Solution 1:

- The inner loss is the loss that is optimized directly by the machine learning model. The outer loss is the loss (or performance measurement) used to evaluate the model.
- Which model is more likely to overfit the training data:
  - k-NN with 1 or with 10 neighbors? **1 neighbor**, because it's an exact memorization of training data.
  - Logistic regression with 10 or 20 features? **20 features**, because the more features, the more coefficients the learner estimates. More coefficients mean more degrees of freedom, which make overfitting more likely.
  - LDA or QDA? **QDA**, because it has more parameters to possibly overfit the data. LDA is more likely to underfit more complex relationships.
- Which of the following methods yield an unbiased generalization error estimate? Performance estimation ...
  - on training data: **Biased, too optimistic**
  - on test data: **Unbiased / Biased, too pessimistic** (Test data is *not included* / *included* in the final model)
  - on training and test data combined: **Biased, too optimistic** (But a little bit less than only using training data).
  - using cross validation: **Biased, too pessimistic** (The higher the ratio of folds / number of observation, the smaller the pessimistic bias)
  - using subsampling: **Biased, too pessimistic** (The smaller the subsampling rate, the larger the pessimistic bias)
- Resampling strategies solve the problem that comes from the randomness of the training and test data split: Error estimation using a single split has a high variance. Resampling estimates are more robust because they average over different splits.
- Nested resampling solves the problem of simultaneously conducting tuning/model selection and performance estimation. When we use the performance estimates from the same data that were used for model selection (as done in simple, not-nested resampling), the final error estimate is too optimistic.

Solution 2:

- The training performance is too optimistic (mmce of 0), because the mmce is higher on new data.
- The test performance is unbiased (if the final model is only trained on the training data), but it depends on the split, as can be seen in the CV folds: Each CV fold represents a training test split and the mmce measure varies between folds.
- The CV estimate averages over the different splits and gives an slightly pessimistic, more robust estimate.
- The CV estimate is preferable over the other two, but more computationally expensive.

Solution 3:

```
## Error in keep_in_bounds(rows, 1L, nrow(private$.data)): Assertion on 'x' failed: Must be
of type 'integer', not 'double'.
## Error: Cannot predict, Learner 'classif.lda' has not been trained yet
## Error in eval(expr, envir, enclos): object 'pred' not found
```

a) Each loss function we have learned so far to fit the model (inner loss) can also be used as performance measure (outer loss).

For classification:

- 0-1 loss (= mean misclassification error),
- Logistic loss (bernoulli loss), ...

For regression:

- L_2-loss (= mean squared error),
- L_1-loss (= mean absolute error), ...

To get a list of all measures you can use `mlr_measures`.

b)
```r
# look at the task
task <- tsk("boston_housing")
task

## <TaskRegr:boston_housing> (506 x 19)
## * Target: medv
## * Properties: -
## * Features (18):
##   - dbl (13): age, b, cmedv, crim, dis, indus, lat, lon, lstat,
##     nox, ptratio, rm, zn
##   - int (3): rad, tax, tract
##   - fct (2): chas, town

n <- task$nrow

# select index vectors to subset the data randomly
set.seed(123)
train_ind <- sample(seq_len(n), 0.5*n)
test_ind <- setdiff(seq_len(n), train_ind)

# specify learner
learner <- lrn("regr.kknn", k = 3)

# train model to the training set
learner$train(task, row_ids = train_ind)

# predict on the test set
pred <- learner$predict(task, row_ids = test_ind)
pred

## <PredictionRegr> for 253 observations:
##     row_id truth response
##          1  24.0 23.22445
##          2  21.6 19.98830
```

```
##          3  34.7 34.97419
## ---
##        504  23.9 22.22775
##        505  22.0 21.76531
##        506  11.9 20.88958
```

c)
```r
# predict on the train set
pred_train <- learner$predict(task, row_ids = train_ind)
pred_train$score(list(msr("regr.mse"), msr("regr.mae")))

##  regr.mse  regr.mae
## 1.2322560 0.7564092

# predict on the test set
pred_test <- learner$predict(task, row_ids = test_ind)
pred_test$score(list(msr("regr.mse"), msr("regr.mae")))

##  regr.mse  regr.mae
## 12.424958  2.596332
```

Unsurprisingly the model performs better on the training data (smaller loss) then on the test data.

d)
```r
# select different index vectors to subset the data randomly
set.seed(321)
train_ind <- sample(seq_len(n), 0.5*n)
test_ind <- setdiff(seq_len(n), train_ind)

# specify learner
learner <- lrn("regr.kknn", k = 3)

# train model to the training set
learner$train(task, row_ids = train_ind)

# predict on the test set
pred_test <- learner$predict(task, row_ids = test_ind)
pred_test

## <PredictionRegr> for 253 observations:
##     row_id truth response
##          2  21.6 29.45474
##          5  36.2 33.14900
##          6  28.7 32.95574
## ---
##        501  16.8 19.61312
##        505  22.0 23.29286
##        506  11.9 21.28301

pred_test$score(list(msr("regr.mse"), msr("regr.mae")))

##  regr.mse  regr.mae
## 12.507468  2.458798
```

Effect: We will predict different observations since the test set is different. The same observations get a slightly different prediction (e.g. observation with id 2). This affects the final error estimation.

e)
```
rdesc <- rsmp("cv", folds = 10)
r <- resample(task, learner, rdesc)

## INFO  [08:24:22.069] Applying learner 'regr.kknn' on task 'boston_housing' (iter 1/10)
## INFO  [08:24:22.107] Applying learner 'regr.kknn' on task 'boston_housing' (iter 2/10)
## INFO  [08:24:22.134] Applying learner 'regr.kknn' on task 'boston_housing' (iter 3/10)
## INFO  [08:24:22.152] Applying learner 'regr.kknn' on task 'boston_housing' (iter 4/10)
## INFO  [08:24:22.169] Applying learner 'regr.kknn' on task 'boston_housing' (iter 5/10)
## INFO  [08:24:22.188] Applying learner 'regr.kknn' on task 'boston_housing' (iter 6/10)
## INFO  [08:24:22.209] Applying learner 'regr.kknn' on task 'boston_housing' (iter 7/10)
## INFO  [08:24:22.232] Applying learner 'regr.kknn' on task 'boston_housing' (iter 8/10)
## INFO  [08:24:22.249] Applying learner 'regr.kknn' on task 'boston_housing' (iter 9/10)
## INFO  [08:24:22.268] Applying learner 'regr.kknn' on task 'boston_housing' (iter 10/10)

r$aggregate(list(msr("regr.mse"), msr("regr.mae")))

##  regr.mse  regr.mae
## 10.045363  2.229458
```

Solution 4:

a) First, sort the table:

ID	Actual Class	Score	Predicted Class
6	0	0.63	1
7	1	0.62	1
10	0	0.57	1
4	1	0.38	0
1	0	0.33	0
8	1	0.33	0
2	0	0.27	0
5	1	0.17	0
9	0	0.15	0
3	1	0.11	0

	Actual Class - 0	Actual Class - 1
Prediction - 0	3	4
Prediction - 1	2	1

so we get

FN	FP	TN	TP
4	2	3	1

b)

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} = \frac{1}{3}$$

$$\text{Sensitivity} = \frac{\text{TP}}{\text{TP} + \text{FN}} = \frac{1}{5}$$

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} = \frac{4}{10}$$

$$\text{Specificity} = \frac{\text{TN}}{\text{TN} + \text{FP}} = \frac{3}{5}$$

$$\text{Error Rate} = \frac{\text{FP} + \text{FN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} = \frac{6}{10}$$

$$\text{F-measure} = \frac{2 \cdot \text{Precision} \cdot \text{Sensitivity}}{\text{Precision} + \text{Sensitivity}} = 0.25$$

$$\text{Negative Predictive Value} = \frac{\text{TN}}{\text{TN} + \text{FN}} = \frac{3}{7}$$

c) First we sort the results by the score:

```
## Error in kable(cdata):  could not find function "kable"
```

Here we see that $\frac{1}{n_+} = \frac{1}{5} = 0.2$ and $\frac{1}{n_-} = \frac{1}{5} = 0.2$. Now we follow the algorithm as described in the lecture slides:

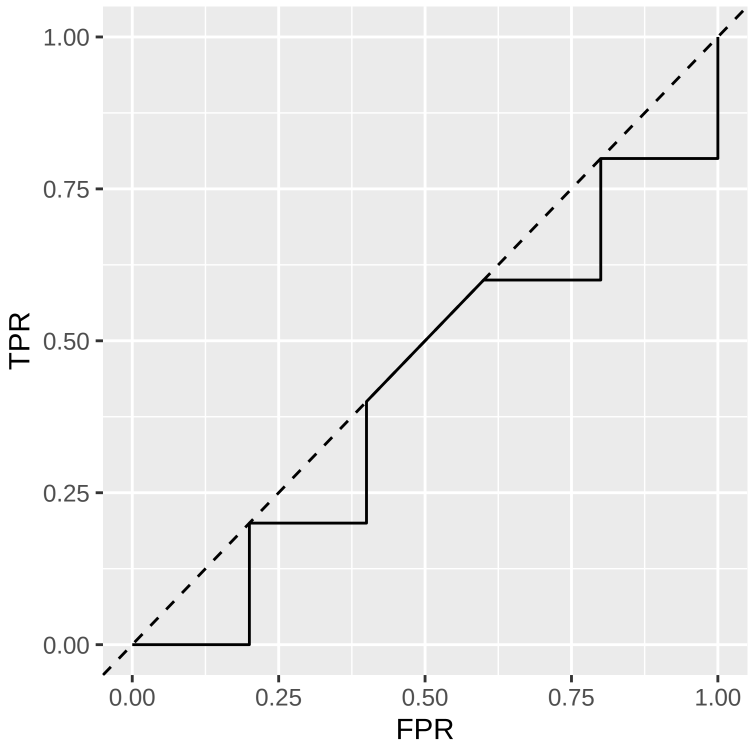
(i) Set $\alpha = 1$, so we start in $(0, 0)$; we predict everything as 1.
(ii) Set threshold $\tau = 0.625$ yields TPR 0 and FPR $0 + \frac{1}{n_-} = 0.2$. (Obs. 6 is "0")
(iii) Set threshold $\tau = 0.6$ yields TPR $0 + \frac{1}{n_+} = 0.2$ and FPR 0.2. (Obs. 7 is "1")
(iv) Set threshold $\tau = 0.5$ yields TPR 0.2 and FPR $0.2 + \frac{1}{n_-} = 0.4$. (Obs. 10 is "0")
(v) Set threshold $\tau = 0.35$ yields TPR $0.2 + \frac{1}{n_+} = 0.4$ and FPR 0.4. (Obs. 4 is "1")
(vi) Set threshold $\tau = 0.3$ yields TPR $0.4 + \frac{1}{n_+} = 0.6$ and FPR $0.4 + \frac{1}{n_-} = 0.6$. (Obs. 1/8 is "0"/"1")
(vii) Set threshold $\tau = 0.2$ yields TPR 0.6 and FPR $0.6 + \frac{1}{n_-} = 0.8$. (Obs. 2 is "0")
(viii) Set threshold $\tau = 0.16$ yields TPR $0.6 + \frac{1}{n_+} = 0.8$ and FPR 0.8. (Obs. 5 is "1")
(ix) Set threshold $\tau = 0.14$ yields TPR 0.8 and FPR $0.8 + \frac{1}{n_-} = 1$. (Obs. 9 is "0")
(x) Set threshold $\tau = 0.09$ yields TPR $0.8 + \frac{1}{n_+} = 1$ and FPR 1. (Obs. 3 is "1")

Therefore we get the polygonal path consisting of the ordered list of vertices

$$(0, 0), (0, 0.2), (0.2, 0.2), (0.2, 0.4), (0.4, 0.4), (0.6, 0.6), (0.6, 0.8), (0.8, 0.8), (0.8, 1), (1, 1).$$

```
library(ggplot2)
roc_data <- data.frame(TPR = c(0, 0,   0.2, 0.2, 0.4, 0.6, 0.6, 0.8, 0.8, 1),
                       FPR = c(0, 0.2, 0.2, 0.4, 0.4, 0.6, 0.8, 0.8,   1,   1))

ggplot(roc_data, aes(x = FPR, y = TPR)) + geom_line() +
  geom_abline(slope = 1, intercept = 0, linetype = 'dashed')
```



We see that the resulting ROC lies below the line from the origin with a slope of 1, which represents a random classifier, i.e., the scoring algorithm performs worse than a random classifier. If this happens while evaluating the training data, the labels of the scoring algorithm should be inverted.

d) We can compute the AUC (*area under the curve*) by looking at the ROC, s.t.

$$AUC = 0.5 - 4 \cdot (0.2 \cdot 0.2 \cdot 0.5) = 0.42.$$