Lab 3.3: Regression Models for Forecasting

In this Lab we fit a simple linear model to every pixel in the SST data set, and we use these models to predict SST for a month in which we have no SST data. The models will simply contain an intercept and a single covariate, namely the Southern Oscillation Index (SOI). The SOI data we use here are supplied with **STRbook** and were retrieved from `https://www.esrl.noaa.gov/psd/gcos_wgsp/Timeseries/SOI/`.

For this Lab we need the usual data-wrangling and plotting packages, as well as the packages **broom** and **purrr** for fitting and predicting with multiple models simultaneously.

```
library("broom")
library("dplyr")
library("ggplot2")
library("STRbook")
library("purrr")
library("tidyr")
```

In the first section of this Lab we tidy up the data to obtain the SST data frame from the raw data. You may also skip this section by loading `SST_df` from **STRbook**, and fast-forwarding to the section that is concerned with fitting the data.

```
data("SST_df", package = "STRbook")
```

Tidying Up the Data

The first task in this Lab is to wrangle the SST data into a long-format data frame that is amenable to linear fitting and plotting. Recall from Lab 2.3 that the SST data are provided in three data frames, one describing the land mask, one containing the SST values in wide format, and one containing the coordinates.

```
data("SSTlandmask", package = "STRbook")
data("SSTdata", package = "STRbook")
data("SSTlonlat", package = "STRbook")
```

We first combine the land mask data with the coordinates data frame.

```
lonlatmask_df <- data.frame(cbind(SSTlonlat, SSTlandmask))
names(lonlatmask_df) <- c("lon", "lat", "mask")
```

Then we form our SST data frame in wide format by attaching `SSTdata` to the coordinate-mask data frame.

```
SSTdata <- cbind(lonlatmask_df, SSTdata)
```

Finally, we use `gather` to put the data frame into long format.

```
SST_df <- gather(SSTdata, date, sst, -lon, -lat, -mask)
```

Our data frame now contains the SST data, but the `date` field contains as entries `V1`, `V2`, . . . , which were the names of the columns in `SSTdata`.

```
SST_df %>% head(3)

##   lon lat mask date    sst
## 1 124 -29    1   V1 -0.363
## 2 126 -29    1   V1 -0.285
## 3 128 -29    1   V1 -0.192
```

We replace this `date` field with two fields, one containing the month and one containing the year. We can do this by first creating a mapping table that links `V1` to January 1970, `V2` to February 1970, and so on, and then merging using `left_join`.

```
date_grid <- expand.grid(Month = c("Jan", "Feb", "Mar", "Apr",
                                   "May", "Jun", "Jul", "Aug",
                                   "Sep", "Oct", "Nov", "Dec"),
                         Year = 1970:2002,
                         stringsAsFactors =  FALSE)
date_grid$date <- paste0("V", 1:396)
SST_df <- left_join(SST_df, date_grid) %>%
          select(-date)
```

For good measure, we also add in the `date` field again but this time in month–year format.

```
SST_df$date <- paste(SST_df$Month, SST_df$Year)
SST_df %>% head(3)

##   lon lat mask    sst Month Year     date
## 1 124 -29    1 -0.363   Jan 1970 Jan 1970
## 2 126 -29    1 -0.285   Jan 1970 Jan 1970
## 3 128 -29    1 -0.192   Jan 1970 Jan 1970
```

Next, we set SST data that are coincident with land locations to `NA`:

```
SST_df$sst<- ifelse(SST_df$mask == 0, SST_df$sst, NA)
```

Our SST data frame is now in place. The following code plots a series of SSTs leading up to the 1997 El Niño event.

```
g <- ggplot(filter(SST_df, Year == 1997 &  # subset by month/year
                      Month %in% c("Apr","Aug","Jun","Oct"))) +
    geom_tile(aes(lon, lat,
                  fill = pmin(sst, 4))) +  # clamp SST at 4deg
    facet_wrap(~date, dir = "v") +         # facet by date
    fill_scale(limits = c(-4, 4),          # color limits
               name = "degC") +            # legend title
    theme_bw() + coord_fixed()             # fix scale and theme
```

Now we need to add the SOI data to the SST data frame. The SOI time series is available as a 14-column data frame, with the first column containing the year, the next 12 columns containing the SOI for each month in the respective year, and the last column containing the mean SOI for that year. In the following, we remove the annual average from the data frame, which is in wide format, and then put it into long format using `gather`.

```
data("SOI", package = "STRbook")
SOI_df <- select(SOI, -Ann) %>%
          gather(Month, soi, -Year)
```

Finally, we use `left_join` to merge the SOI data and the SST data.

```
SST_df <- left_join(SST_df, SOI_df,
                    by = c("Month", "Year"))
```

Fitting the Models Pixelwise

In this section we fit linear time-series models to the SSTs in each pixel using data up to April 1997. We first create a data frame containing the SST data between January 1970 and April 1997.

```
SST_pre_May <- filter(SST_df, Year <= 1997) %>%
               filter(!(Year == 1997 &
                        Month %in% c("May", "Jun", "Jul",
                                     "Aug", "Sep", "Oct",
                                     "Nov", "Dec")))
```

Next, as in Lab 3.2, we use **purrr** and **broom** to construct a nested data frame that contains a linear model fitted to every pixel. We name the function that fits the linear model at a single pixel to the data over time as `fit_one_pixel`.

```
fit_one_pixel <- function(data)
                 mod <- lm(sst ~ 1 + soi, data = data)
```

```
pixel_lms <- SST_pre_May %>%
             filter(!is.na(sst)) %>%
             group_by(lon, lat) %>%
             nest() %>%
             mutate(model = map(data, fit_one_pixel)) %>%
             mutate(model_df = map(model, tidy))
```

The string of commands above describes an operation that is practically identical to what we did in Lab 3.2. We take the data, filter them to remove missing data, group by pixel, create a nested data frame, fit a model to each pixel, and extract a data frame containing information on the linear fit by pixel. The first three records of the nested data frame are as follows.

```
pixel_lms %>% head(3)

## # A tibble: 3 x 5
##     lon   lat data               model    model_df
##   <dbl> <dbl> <list>             <list>   <list>
## 1   154   -29 <tibble [328 x 6]> <S3: lm> <tibble [2 x 5]>
## 2   156   -29 <tibble [328 x 6]> <S3: lm> <tibble [2 x 5]>
## 3   158   -29 <tibble [328 x 6]> <S3: lm> <tibble [2 x 5]>
```

To extract the model parameters from the linear-fit data frames, we use **unnest**:

```
lm_pars <- pixel_lms %>%
           unnest(model_df)
```

For each pixel, we now have an estimate of the intercept and the effect associated with the covariate soi, as well as other information such as the p-values.

```
head(lm_pars, 3)

## # A tibble: 3 x 7
##     lon   lat term     estimate std.error statistic  p.value
##   <dbl> <dbl> <chr>       <dbl>     <dbl>     <dbl>    <dbl>
## 1   154   -29 (Interc~   0.132     0.0266      4.96  1.13e-6
## 2   154   -29 soi        0.0277    0.0223      1.24  2.16e-1
## 3   156   -29 (Interc~   0.0365    0.0262      1.40  1.64e-1
```

We can plot spatial maps of the intercept and the regression coefficient associated with soi directly. We first merge this data frame with the coordinates data frame using **left_join**, which also contains land pixels. In this way, regression coefficients over land pixels are marked as NA, which is appropriate.

```
lm_pars <- left_join(lonlatmask_df, lm_pars)
```

The following code plots the spatial intercept and the spatial regression coefficient associated with `soi`.

```
g2 <- ggplot(filter(lm_pars, term == "(Intercept)" | mask == 1)) +
    geom_tile(aes(lon, lat, fill = estimate)) +
    fill_scale() +
    theme_bw() + coord_fixed()

g3 <- ggplot(filter(lm_pars, term == "soi" | mask == 1)) +
    geom_tile(aes(lon, lat, fill = estimate)) +
    fill_scale() +
    theme_bw() + coord_fixed()
```

Predicting SST Pixelwise

We now use the linear models at the pixel level to predict the SST in October 1997 using the SOI index for that month. The SOI for that month is extracted from `SOI_df` as follows.

```
soi_pred <- filter(SOI_df, Month == "Oct" & Year == "1997") %>%
            select(soi)
```

We next define the function that carries out the prediction at the pixel level. The function takes a linear model `lm` and the SOI at the prediction date `soi_pred`, runs the **`predict`** function for this date, and returns a data frame containing the prediction and the prediction standard error.

```
predict_one_pixel <- function(lm, soi_pred) {
    predict(lm,                              # linear model
            newdata = soi_pred,              # pred. covariates
            interval = "prediction") %>%     # output intervals
    data.frame() %>%                         # convert to df
    mutate(se = (upr-lwr)/(2 * 1.96)) %>%    # comp pred. se
    select(fit, se)                          # return fit & se
  }
```

Prediction proceeds at each pixel by calling **`predict_one_pixel`** on each row in our nested data frame `pixel_lms`.

```
SST_Oct_1997 <- pixel_lms %>%
                mutate(preds = map(model,
                                   predict_one_pixel,
                                   soi_pred = soi_pred)) %>%
                unnest(preds)
```

We have unnested the `preds` data frame above to save the fit and prediction standard error as fields in the `SST_Oct_1997` data frame. You can type `SST_Oct_1997 %>% head(3)` to have a look at the first three records. It is straightforward to plot the prediction and prediction standard error from `SST_Oct_1997`. This is left as an exercise for the reader.