5 – Temporal Classification Stacks

From Single-Date RF Maps to Time-Series Classification Cubes

Chris Reudenbach

Goals

- Use the existing Burgwald data pipeline as a fixed base.
- Run and understand two key scripts:
  - `02-1-basic-classifications_cdse.R` (single-date CDSE classification)
  - `02-2-bfast-classification.R` (BFAST-based kNDVI change detection on a NetCDF cube)
- Create and use training polygons for supervised classification.
- Generalise the single-date RF classification to multiple time points.
- Build the conceptual basis for a temporal classification stack (multi-date RF maps in one cube).
- Prepare the basis for later stratification and similarity mapping.

Written output...

Please export all written answers as a PDF and save it in the project directory `docs/` using the filename convention `<Name1_NameN>_ws-05.pdf`. This helps you maintain a clean and consistent documentation of your work.
You are welcome to use ChatGPT for wording support, but the important part is that you genuinely understand the tasks and the underlying concepts — the AI is only meant to reduce typing effort, not replace your comprehension.

! How to use ChatGPT for this assignment

You may use ChatGPT **only** to refine the wording of your own ideas.
Please follow this workflow:

1. **Write your own bullet points first** (even rough or incomplete).
2. Paste your bullets into ChatGPT and ask for **clarification, shortening, or better phrasing**.

3. ChatGPT will only help with **language and structure**, not with content.
4. If you are unsure about something, write: *"I'm not sure about X"* — ChatGPT may then give **hints or guiding questions**, but **no solutions**.
5. Answers must remain **ultra-short and clear** (1–3 bullets or 1–2 sentences).
6. Do **not** ask ChatGPT for full answers; the goal is that **you understand the task**, not that AI solves it.

This ensures that the work remains genuinely yours while reducing unnecessary typing.

Prerequisites

Before you start, make sure that:

1. You have cloned the course repository and opened the RStudio Project.
2. You have run `01_setup-burgwald.R` once in this project:
   - `here::here()` points to the project root.
   - The object `root_folder` exists and points to the same location.
3. The following input data are available (paths relative to the project root):
   - Monthly Sentinel-2 cube (gdalcubes NetCDF):
     - `data/burgwald_2018_2022_all.nc`
   - CDSE-based predictor stack for one reference date (e.g. 2018):
     - `data/pred_stack_2018.tif`
   - A training polygon layer with a `class` field:
     - `data/processed/train_areas_burgwald.gpkg`
4. The required R packages are installed:
   - `terra`, `sf`, `dplyr`, `exactextractr`, `caret`, `RStoolbox`, `randomForest`, `sp`
   - `gdalcubes`, `stars`, `tmap`, `bfast`

If any of these elements are missing, go back to the corresponding previous exercise or script.

Task 1 – Digitise training polygons for classification purpose

This task prepares the training data required by `02-1-basic-classifications_cdse.R`.

1. Open QGIS (or another GIS) with the Burgwald base data (e.g. Sentinel-2 composite, CHM, existing reference layers).
2. Create a new polygon layer (or extend an existing one) for **training areas**, with at least:

- geometry type: polygons
- coordinate reference system: identical to the CDSE stack (`pred_stack_2018.tif`)
- attribute field: `class` (text or integer)

3. Digitise polygons for the land-cover / disturbance classes you want to separate (e.g. “clearcut”, “forest”, “agriculture”, “other”).
4. Make sure that:

   - each polygon has a valid `class` value,
   - polygons are sufficiently homogeneous (avoid mixing classes inside one polygon),
   - polygons cover representative examples of each class.

5. Save the polygon layer as:

   - `data/processed/train_areas_burgwald.gpkg`
   - with the attribute field exactly named `class`.

Short written output (bullet points/table):

- Which classes did you digitise?
- How many polygons per class (approximately)?
- Any obvious sources of ambiguity (mixed pixels, shadows, edge effects)?

Task 2 – Inspect the predictor stack

Now connect the training polygons with the CDSE stack.

1. In RStudio, load the predictor stack and training layer:

```r
library(here)
library(terra)
library(sf)

pred_stack  <- terra::rast(here::here("data", "pred_stack_2018.tif"))
train_areas <- sf::st_read(here::here("data", "processed", "train_areas_burgwald.gpkg"))
```

2. Short written output (bullet points/table):

   - How many layers does `pred_stack` contain?
   - Which types of predictors do you see in the layer names (e.g. spectral bands, indices, texture, CDSE components)?
   - Do the training polygons overlay the predictor raster correctly (no obvious misalignment)?

3. Create one quick visual check (include the figure in your written PDF output):

- Either a simple RGB plot of three layers;
- or a plot of a single layer with training polygons overlaid.

Written output: add brief comments on what you can clearly observe and detect (e.g., visible clearcuts vs. intact forest, obvious artefacts, unusual patterns, etc.), and also on what you obviously cannot identify from the imagery.

Task 3 – Run the single-date CDSE classification script

In this step you use the script:

```
02-1-basic-classifications_cdse.R
```

This script performs, for the single CDSE predictor stack `pred_stack_2018.tif`:

- unsupervised k-means clustering,
- supervised Maximum Likelihood Classification (MLC) via `RStoolbox::superClass()`,
- supervised Random Forest (RF) via `caret + randomForest`,
- and computes a confusion matrix for the RF model.

1. Open `02-1-basic-classifications_cdse.R` in RStudio.
2. Check the input paths near the top of the script:
   - `pred_stack_file <- here::here("data", "pred_stack_2018.tif")`
   - `train_poly_file <- here::here("data", "processed", "train_areas_burgwald.gpkg")`
   - `class_field <- "class"` Adjust only if your filenames differ.
3. Run the script stepwise from top to bottom.
4. Confirm that the following outputs are written:
   - k-means map:
     - `data/classification_kmeans_2018.tif`
   - MLC map:
     - `data/classification_mlc_2018.tif`
   - RF map:
     - `data/classification_rf_2018.tif`
   - RF model:
     - `data/rf_model_2018_cdse.rds`

- RF confusion matrix:
  - `data/rf_confusion_2018_cdse.rds`

5. Load and plot the classification maps (e.g. for the RF results):

```
rf_map <- terra::rast(here::here("data", "classification_rf_2018.tif"))
plot(rf_map)
```

Short written output:

- Compare the three classification results (k-means, MLC, RF) with respect to your chosen class scheme, especially the distinction between *forest* and *non-forest*.
- Describe where each method appears to produce reliable results and where it does not.
- Identify areas where misclassifications are likely (e.g., mixed land cover, shadows, edges, cloud artefacts).
- Summarise the qualitative differences between the three methods in terms of class separation, spatial coherence, and plausibility.

Task 4 – Identify the RF classification core in the script

Still in `02-1-basic-classifications_cdse.R`:

1. Locate the block where Random Forest is trained:
   - Find the `caret::train(...)` call that creates the RF model.
   - Note:
     - the name of the RF model object,
     - which columns are used as predictors (`predictor_cols`),
     - which column is the response (`class_field`).
2. Locate the block where RF is applied to the raster:
   - Find the line where `terra::predict(pred_stack, rf_model, ...)` is called.
   - Find the line where the resulting RF map is written to disk as `classification_rf_2018.tif`.
3. Short Written output: (bullet points), describe the pipeline:
   - Input for the RF model (type of data, number of samples, predictor variables, class labels).
   - Output of the RF model (raster map, probability or hard classes).
   - Where and how accuracy is assessed (test data, confusion matrix, metrics).

Task 5 – From single-date RF to a temporal classification stack (concept only)

Assume you do not only have `pred_stack_2018.tif`, but equivalent CDSE predictor stacks for several years, for example:

- `pred_stack_2018.tif`
- `pred_stack_2019.tif`
- `pred_stack_2020.tif`
- …

All stacks share the same predictor structure (same bands / indices / CDSE components, same CRS and resolution).

Write a short conceptual plan (max. 10–12 sentences; optionally formulated as commented R code) that explains how you would extend the single-date RF classification workflow to multiple years. Your plan should describe how the trained RF model can be applied to CDSE predictor stacks from several time points, how you would organise the resulting annual RF maps into a temporal classification stack, how you would ensure temporal consistency between years, and what the added value of such a temporal RF stack would be compared to a single-year classification.

Task 6 – Run BFAST-based kNDVI change detection on the cube

Now you move from single-date classification to time-series-based change detection using:

```
02-2-bfast-classification.R
```

This script:

- reads the monthly Sentinel-2 cube `data/burgwald_2018_2022_all.nc`,
- computes a kNDVI time series per pixel from bands `B08` and `B04`,
- runs `bfastmonitor()` for each pixel,
- returns two per-pixel outputs:
  - `change_date` (estimated time of structural change),
  - `change_magnitude` (magnitude of change at the breakpoint),
- writes both outputs to `data/burgwald_bfast_results.nc`,
- and visualises the results using `stars` and `tmap`.

1. Open `02-2-bfast-classification.R` in RStudio.
2. Identify and note the following elements:
   - Input cube path:

    - `cube_file <- file.path(root_folder, "data", "burgwald_2018_2022_all.nc")`
  - Output NetCDF path:
    - `bfast_file <- file.path(root_folder, "data", "burgwald_bfast_results.nc")`
  - The `reduce_time(...)` call and the embedded `FUN = function(x) { ... }`.

3. Written output: Inside the custom reducer function `FUN = function(x)`, describe (in your own words, bullet points):

   - How kNDVI is computed from `B08` and `B04` (formula, scaling).
   - How the time series object `kndvi_ts` is constructed (start date, frequency).
   - How `bfastmonitor()` is configured (arguments `start`, `history`, `level`).
   - Which two values the function returns per pixel (breakpoint and magnitude).

4. Run the script once so that `data/burgwald_bfast_results.nc` is created.

5. Use the plotting section at the end of the script (or an equivalent snippet) to visualise:

   - `change_date`
   - `change_magnitude`

   Inspect the maps interactively (e.g. with `tmap_mode("view")`) and as static plots.

Short written interpretation:

- What does an early vs. late `change_date` indicate in terms of disturbance timing?
- How do positive vs. negative `change_magnitude` values relate to vegetation loss or recovery?
- In which parts of the Burgwald do you see strong BFAST signals (large magnitude changes, clustered patterns)?
- How could these patterns be linked to known events (e.g. bark beetle outbreaks, storms, clearcuts)?

Take-home – Temporal stacks as a basis for stratification and similarity mapping

Write one short paragraph (max. 6–7 sentences) that summarises:

- Why it is useful to have **RF classification maps for multiple time points** in a common stack.
- How such a stack could be used to:
  - define temporal strata (e.g. “always stable forest”, “recent clearcuts”, “gradual change”),
  - detect areas with similar temporal class trajectories (similarity mapping).

- How the RF-based temporal stack and the BFAST-based change layers could complement each other for:
  - sampling design,
  - prioritising field visits,
  - and designing monitoring strategies.

This paragraph will serve as a conceptual bridge to the next exercise on stratified sampling and similarity mapping based on temporal class trajectories.