

Supplementary material for *Using citizen science to parse climatic and landcover influences on bird occupancy within a tropical biodiversity hotspot*

Vijay Ramesh Pratik R. Gupte Morgan W. Tingley VV Robin Ruth DeFries

2020-12-26

Contents

1	Introduction	2
1.1	Attribution	2
2	Selecting species of interest	2
2.1	Prepare libraries	2
2.2	Read species of interest	3
2.3	Load raw data for locations	3
2.4	Get proportional obs counts in 25km cells	4
2.5	Which species are reported sufficiently in checklists?	5
2.6	Figure: Checklist distribution	6
2.7	Prepare the species list	6
3	Landcover classification	6
4	Spatial Autocorrelation of Climatic Predictors	10
4.1	Load libraries	10
4.2	Prepare data	11
4.3	Calculate variograms of environmental layers	11
4.4	Visualise variograms of environmental data	12
5	Climatic raster resampling	13
5.1	Prepare landcover	13
5.2	Prepare spatial extent	14
5.3	Prepare CHELSA rasters	14
5.4	Resample prepared rasters	15
6	Climate in Relation to Landcover	15
6.1	Prepare libraries	15
6.2	Prepare environmental data	15
6.3	Climatic variables over landcover	16
7	Distribution of Observer Expertise	16
7.1	Prepare libraries	17
7.2	Load observer expertise scores and checklist covariates	17
7.3	Species observed in relation to observer expertise	17
7.4	Observer expertise in relation to landcover	18
8	Matching Effort Cutoffs with Spatial Independence Criteria	18

33	8.1	Load libraries	19
34	8.2	Load data	19
35	8.3	Visualise limiting effort by spatial independence limits	20
36	9	Spatial Thinning: A Brief Comparison of Approaches	20
37	9.1	Prepare libraries	20
38	9.2	Traditional grid-based thinning	20
39	9.3	Network-based thinning	24
40	9.4	Finding modularity in proximity networks	24
41	9.5	Process proximity networks in R	26
42	9.6	A function that removes sites	26
43	9.7	Removing non-independent sites	28
44	9.8	Measuring method fallibility	28
45	9.9	Prepare data for Python	29
46	9.10	Count props under threshold in Python	29
47	9.11	Plot metrics for different methods	29
48	10	Predicting Species-specific Occupancy	31
49	10.1	Prepare libraries	31
50	10.2	Read data	31

51 1 Introduction

52 This is supplementary material for a manuscript that uses citizen science data to model the occupancy of birds in the southern
53 Western Ghats, India.

54 The main project can be found here: <https://github.com/pratikunterwegs/eBirdOccupancy>.

55 1.1 Attribution

56 Please contact the following in case of interest in the project.

- 57 • Vijay Ramesh (lead author)
- 58 – PhD student, Columbia University
- 59 • Pratik Gupte (repo maintainer)
- 60 – PhD student, University of Groningen

61 2 Selecting species of interest

62 This script shows the proportion of checklists that report a particular species across every 25km by 25km grid across the
63 Nilgiris and the Anamalais. Using this analysis, we arrived at a final list of species for occupancy modeling.

64 We derived this list from inclusion criteria adapted from the State of India's Birds 2020 (Viswanathan et al., 2020). Initially,
65 we considered all 561 species in eBird that occurred within the outlines of our study area. We then considered only those
66 species that had a minimum of 1000 detections each between 2013 and 2019 (reducing to 303 species). Next, the study area
67 was divided into 25 x 25 km cells following (Viswanathan et al., 2020). We then kept only those species that occurred in at
68 least 5% of all checklists across 50% of the 25 x 25 km cells from where they have been reported (reducing to 93 species).
69 We used the above criteria to ensure as much uniform sampling of a species as possible across our study area and to reduce
70 any erroneous associations between environmental drivers and species occupancy. Across our final list of 93 species, we
71 analyzed a total of ~3.2 million detections (presences) between 2013 and 2019.

72 2.1 Prepare libraries

```
1 # load libraries
2 library(data.table)
```

```

3 library(readxl)
4 library(magrittr)
5 library(stringr)
6 library(dplyr)
7 library(tidyr)
8 library(readr)
9
10 library(ggplot2)
11 library(ggthemes)
12 library(scico)
13
14 # round any function
15 round_any <- function(x, accuracy = 25000) {
16   round(x / accuracy) * accuracy
17 }

```

73 2.2 Read species of interest

74 We initially considered all species that

```

1 # add species of interest
2 specieslist <- read.csv("data/01_list-all-spp-byCount.csv")
3 speciesOfInterest <- specieslist$scientific_name

```

75 2.3 Load raw data for locations

```

1 # read in shapefile of the study area to subset by bounding box
2 library(sf)
3 wg <- st_read("data/spatial/hillsShapefile/Nil_Ana_Pal.shp")
4 box <- st_bbox(wg)
5
6 # read in data and subset
7 # To access the latest dataset, please visit: https://ebird.org/data/download and set the file path accordingly.
8 ebd <- fread("data/ebd_IN_relApr-2020.txt")
9 ebd <- ebd[between(LONGITUDE, box["xmin"], box["xmax"]) &
10   between(LATITUDE, box["ymin"], box["ymax"]), ]
11 ebd <- ebd[year(`OBSERVATION DATE`) >= 2013, ]
12
13 # make new column names
14 newNames <- str_replace_all(colnames(ebd), " ", "_") %>%
15   str_to_lower()
16 setnames(ebd, newNames)
17
18 # keep useful columns
19 columnsOfInterest <- c(
20   "scientific_name", "observation_count", "locality",
21   "locality_id", "locality_type", "latitude",
22   "longitude", "observation_date", "sampling_event_identifier"
23 )
24
25 ebd <- ebd[, ..columnsOfInterest]
26
27 Add a spatial filter and assign grids of 25km x 25km.
28
29 # strict spatial filter and assign grid
30 locs <- ebd[, .(longitude, latitude)]

```

```

3
4 # transform to UTM and get 20km boxes
5 coords <- setDF(locs) %>%
6   st_as_sf(coords = c("longitude", "latitude")) %>%
7   `st_crs<-`(4326) %>%
8   bind_cols(as.data.table(st_coordinates(.))) %>%
9   st_transform(32643) %>%
10  mutate(id = 1:nrow(.))
11
12 # convert wg to UTM for filter
13 wg <- st_transform(wg, 32643)
14 coords <- coords %>%
15   filter(id %in% unlist(st_contains(wg, coords))) %>%
16   rename(longitude = X, latitude = Y) %>%
17   bind_cols(as.data.table(st_coordinates(.))) %>%
18   st_drop_geometry() %>%
19   as.data.table()
20
21 # remove unneeded objects
22 rm(locs)
23 gc()
24
25 coords <- coords[, .N, by = .(longitude, latitude, X, Y)]
26
27 ebd <- merge(ebd, coords, all = FALSE, by = c("longitude", "latitude"))
28
29 ebd <- ebd[(longitude %in% coords$longitude) &
30   (latitude %in% coords$latitude), ]

```

77 2.4 Get proportional obs counts in 25km cells

```

1 # round to 25km cell in UTM coords
2 ebd[, `:=`(X = round_any(X), Y = round_any(Y))]
3
4 # count checklists in cell
5 ebd_summary <- ebd[, nchk := length(unique(sampling_event_identfier)),
6   by = .(X, Y)
7 ]
8
9 # count checklists reporting each species in cell and get proportion
10 ebd_summary <- ebd_summary[, .(nrep = length(unique(
11   sampling_event_identfier
12 ))),
13 by = .(X, Y, nchk, scientific_name)
14 ]
15
16 ebd_summary[, p_rep := nrep / nchk]
17
18 # filter for soi
19 ebd_summary <- ebd_summary[scientific_name %in% speciesOfInterest, ]
20
21 # complete the dataframe for no reports
22 # keep no reports as NA --- allows filtering based on proportion reporting
23 ebd_summary <- setDF(ebd_summary) %>%

```

```

24 complete(
25   nesting(X, Y), scientific_name # ,
26   # fill = list(p_rep = 0)
27 ) %>%
28 filter(!is.na(p_rep))

```

78 2.5 Which species are reported sufficiently in checklists?

```

1  # A total of 42 unique grids (of 25km by 25km) across the study area
2  # total number of checklists across unique grids
3
4  tot_n_chklist <- ebd_summary %>%
5    distinct(X, Y, nchk)
6
7  # species-specific number of grids
8  spp_grids <- ebd_summary %>%
9    group_by(scientific_name) %>%
10   distinct(X, Y) %>%
11   count(scientific_name,
12     name = "n_grids"
13 )
14
15 # Write the above two results
16 write_csv(tot_n_chklist, "data/nchk_per_grid.csv")
17 write_csv(spp_grids, "data/ngrids_per_spp.csv")
18
19 # left-join the datasets
20 ebd_summary <- left_join(ebd_summary, spp_grids, by = "scientific_name")
21
22 # check the proportion of grids across which this cut-off is met for each species
23 # Is it > 90% or 70%?
24 # For example, with a 3% cut-off, ~100 species are occurring in >50%
25 # of the grids they have been reported in
26
27 p_cutoff <- 0.05 # Proportion of checklists a species has been reported in
28 grid_proportions <- ebd_summary %>%
29   group_by(scientific_name) %>%
30   tally(p_rep >= p_cutoff) %>%
31   mutate(prop_grids_cut = n / (spp_grids$n_grids)) %>%
32   arrange(desc(prop_grids_cut))
33
34 grid_prop_cut <- filter(
35   grid_proportions,
36   prop_grids_cut > p_cutoff
37 )
38
39 # Write the results
40 write_csv(grid_prop_cut, "data/chk_5_percent.csv")
41
42 # Identifying the number of species that occur in potentially <5% of all lists
43 total_number_lists <- sum(tot_n_chklist$nchk)
44
45 spp_sum_chk <- ebd_summary %>%
46   distinct(X, Y, scientific_name, nrep) %>%

```

```

47   group_by(scientific_name) %>%
48   mutate(sum_chk = sum(nrep)) %>%
49   distinct(scientific_name, sum_chk)
50
51   # Approximately 90 to 100 species occur in >5% of all checklists
52   prop_all_lists <- spp_sum_chk %>%
53   mutate(prop_lists = sum_chk / total_number_lists) %>%
54   arrange(desc(prop_lists))

```

79 2.6 Figure: Checklist distribution

```

1   # add land
2   library(rnaturalearth)
3   land <- ne_countries(
4     scale = 50, type = "countries", continent = "asia",
5     country = "india",
6     returnclass = c("sf")
7   )
8
9   # crop land
10  land <- st_transform(land, 32643)

```

80 2.7 Prepare the species list

```

1   # write the new list of species that occur in at least 5% of checklists across a minimum of 50% of the grids they h
2
3   new_sp_list <- semi_join(specieslist, grid_prop_cut, by = "scientific_name")
4
5   write_csv(new_sp_list, "data/03_list-of-species-cutoff.csv")

```

81 3 Landcover classification

82 This script was used to classify a 2019 Sentinel composite image across the Nilgiris and the Anamalais into
 83 seven distinct land cover types. The same code can be viewed on GEE here: <https://code.earthengine.google.com/ec69fc4ffad32a532b25202009243d42>.
 84

```

1   // Data: Groundtruthed points from Arasumani et al 2019
2
3   // Function to obtain a Cloud-Free Image //
4
5   /**
6    * Function to mask clouds using the Sentinel-2 QA band
7    * @param {ee.Image} image Sentinel-2 image
8    * @return {ee.Image} cloud masked Sentinel-2 image
9    */
10
11  function maskS2clouds(image) {
12    var qa = image.select('QA60');
13
14    // Bits 10 and 11 are clouds and cirrus, respectively.
15    var cloudBitMask = 1 << 10;
16    var cirrusBitMask = 1 << 11;
17
18    // Both flags should be set to zero, indicating clear conditions.

```

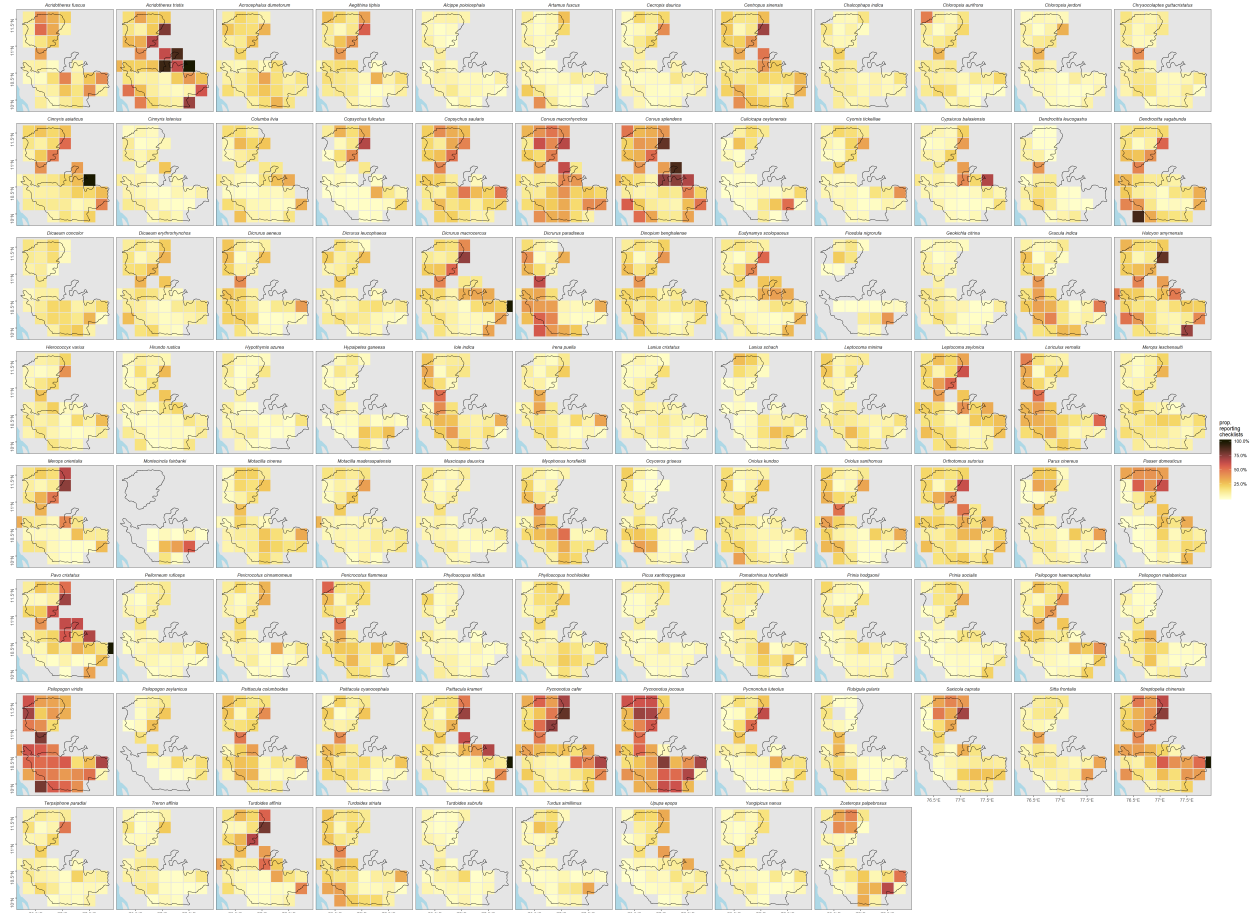


Figure 1: Proportion of checklists reporting a species in each grid cell (25km side) between 2013 and 2019. Checklists were filtered to be within the boundaries of the Nilgiris and the Anamalai hills (black outline), but rounding to 25km cells may place cells outside the boundary. Deeper shades of red indicate a higher proportion of checklists reporting a species.

```

19   var mask = qa.bitwiseAnd(cloudBitMask).eq(0)
20       .and(qa.bitwiseAnd(cirrusBitMask).eq(0));
21
22   return image.updateMask(mask).divide(10000);
23 }
24
25 // Importing shapefile needed for classification
26 var clipper = function(image){
27   return image.clip(WG_Buffer);
28 };
29
30
31 // Import raw Sentinel scenes and clip them over your study area
32 var filtered = sentinel.filterDate('2018-01-01','2018-12-01').map(clipper);
33
34 // Load Sentinel-2 TOA reflectance data.
35 // Pre-filter to get less cloudy granules.
36
37 var dataset = filtered.filter(ee.Filter.lt('CLOUDY_PIXEL_PERCENTAGE', 20))
38     .map(maskS2clouds);
39 var scene = dataset.reduce(ee.Reducer.median());
40
41 Map.addLayer(WG_Buffer, {}, 'Buffer Outline for Nil/Ana/Pal');
42 // Map.addLayer(scene, {}, 'Image for Classification');
43 // Map.addLayer(WG, {}, 'Outline for Nilgiris/Anaimalais/Palanis');
44
45
46 // Step 2: Creating training data manually
47 // Added a new shapefile field manually in ArcMap so that GEE can take a float field for classification
48 // Field: landcover
49 // Values: agriculture (1), forest (2), grassland (3), plantation (4), settlements (5), tea (6), waterbodies (7)
50 // Note - Arasu has classified plantation as Acacia, Pine et al sub classes (for future analysis)
51
52 // Merging the featureCollections to obtain a single featureCollection
53
54 var trainingFeatures = agriculture.merge(forests).merge(forests2).merge(grasslands).merge(grasslands2)
55     .merge(settlements).merge(plantations)
56     .merge(waterbodies).merge(tea).merge(tea2).merge(tea3).merge(forests3);
57
58 // // Specify the bands of the sentinel image to be used as predictors (p)
59 var predictionBands = ['B2_median', 'B3_median', 'B4_median', 'B8_median'];
60
61
62 // // Now a random forest is a collection of random trees. It's predictions are used to compute an
63 // // average (regression) or vote on a label (classification)
64
65 var sample = scene.select(predictionBands)
66     .sampleRegions({
67       collection: trainingFeatures,
68       properties : ['landcover'],
69       scale: 10
70     });
71
72 // Let's run a classifier for randomForest

```



```

73  var classifier = ee.Classifier.randomForest(10).train({
74      features: sample,
75      classProperty: 'landcover',
76      inputProperties: predictionBands
77  });
78
79
80  var classified = scene.select(predictionBands).classify(classifier);
81  Map.addLayer(classified, {min:1, max:7,palette:[
82      'be4fc4', // agriculture, violetish
83      '04a310', // forests, lighter green
84      'cbb315', // grasslands, yellowish
85      'c17111', // plantations, brownish
86      'b0a69d', // settlements, grayish
87      '025a05', // tea, dark greenish
88      '2035df', // waterbodies, royal blue
89  ]}, 'classified');
90
91  // Partitioning training data to run an accuracy assessment
92  // Adding a randomColumn of values ranging from 0 to 1
93  var trainingTesting = sample.randomColumn();
94
95  var trainingSet = trainingTesting.filter(ee.Filter.lt('random',0.8));
96  var testingSet = trainingTesting.filter(ee.Filter.gte('random',0.2));
97
98  // Now run the classifier only with the trainingSet
99  var trained = ee.Classifier.randomForest(10).train({
100      features: trainingSet,
101      classProperty: 'landcover',
102      inputProperties: predictionBands
103  });
104
105  // Now classify the testData and obtain a Confusion matrix
106  var confusionMatrix = ee.ConfusionMatrix(testingSet.classify(trained)
107      .errorMatrix({
108          actual: 'landcover',
109          predicted: 'classification'
110      }));
111
112  // Now print the ConfusionMatrix and expand the object to inspect the matrix()
113  // The entries represent the number of pixels and the items on the diagonal represent
114  // correct classification. Items off the diagonal are misclassifications, where class in row i
115  // is classified as column j
116
117  // One can also obtain basic descriptive statistics from the confusionMatrix
118  // Note this won't work as the number of pixels is too high (Export as .csv to obtain result)
119
120  // print('Confusion matrix:', confusionMatrix);
121  // print('Overall Accuracy:', confusionMatrix.accuracy());
122  // print('Producers Accuracy:', confusionMatrix.producersAccuracy());
123  // print('Consumers Accuracy:', confusionMatrix.consumersAccuracy());
124
125  // Since printing the above is gives you a computation timed out error
126  var exportconfusionMatrix = ee.Feature(null, {matrix: confusionMatrix.array()});

```

```

127 var exportAccuracy = ee.Feature(null, {matrix: confusionMatrix.accuracy()});
128
129 Export.table.toDrive({
130   collection: ee.FeatureCollection(exportConfusionMatrix),
131   description: 'confusionMatrix',
132   fileFormat: 'CSV'
133 });
134
135 Export.table.toDrive({
136   collection: ee.FeatureCollection(exportAccuracy),
137   description: 'Accuracy',
138   fileFormat: 'CSV'
139 });
140
141 // Below code suggests that the current projection system is WGS84
142 // print(classified.projection());
143
144 // To project it to UTM
145 var reprojected = classified.reproject('EPSG:32643', null, 10);
146
147 // Export classified image
148 Export.image.toDrive({
149   image: classified,
150   description: 'Classified Image',
151   scale: 10,
152   region: WG_Buffer,      // .geometry().bounds(),
153   fileFormat: 'GeoTIFF',
154   formatOptions: {
155     cloudOptimized: true
156   },
157   maxPixels: 618539476
158 });
159
160 // Export projected image
161 Export.image.toDrive({
162   image: reprojected,
163   description: 'Reprojected Image',
164   scale: 10,
165   region: WG_Buffer,      // .geometry().bounds(),
166   fileFormat: 'GeoTIFF',
167   formatOptions: {
168     cloudOptimized: true
169   },
170   maxPixels: 618539476
171 });

```

85 4 Spatial Autocorrelation of Climatic Predictors

86 4.1 Load libraries

```

1 # load libs
2 library(raster)
3 library(gstat)
4 library(stars)

```

```

5 library(purrr)
6 library(tibble)
7 library(dplyr)
8 library(tidyr)
9 library(glue)
10 library(scales)
11 library(gdalUtils)
12 library(sf)
13
14 # plot libs
15 library(ggplot2)
16 library(ggthemes)
17 library(scico)
18 library(gridExtra)
19 library(cowplot)
20 library(ggspatial)
21
22 #' make custom function to convert matrix to df
23 raster_to_df <- function(inp) {
24
25   # assert is a raster obj
26   assertthat::assert_that("RasterLayer" %in% class(inp),
27     msg = "input is not a raster"
28   )
29
30   coords <- coordinates(inp)
31   vals <- getValues(inp)
32
33   data <- tibble(x = coords[, 1], y = coords[, 2], value = vals)
34
35   return(data)
36 }

```

87 4.2 Prepare data

```

1 # list landscape covariate stacks
2 landscape_files <- "data/spatial/landscape_resamp01km.tif"
3 landscape_data <- stack(landscape_files)
4
5 # get proper names
6 elev_names <- c("elev", "slope", "aspect")
7 chelsa_names <- c("bio_01", "bio_12")
8 names(landscape_data) <- c(elev_names, chelsa_names, "landcover")
9
10 # get chelsa rasters
11 chelsa <- landscape_data[[chelsa_names]]
12 chelsa <- purrr::map(as.list(chelsa), raster_to_df)

```

88 4.3 Calculate variograms of environmental layers

```

1 # prep variograms
2 vgrams <- purrr::map(chelsa, function(z) {
3   z <- drop_na(z)
4   vgram <- gstat::variogram(value ~ 1, loc = ~ x + y, data = z)

```

```

5   return(vgram)
6 })
7
8 # save temp
9 save(vgrams, file = "data/chelsa/chelsaVariograms.rdata")
10
11 # get variogram data
12 vgrams <- purrr::map(vgrams, function(df) {
13   df %>% select(dist, gamma)
14 })
15 vgrams <- tibble(
16   variable = chelsa_names,
17   data = vgrams
18 )
19
20 wg <- st_read("data/spatial/hillsShapefile/Nil_Ana_Pal.shp") %>%
21   st_transform(32643)
22 bbox <- st_bbox(wg)
23
24 # Plot
25 library(rnaturalearth)
26 land <- ne_countries(
27   scale = 50, type = "countries", continent = "asia",
28   country = "india",
29   returnclass = c("sf")
30 )
31
32 # crop land
33 land <- st_transform(land, 32643)

```

89 4.4 Visualise variograms of environmental data

```

1 # make ggplot of variograms
2 yaxis <- c("semivariance", "")
3 xaxis <- c("", "distance (km)")
4 fig_vgrams <- purrr::pmap(list(vgrams$data, yaxis, xaxis), function(df, ya, xa) {
5   ggplot(df) +
6     geom_line(aes(x = dist / 1000, y = gamma), size = 0.2, col = "grey") +
7     geom_point(aes(x = dist / 1000, y = gamma), col = "black") +
8     scale_x_continuous(labels = comma, breaks = c(seq(0, 100, 25))) +
9     scale_y_log10(labels = comma) +
10    labs(x = xa, y = ya) +
11    theme_few() +
12    theme(
13      axis.text.y = element_text(angle = 90, hjust = 0.5, size = 8),
14      strip.text = element_blank()
15    )
16 })
17 fig_vgrams <- purrr::map(fig_vgrams, ggplot2::ggplotGrob)
18
19 # make ggplot of chelsa data
20 chelsa <- as.list(landscape_data[[chelsa_names]]) %>%
21   purrr::map(stars::st_as_stars)
22
23 # colour palettes

```

```

24 pal <- c("bilbao", "davos")
25 title <- c(
26   "a Annual Mean Temperature",
27   "b Annual Precipitation"
28 )
29 direction <- c(1, 1)
30 lims <- list(
31   range(values(landscape_data$bio_01), na.rm = T),
32   range(values(landscape_data$bio_12), na.rm = T)
33 )
34 fig_list_chelsa <-
35   purrr::pmap(
36     list(chelsa, pal, title, direction, lims),
37     function(df, pal, t, d, l) {
38       ggplot() +
39         stars::geom_stars(data = df) +
40         geom_sf(data = land, fill = NA, colour = "black") +
41         geom_sf(data = wg, fill = NA, colour = "black", size = 0.3) +
42         scale_fill_scico(
43           palette = pal, direction = d,
44           label = comma, na.value = NA, limits = l
45         ) +
46         coord_sf(
47           xlim = bbox[c("xmin", "xmax")],
48           ylim = bbox[c("ymin", "ymax")]
49         ) +
50         ggspatial::annotation_scale(location = "tr", width_hint = 0.4, text_cex = 1) +
51         theme_few() +
52         theme(
53           legend.position = "top",
54           title = element_text(face = "bold", size = 8),
55           legend.key.height = unit(0.2, "cm"),
56           legend.key.width = unit(1, "cm"),
57           legend.text = element_text(size = 8),
58           axis.title = element_blank(),
59           axis.text.y = element_text(angle = 90, hjust = 0.5),
60           panel.background = element_rect(fill = "lightblue"),
61           legend.title = element_blank()
62         ) +
63         labs(x = NULL, y = NULL, title = t)
64     }
65   )
66 # fig_list_chelsa <- purrr::map(fig_list_chelsa, ggplotGrob)

```

5 Climatic raster resampling

5.1 Prepare landcover

To access the classified Sentinel image, please visit: <https://code.earthengine.google.com/ec69fc4ffad32a532b25202009243d42>

```

1 # read in landcover raster location
2 landcover <- "data/landUseClassification/classifiedImage-UTM.tif"
3 # get extent
4 e <- bbox(raster(landcover))

```

```

5
6 # init resolution
7 res_init <- res(raster(landcover))
8 # res to transform to 1000m
9 res_final <- map(c(100, 250, 500, 1e3, 2.5e3), function(x) {
10   x * res_init
11 })
12
13 # use gdalutils gdalwarp for resampling transform
14 # to 1km from 10m
15 for (i in 1:length(res_final)) {
16   this_res <- res_final[[i]]
17   this_res_char <- stringr::str_pad(this_res[1], 5, pad = "0")
18   gdalUtils::gdalwarp(
19     srcfile = landcover,
20     dstfile = as.character(glue("data/landUseClassification/lc_{this_res_char}.tif")),
21     tr = c(this_res), r = "mode", te = c(e)
22   )
23 }
24
25 # read in resampled landcover raster files as a list
26 lc_files <- list.files("data/landUseClassification/", pattern = "lc", full.names = TRUE)
27 lc_data <- map(lc_files, raster)

```

93 5.2 Prepare spatial extent

```

1 # load hills
2 library(sf)
3 hills <- st_read("data/spatial/hillsShapefile/Nil_Ana_Pal.shp")
4 hills <- st_transform(hills, 32643)
5 buffer <- st_buffer(hills, 3e4) %>%
6   st_transform(4326)
7 bbox <- st_bbox(hills)

```

94 5.3 Prepare CHELSA rasters

95 Please download the CHELSA rasters from <https://chelsa-climate.org/bioclim/>

```

1 # list chelsa files
2 chelsaFiles <- list.files("data/chelsa/", full.names = TRUE, pattern = "*.tif")
3
4 # gather chelsa rasters
5 chelsaData <- purrr::map(chelsaFiles, function(chr) {
6   a <- raster(chr)
7   crs(a) <- crs(buffer)
8   a <- crop(a, as(buffer, "Spatial"))
9   return(a)
10 })
11
12 # stack chelsa data
13 chelsaData <- raster::stack(chelsaData)
14 names(chelsaData) <- c("chelsa_bio10_01", "chelsa_bio10_12")

```

5.4 Resample prepared rasters

```
1 # make resampled data
2 resamp_data <- map(lc_data, function(this_scale) {
3   rr <- projectRaster(
4     from = chelsaData, to = this_scale,
5     crs = crs(this_scale), res = res(this_scale)
6   )
7 })
8
9 # make a stars list
10 resamp_data <- map2(resamp_data, lc_data, function(z1, z2) {
11   z2[z2 == 0] <- NA
12   z2 <- append(z2, as.list(z1)) %>% map(stars::st_as_stars)
13 }) %>%
14   flatten()
```

6 Climate in Relation to Landcover

This script showcases how climatic predictors vary as a function of land cover types across our study area.

6.1 Prepare libraries

```
1 # load libs
2 library(raster)
3 library(glue)
4 library(purrr)
5 library(dplyr)
6 library(tidyr)
7
8 # plotting options
9 library(ggplot2)
10 library(ggthemes)
11 library(scico)
12
13 # get ci func
14 ci <- function(x) {
15   qnorm(0.975) * sd(x, na.rm = T) / sqrt(length(x))
16 }
```

6.2 Prepare environmental data

```
1 # read landscape prepare for plotting
2 landscape <- stack("data/spatial/landscape_resamp01km.tif")
3
4 # get proper names
5 elev_names <- c("elev", "slope", "aspect")
6 chelsa_names <- c("bio_01", "bio_12")
7
8 names(landscape) <- as.character(glue('{c(elev_names, chelsa_names, "landcover")}'))
9
10 # make duplicate stack
11 land_data <- landscape[[c("landcover", chelsa_names)]]
12
13 # convert to list
```

```

5 land_data <- as.list(land_data)
6
7 # map get values over the stack
8 land_data <- purrr::map(land_data, raster::getValues)
9 names(land_data) <- c("landcover", chelsa_names)
10
11 # conver to dataframe and round to 100m
12 land_data <- bind_cols(land_data)
13 land_data <- drop_na(land_data) %>%
14   filter(landcover != 0) %>%
15   pivot_longer(
16     cols = contains("bio"),
17     names_to = "clim_var"
18   ) # %>%
19 # group_by(landcover, clim_var) %>%
20 # summarise_all(funs = list(~mean(.), ~ci(.)))

```

6.3 Climatic variables over landcover

Figure code is hidden in versions rendered as HTML and PDF.

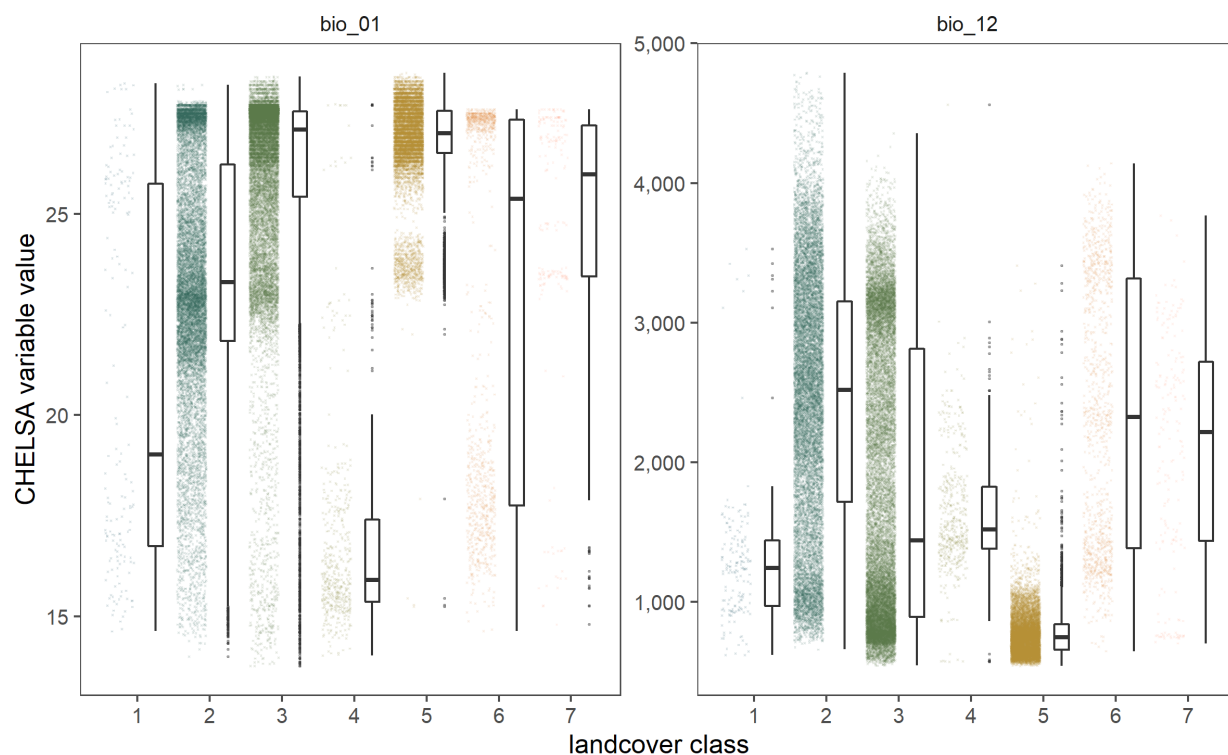


Figure 2: CHELSA climatic variables (Annual Mean Temperature on the left and Annual Precipitation on the right) are plotted as a function of landcover type. Grey points in the background represent raw data.

7 Distribution of Observer Expertise

This script plots observer expertise over time (2013-2019) as well as across land cover types.

7.1 Prepare libraries

```
# load libs
library(raster)
library(glue)
library(purrr)
library(dplyr)
library(tidyr)
library(readr)
library(scales)

# plotting libs
library(ggplot2)
library(ggthemes)
library(scico)

# get ci func
ci <- function(x) {
  qnorm(0.975) * sd(x, na.rm = T) / sqrt(length(x))
}
```

7.2 Load observer expertise scores and checklist covariates

```
# read in scores and checklist data and link
scores <- read_csv("data/03_data-obsExpertise-score.csv")
data <- read_csv("data/03_data-covars-perChklist.csv")

data <- left_join(data, scores, by = c("observer" = "observer"))
data <- dplyr::select(data, score, nSp, nSoi, landcover, year) %>%
  filter(!is.na(score))
```

7.3 Species observed in relation to observer expertise

```
# summarise data by rounded score and year
data_summary01 <- data %>%
  mutate(score = plyr::round_any(score, 0.2)) %>%
  dplyr::select(score, year, nSp, nSoi) %>%
  pivot_longer(
    cols = c("nSp", "nSoi"),
    names_to = "variable", values_to = "value"
  ) %>%
  group_by(score, year, variable) %>%
  summarise_at(vars(value), list(~ mean(.), ~ ci(.)))

# make plot and export
fig_nsp_score <-
  ggplot(data_summary01) +
  geom_jitter(
    data = data, aes(x = score, y = nSp),
    col = "grey", alpha = 0.2, size = 0.1
  ) +
  geom_pointrange(aes(
    x = score, y = mean,
    ymin = mean - ci, ymax = mean + ci,
    col = as.factor(variable)
  ))
```

```

23   ),
24   position = position_dodge(width = 0.05)
25   ) +
26   facet_wrap(~year) +
27   scale_y_log10() +
28   # coord_cartesian(ylim=c(0,50))+
29   scale_colour_scico_d(palette = "cork", begin = 0.2, end = 0.8) +
30   labs(x = "CCI", y = "Number of Species Reported") +
31   theme_few() +
32   theme(legend.position = "none")
33
34 # export figure
35 ggsave(filename = "figs/fig_nsp_score.png", width = 12, height = 7, device = png(), dpi = 300)
36 dev.off()

```

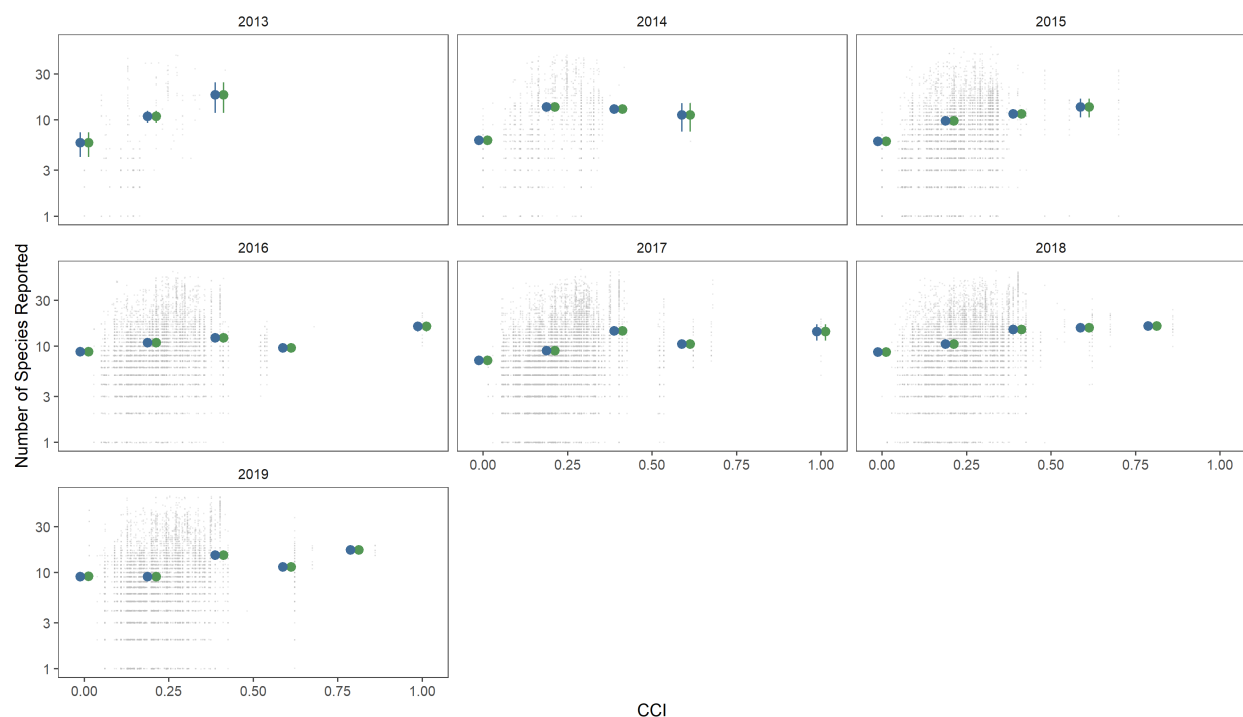


Figure 3: Total number of species (blue) and species of interest to this study (green) reported in checklists from the study area over the years 2013 – 2019, as a function of the expertise score of the reporting observer. Points represent means, with bars showing the 95% confidence intervals; data shown are for expertise scores rounded to multiples of 0.2, and the y-axis is on a log scale. Raw data are shown in the background (grey points).

7.4 Observer expertise in relation to landcover

Figure code is hidden in versions rendered as HTML or PDF.

8 Matching Effort Cutoffs with Spatial Independence Criteria

How many sites would be lost if effort distance was restricted based on spatial independence?

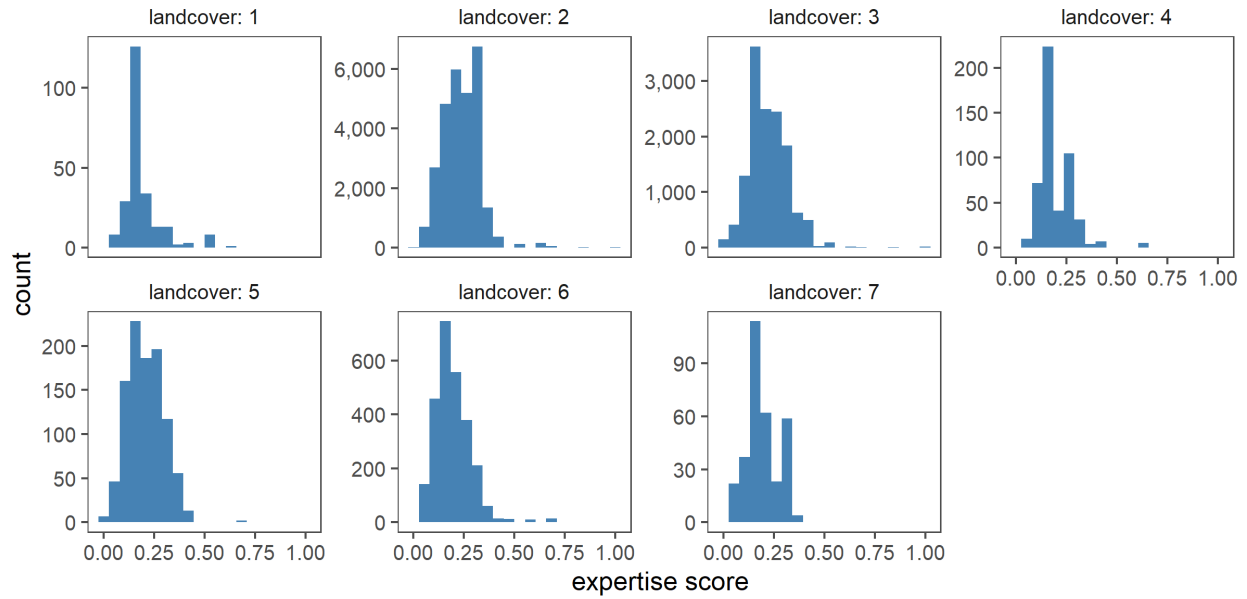


Figure 4: Distribution of expertise scores across the seven landcover classes present in the study site.

8.1 Load librairies

```

1 # load data packages
2 library(data.table)
3 library(dplyr)
4
5 # load plotting packages
6 library(ggplot2)
7 library(scico)
8 library(ggthemes)
9 library(scales)

```

8.2 Load data

```

1 # load checklist covariates
2 data <- fread("data/03_data-covars-perChklist.csv")
3
4 effort_distance_summary <- data[, effort_distance_class :=
5   cut(distance, breaks = c(
6     -1, 0.001, 0.1, 0.25,
7     0.5, 1, 2.5, 5, Inf
8   ), ordered_result = T)][,
9   .N,
10  by = effort_distance_class
11 ][
12   order(effort_distance_class)
13 ]
14
15 effort_distance_summary[
16   ,
17   prop_effort := cumsum(effort_distance_summary$N) / nrow(data)

```

18]

114 8.3 Visualise limiting effort by spatial independence limits

```
1 # plot effort distance class cumulative sum
2 fig_dist_exclusion <- ggplot(effort_distance_summary) +
3   geom_point(aes(effort_distance_class, prop_effort), size = 3) +
4   geom_path(aes(effort_distance_class, prop_effort, group = NA)) +
5   # scale_y_continuous(label=label_number(scale=0.001, accuracy = 1, suffix = "K"))+
6   scale_x_discrete(labels = c(
7     "stationary", "100m", "250m",
8     "500m", "1 km", "2.5 km", "5 km"
9   )) +
10  theme_few() +
11  theme(panel.grid = element_line(size = 0.2, color = "grey")) +
12  labs(x = "effort distance cutoff", y = "proportion of checklists")
13
14 ggsave(
15   plot = fig_dist_exclusion, "figs/fig_cutoff_effort.png",
16   height = 6, width = 8, dpi = 300
17 )
18 dev.off()
```

115 9 Spatial Thinning: A Brief Comparison of Approaches

116 9.1 Prepare libraries

```
1 # load libraries
2 library(tidyverse)
3 library(glue)
4 library(readr)
5 library(sf)
6
7 # plotting
8 library(ggthemes)
9 library(scico)
10 library(scales)
11
12 # ci func
13 ci <- function(x) {
14   qnorm(0.975) * sd(x, na.rm = T) / sqrt(length(x))
15 }
16
17 # load python libs here
18 library(reticulate)
19 # set python path
20 use_python("/usr/bin/python3")
```

117 9.2 Traditional grid-based thinning

```
1 # load the shapefile of the study area
2 wg <- st_read("data/spatial/hillsShapefile/Nil_Ana_Pal.shp") %>%
3   st_transform(32643)
4
```

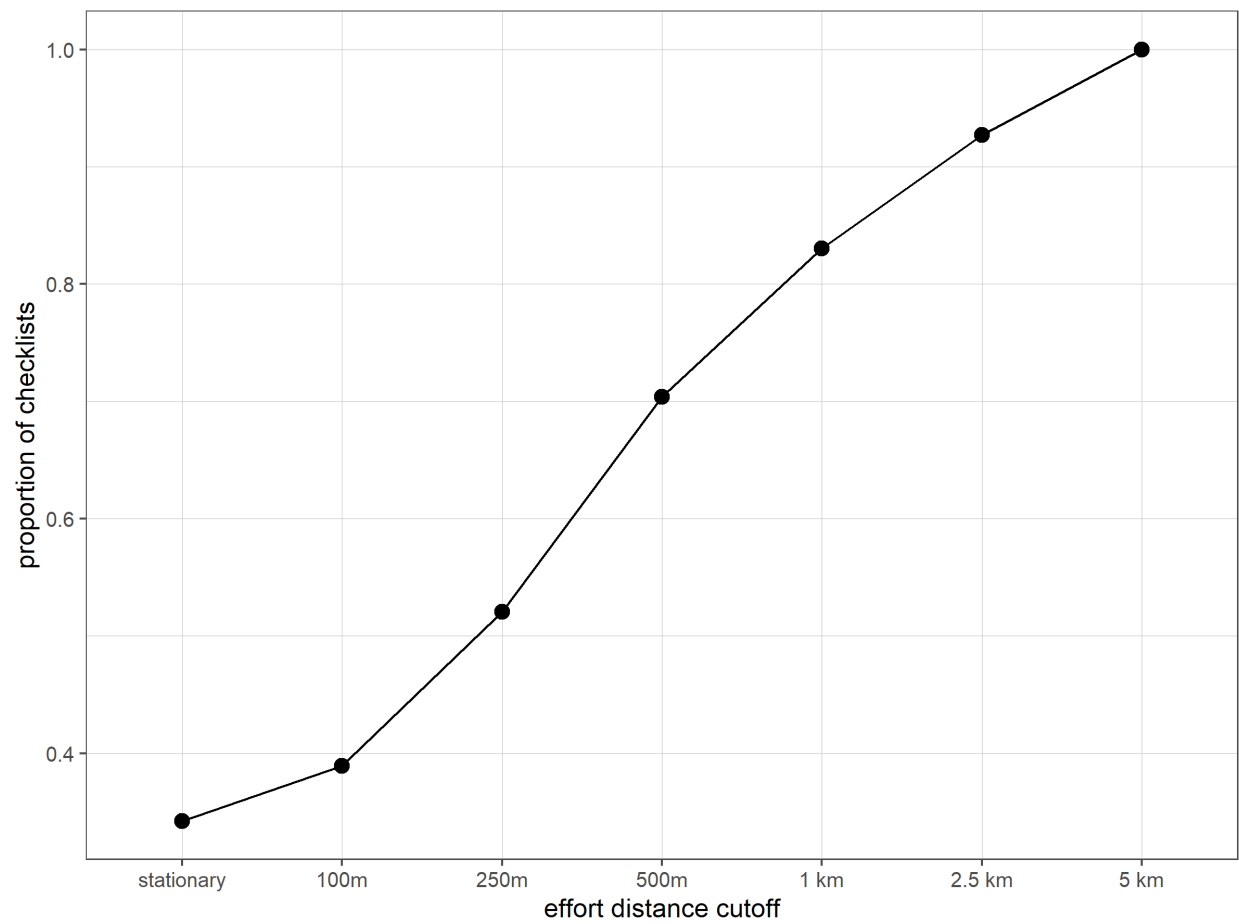


Figure 5: Proportion of checklists that are retained across the study area as a function of the distance cutoff specified. For instance, 80% of checklists are retained with a distance cutoff of 1km.

```

5 # get scales
6 # load checklist data and select one per rounded 500m coordinates
7 {
8   data <- read_csv("data/03_data-covars-perChklist.csv") %>%
9     count(longitude, latitude, name = "tot_effort")
10
11
12 # how many unique points
13 n_all_points <- nrow(data)
14 d_all_effort <- sum(data$tot_effort)
15
16 # round to different scales
17 scale <- c(100, 250, 500, 1000)
18
19 # group data by scale
20 data <- crossing(scale, data) %>%
21   group_by(scale) %>%
22   nest() %>%
23   ungroup()
24 }
25
26 # select one point per grid cell
27 data <- mutate(data, data = map2(scale, data, function(sc, df) {
28   # transform the data
29   df <- df %>%
30     st_as_sf(coords = c("longitude", "latitude")) %>%
31     `st_crs<-`(4326) %>%
32     st_transform(32643) %>%
33     bind_cols(as_tibble(st_coordinates(.))) %>%
34     mutate(
35       coordId = 1:nrow(.),
36       X_round = plyr::round_any(X, sc),
37       Y_round = plyr::round_any(Y, sc)
38     )
39
40 # make a grid
41 grid <- st_make_grid(wg, cellsize = sc)
42
43 # which cell contains which points
44 grid_contents <- st_contains(grid, df) %>%
45   as_tibble() %>%
46   rename(cell = row.id, coordId = col.id)
47
48 rm(grid)
49
50 # what's the max point in each grid
51 points_max <- left_join(df %>% st_drop_geometry(),
52   grid_contents,
53   by = "coordId"
54 ) %>%
55   group_by(cell) %>%
56   filter(tot_effort == max(tot_effort))
57
58 # get summary for max

```

```

59 max_sites <- points_max %>%
60   ungroup() %>%
61   summarise(
62     prop_points = length(coordId) / n_all_points,
63     prop_effort = sum(tot_effort) / d_all_effort
64   ) %>%
65   pivot_longer(
66     cols = everything(),
67     names_to = "variable"
68   )
69
70 # select a random point in each grid
71 points_rand <- left_join(df %>% st_drop_geometry(),
72   grid_contents,
73   by = "coordId"
74 ) %>%
75   group_by(cell) %>%
76   sample_n(size = 1)
77
78 # get summary for rand
79 rand_sites <- points_rand %>%
80   ungroup() %>%
81   summarise(
82     prop_points = length(coordId) / n_all_points,
83     prop_effort = sum(tot_effort) / d_all_effort
84   ) %>%
85   pivot_longer(
86     cols = everything(),
87     names_to = "variable"
88   )
89
90 df <- tibble(
91   grid_rand = list(rand_sites), grid_max = list(max_sites),
92   points_rand = list(points_rand), points_max = list(points_max)
93 )
94 )))
95
96 # unnest data
97 data <- unnest(data, cols = data)
98
99 # save summary as another object
100 data_thin_trad <- data %>%
101   select(-contains("points")) %>%
102   pivot_longer(
103     cols = -contains("scale"),
104     names_to = "method", values_to = "somedata"
105   ) %>%
106   unnest(cols = somedata)
107
108 # save points for later comparison
109 points_thin_trad <- data %>%
110   select(contains("points"), scale)
111
112 rm(data)

```

9.3 Network-based thinning

Load python libraries.

```
# import classic python libs
import numpy as np
import matplotlib.pyplot as plt

# libs for dataframes
import pandas as pd

# network lib
import networkx as nx

# import libs for geodata
import geopandas as gpd

# import ckdtree
from scipy.spatial import cKDTree
```

9.4 Finding modularity in proximity networks

```
# read in checklist covariates for conversion to gpd
# get unique coordinates, assign them to the df
# convert df to geo-df
chkCovars = pd.read_csv("data/03_data-covars-perChklist.csv")
ul = chkCovars[['longitude', 'latitude']].drop_duplicates(subset=['longitude', 'latitude'])
ul['coordId'] = np.arange(0, ul.shape[0])

# get effort at each coordinate
effort = chkCovars.groupby(['longitude', 'latitude']).size().to_frame('tot_effort')
effort = effort.reset_index()

# merge effort on ul
ul = pd.merge(ul, effort, on=['longitude', 'latitude'])

# make gpd and drop col from ul
ulgpd = gpd.GeoDataFrame(ul, geometry=gpd.points_from_xy(ul.longitude, ul.latitude))
ulgpd.crs = {'init': 'epsg:4326'}
# reproject spatial to 43n epsg 32643
ulgpd = ulgpd.to_crs({'init': 'epsg:32643'})
ul = pd.DataFrame(ul.drop(columns="geometry"))

# function to use ckd trees for nearest point finding
def ckd_pairs(gdfA, dist_indep):
    A = np.concatenate([np.array(geom.coords) for geom in gdfA.geometry.to_list()])
    ckd_tree = cKDTree(A)
    dist = ckd_tree.query_pairs(r=dist_indep, output_type='ndarray')
    return dist

# define scales in metres
scales = [100, 250, 500, 1000]

# function to process ckd_pairs
```



```

34 def make_modules(scale):
35     site_pairs = ckd_pairs(gdfA=ulgpd, dist_indep=scale)
36     site_pairs = pd.DataFrame(data=site_pairs, columns=['p1', 'p2'])
37     site_pairs['scale'] = scale
38     # get site ids
39     site_id = np.concatenate((site_pairs.p1.unique(), site_pairs.p2.unique()))
40     site_id = np.unique(site_id)
41     # make network
42     network = nx.from_pandas_edgelist(site_pairs, 'p1', 'p2')
43     # get modules
44     modules = list(nx.algorithms.community.greedy_modularity_communities(network))
45     # get modules as df
46     m = []
47     for i in np.arange(len(modules)):
48         module_number = [i] * len(modules[i])
49         module_coords = list(modules[i])
50         m = m + list(zip(module_number, module_coords))
51     # add location and summed sampling duration
52     unique_locs = ul[ul.coordId.isin(site_id)]
53     module_data = pd.DataFrame(m, columns=['module', 'coordId'])
54     module_data = pd.merge(module_data, unique_locs, on='coordId')
55     # add scale
56     module_data['scale'] = scale
57     return [site_pairs, module_data]
58
59
60 # run make modules on ulgpd at scales
61 data = list(map(make_modules, scales))
62
63 # extract data for output
64 tot_pair_data = []
65 tot_module_data = []
66 for i in np.arange(len(data)):
67     tot_pair_data.append(data[i][0])
68     tot_module_data.append(data[i][1])
69
70 tot_pair_data = pd.concat(tot_pair_data, ignore_index=True)
71 tot_module_data = pd.concat(tot_module_data, ignore_index=True)
72
73 # make dict of positions and array of coordinates
74 # site_id = np.concatenate((site_pairs.p1.unique(), site_pairs.p2.unique()))
75 # site_id = np.unique(site_id)
76 # locations_df = ul[ul.coordId.isin(site_id)][['longitude', 'latitude']].to_numpy()
77 # pos_dict = dict(zip(site_id, locations_df))
78
79 # output data
80 tot_module_data.to_csv(path_or_buf="data/site_modules.csv", index=False)
81 tot_pair_data.to_csv(path_or_buf="data/site_pairs.csv", index=False)
82
83 # ends here

```

9.5 Process proximity networks in R

```
1 # read in pair and module data
2 pairs <- read_csv("data/site_pairs.csv")
3 mods <- read_csv("data/site_modules.csv")
4
5 # count pairs at each scale
6 count(pairs, scale)
7 pairs %>%
8   group_by(scale) %>%
9   summarise(non_indep_pairs = length(unique(c(p1, p2))) / n_all_points)
10 count(mods, scale)
11
12 # nest by scale and add module data
13 data <- nest(pairs, data = c(p1, p2))
14 modules <- group_by(mods, scale) %>%
15   nest() %>%
16   ungroup()
17
18 # add module data
19 data <- mutate(data,
20   modules = modules$data,
21   data = map2(data, modules, function(df, m) {
22     df <- left_join(df, m, by = c("p1" = "coordId"))
23     df <- left_join(df, m, by = c("p2" = "coordId"))
24
25     df <- filter(df, module.x == module.y)
26     return(df)
27   })
28 ) %>%
29   select(-modules)
30
31 # split by module
32 data$data <- map(data$data, function(df) {
33   df <- group_by(df, module.x, module.y) %>%
34     nest() %>%
35     ungroup()
36   return(df)
37 })
```

9.6 A function that removes sites

```
1 # a function to remove sites
2 remove_which_sites <- function(pair_data) {
3   {
4     a <- pair_data %>%
5       select(p1, p2)
6
7     nodes_a_init <- unique(c(a$p1, a$p2))
8
9     i_n_d <- filter(mods, coordId %in% nodes_a_init) %>%
10       select(node = coordId, tot_effort) %>%
11       mutate(s_f_r = NA)
12
13     nodes_keep <- c()
```

```

14   nodes_removed <- c()
15 }
16
17 while (nrow(a) > 0) {
18
19   # how many nodes in a
20   nodes_a <- unique(c(a$p1, a$p2))
21
22   # get node or site efforts and arrange in ascending order
23   b <- i_n_d %>% filter(node %in% nodes_a)
24
25   for (i in 1:nrow(b)) {
26     # which node to remove
27     node_out <- b$node[i]
28     # how much tot_effort lost
29     d_n_o <- b$tot_effort[i]
30
31     # how many rows remain in a if node_out is removed?
32     a_n_o <- filter(a, p1 != node_out, p2 != node_out)
33     indep_nodes <- setdiff(nodes_a, unique(c(a_n_o$p1, a_n_o$p2, node_out)))
34
35     # how much sampling effort made spatially independent
36     indep_sampling <- filter(b, node %in% indep_nodes) %>%
37       summarise(tot_effort = sum(tot_effort)) %>%
38       .$tot_effort
39
40     # message(glue::glue('{node_out} removal frees {indep_sampling} m'))
41     # sampling freed by sampling lost
42     b$s_f_r[i] <- indep_sampling / d_n_o
43   }
44
45   # arrange node data by decreasing sfr and increasing tot_effort
46   # highest tot_effort nodes are processed last
47   b <- arrange(b, -s_f_r, tot_effort)
48
49   nodes_removed <- c(nodes_removed, b$node[1])
50
51   # remove pairs of nodes containing the highest sfr node in b
52   a <- filter(a, p1 != b$node[1], p2 != b$node[1])
53
54   nodes_keep <- c(nodes_keep, setdiff(nodes_a, unique(c(a$p1, a$p2, nodes_removed))))
55 }
56
57 message(glue::glue("keeping {length(nodes_keep)} of {length(nodes_a_init)}"))
58
59 # node_status <- tibble(nodes = c(nodes_keep, nodes_removed),
60 #                         status = c(rep(TRUE, length(nodes_keep)),
61 #                                     rep(FALSE, length(nodes_removed))))
62
63 return(as.integer(nodes_removed))
64 }

```

9.7 Removing non-independent sites

```
1 # remove 5km and 2.5km scale
2 data <- data %>% filter(scale <= 1000)
3 # run select sites on the various modules
4 sites_removed <- map(data$data, function(df) {
5   remove_sites <- unlist(purrr::map(df$data, remove_which_sites))
6 })
7
8 # save as rdata
9 save(sites_removed, file = "data/data_network_sites_removed.rdata")
10
11 # get python sites
12 ul <- py$ul
13
14 load("data/data_network_sites_removed.rdata")
15
16 # subset sites
17 data <- mutate(data,
18   data = map(sites_removed, function(site_id) {
19     as_tibble(filter(ul, !coordId %in% site_id))
20   })
21 )
22
23 # which points are kept
24 points_thin_net <- mutate(data,
25   data = map(data, function(df) {
26     df <- df %>%
27       select("longitude", "latitude") %>%
28       st_as_sf(coords = c("longitude", "latitude")) %>%
29       `st_crs<=`(4326) %>%
30       st_transform(32643) %>%
31       bind_cols(as_tibble(st_coordinates(.))) %>%
32       st_drop_geometry()
33   })
34 )
35
36 # get metrics for method
37 data_thin_net <- unnest(data, cols = "data") %>%
38   group_by(scale) %>%
39   summarise(
40     prop_points = length(coordId) / n_all_points,
41     prop_effort = sum(tot_effort) / d_all_effort
42   ) %>%
43   mutate(method = "network") %>%
44   pivot_longer(
45     cols = -one_of(c("method", "scale")),
46     names_to = "variable"
47   )
```

9.8 Measuring method fallibility

How many points, at different spatial scales, remain after the application of each method?

9.9 Prepare data for Python

```
1
2 # get points by each method
3 points_list <- append(points_thin_net$data, values = append(
4   points_thin_trad$points_rand,
5   points_thin_trad$points_max
6 ))
7
8 # get scales as list
9 scales_list <- list(100, 250, 500, 1000, rep(c(100, 250, 500, 1000), 2)) %>% flatten()
10
11 # send to python
12 py$points_list <- points_list
13 py$scales_list <- scales_list
```

9.10 Count props under threshold in Python

```
1 # a function to convert to gpd
2 def make_gpd(df):
3     df = gpd.GeoDataFrame(df, geometry=gpd.points_from_xy(df.X, df.Y))
4     df.crs = {'init' : 'epsg:32643'}
5     return df
6
7
8 # function for mean nnd
9 # function to use ckdtrees for nearest point finding
10 def ckd_test(gdfA, gdfB, dist_indep):
11     A = np.concatenate([np.array(geom.coords) for geom in gdfA.geometry.to_list()])
12     #simplified_features = simplify_roads(gdfB)
13     B = np.concatenate([np.array(geom.coords) for geom in gdfB.geometry.to_list()])
14     #B = np.concatenate(B)
15     ckd_tree = cKDTree(B)
16     dist, idx = ckd_tree.query(A, k=[2])
17     dist_diff = list(map(lambda x: x - dist_indep, dist))
18     mean_dist_diff = np.asarray(dist_diff).mean()
19     return mean_dist_diff
20
21
22 # apply to all data
23 points_list = list(map(make_gpd, points_list))
24
25 # get nnb all data
26 mean_dist_diff = list(map(ckd_test, points_list, points_list, scales_list))
```

9.11 Plot metrics for different methods

```
1 # combine the thinning metrics data
2 data_plot <- bind_rows(data_thin_net, data_thin_trad)
3
4 # get data for mean distance
5 data_thin_compare <- tibble(
6   scale = unlist(scales_list),
7   method = c(
8     rep("network", 4),
```

```

9     rep("grid_rand", 4),
10    rep("grid_max", 4)
11  ),
12  `mean NND - buffer (m)` = unlist(py$mean_dist_diff)
13 ) %>%
14   pivot_longer(
15     cols = "mean NND - buffer (m)",
16     names_to = "variable"
17   )
18
19 # bind rows with other data
20 data_plot <- bind_rows(data_plot, data_thin_compare)
21
22 # plot results
23 fig_spatial_thinning <-
24   ggplot(data_plot) +
25     geom_vline(xintercept = scale, lty = 3, colour = "grey", lwd = 0.4) +
26     geom_line(aes(x = scale, y = value, col = method)) +
27     geom_point(aes(x = scale, y = value, col = method, shape = method)) +
28     facet_wrap(~variable, scales = "free") +
29     scale_shape_manual(values = c(1, 2, 0)) +
30     scale_x_continuous(breaks = scale) +
31     scale_y_continuous() +
32     scale_colour_scico_d(palette = "batlow", begin = 0.2, end = 0.8) +
33     theme_few() +
34     theme(legend.position = "top") +
35     labs(x = "buffer distance (m)")
36
37 # save
38 ggsave(fig_spatial_thinning,
39   filename = "figs/fig_spatial_thinning_02.png", width = 10, height = 4,
40   dpi = 300
41 )
42 dev.off()

```

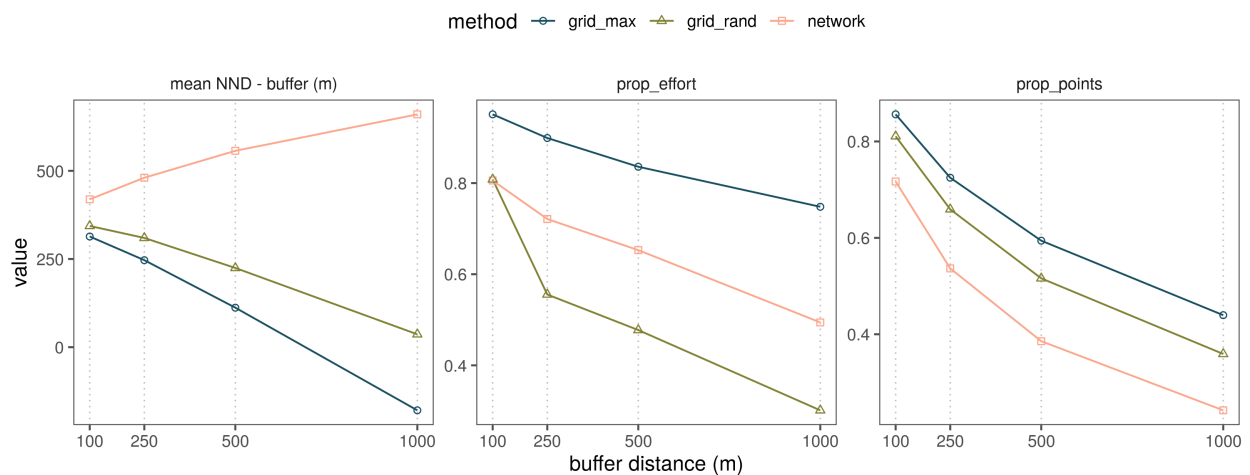


Figure 6: Three spatial thinning approaches were tested and we show that the largest proportion of effort as well as detections could be retained using a traditional grid based approach

129 **10 Predicting Species-specific Occupancy**

130 This supplement plots species-specific probabilities of occupancy as a function of significant environmental predictors.

131 **10.1 Prepare libraries**

```
1 # to load data
2 library(readxl)
3
4 # to handle data
5 library(dplyr)
6 library(readr)
7 library(forcats)
8 library(tidyr)
9 library(purrr)
10 library(stringr)
11
12 # plotting
13 library(ggplot2)
14 library(patchwork)
```

132 **10.2 Read data**

```
1 # read data
2 data <- read_csv("data/results/data_occupancy_predictors.csv")
3
4 # drop na
5 data <- select(
6   data,
7   -ci
8 ) %>%
9   drop_na() %>%
10  nest(data = c(predictor, m_group, seq_x, mean, scale))
```