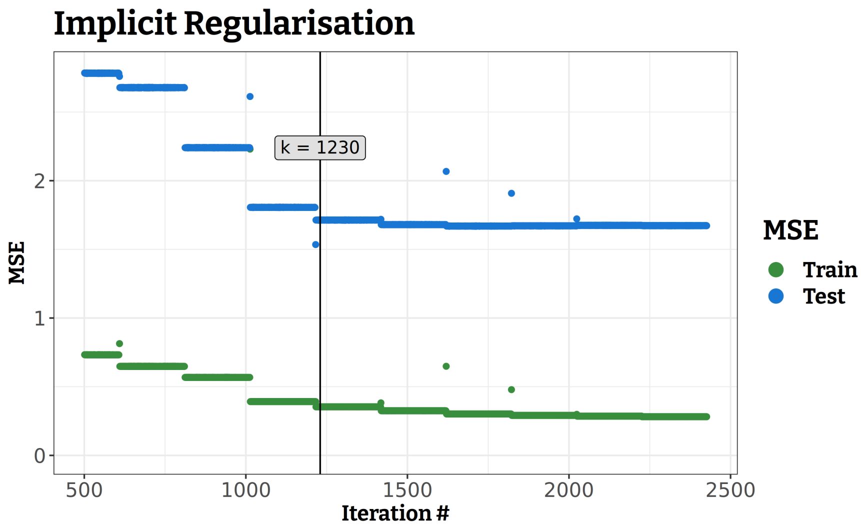
SCIENTIFIC VISUALISATION

Kristina Levina and Farid Musayev 04/10/2022

Statement of Contribution

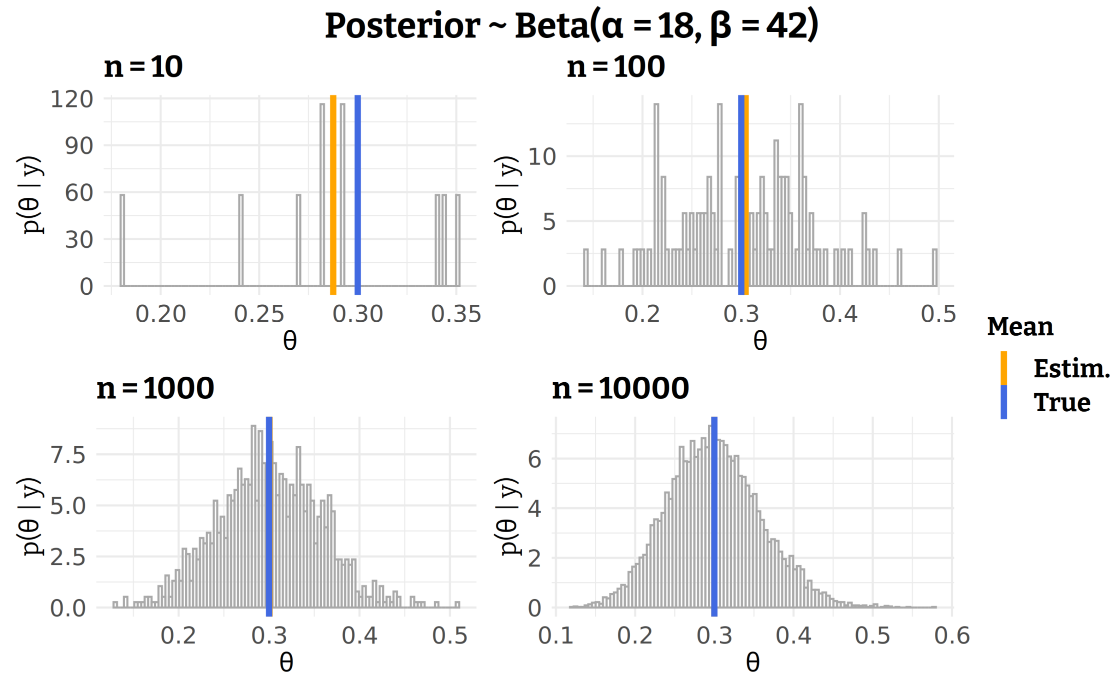
The plots for Convergence to the True Mean and MCMC Parameter Simulation for Poisson Regression were created with collaboration with Farid Musayev.

Implicit Regularisation



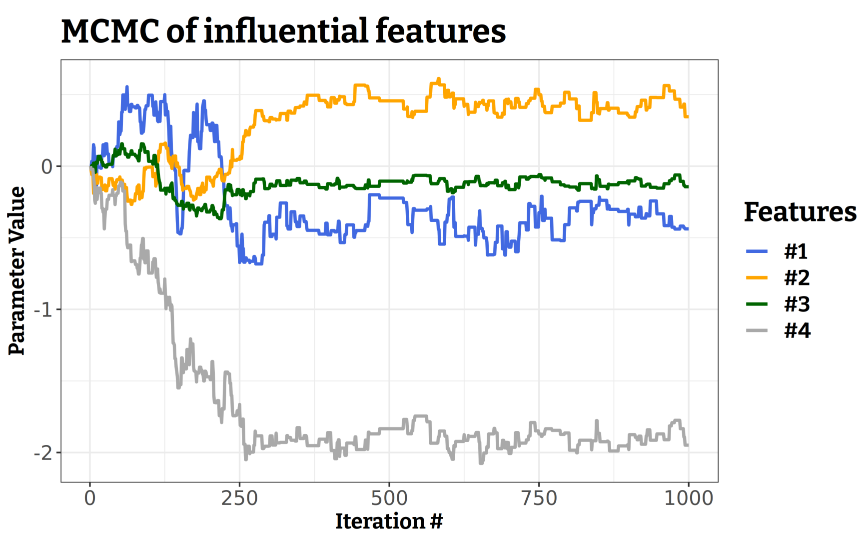
Interpretation: This plot shows how the mean squared error (MSE) changes with the number of iterations of gradient descent. The parameters are optimized implicitly. We see that the test MSE does not decrease significantly after 1230 iterations. Thus, the early stopping criterion can be used to avoid wasting computational resources.

Convergence to the True Mean



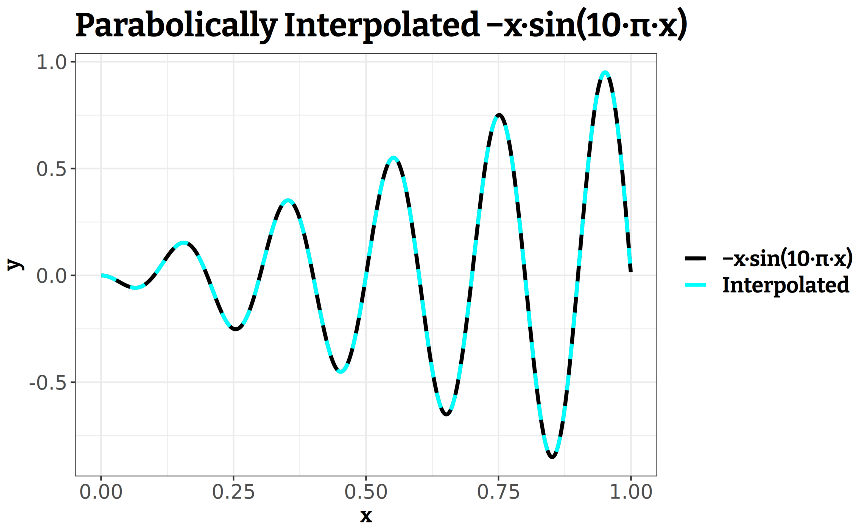
Interpretation: We sample data from the binomial distribution. With increasing number of sampled data points, the mean of the sampled data points converges to the true calculated mean of the binomial distribution. When $n = 1000$, it is already difficult to distinguish between the true and estimated means.

MCMC Parameter Simulation for Poisson Regression



Interpretation: We use the Metropolis Hastings algorithm to simulate the parameters of Poisson Regression. The plot above demonstrates how four parameters corresponding to four features converge after about 250 iterations.

Parabolic Approximation



Interpretation: In the plot above, one can observe how good is parabolic approximation using the gradient descent algorithm. The black line shows the real function, and the dashed cyan line shows the approximated values.

Appendix

Code for implicit regularisation

```
set.seed(12345)

## ggplot2 common theme
dejavu_layer <- list(
  theme_bw(),
  theme(
    text = element_text(family = "Bitter",
                        face = "bold",
                        size = 14),
    title = element_text(size = 15),
    legend.text = element_text(size = 14),
    axis.text = element_text(size = 13,
                             family = "DejaVuSans",
                             face = "plain"),
    axis.title = element_text(size = 14)
  )
)

data <- read.csv("communities.csv")
n <- dim(data)[1]

# Rename the target variable ViolentCrimesPerPop as target
names(data)[names(data) == "ViolentCrimesPerPop"] <- "target"
```

```r
# Divide the data into training and test sets
data <- as.data.frame(scale(data))
id = sample(1:n, floor(n * 0.5))
train = data[id, ]
test = data[-id, ]

# Vectors to store training and test errors
error_tr <- c()
error_test <- c()

# Calculate the MSE cost
cost <- function(par, data){
  tr_cost <- mean((train$target - par %*% t(as.matrix(train[-101])))^2)
  te_cost <- mean((test$target - par %*% t(as.matrix(train[-101])))^2)
  error_tr <<- c(error_tr, tr_cost)
  error_test <<- c(error_test, te_cost)
  return(tr_cost)
}

# Use gradient descent algorithm to minimize the cost
opt <- optim(par = rep(0, 100),
             fn = cost,
             method = "BFGS",
             control = list(maxit = 12))

# Plots:
df <- cbind.data.frame(x = 1:length(error_tr),
                       error_tr = error_tr,
                       error_test = error_test)

plot <- ggplot(df, aes(x = x)) +
  theme_bw() +
  dejavu_layer +
  geom_point(aes(y = error_tr, colour = "1")) +
  geom_point(aes(y = error_test, colour = "2")) +
  xlim(c(501, length(error_tr))) +
  ylim(c(0, error_test[400])) +
  geom_vline(xintercept = 1230) +
  annotate(x = 1230,
           y = 2.5,
           label = "k = 1230",
           vjust = 2,
           geom = "label",
           family = "DejavuSans",
           size = 4,
           fill = "#e0e0e0") +
  labs(title = "Implicit Regularisation",
       x = "# iteration",
       y = "MSE") +
  scale_color_manual(name = "MSE",
                     labels = c("Train", "Test"),
                     values = c("#388e3c", "#1976d2")) +
  guides(colour = guide_legend(override.aes = list(size = 4)))
```

```
  # Legend symbol size

print(plot)
```

Code for convergence to the true mean

```
# Set seed
set.seed(12345)

# Given:
s = 13
n = 50
f = n - s
alpha_0 = 5
beta_0 = 5
# Beta Posterior distribution
beta_posterior <- function(n){
  rbeta(n = n, shape1 = alpha_0 + s, shape2 = beta_0 + f)
}

# True expected value and standard deviation
expected_value <- (alpha_0 + s)/ ((alpha_0 + s) + (beta_0 + f))

variance <- (alpha_0 + s) * (beta_0 + f)/(((alpha_0 + s) + (beta_0 + f))**2*
                                            (alpha_0 + s + beta_0 + f + 1))
std <- variance**0.5

# Theta values generated from beta posterior for different n
df <- data.frame(theta_10 = beta_posterior(n = 10),
                 theta_100 = beta_posterior(n = 100),
                 theta_1000 = beta_posterior(n = 1000),
                 theta_10000 = beta_posterior(n = 10000))


## ggplot2 common theme
dejavu_layer <- list(
  theme_minimal(),
  theme(
    text = element_text(family = "Bitter",
                        face = "bold",
                        size = 13),
    title = element_text(size = 13),
    legend.text = element_text(size = 13),
    axis.text = element_text(size = 12,
                             family = "DejaVuSans",
                             face = "plain"),
    axis.title = element_text(size = 13,
                              face = "plain")
  )
)

# Plots for different n values
```

```r
plot_fun <- function(col_name, title){
  pl <- ggplot(df) +
    dejavu_layer +
    geom_histogram(aes_string(x = col_name, y = "..density.."),
                   fill = "white",
                   bins = 100,
                   color = "darkgrey",
                   alpha = 0.3) +
    geom_vline(aes(xintercept = mean(df[[col_name]]), color = "Estim."),
               size = 1.5) +
    geom_vline(aes(xintercept = expected_value, color = "True"),
               size = 1.5) +
    scale_color_manual(name = "Mean",
                       values = c(Estim. = "orange", True = "blue")) +
    labs(title = title,
         x = "\u03B8",
         y = "p(\u03B8 | y)")

  return(pl)
}

pl1 <- plot_fun("theta_10", "n = 10")
pl2 <- plot_fun("theta_100", "n = 100")
pl3 <- plot_fun("theta_1000", "n = 1000")
pl4 <- plot_fun("theta_10000", "n = 10000")

# Extract the legend
g_legend<-function(a.gplot){
  tmp <- ggplot_gtable(ggplot_build(a.gplot))
  leg <- which(sapply(tmp$grobs, function(x) x$name) == "guide-box")
  legend <- tmp$grobs[[leg]]
  return(legend)
}

shared_legend <- g_legend(pl4)

g <- grid.arrange(arrangeGrob(pl1 + theme(legend.position = "none"),
                              pl2 + theme(legend.position = "none"),
                              pl3 + theme(legend.position = "none"),
                              pl4 + theme(legend.position = "none"),
                              ncol = 2),
             top = textGrob("Posterior ~ Beta(\u03B1 = 18, \u03B2 = 42)",
                            gp = gpar(fontsize = 18,
                                      fontface = "bold",
                                      fontfamily = "Bitter")),
             shared_legend,
             heights = c(10, 0),
             widths = c(8.5, 1.5))

#ggsave("binom.png", g, dpi = 400)
```

Code for MCMC parameter simulation for Poisson regression

```
# Poisson Regression

library(ggplot2)
library(mvtnorm)

set.seed(12345)

## ggplot2 common theme
dejavu_layer <- list(
  theme_bw(),
  theme(
    text = element_text(family = "Bitter",
                        face = "bold",
                        size = 14),
    title = element_text(size = 18),
    legend.text = element_text(size = 14),
    axis.text = element_text(size = 13,
                             family = "DejaVuSans",
                             face = "plain"),
    axis.title = element_text(size = 14)
  )
)

data <- read.table("eBayNumberOfBidderData.dat", header = TRUE)
names(data)[1] <- "target"
X <- as.matrix(data[, -1])
y <- as.matrix(data[, 1])

# Bayesian analysis of the Poisson regression

# Prior: beta ~ N(0, 100*inv((t(X)*X))), where X is the n x p covariate matrix

Sigma <- 100 * solve(t(X) %*% X)
mean = rep(0, 9)
betas_init = rep(0, 9)

# Posterior is assumed multivariate normal: N(beta_mode, inv(J_x(beta_mode))))

# LogPosterior:
logPost <- function(betas, mean, Sigma, X, y){
  logPrior <- dmvnorm(x = betas, mean = mean, sigma = Sigma, log = TRUE)
  # unknown parameters in regression are betas.
  # This is why we write prior for them
  linPred <- X %*% betas
  logLik <- sum(linPred * y - exp(linPred)) #LogLik for the Poisson Model
  return(logPrior + logLik)
}

# Minimize the logLikelihood of the posterior
mode_optim <- optim(betas_init,
                    logPost,
                    gr = NULL,
```

```r
                    mean, Sigma, X, y,
                    method = c("BFGS"),
                    control = list(fnscale = -1),
                    hessian=TRUE)

# Name the coefficient by covariates
names(mode_optim$par) <- names(as.data.frame(X))

# Compute approximate standard deviations.
approxPostStd <- sqrt(diag(-solve(mode_optim$hessian)))
# Name the coefficient by covariates
names(approxPostStd) <- names(as.data.frame(X))

post_mean <- mode_optim$par
print('The posterior mode is')
print(post_mean)
print('The approximate posterior standard deviation is')
print(approxPostStd)
post_cov_mat <- -solve(mode_optim$hessian)
print('The posterior covariance matrix is')
print(post_cov_mat)

# Repeat the analysis using the Metropolis Hastings algorithm

N <- 1000
random_walk_metropolis <- function(logPost, c = 1){
  theta <- matrix(NA, ncol = 9, nrow = N)
  theta[1, ] <- rep(0, 9)
  accept_rate <- 1
  for (i in 2:N){
    proposed <- as.vector(rmvnorm(1, mean = theta[i-1, ],
                                  sigma = c*post_cov_mat))
    post_prev <-logPost(betas = theta[i-1, ], mean, Sigma, X, y)
    post_new <- logPost(betas = proposed, mean, Sigma, X, y)
    accept_pr <- min(1, exp(post_new - post_prev))
    u <- runif(1)
    if (u <= accept_pr){
      theta[i, ] <- proposed
      accept_rate <- accept_rate + 1
    }
    else{
      theta[i, ] <- theta[i-1, ]
    }
  }
  print("Acceptance rate is")
  print(accept_rate/N)
  return(theta)
}

theta <- random_walk_metropolis(logPost)

df_plot <- data.frame(x = 1:N, VerifyId = theta[, 3], Sealed = theta[, 4],
                      Logbook = theta[, 8], MinBidShare = theta[, 9])
```

```r
plot <- ggplot(df_plot, aes(x = x)) +
  dejavu_layer +
  labs(title = "MCMC of influential features") +
  geom_line(aes(y = VerifyId, color = "1"), size = 1) +
  geom_line(aes(y = Sealed, color = "2"), size = 1) +
  geom_line(aes(y = Logbook, color = "3"), size = 1) +
  geom_line(aes(y = MinBidShare, color = "4"), size = 1) +
  xlab("# Iteration") + ylab("Parameter Value") +
  scale_color_manual(values = c("royalblue", "orange", "darkgreen", "darkgray"),
                     labels = c("#1", "#2", "#3", "#4"),
                     name = "Features")

print(plot)

#ggsave("MCMC.png", dpi = 400)
```

Code for parabolic approximation

```r
library(ggplot2)

set.seed(12345)

## ggplot2 common theme
dejavu_layer <- list(
  theme_bw(),
  theme(
    text = element_text(family = "Bitter",
                        face = "bold",
                        size = 14),
    title = element_text(size = 18),
    legend.text = element_text(size = 14),
    axis.text = element_text(size = 13,
                             family = "DejaVuSans",
                             face = "plain"),
    axis.title = element_text(size = 14)
  )
)

f <- function(n, fun, funName){
  #Required for later plot drawing
  x <- c()
  a0 <- c()
  a1 <- c()
  a2 <- c()

  #Interval partition
  lenInt <- 1/n
  largestPoint <- 0
  lowestPoint <- 0

  for (point in 1:n){
    #Three points allocation
```

```r
    largestPoint <- largestPoint + lenInt
    lowestPoint <- largestPoint - lenInt
    middlePoint <- (largestPoint + lowestPoint)/2

    #data to insert into optim
    data <- data.frame(x = c(lowestPoint,
                             middlePoint,
                             largestPoint),
                       y = c(fun(lowestPoint),
                             fun(middlePoint),
                             fun(largestPoint)))
    #print(data)
    optimized <- optim(fn = squaredError,
                       par = c(0, 0, 0),
                       data = data,
                       method = "BFGS")

    #these vectors are required for final plotting
    x <- c(x, middlePoint)
    a0 <- c(a0, optimized$par[1])
    a1 <- c(a1, optimized$par[2])
    a2 <- c(a2, optimized$par[3])
  }

  #The final data.frame for plotting
  dfPlot <- cbind.data.frame(x = x, a0 = a0, a1 = a1, a2 = a2)

  plot <- ggplot(dfPlot, aes(x = x)) +
    dejavu_layer +
    stat_function(fun = fun,
                  aes(color = "1"),
                  n = 1000,
                  size = 1.2) +
    stat_function(fun = function(x) dfPlot$a0 + dfPlot$a1*x + dfPlot$a2*x^2,
                  aes(color = "2"),
                  n = 1000,
                  size = 1.2,
                  linetype = "dashed") +
    scale_color_manual(name = NULL,
                       labels = c("\u2212x\u00B7sin(10\u00B7\u03C0\u00B7x)",
                                  "Interpolated"),
                       values = c("black", "cyan")) +
    labs(title = paste0("Parabolically Interpolated ",
                        "\u2212x\u00B7sin(10\u00B7\u03C0\u00B7x)"),
         x = "x",
         y = "y")

  print(plot)
  #ggsave("parabolic.png", dpi = 400)
}

#To minimize
squaredError <- function(data, par) {
```

```
  MSE <- with(data, sum(par[1] + par[2]*x + par[3]*x*x - y)^2)
  return(MSE)
}

f(n = 1000,
  fun = function(x) -x*sin(10*pi*x),
  funName = "-x*sin(10*pi*x)")

#As we can see, the interpolated function coincides with the real function.
```