Single studies using CohortMethod

Contents

1 Introduction **1**

2 Installation instructions **1**

3 Data extraction **2**

3.1 Configuring the connection to the server 2

3.2 Preparing the exposures and outcome(s) 2

3.3 Extracting the data from the server 4

4 Propensity scores **6**

4.1 Fitting a propensity model 7

4.2 Propensity score diagnostics 7

4.3 Using the propensity score 8

4.4 Evaluating covariate balance 9

5 Outcome models **11**

5.1 Fitting the outcome model 11

5.2 Inpecting the outcome model 12

5.3 Kaplan-Meier plot 13

5.4 Attrition diagram 14

1 Introduction

This vignette describes how you can use the CohortMethod package to perform a single new-user cohort study. We will walk through all the steps needed to perform an examplar study, and we have selected the well-studied topic of the effect of coxibs versus non-selective NSAIDs on GI bleeding-related hospitalization. For simplicity, we focus on one coxib (celecoxib) and one non-selective NSAID (diclofenac).

2 Installation instructions

Before installing the CohortMethod package make sure you have Java and RTools installed. Java can be downloaded from www.java.com. RTools can be downloaded from CRAN.

The CohortMethod package is maintained in a Github repository, and has dependencies on other packages in Github. All these packages can be downloaded and installed from within R using the `devtools` package:

```
install.packages("devtools")
library(devtools)
install_github("ohdsi/SqlRender")
install_github("ohdsi/DatabaseConnector")
install_github("ohdsi/Cyclops")
install_github("ohdsi/CohortMethod")
```

Once installed, you can use `library(CohortMethod)` to load the package.

3 Data extraction

The first step in running the CohortMethod is extracting all necessary data from the server containing the data in Common Data Model format.

3.1 Configuring the connection to the server

We need to tell R how to connect to the server where the data is. CohortMethod uses the DatabaseConnector package, which provides the `createConnectionDetails` function. Type `?createConnectionDetails` for the specific settings required for the various database management systems (DBMS). For example, one might connect to a PostgreSQL database using this code:

```
connectionDetails <- createConnectionDetails(dbms = "postgresql",
                                             server = "localhost/ohdsi",
                                             user = "joe",
                                             password = "supersecret")

cdmSchema <- "my_cdm_data"
resultsSchema <- "my_results"
```

The last two lines define the cdmSchema and resultSchema variables, which we'll use later to tell R where the data in CDM format lives, and where we want to write intermediate and result tables.

3.2 Preparing the exposures and outcome(s)

We need to define the exposures and outcomes for our study. We do this by writing SQL statements against the OMOP Common Data Model that populates a table of events we are interested in. For our example study, we've created a file called *coxibVsNonselVsGiBleed.sql* with the following contents:

```
/**********************************
File coxibsVsNonselVsGiBleed.sql
**********************************/

USE @cdmSchema;

IF OBJECT_ID('@resultsSchema.dbo.coxibVsNonselVsGiBleed', 'U') IS NOT NULL
  DROP TABLE @resultsSchema.dbo.coxibVsNonselVsGiBleed;

CREATE TABLE @resultsSchema.dbo.coxibVsNonselVsGiBleed (
  cohort_definition_id INT,
```

```
    cohort_start_date DATE,
    cohort_end_date DATE,
    subject_id BIGINT
    );

INSERT INTO @resultsSchema.dbo.coxibVsNonselVsGiBleed (
    cohort_definition_id,
    cohort_start_date,
    cohort_end_date,
    subject_id
    )
SELECT 1, -- Exposure
    drug_era_start_date,
    drug_era_end_date,
    person_id
FROM drug_era
WHERE drug_concept_id = 1118084;-- celecoxib

INSERT INTO @resultsSchema.dbo.coxibVsNonselVsGiBleed (
    cohort_definition_id,
    cohort_start_date,
    cohort_end_date,
    subject_id
    )
SELECT 2, -- Comparator
    drug_era_start_date,
    drug_era_end_date,
    person_id
FROM drug_era
WHERE drug_concept_id = 1124300; --diclofenac

INSERT INTO @resultsSchema.dbo.coxibVsNonselVsGiBleed (
    cohort_definition_id,
    cohort_start_date,
    cohort_end_date,
    subject_id
    )
SELECT 3, -- Outcome
    condition_start_date,
    condition_end_date,
    condition_occurrence.person_id
FROM condition_occurrence
INNER JOIN visit_occurrence
    ON condition_occurrence.visit_occurrence_id = visit_occurrence.visit_occurrence_id
WHERE condition_concept_id IN (
        SELECT descendant_concept_id
        FROM concept_ancestor
        WHERE ancestor_concept_id = 192671 -- GI - Gastrointestinal haemorrhage
        )
    AND visit_occurrence.place_of_service_concept_id IN (9201, 9203);
```

This is parameterized SQL which can be used by the SqlRender package. We use parameterized SQL so we do not have to pre-specify the names of the CDM and result schemas. That way, if we want to run the SQL

on a different schema, we only need to change the parameter values, we don't have to change the SQL code. By also making use of SqlRender's translation functionality, we can make sure the SQL code can be run in many different environments.

```
sql <- readSql("coxibVsNonselVsGiBleed.sql")
sql <- renderSql(sql, cdmSchema = cdmSchema, resultsSchema = resultsSchema)$sql
sql <- translateSql(sql, targetDialect = connectionDetails$dbms)$sql

connection <- connect(connectionDetails)
executeSql(connection, sql)
```

In this code, we first read the SQL from the file into memory. In the next line, we replace the two parameter names with the actual values. We then translate the SQL into the dialect appropriate for the DBMS we already specified in the connectionDetails. Next, we connect to the server, and submit the rendered and translated SQL.

If all went well, we now have a table with the events of interest. We can see how many events per type:

```
sql <- paste("SELECT cohort_definition_id, COUNT(*) AS count",
             "FROM @resultsSchema.dbo.coxibVsNonselVsGiBleed",
             "GROUP BY cohort_definition_id")
sql <- renderSql(sql, resultsSchema = resultsSchema)$sql
sql <- translateSql(sql, targetDialect = connectionDetails$dbms)$sql

querySql(connection, sql)
```

```
#>   cohort_concept_id  count
#> 1                 1 113492
#> 2                 2 342370
#> 3                 3 351289
```

3.3 Extracting the data from the server

Now we can tell CohortMethod to define the cohorts based on our events, and extract all necessary data for our analysis:

```
# Get all NSAID Concept IDs for exclusion:
sql <- paste("SELECT concept_id",
             "FROM concept_ancestor",
             "INNER JOIN concept",
             "ON descendant_concept_id = concept_id",
             "WHERE ancestor_concept_id = 21603933")
nsaids <- querySql(connection, sql)
nsaids <- nsaids$CONCEPT_ID

 #Load data:
cohortData <- getDbCohortData(connectionDetails,
                              cdmSchema = cdmSchema,
                              resultsSchema = resultsSchema,
                              targetDrugConceptId = 1,
                              comparatorDrugConceptId = 2,
                              indicationConceptIds = c(),
```

```
                                washoutWindow = 183,
                                indicationLookbackWindow = 183,
                                studyStartDate = "",
                                studyEndDate = "",
                                exclusionConceptIds = nsaids,
                                outcomeConceptIds = 3,
                                outcomeConditionTypeConceptIds = c(),
                                exposureSchema = resultsSchema,
                                exposureTable = "coxibVsNonselVsGiBleed",
                                outcomeSchema = resultsSchema,
                                outcomeTable = "coxibVsNonselVsGiBleed",
                                useCovariateDemographics = TRUE,
                                useCovariateConditionOccurrence = TRUE,
                                useCovariateConditionOccurrence365d = TRUE,
                                useCovariateConditionOccurrence30d = TRUE,
                                useCovariateConditionOccurrenceInpt180d = TRUE,
                                useCovariateConditionEra = TRUE,
                                useCovariateConditionEraEver = TRUE,
                                useCovariateConditionEraOverlap = TRUE,
                                useCovariateConditionGroup = TRUE,
                                useCovariateDrugExposure = TRUE,
                                useCovariateDrugExposure365d = TRUE,
                                useCovariateDrugExposure30d = TRUE,
                                useCovariateDrugEra = TRUE,
                                useCovariateDrugEra365d = TRUE,
                                useCovariateDrugEra30d = TRUE,
                                useCovariateDrugEraEver = TRUE,
                                useCovariateDrugEraOverlap = TRUE,
                                useCovariateDrugGroup = TRUE,
                                useCovariateProcedureOccurrence = TRUE,
                                useCovariateProcedureOccurrence365d = TRUE,
                                useCovariateProcedureOccurrence30d = TRUE,
                                useCovariateProcedureGroup = TRUE,
                                useCovariateObservation = TRUE,
                                useCovariateObservation365d = TRUE,
                                useCovariateObservation30d = TRUE,
                                useCovariateObservationBelow = TRUE,
                                useCovariateObservationAbove = TRUE,
                                useCovariateObservationCount365d = TRUE,
                                useCovariateConceptCounts = TRUE,
                                useCovariateRiskScores = TRUE,
                                useCovariateInteractionYear = FALSE,
                                useCovariateInteractionMonth = FALSE,
                                excludedCovariateConceptIds = nsaids,
                                deleteCovariatesSmallCount = 100)
cohortData
```

```
#> CohortData object
#>
#> Treatment concept ID: 1
#> Comparator concept ID: 2
#> Outcome concept ID(s): 3
```

There are a lot of parameters, but they are all documented in the CohortMethod manual. In short, we're pointing the function to the table created earlier and indicate which concept IDs in that table identify the target, comparator and outcome. Note that in this example, we do not restrict the study to people having a particular indication (`indicationConceptIds = c()`), but this is something you would often want to do. We do instruct that people with prior exposure to any NSAID should be excluded, and that many different covariates should be constructed, including covariates for all conditions, drug exposures, and procedures that were found on or before the index date.

All data about the cohorts, outcomes, and covariates are extracted from the server and stored in the cohortData object. This object uses the package ff to store information in a way that ensures R does not run out of memory, even when there is lots of data.

We can use the generic `summary()` function to view some more information of the data we extracted:

```
summary(cohortData)
```

```
#> CohortData object summary
#>
#> Treatment concept ID: 1
#> Comparator concept ID: 2
#> Outcome concept ID(s): 3
#>
#> Treated persons: 11878
#> Comparator persons: 48415
#>
#> Outcome counts:
#>   Event count Person count
#> 3        2254         1516
#>
#> Covariates:
#> Number of covariates: 22023
#> Number of non-zero covariate values: 39204318
```

3.3.1 Saving the data to file

Creating the cohortData file can take a lot of computing time, and it is probably a good idea to save it for future sessions. Because cohortData uses ff, we cannot use R's regular save function. Instead, we'll have to use the `saveCohortData()` function:

```
saveCohortData(cohortData, "coxibVsNonselVsGiBleed")
```

We can use the `loadCohortData()` function to load the data in a future session.

4 Propensity scores

The CohortMethod can use propensity scores to adjust for potential confounders. Instead of the traditional approach of using a handfull of predefined covariates, CohortMethod typically uses thousands to millions of covariates that are automatically constructed based on conditions, procedures and drugs in the records of the subjects.

4.1 Fitting a propensity model

We can fit a propensity model using the covariates constructed by the `getDbCohortData()` function:

```
ps <- createPs(cohortData, outcomeConceptId = 3)
```

The `createPs()` function uses the Cyclops package to fit a large scale regularized logistic regression. Note that We have to tell createPs what the outcomeConceptId is for which we will use the model so it can remove subjects who had the outcome prior to index date before fitting the model.

To fit the propensity model, Cyclops needs to know the hyperparameter value which specifies the variance of the prior. By default Cyclops will use cross-validation to estimate the optimal hyperparameter. However, be aware that this can take a really long time. You can use the `prior` and `control` parameters of the `createPs()` to specify Cyclops' behaviour, including using multiple CPUs to speed up the cross-validation.

4.2 Propensity score diagnostics

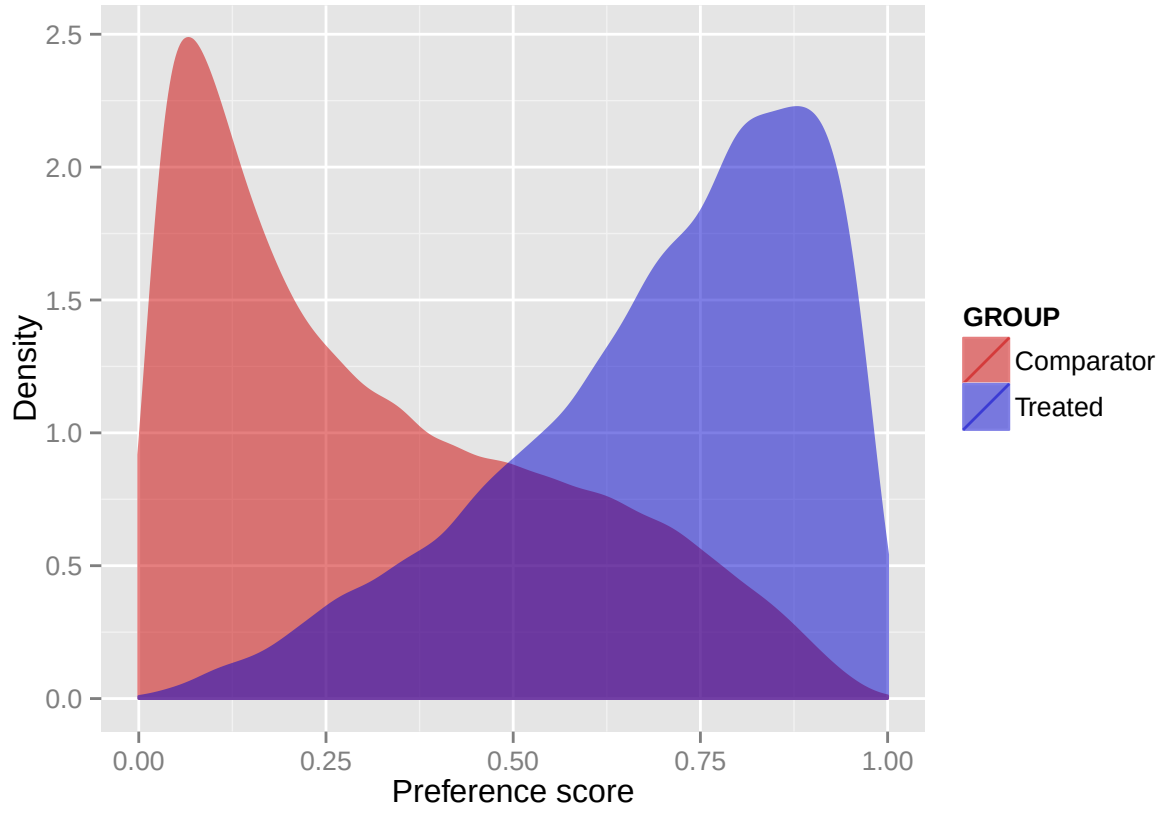
We can compute the Area Under the Receiver-Operator curve:

```
computePsAuc(ps)
```

```
#> [1] 0.8693535
```

We can also plot the propensity score distribution, although we prefer the preference score distribution:

```
plotPs(ps, scale = "preference")
```



It is also possible to inspect the propensity model itself by showing the covariates that have non-zero coefficients:

```
propensityModel <- getPsModel(ps, cohortData)
head(propensityModel)
```

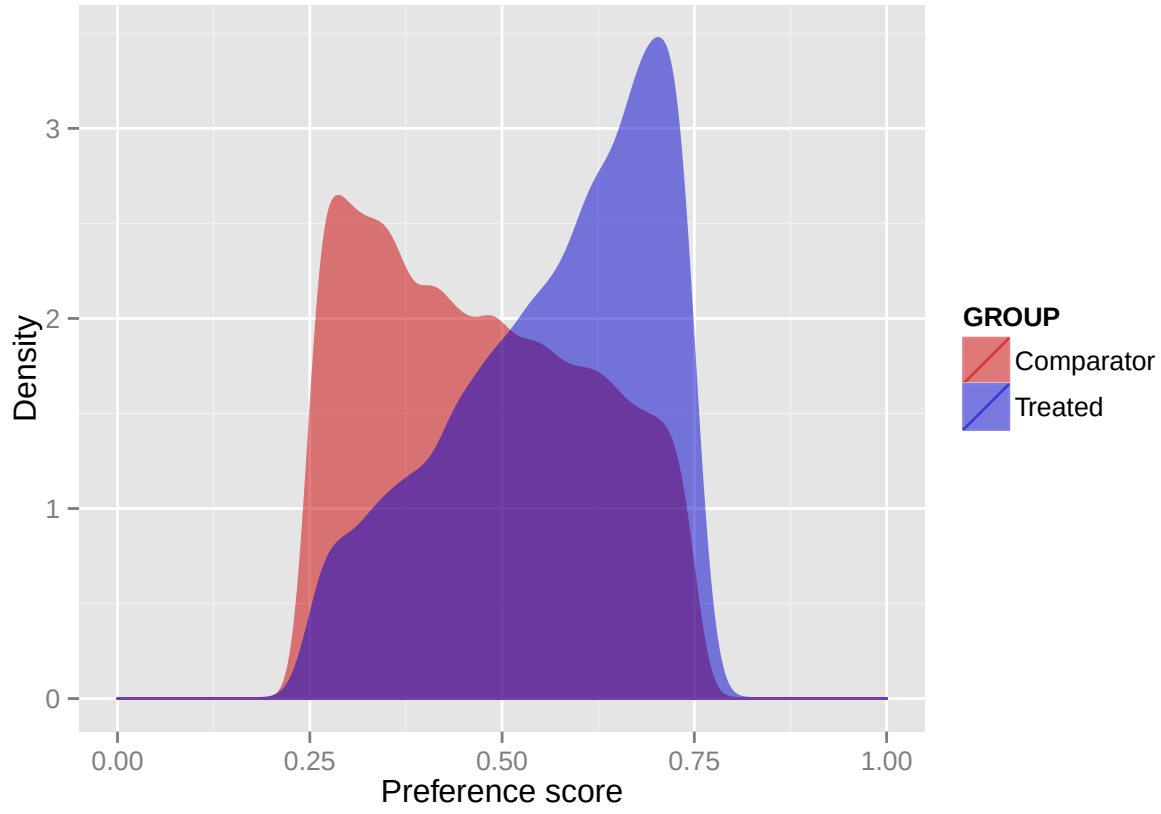
```
#>      coefficient          id                                covariateName
#> 1630  -3.4137454  1150871503 ... with cohort index:  1150871-Misoprostol
#> 1432  -1.2473805          12                                Age group: 10-14
#> 2055   1.1742889 40664232701 ...IC TREATMENT VISIT (AS PART OF CONTRACT)
#> 1     -1.1426356          13                                Age group: 15-19
#> 1718  -1.1110532 19102773402 ...in 5 MG/ML Ophthalmic Solution [Vigamox]
#> 347    0.8423181        2007                                Index year: 2007
```

One advantage of using the regularization when fitting the propensity model is that most coefficients will shrink to zero and fall out of the model. It is a good idea to inspect the remaining variables for anything that should not be there, for example instrumental variables..

4.3 Using the propensity score

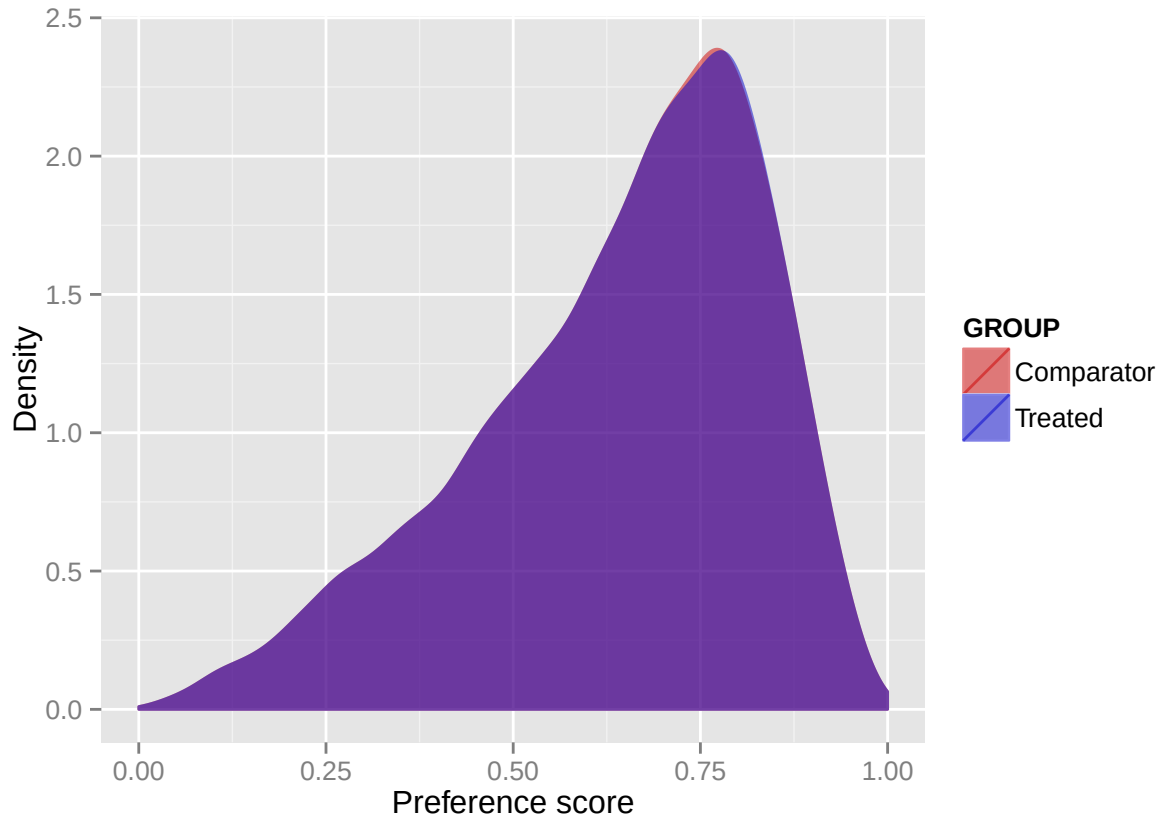
We can use the propensity scores to trim, match, or stratify our population. For example, one could first trim to equipoise, meaning only subjects with a preference score between 0.25 and 0.75 are kept:

```
psTrimmed <- trimByPsToEquipoise(ps)
plotPs(psTrimmed, ps, scale = "preference")
```



We can also match subjects based on propensity scores:

```
strata <- matchOnPs(ps, caliper = 0.25, caliperScale = "standardized", maxRatio = 1)
plotPs(strata, ps)
```

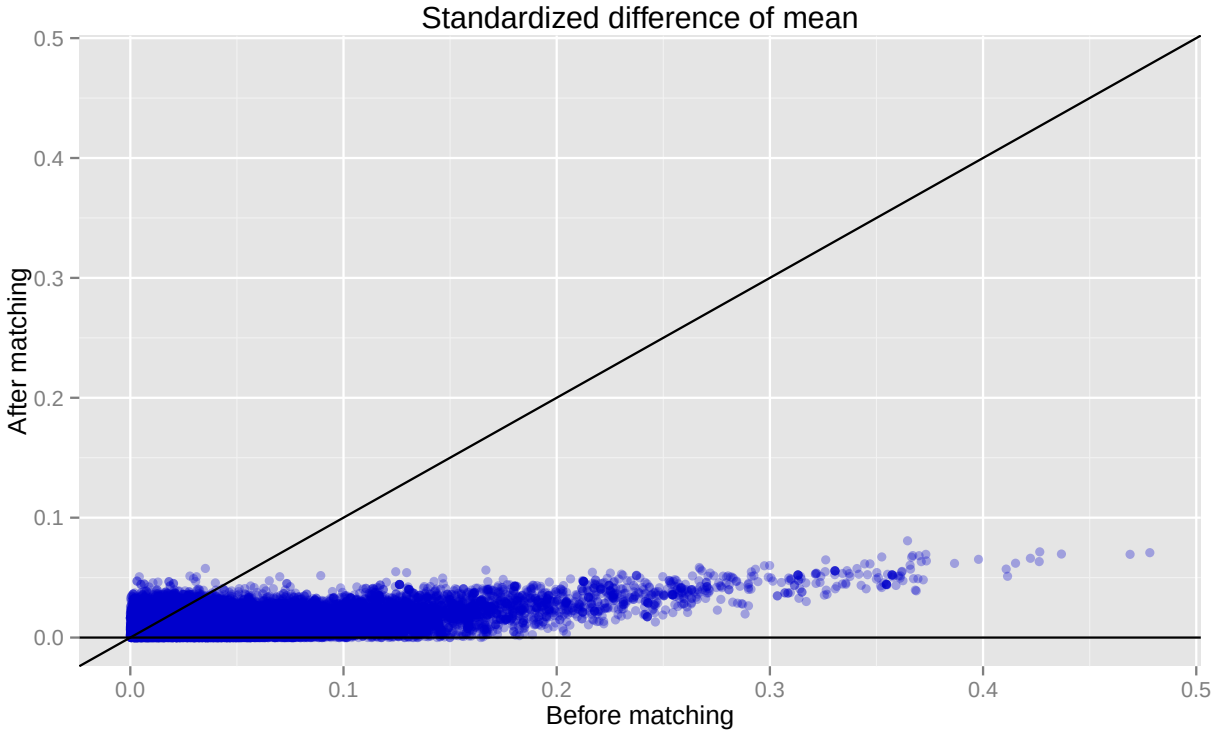


4.4 Evaluating covariate balance

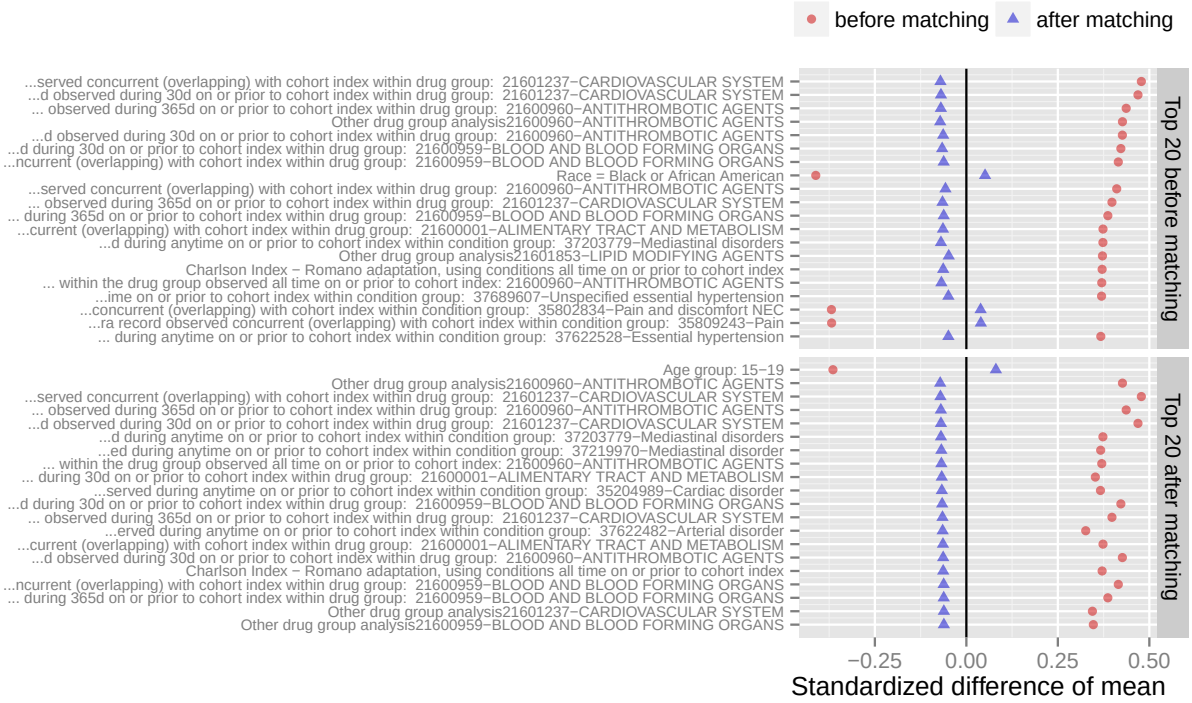
To evaluate whether our use of the propensity score is indeed making the two cohorts more comparable, we can compute the covariate balance before and after trimming, matching, and/or stratifying:

```
balance <- computeCovariateBalance(strata, cohortData, outcomeConceptId = 3)
```

```
plotCovariateBalanceScatterPlot(balance)
```




```
plotCovariateBalanceOfTopVariables(balance)
```



5 Outcome models

The outcome model is a model describing which variables are associated with the outcome.

5.1 Fitting the outcome model

In theory we could fit an outcome model without using the propensity scores. In this example we're fitting an outcome model using a Cox regression. The risk window is defined as time of exposure + 30 days:

```
outcomeModel <- fitOutcomeModel(outcomeConceptId = 3,
                                cohortData = cohortData,
                                riskWindowStart = 0,
                                riskWindowEnd = 30,
                                addExposureDaysToEnd = TRUE,
                                useCovariates = FALSE,
                                modelType = "cox",
                                stratifiedCox = FALSE)
outcomeModel
```

```
#> Model type: cox
#> Status: OK
#>
#> Prior variance: Inf
#>            Estimate lower .95 upper .95     logRr seLogRr
#> treatment  0.926463  0.688573  1.237000 -0.076381  0.1475
```

But of course we want to make use of the matching done on the propensity score:

```
outcomeModel <- fitOutcomeModel(outcomeConceptId = 3,
                                cohortData = cohortData,
                                subPopulation = strata,
                                riskWindowStart = 0,
                                riskWindowEnd = 30,
                                addExposureDaysToEnd = TRUE,
                                useCovariates = FALSE,
                                modelType = "cox",
                                stratifiedCox = TRUE)
outcomeModel
```

```
#> Model type: cox
#> Status: OK
#>
#> Prior variance: Inf
#>           Estimate lower .95 upper .95    logRr seLogRr
#> treatment  0.57143   0.34118   0.93530 -0.55962  0.2514
```

Note that we define the subpopulation to be only those in the *strata* object, which we created earlier by matching on the propensity score. We also now use a stratified Cox model, conditioning on the propensity score match sets.

One final refinement would be to use the same covariates we used to fit the propensity model to also fit the outcome model. This way we are more robust against misspecification of the model, and more likely to

remove bias. For this we use the regularized Cox regression in the Cyclops package. (Note that the treatment variable is automatically excluded from regularization.)

```
outcomeModel <- fitOutcomeModel(outcomeConceptId = 3,
                                cohortData = cohortData,
                                subPopulation = strata,
                                riskWindowStart = 0,
                                riskWindowEnd = 30,
                                addExposureDaysToEnd = TRUE,
                                useCovariates = TRUE,
                                modelType = "cox",
                                stratifiedCox = TRUE)
outcomeModel
```

```
#> Model type: cox
#> Status: OK
#>
#> Prior variance: 0.00576758505416434
#>           Estimate lower .95 upper .95    logRr seLogRr
#> treatment  0.52576   0.28105   0.93996 -0.64291  0.2964
```

5.2 Inpecting the outcome model

We can inspect more details of the outcome model:

```
summary(outcomeModel)
```

```
#> Model type: cox
#> Status: OK
#>
#> Counts
#>                     Comparator Treated
#> Nr. of persons            9006    9007
#> Nr. of events               54      49
#> Person time (days)      770542 1159332
#>
#> Model
#>          Nr. of betas Nr. of non-zero betas      Number of strata
#>                 16572                     4                  9015
#>
#> Coefficients
#>           Estimate lower .95 upper .95    logRr seLogRr
#> treatment  0.52576   0.28105   0.93996 -0.64291  0.2964
#>
#> Prior variance: 0.00576758505416434
```

```
coef(outcomeModel)
```

```
#> [1] -0.6429091
```

```
confint(outcomeModel)
```

```
#> [1] -1.26921959 -0.06191823
```

We can also see the covariates that ended up in the outcome model:

```
fullOutcomeModel <- getOutcomeModel(outcomeModel, cohortData)
head(fullOutcomeModel)
```

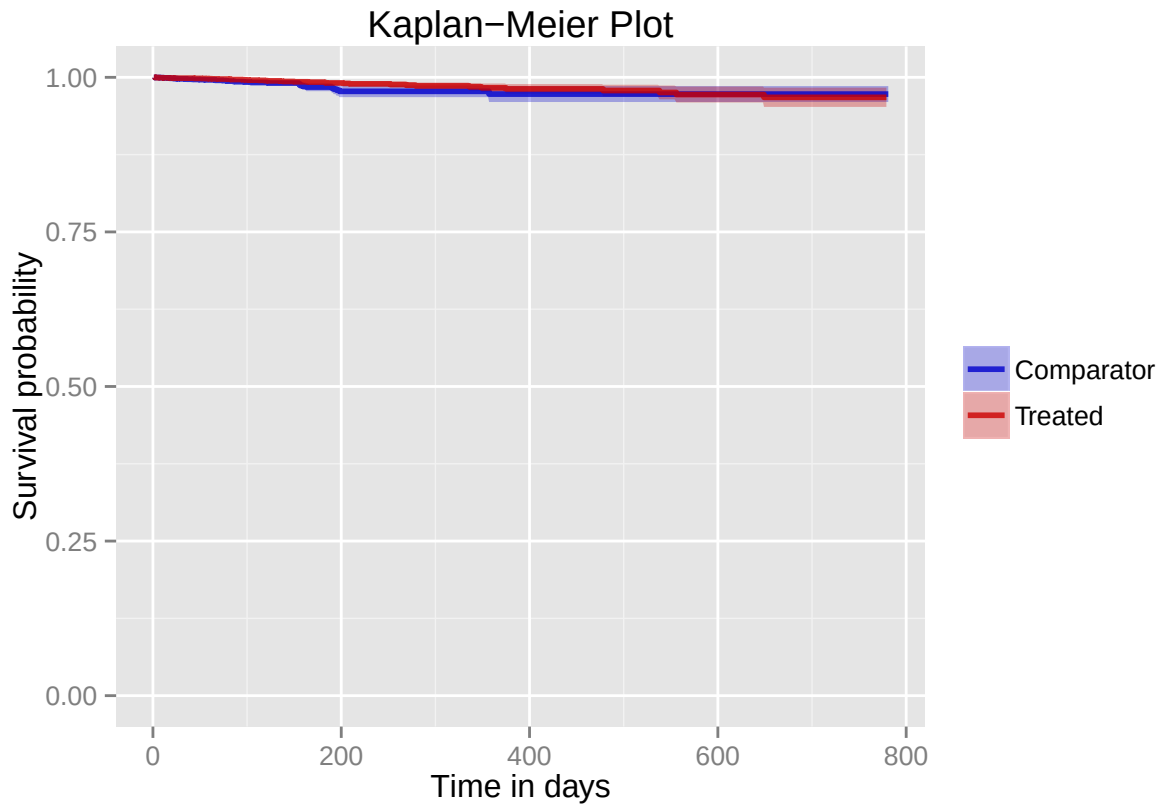
```
#>     coefficient   id                                covariateName
#> 1 -0.6429090577    1                              Exposure status
#> 3  0.0402141419 1001 ...rved in 365d on or prior to cohort index
#> 2  0.0383553325 1000 ...rved in 365d on or prior to cohort index
#> 4  0.0008649064 1004 ...rved in 365d on or prior to cohort index
```

5.3 Kaplan-Meier plot

We can create the Kaplan-Meier plot:

```
plotKaplanMeier(outcomeModel)
```

```
#> Warning in plotKaplanMeier(outcomeModel): The outcome model is stratified,
#> but the stratification is not visible in the plot
```



5.4 Attrition diagram

We can also investigate how we got to the study population by drawing the attrition diagram:

```
drawAttritionDiagram(outcomeModel)
```