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:
  - knn 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**
  - on training and test data combined: **Biased, too optimistic** (But a little bit less than only using training data).
  - using cross validation: **Unbiased**
  - using subsampling: **Unbiased**
- 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 (kappa of 1), because the kappa is lower on new data.
- The test performance is unbiased, but it depends on the split, as can be seen in the CV folds: Each CV fold represents a training test split and the kappa measure varies between folds.
- The CV estimate averages over the different splits and gives an unbiased, more robust estimate.
- The CV estimate is preferable over the other two, but more computationally expensive.

Solution 3:

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 `listLearners()`.

b)
```
# look at the task
bh.task

## Supervised task: BostonHousing-example
## Type: regr
## Target: medv
## Observations: 506
## Features:
##    numerics     factors     ordered functionals
##          12           1           0           0
## Missings: FALSE
## Has weights: FALSE
## Has blocking: FALSE
## Has coordinates: FALSE

n = getTaskSize(bh.task)

# 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
lrn = makeLearner("regr.kknn", k = 3)

# train model to the training set
mod = train(lrn, bh.task, subset = train.ind)

# predict on the test set
pred = predict(mod, bh.task, subset = test.ind)
pred

## Prediction: 253 observations
## predict.type: response
## threshold:
## time: 0.01
##   id truth response
## 2  2  21.6    25.27
## 3  3  34.7    28.49
## 5  5  36.2    30.97
## 6  6  28.7    28.17
```

```
## 7  7  22.9    19.37
## 8  8  27.1    18.23
## ... (#rows: 253, #cols: 3)
```

c)
```r
# predict on the test set
pred.test = predict(mod, bh.task, subset = test.ind)
performance(pred.test, measures = list(mlr::mae, mlr::mse))

##    mae    mse
##  2.644 13.732

# predict on the test set
pred.train = predict(mod, bh.task, subset = train.ind)
performance(pred.train, measures = list(mlr::mae, mlr::mse))

##    mae    mse
## 0.9961 2.8029
```

The generalization error estimate is much higher on the training data.

d)
```r
# select different index vectors to subset the data randomly
set.seed(321)
train.ind = sort(sample(seq_len(n), 0.5*n))
test.ind = setdiff(seq_len(n), train.ind)

# specify learner
lrn = makeLearner("regr.rpart")

# train model to the training set
mod = train(lrn, bh.task, subset = train.ind)

# predict on the test set
pred = predict(mod, bh.task, subset = test.ind)
pred

## Prediction: 253 observations
## predict.type: response
## threshold:
## time: 0.00
##    id truth response
## 1   1  24.0    28.33
## 2   2  21.6    21.67
## 7   7  22.9    21.67
## 8   8  27.1    17.36
## 9   9  16.5    17.36
## 10 10  18.9    17.36
## ... (#rows: 253, #cols: 3)

pred.test = predict(mod, bh.task, subset = test.ind)
performance(pred.test, measures = list(mlr::mae, mlr::mse))

##    mae    mse
##  3.343 26.492
```

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 = makeResampleDesc("CV", iters = 10)
r = resample(lrn, bh.task, rdesc, measures = list(mlr::mae, mlr::mse))
```

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)

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

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

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

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

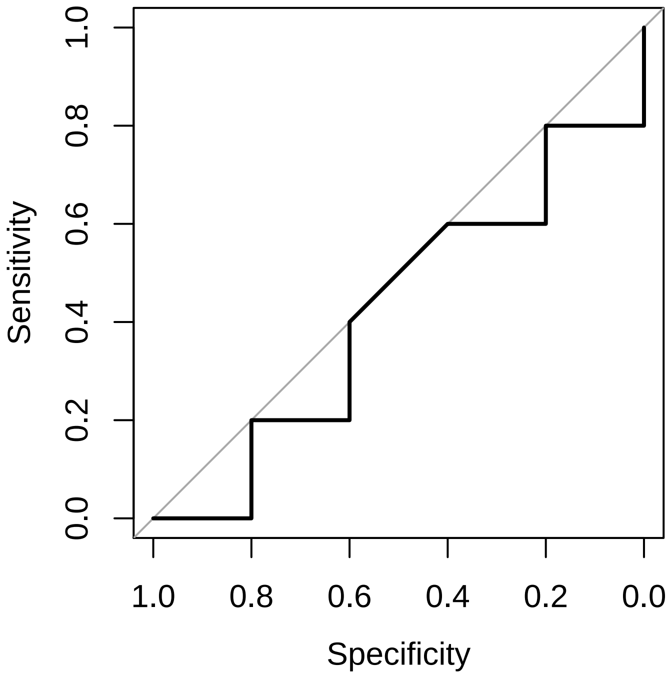
$$\text{Error Rate} = \frac{\text{FP} + \text{FN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} = \frac{6}{10}$$

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

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

c) The ROC plot (slightly different then our approach), drawn by using the package `pROC`:

```r
library(pROC)
cdata = data.frame(
  true_labels = c(0,0,1,1,1,0,1,1,0,0),
  scores = c(0.33,0.27,0.1,0.38,0.17,0.63,0.62,0.33,0.15,0.57)
)
roc_res = roc(true_labels ~ scores, cdata)
plot(roc_res)
```



d) The AUC:

$$AUC = 0.2 \cdot 0.2 + 0.4 \cdot 0.4 + 0.2 \cdot 0.2 + 0.2 \cdot 0.8 = 0.4$$

or by using the plot given from `R`:

$$AUC = 0.2 \cdot 0.2 + 0.4 \cdot 0.4 + 0.2 \cdot 0.2 \cdot 1.5 + 0.2 \cdot 0.8 = 0.42$$

or by using `R`:

```r
roc_res$auc

## Area under the curve: 0.42
```