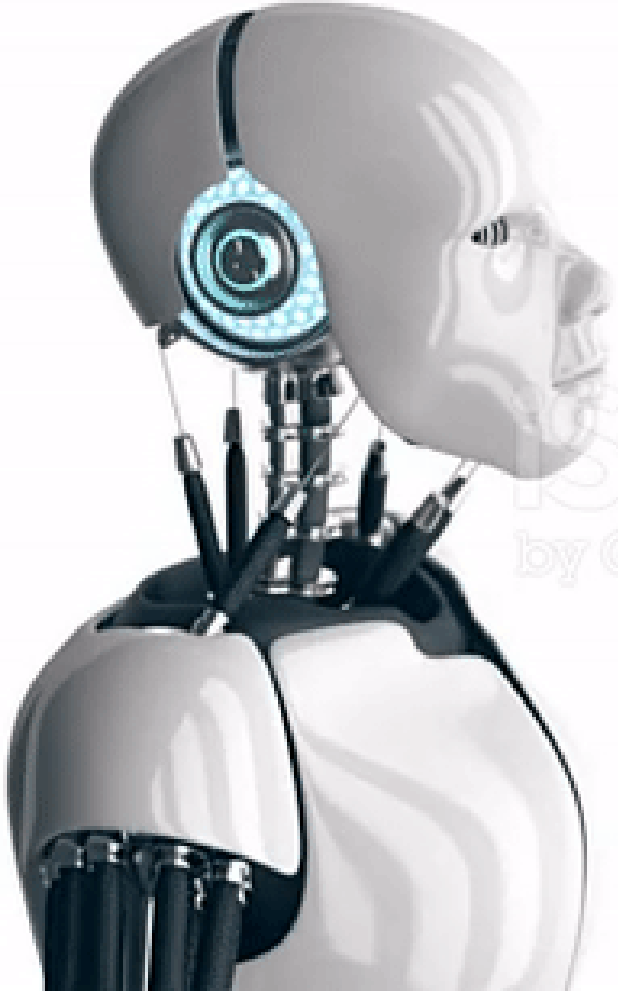
Selecting and Tuning Hyperparameters...



Alastair K. Muir, PhD, MBB

Muir&Associates Consulting

Calgary, Alberta, Canada

January 9, 2020

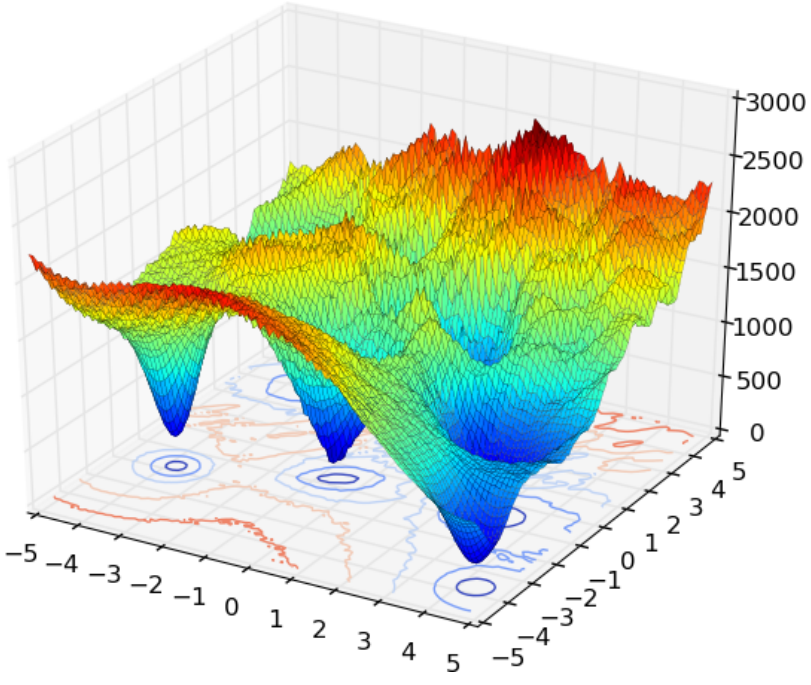
Can You Come To A Meeting at 2:30 Today?

... because you are that "data specialist", you have just been summoned to a strategy meeting to solve a business problem. The crux of the situation is the business owners, board members, stakeholders, customers, process owners, and special interest groups, need to know how *exactly* to produce the product, output, or service that is the best solution.

This is an intensely complex, non-linear system with plenty of "magic bullet" solutions in the ditch behind you.

- It's not really your area of expertise
- How do you learn about what you don't know?
- **When do you stop?**

Problems Are Non-Linear and Non-Convex



Our problem is that industrial processes, transactional business processes, biological and chemical systems are non-linear and these factors also interact with each other to produce the final result.

Our primary goal in **machine learning** is to fit a non-linear function to the observed data.

Our primary goal in **optimization** is to find the global minimum in a non-convex hyperspace.

Parameters and HyperParameters

Parameters Configurations and values your machine learning algorithm learns from your data during training. These parameters can interact to describe complex, non-linear systems.

> The algorithms themselves also have a large number of unknown **hyper**parameters. These are not determined directly from the data and can dramatically increase model performance and improve training times.

Hyperparameters Express high level concepts, such as statistical assumptions(eg. regularization), are fixed before training or are hard to learn from data.

> ...but this only opens up the problem of **optimizing** a different, non-convex, non-linear system.

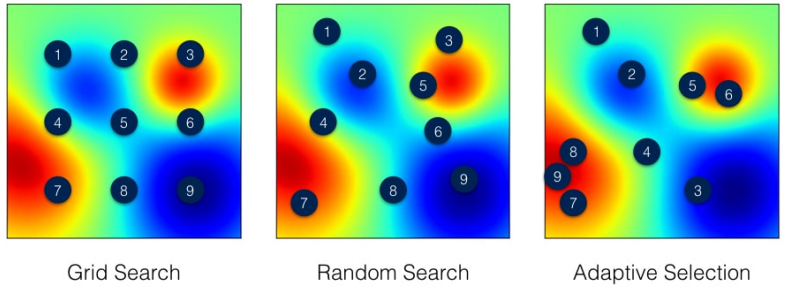
The Set of Hyperparameters Depend On The Algorithm

Algorithm	Hyperparameter	Type	Lower	Upper	Trafo
glmnet					
	alpha	numeric	0	1	-
	lambda	numeric	-10	10	2^x
rpart					
	cp	numeric	0	1	-
	maxdepth	integer	1	30	-
	minbucket	integer	1	60	-
	minsplit	integer	1	60	-
kknn	-	-			
	k	integer	1	30	-
svm					
	kernel	discrete	-	-	-
	cost	numeric	-10	10	2^x
	gamma	numeric	-10	10	2^x
	degree	integer	2	5	-
ranger					
	num.trees	integer	1	2000	-
	replace	logical	-	-	-
	sample.fraction	numeric	0.1	1	-
	mtry	numeric	0	1	$x \cdot p$
	respect.unordered.factors	logical	-	-	-
	min.node.size	numeric	0	1	n^x
xgboost					
	nrounds	integer	1	5000	-
	eta	numeric	-10	0	2^x
	subsample	numeric	0.1	1	-
	booster	discrete	-	-	-
	max_depth	integer	1	15	-
	min_child_weight	numeric	0	7	2^x
	colsample_bytree	numeric	0	1	-
	colsample_bylevel	numeric	0	1	-
	lambda	numeric	-10	10	2^x
	alpha	numeric	-10	10	2^x

- Some optimization routines will allow particular sampling distributions, but most assume a uniform distribution between upper and lower values
- But the outcome from a CNN will be substantially different for
  - **`eta`** changes from 0.010 to 0.011
  - **`eta`** changes from 0.001 to 0.002
- Transform the hyperparameter
  - Desired range for **`eta`** is on a log scale
    - **range_eta <- c(0.1, 0.00001)**
  - Parameter passed to the optimization routine is -1 to -5
    - **`x <- c(-1, 5)`**
  - Sampling is uniform on x within the optimization
  - Internally, the fitness function uses
    - **`eta <- 10^x`**

[1] Philipp Probst, Anne-Laure Boulesteix and Bernd Bischl, arXiv:1802.09596v3 [stat.ML] 22 Oct 2018

Hyperparameter Search Strategies



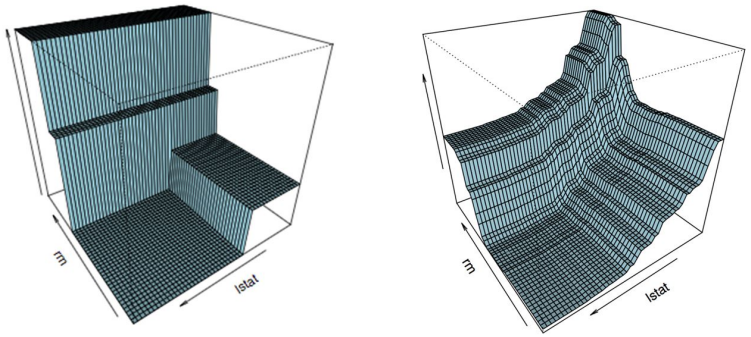
- Conduct a very wide search to investigate potentially widely different local minima.
- Concentrate on single important hyperparameters such as `eta` (learning rate) before optimizing the complete set of hyperparameters.
- Don't worry about fine tuning such parameters as learning rate decay.
- Incorporate early stopping in place of L2 regularization.

- The ground truth can only be determined by an exhaustive search of all combinations of hyperparameters using a grid search. While a **grid search** is comprehesive and complete, it is usually impractical and would take nearly an infinite length of time.

- A **random search** is superior to a systematic search, can cover a similar domain, and can incorporate empirically determined sampling probability distributions.

- An **adaptive search** (there are a large number) begin with a sparse random or grid search, but incorporate early results in subsequent sampling. In other words, if the sampling has identified regions with very poor performance, those regions should be avoided while concentrating on regions with better performance. This is a *bet on a winner* strategy.

Eg. Predict The Strength of High Performance Concrete

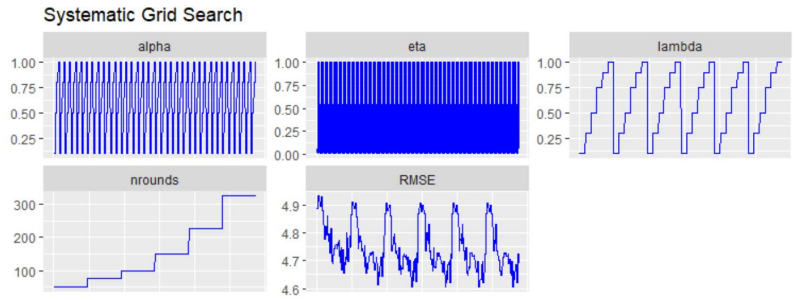


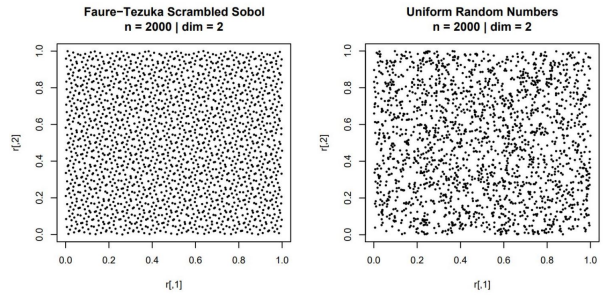
Use XGBoost to predict concrete compressive strength of concrete given the weight per cubic meter of:

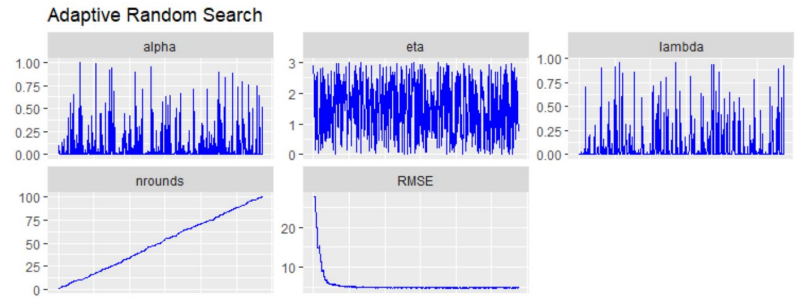
- Cement
- Blast Furnace Slag
- Fly Ash
- Water
- Superplasticizer
- Coarse Aggregate
- Fine Aggregate
- Age (days)

- Grid search
  - define an n-dimensional grid with a full range of permissible values for each hyperparameter
- Random and adaptive random search
  - a set of values of hyperparameters is generated that depend on the previously evaluated sets of hyperparameters
- Genetic algorithms and differential evolution
  - choose a random population of hyperparameters, then create offspring of your better solutions with some mutation, repeat the whole process until it converges
- Particle swarm optimization
  - an initial array of random points. Allow good solutions to cluster
- Bayesian methods
  - an initial array of random points is evaluated. Results are used describe a function to refine future sampling

Grid And Random Search







Grid Search

- Full domain search
- Parallel processing
- Computationally intensive

Sobol Search

- Tends to separate random points that are close to each other

Adaptive Random Search

- Why continue to sample where you already know the RMSE is poor?
- Fit a local GLM model and resample where previous results are predicted to be good
- Can run with far fewer runs than a full grid search over the same domain

Grid Search and Sobol Random Sampling

```
sobolGrid <- data.frame(runif.sobol(n = 200, dim = 4, scrambling = 3, seed = 1, init = TRUE))
names(sobolGrid) <- c("nrounds", "eta", "alpha", "lambda")
SSBoost_grid <- data.table(nrounds = round(sobolGrid$nrounds*400) + 100,
                           eta = 10^((sobolGrid$eta * 4) - 3),  # learning rate, low value less
                           alpha = 10^((sobolGrid$alpha * 4) - 3), # L2 Regularization (Ridge Re
                           lambda = 10^((sobolGrid$lambda *4) - 3)) # L1 Regularization (Lasso R
cluster <- makeCluster(detectCores() - 1) # number of cores, convention to leave 1 core for OS
registerDoParallel(cluster) # register the parallel processing
# Train model with grid search
SS_XGBoost_Linear_model <- caret::train(
  Strength ~.,
  data = train.data,
  method = "xgbLinear",
  trControl = adapt_control_grid,
  verbose = TRUE,
  silent = 1,
  # tuneLength = 20
  tuneGrid = SSBoost_grid
)
stopCluster(cluster) # shut down the cluster
registerDoSEQ() #  force R to return to single threaded processing
```

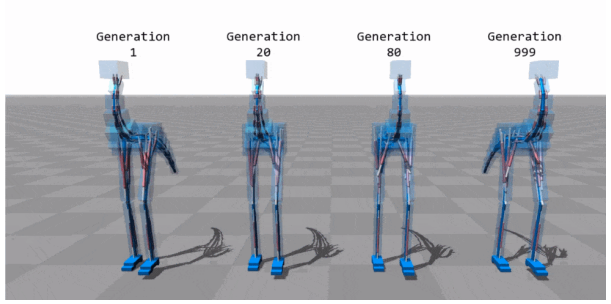
Adaptive Optimization Algorithms - Define Fitness Function

```r
# Create custom function for assessing solutions
eval_function_XGBoost_Linear <- function(x1, x2, x3, x4, data, train_settings) {
  suppressWarnings(
    XGBoost_Linear_model <- caret::train(
      Strength ~.,
      data = data,
      method = "xgbLinear",
      trControl = train_settings,
      verbose = FALSE,
      silent = 1,
      tuneGrid = expand.grid(
        nrounds = round(x1), # number of boosting iterations
        eta = 10^x2, # learning rate, low value means model is more robust to overfitting
        alpha = 10^x3, # L1 Regularization (equivalent to Lasso Regression) on weights
        lambda = 10^x4 # L2 Regularization (equivalent to ridge Regression) on weights
      )
    )
  )
return(-XGBoost_Linear_model$results$RMSE) # minimize RMSE
}
```

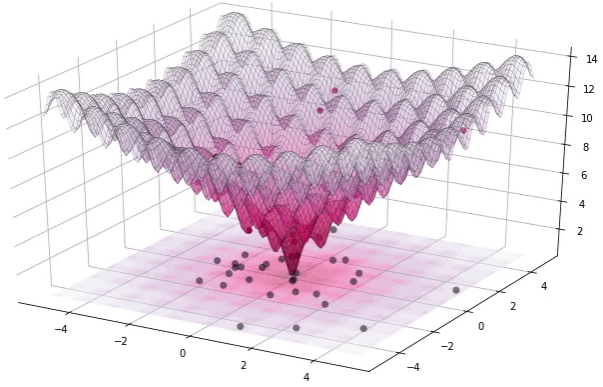
Adaptive Optimization Algorithms - Run Optimization

```
# Define minimum and maximum values for each input
nrounds_min_max <- c(10,10^3)
eta_min_max <- c(-3,1)
alpha_min_max <- c(-3,1)
lambda_min_max <- c(-3,1)
# Set parameter settings for search algorithm
max_iter <- 10 # maximum number of iterations
pop_size <- 10 # population size
GA_XGBoost_Linear_model <- GA::ga(
  type = "real-valued",
  fitness = function(x) eval_function_XGBoost_Linear(x[1],x[2],x[3],x[4],
                                                     data = train.data,
                                                     train_settings = train_control),
  lower = c(nrounds_min_max[1], eta_min_max[1], alpha_min_max[1], lambda_min_max[1]),
  upper = c(nrounds_min_max[2], eta_min_max[2], alpha_min_max[2], lambda_min_max[2]),
  popSize = pop_size, maxiter = max_iter, #population size, number of iterations
  pmutation = 0.5, elitism = 0.3, # probability of mutation,  percentage of elitism
  parallel = n_cores, optim = F, keepBest = T, seed = 1
)
```

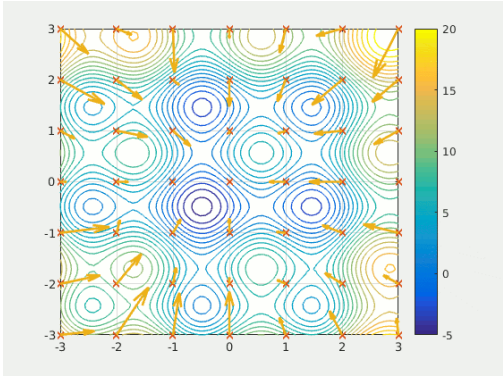
Adaptive Algorithms



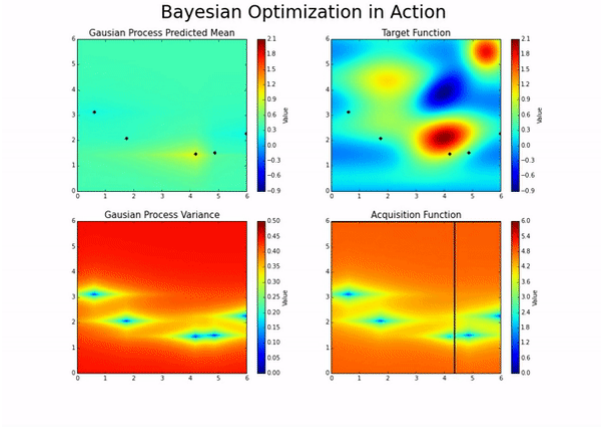
Genetic Algorithm



Differential Evolution

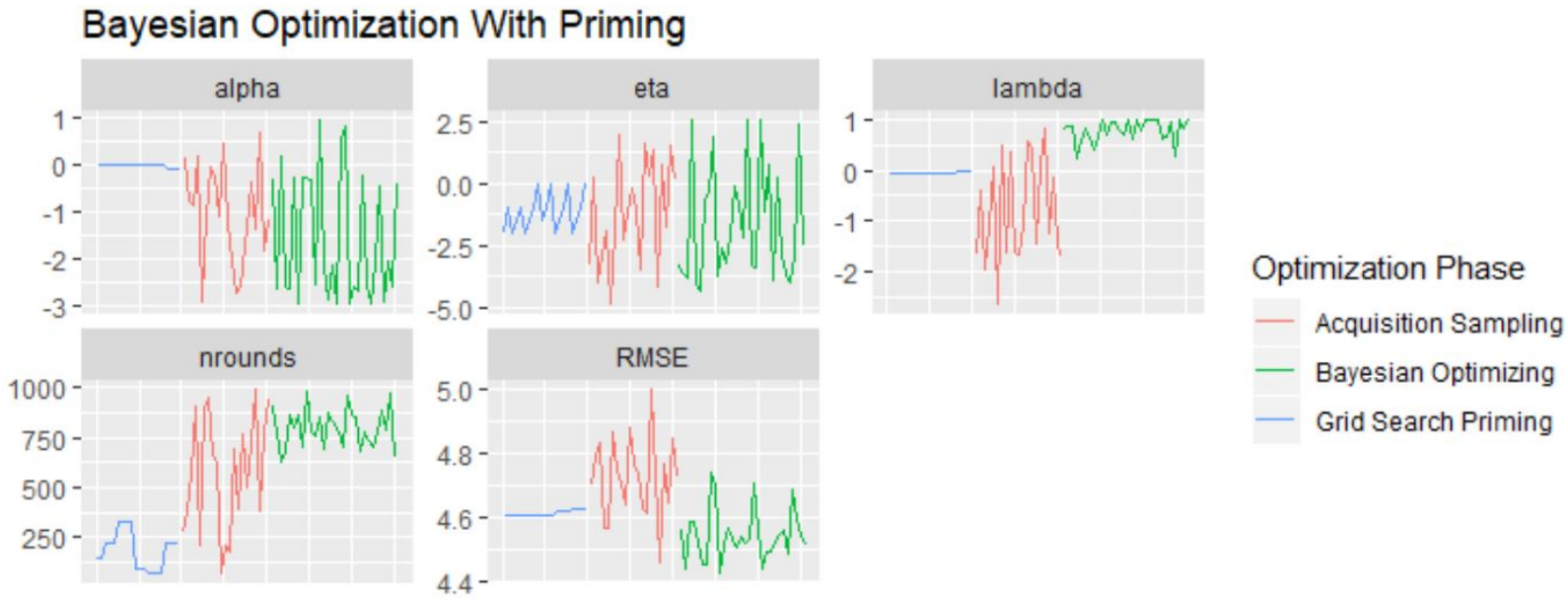


Particle Swarm Optimization



Bayesian Optimization

Primed Sampling With Bayesian Optimization



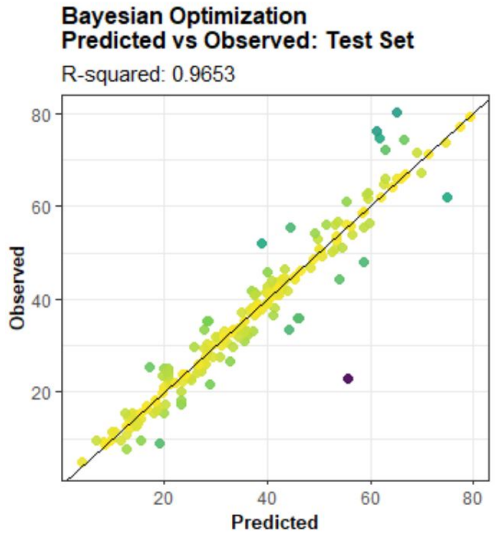
Bayesian optimization can be initialized randomly or with a subset of chosen hyperparameter values

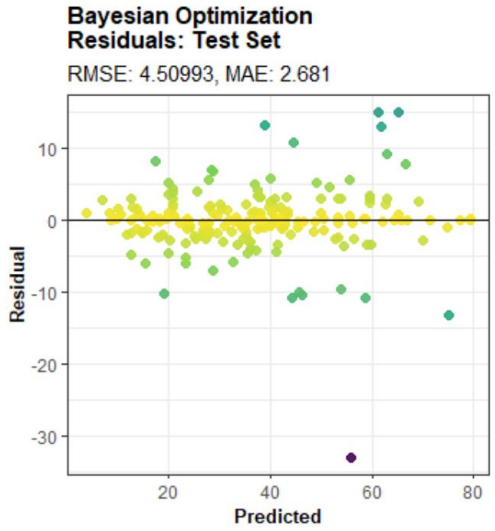
This run was primed with 20 of the first values from the adaptive random search to ensure the algorithm includes favourable regions of hyperparameters.

Training Metrics And Results

Method	RMSE	MAE	Rsquared	nrounds	eta	lambda	alpha	Processing Time
Grid Search	4.62847	3.06958	0.92303	150	0.03000	0.90000	1.00000	14 mins
Adaptive Random Search	4.68657	3.17616	0.92150	64	2.10476	0.34008	0.00045	5 mins
Genetic Algorithm	4.67208	3.08034	0.92125	464	0.59393	1.19523	0.35875	10 mins
Differential Evolution	4.48896	2.91747	0.92742	386	269.92122	8.71728	0.47104	55 mins
Particle Swarm Optimization	4.70503	3.12953	0.92044	682	2.01697	1.00000	1.00000	11 mins
Bayesian Optimization	4.45722	2.93983	0.92849	773	0.00018	10.00000	0.48359	22 mins

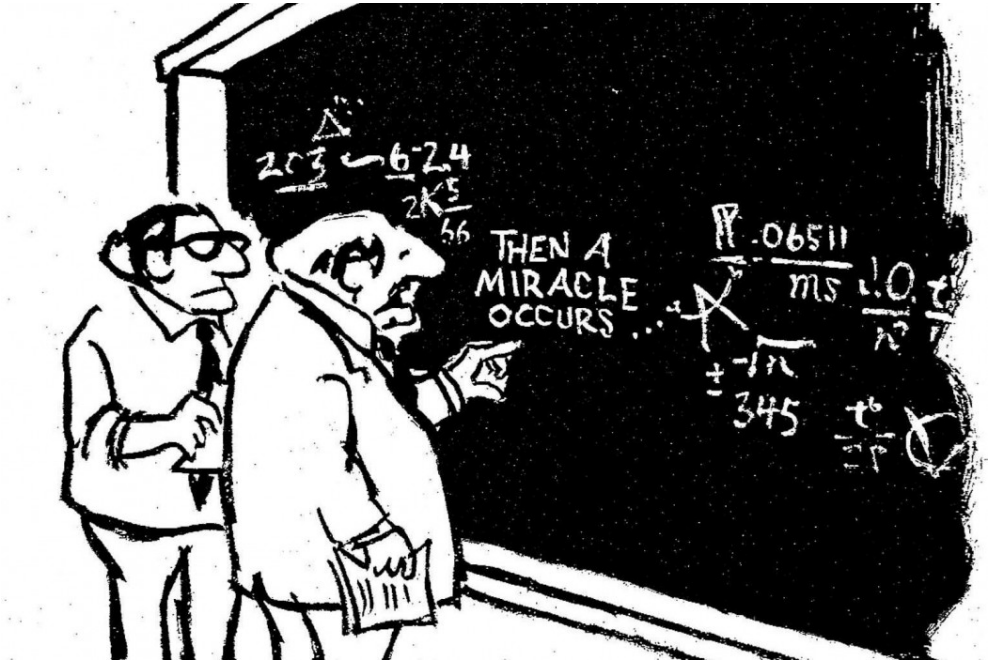
Test Metrics And Results





Method	RMSE	MAE	R-squared
Grid Search	4.96996	2.97559	0.9580
Random Search	5.10291	3.24413	0.9555
Genetic Algorithm	4.77734	2.78170	0.9611
Differential Evolution	4.63936	2.66680	0.9633
Particle Swarm Optimization	4.99273	3.01354	0.9577
Bayesian Optimization	4.50993	2.68133	0.9653

Final Recommendations



- Explore a wide domain with a grid search
- Transform linear hyperparameters or specify a logarithmic distribution
- Optimize `eta` first before other parameters
- Grid and random sampling sometimes don't generalize well
- Differential evolution can be time consuming
- Particle swarm optimization can show multiple minima
- Primed sampling With Bayesian Optimization showed the best performance and generalization

Questions

(Model Design + Hyperparameters) → Model Paramete

The building blocks:

- # Layers
- Activations
- Optimizers

...

The knobs that you can turn:

- Learning Rate
- Dropout

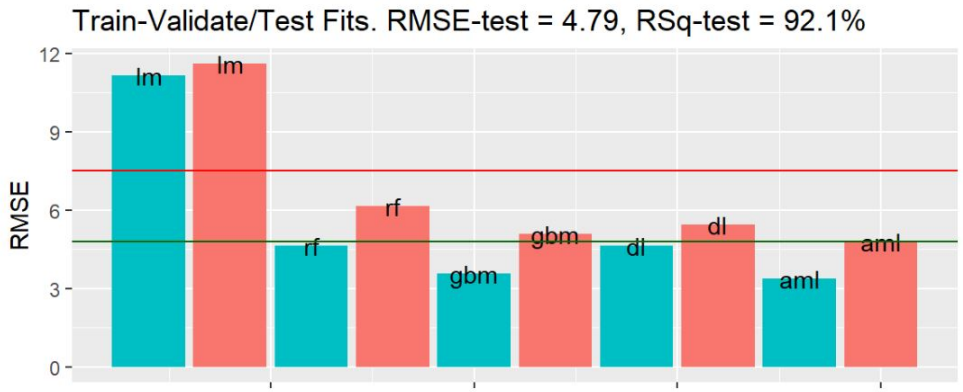
...

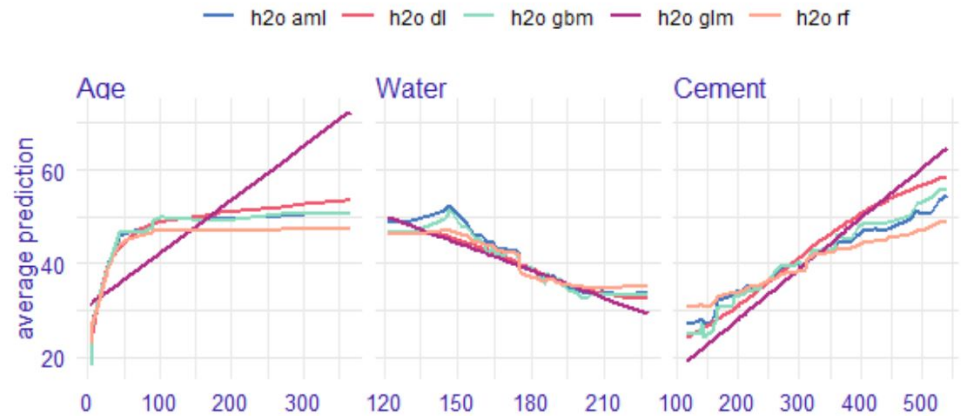
The variables learned from the data:

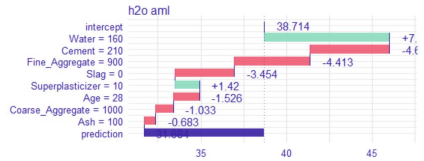
- weights

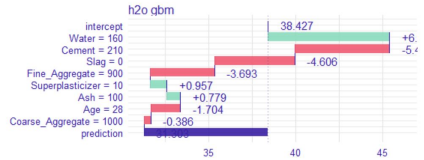
...

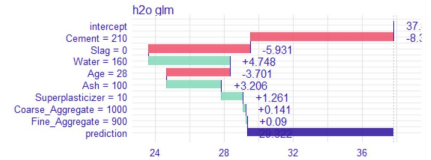
What other models did you try? Why XGBoost?

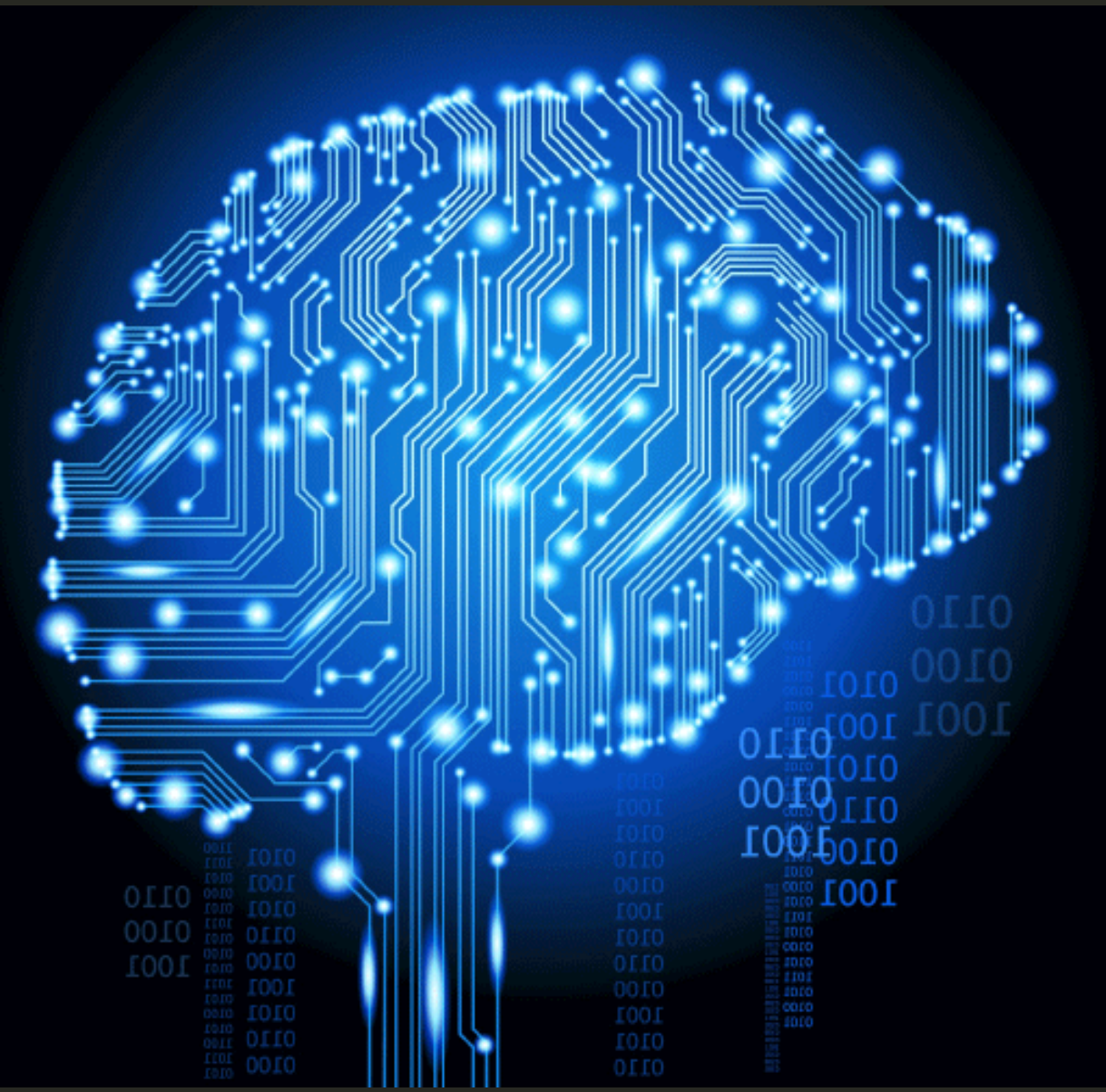












Thank you

Alastair Kerr Muir
AlastairKerrMuir@gmail.com
www.linkedin.com/in/alastairkerrmuir

This analysis, presentation and graphs were produced in using R, a programming language and software environment for statistical computing,

and the RMarkdown and the Xaringan packages.