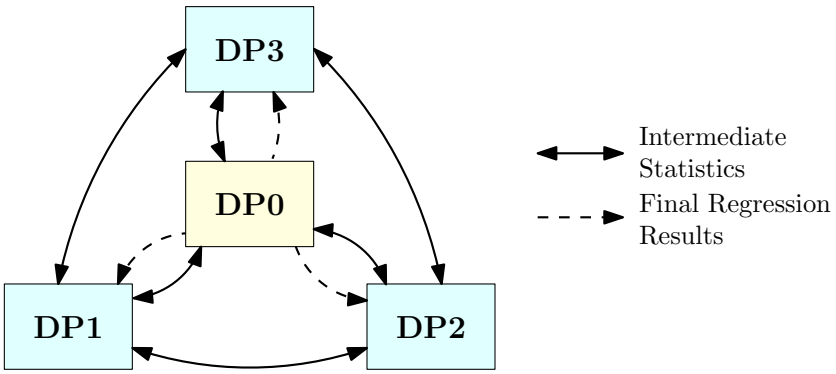
1.3 2^T-party VDRA

With 2^T-party VDRA, there are only two data holders, DP1 and DP2, and an analysis center, DP0. DP1 and DP2 cannot communicate directly with each other, but all communication must pass through DP0. DP0 is a trusted third party that helps facilitate communication and performs much of the computation that was performed by DP1 in the 2-party protocol. Whenever possible, any intermediate statistics shared with DP0 from one data partner (never patient level data) are multiplied by a random orthonormal matrix before being sent to the other data partner. This adds an extra layer of security at the expense of sending more data. The final regression results are computed by DP0 and shared with DP1 and DP2. A pictorial representation of the data flow between the parties is shown below.



1.4 k^T-party VDRA

With k^T-party VDRA, there are at least two data holders, DP1, DP2, ... DPk, and one analysis center, DP0. We have tested the package with up to 10 data holders, but there is no reason why there could not be more. With this protocol, all data partners are able to communicate with each other with the benefit that less data is transferred. As with 2^T-party VDRA, DP0 facilitates the computations and computes the final regression results, which are then sent to all the data partners. The one possible concern with this method is that a data breach at both the analysis center and one of the data partners could expose another data partner's data, even though neither the analysis center nor any data partner have enough information to reconstruct any part of any other data partners' data on their own. A pictorial representation of the data flow between the parties is shown below.



2 PopMedNet and the PopMedNet Simulator

PopMedNet (`https://www.popmednet.org/`), short for Population Medicine Network, is a "scalable and extensible open-source informatics platform designed to facilitate the implementation and operation of distributed health data networks." PopMedNet is maintained by Department of Population Medicine at the Harvard Medical School and the Harvard Pilgrim Health Care Institute. Through two generous National Institute of Health Grants, we were able to make modification to PopMedNet which allowed us to transmit data between data partners in a secure and automatic fashion. These modification were made specifically with the intent of the creation of this package. While this package has been designed to work seamlessly with

PopMedNet as a means of communication, it should be possible to use other file transfer software perform the same task.

If a group of data partners wishes to use PopMedNet and this package for analysis of vertically distributed data, please refer to the vignette *How to use PopMedNet* for further information. However, for the individuals which are interested in testing out the package to see how it works, a PopMedNet simulator, `pmn()`, has been supplied as part of the package. This allows the individual to play the part of the all the data partners and the analysis center on a single computer in order to gain an understanding of the steps involved.

We demonstrate the usage of this package, using `pmn()` as the file transfer protocol for 2-party, 2^T-party, and k^T-party situations. Take careful note the use of the parameter `popmednet = FALSE` in all function calls. When `popmednet = TRUE` is used, an offset is added to make sure that when two or more data partners are running in parallel, there is a at least a 15 second window between when they signal PopMedNet that they are ready to upload files. There are technical reasons for this that are specific to PopMednet, but are not applicable when other file transfer protocols are utilized.

3 Data

The `VDRA` package comes with a data set `vdra_data` which contains simulated data from a BMI study that was performed previously. There are 5,740 subjects with four possible response variables and seven covariates. The four response variables are:

Variable(s)	Intended Regression
`Change_BMI`	Linear
`WtLost`	Logistic
`Time`, `Status`	Cox

If each data partner is using their own data, it is assumed that each data partner has the same number of observations and that observations on corresponding rows are for the same patient. In other words, it is assumed that the data are already aligned according to some common key, and it will be treated as such. Additionally, it is expected that the data will have already been cleaned and and only valid values are presented. Some checking of the data is performed, including looking for invalid numerical or categorical values. During this step, if any problems are found a descriptive error message will be given and the program will terminate.

4 2-party Vertically Distributed Regression

For this portion of the vignette, we assume that the directory `~/vdra` is empty. If it is not empty, such as having left over files from previous runs of the program, then unintended results may happen causing the program to terminate prematurely. We must first install and load the package `vdra`. Additionally, we will run `pmn()` telling it that there will be one data partner beyond the analysis center data partner, and that the working directory is `~/vdra`.

```
> install.packages("vdra")
> library(vdra)
> if (!dir.exists("~/vdra")) dir.create("~/vdra")
> pmn(1, "~/vdra")
```

At this point, `pmn()` will have created directories `~/vdra/dp0` and `~/vdra/dp1` which are used in the communication process. For the sake of this vignette, Data Partner 0 has the response variable(s) and covariates the covariates stored in columns 5 through 7, while Data Partner 1 has the response variable and the covariates stored in columns 8 through 11.

4.1 Linear Regression

In order to perform linear regression in a 2-party setting, open up a second instance of R, which will run simultaneously with `pmn()`, and run the following script. This will be the analysis center. In reality, it does not matter which data partner is run first.

```
> library(vdra)
> fit = AnalysisCenter.2Party(regression    = "linear",
                              data          = vdra_data[, c(1, 5:7)],
                              response      = "Change_BMI",
                              monitorFolder = "~/vdra/dp0",
                              popmednet     = FALSE)
> summary(fit)
```

We now open up a third instance of R, which will also run simultaneously with the other two instances, and run the following script. This will be Data Partner 1.

```
> library(vdra)
> fit = DataPartner.2Party(regression    = "linear",
                           data          = vdra_data[, 8:11],
                           monitorFolder = "~/vdra/dp1",
                           popmednet     = FALSE)
> summary(fit)
```

After a few minutes, you should see the following output which is very similar to the output we would have obtained if we had used `lm()` on the aggregated data.

```
              Party   Estimate Std. Error    t value   Pr(>|t|)
(Intercept)    dp0   2.284e+01   1.620344  1.409e+01 < 2.22e-16 ***
Exposure       dp0  -5.779e+00   0.394282 -1.466e+01 < 2.22e-16 ***
Age            dp0   2.024e-01   0.020086  1.007e+01 < 2.22e-16 ***
ComorbidScore  dp0   2.626e-01   0.106151  2.474e+00  0.0133813 *
NumRx          dp1  -1.324e-01   0.096882 -1.367e+00  0.1717263
BMI_pre        dp1  -1.176e-02   0.021796 -5.394e-01  0.5896362
Race:Race 1    dp1   2.108e+00   0.788397  2.673e+00  0.0075283 **
Race:Race 2    dp1   1.675e+00   0.740472  2.262e+00  0.0237404 *
Race:Race 3    dp1   3.641e+00   0.733589  4.964e+00 7.1126e-07 ***
Race:Race 4    dp1   4.722e+00   0.738904  6.391e+00 1.7827e-10 ***
Race:Race 5    dp1   4.033e-02   0.746852  5.400e-02  0.9569363
Sex:M          dp1  -1.244e+00   0.563097 -2.208e+00  0.0272501 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error:  14.92 on 5728 degrees of freedom
Multiple R-squared:  0.06851 , Adjusted R-squared:  0.06672
F-statistic: 38.3 on 11 and 5728 DF, p-value: < 2.22e-16
```

For comparison, we also provide the output for the following code:

```
> fit = lm(Change_BMI ~ ., vdra_data[, c(1, 5:11)])
> summary(fit)
```

Output:

```
              Estimate Std. Error t value Pr(>|t|)
(Intercept)   22.83755    1.62034  14.094  < 2e-16 ***
```

```
Exposure      -5.77936    0.39428 -14.658  < 2e-16 ***
Age            0.20235    0.02009  10.074  < 2e-16 ***
ComorbidScore  0.26264    0.10615   2.474  0.01338 *
NumRx         -0.13242    0.09688  -1.367  0.17173
BMI_pre       -0.01176    0.02180  -0.539  0.58964
RaceRace 1     2.10776    0.78840   2.673  0.00753 **
RaceRace 2     1.67488    0.74047   2.262  0.02374 *
RaceRace 3     3.64139    0.73359   4.964 7.11e-07 ***
RaceRace 4     4.72205    0.73890   6.391 1.78e-10 ***
RaceRace 5     0.04033    0.74685   0.054  0.95694
SexM          -1.24359    0.56310  -2.208  0.02725 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 14.92 on 5728 degrees of freedom
Multiple R-squared:  0.06851,Adjusted R-squared:  0.06672
F-statistic:  38.3 on 11 and 5728 DF,  p-value: < 2.2e-16
```

4.2 Logistic Regression

In order to perform logistic regression in a 2-party setting, after running `pmn()` as above, open up a second instance of R, which will run simultaneously with `pmn()`, and run the following script. We first run DP0 which is also the analysis center. In reality, we can run either data partner first. Notice that this time we are using the binary variable `WtLost` as the response.

```
> library(vdra)
> fit = AnalysisCenter.2Party(regression    = "logistic",
                              data          = vdra_data[, c(2, 5:7)],
                              response      = "WtLost",
                              monitorFolder = "~/vdra/dp0",
                              popmednet     = FALSE)
> summary(fit)
```

We now open up a third instance of R, which will also run simultaneously with the other two instances, and run the following script. This will be Data Partner 1.

```
> library(vdra)
> fit = DataPartner.2Party(regression    = "logistic",
                           data          = vdra_data[, 8:11],
                           monitorFolder = "~/vdra/dp1",
                           popmednet     = FALSE)
> summary(fit)
```

The output for the logistic regression is similar to

```
> fit = glm(WtLost ~ ., vdra_data[c(2, 5:11)], family = binomial)
> summary(fit)
```

4.3 Cox Regression

In order to perform Cox regression in a 2-party setting, open up a second instance of R, which will run simultaneously with `pmn()`, and run the following script. In this example, we first run DP0 which is also the analysis center. In reality, we can run either data partner first.

```
> library(vdra)
> fit = AnalysisCenter.2Party(regression    = "cox",
                              data          = vdra_data[, c(3:4, 5:7)],
                              response      = c("Time", "Status"),
                              monitorFolder = "~/vdra/dp0",
                              popmednet     = FALSE)
> summary(fit)
```

We now open up a third instance of R, which will also run simultaneously with the other two instances, and run the following script. This will be Data Partner 1.

```
> library(vdra)
> fit = DataPartner.2Party(regression    = "cox",
                           data          = vdra_data[, 8:11],
                           monitorFolder = "~/vdra/dp1",
                           popmednet     = FALSE)
> summary(fit)
```

The output for Cox regression is similar to

```
> library(survival)
> fit = coxph(Surv(Time, Status) ~ ., data = vdra_data[, 3:11])
> summary(fit)
```

5 2^T-party Vertically Distributed Regression

This section is very similar to the section for 2-party Vertically Distributed Regression. As above, we assume that the directory `~/vdra` is empty. If it is not empty, such as having left over files from previous runs of the program, then unintended results may happen causing the program to terminate prematurely. We must first install and load the package `vdra`. Additionally, we will run `pmn()` telling it that there will be two data partners beyond the analysis center, and that the working directory is `~/vdra`.

```
> install.packages("vdra")
> library(vdra)
> if (!dir.exists("~/vdra")) dir.create("~/vdra")
> pmn(2, "~/vdra")
```

At this point, `pmn()` will have created directories `~/vdra/dp0`, `~/vdra/dp1`, `~/vdra/dp2` which will be used in the communication process. For the sake of this vignette, Data Partner 1 has the response variable(s) and covariates corresponding to columns 5 through 7, while Data Partner 2 has the covariates corresponding to columns 8 through 11.

5.1 Linear Regression

In order to perform linear regression in a 2^T-party setting, open up a second instance of R, which will run simultaneously with `pmn()`. In this example, we first run DP0 which is also the analysis center. In reality, we can run either data partner or the analysis center first.

```
> library(vdra)
> fit = AnalysisCenter.3Party(regression    = "linear",
                              monitorFolder = "~/vdra/dp0",
                              popmednet     = FALSE)
> summary(fit)
```

Open up a third instance of R, which will run simultaneously with `pmn()` and run the following script. This will be Data Partner 1, which has the response variable.

```
> library(vdra)
> fit = DataPartner1.3Party(regression    = "linear",
                            data          = vdra_data[, c(1, 5:7)],
                            response      = "Change_BMI",
                            monitorFolder = "~/vdra/dp1",
                            popmednet     = FALSE)
> summary(fit)
```

We now open up a fourth instance of R, which will also run simultaneously with the other three instances, and execute the following code. This will be Data Partner 2.

```
> library(vdra)
> fit = DataPartner2.3Party(regression    = "linear",
                              data          = vdra_data[, 8:11],
                              monitorFolder = "~/vdra/dp2",
                              popmednet     = FALSE)
> summary(fit)
```

After a few minutes, you should see the same output as in the 2-party scenario.

5.2 Logistic Regression

In order to perform logistic regression in a 2^T-party setting, open up a second instance of R, which will run simultaneously with `pmn()`. In this example first run DP0 which is also the analysis center. In reality, we can run either data partner or the analysis center first.

```
> library(vdra)
> fit = AnalysisCenter.3Party(regression    = "logistic",
                              monitorFolder = "~/vdra/dp0",
                              popmednet     = FALSE)
> summary(fit)
```

Open up a third instance of R, which will run simultaneously with the previous two scripts and run the following script. This will be Data Partner 1, which has the response variable.

```
> library(vdra)
> fit = DataPartner1.3Party(regression    = "logistic",
                            data          = vdra_data[, c(2, 5:7)],
                            response      = "WtLost",
                            monitorFolder = "~/vdra/dp1",
                            popmednet     = FALSE)
> summary(fit)
```

We now open up a fourth instance of R, which will also run simultaneously with the other three instances, and execute the following code. This will be Data Partner 2.

```
> library(vdra)
> fit = DataPartner2.3Party(regression    = "logistic",
                              data          = vdra_data[, 8:11],
                              monitorFolder = "~/vdra/dp2",
                              popmednet     = FALSE)
> summary(fit)
```

After a few minutes, you should see the same output as in the 2-party scenario

5.3 Cox Regression

In order to perform Cox regression in a 2^T-party setting, open up a second instance of R, which will run simultaneously with `pmn()`. In this example, we first run DP0 which is also the analysis center. In reality, we can run either data partner or the analysis center first.

```
> library(vdra)
> fit = AnalysisCenter.3Party(regression    = "cox",
                              monitorFolder = "~/vdra/dp0",
                              popmednet     = FALSE)
> summary(fit)
```

Open up a third instance of R, which will run simultaneously with `pmn()` and run the following script. This will be Data Partner 1, which has the response variable.

```
> library(vdra)
> fit = DataPartner1.3Party(regression    = "cox",
                            data          = vdra_data[, c(3:4, 5:7)],
                            response      = c("Time", "Status"),
                            monitorFolder = "~/vdra/dp1",
                            popmednet     = FALSE)
> summary(fit)
```

We now open up a fourth instance of R, which will also run simultaneously with the other three instances, and execute the following code. This will be Data Partner 2.

```
> library(vdra)
> fit = DataPartner2.3Party(regression    = "cox",
                              data          = vdra_data[, 8:11],
                              monitorFolder = "~/vdra/dp2",
                              popmednet     = FALSE)
> summary(fit)
```

After a few minutes, you should see the same output as in the 2-party scenario

6 k^T-party Vertically Distributed Regression

This section is very similar to the section for 2^T-party Vertically Distributed Regression. As above, we assume that the directory `~/vdra` is empty. If it is not empty, such as having left over files from previous runs of the program, then unintended results may happen causing the program to terminate prematurely. We must first install and load the package `vdra`. Additionally, we will run `pmn()` telling it that there will be two data partners beyond the analysis center, and that the working directory is `~/vdra`. Note, for k^T-party, we are not limited to two data partners, but we can have any number. In fact, we have successfully run this program for $k = 10$ data partners. However, for the sake of this vignette, we will restrict ourselves to just two and note that would only have to change the value of `numDataPartners` for $k > 2$. For example, if we had $k = 10$ data partners, we would need to set `numDataPartners = 10` in each of the following code blocks.

```
> install.packages("vdra")
> library(vdra)
> if (!dir.exists("~/vdra")) dir.create("~/vdra")
> pmn(2, "~/vdra")
```

At this point, `pmn()` will have created directories `~/vdra/dp0`, `~/vdra/dp1`, `~/vdra/dp2` which will used in the communication process. For the sake of this vignette, Data Partner 1 has the response variable(s) and covariates 5 through 7, while Data Partner 2 has the response variable and covariates 8 though 11.

6.1 Linear Regression

In order to perform linear regression in a k^T-party setting, open up a second instance of R, which will run simultaneously with `pmn()`. In this example, we first run DP0 which is also the analysis center. In reality, we can run any data partner or the analysis center first.

```
> library(vdra)
> fit = AnalysisCenter.KParty(regression      = "linear",
                              numDataPartners = 2,
                              monitorFolder   = "~/vdra/dp0",
                              popmednet       = FALSE)
> summary(fit)
```

Open up a third instance of R, which will run simultaneously with `pmn()` and run the following script. This will be Data Partner 1, which has the response variable.

```
> library(vdra)
> fit = DataPartner.KParty(regression      = "linear",
                           data            = vdra_data[, c(1, 5:7)],
                           response        = "Change_BMI",
                           numDataPartners = 2,
                           dataPartnerID   = 1,
                           monitorFolder   = "~/vdra/dp1",
                           popmednet       = FALSE)
> summary(fit)
```

We now open up a fourth instance of R, which will also run simultaneously with the other three instances, and execute the following code. This will be Data Partner 2. If we had more than two data partners, then we execute this same code in a new instance of R for each data partner, incrementing the value of `dataPartnerID` appropriately.

```
> library(vdra)
> fit = DataPartner.KParty(regression      = "linear",
                           data            = vdra_data[, 8:11],
                           numDataPartners = 2,
                           dataPartnerID   = 2,
                           monitorFolder   = "~/vdra/dp2",
                           popmednet       = FALSE)
> summary(fit)
```

6.2 Logistic Regression

In order to perform linear regression in a k^T-party setting, open up a second instance of R, which will run simultaneously with `pmn()`. In this example, we first run DP0 which is also the analysis center. In reality, we can run any data partner or the analysis center first.

```
> library(vdra)
> fit = AnalysisCenter.KParty(regression      = "logistic",
                              numDataPartners = 2,
                              monitorFolder   = "~/vdra/dp0",
                              popmednet       = FALSE)
> summary(fit)
```

Open up a third instance of R, which will run simultaneously with `pmn()` and run the following script. This will be Data Partner 1, which has the response variable.

```
> library(vdra)
> fit = DataPartner.KParty(regression      = "logistic",
                           data            = vdra_data[, c(2, 5:7)],
                           response        = "WtLost",
                           numDataPartners = 2,
                           dataPartnerID   = 1,
                           monitorFolder   = "~/vdra/dp1",
                           popmednet       = FALSE)
> summary(fit)
```

We now open up a fourth instance of R, which will also run simultaneously with the other three instances, and execute the following code. This will be Data Partner 2. If we had more than two data partners, then we execute this same code in a new instance of R for each data partner, incrementing the value of `dataPartnerID` appropriately.

```
> library(vdra)
> fit = DataPartner.KParty(regression      = "logistic",
                           data            = vdra_data[, 8:11],
                           numDataPartners = 2,
                           dataPartnerID   = 2,
                           monitorFolder   = "~/vdra/dp2",
                           popmednet       = FALSE)
> summary(fit)
```

6.3 Cox Regression

In order to perform linear regression in a k^T-party setting, open up a second instance of R, which will run simultaneously with `pmn()`. In this example, we first run DP0 which is also the analysis center. In reality, we can run any data partner or the analysis center first.

```
> library(vdra)
> fit = AnalysisCenter.KParty(regression      = "cox",
                              numDataPartners = 2,
                              monitorFolder   = "~/vdra/dp0",
                              popmednet       = FALSE)
> summary(fit)
```

Open up a third instance of R, which will run simultaneously with `pmn()` and run the following script. This will be Data Partner 1, which has the response variable.

```
> library(vdra)
> fit = DataPartner.KParty(regression      = "cox",
                           data            = vdra_data[, c(3:4, 5:7)],
                           response        = c("Time", "Status"),
                           numDataPartners = 2,
                           dataPartnerID   = 1,
                           monitorFolder   = "~/vdra/dp1",
                           popmednet       = FALSE)
> summary(fit)
```

We now open up a fourth instance of R, which will also run simultaneously with the other three instances, and execute the following code. This will be Data Partner 2. If we had more than two data partners, then we execute this same code in a new instance of R for each data partner, incrementing the value of `dataPartnerID` appropriately.

```
> library(vdra)
> fit = DataPartner.KParty(regression      = "cox",
                           data            = vdra_data[, 8:11],
                           numDataPartners = 2,
                           dataPartnerID   = 2,
                           monitorFolder   = "~/vdra/dp2",
                           popmednet       = FALSE)
> summary(fit)
```

7 Utilities

In order to facilitate the analysis of the data, four utilities have been provided which allow the user to further analyze the data and the model. We present these for each of the three types of regression that we perform.

7.1 Linear Regression

For linear regression, we allow the creation of sub-models. If the the result of a distributed linear regression is stored in `vdra_fit_linear_A` (which comes with the package), then we can check different sub-models as shown in the following script. All data partners can use this function.

```
> library(vdra)
> fit1 = differentModel(Age ~ ., vdra_fit_linear_A)
> summary(fit1)
> fit2 = differentModel(Change_BMI ~ Exposure + Age, vdra_fit_linear_A)
> summary(fit2)
> fit3 = differentModel(Change_BMI ~ Exposure, vdra_fit_linear_A)
> summary(fit3)
> fit4 = differentModel(Change_BMI ~ Exposure + Age + `Sex:M` + `Race:Race 1`, vdra_fit_linear_A)
> summary(fit4)
```

7.2 Logistic Regression

We can perform both Hosmer-Lemeshow goodness of fit test for logistic regression and plot the ROC according to the following script. The results of a distributed logistic regression is stored in `vdra_fit_logistic_A` (which comes with the package). Only the data partner which holds the response can run these functions.

```
> HoslemTest(vdra_fit_logistic_A)
> HoslemTest(vdra_fit_logistic_A, 20)
> RocTest(vdra_fit_logistic_A)
> RocTest(vdra_fit_logistic_A, 20)
```

7.3 Cox Regression

We can compute the survival curve for a distributed Cox regression. This function is works similar to `survfit()` in the `survival` package. The results can be both printed and plotted. Only the data partner which holds the response can run this function.

```
> sf = survfitDistributed(vdra_fit_cox_A)
> print(sf)
> plot(sf)
```