Introduction to Machine Learning

Module 2: Supervised Learning

Tobias Rebholz

University of Tübingen

Fall 2024, SMiP Workshop

Regularized Regression

Refresher: Multiple Linear Regression

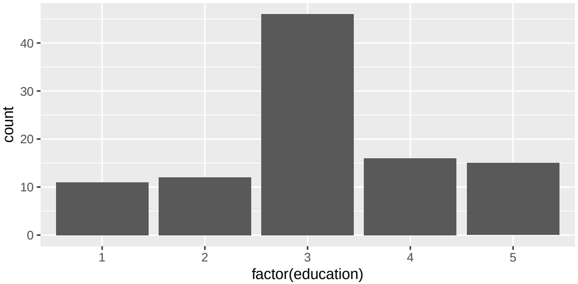
- For each instance i in a population, we have:
  - A vector of features, $X_i = (x_{i1}, x_{i2}, \dots, x_{ip})$
  - Continuous target, $y_i \in \mathbb{R}$
- Goal: Predict the target for new instances of which we know the values of the features but not the value of the target:

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

  - 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

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



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

Refresher: Multiple Linear Regression

```r
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
```

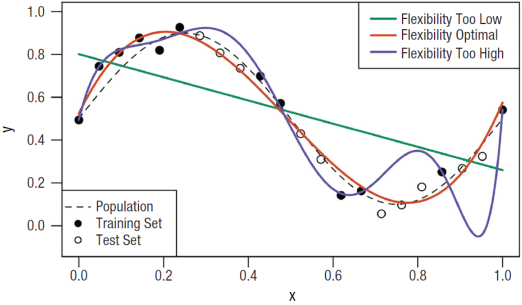
(Automatic) Variable Selection

If we have many features (e.g., personality traits) that may potentially explain the target (e.g., education), 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!

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 are low
    - E.g., 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

- Least Absolute Shrinkage and Selection Operator (LASSO): Regression models that penalize the absolute size of the estimated coefficients
  - Relies upon the linear model but uses an alternative fitting procedure for estimating the coefficients $\beta_0, \beta_1, \ldots, \beta_p$:

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

  - Tends to use a lower number of features, effectively **selecting the most important ones**
- λ is called the "regularization", "tuning" or "hyper-" parameter
  - Controls the trade-off between:
    - minimizing the error on the training data (i.e., fitting the data well; first term)
    - 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

- Penalization/Regularization shrinks some of the regression coefficients towards zero $\Rightarrow$ Original interpretability is lost:
  - Biased coefficients: Size no longer corresponds to the expected change in the response variable for a one-unit change in the predictor
  - Shrinkage is uneven: Depends on the relative importance of features and their correlation with other features
    - Thus, comparisons between coefficients may also be misleading

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(., 2)
## 8 x 1 sparse Matrix of class "dgCMatrix"
##                  s0
## (Intercept)    2.71
## age            0.01
## agree         -0.03
## conscientious  0.15
## extra          .
## gender         0.03
## neuro         -0.13
## open           .
```

Regularized Regression in mlr3

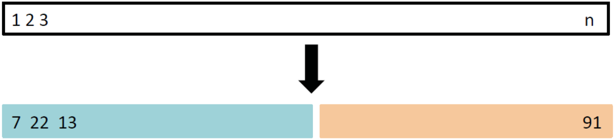
- Instead of arbitrarily choosing $\lambda = 0.1$, we can (rather: should!) try different values:

```
mdl = lrn("regr.glmnet", nlambda = 10)
mdl$train(tsk)

coef(mdl$model) %>% round(., 2)
## 8 x 9 sparse Matrix of class "dgCMatrix"
##                 s0    s1    s2    s3    s4    s5    s6    s7    s8
## (Intercept)   3.12  2.71  2.60  2.48  2.44  2.43  2.42  2.42  2.42
## age              .  0.01  0.02  0.02  0.02  0.03  0.03  0.03  0.03
## agree            . -0.03 -0.24 -0.32 -0.34 -0.35 -0.36 -0.36 -0.36
## conscientious .     0.16  0.33  0.39  0.42  0.43  0.43  0.43  0.44
## extra            .     .     .  0.03  0.05  0.06  0.07  0.07  0.07
## gender           .  0.04  0.25  0.32  0.34  0.35  0.36  0.36  0.36
## neuro            . -0.14 -0.23 -0.27 -0.28 -0.29 -0.29 -0.29 -0.29
## open             .     .     .     . -0.02 -0.03 -0.03 -0.03 -0.03

lambdas <- setNames(mdl$model$lambda, colnames(coef(mdl$model)))
lambdas %>% round(., 4)
##     s0     s1     s2     s3     s4     s5     s6     s7     s8
## 0.2714 0.0975 0.0351 0.0126 0.0045 0.0016 0.0006 0.0002 0.0001
```

Validation Set Approach

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

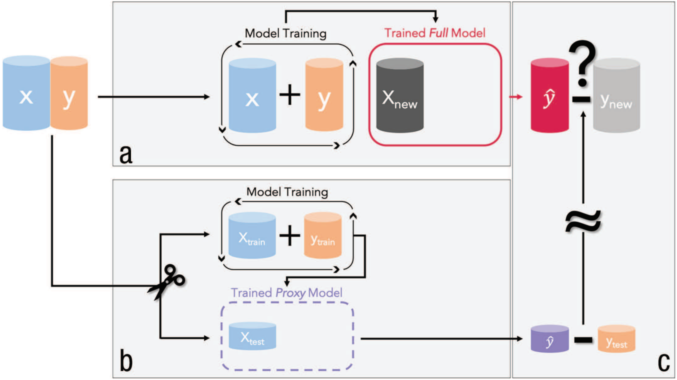
1. Dataset is split into training and validation sets
2. Model is trained on training set; performance is evaluated on validation set



(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 a model's real-world testing performance



(Pargent et al., 2023, Figure 2)

Validation Set Approach in mlr3

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

```
set.seed(42)
row_ids <- partition(tsk)
row_ids
## $train
##  [1]   2   3   4   5   6   8   9  10  12  15  16  18  20  21  22  24  25  26  27
## [20]  28  29  30  33  34  35  36  37  38  39  40  41  42  43  44  45  47  49  50
## [39]  51  52  54  55  58  61  63  65  66  67  68  71  74  76  79  80  81  83  84
## [58]  87  88  89  91  92  93  94  95  96 100
##
## $test
##  [1]  1  7 11 13 14 17 19 23 31 32 46 48 53 56 57 59 60 62 64 69 70 72 73 75 77
## [26] 78 82 85 86 90 97 98 99
##
## $validation
## integer(0)
```

Validation Set Approach in mlr3

2. Building the model with the training data and predicting the validation data's target
   - Note the issue of treating the categorical `eduction` variable as continuous target: Predicting nonexistent education levels
   - **Problem:** `mlr3`'s predict() does not (yet) support multiple lambda values (see https://github.com/mlr-org/mlr3learners/issues/10)

```
mdl = lrn("regr.glmnet", nlambda = 10)
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').
lambdas <- setNames(mdl$model$lambda, colnames(coef(mdl$model)))
lambdas %>% round(., 4)
##     s0     s1     s2     s3     s4     s5     s6     s7     s8
## 0.3750 0.1348 0.0484 0.0174 0.0063 0.0022 0.0008 0.0003 0.0001

tail(cbind('true' = dat[row_ids$test,]$education, 'pred' = round(pred$response, 2)))
##        true pred
## [28,]     2 2.98
## [29,]     2 2.59
## [30,]     3 2.80
## [31,]     3 4.07
```

Validation Set Approach in mlr3

2. cont'd
   - **Solution:** We can use the native package glmnet's predict()

```
# 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, round(pred, 2)))
##       true  s0   s1   s2   s3   s4   s5   s6   s7   s8
## [28,]    2 3.1 3.23 3.09 3.00 2.97 2.96 2.96 2.96 2.96
## [29,]    2 3.1 2.89 2.74 2.62 2.58 2.57 2.56 2.56 2.56
## [30,]    3 3.1 2.87 2.83 2.81 2.80 2.80 2.80 2.80 2.80
## [31,]    3 3.1 3.78 4.10 4.08 4.06 4.06 4.05 4.05 4.05
## [32,]    3 3.1 3.19 3.19 3.09 3.05 3.04 3.03 3.03 3.03
## [33,]    3 3.1 2.69 2.61 2.61 2.62 2.62 2.62 2.62 2.62
```

Validation Set Approach: Hyperparameter Selection

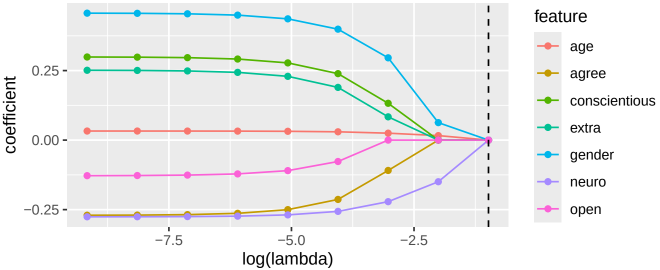
3. Minimizing the out-of-sample MSE

```
MSE_pred <- colMeans((pred - dat[row_ids$test,]$education)^2)
MSE_pred %>% round(., 4)
##     s0     s1     s2     s3     s4     s5     s6     s7     s8
## 1.1611 1.1920 1.1788 1.1746 1.1800 1.1828 1.1839 1.1843 1.1845

# Which value of the hyperparameter (lambda) yields the smallest out-of-sample MSE?
idx_lambda_best <- which.min(MSE_pred)
idx_lambda_best
## s0
##  1

lambda_best <- lambdas[idx_lambda_best]
lambda_best %>% round(., 4)
##    s0
## 0.375

# Choosing the model with the best out-of-sample prediction performance:
coef(mdl$model)[,idx_lambda_best] %>% round(., 2)
##   (Intercept)            age          agree conscientious         extra
##           3.1            0.0            0.0           0.0           0.0
##        gender          neuro           open
##           0.0            0.0            0.0
```

Validation Set Approach: Hyperparameter Selection

- Trace plot: Visualizes the model selection process
  - I.e., how the coefficients change as the regularization parameter λ varies
  - Best model: The λ value at the dashed vertical line

```
ggplot(df_coef_long, aes(x = log(lambda), y = coefficient, color = feature)) +
  geom_line() +
  geom_point() +
  geom_vline(xintercept = log(lambda_best), linetype = "dashed")
```



Excurse: Ridge Regression

- Similar to LASSO, but stabilizing predictions by a shrinkage factor that only **reduces** the size of the coefficients
  - Instead of setting some of them to exactly zero (for any value of `lambda`, incl. $s0$):

```
options(digits=2) #reduce number of digits printed in output

mdl = lrn("regr.glmnet", nlambda = 10, alpha = 0)
mdl$train(tsk)
coef(mdl$model)
## 8 x 10 sparse Matrix of class "dgCMatrix"
##   [[ suppressing 10 column names 's0', 's1', 's2' ... ]]
##
## (Intercept)    3.1e+00  3.09485  3.05323  2.9554  2.7749  2.5641  2.4399  2.411
## age            2.4e-38  0.00027  0.00074  0.0019  0.0045  0.0091  0.0151  0.020
## agree         -1.7e-38 -0.00024 -0.00084 -0.0035 -0.0154 -0.0559 -0.1409 -0.239
## conscientious  3.0e-37  0.00342  0.00936  0.0249  0.0622  0.1356  0.2392  0.334
## extra          1.2e-37  0.00134  0.00351  0.0083  0.0161  0.0237  0.0316  0.045
## gender         1.8e-37  0.00212  0.00585  0.0159  0.0412  0.0955  0.1803  0.264
## neuro         -2.2e-37 -0.00254 -0.00690 -0.0181 -0.0437 -0.0920 -0.1591 -0.221
## open           1.0e-37  0.00117  0.00304  0.0071  0.0127  0.0129  0.0012 -0.015
##
## (Intercept)    2.413  2.418
## age            0.023  0.025
## agree         -0.307 -0.339
```

Excurse: Elastic Net

- Elastic Net: Combination of LASSO and Ridge regularization
  - Metaphorically, represents the idea of a "net" that addresses each method's individual limitations by retaining and grouping correlated predictors effectively
    - Ridge: Effectively shrinks coefficients for correlated predictors, but does not perform feature selection
    - LASSO: Selects features but struggles when predictors are highly correlated, arbitrarily choosing one
  - Note that this is actually the algorithm we used by default in `mlr3` as learner: `"regr.glmnet"`
- The "mixing parameter" `alpha` determines the penalty term:

$$\frac{(1-\alpha)}{2}\|\beta\|_2^2 + \alpha\|\beta\|_1 \tag{3}$$

  - `alpha` $= 1$: Pure L1 regularization $\Rightarrow$ LASSO
  - `alpha` $= 0$: Pure L2 regularization $\Rightarrow$ Ridge
  - $0 <$ `alpha` < 1: Elastic Net, blending the two methods

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, $X_i = (x_{i1}, x_{i2}, \dots, x_{ip})$
  - **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 of which we know the vector of features but not the value of the target:

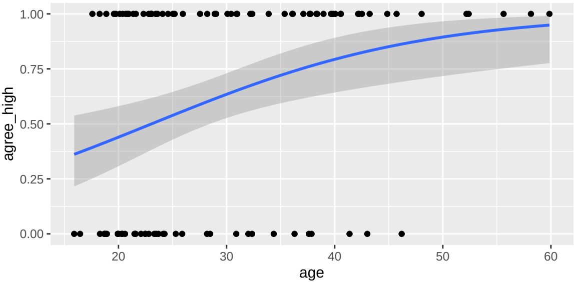
$$X_{new} \rightarrow \hat{p}_{new} \in (0, 1) \tag{4}$$

  - Predicted probability of class 1:

$$\hat{p} = \frac{e^{\hat{\beta}_0 + \hat{\beta}_1 x_1 + \cdots + \hat{\beta}_p x_p}}{1 + e^{\hat{\beta}_0 + \hat{\beta}_1 x_1 + \cdots + \hat{\beta}_p x_p}} \tag{5}$$

Refresher: Logistic Regression

```r
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

```r
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
```

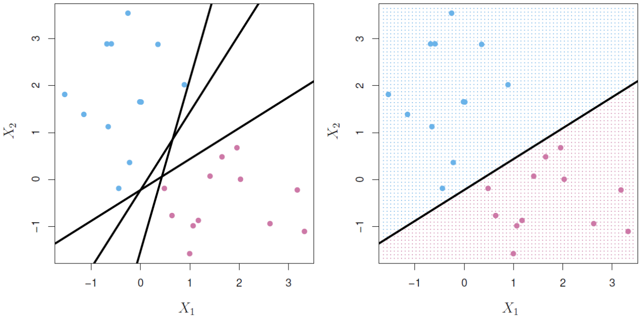
Other Types of Classifiers

- Linear:
  - Linear Discriminant Analysis (not discussed!)
  - Support Vector Machines (next up!)
  - ...
- Nonparamtertic:
  - Classification trees (see below)
  - Random forests (see below)
  - Nearest neighbors (not discussed!)
  - ...

- The remaining module is about using these methods for classification tasks
  - But: They can analogously be used for regression tasks (not discussed!)
    - Usually requires some minor adaptions (e.g., specifying the respective task in `mlr3`)

Support Vector Classifier

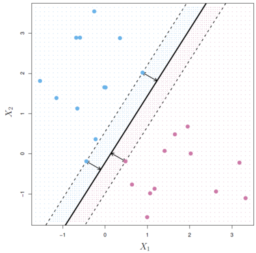
- There is a linear decision boundary (or "hyperplane") used to define the prediction: $\beta_0 + \sum_{j=1}^{p} \beta_j x_j = 0$
  - Prediction depends on whether an instance is above or below this boundary:



(James et al., 2021, Figure 9.2)

Support Vector Classifier

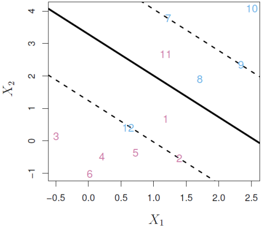
- Support Vector Classifier (SVC): Separating the classes with a hyperplane that maximizes the margin
  - Margin (dashed line): The distance between the hyperplane (i.e., decision boundary) and the training data
  - Predicted class: $\hat{y} = \begin{cases} 1 \text{ if } \hat{\beta}_0 + \sum_{j=1}^{p}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



(James et al., 2021, Figure 9.3)

Support Vector Classifier

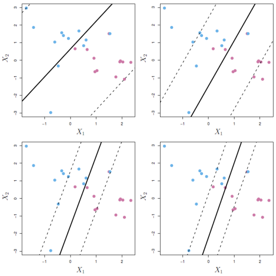
- SVCs typically have a very good classification accuracy compared to other linear methods, but require more technicalities:
  - Hard margin: Requires correct classification for all instances (see previous slide)
    - Overfitting ⇒ Poor generalization!
  - Soft margin: Does not require all instances to be 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
```

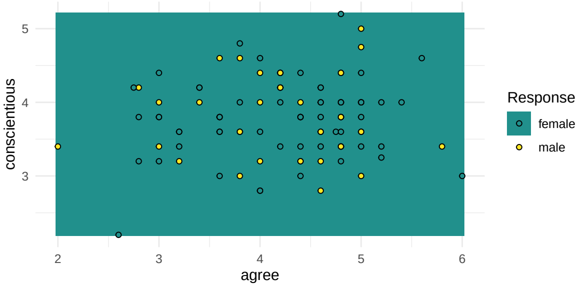
Support Vector Classifier in mlr3

```
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 (in terms of model interpretability) for SVCs as for regression models
  - But the plot of the hyperplane reveals some serious problems:

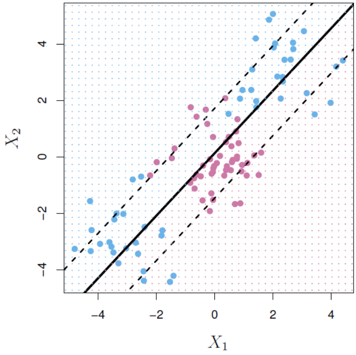
```
autoplot(mdl, task = tsk) + scale_fill_viridis_d(begin = .5)
## Scale for fill is already present.
## Adding another scale for fill, which will replace the existing scale.
```



Support Vector Machines

Support Vector Machines

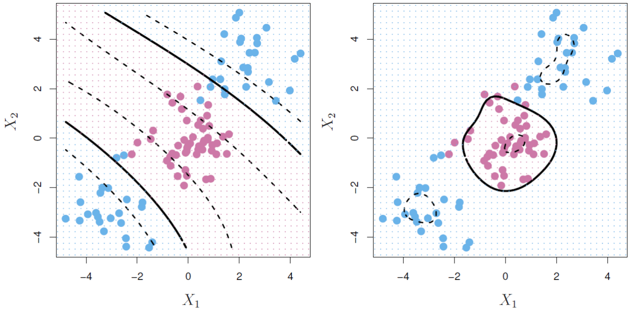
- Challenges for linear classifiers:



(James et al., 2021, Figure 9.8)

Support Vector Machines

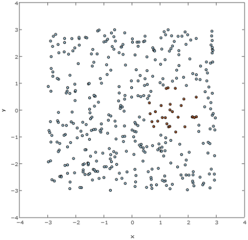
- Solution: Nonlinear decision boundaries

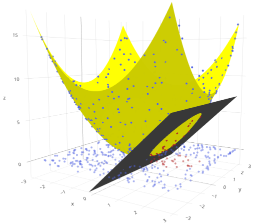


(James et al., 2021, Figure 9.9)

Excurse: The Kernel Trick

- Nonlinear decision boundaries can be achieved via the "kernel trick"
  - Left: Two linearly non-separable classes in 2D space spanned by original features x and y
  - Right: Linear separability with a plane in 3D space by adding a new feature which was constructed from the original two
    - Technically: Mapping the original 2D input data $x = (x, y)$ to a 3D feature space by a (polynomial) function $\Phi(x) = (x, y, x^2 + y^2)$
- Tuning parameters: Properties of kernel function Φ (e.g., radial)





(https://www.efavdb.com/svm-classification)

Support Vector Machines in `mlr3`

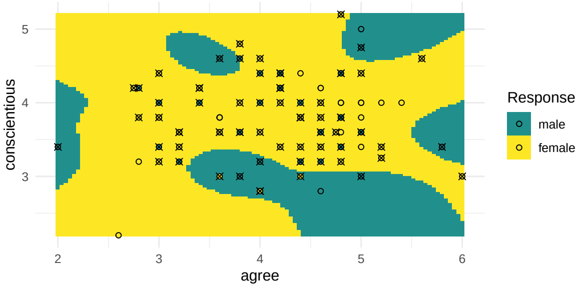
- Using a nonlinear `kernel` (default: "radial"):

```
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`

- **Support Vectors:** Only instances that lie directly on the margin, or on the wrong side of the margin for their class, affect the classifier
  - These instances are marked with a cross
  - All remaining instances play no role for the classification

```
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.
```



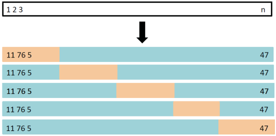
Support Vector Machines in mlr3

- Training classification performance:
  - Overfitting $\Rightarrow$ 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 multiple (k) parts ($=$ "folds")
  2. One fold (e.g., 20% in 5-fold CV) is left out as validation set; the remaining folds are used as training set
  3. Model is trained on the current fold's training set and evaluated on the current fold's validation set
  4. Steps 2 and 3 are repeated for each fold, and the average validation performance is reported as: $CV_{(k)} = \frac{1}{k}\sum_{i=1}^{k} MMCE_i$



(James et al., 2021, Figure 5.5)

Cross-Validation in mlr3

- The `mlr3` library includes a built-in function to perform CV
  - Note: The estimated out-of-sample performance is worse than the in-sample performance (to be expected!)

```r
set.seed(42)
cv <- rsmp("cv", folds = 5)
mdl_cv <- resample(learner = mdl, task = tsk, resampling = cv)
## INFO  [08:46:40.517] [mlr3] Applying learner 'classif.svm' on task 'dat' (iter 1/5)
## INFO  [08:46:40.675] [mlr3] Applying learner 'classif.svm' on task 'dat' (iter 2/5)
## INFO  [08:46:40.720] [mlr3] Applying learner 'classif.svm' on task 'dat' (iter 3/5)
## INFO  [08:46:40.751] [mlr3] Applying learner 'classif.svm' on task 'dat' (iter 4/5)
## INFO  [08:46:40.799] [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
```

Cross-Validation: Hyperparameter Tuning

- CV can be used to choose a good value for the tuning- or hyperparameter
  - 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 the tuning conditions using `auto_tuner()`
   - 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 actual tuning
   - I.e., 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  [08:53:00.187] [bbotk] Starting to optimize 1 parameter(s) with '<OptimizerBatchGri
## INFO  [08:53:00.238] [bbotk] Evaluating 1 configuration(s)
## INFO  [08:53:00.267] [mlr3] Running benchmark with 5 resampling iterations
## INFO  [08:53:00.333] [mlr3] Applying learner 'classif.svm' on task 'dat' (iter 1/5)
## INFO  [08:53:00.368] [mlr3] Applying learner 'classif.svm' on task 'dat' (iter 2/5)
## INFO  [08:53:00.403] [mlr3] Applying learner 'classif.svm' on task 'dat' (iter 3/5)
## INFO  [08:53:00.436] [mlr3] Applying learner 'classif.svm' on task 'dat' (iter 4/5)
## INFO  [08:53:00.466] [mlr3] Applying learner 'classif.svm' on task 'dat' (iter 5/5)
## INFO  [08:53:00.500] [mlr3] Finished benchmark
## INFO  [08:53:00.567] [bbotk] Result of batch 1:
## INFO  [08:53:00.583] [bbotk]  cost classif.ce warnings errors runtime_learners
## INFO  [08:53:00.583] [bbotk]  1000       0.48        0      0             0.07
## INFO  [08:53:00.583] [bbotk]                                 uhash
## INFO  [08:53:00.583] [bbotk]  f73006f6-d72d-4f66-ac76-9f579497f8aa
## INFO  [08:53:00.587] [bbotk] Evaluating 1 configuration(s)
## INFO  [08:53:00.602] [mlr3] Running benchmark with 5 resampling iterations
## INFO  [08:53:00.610] [mlr3] Applying learner 'classif.svm' on task 'dat' (iter 1/5)
## INFO  [08:53:00.632] [mlr3] Applying learner 'classif.svm' on task 'dat' (iter 2/5)
## INFO  [08:53:00.654] [mlr3] Applying learner 'classif.svm' on task 'dat' (iter 3/5)
## INFO  [08:53:00.678] [mlr3] Applying learner 'classif.svm' on task 'dat' (iter 4/5)
```

Hyperparameter Tuning in mlr3

4. Compare the performance for each potential value of C and select (or rather extract) the best hyperparameter

```
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

5. Select the final model (i.e., optimal hyperparameter settings)

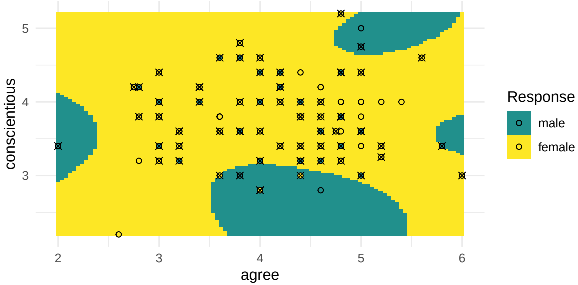
- This model is expected to predict best out-of-sample

```
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

6. Optional plotting of the best model's classification surface

```
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.
```

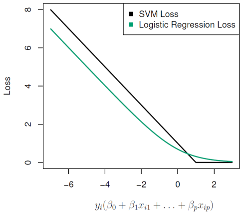


Excurse: Relationship between SVM & Logistic Regression

- Both, SVM and logistic regression can be rewritten as minimizing the so-called loss function

$$\underset{\beta_0,\beta_1,\ldots,\beta_p}{\text{minimize}}\{L(X,y,\beta)+\lambda P(\beta)\} \tag{6}$$

  - Loss: Quantifies the extent to which the model, parametrized by $\beta = (\beta_0, \beta_1, \ldots, \beta_p)$, fits the data (X, y)
- 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 training data (i.e., "hard" margin):

$$y_i \hat{y}_i \geq 1 \, \forall i = 1, \ldots, N \tag{7}$$

  - $y_i = 1$ 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:

$$y_i \left(\beta_0 + \sum_{j=1}^{p} \beta_j x_j \right) > 0 \tag{8}$$

  - If this condition is true, only the two cases from above are possible
    - At least for hard margins

Excurse: The Optimization Problem

- Maximizing the "soft" margin is equivalent to

$$\underset{\beta_0,\beta_1,\ldots,\beta_p,\xi}{\text{minimize}} \frac{1}{2}\sum_{j=1}^{p}\beta_j^2 + \frac{C}{N}\sum_{i=1}^{N}\xi_i \tag{9}$$

s.t.

$$y_i\left(\beta_0 + \sum_{j=1}^{p}\beta_j x_{ij}\right) \geq 1 - \xi_i \,\forall i = 1,\ldots,N \tag{10}$$

$$\xi_i \geq 0 \,\forall i = 1,\ldots,N \tag{11}$$

- **"Slack variables" ξ:** Allow instances to be on the wrong side of the margin and 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

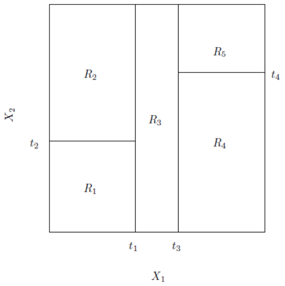
- Now it's your turn:
  - Go to https://tobiasrebholz.github.io/teaching
  - Select "Introduction to Machine Learning" > "Materials"
    - Password: **smip24**
  - Download the "Support Vector Machines" tutorial
  - Work through the tasks

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:

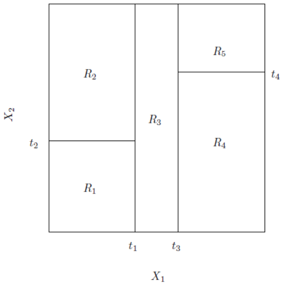
$$\text{If } X_{new} \in \mathbf{R}_l, \text{then } \hat{y}_{new} = y_l \tag{12}$$

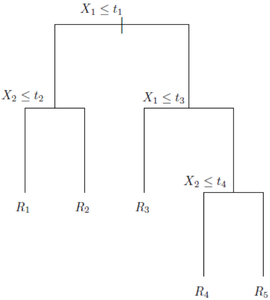


(James et al., 2021, Figure 8.3)

Classification Trees

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





(James et al., 2021, Figure 8.3)

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 (as measured by, 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 only one single class: $\pi_1\pi_{-1} = 0$
- 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!

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; between-subjects) to determine their performance-dependent bonus payment
  - **"Algorithm aversion":** General preference for humans over algorithms (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
```

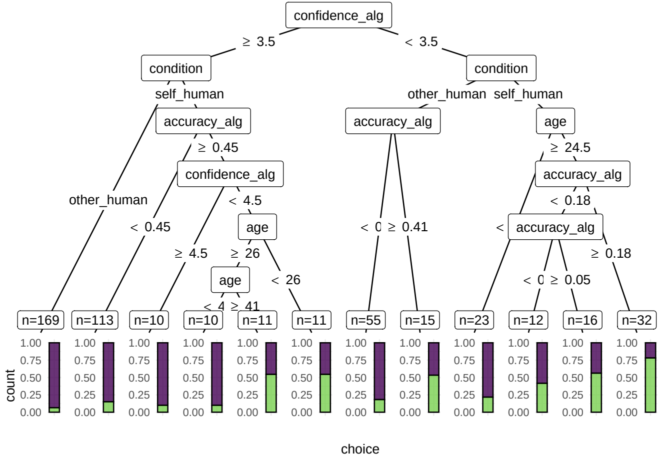
Classification Trees in mlr3

```
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`

- Compact tree representation of this complex partitioning:

```
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

pred$score(msrs("classif.acc"))
## classif.acc
##    0.830189
```

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

Classification Trees in mlr3

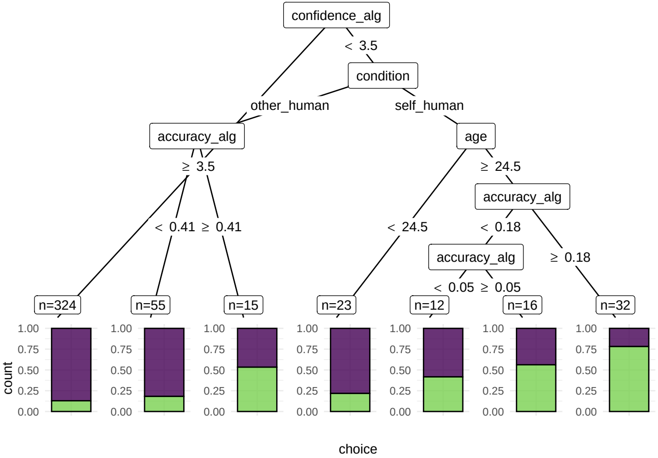
- Many partitions in our example lead to the same prediction
  - In other words, they are redundant/uninformative
  - Solution: "Pruning" the tree by increasing the penalty on complexity cp

```
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

pred$score(msrs("classif.acc"))
## classif.acc
##    0.825996
```

Hyperparameter Tuning in mlr3

- Better than pruning the tree manually to remove unnecessary partitions:
  - Tuning the hyperparameter cp by means of CV (e.g., 5-fold)

```
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  [08:53:27.688] [bbotk] Starting to optimize 1 parameter(s) with '<OptimizerBatchGr
## INFO  [08:53:27.711] [bbotk] Evaluating 1 configuration(s)
## INFO  [08:53:27.738] [mlr3] Running benchmark with 5 resampling iterations
## INFO  [08:53:27.755] [mlr3] Applying learner 'classif.rpart' on task 'dat' (iter 1/5)
## INFO  [08:53:27.790] [mlr3] Applying learner 'classif.rpart' on task 'dat' (iter 2/5)
## INFO  [08:53:27.825] [mlr3] Applying learner 'classif.rpart' on task 'dat' (iter 3/5)
## INFO  [08:53:27.858] [mlr3] Applying learner 'classif.rpart' on task 'dat' (iter 4/5)
## INFO  [08:53:27.890] [mlr3] Applying learner 'classif.rpart' on task 'dat' (iter 5/5)
## INFO  [08:53:27.920] [mlr3] Finished benchmark
## INFO  [08:53:27.987] [bbotk] Result of batch 1:
## INFO  [08:53:27.991] [bbotk]    cp classif.ce warnings errors runtime learners
```

Hyperparameter Tuning in `mlr3`

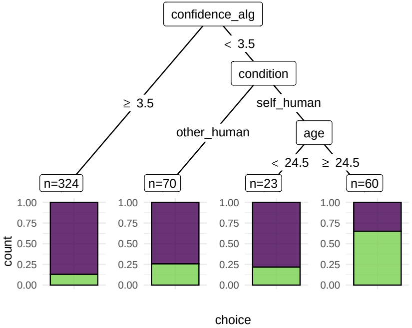
- Ideally, the best value for the hyperparamter lies somewhere "in the middle" of the grid to be searched
  - Why? – If it lies at the borders, there might be a better model for which the hyperparameter is smaller (larger) than the minimum (maximum) value tested

```
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`

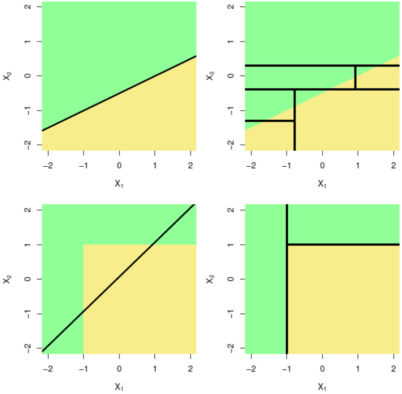
- CV-pruned tree:

```
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)

Hands-on Practical Tutorial

- Now it's your turn:
  - Go to https://tobiasrebholz.github.io/teaching
  - Select "Introduction to Machine Learning" > "Materials"
    - Password: **smip24**
  - Download the "Trees" tutorial
  - Work through **tasks 1–5**

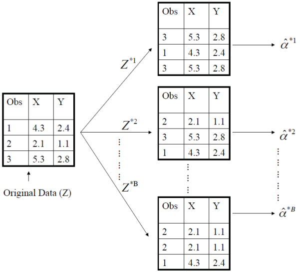
Random Forests

Random Forests

- Advantages of trees:
  - Can easily handle both numerical and categorical variables
  - Can easily handle multiclass problems
  - 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:** Random Forests
    - Building a collection of deep trees
    - Classifying a new instance X_{new} according to the class which is assigned to it by the **majority** of deep trees
  - Bias-variance trade-off:
    - Deep trees naturally have low bias
    - Variance reduction is achieved by aggregating the predictions of many deep trees

Random Forests

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



(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`

- Simultaneous tuning of **multiple** important hyperparameters:
  - `num.trees`: Number of trees in the forest
  - `mtry`: Number of features to be considered for each split

```r
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  [08:53:33.717] [bbotk] Starting to optimize 2 parameter(s) with '<OptimizerBatchGri
## INFO  [08:53:33.743] [bbotk] Evaluating 1 configuration(s)
## INFO  [08:53:33.763] [mlr3] Running benchmark with 5 resampling iterations
## INFO  [08:53:33.774] [mlr3] Applying learner 'classif.ranger' on task 'dat' (iter 1/5)
## INFO  [08:53:34.007] [mlr3] Applying learner 'classif.ranger' on task 'dat' (iter 2/5)
## INFO  [08:53:34.210] [mlr3] Applying learner 'classif.ranger' on task 'dat' (iter 3/5)
## INFO  [08:53:34.440] [mlr3] Applying learner 'classif.ranger' on task 'dat' (iter 4/5)
```

Random Forests in mlr3

- Testing multiple combinations of hyperparameters:
  - Computationally intensive with grid search!

```
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   0.224539
##  3:       100      4   0.226645
##  4:       100      5   0.245461
##  5:       500      2   0.220373
##  6:       500      3   0.228816
##  7:       500      4   0.235066
##  8:       500      5   0.241294
##  9:      1000      2   0.216184
## 10:      1000      3   0.226689
## 11:      1000      4   0.235066
## 12:      1000      5   0.232939

mdl_cv$tuning_result
##      mtry num.trees learner_param_vals  x_domain classif.ce
##    <char>    <char>             <list>    <list>      <num>
## 1:      2      1000          <list[4]> <list[2]>   0.216184
```

Random Forests in mlr3

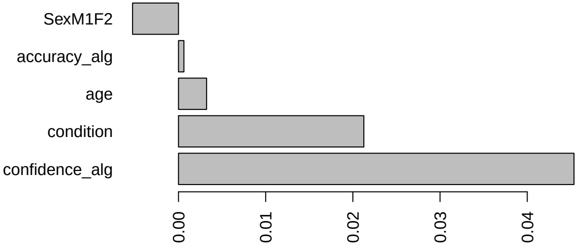
- Final, optimal model:

```
mdl_cv$learner$model
## Ranger result
##
## Call:
##  ranger::ranger(dependent.variable.name = task$target_names, data = task$data(),      pr
##
## Type:                             Classification
## Number of trees:                  1000
## Sample size:                      477
## Number of independent variables:  5
## Mtry:                             2
## Target node size:                 1
## Variable importance mode:         permutation
## Splitrule:                        gini
## OOB prediction error:             22.22 %
```

Random Forests in mlr3

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

```
par(mar = c(5, 10, 2.5, 2.5))  # Bottom, Left, Top, Right margins
barplot(mdl_cv$importance(), horiz = T, las = 2)
```



- Note: The importance of a feature can also be negative, especially for noisy features (e.g., SexM1F2)
  - We would expect improved predictive performance if these "bad" features were actually removed from the model

Excurse: Interpretable ML

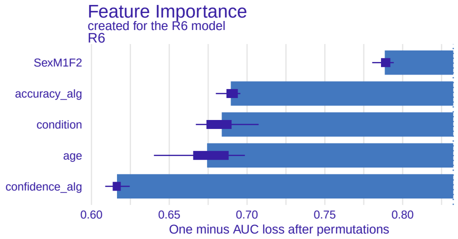
- **Permutation importance:** The importance of feature x_j is defined as the change in accuracy by randomly reshuffling the values of x_j
  - Note: **"Model-agnostic"** measure of feature importance (see also Module 5)
    - Can be applied with any trained predictive model (not only RFs)
- `DALEXtra`: Package that offers many tools for interpretable ML (e.g., permutation importance)
  - I.e., useful for making sense of the prediction behavior of black-box models

```
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) #B = number of permutations
```

Excurse: Interpretable ML

- Visualizing the variability of importance across permutations:
  - By default, `DALEXtra` calculates AUC-based importance measures

```
plot(varimp)
```



- Note: The ordering of `age` and `condition` is reversed now
  - Classical scores (e.g., permutation) provide a **relative** measure of feature importance ⇒ Absolute permutation score values are not very informative

Hands-on Practical Tutorial

- Your turn:
  - Finish the remaining tasks of the “Trees” tutorial

Summary

Summary

- Supervised Learning is a main task in data-driven decision-making
  - E.g., classification: The target to predict is class membership
  - We have discussed regularized regression, Support Vector Machines and tree-based algorithms, incl. ensamble methods (i.e., RFs)
- Advanced ML is much **more intuitive** than classical statistics
  - No distributional assumptions, p-values, . . .
  - But: No magical black box
    - Essentially consisting of 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, . . .