Machine Learning for Data Science: Homework #5 Loss Estimation

Vid Stropnik

5/3/2021

Setup

First, let's define the generator for our data, as shown in the Homework instructions. We can note that features *x.6* - *x.8* have no effect on our target variable ***y***. Let's also define the *log_loss* function, as seen in the instructions.

```
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))
}
```

Finally, let us also generate a proxy of all true data, which we will later use to compute the proxy of true risk.

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

General Idea

Throughout this homework, we will investigate how different empirical estimates of the model risk differ from the true risk, as represented by the true risk proxy. To do so, we will use a Logistic Regression model, fit on data points from the same data generating process as the huge dataset, created above.

For logistic regression, we use a **binomial** family *generalized linear model (glm)* in R. We estimate the risk using the expected value of loss between the model's prediction and the ground truth.

To generate our True Risk proxy, we compute the expected loss of the predictions on the huge dataset above.

The details of different empirical model risks are given throughout the following sections.

To avoid redundancy, assume the following code to mean the estimation of a model's risk (in this case, true risk using the ***df_dgp*** dataset) for the remainder of this report:

```
set.seed(0)
t0 <- toy_data(50)
model<- glm( y ~ ., data = t0, family = binomial)
prediction <- predict(model, newdata=df_dgp, type= "response")
gt <- df_dgp$y
trp <- mean(log_loss(gt, prediction))
sprintf("True risk proxy of h; R(h) = %.4f", trp)
```

```
## [1] "True risk proxy of h; R(h) = 0.6971"
```

Finally, let's introduce some more principles that repeat often in this report. When computing the estimated risk, the standard error of the estimation is achieved as follows:

```
se <- function(x) {
  return (sd(x) / sqrt(length(x)))
}
std_error <- se(log_loss(gt, prediction))
sprintf("Standard error of True risk proxy is: %.4f", std_error)
```

```
## [1] "Standard error of True risk proxy is: 0.0037"
```

It is worth noting that the true risk's standard error is of the order *1e-3*. **This is is also how we know that reporting our results with three decimal places is sufficient to generalize to the true risk.**

Finally, by the 68-95-99.7 rule of statistics, and the central limit theorem, we know that under the assumption of independent and identically distributed variables (which we have here), a 95% confidence interval is equivalent to 1.96 standard errors in both directions from the mean of the distribution. Consequently, the following function computes if a given value is inside the 95 percent confidence interval of a distribution:

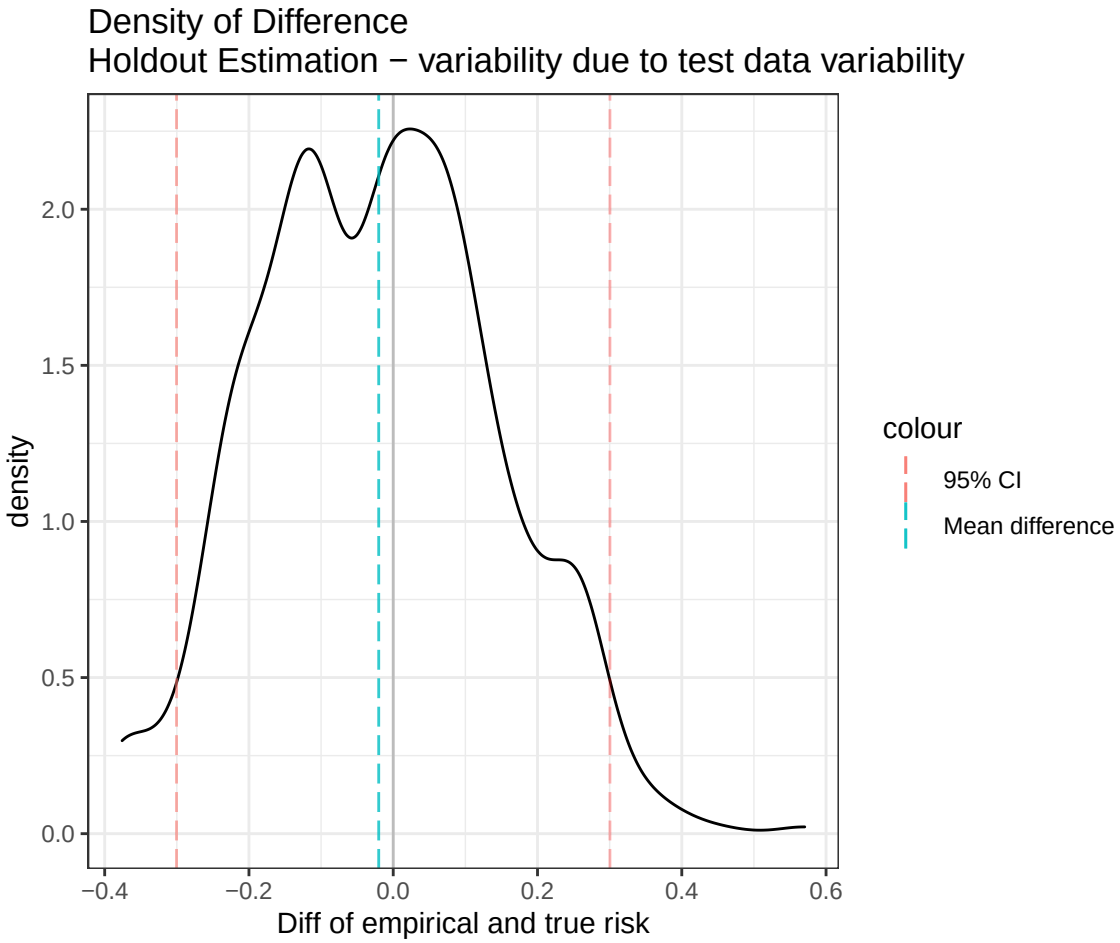
```
inci95 <- function(samps, x) {
  m <- mean(samps)
  err <- se(samps)
  lower <- m - 1.96 * err
  upper <- m + 1.96 * err
  return (lower <= x & x <= upper)
}
```

Throughout this homework, we will be interested exclusively in whether the true risk proxy can be found inside the 95% confidence interval.

Holdout Estimation

We train our model on 50 samples from the same data generating process as we used to estimate our true risk proxy, then assess the empirical risk on an independent group of 50 different samples. Due to these samples being completely withheld from the learning process, we call this method *Holdout estimation*.

To get some more reliable results, we run this example 1000 times. The following plot shows the differences between the empirical estimation of risk and the true risk proxy. Some key information is reported below;




```
## [1] "True risk proxy: 0.697"
```

```
## [1] "Mean difference between Empirical and True Risk (bias): -0.020"
```

```
## [1] "True risk of 0.5-0.5 predictions: 0.693"
```

```
## [1] "Median standard error: 0.153"
```

```
## [1] "Frequency of True Risk in 95% CI: 91.1%"
```

We can notice that the bias of our model is distributed around 0, with a slight skewedness to the left. We see that the mean empirical risk is very close to the true risk (the mean difference is of the same size order as the standard error of the true risk)! We conclude that Holdout estimation is unbiased. Finally, we can see that the risk of the 50-50 estimator falls outside our the naively constructed 68% confidence interval that can be achieved by observing a one standard error deviation to each side of the estimated bias. This establishes that our model outperforms a naive classifier most of the time.

Finally, we observe that the true risk lies in the 95% Confidence interval slightly less often than expected. We can assume that this is due to our CI construction process assuming a normal distribution and thus being slightly naive in the context of having a relatively small amount of available data to learn from.

All in all, we can state that for practical use, holdout estimation is a really good and stable method for model risk estimation and should be used when applicable.

Effect of changing parameters Should we increase the size of the *training set*, the value of all computed risks (incl. the true risk proxy) would decrease due to a better understanding of the underlying data generating process. Intuitively, this should produce a smoother distribution closer to the bell curve. With this, of course,

come the shrinking of the confidence interval and bias. Conversely, should the size of the training set decrease, the resulting risks and standard errors would increase.

Increasing the *test set* would not effect the true risk proxy; however, our estimator would generally be more robust towards random noise in the test data, which should again, smooth out the distribution plot.

Should we reduce the size of the test set, the produced estimator might overestimate the model's performance, as cases where the learner might fail are less likely to be detected.

Overestimation of the Deployed Model's risk

In this section, we demonstrate why it is important to always deploy a model using all available data for learning. We generate two toy datasets of equal size. We can assume that one was used for training and the other for testing in the Holdout estimation step, as described in the preceding section. Here, we demonstrate the difference in true risk between one model, trained only on the **training** data in said scenario, while the other is fit on both.

For better understanding, consider the following code:

```
risk_only_train <- c()
risk_full <- c()
for (s in c(1:50)){
  train <- toy_data(50,)
  test <- toy_data(50,)
  t_both <- rbind(train, test)
  h_only_train <- glm( y ~ ., data = train, family = binomial)
  h_both <- glm( y ~ ., data = t_both, family = binomial)
  pred_train <- predict(h_only_train, newdata=df_dgp, type="response")
  pred_both <- predict(h_both, newdata=df_dgp, type="response")
  trp_only_train <- mean(log_loss(df_dgp$y, pred_train))
  trp_both <- mean(log_loss(df_dgp$y, pred_both))
  risk_only_train <- c(risk_only_train, trp_only_train)
  risk_full <- c(risk_full, trp_both)
}
summary(risk_only_train - risk_full)
```

```
##     Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.00000 0.00000 0.00000 0.06457 0.00000 0.70953
```

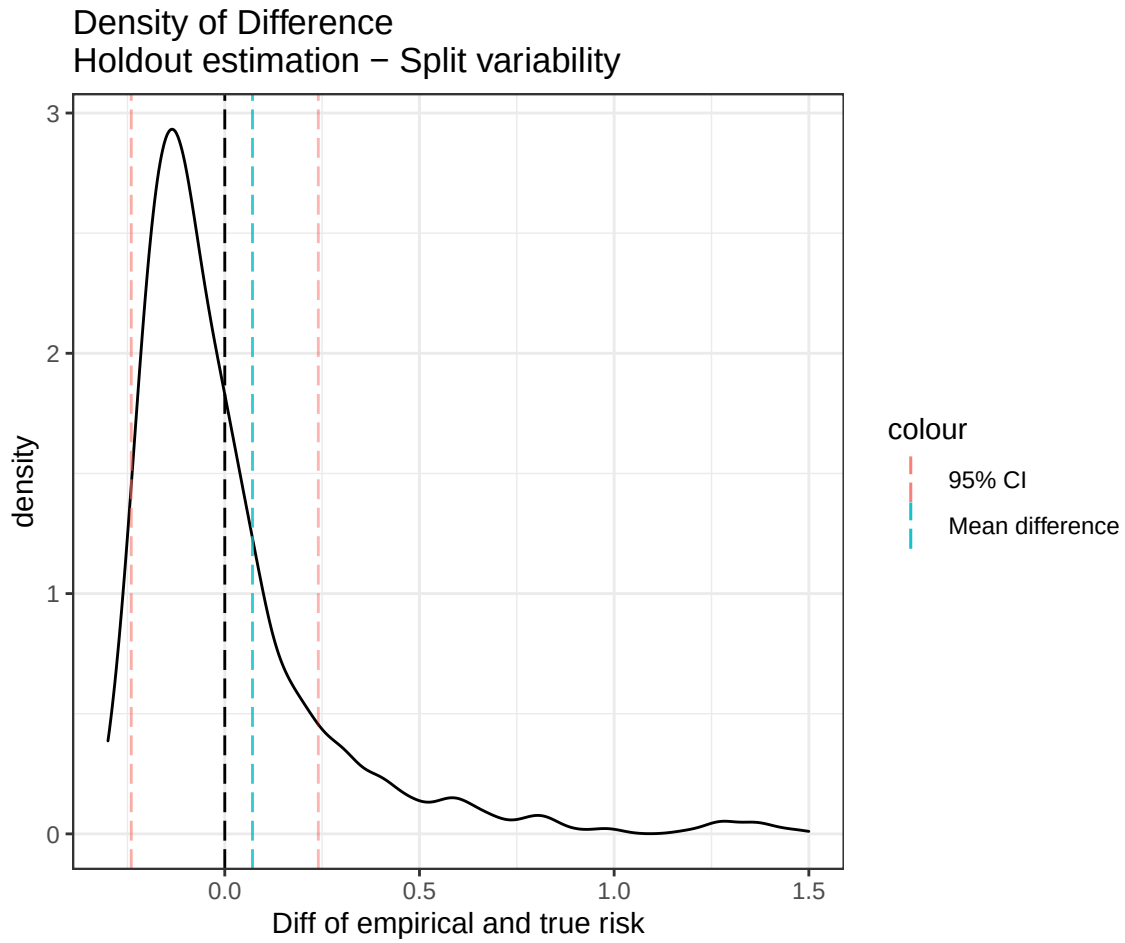
Here, we show that, in expectation, the risk of a model trained on less data is greater than that of the one, trained on more. However, we can see that our expectation estimator (the mean) is skewed by the extreme values, as the two estimators converge in a majority of test runs. This means that, by relying on expectation, there is always a probabilistic chance that we might be overestimating our model's risk due to random noise in our training data. Thus, the bigger the amount of training data (and our model's understanding of the underlying DGP), the better. Consequently, after estimating our model's risk, **all** data should be used in deployment.

Effect of changing parameters By incresing the amount of samples, the mean difference between the models should get progressively smaller, while reducing it should greatly reduce consistency, vastly incresing the absolute values of the min and max samples differences. We would expect that the median value would remain close to 0 in both cases, given that the underlying DGP remains the same.

Loss estimator variability due to split variability

In this section, we explore how split variability might affect our estimator's performance. Given only 100 samples from the underlying DGP, we split our data into two equally sized partitions (for training and

evaluation). Below, we show the distribtion of differences between the empirically estimated risk and the true risk over 1000 different splits;




```
## [1] "True risk proxy of h0; R(h) = 0.697"
```

```
## [1] "Average diff between empirical and true risk (bias): 0.071"
```

```
## [1] "Median standard error: 0.123"
```

```
## [1] "Frequency of True Risk in 95% CI: 76.2%"
```

We've already established that the true risk is overestimated when omitting a certain amount of available datapoints from the learning process. Here, however, we can see that a random split might really heavily influence the amount by which we overestimate our risk. Notice that in approximately one out of ten cases, the true risk isn't even in the median confidence interval.

From this, we can conclude that several passes of the train-test split should be performed to get more stable results, such as the approaches shown in the Cross Validation section that follows.

Effect of changing parameters Even though we're currently over-estimating the model's risk, we can see that the mode of the shown distribution is to the left of 0. Hence, should we increase the dataset and remain with the same split proportion, we can expect this sort of estimation to actually slightly underestimate the model's risk, as the proportion of extreme values gets smaller with the larger dataset.

Conversely, these inherent overestimations will represent a larger proportion in a smaller set, thus resulting in the overestimation of risk by the empiric model.

By incresing the proportion of training samples would allow our model to gain more understanding of the underlying data generating process, thus lowering the bias. This is why, in practice, we often use something more similar to a 70-30 or a 80-20 split.

Cross Validation

Finally, let's see how all of our results so far fare in the context of k-fold cross validation.

A model is trained on 100 samples of the now very well known data generating process. Below, we show the results for 5 different cross-validation scenarios, all of which are carried out on new, shuffled data samples over 500 iterations:

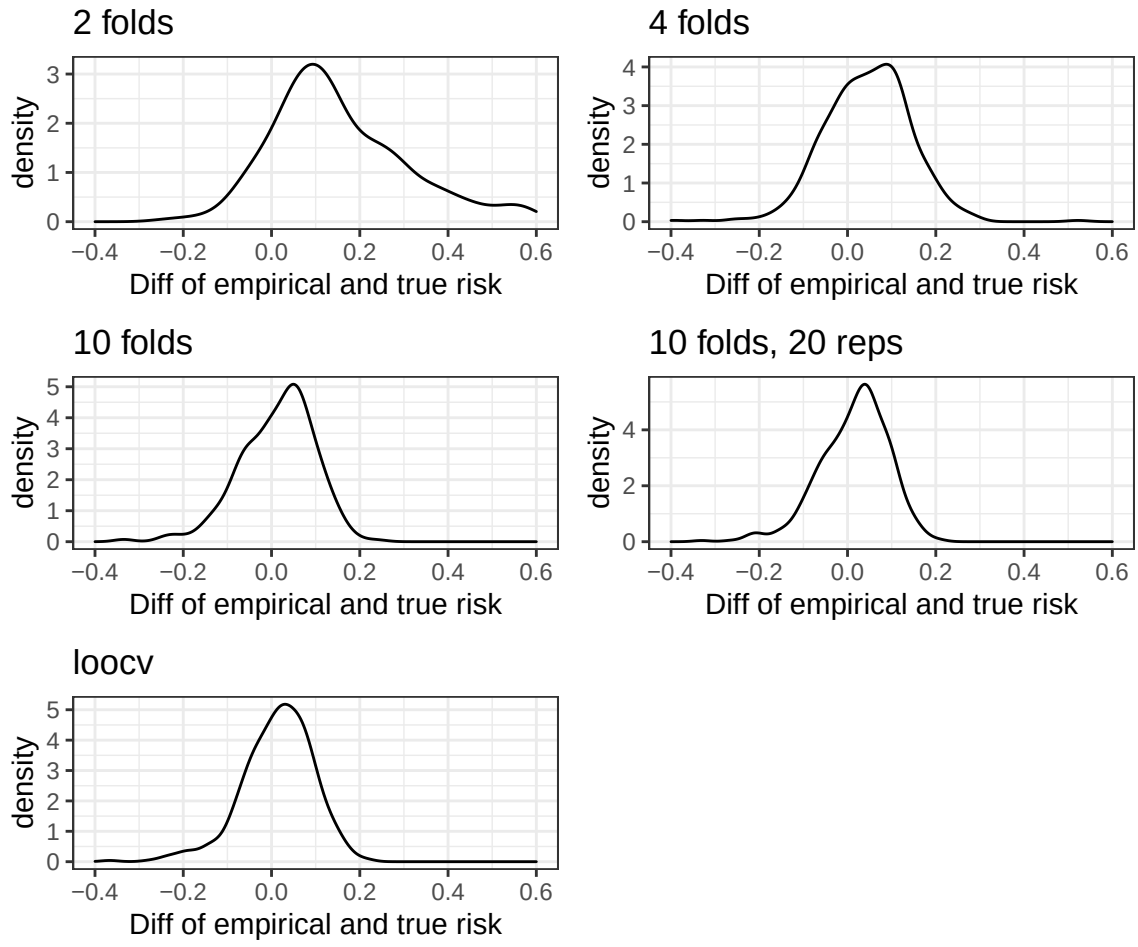
-2-fold cross validation without repetitions,

-4-fold cross validation without repetitions,

-10-fold cross validation without repetitions,

-10-fold cross validation with 20 repetitions,

-Leave-one-out cross validation (LOOCV) without repetitions.




```
## [1] "Mean true risk proxy: 0.515"
```

```
## [1] "***MEAN DIFFERENCES - BIAS***"
```

```
## [1] "2-fold: 0.386"
```

```
## [1] "4-fold: 0.046"
## [1] "10-fold: 0.011"
## [1] "10-fold, 20 reps: 0.013"
## [1] "LOOCV: 0.012"
## [1] "***MEDIAN STANDARD ERRORS***"
## [1] "2-fold: 0.110"
## [1] "4-fold: 0.084"
## [1] "10-fold: 0.077"
## [1] "10-fold, 20 reps: 0.077"
## [1] "LOOCV: 0.077"
## [1] "***Proportion of time in 95CI***"
## [1] "2-fold: 70.0%"
## [1] "4-fold: 92.2%"
## [1] "10-fold: 93.2%"
## [1] "10-fold, 20 reps: 92.8%"
## [1] "LOOCV: 92.6%"
```

In this example, we can clearly observe the Law of Large numbers at work, as the differences are getting closer to their expected value (0) as a larger number of trials is performed. From this, we can conclude that a larger number of folds is preferable, if computationally applicable and sensible for the problem at hand.

As we've already established, Cross validation over-estimates the model performance (under-estimates it's risk). We can see that, as the number of folds increases, this over-estimation gets gradually less extreme, partially due to the conclusion above, as well as a better understanding of the underlying DGP when fitting the model (more training samples in each pass).

It makes sense, then, that LOOCV performs well in this regard, as it's mode most closely coincides with the true risk proxy (diff=0). Thus, should we want a reliable estimate of our model's risk, LOOCV should always be considered first.

When we compare the 10-fold cross validation without repetitions, and the one with 20, we can see that the performance of the two estimators is quite similar. It is notable, though, that the results of the 20-repetitions model converge towards those of the Leave-one-out cross validation. We conclude that several passes of a high k-fold cross validation technique is equivalent to LOOCV in performance, in cases where said technique might not be applicable (ie. ordinal data).

Different Scenario

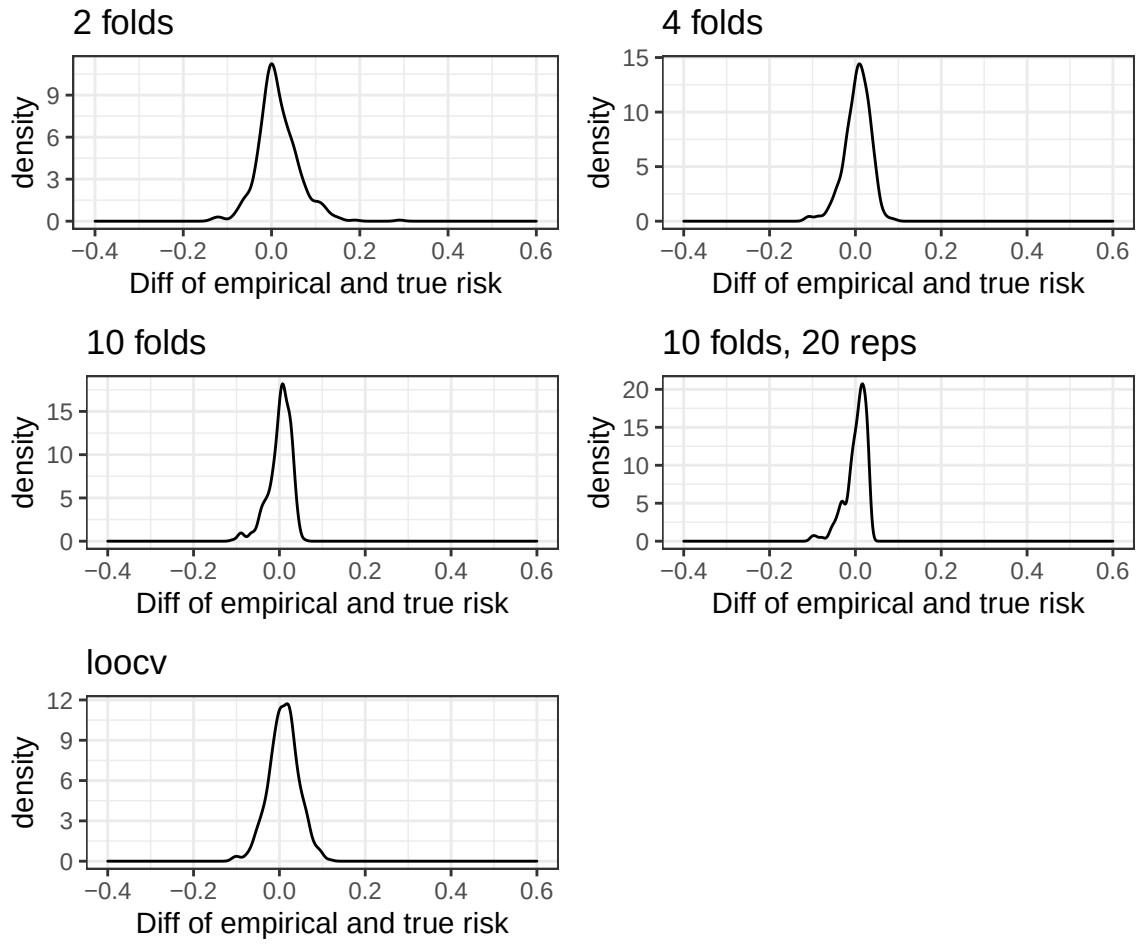
For this last task, we try to find some change to our data generating process, learner or dataset size that disagrees with the findings about CV, shown above.

Let's consider the following data generating process and perform the same experiment as above on it:

```
different_dgp <- function(n, seed = NULL) {
 set.seed(seed)
  t
 x <- matrix(floor(runif(2 * n, 1, 10)), ncol = 2)
 z <- (-1)^ x[,1]
 y <- runif(n) > 1 / (1 + exp(-z))
```

```
 return (data.frame(x = x, y = y))
}

df_dgp_fake <- different_dgp(100000)
```




```
## [1] "Mean true risk proxy: 0.515"
## [1] "***MEAN DIFFERENCES - BIAS***"
## [1] "2-fold: 0.015"
## [1] "4-fold: 0.004"
## [1] "10-fold: 0.001"
## [1] "10-fold, 20 reps: 0.001"
## [1] "LOOCV: 0.010"
## [1] "***MEDIAN STANDARD ERRORS***"
## [1] "2-fold: 0.025"
## [1] "4-fold: 0.021"
## [1] "10-fold: 0.019"
## [1] "10-fold, 20 reps: 0.019"
```

```
## [1] "LOOCV: 0.022"
## [1] "***Proportion of time in 95CI***"
## [1] "2-fold: 78.2%"
## [1] "4-fold: 80.6%"
## [1] "10-fold: 79.4%"
## [1] "10-fold, 20 reps: 80.2%"
## [1] "LOOCV: 78.0%"
```

We notice that, while LOOCV was the densest method around the expected value before , most k-fold cross validation approaches have a smaller variance than it for the considered novel dataset. This is probably due to different training sets in LOOCV having more overlap (due to X.1, X.2 being discrete), consequently resulting in more inter-dependent estimates that might produce the same result.

Conclusion

In this homework, we conducted several practical experiments that gave us useful, first hand insight into the actual meaning behind the numbers we often use when evaluating our machine learning models. We learned when methods might over or underestimate the true risk present in our data and reaffirmed some truths that a data scientist should always be aware of when deploying and evaluating a model. The presented empirical results coincide with the literature and the expected results from the homework instructions, while the final task allowed me to think about the properties of Leave-One-Out cross validation on my own, again coming to sensible conclusions that further elevated my level of understanding the subject matter at hand.