European rainfall quantile mapping exercise

Jordan Richards

2023-11-17

Dataset

- The data consists of:
  - hourly summer rainfall (mm) over 500 grid-boxes (0.25deg by 0.25deg) across Western Europe (2021-2022).
  - we will treat grid-box as point location.
- collection of 11 relevant meteorological and orographical covariates at each site.
- Obtained from ERA5:
  - https://cds.climate.copernicus.eu/cdsapp#!/dataset/reanalysis-era5-single-levels
  - we will use a GNN model to map high quantiles of hourly rainfall, as well as estimate the probability of extreme rainfall at a single site.

Load data

```r
load("../Data/EuroRain.Rdata")
dim(Y)
```

```
## [1] 4416  500
```

```r
dim(X)
```

```
## [1] 4416  500   11
```

Predictors

- The predictors are:
  - meteorological: air temperature at 2m, mean sea level pressure, surface pressure, ozone, 10m u- and v-wind speed components
  - orographical: angle, isotropy, land-sea mask, slope, standard deviation

```
print(cov_names[1:5])
```

```
## [1] "t2m"  "msl"  "sp"   "tco3" "u10"
```

```
print(cov_names[5:11])
```

```
## [1] "u10"  "v10"  "anor" "isor" "lsm"  "slor" "sdor"
```

Data normalisation

- We first scale the input data to improve the numerical stability of training

```
#Normalise inputs
X_scaled <- X
for(i in 1:dim(X)[3]){
  temp <- X[,,i]
  m <- mean( temp,na.rm=T)
  s <- sd( temp,na.rm=T)
  temp <- ( temp-m)/s
  X_scaled[,,i] <- temp
}
```

Get some validation data

```
#Normalise inputs
set.seed(1)
validation.inds <- sample(1:nrow(Y), nrow(Y)/5) #20%
X.valid <- X_scaled[validation.inds,,]
Y.valid <-  Y[validation.inds,]
X.train <- X_scaled[-validation.inds,,]
Y.train <-  Y[-validation.inds,]
```

Response

- We focus on estimating the 90% quantile of $Y|\mathbf{X}$, which is the hourly rainfall. This can be done using the tilted loss.

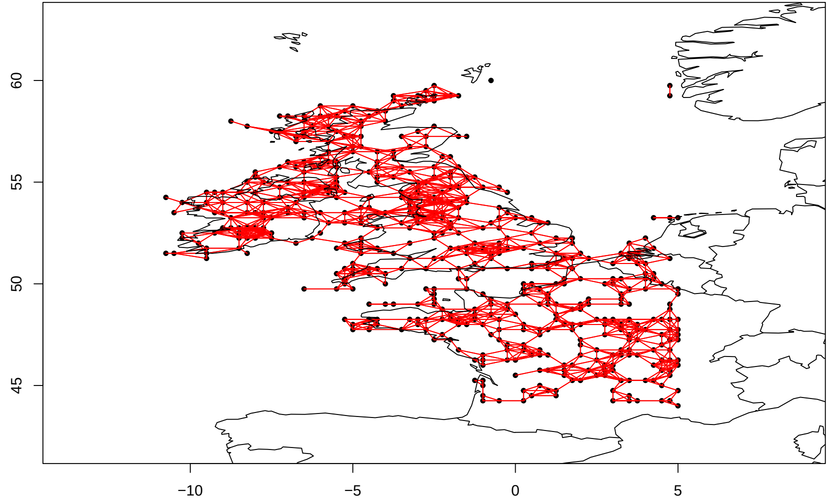
```
quant.level<-0.9

tilted_loss <- function( y_true, y_pred) {
    K <- backend()



    error = y_true - y_pred
    return(
      100*K$mean(K$maximum(quant.level*error,
                            (quant.level-1)*error))
      )
  }
```

Building a graph

- We can build a graph for our data using the location coordinates. The graph structure will be stored in an adjacency matrix. This matrix will have entries 1 if sites are within 75km of each other.

```
dist<-fields::rdist.earth(coords,miles=F)
cut.off.dist <- 75
A <- dist
A[dist>cut.off.dist]=0; A[dist <= cut.off.dist]=1
diag(A)=0
```

Building a graph



Graph layers

```
spk <<- reticulate::import("spektral")
print(names(spk$layers)[1:9])
```

[1] "activations" "agnn_conv" "AGNNConv" "appnp_conv" "APPNPConv"
[6] "arma_conv" "ARMAConv" "base" "censnet_conv"

We can create a custom Keras layer with an embedded spektral layer. Here we used a graph convolutional layer with trainable skip connection.

See also https://graphneural.network/layers/convolution/

Custom graph layers

```
layer_graph_conv <- function(
      object,
      channels,
      activation = NULL,
      use_bias = TRUE,
      kernel_initializer = 'glorot_uniform',
      bias_initializer = 'zeros',
      kernel_regularizer = NULL,
      bias_regularizer = NULL,
      activity_regularizer = NULL,
      kernel_constraint = NULL,
      bias_constraint = NULL,
      name=NULL,
      ...)
{... #REPLACE WITH FOLLOWING}
```

Custom graph layers (2)

```
args <- list(channels = as.integer(channels),
          activation = activation,
          use_bias = use_bias,
          kernel_initializer = kernel_initializer,
          bias_initializer = bias_initializer,
          kernel_regularizer = kernel_regularizer,
          bias_regularizer = bias_regularizer,
          activity_regularizer = activity_regularizer,
          kernel_constraint = kernel_constraint,
          bias_constraint = bias_constraint,
          name=name
)
keras::create_layer(spk$layers$GCSConv, object, args)
```

Keras GNN

We can now build a Keras model as we have done previously, just by swapping layer_dense (or equivalent) with layer_graph_conv.

However, we need an extra input to the graph layer that includes information about the graph itself. This extra input depends on the type of layer. For GCGSconv, we need the normalised adjacency matrix.

```
ML<-spk$utils$convolution$normalized_adjacency(A)
```

Keras GNN

```
library(keras)

input.lay <- layer_input(shape = dim(X)[2:3])

hidden.lay1<- list(input.lay, ML) %>%
  layer_graph_conv(channels = 32,   activation = "relu")

hidden.lay2<- list(hidden.lay1, ML) %>%
  layer_graph_conv(channels = 32,   activation = "relu")
```

Keras GNN

```
library(keras)
# I want strictly positive quantiles only,
# so lets apply an exponential transformation
output.lay<- hidden.lay2 %>%
  layer_dense(units = 1,   activation = "exponential")

 model <- keras_model(
    inputs = c(input.lay),
    outputs = c(output.lay)
  )
```

Compiling

We will compile with the adam optimiser and our quantile loss function

```
model %>% compile(
  loss = tilted_loss,
  optimizer = "adam"
)
```

Fitting the model

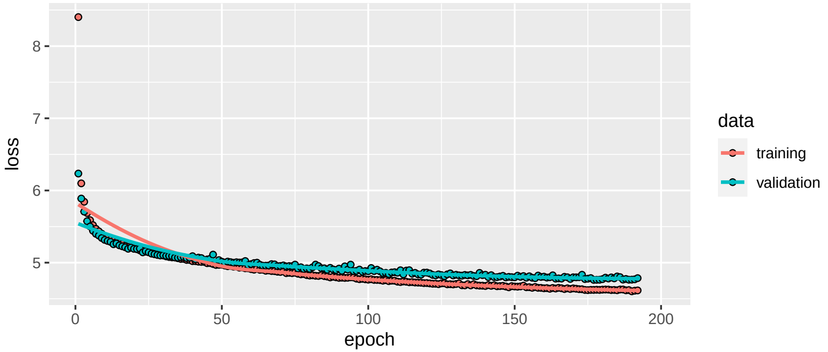
I'll also shuffle the training data to break up any temporal dependence. This will help the training procedure.

```
history <- model %>% fit(
  x = X.train, y = Y.train, shuffle=T,
  epochs = 200, batch_size = 32,
  validation_data = list(
    X.valid, Y.valid),
  callbacks =  list(
    callback_model_checkpoint(filepath = "model_weights",
      verbose=0, monitor="val_loss",
      save_best_only = TRUE, save_weights_only = TRUE),
      callback_early_stopping(monitor = "val_loss",
                              min_delta = 0, patience = 15)),
  verbose=0
)
model <- load_model_weights_tf(model,
                               filepath="model_weights")
```

Fitting the model

- Check for overfitting

```
plot(history)
```



Predictions

```r
qhat <- model %>% predict(X_scaled)
```

```
## 138/138 - 1s - 1s/epoch - 10ms/step
```

```r
mean(qhat[,,1] > Y)
```

```
## [1] 0.8925417
```

Weighted adjacency

- Let's try weighting the adjacency matrix, and see if that provides better predictions. We will use a Gaussian weighting kernel with range 50

```
A2 <- exp(-(dist/50)^2)
diag(A2) <- 0
A2[dist > cut.off.dist] <- 0
ML2<-spk$utils$convolution$normalized_adjacency(A2)
```

New model

```
library(keras)
input.lay2 <- layer_input(shape = dim(X)[2:3])
hidden.lay21<- list(input.lay2, ML2) %>%
  layer_graph_conv(channels = 32,   activation = "relu")
hidden.lay22<- list(hidden.lay21, ML2) %>%
  layer_graph_conv(channels = 32,   activation = "relu")
output.lay2 <- hidden.lay22 %>%
  layer_dense(units=1, activation = 'exponential')


 model2 <- keras_model(
    inputs = c(input.lay2),
    outputs = c(output.lay2)
  )
```

New model fitting

```r
model2 %>% compile(
  loss = tilted_loss,
  optimizer = "adam"
)
```

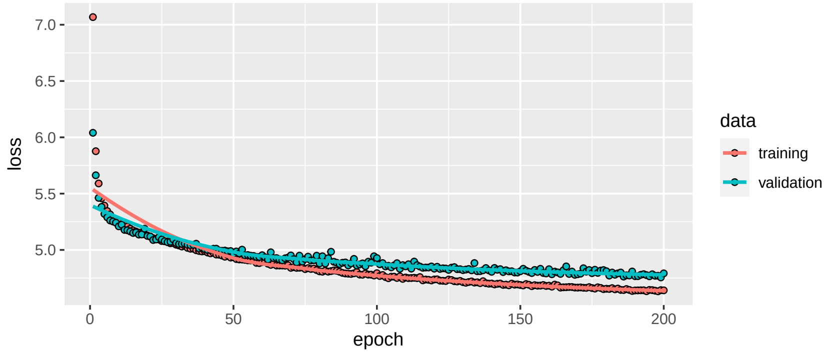
New model fitting

```
history2 <- model2 %>% fit(
  x = X.train, y = Y.train, shuffle=T,
  epochs = 200, batch_size = 32,
  validation_data = list(
    X.valid, Y.valid),
 callbacks =  list(
    callback_model_checkpoint(filepath = "model_weights",
      verbose=0, monitor="val_loss",
      save_best_only = TRUE, save_weights_only = TRUE),
      callback_early_stopping(monitor = "val_loss",
                              min_delta = 0, patience = 15)),
 verbose=0
)
model2 <- load_model_weights_tf(model2,
                                 filepath="model_weights")
```

Fitting the model

- Check for overfitting

```
plot(history2)
```



Comparison

```r
qhat2 <- model2 %>% predict(X_scaled)
```

```
## 138/138 - 2s - 2s/epoch - 11ms/step
```

```r
mean(qhat2[,,1]>= Y)
```

```
## [1] 0.8959579
```

```r
print(tilted_loss(Y.valid,predict(model,X.valid)[,,1]))
```

```
## 28/28 - 0s - 493ms/epoch - 18ms/step
## tf.Tensor(4.760883725181087, shape=(), dtype=float64)
```

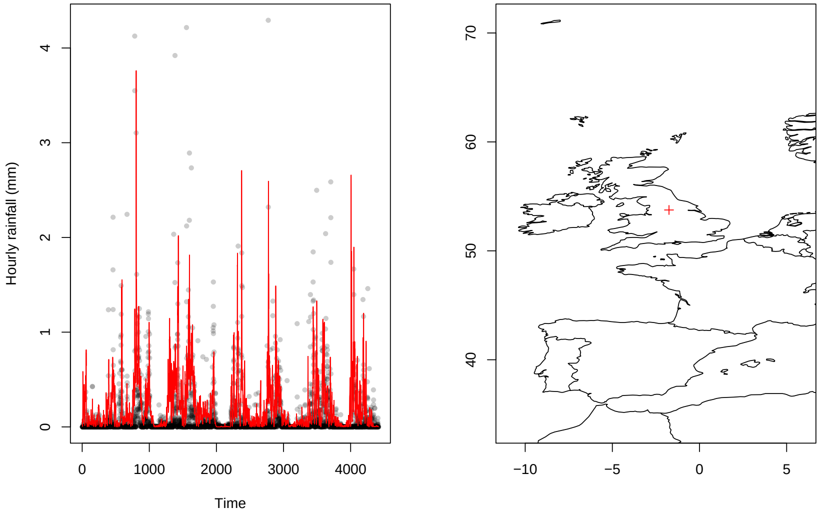
```r
print(tilted_loss(Y.valid,predict(model2,X.valid)[,,1]))
```

```
## 28/28 - 0s - 485ms/epoch - 17ms/step
## tf.Tensor(4.755669896862342, shape=(), dtype=float64)
```

Lower validation loss when using the weighted A matrix.

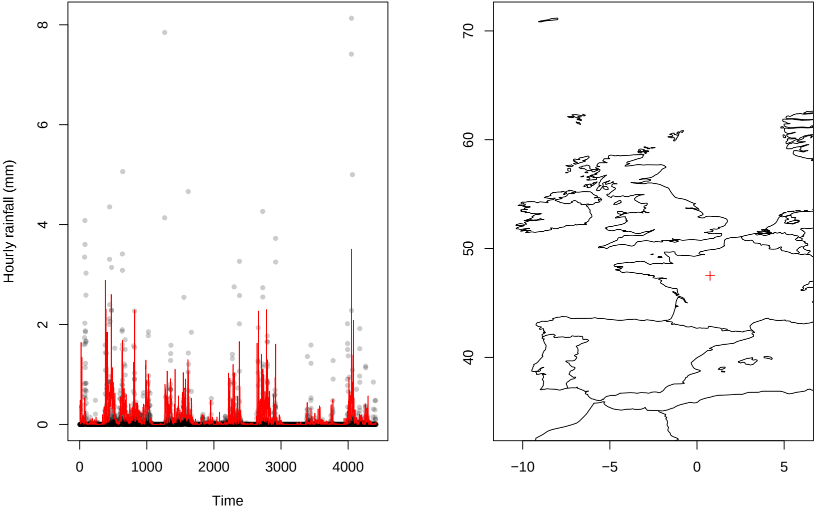
Predictions

- Check predictions

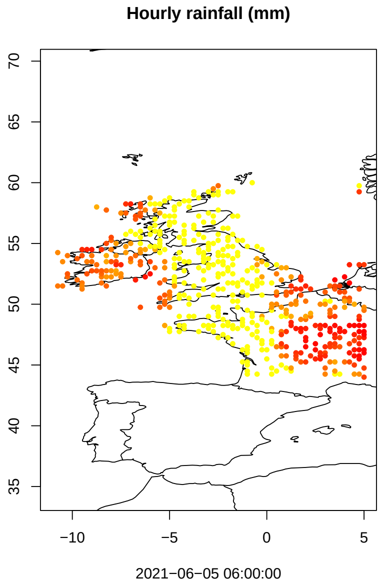


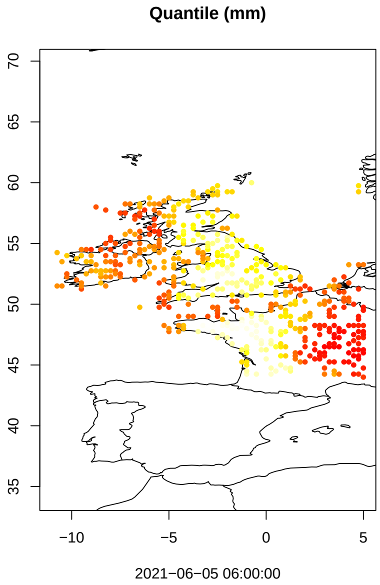
Predictions

- Check predictions



Prediction maps





Prediction maps