# Exercise 6

Business Analytics and Data Science WS16/17

## Introduction

The key element in the data-driven paradigma even beyond machine learning is empirical evaluation. If the application and modern technology allow us to collect data on the performance of our prediction model, marketing campaign, medical treatment or diet at a reasonable cost, we can develop procedures to monitor and test our approaches in a scientific way. The most important guideline is to never test on the same data that you used for training (in the widest sense). But we rarely want to wait for new data to come in, so there are several strategies how to create a good test set from the raw data. By the way, this is how tests work in general. You don't expect the exam to be a collection of the exact homework exercises and you drive an unknown route when taking your driving test.

## Regularization revisited (and done right)

Your toolbox of predictive models currently contains two models that we have discussed in some detail: regularized regression and decision trees. One of these may perform better than the other depending on the data. For example, linear regression fits well if the effects are indeed linearly additive and the error is random. A decision tree works fits the non-linear effects in the data better. We could investigate the data structure and make convincing arguments for each model and this is not a bad starting point if there exist strong theoretical insights. Most of the time, we can avoid this discussion by comparing model performance. Of course, we can't compare the performance of each model and each parameter choice within one model without a metric to evaluate performance. This brings us to
**Question Evaluation metric: How to measure how good a prediction is in the setting?**
Once we can compare models, we want a fair answer to
**Question Algorithm selection: Which model gives the best predictions in the setting?**
But as we have seen, regularized regression and decision tress (and most other complex models) depend on our choice of meta-parameters, i.e. the options that guide the algorithms. For your two models, you can choose any value for *lambda* and *cp*, the regularization parameters of each model. To make a fair comparison, both models should have optimal conditions. So before we can tackle Question A, we need an answer to
**Question Parameter tuning: Which parameter choice gives the best prediction in the setting?**

## Evaluation metric

How to measure model performance is an important fundamental question that is unfortunately hard to answer. Let's look at our choices before making a decision.

1. Load the *loans* dataset and split it into the predictors **x** and target variable **y**.
2. Train a logistic regression **lr**, a lasso-regularized logistic regression **lasso** and a decision tree **dt** on the dataset. We ignore question A and B for now, so any metaparameter setting is fine.
3. If you used the default metaparameters for model training, find out what the default parameters are. It's always good to know what you are doing.
4. Predict the probability of each applicant to default on their loan and save the result to a dataframe called **yhat**.

Now let's have a look at those metrics.

1. Create a readable confusion matrix using the function **confusionMatrix()** from package **caret**. For the logit and pruned decision tree model above, plot the true values vs. the model prediction in a confusion matrix.

Hint: Confusion tables require discrete class predictions and discrete target values. Choose a threshold value **tau** and transform you probability predictions to class predictions. Which threshold did you choose and why?

2. Use package **hmeasure** to plot the ROC curves for the logistic regression *lr* and the decision tree *dt*. The package requires the predictions to be in a data frame with one column for every model (call it **predictions**). Use the **HMeasure()** function to create an **HMeasure** object **h** containing all the necessary information for the plot. Then create the plot using function **plotROC()**.
   Receiver operating characteristic curves, whose strange name is a remnant of their original use with radar systems in WWII, illustrates the performance of a binary classifier for different cut-offs (probability thresholds). For each cut-off **tau** between 0 and 1, it plots the true positive rate (a.k.a. sensitivity) against the false positive rate (a.k.a. 1 - specificity). Take a minute to see how these concepts relate to each other.
   The resulting ROC curves lie between the 45° baseline (no predictive power) and the upper left corner of the plot. The further the ROC curve lies towards the corner, the more accurate the model. Again, take a minute to really understand why this is the case. Keep in mind that each point on the ROC curve visualizes the true and false positive rates over several values for the threshold **tau**.

3. Compute the Area-under-the-ROC-curve (AUC) for both ROC curves. HINT: Extract the AUC values from the **HMeasure** object.
   The AUC is a way to quantify the performance of a classifier illustrated by the ROC curve in a single value. It represents the area under a ROC curve, so it takes on values between 0.5 and 1, where higher is better.

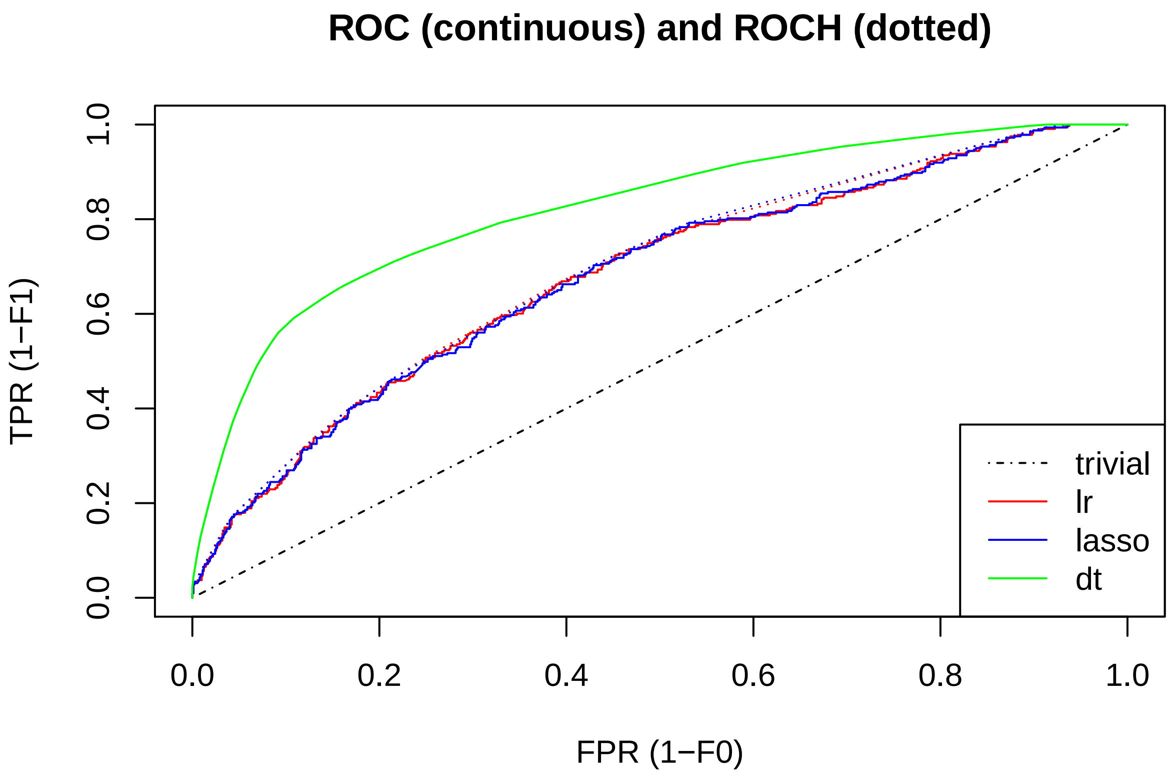
```
##        lr     lasso        dt
## 0.1778411 0.1778565 0.1363362
```

```
##
##        good bad
##   good  557 113
##   bad   345 210
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction good bad
##       good  553 115
##       bad   349 208
##
##                Accuracy : 0.6212
##                  95% CI : (0.5934, 0.6485)
##     No Information Rate : 0.7363
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.2086
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.6440
##             Specificity : 0.6131
##          Pos Pred Value : 0.3734
##          Neg Pred Value : 0.8278
##              Prevalence : 0.2637
##          Detection Rate : 0.1698
##    Detection Prevalence : 0.4547
##       Balanced Accuracy : 0.6285
##
```

```
##        'Positive' Class : bad
##
## Confusion Matrix and Statistics
##
##           Reference
## Prediction good bad
##       good  739 104
##       bad   163 219
##
##                Accuracy : 0.782
##                  95% CI : (0.7579, 0.8049)
##     No Information Rate : 0.7363
##     P-Value [Acc > NIR] : 0.0001230
##
##                   Kappa : 0.4698
##  Mcnemar's Test P-Value : 0.0003859
##
##             Sensitivity : 0.6780
##             Specificity : 0.8193
##          Pos Pred Value : 0.5733
##          Neg Pred Value : 0.8766
##              Prevalence : 0.2637
##          Detection Rate : 0.1788
##    Detection Prevalence : 0.3118
##       Balanced Accuracy : 0.7487
##
##        'Positive' Class : bad
##
## Confusion Matrix and Statistics
##
##           Reference
## Prediction good bad
##       good  709  94
##       bad   193 229
##
##                Accuracy : 0.7657
##                  95% CI : (0.741, 0.7892)
##     No Information Rate : 0.7363
##     P-Value [Acc > NIR] : 0.01003
##
##                   Kappa : 0.4507
##  Mcnemar's Test P-Value : 7.262e-09
##
##             Sensitivity : 0.7090
##             Specificity : 0.7860
##          Pos Pred Value : 0.5427
##          Neg Pred Value : 0.8829
##              Prevalence : 0.2637
##          Detection Rate : 0.1869
##    Detection Prevalence : 0.3445
##       Balanced Accuracy : 0.7475
##
##        'Positive' Class : bad
```

```
##
##                F
## lr    0.2382134
## lasso 0.2250000
## dt    0.6166951
```



```
##             AUC
## lr    0.6788870
## lasso 0.6785540
## dt    0.8201262
```

## Split-sample testing

### Model Selection: Picking the model

You have chosen a metric and are ready to compare models. Let's ignore the question of metaparameters for now and focus on a fair comparison of models. Training a model on one set of examples and testing it on the same observations is usually not a good idea, since we may pick a model that is too focused on the specific examples. A better strategy is to keep some examples from the model in the first place and use this data to evaluate how well the model is able to predict new observations.

A straight-forward way to do this, is to separate the full data into a training and a test set. 1. Split your loan data randomly into a training set of 75% and a test set of 25% by creating two new data frames **tr** and **ts** to which you randomly assign the correct proportion of observations from data set **loans**. Hint: Check out function **sample()**.
Note: Package **caret** has a function **createDataPartition()** that does stratified sampling, i.e. keeps the same ratio of defaulters to non-defaulters.

2. Train the logistic regression models and the decision tree *on the training data only*. Use your models to make a prediction for the credit risk of the applicants *in the test data set*.

3. Compare the predicted outcomes to the actual outcomes on the test data set.

```
##             AUC
## lr    0.6788870
## lasso 0.6785540
## dt    0.8201262
```

```
##             AUC
## lr    0.6066111
## lasso 0.6116667
## dt    0.5222222
```

### Model Selection: Picking the metaparameters

The metaparameters that you choose for each algorithm can make a substantial difference. In this case, the decision tree may seem much worse than the simple logit model even though it is able to detect non-linear effects in the data. You should select the best choice of parameters for each model when comparing them to evaluate them at their best performance. Selecting the best parameter set is another step in our testing procedure and you should follow the same guideline as before: Evaluate on fresh data. If you optimize the parameters based on a set of observations (by selecting the parameters for which the model performs best), you will need different data to compare the algorithms and make sure that model performance holds up.

1. Sketch out in a tree structure on a piece of paper how the data is split up and for what each data subset is used.
2. Split off another test set from the *training data* and call it **val** for *validation set.* This leaves a smaller training set **tr_val**.
3. Choose some values for *lambda* and *cp* that you want to evaluate. How do you explain why you picked these values in particular?
4. Function **glmnet()** saves a model for lambda over a range. Train a lasso model on the *validation training* data, loop over these lambda candidates, make a prediction on the validation set and save the performance value.
5. In a loop, build a decision tree for each of the metaparameters and evaluate it on the validation set.
6. Compare the performance of each model to find the best *lambda* and *cp*, respectively.
7. Retrain the models with their best metaparameter on the full training data and compare them on the original *test* dataset.

```
##         lambda       AUC
## 25 0.009852435 0.6652845
```

```
##             cp       AUC
## 19 0.009183673 0.5474928
```

```
##             AUC
## lr    0.6788870
## lasso 0.6785540
## dt    0.8201262
```

```
##             AUC
## lr    0.6066111
## lasso 0.6116667
## dt    0.5222222
```

```
##             AUC
## lr    0.6066111
## lasso 0.6139722
## dt    0.5233611
```