Optimal sample size for machine learning using Progressive Sampling

Gertjan Verhoeven

19 juni 2018

Summary

With big data sets and relatively simple models, we can have (much) more data than we actually need to get stable estimates. The goal of this notebook is to demonstrate a heuristic procedure to determine the optimal sample size to get estimates with a predetermined level of accuracy. This involves plotting accuracy versus sample size. By using progresively increasing sample sizes (i.e. each time doubling the data size), we can efficiently find the point where the benefit of the additional accuracy no longer outweights the additional computational cost.

We demonstrate the principle both using simulated data (of which we have infinite amounts) as well as a real (not so big) dataset of wine tastes. When the samples we take are not small relative to the size of the data, random samples will be increasingly correlated, and we underestimate the test error variance. We solve this by partitioning the dataset in non-overlapping samples (sampling without replacement) first.

Simulated data

We mimick the process of grabbing a new random sample from a big data set by repeatedly generating simulated data from the same Data generating process. We pick a sequence of sample sizes, starting out small (say a 100 datapoints) and each time roughly doubling the amount.

For each sample size, we grab 30 samples from our big dataset (i.e. we simulate 30 datasets). We treat each dataset as if it was our only dataset for the analysis task at hand. For a prediction problem, we would typically select a statistical learning algorithm, and tune it using cross-validation. For this step we use the `caret` package. We get an smoothed average test error by doing 5-fold cross validation, repeated 6 times (so repeately partioning the data in 5 folds).

We now have for each sample size a set of 30 smoothed test errors. By examining the variance of these test errors, we get an impression of the variability / uncertainty in the test error estimate. To choose an optimal sample size, for a particalar tolerance level (i.e. we accept 1% variation in the test error estimate) we can simply check at which sample size the variation in the test errors drops below 1% tolerance.

Function to generate simulated clean data

```
rdunif <- function(n,k) sample(1:k, n, replace = T)

# simulate noise dataset with signal on x2 as function of relevance
generateData <- function(nsize, relevance, interaction = 0){
  y <- rbinom(n = nsize, size = 1, prob = 0.5)
  x1 <- rnorm(n = nsize, mean = 0, sd = 1)
  x3 <- rdunif(n = nsize, k = 4)
  x4 <- rdunif(n = nsize, k = 10)
  x5 <- rdunif(n = nsize, k = 20)
```

```
  x2 <- rep(-1, nsize)
  x2[y == 1 & x1 < 0] <- rbinom(n = sum(y == 1 & x1 < 0), size = 1, prob = 0.5 - relevance - interactio
  x2[y == 1 & x1 >= 0] <- rbinom(n = sum(y == 1 & x1 >= 0), size = 1, prob = 0.5 - relevance + interact
  x2[y == 0 & x1 < 0] <- rbinom(n = sum(y == 0 & x1 < 0), size = 1, prob = 0.5 + relevance - interactio
  x2[y == 0 & x1 >= 0] <- rbinom(n = sum(y == 0 & x1 >= 0), size = 1, prob = 0.5 + relevance + interact

  my_df <- data.frame(y = as.factor(y), x1, x2, x3, x4, x5)
  my_df
}
```

Set train control and tuning grid search

```
train.control <- trainControl(method = "repeatedcv",
                              number = 5, repeats = 6,
                              savePredictions = F)


# tuning pars mtry, splitrule, min.node.size
rf.grid <- expand.grid(mtry = 3,
                       splitrule = "gini",
                       min.node.size = 1)
```

Run simulation

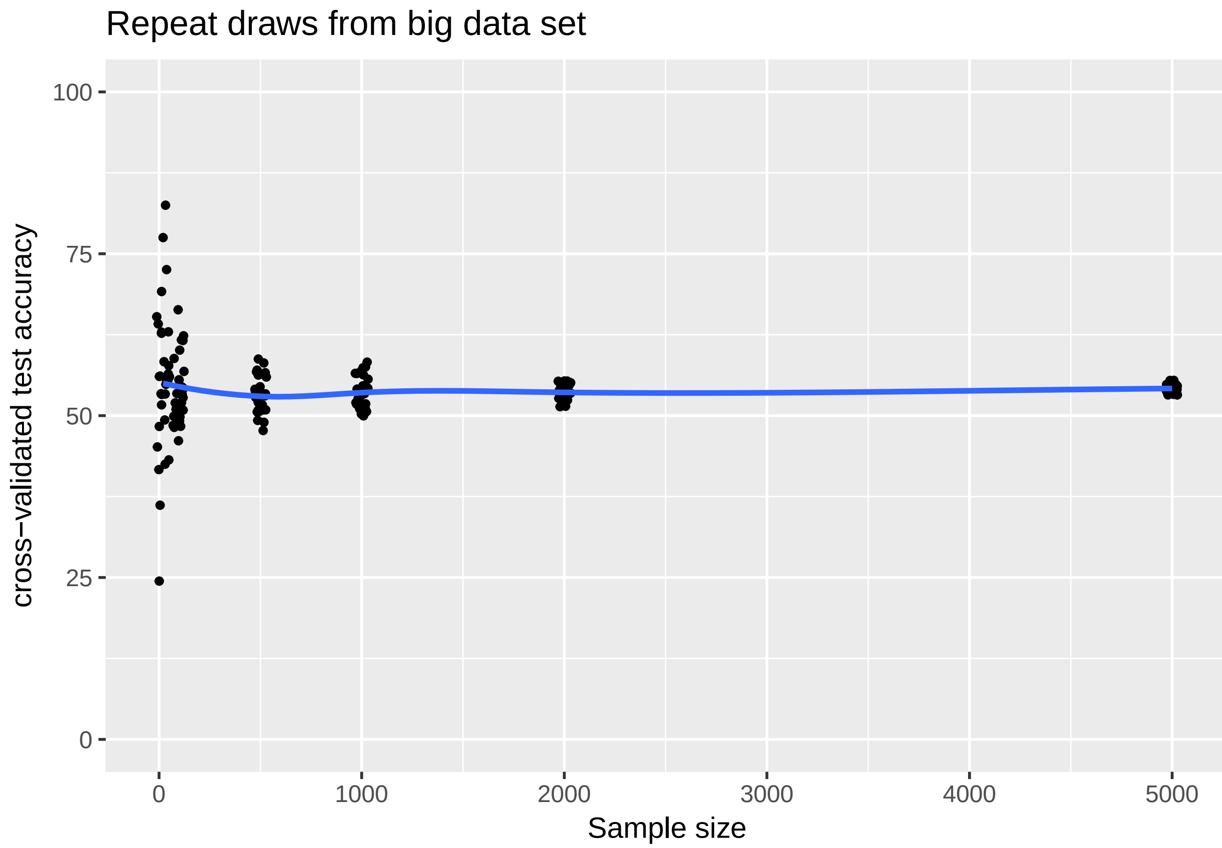
```
fullrun <- 0

sample_size_vec <- c(20, 100, 500, 1000, 2000, 5000)
repeats <- 30

myres <- data.frame()

set.seed(1)

if(fullrun){
  z <- 1
  for(i in 1:length(sample_size_vec)){
      for(j in 1:repeats){
        # grab a new sample from our unlimited big data set
        my_df <- generateData(sample_size_vec[i], 0.1)

        caret_fit <- train(y ~ .,
                           data = my_df,
                           method = "ranger",
                           trControl = train.control,
                           tuneGrid = rf.grid)
        myres[z,1]<- sample_size_vec[i]
        myres[z,2] <- j
        myres[z,3] <- mean(caret_fit$resample$Accuracy)
        z <- z + 1
```

```
    }
  }
  colnames(myres) <- c("Sample_size", "repeat", "Mean_accuracy")
  myres <- data.table(myres)
  myres <- myres[, Mean_accuracy := Mean_accuracy * 100]
  saveRDS(myres, file = "output/myres.rds")
} else { myres <- readRDS("output/myres.rds") }
```

Plot result

```
res <- myres[, .(Min = min(Mean_accuracy),
                 Mean = mean(Mean_accuracy),
                 Max = max(Mean_accuracy)), .(Sample_size)]
res$order <- 1:nrow(res)


gp <- ggplot(myres, aes(x = Sample_size , y = Mean_accuracy)) +
geom_point(size = 1, col = "black", position = "jitter") + ylim(0, 100) +
  ylab("cross-validated test accuracy") +
  xlab("Sample size") +
  ggtitle("Repeat draws from big data set") + geom_smooth(fill = NA)

gp
```

```
## `geom_smooth()` using method = 'loess'
```



Here, the increase in sample size appears mostly to affect the reduction in variance of the test error, and not so much better learning from the training. This makes sense, since there is a only a simple signal present in the simulated data.

Real dataset: Wine Quality

```
fullrun <- 0

if(fullrun){
  url <- 'https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-white.csv'
  wine <- read.table(url, sep = ';', header = TRUE)
  #head(wine)
  wine$taste <- ifelse(wine$quality < 6, 'bad', 'good')
  wine$taste[wine$quality == 6] <- 'normal'
  wine$taste <- as.factor(wine$taste)
  wine$quality <- NULL
  saveRDS(wine, "wine.rds")
} else { wine <- readRDS("wine.rds")}
```

Run calculation for Wine Quality

The Wine Quality dataset consists of 4898 records. This means that if we want at least 10 independent samples, we are limited to a sample size that is one tenth of the full dataset.

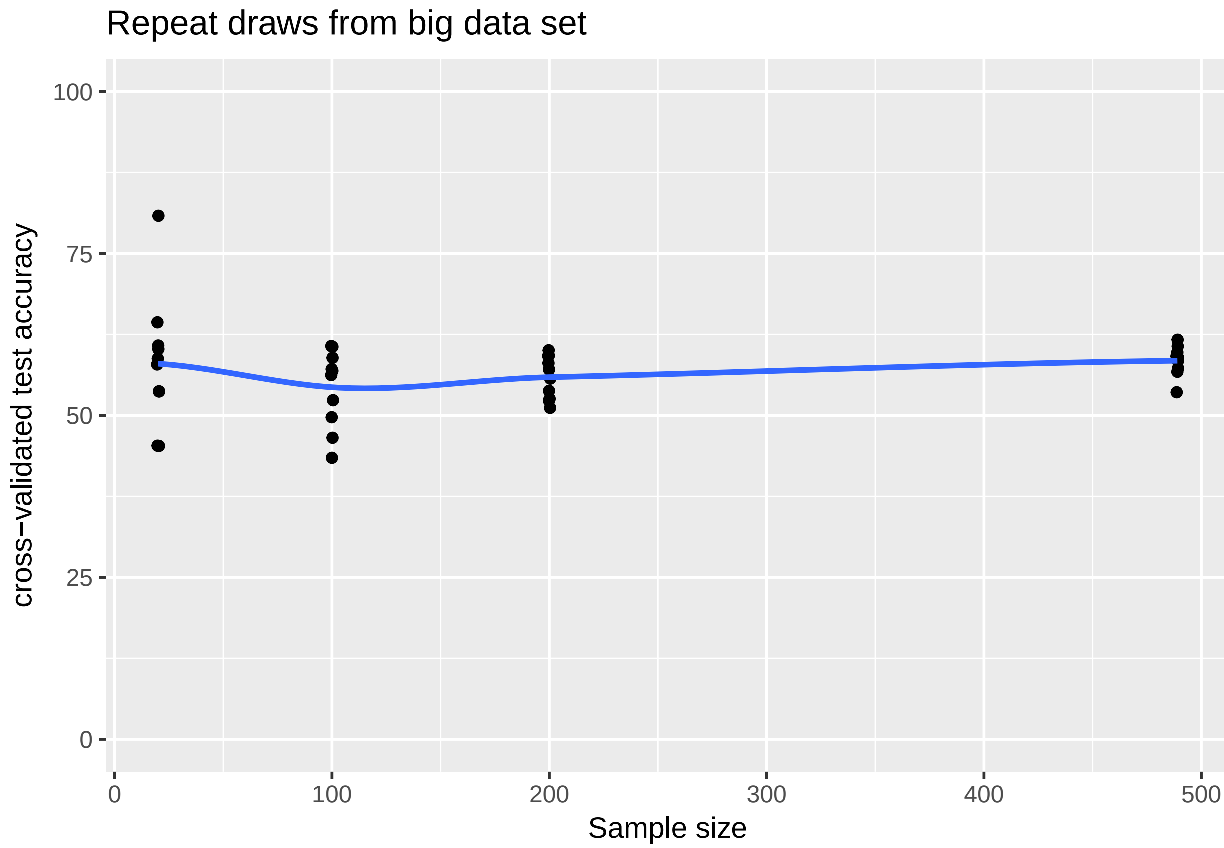
This also means that we have to partition the full dataset in ten separate samples, and performing the model fitting and repeated cross-validation on these separate samples. Here we choose 5 folds and repeated 2 times.

```
train.control <- trainControl(method = "repeatedcv",
                              number = 5, repeats = 2,
                              savePredictions = F)
```

```
fullrun <- 0

sample_size_vec <- c(20, 100, 200, 489)
repeats <- 10

myres <- data.frame()
wine$test_zero_var <- 0
set.seed(1)

if(fullrun){
  z <- 1
  for(i in 1:length(sample_size_vec)){
      # total number of datapoints required
      size <- sample_size_vec[i] * repeats
      # take datapoints, NO replacement
      my_sample <- sample(1:nrow(wine), size, replace = F)
      my_df <- wine[my_sample,]
      # split in repeats
      my_splits <- sample(1:repeats, nrow(my_df), replace = T)

      for(j in 1:repeats){
        df_refactor <- my_df[my_splits == j,]
        # caret complains if the outcome factor levels contain a factor that is not included in the dat
        df_refactor$taste <- factor(df_refactor$taste)
        #print(j)
        caret_fit <- train(taste ~ .,
                              data = df_refactor,
                              method = "ranger",
                              preProcess = c("nzv"), # remove near zero variance
                              trControl = train.control,
                              tuneGrid = rf.grid)
        myres[z,1]<- sample_size_vec[i]
        myres[z,2] <- j
        myres[z,3] <- mean(caret_fit$resample$Accuracy)
        z <- z + 1
      }
  }
  colnames(myres) <- c("Sample_size", "repeat", "Mean_accuracy")
  myres <- data.table(myres)
  myres <- myres[, Mean_accuracy := Mean_accuracy * 100]
  saveRDS(myres, file = "output/myres2.rds")
} else { myres <- readRDS("output/myres2.rds") }
```

Plot result

```
res <- myres[, .(Min = min(Mean_accuracy),
                 Mean = mean(Mean_accuracy),
                 Max = max(Mean_accuracy)), .(Sample_size)]
res$order <- 1:nrow(res)


gp <- ggplot(myres, aes(x = Sample_size , y = Mean_accuracy)) +
  geom_jitter(width = 0.5, height = 0.5) + ylim(0, 100) +
  ylab("cross-validated test accuracy") +
  xlab("Sample size") +
  ggtitle("Repeat draws from big data set") + geom_smooth(fill = NA)

gp
```



As in the simulated dataset, two effects can be seen: * the benefit of having more training data for better learning, as well as * the reduction in variance in test error due to having more test data is visible

References

- Provost, Jensen & Oates (1999), Efficient progressive sampling, Proceedings of the fifth ACM SIGKDD International conference on Knowledge discovery and data mining.