**Practical worksheet**

**Week 2: Evaluating performance**

*Written by Louise Millard*

Intended learning outcomes:

Be able to

* Implement a machine learning pipeline to train and evaluate the performance of machine learning algorithms and models.
* Evaluate the performance of machine learning algorithms and models for a particular task, and compare performance across them.

In the practical last week you trained a K-nearest neighbour (KNN) model to predict heart disease. You used a single train-test split, and evaluated the model on the test set using the contingency table and accuracy.

Imagine the following scenario:

* You have two algorithms you would like to try, to predict heart disease.
* You want to pick one of these models to deploy for others to use (e.g. imagine having a web site where clinicians could submit a patient’s data, and receive a heart disease risk to then advise the patient).

**Setting up your workspace**

In this practical you will gradually build a script to tackle the scenario above. You should set up a directory where you will work, as you learnt about in the Applied Health Data Science unit. The directory should have three folders: code, data, and results.

Download the file *heart-adapted.csv* from blackboard and put this in the data directory.

You will now write your script(s) in the *code* directory, and store results files in the *results* directory.

Ideally, you will make a git repository (e.g. on GitHub) and commit each logical change to this repository, as you go through this practical. If you are not proficient with Git yet (you should be!) then go through the practical without using Git, so you can focus your learning in this session on machine learning skills. After the session, go through the practical again, this time using Git. You should be able to conduct reproducible machine learning research by the end of this course, so it is really important to be able to combine the skills you learnt in Applied Health Data Science with the machine learning skills you learn in this unit.

**Loading mlr3verse**

We will use the packages in mlr3verse in this practical. Load them with:

library(mlr3verse)

**Loading the data**

We will be working with a larger version of the heart disease dataset, which can be downloaded from blackboard and is called heart-adapted.csv.

Download the data, load it into R and become familiar with it, e.g. the columns and its size.

dat <- read.csv('../data/heart-adapted.csv')

head(dat)

dim(dat)

Convert the heart disease column into a factor:

dat$hd = factor(dat$hd, levels=c("absence", "presence"))

**Partitioning the data**

As described above we want to first evaluate algorithms, and then train and evaluate a specific deployable model. We can do this using the data partition described in the lecture, where we use cross validation to evaluate algorithms, and a hold-out dataset to evaluate a deployable model:

**A blue rectangular box with white text

Description automatically generated**

Divide your data into two parts – one (75%) to use for algorithm evaluation (cross validation) and the other (25%) to use for model evaluation (i.e. the hold-out data).

This should result in two data tables, one call dat.train and one called dat.holdout.

task <- as\_task\_classif(heartdata, target='hd', positive = "presence", id = "heart disease")

parts <-partition(task,ratio = 3/4)

dat.train <- parts$train

dat.holdout <- parts$test

**Evaluating the performance of algorithms**

To practice comparing algorithm performance we will compare the performance of KNN with logistic regression. We want to choose the algorithm that is able to rank examples better, and hence evaluate performance using AUC. The following code uses dat.train to evaluate the KNN algorithm using 10 fold cross validation:

# define the classification task

task = as\_task\_classif(dat.train, target="hd", positive="presence", id="heart disease")

# stratify on class variable

task$set\_col\_roles("hd",c("target","stratum"))

# define the learning algorithm to use. We set scale to TRUE so that each feature is scaled (using the mean/SD of the training set to scale both the training and test set)

knn = lrn("classif.kknn", predict\_type="prob", scale=TRUE)

# use 10 fold cross validation

cvscheme = rsmp("cv", folds=10)

# instantiate the cv scheme – this means we can use the same CV fold split to evaluate both KNN and logistic regression

cvscheme$instantiate(task)

# Run KNN 10 fold cross validation evaluation

knncv = resample(task, knn, cvscheme, store\_models=T)

# calculate average AUC across the 10 folds

meanAUCKnn = knncv$aggregate(msr("classif.auc"))

# plot average ROC curve

pdf("../results/roc-knncv.pdf")

autoplot(knncv, type="roc")

dev.off()

Run this code line by line and check the objects created so you can see what is happening in each of the steps.

Write code to evaluate logistic regression with 10-fold cross validation, with the same folds split as used for KNN. This should create a variable called *lregcv*, that is equivalent to knncv but contains the results for logistic regression rather than KNN.

*Hint: the learner is called "classif.log\_reg"*

logreg <- lrn("classif.log\_reg", predict\_type="prob")

lrcvscheme$instantiate(tasktrain)

lregcy = resample(tasktrain, logreg, cvscheme, store\_models=T)

meanAUCKnn = lregcy$aggregate(msr("classif.auc"))

meanAUCKnn

autoplot(lregcy, type="roc")

* KNN performs slightly better

Check the fold split is the same, using the following code:

foldsKNN = knncv$resampling$instance

foldsLogistic = lregcv$resampling$instance

all.equal(foldsKNN, foldsLogistic)

The ROC curves are generated by calculating the average sensitivity across the 10 folds, at each specificity. The confidence intervals are calculated by calculating the standard deviation of the sensitivity at each specificity, and then assuming these sensitivity values are normally distributed.

Compare the ROC curves of the two algorithms.

In order to do this, we can update the code above, to use a benchmark grid design that succinctly defines the learners we want to use for a given task and resampling strategy, and plots the ROC curves together using autoplot:

# see https://mlr3.mlr-org.com/reference/benchmark\_grid.html

comp.design = benchmark\_grid(

tasks = task,

learners = c(knn, lreg),

resamplings = cvscheme

)

comp = benchmark(comp.design)

pdf("../results/roc-lreg-knn-cv.pdf")

autoplot(comp, type = "roc")

dev.off()

*Does one dominate the other?*

*KNN slightly outperforms log reg.*

*Are there any points where we might be able to conclude that one algorithm performs better than the other (in terms of sensitivity)?*

Yes generally KNN performs better for 0.1-0.6ish region on 1-specificity

We are using AUC as our evaluation metric, and therefore we want to compare the AUC of the KNN and logistic regression algorithms, across the cross validation folds.

The following code creates a data frame with the AUC values across the 10 folds for the two algorithms:

aucFoldsKnn = knncv$score(msr("classif.auc"))

aucFoldsLreg = lregcv$score(msr("classif.auc"))

resultsCompare = cbind(aucFoldsKnn[,.(iteration, classif.auc)], aucFoldsLreg[,.(classif.auc)])

colnames(resultsCompare) = c('iteration', 'auc.knn', 'auc.lreg')

Remember that we have carefully set up the cross validation so that the same instances are included in each of the CV folds. This means we can compare performance for each fold in a paired way. Just by looking at the fold AUC values, describe how they compare? E.g. is the AUC of one algorithm always larger than for the other algorithm?

| **iteration**  <int> | **auc.knn**  <dbl> | **auc.lreg**  <dbl> |
| --- | --- | --- |
| 1 | 0.8523475 | 0.8306275 |
| 2 | 0.9534884 | 0.8771391 |
| 3 | 0.9163685 | 0.9110018 |
| 4 | 0.9762970 | 0.9163685 |
| 5 | 0.9226297 | 0.9320215 |
| 6 | 0.9105546 | 0.8908766 |
| 7 | 0.9394007 | 0.9199463 |
| 8 | 0.8694097 | 0.9118962 |
| 9 | 0.9385063 | 0.9020572 |
| 10 | 0.9237478 | 0.9199463 |

Here KNN is generally larger for all iterations except iteration 8.

Create boxplots to compare fold AUCs for the two algorithms:

pdf("../results/boxplot-knncv.pdf")

autoplot(knncv, measure = msr("classif.auc"), type = "boxplot")

dev.off()

pdf("../results/boxplot-lregcv.pdf")

autoplot(lregcv, measure = msr("classif.auc"), type = "boxplot")

dev.off()

Inspecting visually, there doesn’t look to be much difference in performance (in terms of AUC). We can check this with a paired t-test:

t.test(resultsCompare$auc.knn, y=resultsCompare$auc.lreg, paired=T)

Note, a T-test is not ideal for this due to the fact that the training sets are not independent. You can read up about other possible tests online e.g. –

Raschka, Sebastian. "Model evaluation, model selection, and algorithm selection in machine learning." *arXiv preprint arXiv:1811.12808* (2018).

Choose an algorithm to take forward and justify your reasons here.

The t-test reports a p-value of 0.1078, showing there is not sufficient evidence that either algorithm performed significantly better. Hence we should take the simpler algorithm to implement: KNN.

Hence select KNN.

**Building a model to deploy, and evaluating it**

Write code to train your chosen algorithm on the training data (i.e. the data you used for cross validation) and test your algorithm on the hold-out data. This should result in a variable called *responsesHoldout* containing the predictions on the hold-out data.

*Hint: Refer back to the practical in week 1.*

*task = as\_task\_classif(dat, target="hd", positive="presence", id="heart disease")*

*knn = lrn("classif.kknn", predict\_type="prob")*

*knn$train(task, parts$train)*

*responsesHoldout = lreg$predict(task, parts$test)*

Plot the ROC curve using the following code:

library(pROC)

holdoutPreds = responsesHoldout$prob

rocHoldout <- roc(response=responsesHoldout$truth, predictor=holdoutPreds[,"presence"], levels=c("absence","presence"))

pdf("../results/roc-lreg-holdout.pdf")

rocobj = plot(rocHoldout)

ci.se.obj <- ci.se(rocobj, boot.n=10000, specificities=rocHoldout$specificities)

plot(ci.se.obj, type='shape')

dev.off()

Note, here we are using the pROC package (rather than mlr3) so that we can generate confidence intervals using bootstrapping, with the ci.se function.

Calculate the AUC of this ROC curve:

responsesHoldout$score(msrs("classif.auc"))

Calculate a confidence interval around this AUC using bootstrapping:

boots = ci.auc(rocHoldout, conf.level=0.95, method="bootstrap", boot.n=10000)

Explain how bootstrapped confidence intervals are generated and the interpretation of these confidence intervals. You can look online at the help information for the ci.auc function.

We create 10000 ‘replicas’ of our sample at random where each example is sampled with replacement. We then rank these and observe the 250th and 9750th to find 95% CI.

**Examining classification performance**

Recall that ROC curves show the performance across all thresholds that can be used to classify instances. Now we are going to examine how we can evaluate the classification performance for specific thresholds using this ROC curve.

Plot the point on the ROC curve that corresponds to using a score threshold equal to 0.5 (i.e. where an individual is predicted to have heart disease for scores >= 0.5 and to not have heart disease for scores <0.5). This is the default classification threshold for your model (either logistic regression or KNN).

To do this, we have given you the following code, and you will need to use the rocHoldout object to extract the sensitivity and specificity that corresponds to a score of 0.5.

# set the sensitivity and specificity values in these variables

*threshSens = XXX*

*threshSpec = XXX*

*# plot roc curve and point on it representing classification threshold*

*g = autoplot(responsesHoldout, type="roc")*

*g + geom\_point(*aes(x=1-threshSpec, y=threshSens), color = "blue"*)*

A graph with a blue dot

Description automatically generated

**Is this an effective classifier?**

To evaluate if this classifier is suitable, we need to consider the specific task for which the classifier will be used.

Imagine we want to identify those most at risk of heart disease, to send out a lifestyle intervention where they are sent a smart watch to encourage them to do more exercise. However, there is only a set budget and this limits the number of people that can be given the intervention.

Using the default threshold of the classifier, what proportion of people would you expect to be given the intervention?

*Hint: corresponds to one of the metrics discussed in the previous lecture.*

Use PPR, PPR = 0.4371

Plot the isometric for this PPR. To do this first rework the following equation for PPR, so that it is in terms of TPR, TNR and w=POS/N:

PPR = (TP + FP)/N

A graph with a red line

Description automatically generated

If only 20% of the population can be given the intervention this PPR is too high.

What threshold should we use to ensure our PPR is around 0.2?

Require threshold of 0.9041

What sensitivity and specificity would this have?

Sens = 0.4375

Spec = 0.9943

Plot the point on the ROC curve for this threshold

A graph with a line and a red line

Description automatically generated

Plot the isometric for this PPR and check it intersects the ROC curve where you expect.

A graph with lines and numbers

Description automatically generated

What are the implications of deploying this model for the intended task?

Low sens means that we will miss many cases of the disease.

High spec means very few wrongly diagnosed which will save money.

**Optional: Investigating accuracy and sensitivity to class distribution**

What is the accuracy for this classification threshold?

*Hint: as this is the default threshold for the algorithm (KNN or logistic regression) you can just get the accuracy from the responsesHoldout object, just like we did for AUC.*

Plot the ROC isometric for this accuracy value. Here is the equation for accuracy in terms of class distribution, TPR and TNR given in the lecture:

acc = tpr \*w + tnr \*(1-w)

To do this first rework this equation to find the intercept and slope of the isometric, for a given accuracy.

You can then calculate these values and add the accuracy isometric to the plot with the following code:

w = XXXX

intercept= XXXX

slope = XXXX

g + geom\_abline(intercept=intercept, slope = slope, color = 'red')

Now imagine the class distribution is highly imbalanced e.g. only 10% of the examples are positive (have heart disease).

Explore how changing the class distribution impacts the accuracy of the classifier, and the accuracy of a classifier at different points of the ROC space.

Where is accuracy impacted less by the class distribution and why?