Loss estimation

Matej Miočić

10/5/2022

Setup

Generate toy dataset

First, we prepare the generator for our toy binary classification data, which has 8 independent variables, 3 of which are unrelated to the target variable. Because we generate the data, we know all the properties of the data generating process, which will allow us to study the quality of our loss estimation.

Note that we'll be using negative log-loss (smaller is better) throughout this homework.

```r
library(ggplot2)

toy_data <- function(n, seed = NULL) {
set.seed(seed)
x <- matrix(rnorm(8 * n), ncol = 8)
z <- 0.4 * x[,1] - 0.5 * x[,2] + 1.75 * x[,3] - 0.2 * x[,4] + x[,5]
y <- runif(n) > 1 / (1 + exp(-z))
return (data.frame(x = x, y = y))
}
log_loss <- function(y, p) {
-(y * log(p) + (1 - y) * log(1 - p))
}
```

A proxy for true risk

We'll be using this huge dataset as a proxy for the DGP and determining the ground-truth true risk of our models. Of course, this is itself just an estimate, but the dataset is large enough so that a model's risk on this dataset differs from its true risk at most on the 3rd decimal digit. How did I determine that 100000 is enough to reduce the error to the 3rd decimal digit?

A: We know that using a dataset with 100000 instances is enough by using VC theory. With probability 1 - delta, we can compute the upper bound, which is approximately 0.019, since the VC dimension of Logistic Regression is 1 more than the number of features (in our case 9). This means that the error is approximately reduced to 3rd decimal digit.

```r
df_dgp <- toy_data(100000, 0)
```

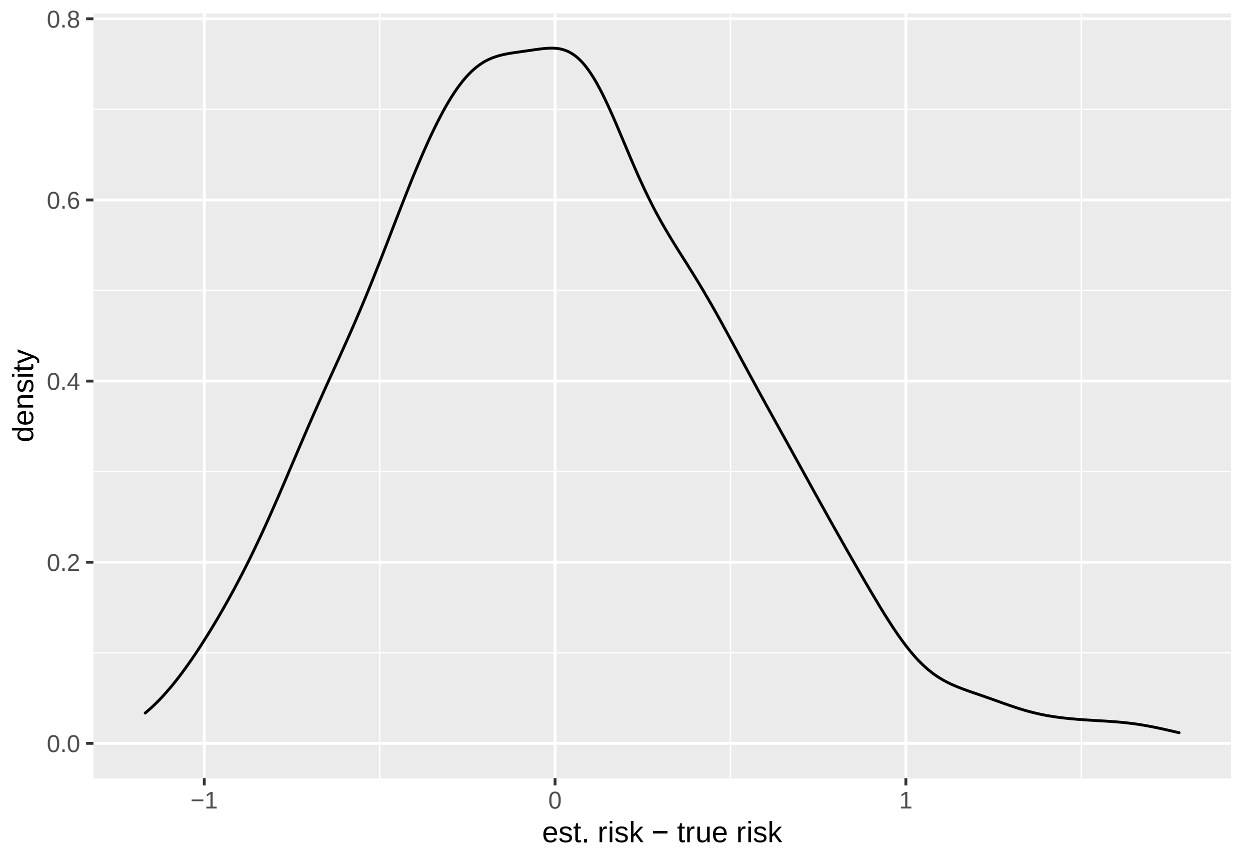
Holdout estimation

Holdout estimation or, as it is commonly referred to, train-test splitting, is the most common approach to estimating a model's risk. The first and most important thing to understand is that the model's risk on the test data is just an estimate of the model's true risk. As such, it should always contain some sort of quantification of uncertainty, such standard errors or 95% confidence intervals. We've learned a couple of techniques for doing this, but in this homework we'll use the most simple one - standard errors and 95% CI based on the asymptotic argument that the mean loss will be normally distributed with variance n times lower than the variance of the loss. When interpreting the results, we will also inspect the coverage of these confidence intervals to see how this simple approach does in practice. In the remainder of this section we will investigate some of the sources of variability and bias that contribute to the difference between the test data risk estimate and the true risk of a model trained on the training data.

Model loss estimator variability due to test data variability

First, we'll investigate how our test data risk estimate varies with test data: 1. Generate a toy dataset with 50 observations. 2. Train model h using a learner (I'm using a Bernoulli-logit GLM or logistic regression). 3. Compute the true risk proxy for h using the huge dataset. 4. Generate a toy dataset with 50 observations, estimate the risk of h using this dataset, compute the standard error of the estimate, record if the 95% CI contains the true risk. 5. Repeat (4) 1000 times. 6. Plot a density estimate of the differences between estimates and the true risk proxy. Compute the true risk proxy. Compute the average difference between the estimate and the true risk (bias). Compute the true risk of always making 0.5 - 0.5 predictions (this will help us interpret the variability). Compute the median standard error of the estimates. Compute the % of times that the 95%CI contained the true risk.

```r
df_dgp <- toy_data(100000, 0)
toy_dataset <- toy_data(50, 1)

h <- glm(y ~ ., data=toy_dataset, family="binomial")

y_pred <- predict(h, df_dgp[,-9], type="response")

true_risk_proxy <- mean(log_loss(df_dgp$y, y_pred))
true_risk_5050 <- mean(log_loss(df_dgp$y, 0.5))

standard_errors <- c()
in95CI <- c()
risk_estimates <- c()

# starting with 2, to avoid having same seed with original toy_dataset
for(i in 2:1001){
  toy_dataset <- toy_data(50, i)
  y_pred <- predict(h, toy_dataset[,-9], type="response")
  loss <- log_loss(toy_dataset$y, y_pred)
  risk <- mean(loss)
  se <- sd(loss)/sqrt(length(loss))

  in95CI <- c(in95CI, risk + 1.96 * se > true_risk_proxy && risk - 1.96 * se < true_risk_proxy)
  risk_estimates <- c(risk_estimates, risk)
  standard_errors <- c(standard_errors, se)
}

differences <- risk_estimates - true_risk_proxy

ggplot(data.frame(differences), aes(x=differences)) +
  geom_density() +
  xlab("est. risk - true risk") +
  ylab("density")
```



```
## True risk proxy: 1.4547

## Mean difference: 8e-04

## 0.5-0.5 baseline true risk: 0.6931

## Median standard error: 0.4825

## Percentage of 95CI that contain the true risk proxy: 89.0
```

What do we see and what are the implications for practical use? Without experimentation, try to answer: How would these results change if the training set was larger/smaller? How would these results change if the test set was larger/smaller?

A: We tried to estimate the true risk with resampling from data generating process 1000 times. Since our model h has a bigger true risk than the one in reference, we obtained a bigger median standard deviation, although we still obtained a small mean risk difference. From results we see that 89% of the 95% CI contained the true risk proxy. If we increase the training set we hypothesise that the mean difference and the median standard error would decrease, since true risk proxy would be smaller. If we increase the test data, we think that risk estimates would be closer to the true risk and therefore outputing smaller difference.

Overestimation of the deployed model's risk

In practice we rarely deploy the model trained only on the training data. Similarly, we nevery deploy any of the k models that are learned during k-fold cross-validation. Instead, we use the estimation as a means to select the best learner and then deploy the model trained on more data, typically all the available data. More data typically means that the learner will produce a model with lower true risk. Let's demostrate this: 1. Generate two toy datasets, each with 50 observations. 2. Train model h1 using a learner and the first dataset only. 3. Train model h2 using the same learner and both datasets combined for a total of 100 training observations. 4. Compute the true risk proxies for h1 and h2 using the huge dataset. 5. Repeat (1-4) 50 times. 6. Compute a summary of the differences between the true risk proxies of h1 and h2.

```
df_dgp <- toy_data(100000, 0)
differences <- c()

for(i in 1:50){
  toy_dataset1 <- toy_data(50, i)
  toy_dataset2 <- toy_data(50, i+50)

  h1 <- glm(y ~ ., data=toy_dataset1, family="binomial")
  h2 <- glm(y ~ ., data=rbind(toy_dataset1, toy_dataset2), family="binomial")

  y_pred1 <- predict(h1, df_dgp[,-9], type="response")
  y_pred2 <- predict(h2, df_dgp[,-9], type="response")

  true_risk_proxy1 <- mean(log_loss(df_dgp$y, y_pred1))
  true_risk_proxy2 <- mean(log_loss(df_dgp$y, y_pred2))

  differences <- c(differences, true_risk_proxy1-true_risk_proxy2)
}

summary(differences)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -0.07669  0.02786  0.12974  0.17926  0.21089  0.86056
```

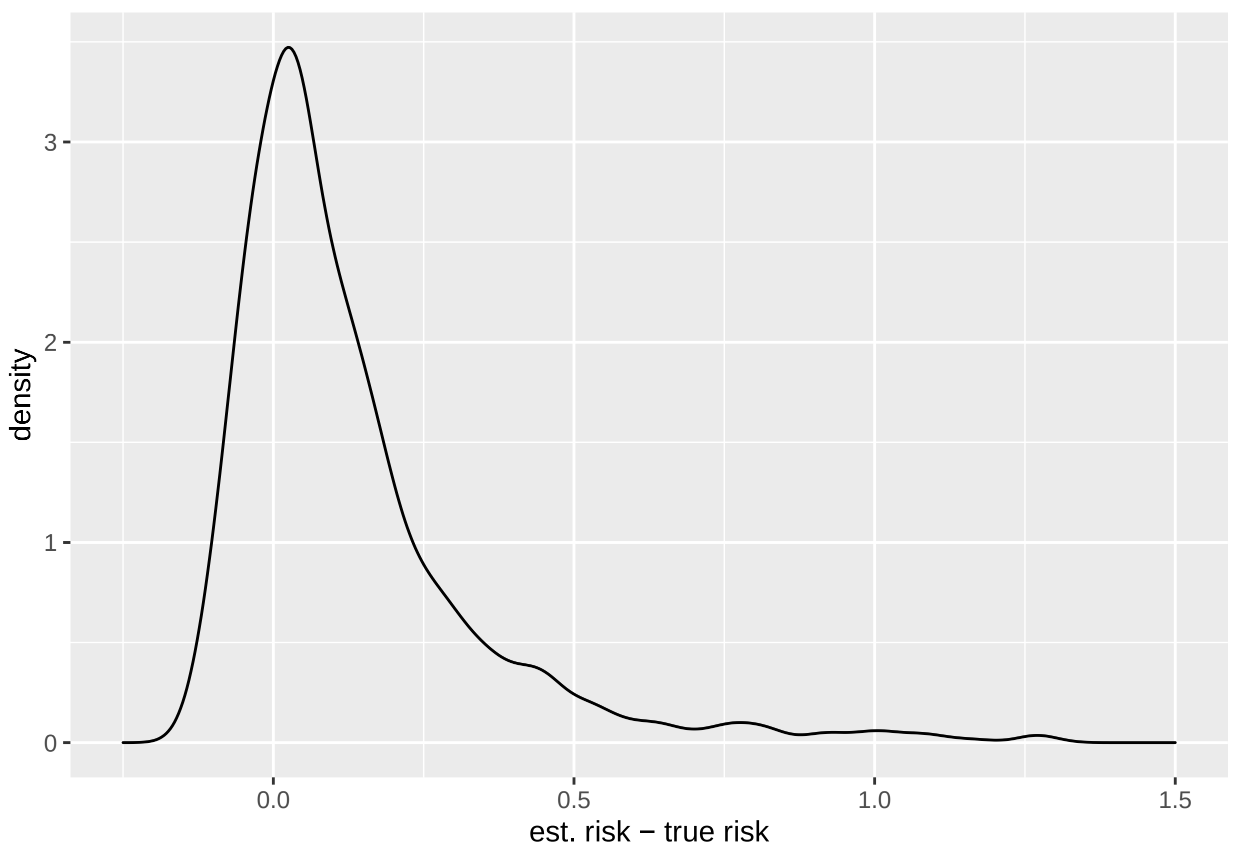
What do we see and what are the implications for practical use? Without experimentation, try to answer: How would these results change if the data sets were larger/smaller?

A: From results we see that with the additional 50 instances we reduce the true_risk_proxy on average by 0.17. If our 2 datasets were both larger, we hypothesise, that the differences would be smaller, since datasets would be more similar, have less variance.

Loss estimator variability due to split variability

In a practical application of train-test splitting, we would choose a train-test split proportion, train on the training data, and test on the test data. We would then use this result on the test data as an estimate of the true risk of the model trained on all data. From the experiments so far, we can gather that this estimate will be biased, because the tested model is trained on less data than the model we are interested in. It will also have variance due to which observations ended up in the training set and which in the test set. To that we can add the most basic source of variability - the variability of the losses across observations. Let's explore this: 1. Generate a toy dataset with 100 observations. 2. Train model h0 using some learner on all 100 observations and compute its true risk proxy using the huge dataset. 3. Split the dataset into 50 training and 50 test observations at random. 4. Train model h on the training set and use its test set risk as an estimate of h0's true risk. Record if the 95% CI of the estimate contains the true risk of h0. 5. Repeat (3-4) 1000 times. 6. Plot a density estimate of the differences between estimates and the true risk proxy. Compute the true risk proxy. Compute the average difference between the estimate and the true risk (bias). Compute the median standard error of the estimates. Compute the % of times that the 95%CI contained the true risk.

```r
train_test_split <- function(data, size, seed = NULL) {
  set.seed(seed)
  split <- sample.int(n = nrow(data), size = size, replace = F)
}

standard_errors <- c()
in95CI <- c()
risk_estimates <- c()

df_dgp <- toy_data(100000, 0)
toy_dataset <- toy_data(100, 1)

h0 <- glm(y ~ ., data=toy_dataset, family="binomial")
y_pred <- predict(h0, df_dgp[,-9], type="response")
true_risk_proxy <- mean(log_loss(df_dgp$y, y_pred))

for(i in 1:1000){
  split = train_test_split(toy_dataset, 50, i)
  train <- toy_dataset[split, ]
  test  <- toy_dataset[-split, ]

  h <- glm(y ~ ., data=train, family="binomial")
  y_pred <- predict(h, test[,-9], type="response")

  loss <- log_loss(test$y, y_pred)
  risk <- mean(loss)
  se <- sd(loss)/sqrt(length(loss))


  in95CI <- c(in95CI, risk + 1.96 * se > true_risk_proxy && risk - 1.96 * se < true_risk_proxy)
  risk_estimates <- c(risk_estimates, risk)
  standard_errors <- c(standard_errors, se)
}
differences <- risk_estimates - true_risk_proxy

ggplot(data.frame(differences), aes(x=differences)) +
  geom_density() +
  xlab("est. risk - true risk") +
  ylab("density") +
  xlim(-0.25, 1.5)
```



```
## True risk proxy: 0.5332
```

```
## Mean difference: 0.1857
```

```
## Median standard error: 0.1187
```

```
## Percentage of 95CI that contain the true risk proxy: 91.4
```

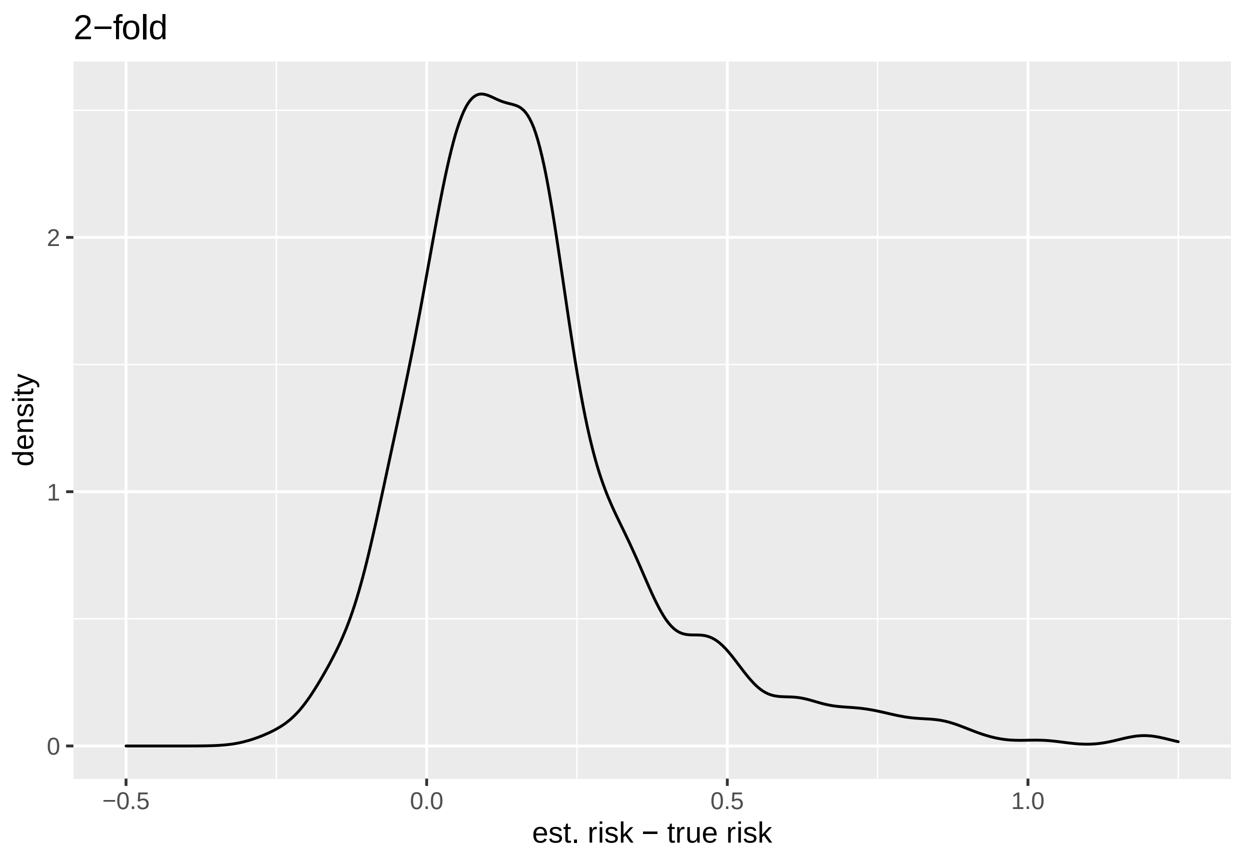
What do we see and what are the implications for practical use? Without experimentation, try to answer: How would these results change if the data set was larger/smaller? How would these results change if the proportion of training data was larger/smaller?

A: Computing model risk from the same dataset yields worse results. We can see that the mean of differences is bigger than in previous experiment. Since we are dealing with small test set, we can see that the standard error is big too. If we were to increase our dataset, we would also decrease the standard error. If we were to increase the proportion of training data, risk estimates will be closer to the true risk but the variance would also be bigger, since we decreased the proportion of testing set.

Cross-validation

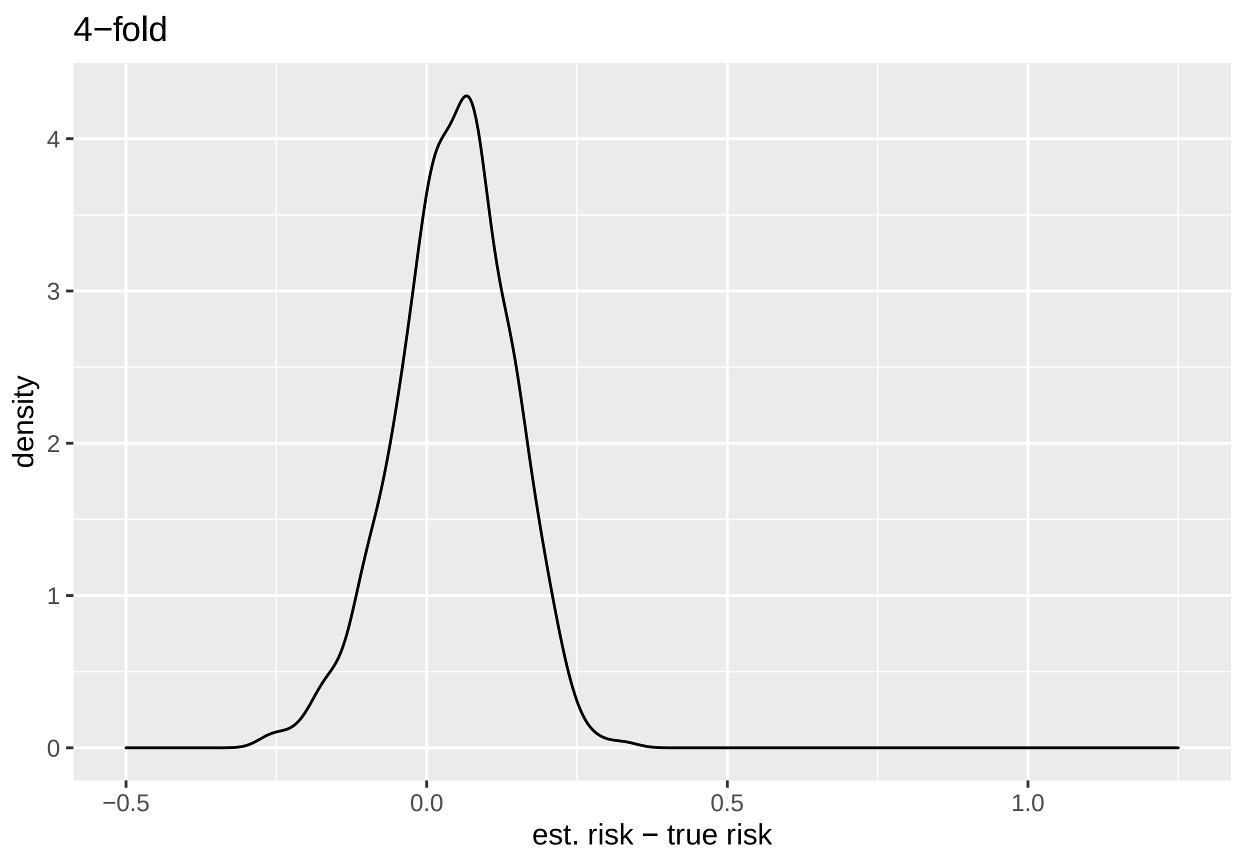
If we extrapolate the results so far to cross-validation, we can conclude that cross-validation estimates of true risk will also be biased and will contain a lot of variability if the dataset is relatively small. This variability will be both due to training set and due to test set variability on top of the inherent variability of the losses. Let's explore this by also finally incorporating the variability caused by the draw of the dataset from the DGP, which we have up to this point kept fixed: 1. Generate a toy dataset with 100 observations. 2. Train model h0 using some learner on all 100 observations and compute its true risk proxy using the huge dataset. 3. Estimate h0's risk with these 5 estimators: 2-fold cross-validation (equivalent to a 50-50 train-test split evaluated both ways; similar to our previous experiment), leave-one-out cross-validation, 10-fold cross-validation, 4-fold cross-calidation, and repeated 20 times 10-fold cross-validation (= repeat 10-fold cross-validation for 20 different partitions and let each observation's loss be the average of these 10). 4. For each of the 5 estimators compute the average difference between the estimate and the true risk (bias) and record if the 95% CI of the estimate contains the true risk of h0. 5. Repeat (1-5) 500 times. Note: It might take a while. 6. For each of the 5 estimators: Plot a density estimate of the differences between estimates and the true risk proxy. Compute the true risk proxy. Compute the median standard error of the estimates. Compute the % of times that the 95%CI contained the true risk.

```r
k_fold_split <- function(data, k, seed = NULL) {
  set.seed(seed)
  split <- sample(rep(1:k, nrow(data)/k))
}

k_fold_validation <- function(data, k, split){
  losses <- c()
  for(j in 1:k){
    train <- data[split != j,]
    test <- data[split == j,]

    h <- glm(y ~ ., data=train, family="binomial")
    y_pred <- predict(h, test[,-9], type="response")

    losses <- c(losses, log_loss(test$y, y_pred))
  }
  return (losses)
}

k_fold <- function(k, repetition=1, title=""){
  df_dgp <- toy_data(100000, 0)

  standard_errors <- c()
  in95CI <- c()
  differences <- c()

  for(i in 1:500){
    toy_dataset <- toy_data(100, i)

    h0 <- glm(y ~ ., data=toy_dataset, family="binomial")
    y_pred <- predict(h0, df_dgp[,-9], type="response")
    true_risk_proxy <- mean(log_loss(df_dgp$y, y_pred))


    losses <- rep(0, 100)

    for(j in 1:repetition){
      loss <- k_fold_validation(toy_dataset, k, k_fold_split(toy_dataset, k, j))
      losses <- losses + loss[sort(names(loss))]
    }
    losses <- losses/repetition
    risk <- mean(losses)
    se <- sd(losses)/sqrt(length(losses))

    in95CI <- c(in95CI, risk + 1.96 * se > true_risk_proxy && risk - 1.96 * se < true_risk_proxy)
    differences <- c(differences, risk - true_risk_proxy)
    standard_errors <- c(standard_errors, se)
  }

  print(ggplot(data.frame(differences), aes(x=differences)) +
    ggtitle(title) +
    geom_density() +
    xlab("est. risk - true risk") +
    ylab("density") +
    xlim(-0.5, 1.25))


  cat("True risk proxy:", round(true_risk_proxy, 4), "\n")
  cat("Mean difference:", round(mean(differences),4), "\n")
  cat("Median standard error:", round(median(standard_errors),4), "\n")
  cat("Percentage of 95CI that contain the true risk proxy:", format(round(mean(in95CI)*100, 1), nsmall
}

k_fold(2, title="2-fold")
```



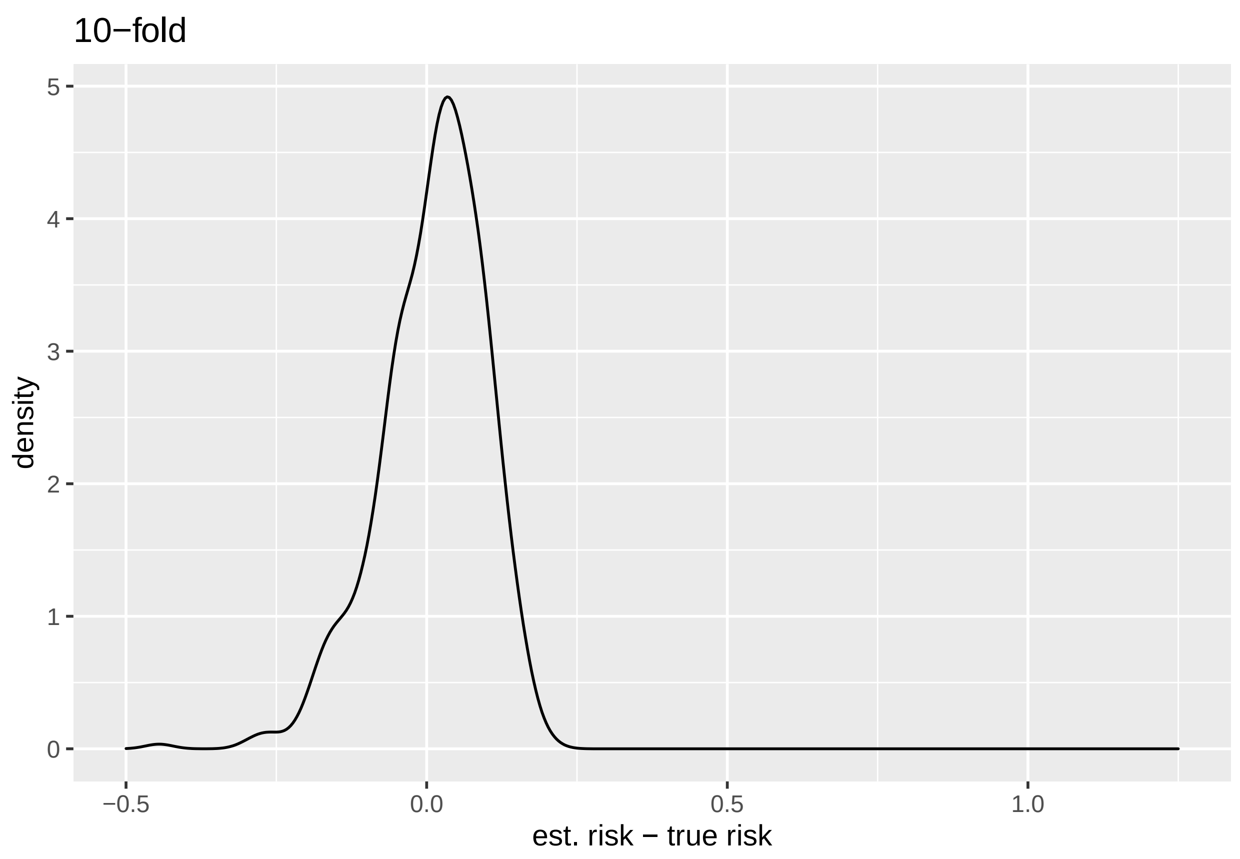
```
## True risk proxy: 0.486
## Mean difference: 0.4897
## Median standard error: 0.1093
## Percentage of 95CI that contain the true risk proxy: 66.6
```

```
k_fold(4, title="4-fold")
```



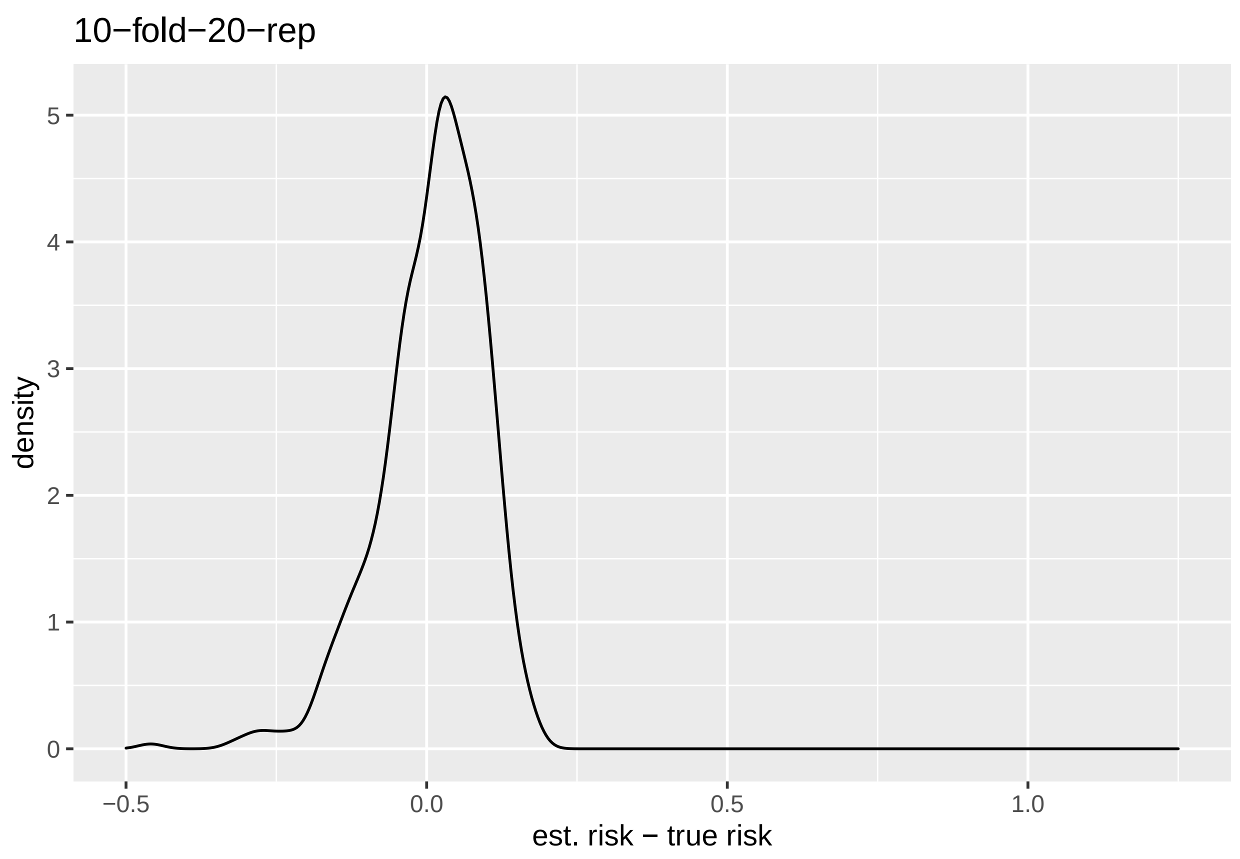
```
## True risk proxy: 0.486
## Mean difference: 0.0429
## Median standard error: 0.084
## Percentage of 95CI that contain the true risk proxy: 90.2
```

```
k_fold(10, title="10-fold")
```



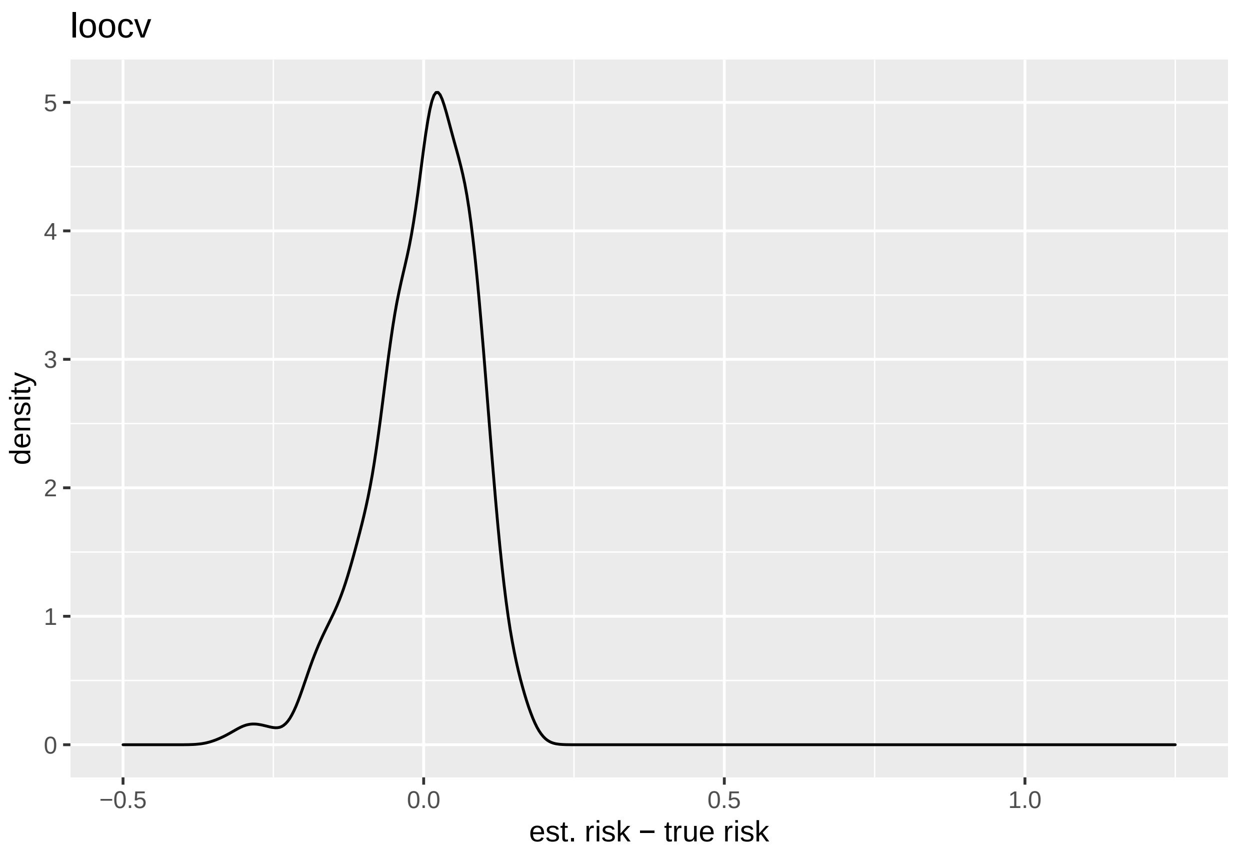
```
## True risk proxy: 0.486
## Mean difference: 0.0102
## Median standard error: 0.0775
## Percentage of 95CI that contain the true risk proxy: 92.0
```

```
k_fold(10, repetition = 20, title = "10-fold-20-rep")
```



```
## True risk proxy: 0.486
## Mean difference: 0.0093
## Median standard error: 0.0765
## Percentage of 95CI that contain the true risk proxy: 93.2
```

```
k_fold(100, title="loocv")
```



```
## True risk proxy: 0.486
## Mean difference: -0.0013
## Median standard error: 0.0747
## Percentage of 95CI that contain the true risk proxy: 92.0
```

What do we see and what are the implications for practical use?

A: If we compare results with previous experiments, we observe lower bias and variance. We see that this is the best method to use if we do not know the data generating process. We see that with bigger folds, we obtain lower mean difference of risks. We also observe that repeating 10-fold cross-validation does not yield much different results.

A different scenario

Can you by choosing a different learner, DGP, dataset size, or something else, produce an example where the results disagree with our last experiment?

If our DGP would be creating instances of random integers, where each toy_dataset would be very different from one another, I think that results would be very different.