Machine Learning (ML) for Data-Driven Decision-Making (DDDM)

Module 2: Supervised Learning

Tobias Rebholz

University of Tübingen

Summer Term 2024

Organization

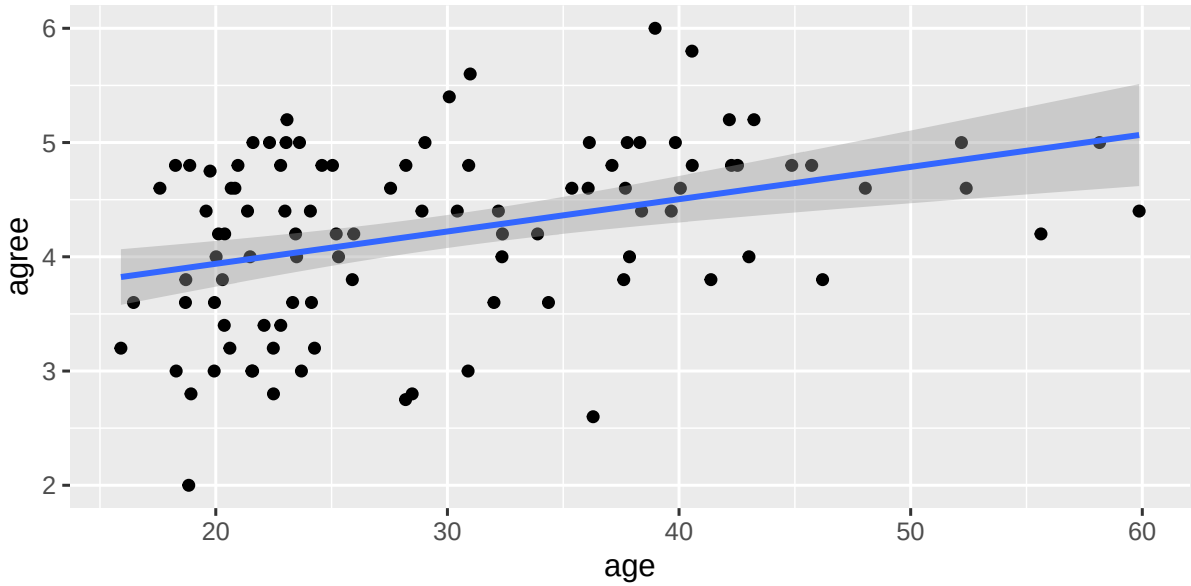
Literature

- Unfortunately, I cannot share my pdf copy of Jacobucci et al. (2023)
  - You have to work with the 14-days period of the version provided by the library
  - Or use other ways to get a copy ;)
- Who did the reading for today, especially Pargent et al. (2023)?
  - This resource contains important details about best practices for performing your own group projects!

Regularized Regression

Refresher: Simple Linear Regression

```
ggplot(dat, aes(y = agree, x = age)) +
  geom_point() +
  stat_smooth(method = "lm")
## `geom_smooth()` using formula = 'y ~ x'
```



Refresher: Multiple Linear Regression

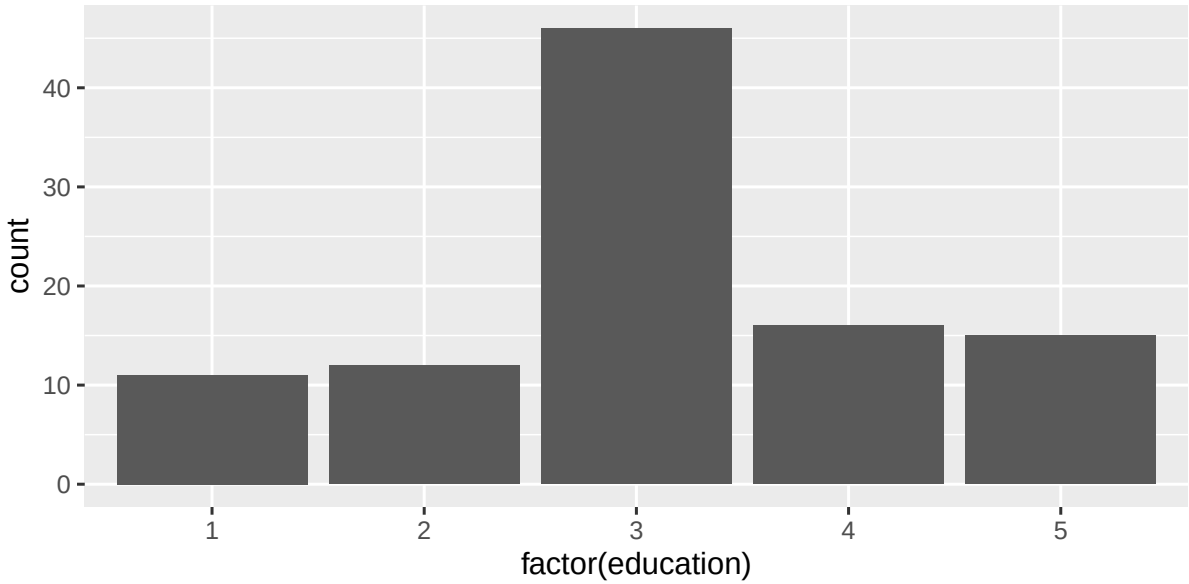
- For each instance i in a population, we have:
  - A vector of features, $X_i = (x_{i1}, x_{i2}, \ldots, x_{ip})$
  - Continuous target, $y_i \in \mathbb{R}$
- Goal: Predict the target for new instances for whom we know the values of the features but not the value of the target:

$$X_{new} \rightarrow \hat{y}_{new} \in \mathbb{R} \tag{1}$$

  - We assume that there is a linear relationship between the features and the target: $\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \cdots + \hat{\beta}_p x_p$

Refresher: Multiple Linear Regression

```
ggplot(dat, aes(x = factor(education))) +
  geom_bar()
```



- Note the imbalance: Most instances have $\texttt{education} = 3$
- For simplicity, let's assume that educational level is a continuous variable in the following

Refresher: Multiple Linear Regression

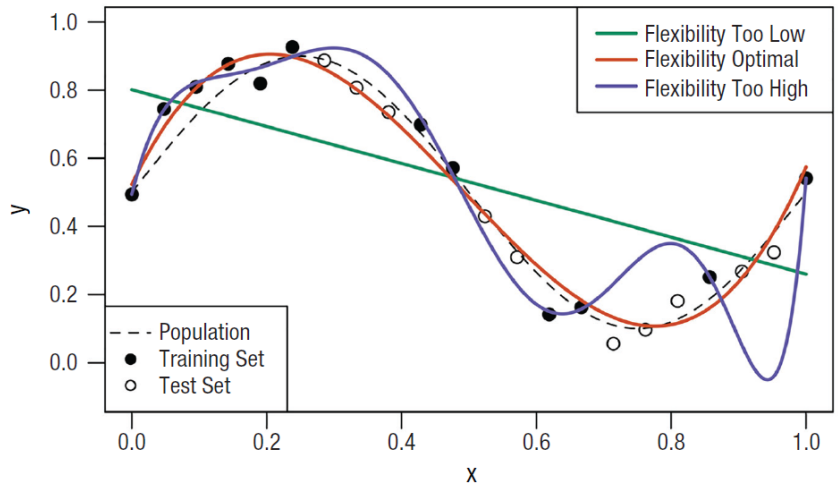
```
tsk <- as_task_regr(education ~ ., data = dat %>% select(-CASE))
mdl <- lrn('regr.lm')
mdl$train(tsk)
summary(mdl$model)
##
## Call:
## stats::lm(formula = task$formula(), data = task$data())
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.81463 -0.55259  0.06536  0.60809  2.38311
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)   2.42087    1.33230   1.817   0.0725 .
## age           0.02552    0.01183   2.158   0.0335 *
## agree        -0.35970    0.16913  -2.127   0.0361 *
## conscientious 0.43552    0.21520   2.024   0.0459 *
## extra         0.06800    0.23578   0.288   0.7737
## gender        0.35721    0.23582   1.515   0.1333
## neuro        -0.28817    0.09841  -2.928   0.0043 **
## open         -0.03406    0.20703  -0.165   0.8697
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.089 on 92 degrees of freedom
## Multiple R-squared:  0.1643,  Adjusted R-squared:  0.1007
```

(Automatic) Variable Selection

- If we have many features that may potentially explain the target (e.g., personality traits), how to choose among them?
  - **Overfitting:** Including too many features can result in a model that fits the training data too closely
    - High risk of capturing noise rather than the underlying relationships!
    - Cf. learning something by heart: Exactly recognizing each training instance but inability to transfer this knowledge to new observations
  - **High dimensionality:** A large number of features increases the complexity of the model
    - Computationally intensive and difficult to interpret!
  - **Multicollinearity:** Many features have a higher risk of being linearly dependent on each other
    - Difficult to determine the unique contribution of each feature!
- Least Absolute Shrinkage and Selection Operator (LASSO): Regression models that penalize the absolute size of the estimated coefficients
  - Penalization/Regularization shrinks some of the coefficients towards zero $\Rightarrow$ LASSO tends to use a lower number of features, effectively **selecting the most important ones**

Overfitting

- Remember the bias-variance trade-off: Good test set performance requires low variance as well as low squared bias
  - The challenge lies in finding a model for which both the variance and the squared bias are low
  - I.e., we can get the red model by removing polynomial terms (i.e., flexibility) from the blue model



(Pargent et al., 2023, Figure 3a)

Regularized Regression

- The LASSO relies upon the linear model but uses an alternative fitting procedure for estimating the coefficients $\beta_0, \beta_1, \ldots, \beta_p$
  - In contrast to least squares only, we minimize the difference between the predicted and observed target values as follows:

$$\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{p} \|\beta_j\| \tag{2}$$

- λ is called the "regularization", "tuning parameter" or "hyperparameter"
  - It controls the trade-off between minimizing the error on the training data (i.e., fitting the data well; first term) and penalizing model complexity (second term)
  - A larger value penalizes the coefficients more heavily, leading to a simpler model (i.e., less features) with potentially higher bias but lower variance

Regularized Regression in mlr3

```
tsk = as_task_regr(education ~ ., data = dat %>% select(-CASE))
mdl = lrn("regr.glmnet", lambda = 0.1)
mdl$train(tsk)
coef(mdl$model) %>% round(., 4)
## 8 x 1 sparse Matrix of class "dgCMatrix"
##                      s0
## (Intercept)    2.7135
## age            0.0105
## agree         -0.0260
## conscientious  0.1505
## extra          .
## gender         0.0333
## neuro         -0.1331
## open           .
```

Regularized Regression in mlr3

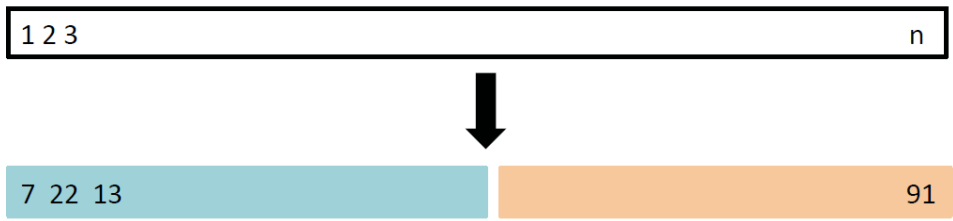
- Instead of arbitrarily choosing $\lambda = 0.1$, we can (rather: should!) try different values:

```
mdl = lrn("regr.glmnet", nlambda = 5)
mdl$train(tsk)

mdl$model$lambda %>% round(., 4)
## [1] 0.2714 0.0271 0.0027 0.0003 0.0000

coef(mdl$model) %>% round(., 4)
## 8 x 5 sparse Matrix of class "dgCMatrix"
##                    s0      s1      s2      s3      s4
## (Intercept)   3.12  2.5881  2.4334  2.4221  2.4209
## age            .     0.0216  0.0251  0.0255  0.0255
## agree          .    -0.2635 -0.3504 -0.3588 -0.3596
## conscientious  .     0.3514  0.4264  0.4346  0.4354
## extra          .     .       0.0587  0.0671  0.0679
## gender         .     0.2711  0.3488  0.3564  0.3571
## neuro          .    -0.2461 -0.2839 -0.2877 -0.2881
## open           .     .      -0.0261 -0.0333 -0.0340
```

Validation Set Approach

- How to select the tuning- or hyperparameter?
  - Easiest possibility: Validation set approach

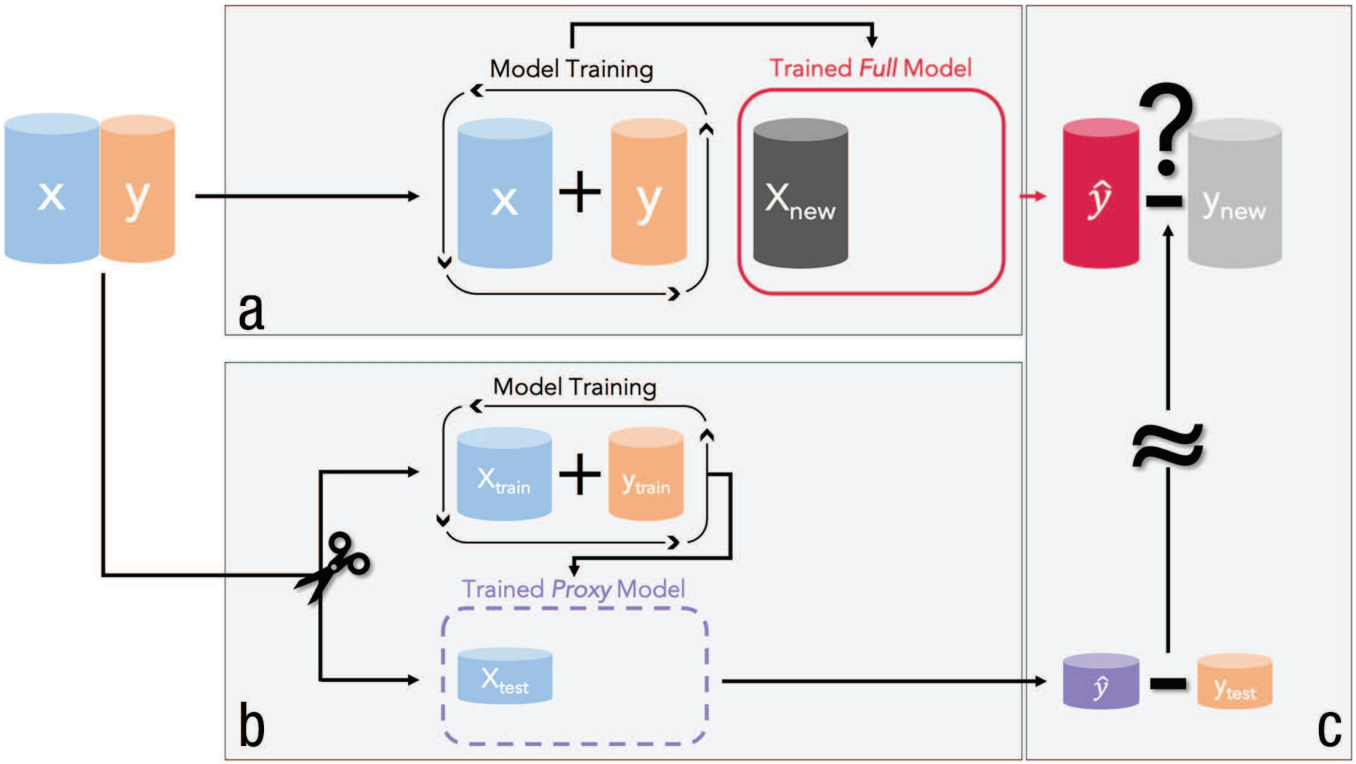
1. Dataset is split into training set and validation set
2. Classifier is trained on training set, and performance is reported on validation set



(James et al., 2021, Figure 5.1)

Validation Set Approach

- The out-of-sample prediction performance on the validation set is a good (but conservative!) proxy for the real-world testing performance of a model



(Pargent et al., 2023, Figure 2)

Validation Set Approach in mlr3

1. Separating the data into 2/3 training and 1/3 test or validation data (mlr3's default; see ?partition)

```
set.seed(42)
row_ids <- partition(tsk)
row_ids
## $train
##  [1]   1   6   9  12  13  14  19  20  24  25  28  32  39  45  53  55  56  62  65
## [20]  66  73  75  80  82  90  94  95  97  98  99 100   3   7   8  10  15  16  33
## [39]  44  48  52  58  59  68  70  71  72  77  78  81  83  93   4  11  17  18  38
## [58]  41  42  50  51  57  60  69  74  79  89
##
## $test
##  [1] 23 26 29 35 36 43 46 47 49 63 64 67 76 91 92  2 21 22 27 30 31 34 37 40 88
## [26]  5 54 61 84 85 86 87 96
```

Validation Set Approach in mlr3

2. Building the model with the training data and predicting the validation data
   - Problem: mlr3's predict() ability/method does not (yet) support multiple lambda values (see https://github.com/mlr-org/mlr3learners/issues/10)

```
mdl = lrn("regr.glmnet", nlambda = 5)
mdl$train(tsk, row_ids = row_ids$train)

pred <- mdl$predict(tsk, row_ids = row_ids$test)
## Warning: Multiple lambdas have been fit. Lambda will be set to 0.01 (see
## parameter 's').

tail(cbind('true' = dat[row_ids$test,]$education, 'pred' = pred$response))
##       true     pred
## [28,]    1 3.110266
## [29,]    1 2.995154
## [30,]    2 2.839012
## [31,]    2 2.653498
## [32,]    1 3.356862
## [33,]    1 2.989942
```

- Note the issue of treating the categorical eduction variable as continuous target: We predict nonexistent education levels

Validation Set Approach in mlr3

2. Building the model with the training data and predicting the validation data
   - Solution: We can use glmnet's predict() function

```
# Separation of X and y (needed for glmnet):
X <- tsk$data(rows = row_ids$test) %>% select(-education)

# Prediction:
pred <- predict(mdl$model, newx = as.matrix(X))
tail(cbind('true' = dat[row_ids$test,]$education, pred))
##       true       s0       s1       s2       s3       s4
## [28,]    1 3.134328 3.123115 3.099214 3.096784 3.096514
## [29,]    1 3.134328 3.017492 2.975942 2.971788 2.971362
## [30,]    2 3.134328 2.861578 2.819603 2.815336 2.814942
## [31,]    2 3.134328 2.708138 2.606503 2.596275 2.595401
## [32,]    1 3.134328 3.345997 3.366208 3.368253 3.368341
## [33,]    1 3.134328 2.995502 2.985160 2.984068 2.983959
```

- Note the issue of treating the categorical eduction variable as continuous target: We predict nonexistent education levels

Selecting the Hyperparameter: Validation Set Approach

3. Minimizing the out-of-sample MSE

```
MSE_pred <- colMeans((pred - dat[row_ids$test,]$education)^2)
MSE_pred
##       s0       s1       s2       s3       s4
## 1.539075 1.401265 1.382790 1.381286 1.381108

# Which value of the hyperparameter (lambda) yields the smallest out-of-sample MSE?
lambda_best <- which.min(MSE_pred)
lambda_best
## s4
##  5

# Choosing the model with the best out-of-sample prediction performance:
coef(mdl$model)[,lambda_best]
##   (Intercept)           age         agree conscientious         extra
##    2.51786287    0.01721992   -0.45013862    0.27807424    0.07246447
##        gender         neuro          open
##    0.50542213   -0.16124003    0.08723527
```

Excurse: Ridge Regression

- Similar to LASSO, but stabilizing predictions by shrinking (i.e., making smaller) the coefficients, instead of setting some of them to exactly zero
  - As expected, none of the coefficients is exactly zero for any value of `lambda`:

```
options(digits=4) #reduce number of digits printed in output

mdl = lrn("regr.glmnet", nlambda = 5, alpha = 0)
mdl$train(tsk)
coef(mdl$model)
## 8 x 5 sparse Matrix of class "dgCMatrix"
##                       s0         s1        s2       s3       s4
## (Intercept)    3.120e+00  3.0355299  2.665439  2.41302  2.41756
## age            2.382e-38  0.0009393  0.006565  0.01911  0.02466
## agree         -1.683e-38 -0.0011843 -0.030462 -0.21649 -0.33895
## conscientious  2.977e-37  0.0120058  0.093894  0.31366  0.41884
## extra          1.196e-37  0.0044062  0.020176  0.04116  0.06343
## gender         1.837e-37  0.0075193  0.063956  0.24549  0.34166
## neuro         -2.217e-37 -0.0088155 -0.064815 -0.20760 -0.27706
## open           1.048e-37  0.0038101  0.014118 -0.01108 -0.03079

options(digits=6) #change back to default
```

Support Vector Classifier

Refresher: Logistic Regression

- Everything is the same as in linear regression, except that we have discrete target
- For each instance i in a population, we have:
  - A vector of features, $X_i = (x_{i1}, x_{i2}, \ldots, x_{ip})$
  - **Binary** class membership, $y_i \in \{0, 1\}$
    - E.g., buying vs. not buying a specific product
  - Probability of membership in class 1, p, and probability of membership in class 0, $1 - p$
    - **Continuous, but bounded** target
- Goal: Predict the target for new instances for whom we know the vector of features but not the value of the target:

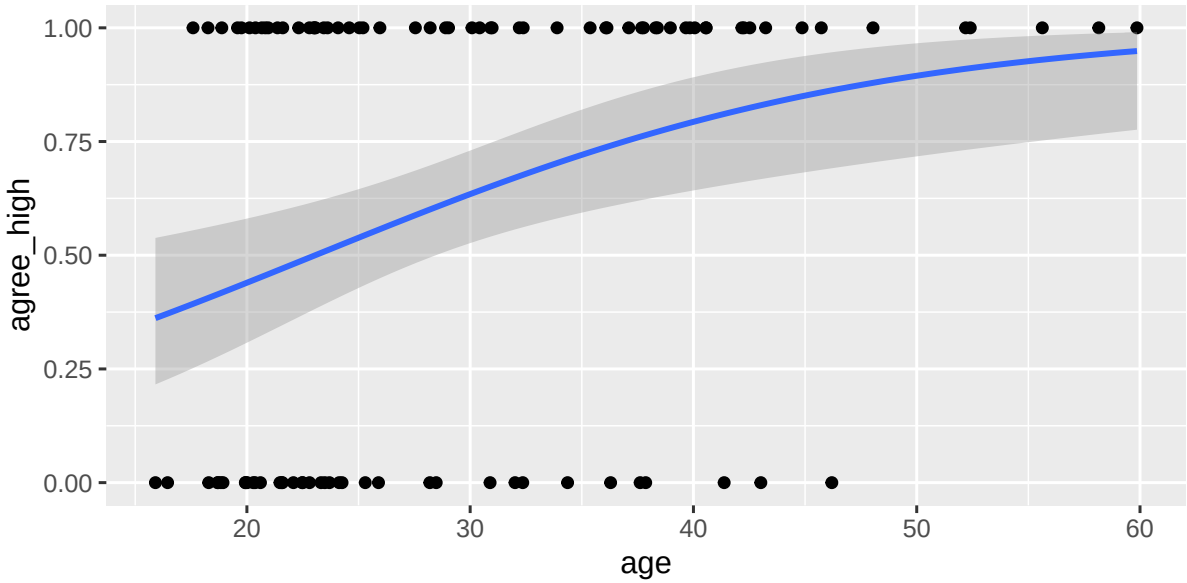
$$X_{new} \rightarrow \hat{p}_{new} \in (0, 1) \tag{3}$$

  - Predicted probability of class 1:

$$\hat{p} = \frac{e^{\hat{\beta}_0 + \hat{\beta}_1 x_1 + \cdots + \hat{\beta}_p x_p}}{1 + e^{\hat{\beta}_0 + \hat{\beta}_1 x_1 + \cdots + \hat{\beta}_p x_p}} \tag{4}$$

Refresher: Logistic Regression

```
df <- dat
df$agree_high <- ifelse(df$agree > 4, 1, 0)
df %>%
  ggplot(aes(y = agree_high, x = age)) +
  geom_point() +
  geom_smooth(method = "glm", method.args = list(family = binomial(link = "logit")))
## `geom_smooth()` using formula = 'y ~ x'
```



Refresher: Logistic Regression

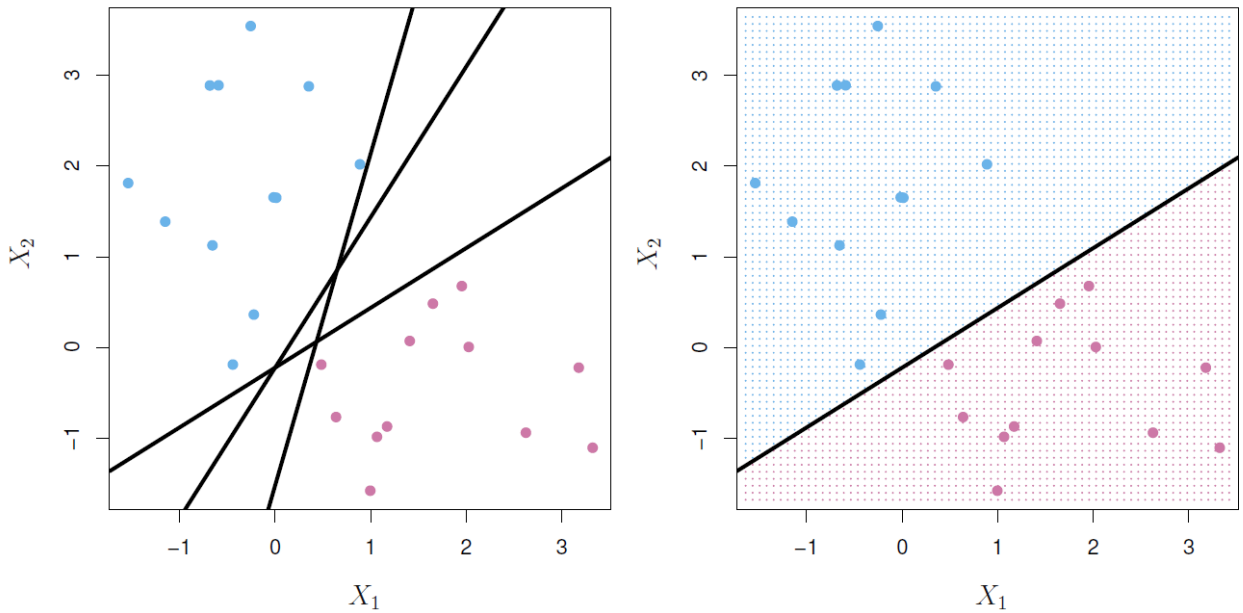
```
tsk = as_task_classif(agree_high ~ age, data = df, positive = '1')
mdl = lrn("classif.log_reg")
mdl$train(tsk)
summary(mdl$model)
##
## Call:
## stats::glm(formula = task$formula(), family = "binomial", data = data,
##     model = FALSE)
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -1.8312     0.7368   -2.49   0.0129 *
## age           0.0794     0.0256    3.10   0.0019 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 133.75  on 99  degrees of freedom
## Residual deviance: 121.83  on 98  degrees of freedom
## AIC: 125.8
##
## Number of Fisher Scoring iterations: 4
```

Other Types of Classifiers

- Linear classifiers:
  - Linear Discriminant Analysis
  - Logistic Regression
  - Support Vector Machines
  - ...
- Nonparamtertic classifiers:
  - Classification trees
  - Random forests
  - Nearest neighbors
  - ...
- The rest of the day is mainly about using these methods for classification tasks
  - But they can also be used for regression tasks (not discussed!), which usually requires a few adaptations

Support Vector Classifier

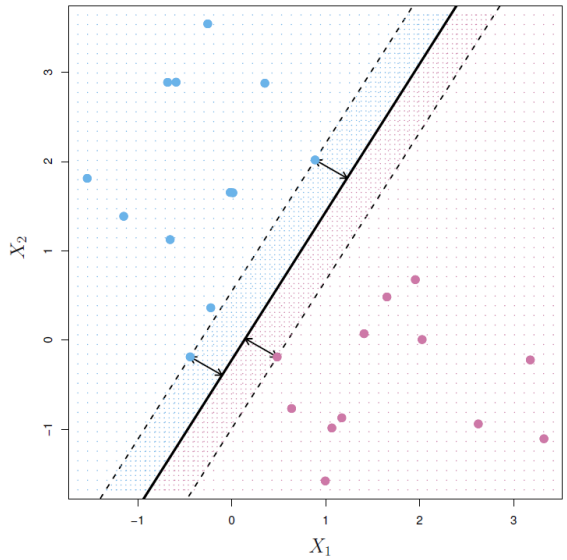
- There is a linear decision boundary (or "hyperplane") used to define the prediction: $\beta_0 + \sum_{j=1}^{p} \beta_j x_j = 0$
  - Prediction depends on whether an instance is above or below this boundary:



(James et al., 2021, Figure 9.2)

Support Vector Classifier

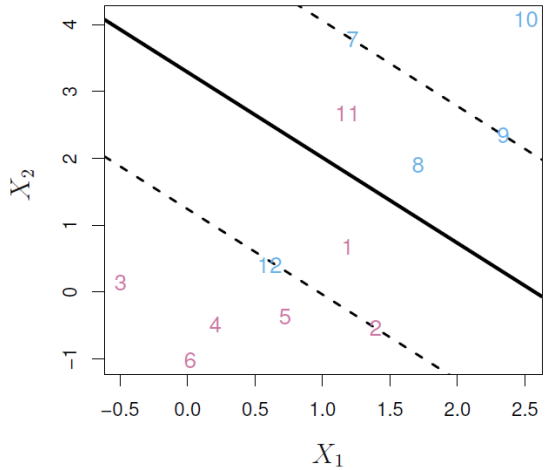
- Support Vector Classifier (SVC): Separating the classes with a hyperplane that maximizes the margin
  - Margin (dashed line): The distance between the hyperplane representing the decision boundary and the data
  - Predicted class: $\hat{y} = \begin{cases} 1 \text{ if } \hat{\beta}_0 + \sum_{j=1}^{p} \hat{\beta}_j x_j > 0 \\ -1 \text{ else} \end{cases}$



(James et al., 2021, Figure 9.3)

Support Vector Classifier

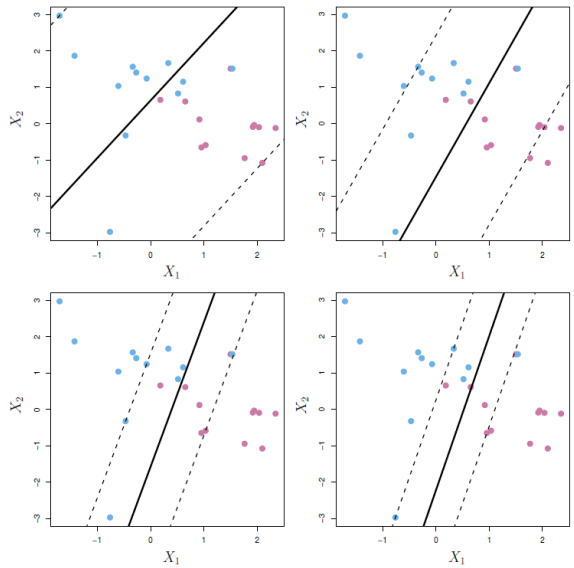
- Very good accuracy compared to other linear classifiers, but requires more technicalities:
  - Hard margin: Requires correct classification for all instances (see previous slide)
    - Overfitting $\Rightarrow$ Poor generalization!
  - Soft margin: We do not require that all instances are correctly classified
    - I.e., some instances can be on the wrong side of the hyperplane



(James et al., 2021, Figure 9.6)

Support Vector Classifier

- C is the hyperparameter for the trade-off between the size of the (soft) margin and correct classification
  - Cf. λ in regularized regression: Controlling the trade-off between minimizing the error on the training data and penalizing model complexity
  - Larger C (top left; decreasing to bottom right) = Higher tolerance (i.e., less penalization) of misclassification in the training dataset



(James et al., 2021, Figure 9.7)

Support Vector Classifier in `mlr3`

- Data preparation:

```
dat <- dat %>%
  mutate(gender = ifelse(gender == 1, 'male', 'female'))

head(dat)
##    CASE gender education     age agree conscientious extra neuro open
## 1 63116   male         3 40.5575   5.8           3.4   3.6   1.5  3.8
## 2 63967   male         4 35.3734   4.6           3.6   4.0   1.0  3.6
## 3 63955 female         4 21.5592   3.0           3.2   4.0   3.8  4.4
## 4 62547 female         1 22.8009   4.8           5.2   4.4   2.0  4.8
## 5 63493 female         2 23.0748   5.2           3.4   4.4   2.6  4.6
## 6 62419 female         3 30.0812   5.4           4.0   4.4   4.0  3.8
```

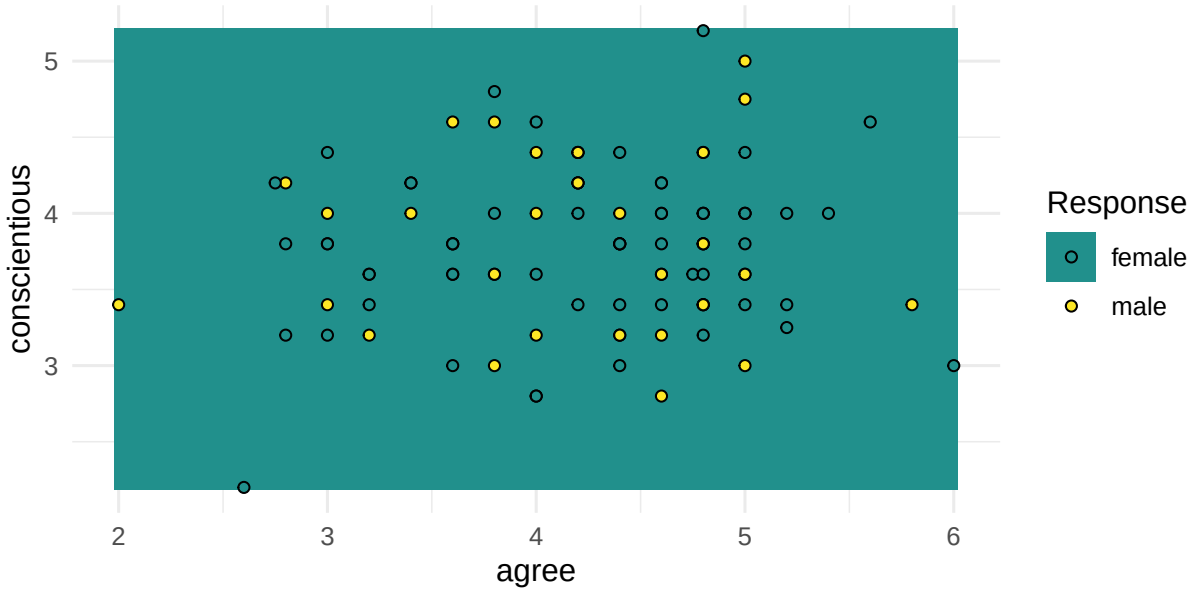
Support Vector Classifier in mlr3

```
tsk = as_task_classif(gender ~ agree + conscientious, data = dat, positive = 'male')
mdl = lrn("classif.svm", type = 'C-classification', cost = 100, kernel = 'linear')
mdl$train(tsk)
summary(mdl$model)
##
## Call:
## svm.default(x = data, y = task$truth(), type = "C-classification",
##     kernel = "linear", cost = 100, probability = (self$predict_type ==
##         "prob"))
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  100
##
## Number of Support Vectors:  78
##
##  ( 34 44 )
##
##
## Number of Classes:  2
##
## Levels:
##  male female
```

Support Vector Classifier in `mlr3`

- The output summary is not as informative for SVMs as for regression models
  - But the plot of the hyperplane reveals some serious problems:

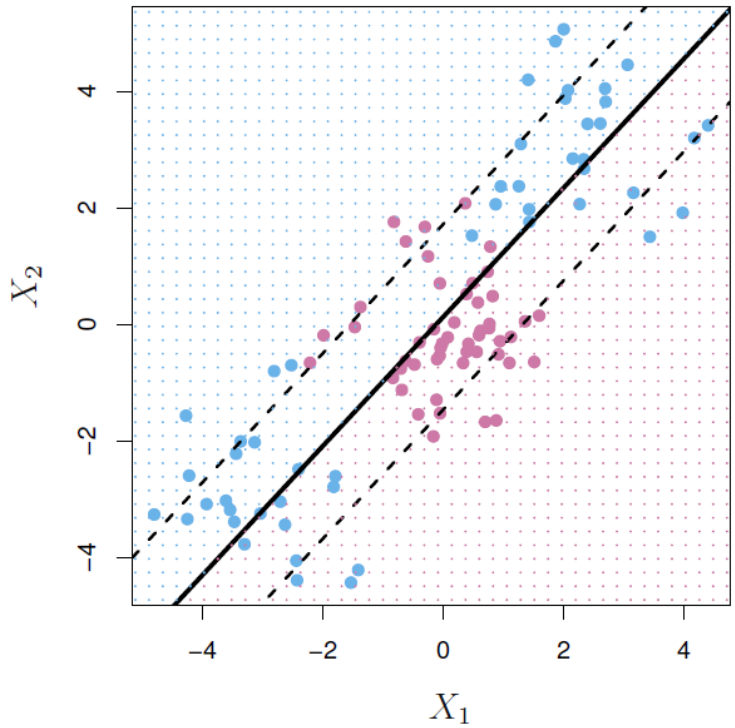
```
autoplot(mdl, task = tsk) + scale_fill_viridis_d(begin = .5)
## Scale for fill is already present.
## Adding another scale for fill, which will replace the existing scale.
```



Support Vector Machines

Support Vector Machines

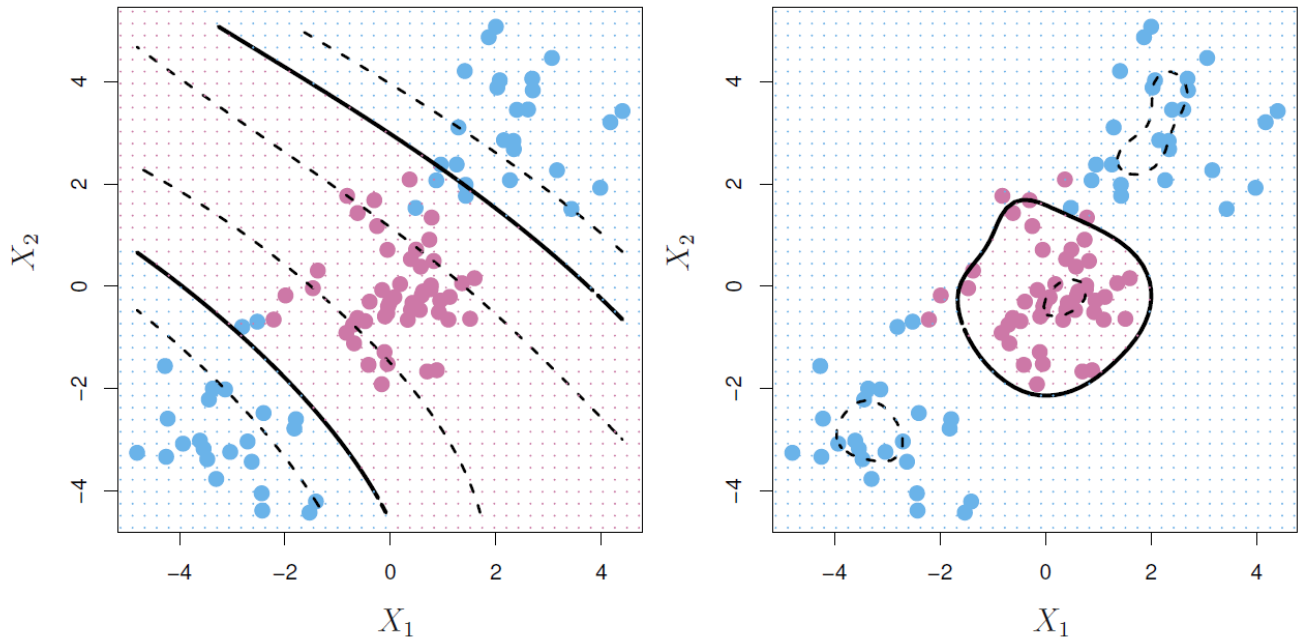
- Challenges for linear classifiers:



(James et al., 2021, Figure 9.8)

Support Vector Machines

- Solution: Nonlinear decision boundaries



(James et al., 2021, Figure 9.9)

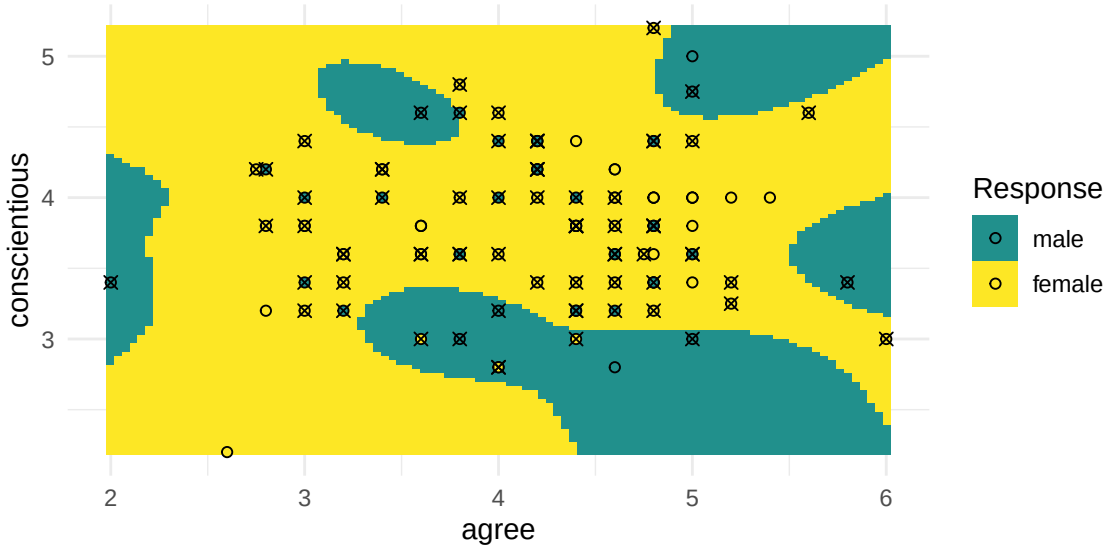
Support Vector Machines in mlr3

- Using a nonlinear `kernel` (default: "radial"):

```
mdl = lrn("classif.svm", type = 'C-classification', cost = 100, kernel = 'radial')
mdl$train(tsk)
summary(mdl$model)
##
## Call:
## svm.default(x = data, y = task$truth(), type = "C-classification",
##     kernel = "radial", cost = 100, probability = (self$predict_type ==
##         "prob"))
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  100
##
## Number of Support Vectors:  75
##
##  ( 32 43 )
##
##
## Number of Classes:  2
##
## Levels:
##  male female
```

Support Vector Machines in mlr3

```
autoplot(mdl, task = tsk) + scale_fill_viridis_d(begin = .5) +
  geom_point(data = tsk$data()[mdl$model$index,], shape = 4, size = 2)
## Scale for fill is already present.
## Adding another scale for fill, which will replace the existing scale.
```



- Only instances that lie directly on the margin, or on the wrong side of the margin for their class, affect the SVM classifier
  - These instances are the so-called "support vectors" and are marked with crosses
  - The remaining instances play no role for classification

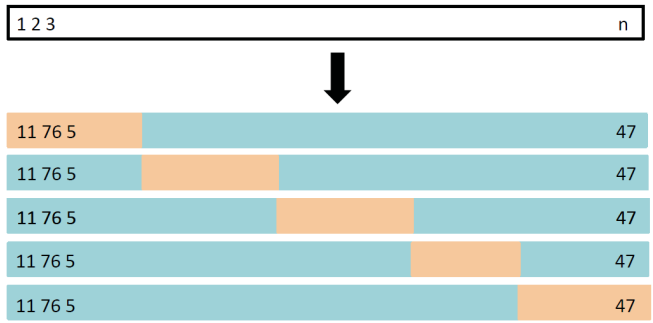
Support Vector Machines in mlr3

- Training classification performance
  - Overfitting ⇒ Too optimistic!

```
pred <- mdl$predict(tsk)

pred$confusion
##          truth
## response male female
##   male     10      3
##   female   24     63

mes <- msrs(c("classif.ce", "classif.acc", "classif.recall", "classif.specificity"))
pred$score(mes)
##          classif.ce         classif.acc      classif.recall classif.specificity
##            0.270000            0.730000            0.294118            0.954545
```

Cross-Validation

- k-fold cross-validation (CV): More elaborated extension of the validation set approach to assess out-of-sample prediction performance
  1. Dataset is split into k folds
  2. One fold (e.g., 20% in 5-fold CV) is left out as validation set, the remaining is used as training set
  3. Classifier is trained on the training set and tested on the validation set
  4. Steps 2 and 3 are repeated for each fold, and average validation performance is reported: $CV_{(k)} = \frac{1}{k} \sum_{i=1}^{k} MMCE_i$



(James et al., 2021, Figure 5.5)

Cross-Validation in mlr3

- The mlr3 library includes a built-in function to perform CV

```
set.seed(42)
cv <- rsmp ("cv", folds = 5)
mdl_cv <- resample(learner = mdl, task = tsk, resampling = cv)
## INFO  [12:33:20.746] [mlr3] Applying learner 'classif.svm' on task 'dat' (iter 1/5)
## INFO  [12:33:20.851] [mlr3] Applying learner 'classif.svm' on task 'dat' (iter 2/5)
## INFO  [12:33:20.917] [mlr3] Applying learner 'classif.svm' on task 'dat' (iter 3/5)
## INFO  [12:33:20.948] [mlr3] Applying learner 'classif.svm' on task 'dat' (iter 4/5)
## INFO  [12:33:21.003] [mlr3] Applying learner 'classif.svm' on task 'dat' (iter 5/5)

# Out-of-sample performance
mdl_cv$aggregate(mes)
##          classif.ce         classif.acc      classif.recall classif.specificity
##           0.4600000           0.5400000           0.0285714           0.8042125

# Remember: In-sample performance
pred$score(mes)
##          classif.ce         classif.acc      classif.recall classif.specificity
##            0.270000            0.730000            0.294118            0.954545
```

- Note: Out-of-sample performance is worse than in-sample performance (to be expected!)

Hyperparameter Tuning

- CV can be used to choose a good value for the tuning- or hyperparameter C
  - E.g., choosing the `cost` parameter C for SVM to maximize out-of-sample classification accuracy
- Remember: Hyperparameters are external configuration variables that control the training/behavior of the ML model
  - Their values are manually set before training a model (e.g., regularization constant λ in regularized regression)
  - In contrast, values of internal parameters are automatically derived during the learning process (e.g., regression coefficients β)

Hyperparameter Tuning in `mlr3`

1. Define the set of values for C that should be tested

```
C_cv <- c(10, 50, 100, 500, 1000)
```

2. Set up the conditions for the hyperparameter tuning using the `auto_tuner()` function
   - Which model should be trained? Which resampling method (i.e., validation approach) should be used? How should performance be assessed? ...

```
mdl_cv = auto_tuner(
  learner = lrn("classif.svm", type = 'C-classification', cost = to_tune(levels = C_cv)),
  resampling = rsmp("cv", folds = 5),
  measure = msr("classif.ce"),
  tuner = tnr("grid_search"),
  terminator = trm("none")
)
```

Hyperparameter Tuning in `mlr3`

3. Perform the hyperparameter tuning
   3.1. For each potential value of C defined in Step 1, perform a k-fold CV using the `train()` argument on the to-be-tuned model from Step 2:

```
set.seed(42)
mdl_cv$train(tsk)
## INFO  [12:33:24.372] [bbotk] Starting to optimize 1 parameter(s) with '<TunerGridSearch>
## INFO  [12:33:24.376] [bbotk] Evaluating 1 configuration(s)
## INFO  [12:33:24.397] [mlr3] Running benchmark with 5 resampling iterations
## INFO  [12:33:24.404] [mlr3] Applying learner 'classif.svm' on task 'dat' (iter 1/5)
## INFO  [12:33:24.450] [mlr3] Applying learner 'classif.svm' on task 'dat' (iter 2/5)
## INFO  [12:33:24.498] [mlr3] Applying learner 'classif.svm' on task 'dat' (iter 3/5)
## INFO  [12:33:24.681] [mlr3] Applying learner 'classif.svm' on task 'dat' (iter 4/5)
## INFO  [12:33:24.732] [mlr3] Applying learner 'classif.svm' on task 'dat' (iter 5/5)
## INFO  [12:33:24.784] [mlr3] Finished benchmark
## INFO  [12:33:24.856] [bbotk] Result of batch 1:
## INFO  [12:33:24.862] [bbotk]  cost classif.ce warnings errors runtime_learners
## INFO  [12:33:24.862] [bbotk]  1000       0.48        0      0             0.22
## INFO  [12:33:24.862] [bbotk]                                        uhash
## INFO  [12:33:24.862] [bbotk]  7bb41f05-c831-4aed-857a-694691346c08
## INFO  [12:33:24.866] [bbotk] Evaluating 1 configuration(s)
## INFO  [12:33:24.888] [mlr3] Running benchmark with 5 resampling iterations
## INFO  [12:33:24.900] [mlr3] Applying learner 'classif.svm' on task 'dat' (iter 1/5)
## INFO  [12:33:24.935] [mlr3] Applying learner 'classif.svm' on task 'dat' (iter 2/5)
## INFO  [12:33:24.985] [mlr3] Applying learner 'classif.svm' on task 'dat' (iter 3/5)
```

Hyperparameter Tuning in `mlr3`

3. Perform the hyperparameter tuning
   3.2. Compare the CV results (i.e., performance) for each potential value of C:

```
mdl_cv$archive %>%
  as.data.table() %>%
  select(cost, classif.ce) %>%
  arrange(as.numeric(cost))
##      cost classif.ce
##    <char>      <num>
## 1:     10       0.39
## 2:     50       0.47
## 3:    100       0.46
## 4:    500       0.46
## 5:   1000       0.48

mdl_cv$tuning_result
##      cost learner_param_vals  x_domain classif.ce
##    <char>             <list>    <list>      <num>
## 1:     10          <list[2]> <list[1]>       0.39
```

Hyperparameter Tuning in mlr3

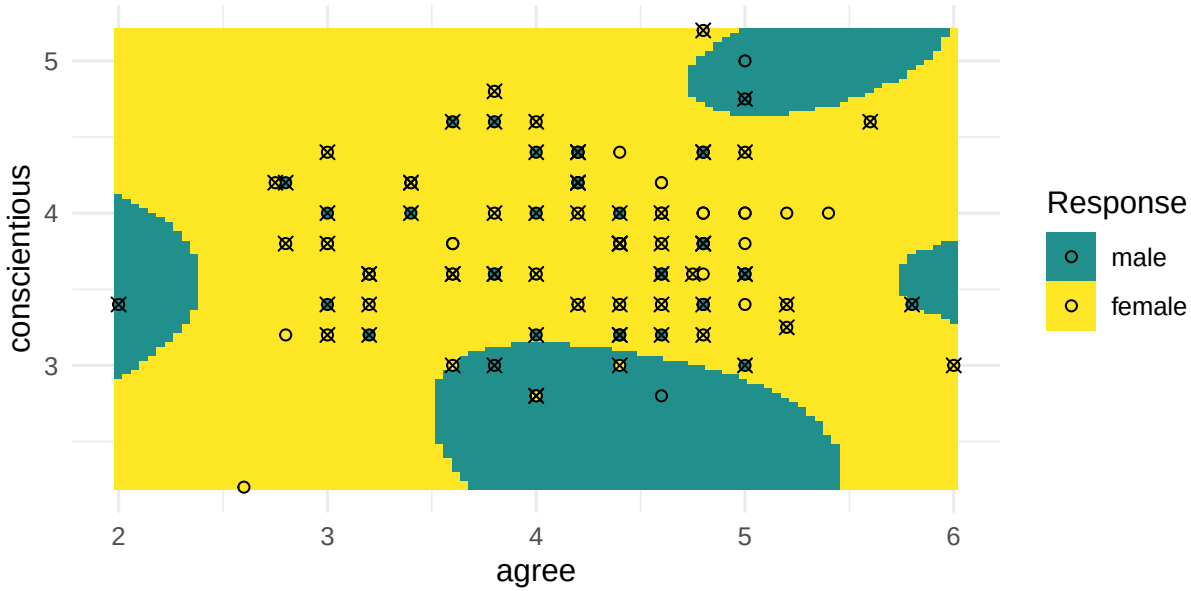
- Final model with optimal expected out-of-sample performance (i.e., best hyperparameter setting):

```
summary(mdl_cv$learner$model)
##
## Call:
## svm.default(x = data, y = task$truth(), type = "C-classification",
##     cost = 10, probability = (self$predict_type == "prob"))
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  10
##
## Number of Support Vectors:  79
##
##  ( 34 45 )
##
##
## Number of Classes:  2
##
## Levels:
##  male female
```

Hyperparameter Tuning in mlr3

- Final model with optimal expected out-of-sample performance (i.e., best hyperparameter setting):

```
autoplot(mdl_cv$learner, task = tsk) + scale_fill_viridis_d(begin = .5) +
  geom_point(data = tsk$data()[mdl$model$index,], shape = 4, size = 2)
## Scale for fill is already present.
## Adding another scale for fill, which will replace the existing scale.
```

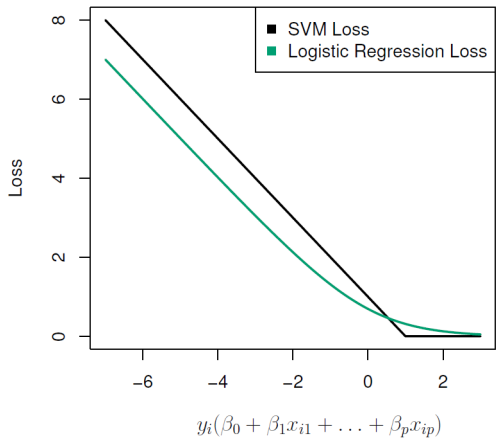


Excurse: Relationship between SVM & Logistic Regression

- Both, SVM and logistic regression can be rewritten to minimize the so-called loss function
  - Loss: Quantifies the extent to which the model, parametrized by $\beta = (\beta_0, \beta_1, \ldots, \beta_p)$, fits the data (X, y)

$$\underset{\beta_0, \beta_1, \ldots, \beta_p}{\text{minimize}} \{L(X, y, \beta) + \lambda P(\beta)\} \tag{5}$$

  - Overall, the two loss functions have quite similar shape and thus behavior:



(James et al., 2021, Figure 9.9)

Excurse: The Optimization Problem

- Remember:
  - Linear decision boundary (or "hyperplane"): $\beta_0 + \sum_{j=1}^{p} \beta_j x_j = 0$
  - Predicted class of instance i: $\hat{y}_i = \hat{\beta}_0 + \sum_{j=1}^{p} \hat{\beta}_j x_{ij}$
- Condition for correct classification of **all** instances in the data (i.e., "hard" margin):

$$y_i \hat{y}_i \geq 1 \, \forall i = 1, \ldots, N \tag{6}$$

  - $y_i = 1$ and $\hat{y}_i = 1 \Rightarrow y_i \hat{y}_i = 1$
  - $y_i = -1$ and $\hat{y}_i = -1 \Rightarrow y_i \hat{y}_i = 1$
- In general, correct classification can be written as:

$$y_i \left(\beta_0 + \sum_{j=1}^{p} \beta_j x_j \right) > 0 \tag{7}$$

  - If this condition is true, there are only the two possible cases from above (at least for "hard" margins)

Excurse: The Optimization Problem

- Maximizing the "soft" margin is equivalent to

$$\underset{\beta_0,\beta_1,\ldots,\beta_p,\xi}{\text{minimize}} \frac{1}{2}\sum_{j=1}^{p}\beta_j^2 + \frac{C}{N}\sum_{i=1}^{N}\xi_i \tag{8}$$

s.t.

$$y_i\left(\beta_0 + \sum_{j=1}^{p}\beta_j x_{ij}\right) \geq 1 - \xi_i \,\forall i = 1,\ldots,N \tag{9}$$

$$\xi_i \geq 0 \,\forall i = 1,\ldots,N \tag{10}$$

- "Slack variables" ξ: Allow instances to be on the wrong side of the margin or the hyperplane
  - If $\xi_i = 0$: Instance i is correctly classified
  - Else if $0 < \xi_i \leq 1$: Instance i is inside the margin but still on the correct side of the hyperplane (i.e., correctly classified)
  - Else if $\xi_i > 1$: Instance i is misclassified
- $C = 0$: No budget for violations to the margin $\Rightarrow \xi_1 = \ldots = \xi_N = 0$

Hands-on Practical Tutorial

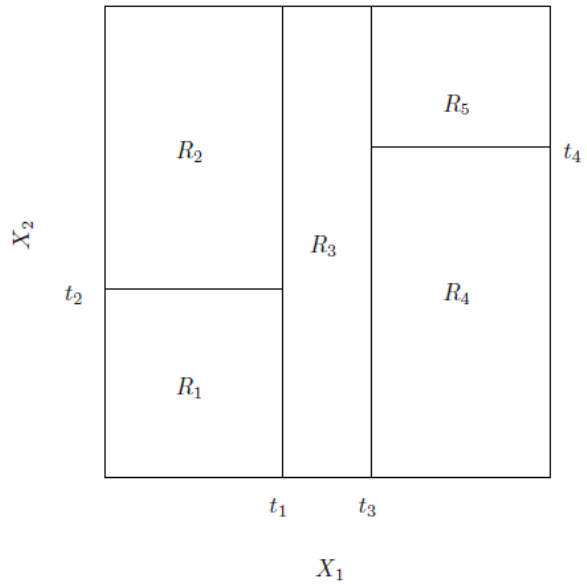
- Now it's your turn:
  - Go to ILIAS
  - Navigate to the "Tutorials" folder
  - Download the "Module 2" folder
  - Work on the tutorial "module2-SVM"

Classification Trees

Classification Trees

- Classification trees (CTs): Recursively partition the feature space into a set of rectangular areas using `if` statements
- Prediction: A class y_l is assigned to each partition $\mathbf{R}_l$, and new objects receive the class assigned to their regions:

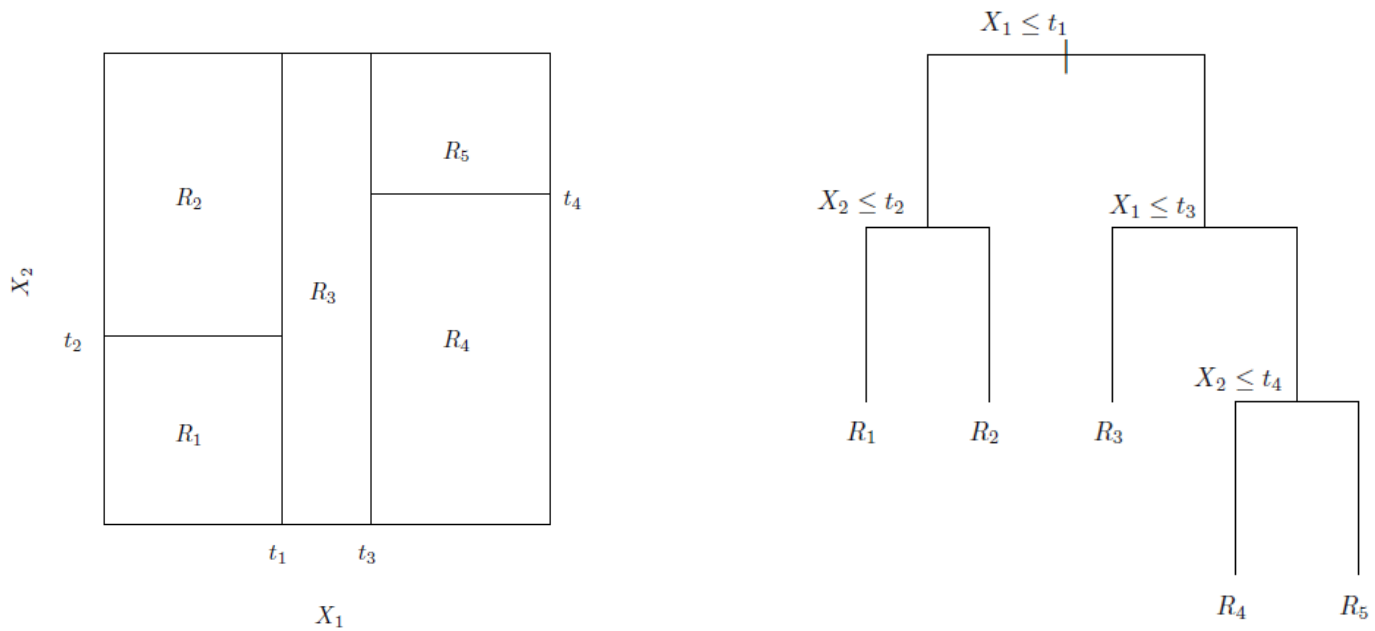
$$\text{If } X_{new} \in \mathbf{R}_l, \text{then } \hat{y}_{new} = y_l \tag{11}$$



(James et al., 2021, Figure 8.3)

Classification Trees

- The rectangular partitioning can alternatively be represented as a (binary decision) tree:



(James et al., 2021, Figure 8.3)

Classification Trees in `mlr3`

- Participants in Logg et al. (2019, Experiment 3) had the choice between an algorithm and a human (other participant vs. self, depending on condition) to determine their performance-dependent bonus payment
  - "Algorithm aversion": General preference for human over algorithm (Mahmud et al., 2022)

```
dat <- haven::read_sav('https://osf.io/download/kt47s')
```

```
tail(dat)
## # A tibble: 6 x 6
##   choice      age SexM1F2 condition  confidence_alg accuracy_alg
##   <fct>     <dbl> <fct>   <fct>                <dbl>        <dbl>
## 1 algorithm    30 1       self_human               4         0.04
## 2 algorithm    37 2       self_human               3         0
## 3 algorithm    32 2       self_human               4         0.2
## 4 human        28 1       self_human               2         0.2
## 5 human        25 2       self_human               2         0.02
## 6 algorithm    31 2       self_human               4         0.2
```

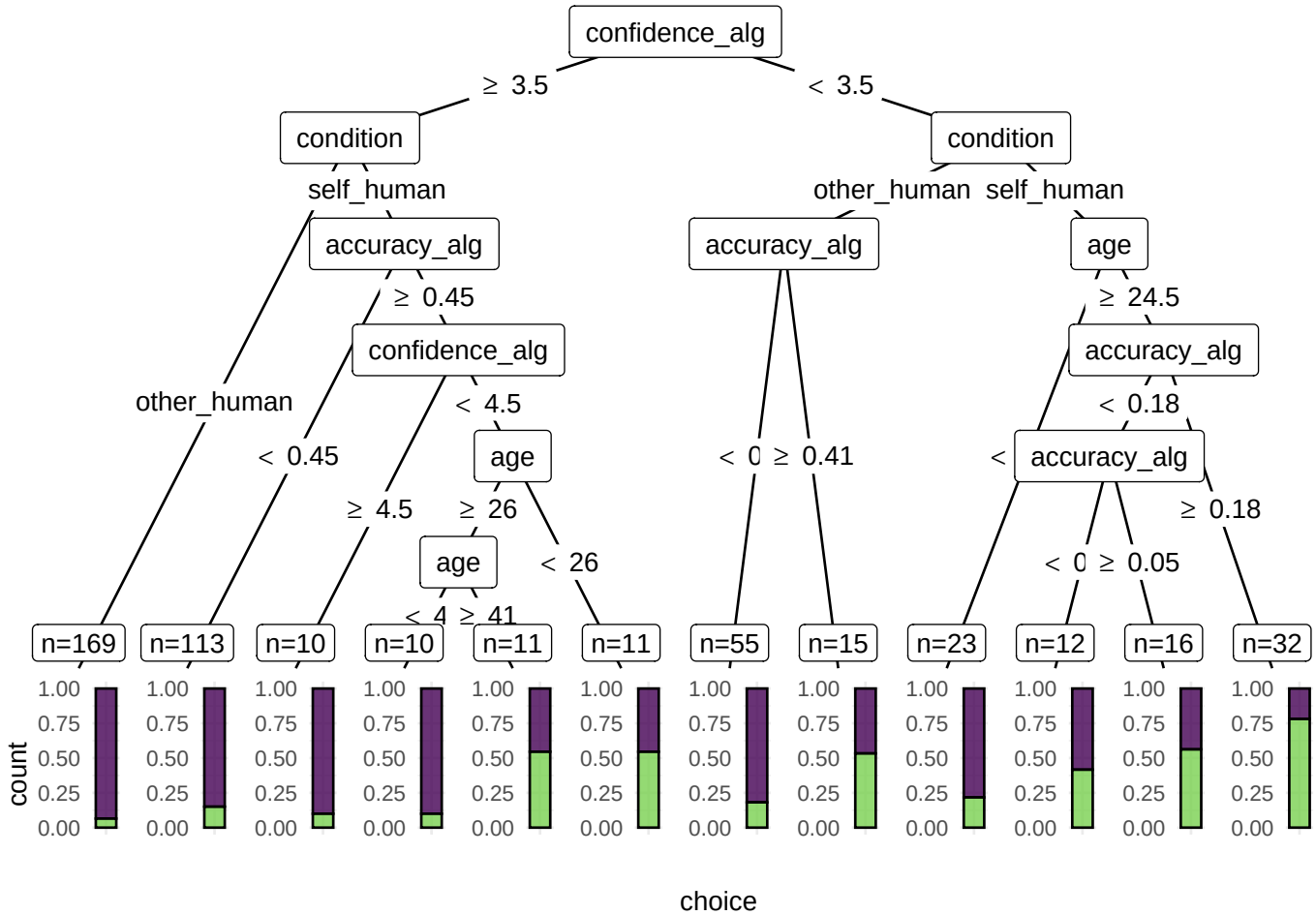
Classification Trees in mlr3

```
tsk = as_task_classif(choice ~ ., data = dat, positive = 'algorithm')
mdl = lrn("classif.rpart", keep_model = TRUE, cp = 0)
mdl$train(tsk)
mdl$model
## n= 477
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 477 104 algorithm (0.7819706 0.2180294)
##    2) confidence_alg>=3.5 324  42 algorithm (0.8703704 0.1296296)
##      4) condition=other_human 169  11 algorithm (0.9349112 0.0650888) *
##      5) condition=self_human 155  31 algorithm (0.8000000 0.2000000)
##       10) accuracy_alg< 0.45 113  17 algorithm (0.8495575 0.1504425) *
##       11) accuracy_alg>=0.45 42  14 algorithm (0.6666667 0.3333333)
##         22) confidence_alg>=4.5 10   1 algorithm (0.9000000 0.1000000) *
##         23) confidence_alg< 4.5 32  13 algorithm (0.5937500 0.4062500)
##           46) age>=26 21   7 algorithm (0.6666667 0.3333333)
##             92) age< 41 10   1 algorithm (0.9000000 0.1000000) *
##             93) age>=41 11   5 human (0.4545455 0.5454545) *
##           47) age< 26 11   5 human (0.4545455 0.5454545) *
##    3) confidence_alg< 3.5 153  62 algorithm (0.5947712 0.4052288)
##      6) condition=other_human 70  18 algorithm (0.7428571 0.2571429)
##       12) accuracy_alg< 0.41 55  10 algorithm (0.8181818 0.1818182) *
##       13) accuracy_alg>=0.41 15   7 human (0.4666667 0.5333333) *
##      7) condition=self_human 83  39 human (0.4698795 0.5301205)
##       14) age< 24.5 23   5 algorithm (0.7826087 0.2173913) *
```

Classification Trees in `mlr3`

- The complex partitioning can be represented as a (binary decision) tree:

```
autoplot(mdl, type = "ggparty")
```



Classification Trees in `mlr3`

- Class assignment: Majority class in a leaf/terminal node (i.e., partition)

```
set.seed(42)
pred <- mdl$predict(tsk)
pred$confusion
##             truth
## response     algorithm human
##   algorithm       342    50
##   human            31    54
```

- Note: If you re-run the `predict()` method for classification trees, you may get slightly different results, e.g., due to ties
  - Ties: Same amount of training instances per class in a terminal node

Classification Trees in mlr3

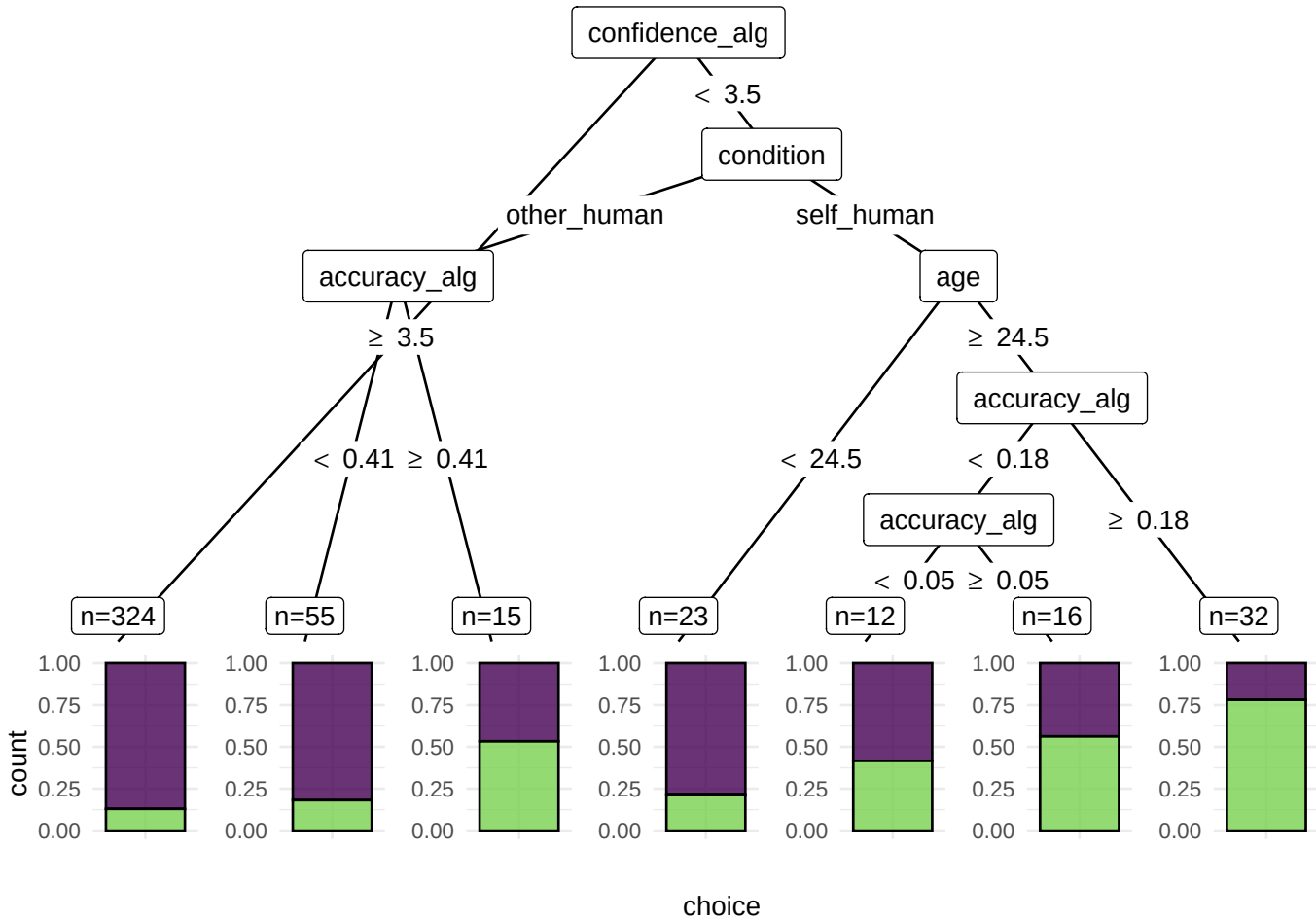
- Many partitions in our example lead to the same prediction
  - In other words, they are redundant/uninformative
  - Solution: "Pruning" the tree by increasing the penalty on complexity cp

```
mdl = lrn("classif.rpart", keep_model = TRUE, cp = 0.005)
mdl$train(tsk)
mdl$model
## n= 477
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 477 104 algorithm (0.781971 0.218029)
##    2) confidence_alg>=3.5 324  42 algorithm (0.870370 0.129630) *
##    3) confidence_alg< 3.5 153  62 algorithm (0.594771 0.405229)
##      6) condition=other_human 70  18 algorithm (0.742857 0.257143)
##       12) accuracy_alg< 0.41 55  10 algorithm (0.818182 0.181818) *
##       13) accuracy_alg>=0.41 15   7 human (0.466667 0.533333) *
##      7) condition=self_human 83  39 human (0.469880 0.530120)
##       14) age< 24.5 23   5 algorithm (0.782609 0.217391) *
##       15) age>=24.5 60  21 human (0.350000 0.650000)
##         30) accuracy_alg< 0.18 28  14 algorithm (0.500000 0.500000)
##           60) accuracy_alg< 0.05 12   5 algorithm (0.583333 0.416667) *
##           61) accuracy_alg>=0.05 16   7 human (0.437500 0.562500) *
##         31) accuracy_alg>=0.18 32   7 human (0.218750 0.781250) *
```

Classification Trees in `mlr3`

- Pruned tree:

```
autoplot(mdl, type = "ggparty")
```



Classification Trees in mlr3

- Class assignment:

```
set.seed(42)
pred <- mdl$predict(tsk)
pred$confusion
##            truth
## response    algorithm human
##   algorithm       352    62
##   human            21    42
```

Hyperparameter Tuning in mlr3

- Better than pruning the tree manually to remove unnecessary partitions: Using `auto_tuner()`, tune the hyperparameter `cp` by means of (e.g., 5-fold) CV to find the optimal value

```
cp_cv <- seq(0, 0.05, 0.01)

mdl_cv = auto_tuner(
  learner = lrn("classif.rpart", keep_model = TRUE, cp = to_tune(levels = cp_cv)),
  resampling = rsmp("cv", folds = 5),
  measure = msr("classif.ce"),
  tuner = tnr("grid_search"),
  terminator = trm("none")
)

set.seed(42)
mdl_cv$train(tsk)
## INFO  [12:33:44.110] [bbotk] Starting to optimize 1 parameter(s) with '<TunerGridSearch>
## INFO  [12:33:44.124] [bbotk] Evaluating 1 configuration(s)
## INFO  [12:33:44.197] [mlr3] Running benchmark with 5 resampling iterations
## INFO  [12:33:44.223] [mlr3] Applying learner 'classif.rpart' on task 'dat' (iter 1/5)
## INFO  [12:33:44.293] [mlr3] Applying learner 'classif.rpart' on task 'dat' (iter 2/5)
## INFO  [12:33:44.363] [mlr3] Applying learner 'classif.rpart' on task 'dat' (iter 3/5)
## INFO  [12:33:44.439] [mlr3] Applying learner 'classif.rpart' on task 'dat' (iter 4/5)
## INFO  [12:33:44.517] [mlr3] Applying learner 'classif.rpart' on task 'dat' (iter 5/5)
## INFO  [12:33:44.606] [mlr3] Finished benchmark
```

Hyperparameter Tuning in mlr3

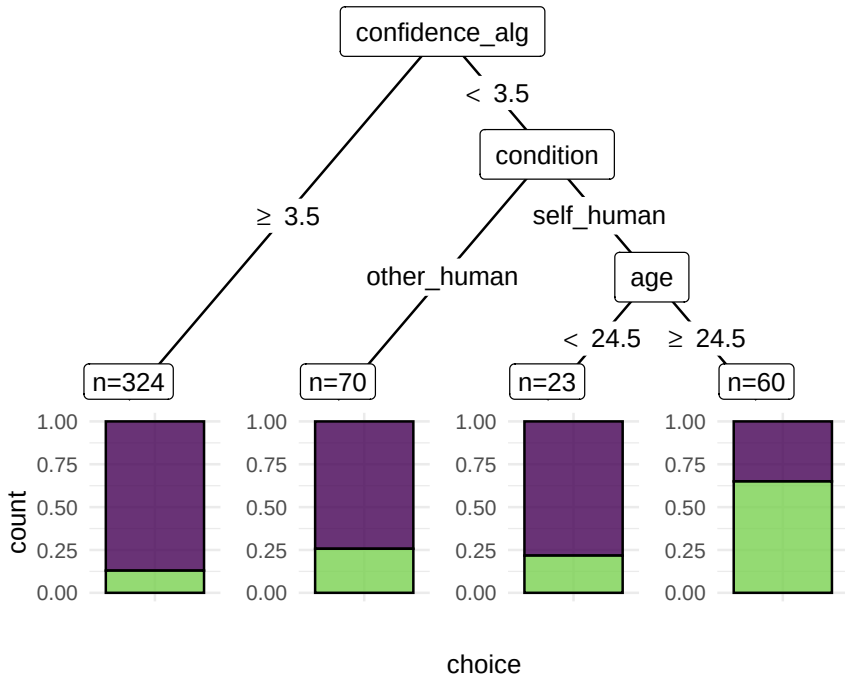
- Ideally, the best value for the hyperparamter lies "in the middle" of the grid to be searched
  - Why? – If it lies at the borders, there might be a better model for which the hyperparameter is smaller (larger) than the minimum (maximum) value tested

```
mdl_cv$archive %>%
  as.data.table() %>%
  select(cp, classif.ce) %>%
  arrange(as.numeric(cp))
##        cp classif.ce
##    <char>      <num>
## 1:      0   0.230702
## 2:   0.01   0.209649
## 3:   0.02   0.209649
## 4:   0.03   0.201272
## 5:   0.04   0.197061
## 6:   0.05   0.217895

mdl_cv$tuning_result
##        cp learner_param_vals  x_domain classif.ce
##    <char>             <list>    <list>      <num>
## 1:   0.04          <list[3]> <list[1]>   0.197061
```

Hyperparameter Tuning in mlr3

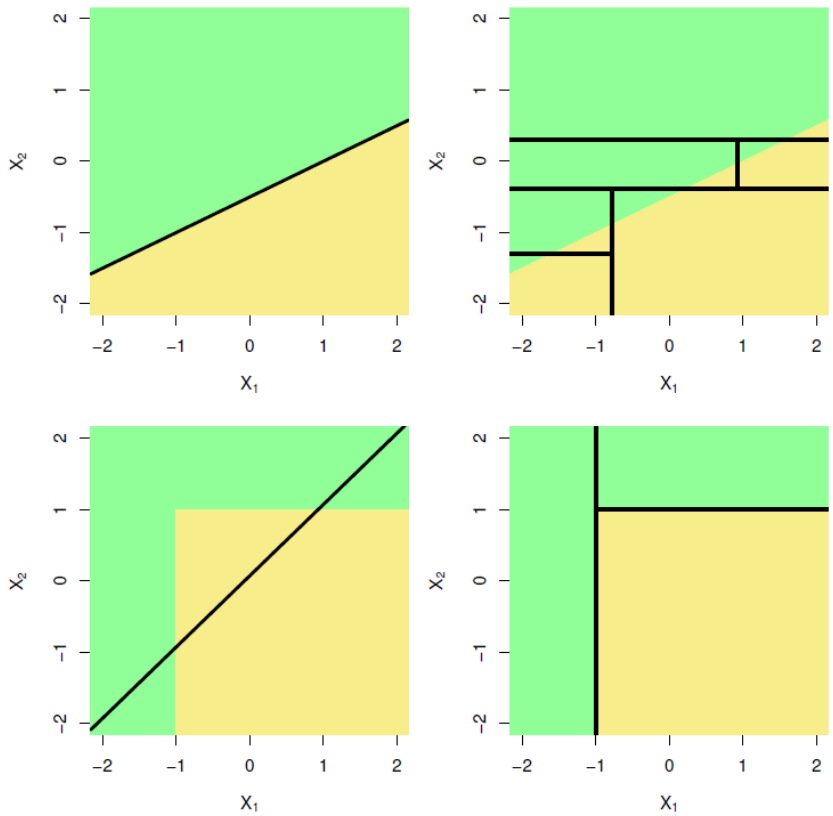
- CV-pruned tree:

```
autoplot(mdl_cv$learner, type = "ggparty")
```



Excurse: Classification Tree vs. SVM

- Tree-based classifiers are ideal for nonlinear decision boundaries (bottom), but very bad for linear decision boundaries (top):



(James et al., 2021, Figure 8.7)

Excurse: Splitting and Stopping

- Splitting rule: Choose the feature j and its threshold $\bar{x}$ to maximize the gain in purity
  - Branches: $x_j \leq \bar{x}$ & $x_j > \bar{x}$
  - Aim: Decrease the impurity of the parent node, e.g., Gini index $= 2\pi_1\pi_{-1}$
    - π_1: proportion of instances in class 1
    - π_{-1}: proportion of instances in class -1
  - A node is pure if it contains instances from one class only: $\pi_1\pi_{-1} = 0$
- Stopping criteria: Number of instances in each node should be above a minimum (e.g., 10)
  - Branching improves the purity of the children nodes, but decreases the amount of instances in each children node
    - Going too deep $\Rightarrow$ Overfitting!

Hands-on Practical Tutorial

- Your turn:
  - Work on tasks 1–5 in tutorial “module2-tree”

Random Forests

Random Forests

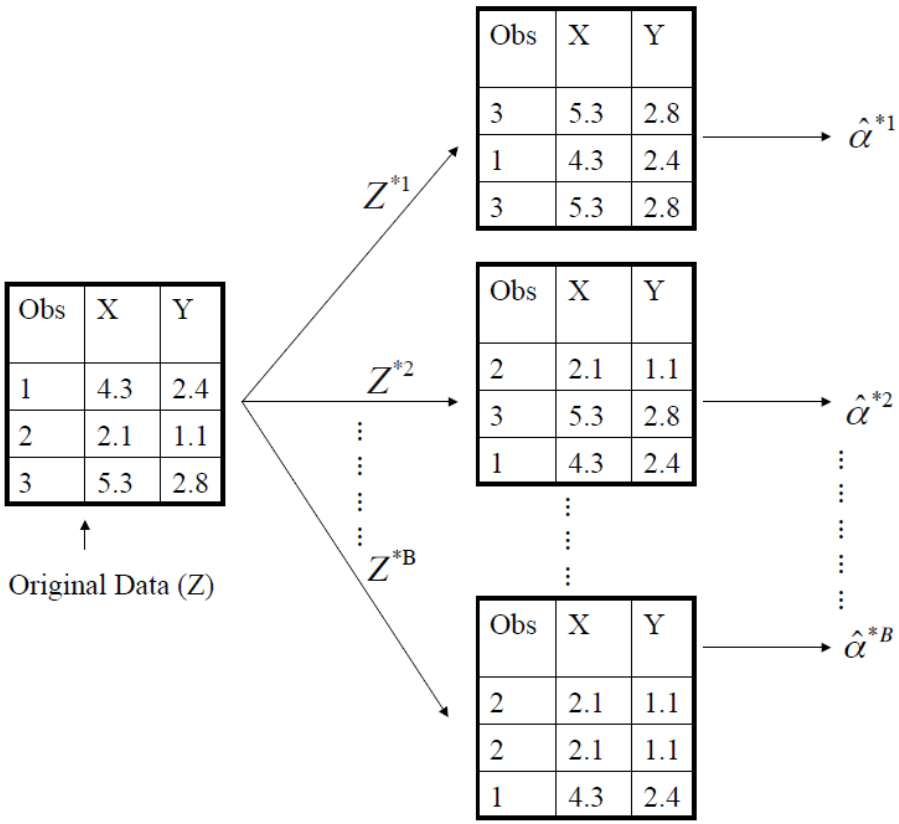
- Trees can easily handle both numerical and categorical variables
- Trees can easily handle multiclass problems
- Trees can easily handle imbalanced datasets
- But: Trees are highly sensitive to small changes in the training data, especially if they are very deep (i.e., more complex)
  - Solution: Building a collection of deep trees and classify a new instance X_{new} according to which class is assigned to it by the majority of trees
  - Bias-variance trade-off: Deep trees have low bias, and variance reduction is achieved by aggregating the predictions of many deep trees

Random Forests

- We build a collection of trees using a different training sample for each individual tree
  - E.g., boostrapping: Sampling from the training data with replacement to build one specific tree
  - Intuition: Simulates drawing multiple samples from the population of interest
- Additionally, each cut only considers a random sample of features to split the data
  - Goal: Reducing the similarity of trees in the forest

Random Forests

- Bootstrapping:



(James et al., 2021, Figure 5.11)

Random Forests in mlr3

```
set.seed(42)
mdl = lrn("classif.ranger", importance = "permutation")
mdl$train(tsk)
mdl$model
## Ranger result
##
## Call:
##  ranger::ranger(dependent.variable.name = task$target_names, data = task$data(),      pr
##
## Type:                             Classification
## Number of trees:                  500
## Sample size:                      477
## Number of independent variables:  5
## Mtry:                             2
## Target node size:                 1
## Variable importance mode:         permutation
## Splitrule:                        gini
## OOB prediction error:             21.80 %
```

Random Forests in mlr3

- Multiple important hyperparameters:
  - `num.trees`: Number of trees in the forest
  - `mtry`: Number of features considered for each split

```
num.trees_cv <- c(100, 500, 1000)
mtry_cv <- seq(2, 5)

mdl_cv = auto_tuner(
  learner = lrn("classif.ranger", importance = "permutation",
                num.trees = to_tune(levels = num.trees_cv),
                mtry = to_tune(levels = mtry_cv)),
  resampling = rsmp("cv", folds = 5),
  measure = msr("classif.ce"),
  tuner = tnr("grid_search"),
  terminator = trm("none")
)

set.seed(42)
mdl_cv$train(tsk)
## INFO  [12:33:50.619] [bbotk] Starting to optimize 2 parameter(s) with '<TunerGridSearch>
## INFO  [12:33:50.625] [bbotk] Evaluating 1 configuration(s)
## INFO  [12:33:50.642] [mlr3] Running benchmark with 5 resampling iterations
## INFO  [12:33:50.651] [mlr3] Applying learner 'classif.ranger' on task 'dat' (iter 1/5)
## INFO  [12:33:50.872] [mlr3] Applying learner 'classif.ranger' on task 'dat' (iter 2/5)
## INFO  [12:33:51.082] [mlr3] Applying learner 'classif.ranger' on task 'dat' (iter 3/5)
## INFO  [12:33:51.290] [mlr3] Applying learner 'classif.ranger' on task 'dat' (iter 4/5)
```

Random Forests in mlr3

- Grid search: Can also be used to test multiple combinations of hyperparameters

```
mdl_cv$archive %>%
  as.data.table() %>%
  select(num.trees, mtry, classif.ce) %>%
  arrange(as.numeric(num.trees), as.numeric(mtry))
##     num.trees   mtry classif.ce
##         <char> <char>      <num>
##  1:       100      2   0.220373
##  2:       100      3   0.232982
##  3:       100      4   0.230877
##  4:       100      5   0.226601
##  5:       500      2   0.211996
##  6:       500      3   0.228816
##  7:       500      4   0.235022
##  8:       500      5   0.230855
##  9:      1000      2   0.214079
## 10:      1000      3   0.224605
## 11:      1000      4   0.241360
## 12:      1000      5   0.232939

mdl_cv$tuning_result
##    num.trees   mtry learner_param_vals  x_domain classif.ce
##        <char> <char>             <list>    <list>      <num>
## 1:       500      2          <list[4]> <list[2]>   0.211996
```

Random Forests in mlr3

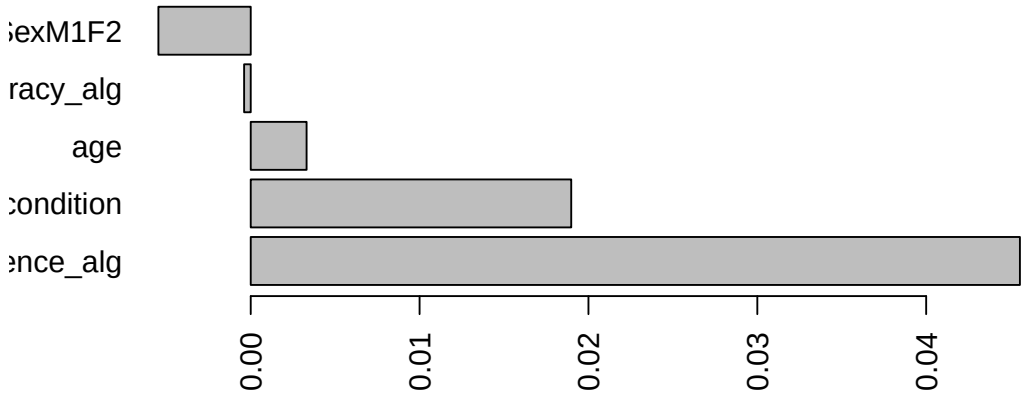
- Final model:

```
mdl_cv$learner$model
## Ranger result
##
## Call:
##  ranger::ranger(dependent.variable.name = task$target_names, data = task$data(),      pr
##
## Type:                             Classification
## Number of trees:                  500
## Sample size:                      477
## Number of independent variables:  5
## Mtry:                             2
## Target node size:                 1
## Variable importance mode:         permutation
## Splitrule:                        gini
## OOB prediction error:             22.01 %
```

Random Forests in `mlr3`

- Nice by-product: Measures for the importance of each feature for the classification task

```
barplot(mdl_cv$importance(), horiz = T, las = 2)
```



- Note: The importance of a feature can also be negative, especially for noisy features
  - We would expect improved predictive performance if these "bad" features were actually removed from the model

Random Forests in mlr3

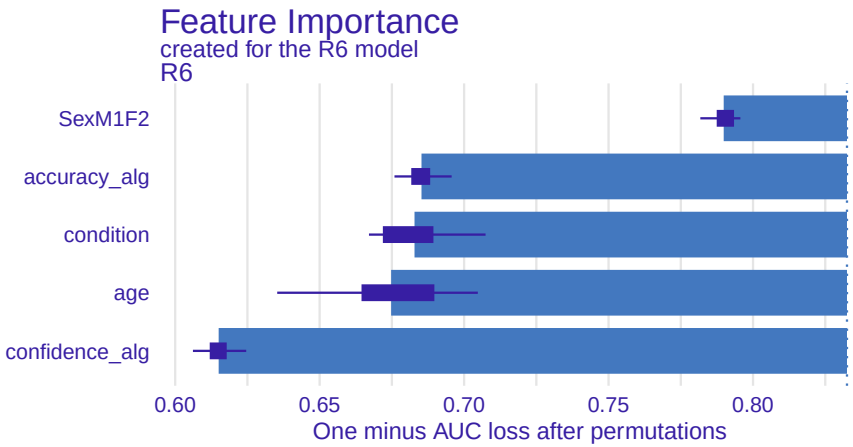
- Permutation importance: The importance of feature x_j is defined as the change in accuracy by randomly reshuffling the values of x_j
  - Note: This is a "model-agnostic" measure of feature importance, which means that it can be applied with any trained predictive model and is not limited to RFs
- We can visualize the variability of feature importance across permutations by adding boxplots from the DALEXtra package:

```
set.seed(42)

library(DALEXtra)
expl <- explain_mlr3(mdl_cv$learner,
                     data = dat %>% select(-choice),
                     y = ifelse(dat$choice == "algorithm", 1, 0),
                     predict_function = function(mdl, newdat) {
                       mdl$predict_newdata(newdata = newdat)$response
                     },
                     verbose = FALSE
                     )
varimp <- model_parts(expl, B = 3)
```

Random Forests in `mlr3`

- The `DALEXtra` package offers tools for interpretable ML, that is, making sense of the prediction behavior of black-box models
  - By default, it calculates a AUC-based importance measure for RFs:

```
plot(varimp)
```



- Note: The ordering of `age` and `condition` is reversed
  - Feature importance scores provide a relative measure of the importance of each feature in the model $\Rightarrow$ Absolute score values are not very informative

Hands-on Practical Tutorial

- Your turn:
  - Finish the tutorial "module2-tree"

Summary

Summary

- Supervised Learning is a main task in DDDM
  - E.g., classification: The target to predict is class membership
  - We have discussed regularized regression, Support Vector Machines and tree-based (incl. ensamble) algorithms
- Advanced ML is much more intuitive than classical statistics
  - No distributional assumptions, p-values, ...
  - But: It's not a magical black box, but a set of well-motivated mathematical principles for modelling data and predicting future instances
- Challenges:
  - For classification: Inseparable classes, class imbalance, ...
  - In general: Hyperparameter selection, bias-variance trade-off, curse of dimensionality, ...

Homework

- Finish/Revisit the programming tutorials
- Readings for next week, in particular:
  - Wulff, D. U., Kieslich, P. J., Henninger, F., Haslbeck, J. M. B., & Schulte-Mecklenbeck, M. (2021). *Movement tracking of psychological processes: A tutorial using mousetrap*. PsyArXiv. https://doi.org/10.31234/osf.io/v685r
    - Introduction (pp. 1–3) and movement trajectory clustering (pp. 14–16)