Code

Personalized Medicine

Ruoqing Zhu

Last Updated: November 18, 2024

Overview

Precision medicine is a rapidly growing field in statistics, machine learning. It takes individual heterogeneity, environment, and lifestyles into consideration while making decisions to improve human health. Recent advances in large-scale biomedical databases and computational devices dramatically improve the prospect of optimizing an individual's treatment. The launch of the Precision Medicine Initiative (https://obamawhitehouse.archives.gov/precision-medicine) sparks much more research in discovering individualized treatment. In this tutorial, we will introduce three methods for personalized medicine: linear regression that models interactions, virtual twin model, and outcome weighted learning.

Basic Concept

Different from a regular regression or classification problem, a personalized medicine problem involves one more type of variable: the treatment label, denoted as A (action). In the simplest case, we may assume that $A \in \{-1, 1\}$, a binary label. However, the action can also be multiple categorical and continuous. The outcome we observe is denoted as R (reward). These notations are borrowed from a reinforcement learning literature. In a traditional medical problem, we often assume that the treatment effects (using a drug) on all subjects are the same. In certain sense, we may think about the following linear model:

$$E(R|X, A) = \boldsymbol{\beta}^T X + \alpha A.$$

In this case, those who take treatment $A = 1$ (e.g., taking a drug) will have $2 \times \alpha$ higher mean reward than treatment $A = 1$ (e.g., taking placebo). Hence, if $\alpha > 0$, everyone should use treatment 1, otherwise use -1. However, as medical science evolves, we know that this "one-size-fits-all" concept is not true in practice, and we should explore heterogeneous treatment effects. If we still restrict ourselves in a linear model framework, we could have

$$E(R|X,A) = \boldsymbol{\beta}^T X + A \cdot \boldsymbol{\alpha}^T X.$$

Here, the notation $A \cdot X$ means that we are using the interaction between A and each component in X. Note that α also becomes a vector to incorporate all information in X.

Our question in personalized medicine is: **Given a patient's covariate information X, which treatment label A provides better reward?**. Formally, we want to compare

$$E(R|X,A=1) \quad \text{vs.} \quad E(R|X,A=-1)$$

Hence, a simple strategy is to estimate the regression function and compare the expected rewards. However, we should also notice that, as long as $\alpha^T X$ is positive, $A = 1$ is more beneficial. Hence, this leads to two main strategies:

- Indirect-learning approach: learn the regression model to estimate the expected reward $E(R|X,A)$, and suggest treatment by comparing the expected reward under $A = 1$ or -1.
- Direct-learning approach: directly learn the treatment rule $\text{sign}(\boldsymbol{\alpha}^T X)$ without estimating other parameters $\boldsymbol{\beta}$ since they are not useful for choosing treatment.

We will explore these two strategies with a simulation study and use Lasso to perform the analysis. In this problem, 10 variables are involved in $\boldsymbol{\beta}$, while two variables are involved in the decision rule parameter $\boldsymbol{\alpha}$.

Hide

```
set.seed(1)
n = 100
p = 200
X = matrix(rnorm(n*p), n, p)
A = rbinom(n, 1, 0.5)*2-1
beta = c(rep(0.2, 10), rep(0, p-10))
alpha = c(rep(0, p-2), 0.75, -0.75)
R = X %*% beta + A * (X %*% alpha) + rnorm(n, sd = 0.5)

testn = 500
testX = matrix(rnorm(testn*p), testn, p)
TRUE_BEST = sign(testX %*% alpha)
```

Indirect-learning with Lasso

The idea of direct learning is very simple. Let's just proceed with the Lasso model:

Hide

```
  library(glmnet)
## Loading required package: Matrix
## Loaded glmnet 4.1-8

  # fit two regression models
  Apos.fit <- cv.glmnet(x = cbind(X, A)[A == 1,], y = R[A == 1])
  Aneg.fit <- cv.glmnet(x = cbind(X, A)[A == -1,], y = R[A == -1])

  # compare the models for new data
  Apos.pred = predict(Apos.fit, newx = cbind(testX, 1))
  Aneg.pred = predict(Aneg.fit, newx = cbind(testX, -1))

  # the predicted treatment label
  indirect.pred = sign(Apos.pred - Aneg.pred)

  # the accuracy
  mean(indirect.pred == TRUE_BEST)
## [1] 0.876
```

This seems to be fairly accurate. We got 88% of accuracy, meaning that, for around 88% of the new patients, we suggested the better treatment.

Direct-learning with Lasso

Let's explore an interesting fact about our previous equation. This method was proposed by Tian et. al, 2014 (https://www.tandfonline.com/doi/full/10.1080/01621459.2014.951443). Suppose we do not use R as our outcome of interest. Instead, we use AR, the product of treatment label and the reward. Then, we have

$$\begin{aligned} E(AR|X) =& E(A|X) \cdot \boldsymbol{\beta}^T X + E(A^2|X) \cdot \boldsymbol{\alpha}^T X. \\ =& \Big[P(A=1|X) - P(A=-1|X)\Big] \cdot \boldsymbol{\beta}^T X + \boldsymbol{\alpha}^T X \end{aligned}$$

If our study is collected from a **randomized trial**, meaning that in the observed data, the treatment decision A does not depend on the covariate information, and is balanced, i.e., $P(A=1|X) = P(A=-1|X) = 0.5$. Then the first term is zero. And we only have the second term

$$E(AR|X) = \boldsymbol{\alpha}^T X$$

This suggests a very simple idea of directly modeling the best treatment rule.

Hide

```
  # fitting the direct learning model
  direct.fit <- cv.glmnet(x = X, y = R*A)

  # direct learning predicted label
  direct.pred <- sign(predict(direct.fit, testX))

  # the accuracy
  mean(direct.pred == TRUE_BEST)
## [1] 0.916
```

This time, the accuracy 92% is slightly better than the previous one.

Observational Study

There is an importance remark we should make is that the assumption of **randomized trial** is crucial. Without this assumption, the first term cannot be removed. This is very frequent in observational study. However, we can still modify the previous equation using a **inverse propensity weighting**,

$$
\begin{aligned}
& E\left(\frac{A}{P(A|X)}R \mid X\right) \\
=&\left[\frac{1}{P(A=1|X)}P(A=1|X)+\frac{-1}{P(A=-1|X)}P(A=-1|X)\right]\cdot\boldsymbol{\beta}^T X+ \\
& \left[\frac{1^2}{P(A=1|X)}P(A=1|X)+\frac{(-1)^2}{P(A=-1|X)}P(A=-1|X)\right]\boldsymbol{\alpha}^T X \\
=&\left[1-1\right]\cdot\boldsymbol{\beta}^T X+\left[1+1\right]\boldsymbol{\alpha}^T X \\
=&\mathbf{2}\boldsymbol{\alpha}^T X
\end{aligned}
$$

So the first term can still be canceled out. Then, our job simply involves one more step: estimate the **propensity scores** $P(A|X)$, using, e.g. a logistic regression, and then plug into the previous linear regression estimation using the subject weights $\widehat{P}(A=a_i|X=x_i)$. This weighting idea is frequently used in the literature. We shall see this again the outcome weighted learning section.

Virtual Twins

When the observed treatment label assignment mechanism is unknown, indirect approach can still have advantage. A popular approach called the Virtual Twins model was proposed by Foster et al. (2011) (https://doi.org/10.1002/sim.4322). This is a indirect learning appraoch. The key idea is to use random forests to learn the regression models, and use a single tree model to summarize the decision and make it interpretable. This example using sepsis data is mainly based on the vignettes (https://cran.r-project.org/web/packages/aVirtualTwins/vignettes/full-example.html) of the `aVirtualTwins` R package. We also need the packages for random forests and CART.

Let's consider a simulated data set derived from the SIDES package (http://biopharmnet.com/subgroup-analysis-software/). The data in `.csv` format can be downloaded from our course website. In this data set, 470 patients and 14 variables are collected. The variables are listed below.

- `Health` : Health outcome (larger the better)
- `THERAPY` : 1 for active treatment, 0 for the control treatment
- `TIMFIRST` : Time from first sepsis-organ fail to start drug
- `AGE` : Patient age in years
- `BLLPLAT` : Baseline local platelets
- `blSOFA` : Sum of baseline sofa score (cardiovascular, hematology, hepatorenal, and respiration scores)
- `BLLCREAT` : Base creatinine
- `ORGANNUM` : Number of baseline organ failures
- `PRAPACHE` : Pre-infusion apache-ii score
- `BLGCS` : Base GLASGOW coma scale score
- `BLIL6` : Baseline serum IL-6 concentration
- `BLADL` : Baseline activity of daily living score

- `BLLBILI` : Baseline local bilirubin
- `BEST` : The true best treatment suggested by doctors. **You should not use this variable when fitting models!**

For each patient, sepsis was observed during their hospital stay. Hence, one of the two treatments (indicated by variable `THERAPY`) must be chosen to prevent further adverse events. After the treatment, the patient's health outcome (`Health`) was measured, with a larger value being the better outcome. The `BEST` variable is a doctor suggested best treatment, which is not observed. This can be regarded as the unknown truth.

Run the cell below to load the data set and display the first few rows. It will only take a few seconds to complete. Make sure that you have the working directory setup correctly. You can do this by putting your `.rmd` file and the `.csv` file in the same folder, then open your `.rmd` file to pop up RStudio.

Hide

```
Sepsis <- read.csv("Sepsis.csv")
# remove the first column, which is observation ID
Sepsis = Sepsis[, -1]
head(Sepsis)
```

	Health <dbl>	THER... <int>	PRAPA... <int>	AGE <dbl>	BL... <int>	ORGAN... <int>	BLIL6 <dbl>	BLLPLAT <dbl>	BLLBILI <dbl>
1	-3.188260	0	19	42.921	15	1	301.80	191.0000	2.913416
2	-3.546312	0	48	68.818	11	2	118.90	264.1565	0.400000
3	1.188689	1	20	68.818	15	2	92.80	123.0000	5.116471
4	2.693554	1	19	33.174	14	2	1232.00	244.0000	3.142092
5	3.007590	0	48	46.532	3	4	2568.00	45.0000	4.052668
6	3.870876	1	21	56.098	14	1	162.65	137.0000	0.500000

6 rows | 1-10 of 15 columns

Virtual Twins: Step 1

We will fit two random forests to model the outcome `Health`: one model is based on all patients who received treatment (`THERAPY`) 1, and the other model is based on all patients who received treatment 0 (corresponding to our previous label -1). Denote these two models as $\widehat{f}_1(x)$ and $\widehat{f}_0(x)$, respectively. There is one more step we need to do, compared with our previous linear regression example, that is to produce the in-sample prediction of all training data based on these two models. This is because we need to use these training data predictions to learn an interpretable tree model in Step 2. The key mechanism we use here is the out-of-bag predictions from random forests. This prevents over-fitting of in-sample observations. In the meantime, we should also tune both `mtry` and `nodesize`, but that step will be skipped in the demonstration. Read the following code.

Hide

```
  library(randomForest)
## randomForest 4.7-1.1
## Type rfNews() to see new features/changes/bug fixes.

  # fit model for treatment 0
  model0 <- randomForest(Health ~ . - BEST,
                         data = Sepsis[Sepsis$THERAPY == 0, ],
                         ntree=2000,
                         mtry = 5,
                         nodesize = 5)

  # fit model for treatment 1
  model1 <- randomForest(Health ~ . - BEST,
                         data = Sepsis[Sepsis$THERAPY == 1, ],
                         ntree=2000,
                         mtry = 5,
                         nodesize = 5)

  # out-of-bag prediction for treatment 0 group with model 0
  model0.treat0 = model0$predicted

  # out-of-bag prediction for treatment 1 group with model 1
  model1.treat1 = model1$predicted

  # prediction for treatment 1 group with model 0
  model0.treat1 = predict(model0, Sepsis[Sepsis$THERAPY == 1, ])

  # prediction for treatment 0 group with model 1
  model1.treat0 = predict(model1, Sepsis[Sepsis$THERAPY == 0, ])

  # combine predictions together
  pred.treat0 = rep(NA, nrow(Sepsis))
  pred.treat0[Sepsis$THERAPY == 0] = model0.treat0
  pred.treat0[Sepsis$THERAPY == 1] = model0.treat1

  pred.treat1 = rep(NA, nrow(Sepsis))
  pred.treat1[Sepsis$THERAPY == 0] = model1.treat0
  pred.treat1[Sepsis$THERAPY == 1] = model1.treat1

  # which is better?
  pred.best = (pred.treat1 > pred.treat0)

  # is this good?
  mean(pred.best == Sepsis$BEST)
## [1] 0.8148936
```

So it looks like we got pretty good accuracy, 81%. But random forest is not interpretable, and a doctor cannot directly know why the patient was assigned to a certain treatment.

Virtual Twins: Step 2

In this second step, we will construct a single-tree model (CART) to represent the choice of best treatment. The idea is very simple, use all of our covariates to model the best label we learned.

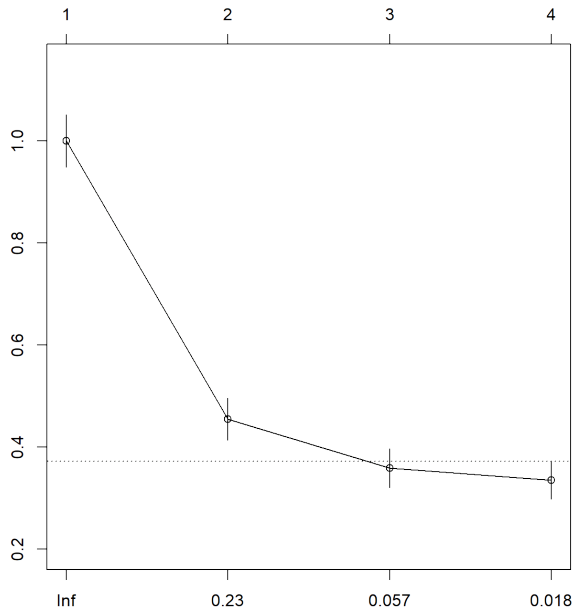
Hide

```
library(rpart)

# fit a decision tree to predict pred.best,
# excluding BEST, Health, and THERAPY from the dataset
best.label = ifelse(pred.treat1 > pred.treat0, "Treatment 1", "Treatment 0")
best.label = as.factor(best.label)

rpart.fit <- rpart(best.label ~ . - BEST - Health - THERAPY, data = Sepsis)

# in the coming cells, we will prune the tree
# start by plotting the cross-validation relative error at different tree sizes
plotcp(rpart.fit)
```

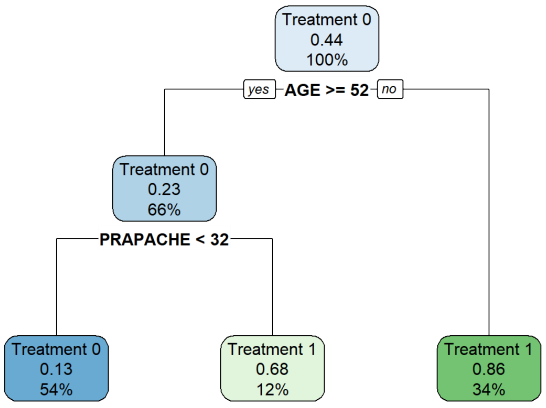


Decide a `cp` value, and we can view the fitted tree model.

Hide

```
# cut the tree
cutted.tree = prune(rpart.fit, cp = 0.04)

library(rpart.plot)
# make a good plot
rpart.plot(cutted.tree)
```



Hence, the conclusion is that, when $\text{Age} < 52$, we should use treatment 1. While for $\text{Age} \geq 52$, if the apache score is less than 32, we should use treatment 0 and otherwise treatment 1.

Outcome Weighted Learning

Outcome Weighted Learning (Zhao et. al, 2012 (https://www.tandfonline.com/doi/full/10.1080/01621459.2012.695674)) is a direct learning approach. The logic of outcome weighted learning is a bit more tricky, since involves dealing with the probabilities $P(A = 1|X)$ and $P(A = -1|X)$, when it is not a randomized trial. We often call this observational data. Making inference on observational data is what casual inference (https://scholar.google.com/scholar?hl=en&as_sdt=0%2C14&q=casual+inference&btnG=) mainly concerns. Formally, let's be precise on what is our target of estimation. In Qian and Murphy (2011) (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3110016/), we first define the **value function** of a decision rule $d(x)$:

$$\begin{aligned} \square(d) &= E^d(R) \\ &= E_X\left[E_R[R \mid X, A = d(X)]\right] \end{aligned}$$

This means that we first (inner expectation E_R) let everyone take the treatment label suggested by the decision rule d, i.e., $A = d(X)$, and obtain the expected reward, and then average over the entire population (E_X). Hence, our goal is to maximize $\square(d)$ by searching for the best $d(X)$

$$d_{\text{opt}} = \arg\max_d \square(d)$$

However, we do not observe the distribution that everyone follows the suggested $d(X)$ treatment. Instead, our observational study is collected under its own mechanism, or distribution, denoted as $(R, X, A) \sim \mu$. Our target is to estimate the value function under a restricted policy that gives $(R, X, A = d(X)) \sim \mu^d$. A tool we could utilize is the Radon-Nikodym theorem (https://en.wikipedia.org/wiki/Radon%E2%80%93Nikodym_theorem), which suggests that

$$
\begin{aligned}
\square(d) &= \int R d\mu^d \\
&= \int R \frac{d\mu^d}{d\mu} d\mu \\
&= \int \frac{R \cdot 1\{A = d(X)\}}{P(A|X)} d\mu
\end{aligned}
$$

Now, since $d\mu$ is observed, we can use the sample (empirical) estimation of this value function.

$$
\hat{d}_{\text{opt}} = \arg\max_d \frac{1}{n} \sum_{i=1}^{n} \frac{R_i \cdot 1\{A_i = d(X_i)\}}{\widehat{P}(A_i|X_i)}.
$$

We could specify $d(X_i)$ as a linear function, or nonlinear function using the Reproducing Kernel Hilbert Space $\square$, and that would require adding a penalty term:

$$
\hat{d}_{\text{opt}} = \arg\max_d \frac{1}{n} \sum_{i=1}^{n} \frac{R_i \cdot 1\{A_i = d(X_i)\}}{\widehat{P}(A_i|X_i)} - \lambda \|d\|_{\square}^2.
$$

Lastly, we can change this formulation to minimization by switching the equal sign to unequal sign:

$$
\hat{d}_{\text{opt}} = \arg\min_d \frac{1}{n} \sum_{i=1}^{n} \frac{R_i \cdot 1\{A_i \neq d(X_i)\}}{\widehat{P}(A_i|X_i)} + \lambda \|d\|_{\square}^2.
$$

The interesting view of this formulation is that, this becomes a classification problem, where * $d(X_i)$ is the decision rule * A_i is the class label * $\frac{R_i}{\widehat{P}(A_i|X_i)}$ is the subject specific weight

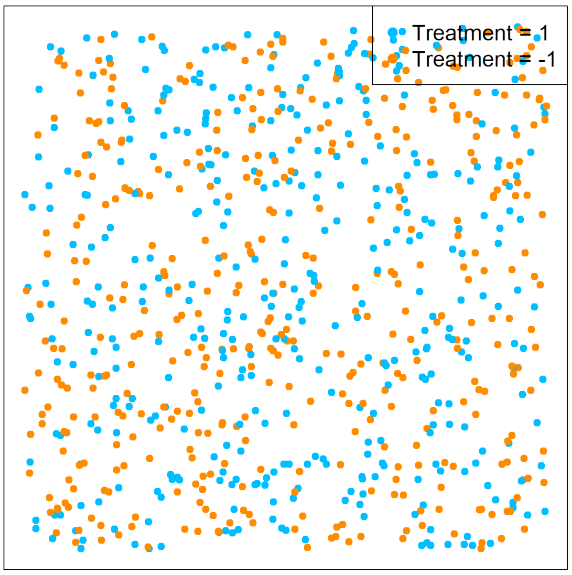
Hence, our goal is to perform classification to model the labels, but we want to encourage the model to do better job for those with high rewards. Let's look at an example, where we assume that $P(A|X) = 1/2$.

Hide

```
set.seed(1)
n = 800
p = 2
x1 <- runif(n, 0, 1)
x2 <- runif(n, 0, 1)
a = rbinom(n, 1, 0.5)*2-1
side = sign(x2 - sin(2*pi*x1)/3-0.5)

R <- rnorm(n, mean = ifelse(side==a, 1.5, 0.5), sd = 0.5)

# the observed data, with class labels
par(mar=rep(1,4))
plot(x1, x2, pch = 19, xaxt = "n", yaxt = "n", xlab = "", ylab = "",
     col = ifelse(a==1, "deepskyblue", "darkorange"))
legend("topright", c("Treatment = 1", "Treatment = -1"), pch = 19,
       col = c("deepskyblue", "darkorange"), cex = 1.3)
```



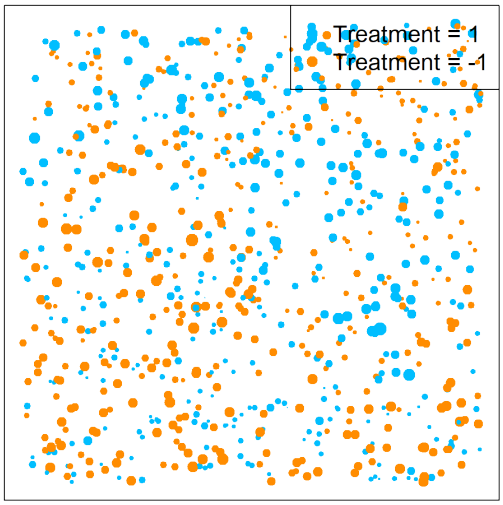
Now lets add the reward as weights, indicated by the size. Its clear that on the lower side, treatment -1 is more dominating and one the upper side, 1 is more dominating. The true decision line is also given below.

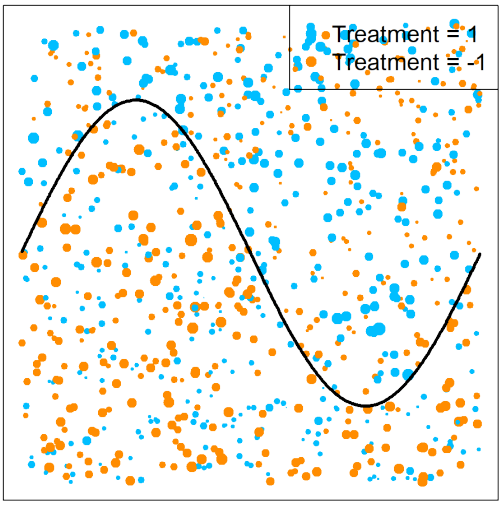
Hide

```
par(mfrow = c(1, 2))
par(mar=rep(1,4))

plot(x1, x2, pch = 19, xaxt = "n", yaxt = "n", xlab = "", ylab = "",
     cex = (R+1.5)/3, col = ifelse(a==1, "deepskyblue", "darkorange"))
legend("topright", c("Treatment = 1", "Treatment = -1"), pch = 19,
       col = c("deepskyblue", "darkorange"), cex = 1.3)

plot(x1, x2, pch = 19, xaxt = "n", yaxt = "n", xlab = "", ylab = "",
     cex = (R+1.5)/3, col = ifelse(a==1, "deepskyblue", "darkorange"))
legend("topright", c("Treatment = 1", "Treatment = -1"), pch = 19,
       col = c("deepskyblue", "darkorange"), cex = 1.3)
x1_grid = sort(x1)
lines(x1_grid, sin(2*pi*x1_grid)/3+0.5, lwd = 3)
```





Let's use an R package to solve this problem. We first try a linear kernel.

Hide

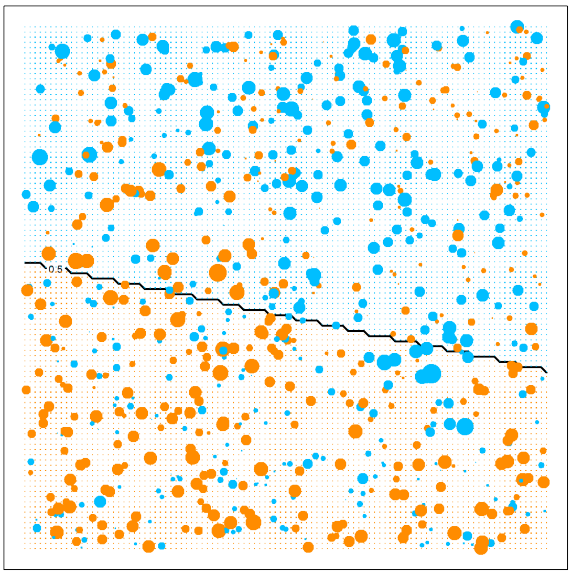
```
  # you need to install the package from github
  # devtools::install_github("cran/DTRlearn")

  library(DTRlearn)
## Loading required package: kernlab
##
## Attaching package: 'kernlab'
## The following object is masked from 'package:scales':
##
##     alpha
## Loading required package: MASS
## Loading required package: ggplot2
##
## Attaching package: 'ggplot2'
## The following object is masked from 'package:kernlab':
##
##     alpha
## The following object is masked from 'package:randomForest':
##
##     margin
  par(mfrow = c(1, 1))
  # fit OWL using a linear kernel
  owl.fit = wsvm(X = cbind(x1, x2), A = a, wR = R-min(R), kernel = "linear")

  # plot results
  xgrid1 = seq(0, 1, 0.01)
  xgrid2 = seq(0, 1, 0.01)
  mesh = expand.grid(xgrid1, xgrid2)
  colnames(mesh) = c("x1", "x2")

  meshlabel = predict(owl.fit, data.matrix(mesh))

  # the decision line
  contour(xgrid1, xgrid2, xaxt = "n", yaxt = "n", lwd = 2,
          matrix(meshlabel, length(xgrid1), length(xgrid2)), levels = c(0.5))
  points(mesh, pch=".", cex=1.2,
         col=ifelse(meshlabel==1,  "deepskyblue", "darkorange"))
  points(x1, x2, pch = 19, cex = (R+0.5)/1.5,
         col = ifelse(a==1, "deepskyblue", "darkorange"))
```

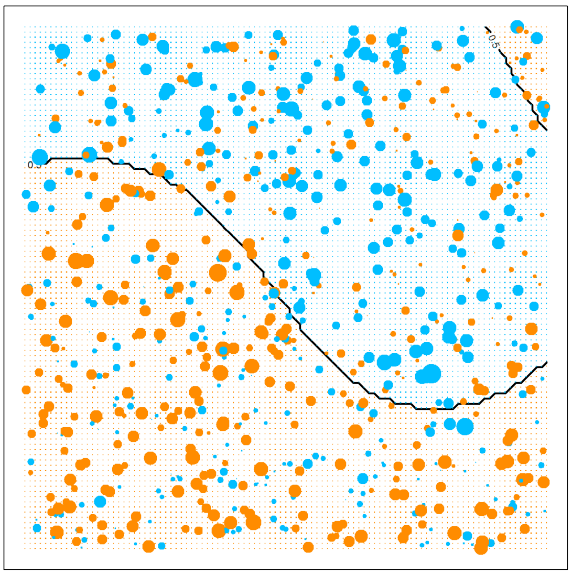


Clearly, this is not ideal. Hence, we can try rbf kernel

Hide

```
  owl.fit = wsvm(X = cbind(x1, x2), A = a, wR = R-min(R), kernel = "rbf", sigma =
2)
  meshlabel = predict(owl.fit, data.matrix(mesh))

  # the decision line
  contour(xgrid1, xgrid2, xaxt = "n", yaxt = "n", lwd = 2,
          matrix(meshlabel, length(xgrid1), length(xgrid2)), levels = c(0.5))
  points(mesh, pch=".", cex=1.2,
         col=ifelse(meshlabel==1,  "deepskyblue", "darkorange"))
  points(x1, x2, pch = 19, cex = (R+0.5)/1.5,
         col = ifelse(a==1, "deepskyblue", "darkorange"))
```



We can also use random forests to perform this weighted classification.

Hide

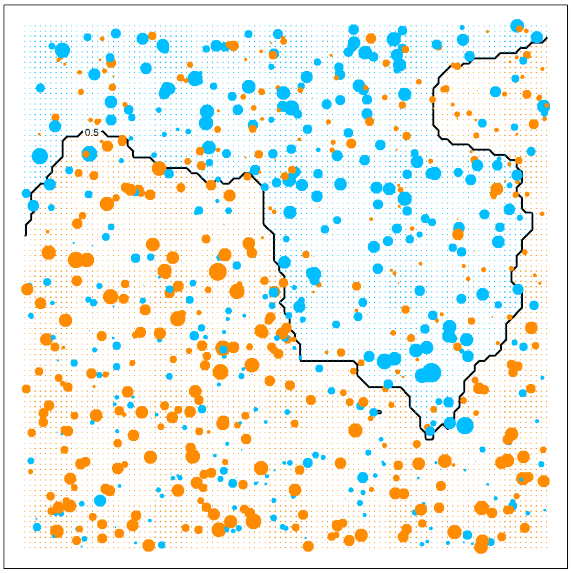
```
  # fit OWL using a random forest
  library(RLT)
## RLT and Random Forests v4.2.6
## pre-release at github.com/teazrq/RLT

  owl.fit = RLT(cbind(x1, x2), as.factor(a), model = "classification",
                obs.w = R - min(R) + 0.1, ntrees = 1000,
                resample.prob = 0.75, resample.replace = FALSE,
                nmin = 45, mtry = 2)

  meshlabel = predict(owl.fit, mesh)$Prediction

  # owl.fit = wsvm(X = cbind(x1, x2), A = a, wR = R-min(R), kernel = "rbf", sigma =
1.5)
  # meshlabel = predict(owl.fit, data.matrix(mesh))

  # the decision line
  contour(xgrid1, xgrid2, xaxt = "n", yaxt = "n", lwd = 2,
          matrix(meshlabel, length(xgrid1), length(xgrid2)), levels = c(0.5))
  points(mesh, pch=".", cex=1.2,
         col=ifelse(meshlabel==1,  "deepskyblue", "darkorange"))
  points(x1, x2, pch = 19, cex = (R+0.5)/1.5,
         col = ifelse(a==1, "deepskyblue", "darkorange"))
```



Other Settings

Personalized medicine does not just end here. There are many settings we need to consider, for example when the decision is continuous (Chen, et. al, 2016 (https://www.tandfonline.com/doi/full/10.1080/01621459.2016.1148611)), subject to censoring (Cui, et. al, 2022 (https://arxiv.org/pdf/2001.09887.pdf)), dynamic decisions (Chakraborty and Murphy, 2014 (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4231831/)). And the problem eventually falls into the domain of reinforcement learning when there are a very large number of decisions to make.