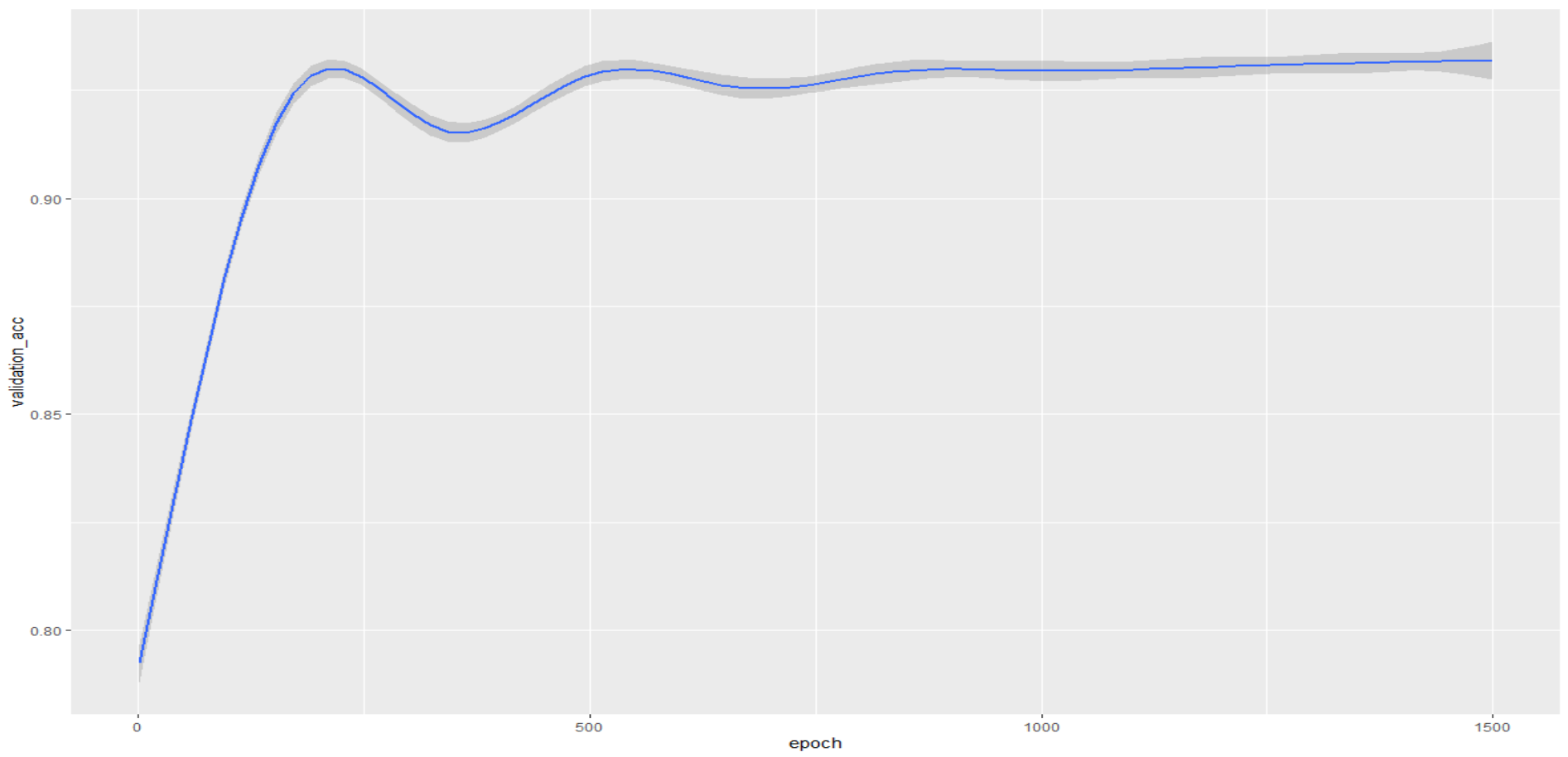
Max validation\_acc: 0.9355249



##Cleanup

```{r}

# Check for duplicates

sum(duplicated(data))

# No duplicates

#Remove ID (irrelevant) and FLAG\_MOBIL (always 1)

data <- data %>% select(-ID, -FLAG\_MOBIL)

cols <- c("CODE\_GENDER","FLAG\_OWN\_CAR","FLAG\_OWN\_REALTY","NAME\_INCOME\_TYPE","NAME\_EDUCATION\_TYPE", "NAME\_FAMILY\_STATUS", "NAME\_HOUSING\_TYPE","FLAG\_WORK\_PHONE","FLAG\_PHONE","FLAG\_EMAIL", "OCCUPATION\_TYPE","status")

cols

data[cols] <- lapply(data[cols],factor)

# Replacing empty values with "Unknown"

levels(data$OCCUPATION\_TYPE) <- c(levels(data$OCCUPATION\_TYPE), "Unknown")

data$OCCUPATION\_TYPE[is.na(data$OCCUPATION\_TYPE)] <- "Unknown"

# Replacing C and X in Status

levels(data$status)[levels(data$status)=="C"] <- "6"

#data$status[data$status == "X"] <- 7

levels(data$status)[levels(data$status)=="X"] <- "7"

# #Convert factors into numericals

# data %<>% mutate\_if(is.factor, as.numeric)

summary(data)

```

# Preprocessing

```{r Create a recipe for preproc}

set.seed(1)

trainIndex <- initial\_split(data, prop = 0.7, strata = status)

trainingSet <- training(trainIndex)

testSet <- testing(trainIndex)

status\_folds <- vfold\_cv(trainingSet, v = 10, strata = status)

status\_folds

```

```{r Create a recipe for preproc2}

set.seed(5)

preprocRecipe <-

recipe(status ~., data = data) %>%

step\_dummy(all\_nominal(), -status, one\_hot = TRUE) %>%

step\_range(all\_predictors(), -all\_nominal(), min = 0, max = 1)%>%

# step\_downsample(status, over\_ratio = 1) %>%

step\_smote(status, over\_ratio = 0.5, skip=TRUE) %>%

# step\_smotenc(status, over\_ratio = 1) %>%

#step\_adasyn(status, over\_ratio = 1) %>%

#step\_nearmiss(status, over\_ratio = 1) %>%

step\_dummy(status, one\_hot = TRUE)# %>%

```

# In this step the above defined receipt is extracted using the `prep()` function, and then use the `bake()` function to transform a set of data based on that recipe.

```{r Prep and bake the defined recipe}

# retain = TRUE and new\_data = NULL ensures that pre-processed trainingSet is returned

trainingSet\_processed <- preprocRecipe %>%

prep(trainingSet, retain = TRUE) %>%

bake(new\_data = NULL)

testSet\_processed <- preprocRecipe %>%

prep(testSet) %>%

bake(new\_data =testSet)

#summary(trainingSet\_processed)

```

## Check data

```{r}

# dim(data)

# sapply(data, class)

# levels(data$status)

#

# #Replace C and X in status

# #data$status[data$status == "C"] <- 6

# levels(data$status)[levels(data$status)=="C"] <- "6"

# #data$status[data$status == "X"] <- 7

# levels(data$status)[levels(data$status)=="X"] <- "7"

# #Convert factors into numericals

# data %<>% mutate\_if(is.factor, as.numeric)

# levels(data$status)

# sapply(data, class)

# summarize the class distribution

percentage <- prop.table(table(data$status)) \* 100

cbind(freq=table(data$status), percentage=percentage)

# Turn data frame into data matrix

matrix\_data <- trainingSet\_processed %>% select(-tail(names(trainingSet\_processed), 8))

#matrix\_data <- subset(data, select = c(CNT\_CHILDREN, AMT\_INCOME\_TOTAL))

#matrix\_targets <- data.matrix(trainingSet\_processed[])

matrix\_targets <- trainingSet\_processed %>% select(tail(names(trainingSet\_processed), 8))

matrix\_data\_test <- testSet\_processed %>% select(-tail(names(testSet\_processed), 8))

matrix\_targets\_test <- testSet\_processed %>% select(tail(names(testSet\_processed), 8))

#Subset only 100 entries for testing

#matrix\_data <- matrix\_data[1:100, ]

#matrix\_targets <- matrix\_targets[1:100, ]

```

## Build Model

```{r}

#train\_data <- matrix\_data

train\_data <- data.matrix(matrix\_data)

test\_data <- data.matrix(matrix\_data\_test)

train\_targets <- data.matrix(matrix\_targets)

test\_targets <- data.matrix(matrix\_targets\_test)

# Function to build the model

build\_model <- function() {

model <- keras\_model\_sequential() %>%

#layer\_batch\_normalization(axis = -1L, input\_shape = dim(train\_data)[[2]]) %>%

layer\_dense(units = 64, activation = "relu", input\_shape = dim(train\_data)[[2]]) %>%

layer\_dense(units = 128, activation = "relu") %>%

layer\_dense(units = 64, activation = "relu") %>%

layer\_dense(units = 8, activation = "softmax")

model %>% compile(

optimizer = optimizer\_sgd(learning\_rate = 0.2),

loss = "categorical\_crossentropy",

metrics = "categorical\_accuracy"

)

}

```

## K-Fold-Validation

```{r}

# mean <- apply(matrix\_data, 2, mean)

# std <- apply(matrix\_data, 2, sd)

# train\_data <- scale(matrix\_data, center = mean, scale = std)

# test\_data <- scale(matrix\_data, center = mean, scale = std)

# train\_targets <- matrix\_targets

k <- 10

indices <- sample(1:nrow(train\_data))

folds <- cut(indices, breaks = k, labels = FALSE)

num\_epochs <- 1500

all\_acc\_histories <- NULL

for (i in 1:k) {

cat("processing fold #", i, "\n")

val\_indices <- which(folds == i, arr.ind = TRUE)

val\_data <- train\_data[val\_indices,] #test\_data#

val\_targets <- train\_targets[val\_indices,] #test\_targets#

partial\_train\_data <- train\_data[-val\_indices,]

partial\_train\_targets <- train\_targets[-val\_indices,]

model <- build\_model()

# Train the model (in silent mode, verbose=0)

# Batch size https://stats.stackexchange.com/questions/153531/what-is-batch-size-in-neural-network

# One epoch = one forward pass and one backward pass of all the training examples

# Batch size = the number of training examples in one forward/backward pass. The higher the batch size, the more memory space you'll need.

# Number of iterations = number of passes, each pass using [batch size] number of examples. To be clear, one pass = one forward pass + one backward pass (we do not count the forward pass and backward pass as two different passes).

# Batch size 32 much faster than 1, also the smaller the batch the less accurate the estimate of the gradient will be.

history <- model %>% fit(

partial\_train\_data, partial\_train\_targets,

validation\_data = list(val\_data, val\_targets),

epochs = num\_epochs, batch\_size = 512, verbose = 1

)

acc\_history <- history$metrics$val\_categorical\_accuracy

all\_acc\_histories <- rbind(all\_acc\_histories, acc\_history)

}

#reticulate::py\_last\_error()

```