Supplementary

A Appendix: code

The ICEnet was implemented using the `R` language version of the `Keras` package (Chollet et al., 2017).

The constraints used for the ICEnet are implemented using a custom layer, which is shown in Listing 1. This custom layer works by either estimating the squared third differences (smoothing) or the squared value of negative first differences (monotonicity) (or both of these) and then adds the value of these penalties to the loss of the model. The monotonicity constraint can also be reversed to enforce monotonically decreasing constraints.

The input processing for the ICEnet is shown in the following two listings in this section. The first of these, Listing 2, shows the input processing for the prediction part of the network and the second of these, Listing 3, shows the input processing to produce the ICE output of the ICEnet. The main idea of Listing 3 is to define a matrix of covariates, where all of the covariates entered into the prediction network are held constant, and only those covariates which relate to the ICE outputs vary. These are varied in turn for each covariate for which an ICE must be produced.

Finally, Listing 4 shows the rest of the ICEnet. A single FCN is used to derive predictions from the inputs. This same network is used to make predictions for the ICE outputs, using the `time_distributed` layer. The constraints are then enforced on each of the ICE outputs.

Listing 1: Code for the Combined Loss Layer in R.

```
# Custom combined loss layer
CombinedLossLayer <- Layer(
  classname = "CombinedLossLayer",
  initialize = function(lambda_smooth = 0, lambda_mono = 0, direction = 1) {
    super()$`__init__`()
    self$lambda_smooth <- lambda_smooth
    self$lambda_mono <- lambda_mono
    self$direction <- direction
  },
  call = function(inputs, ...) {
    input_shape <- k_shape(inputs)
    #total_loss <- k_constant(0, dtype = 'float32')

    # Smoothing loss
    if (self$lambda_smooth > 0) {
      input1 <- tf$slice(inputs, begin = c(0L, 0L), size = c(-1L, input_shape[2] - 3L))
      input2 <- tf$slice(inputs, begin = c(0L, 1L), size = c(-1L, input_shape[2] - 3L))
      input3 <- tf$slice(inputs, begin = c(0L, 2L), size = c(-1L, input_shape[2] - 3L))
      input4 <- tf$slice(inputs, begin = c(0L, 3L), size = c(-1L, input_shape[2] - 3L))

      diff1 <- input2 - input1
      diff2 <- input3 - input2
      diff3 <- input4 - input3

      third_diff <- diff3 - 2 * diff2 + diff1

      squared_diff3 <- k_sum(k_square(third_diff), axis=2)
      smooth_loss <- k_mean(squared_diff3, axis=1) * self$lambda_smooth
      #total_loss = total_loss + smooth_loss
      self$add_loss(smooth_loss)
    }

    # Monotonicity loss
    if (self$lambda_mono > 0) {
      input1 <- tf$slice(inputs, begin = c(0L, 0L), size = c(-1L, input_shape[2] - 1L))
      input2 <- tf$slice(inputs, begin = c(0L, 1L), size = c(-1L, input_shape[2] - 1L))

      first_diff <- input2 - input1
      adjusted_diff <- first_diff * self$direction
      penalty <- k_sum(k_square(k_maximum(adjusted_diff, 0)), axis=2)
      mono_loss <- k_mean(penalty, axis=1) * self$lambda_mono
      #total_loss = total_loss + mono_loss
      self$add_loss(mono_loss)
    }

    # Return the input tensor
    inputs
  }
)
```

Listing 2: Code for the Input Processing in R.

```
embed_layer <- function(input, input_dim, output_dim, name) {
  input %>%
    layer_embedding(input_dim=input_dim, output_dim=output_dim, input_length=1, name=name) %>%
    layer_flatten()
}

  # Define inputs
  inputs <- list(
    Exposure = layer_input(shape = 1, dtype = 'float32', name = 'Exposure'),
    VehGas = layer_input(shape = 1, dtype = 'int32', name = 'VehGas'),
    VehBrand = layer_input(shape = 1, dtype = 'int32', name = 'VehBrand'),
    Region = layer_input(shape = 1, dtype = 'int32', name = 'Region'),
    Area = layer_input(shape = 1, dtype = 'int32', name = 'Area'),
    DrivAge = layer_input(shape = 1, dtype = 'float32', name = 'DrivAge'),
    VehAge = layer_input(shape = 1, dtype = 'float32', name = 'VehAge'),
    BonMal = layer_input(shape = 1, dtype = 'float32', name = 'BonMal'),
    Density = layer_input(shape = 1, dtype = 'float32', name = 'Density'),
    VehPower = layer_input(shape = 1, dtype = 'float32', name = 'VehPower')
  )

  # Define embeddings
  embeddings <- list(
    VehGas_embed = embed_layer(inputs$VehGas, 2, 5, 'VehGas_embed'),
    VehBrand_embed = embed_layer(inputs$VehBrand, 11, 5, 'VehBrand_embed'),
    Region_embed = embed_layer(inputs$Region, 22, 5, 'Region_embed'),
    Area_embed = embed_layer(inputs$Area, 6, 5, 'Area_embed')
  )

  # Define reshape functions for ICEs
  embed_ice <- function(input, len) {
    if(len > 1){
      input %>% layer_reshape(target_shape = c(len, 1))
    }
    else {
      input %>% layer_reshape(target_shape = c(1))
    }

  }

  num_embeds <- list(
    DrivAge_embed = embed_ice(inputs$DrivAge, 1),
    VehAge_embed = embed_ice(inputs$VehAge, 1),
    BonMal_embed = embed_ice(inputs$BonMal, 1),
    Density_embed = embed_ice(inputs$Density, 1),
    VehPower_embed = embed_ice(inputs$VehPower, 1)
  )
```

Listing 3: Code for the ICE Processing in R.

```
# Define ICE embeddings
ices <- list(
  DrivAge_ice = layer_input(shape = c(DrivAge_ice_len), dtype = 'float32',
  name = 'DrivAge_ice'),
  VehAge_ice = layer_input(shape = c(VehAge_ice_len), dtype = 'float32',
  name = 'VehAge_ice'),
  BonMal_ice = layer_input(shape = c(BonusMalus_ice_len), dtype = 'float32',
  name = 'BonMal_ice'),
  Density_ice = layer_input(shape = c(Density_ice_len), dtype = 'float32',
  name = 'Density_ice'),
  VehPower_ice = layer_input(shape = c(VehPower_ice_len), dtype = 'float32',
  name = 'VehPower_ice')
)

ices_embed <- list(
  DrivAge_ice_embed = embed_ice(ices$DrivAge_ice, DrivAge_ice_len),
  VehAge_ice_embed = embed_ice(ices$VehAge_ice, VehAge_ice_len),
  BonMal_ice_embed = embed_ice(ices$BonMal_ice, BonusMalus_ice_len),
  Density_ice_embed = embed_ice(ices$Density_ice, Density_ice_len),
  VehPower_ice_embed = embed_ice(ices$VehPower_ice, VehPower_ice_len))

embeds <- embeddings %>% unname %>% layer_concatenate()
num_embeds_flat <- num_embeds %>% unname %>% layer_concatenate()
main = list(embeds, num_embeds_flat) %>% layer_concatenate() %>%
layer_reshape(target_shape = c(1,25))

#### Define ICE models
concat_embeds <- function(embeddings, embeds, ice_embeds, ice_len, ice_idx) {
  repeated_embeddings <- lapply(embeddings, function(x) x %>% layer_repeat_vector(ice_len))
  repeated_embeds <- lapply(embeds, function(x) x %>% layer_repeat_vector(ice_len))
  repeated_embeds[[ice_idx]] <- ice_embeds
  temp1 = repeated_embeddings %>% unname %>% layer_concatenate()
  temp2 = repeated_embeds %>% unname %>% layer_concatenate()
  list(temp1, temp2) %>% layer_concatenate()
}

# ICE names, lengths and indices
ice_info <- list(
  DrivAge_ice_embed = list(len = DrivAge_ice_len, idx = 1),
  VehAge_ice_embed = list(len = VehAge_ice_len, idx = 2),
  BonMal_ice_embed = list(len = BonusMalus_ice_len, idx = 3),
  Density_ice_embed = list(len = Density_ice_len, idx = 4),
  VehPower_ice_embed = list(len = VehPower_ice_len, idx = 5)
)

# Concatenate ICE embeddings
ice_embed_vectors <- list()
for (name in names(ices_embed)) {
  ice_embed_vectors[[paste0(name)]] <-
    concat_embeds(embeddings, num_embeds, ices_embed[[name]],
    ice_info[[name]]$len, ice_info[[name]]$idx)
  }
```

Listing 4: Code for the rest of the ICEnet model in R.

```
#### define main model
 middle_in = layer_input(shape = 25, dtype = "float32")
 middle_layers = middle_in %>%
   layer_dense(units = 32, activation = "relu", input_shape = c(25)) %>%
   layer_dense(units = 16, activation = "relu") %>%
   layer_dense(units = 8, activation = "relu") %>%
   layer_dense(units = 1, activation = 'exponential', name = 'middle',
   weights = list(rep(0,8) %>% matrix(nrow = 8, ncol = 1),
   array(poiss_hom)))

 middle_mod = keras_model(middle_in, middle_layers)

 main_output = main %>% time_distributed(middle_mod)
 main_output_N <- layer_multiply(list(main_output,inputs[[1]])) %>% layer_flatten()

 ### apply main model to ICES
 # Apply time_distributed and create combined_loss_layer
 outputs <- list()
 losses <- list()

 combined_loss_params <- list(
   DrivAge = list(lambda_smooth = 100, lambda_mono = 0, direction = 1),
   VehAge = list(lambda_smooth = 10, lambda_mono = 0, direction = -1),
   BonMal = list(lambda_smooth = 10,  lambda_mono = 100,  direction = 1),
   Density = list(lambda_smooth = 10,  lambda_mono = 100, direction = 1),
   VehPower = list(lambda_smooth = 10, lambda_mono = 100, direction = 1)
 )


 for (name in names(ice_embed_vectors)) {
   outputs[[name]] <- ice_embed_vectors[[name]] %>% time_distributed(middle_mod)
   args = combined_loss_params[[gsub("_ice_embed", "", name)]]
   smooth_layer = CombinedLossLayer(lambda_smooth = args[[1]],
                                    lambda_smooth2 = args[[2]],
                                    lambda_mono=args[[3]],
                                    lambda_fl = args[[4]],
                                    direction=args[[5]])
   losses[[name]] <- outputs[[name]] %>% layer_flatten() %>% smooth_layer()
 }

 losses_concat = losses %>% unname %>% layer_concatenate()

 all_out = list(main_output_N, losses_concat) %>% layer_concatenate()
```

B Appendix: Other performance metrics

Metric	Description	Train		Test	
MSE	FCN	0.04076	(0.000015)	0.04051	(0.000038)
MSE	FCN (nagging)	0.04072		0.04047	
MSE	ICEnet	0.04079	(0.000024)	0.04055	(0.000031)
MSE	ICEnet (nagging)	0.04077		0.04053	
MAE	FCN	0.07171	(0.000859)	0.07188	(0.000857)
MAE	FCN (nagging)	0.07171		0.07188	
MAE	ICEnet	0.07190	(0.000653)	0.07205	(0.000653)
MAE	ICEnet (nagging)	0.07190		0.07205	

Table 11: MSE and MAE losses for the FCN and the ICEnet applied to non-life pricing, training and testing losses. For multiple runs, the average and standard deviation (in parentheses) are reported.

C Appendix: Extra ICEnet plots

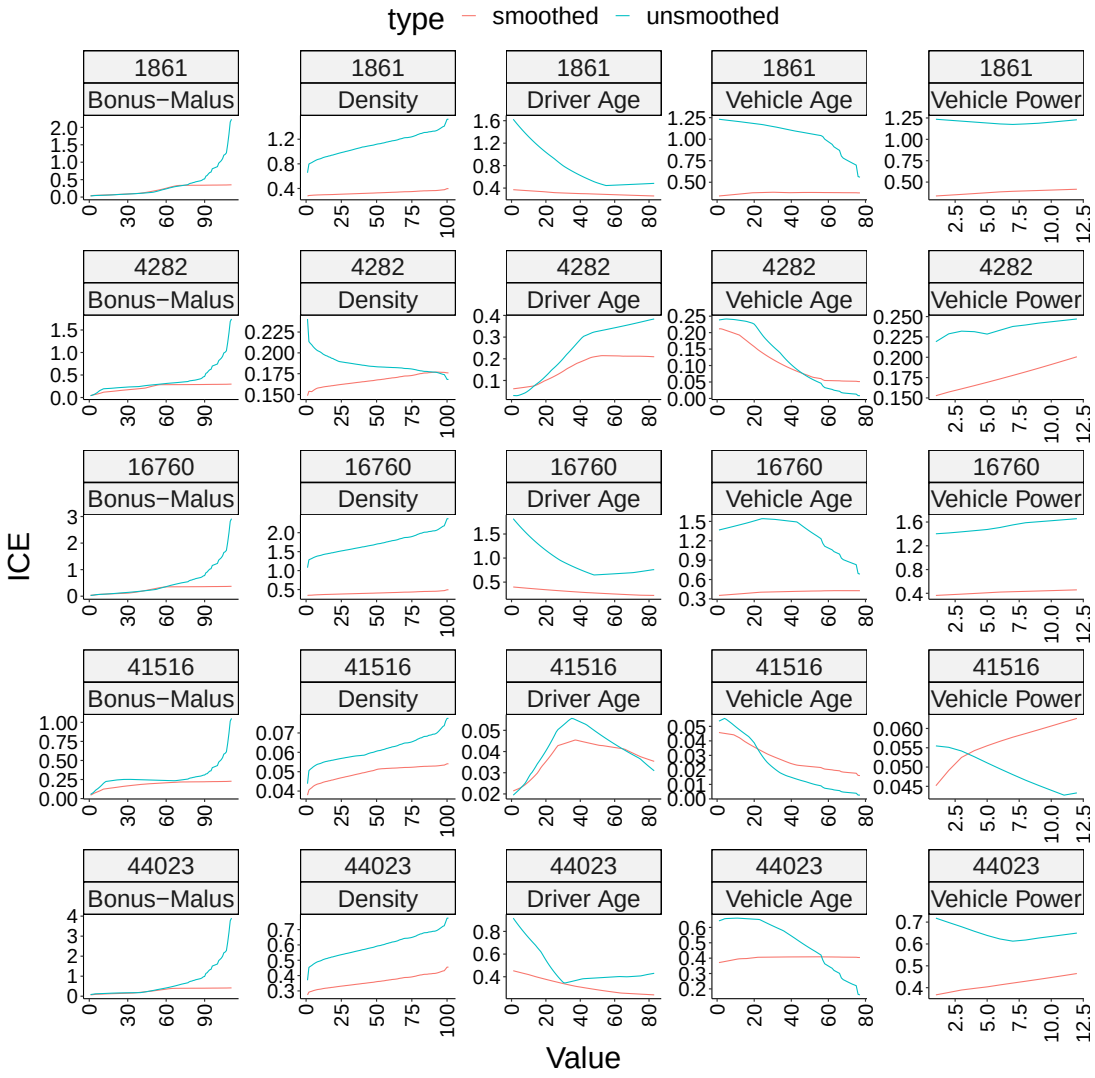

Figure 14: ICE plots of the output of the FCN and the ICEnet for instances n chosen to be the least monotonic based on the monotonicity score evaluated for each instance in the test set on the outputs of the FCN. Note that the smoothed model is the ICEnet and unsmoothed model is the FCN. x-axis shows ordinal values of the pseudo-data on which the ICE is evaluated.

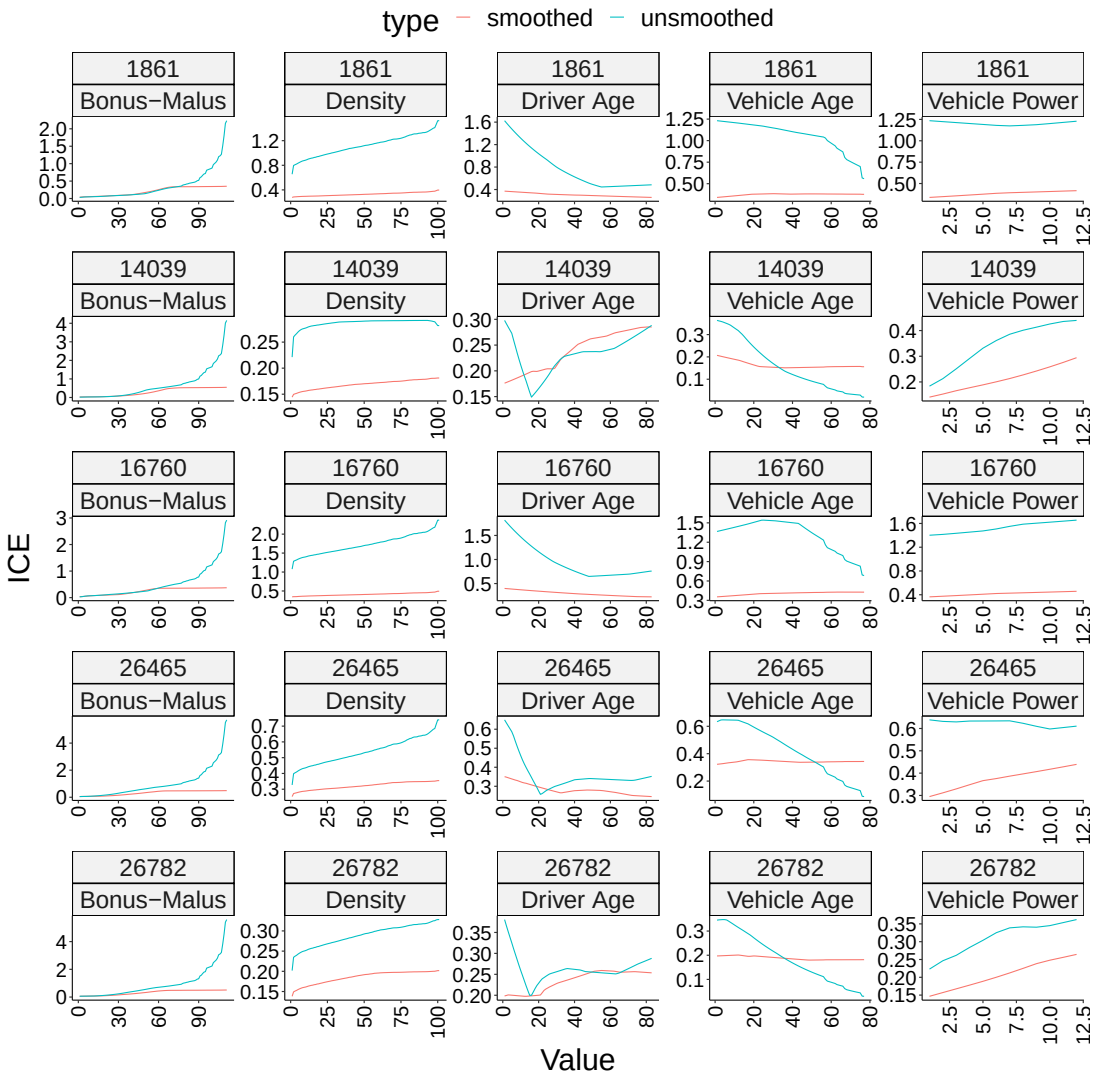

Figure 15: ICE plots of the output of the FCN and the ICEnet for instances n chosen to be the least smooth based on the smoothness score evaluated for each instance in the test set on the outputs of the FCN. Note that the smoothed model is the ICEnet and unsmoothed model is the FCN. x-axis shows ordinal values of the pseudo-data on which the ICE is evaluated.

D Appendix: Local ICEnet plots

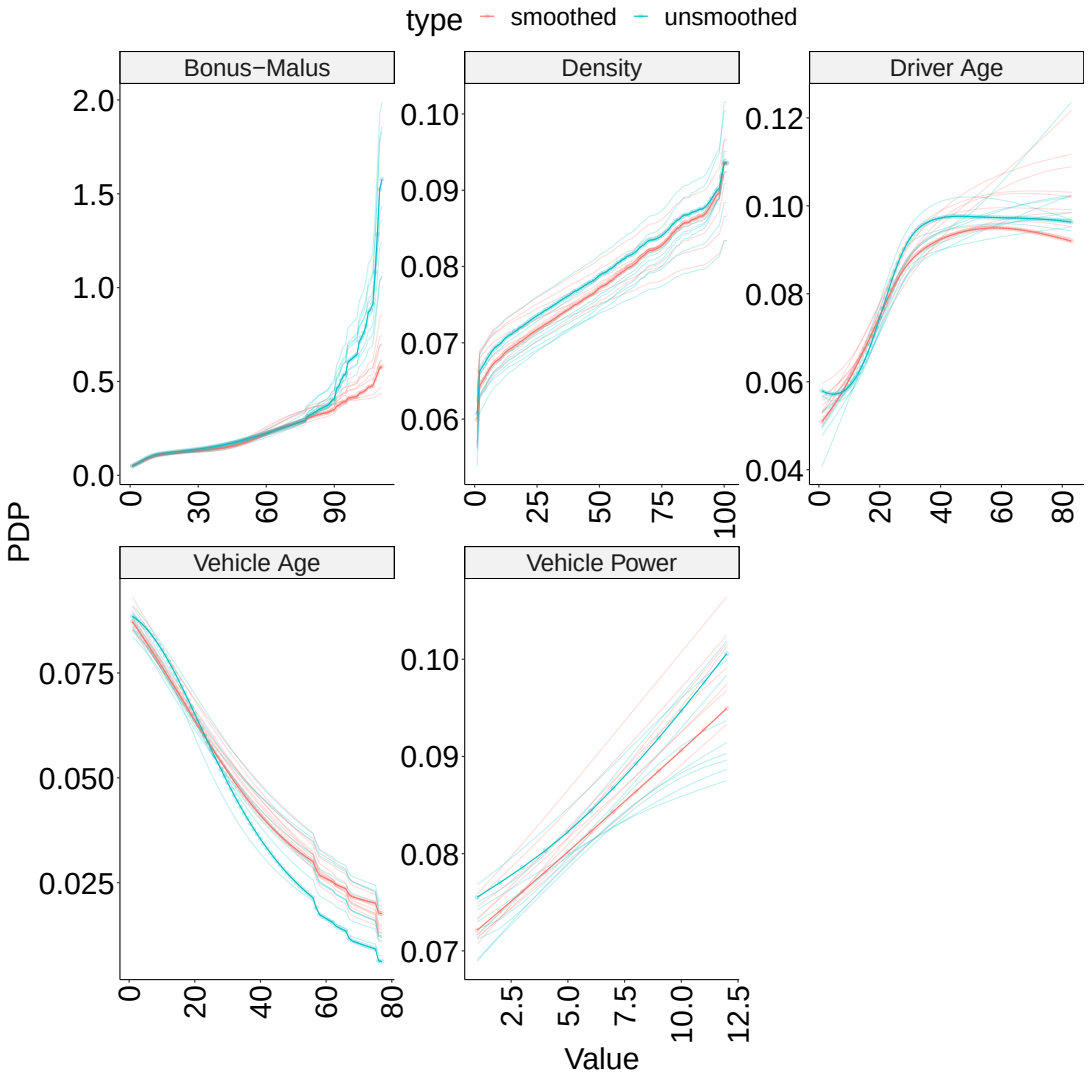

Figure 16: PDPs for each of the Bonus-Malus, Density, Driver Age, Vehicle Age and Vehicle Power fields shown in separate panels, test set only. Blue lines are PDPs from the FCNs (unsmoothed) and red lines are PDPs from the Local ICEnet (smoothed). Bold lines relate to the PDPs from the first of 10 runs; the lighter lines relate to the remaining runs. Note that the scale of the y-axis varies between each panel. x-axis shows ordinal values of the pseudo-data on which the PDP is evaluated.

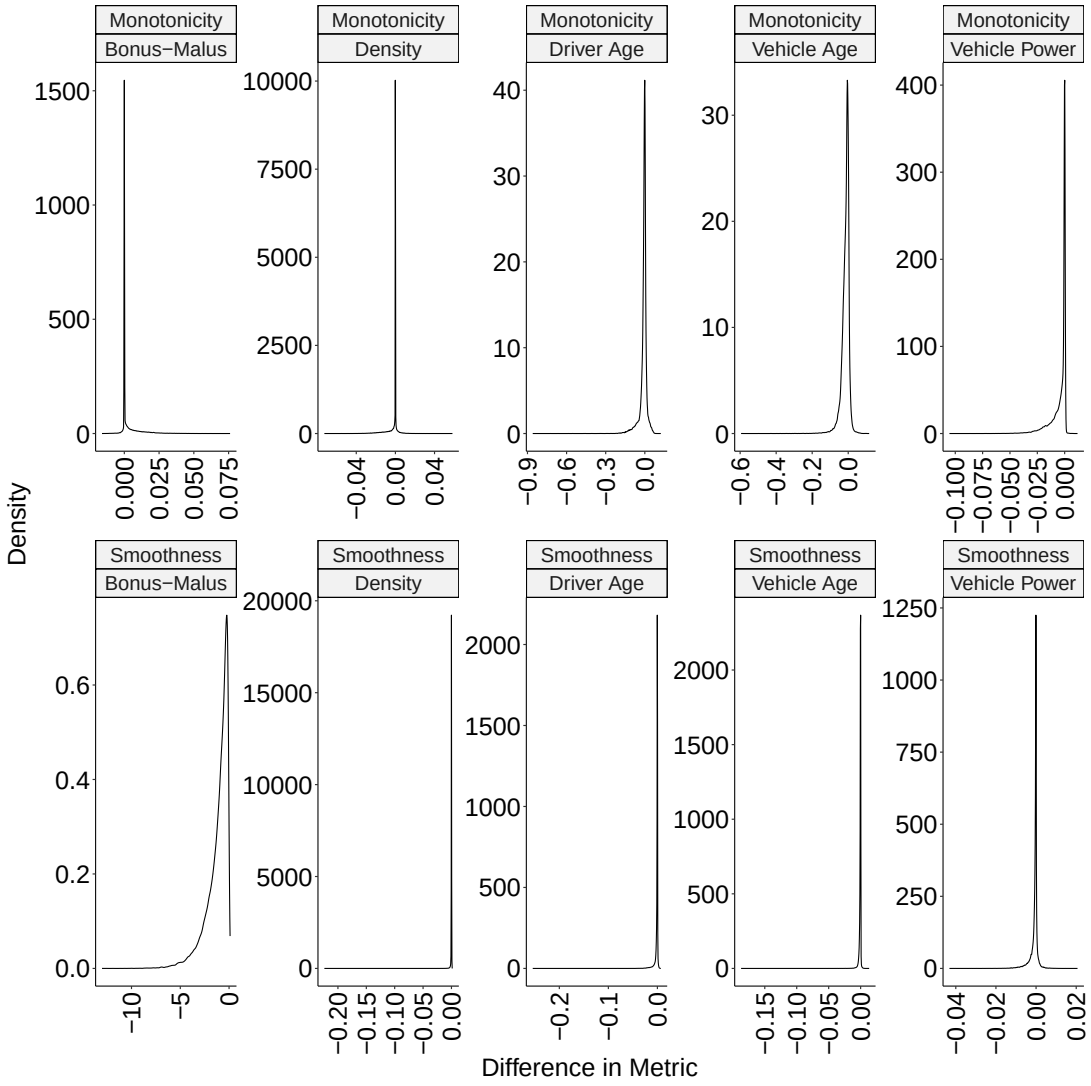

Figure 17: Density plots of the difference between the monotonicity and smoothness components of the Local ICEnet loss function (3.4) evaluated for each instance in the test set.

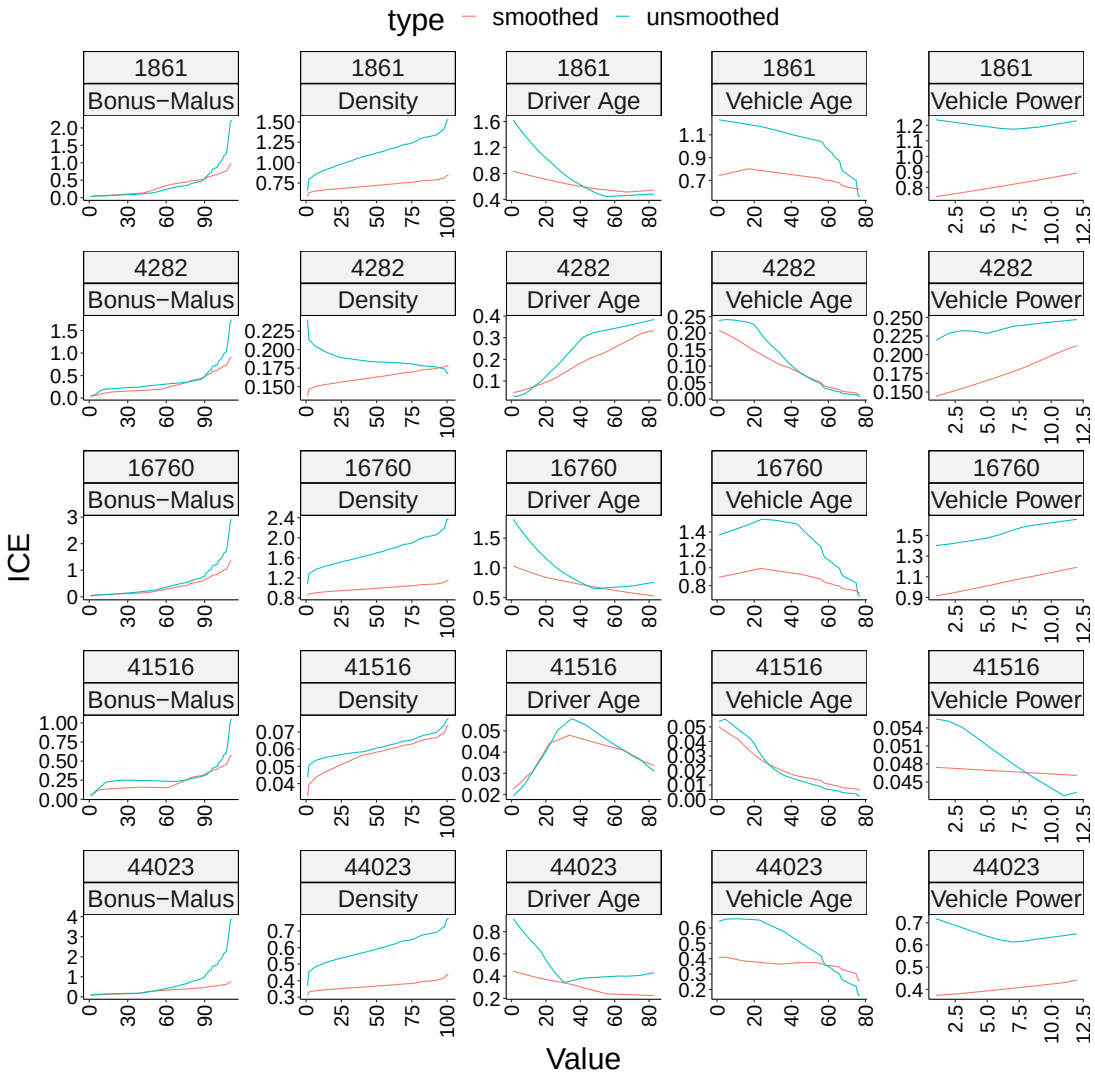

Figure 18: ICE plots of the output of the FCN and the Local ICEnet for instances n chosen to be the least monotonic based on the monotonicity score evaluated for each instance in the test set on the outputs of the FCN. Note that the smoothed model is the Local ICEnet and unsmoothed model is the FCN. x-axis shows ordinal values of the pseudo-data on which the PDP is evaluated. x-axis shows ordinal values of the pseudo-data on which the ICE is evaluated.

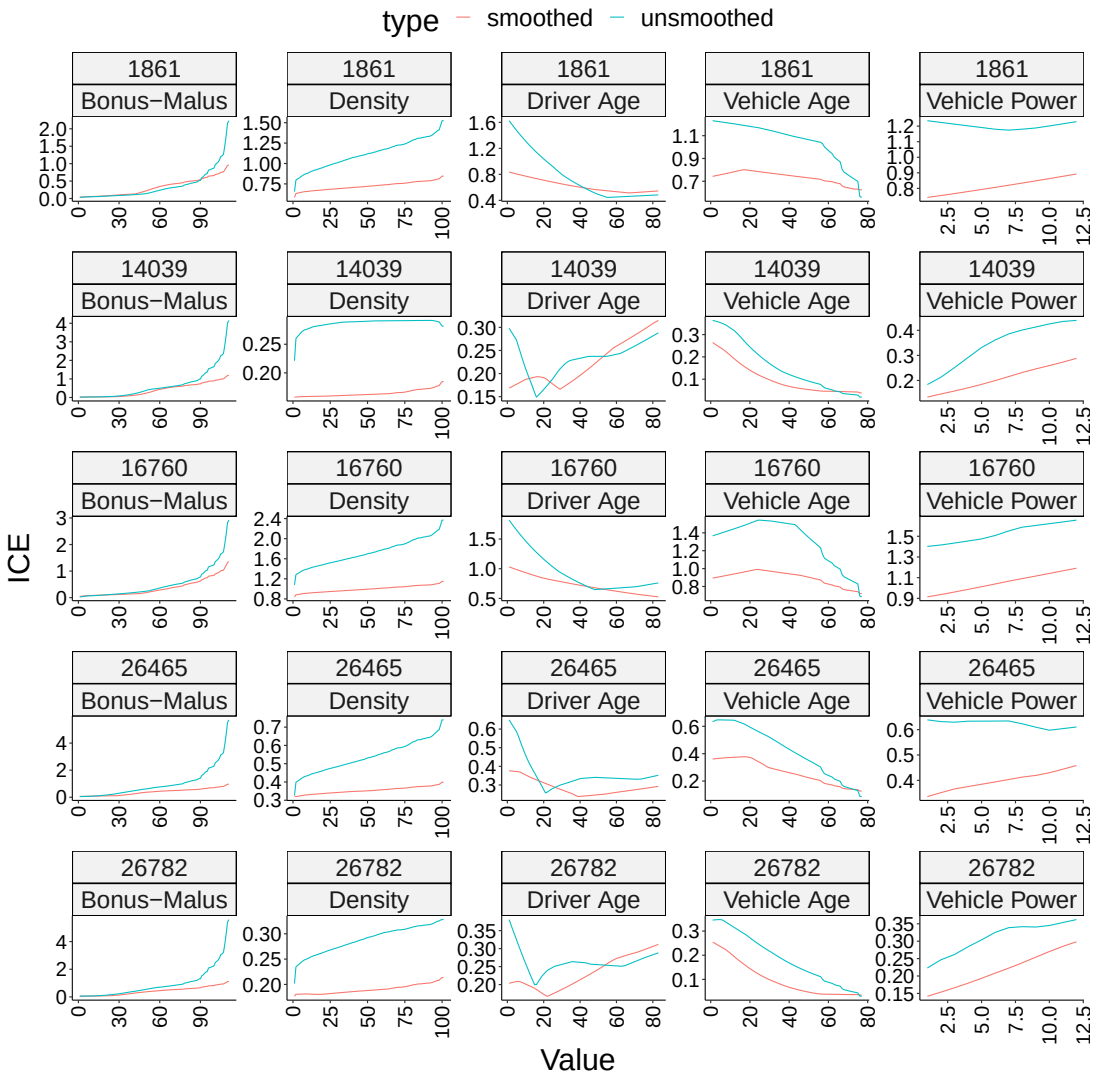

Figure 19: ICE plots of the output of the FCN and the Local ICEnet for instances n chosen to be the least smooth based on the smoothness score evaluated for each instance in the test set on the outputs of the FCN. Note that the smoothed model is the Local ICEnet and unsmoothed model is the FCN. x-axis shows ordinal values of the pseudo-data on which the ICE is evaluated.