Genetic Algorithms for Feature Selection

Kevin R. Coombes

June 9, 2010

Contents

1 Introduction

OOMPA is a suite of object-oriented tools for processing and analyzing large biological data sets, such as those arising from mRNA expression microarrays or mass spectrometry proteomics. The *ClassPrediction* package in OOMPA provides tools to help with the "class prediction" problem. Class prediction is one of the three primary types of applications of microarrays described by Richard Simon and colleagues. The point of these problems is to select a subset of "features" (genes, proteins, etc.) and combine them into a fully specified model that can predict the "outcome" for new samples. Here "outcome" may be a binary classification, a continuous variable of interest, or a time-to-event outcome.

Most prediction methods involve (at least) two parts:

1. feature selection: deciding which of the potential predictors (features, genes, proteins, etc.) to include in the model
2. model optimzation: selecting parameters to combine the selected features in a model to make predictions.

In this vignette, we illustrate the use of a *genetic algorithm* for feature selection.

2 Getting Started

No one will be surprised to learn that we start by loading the package into the current R session:

```
> library(GenAlg)
```

```
by using mclust, invoked on its own or through another package,
you accept the license agreement in the mclust LICENSE file
and at http://www.stat.washington.edu/mclust/license.txt
```

We also use some plotting routines from the *ClassDiscovery* package.

```
> library(ClassDiscovery)
```

3 The Generic Genetic Algorithm

The `GenAlg` class in the *ClassPrediction* library provides generic tools for running a generic algorithm. Here the basic idea is that we have a list of features, and we want to select a fixed number, k, of those features to include in a predictive model. Thus, a candidate solution consists of a vector of length k containing the indices of the features to be included. The genetic algorithm is intialized by creating a *population* of such candidate vectors, and storing it in a `data` matrix, which is passed as the first argument to the `GenAlg` function. After supplying all the necessary arguments (as described below) to this function, you update the population by calling the `newGeneration` function as many times as necessary. Each iteration computes the *fitness* of each candidate solution, selects pairs of individuals to *reproduce* (with more fit individuals being more likely to reproduce), and generates a new population that, on average, is expected to be more "fit" than the previous population.

The syntax of the `GenAlg` function is as follows:

```
> args(GenAlg)
```

```
function (data, fitfun, mutfun, context, pm = 0.001, pc = 0.5,
    gen = 1)
NULL
```

As just explained, `data` is the population matrix containing individual candidate solutions as its rows. You must also supply a fitness function (`fitfun`) and a mutation function (`mutfun`) customized to the current problem. The `context` argument is a list or data frame that is passed back to `fitfun` and `mutfun` to enable them to take advantage of any auxiliary information needed for them to perform their functions. The `pm` argument is the probability that an individual "allele" in the candidate solutions will mutate in any generation; the `pc` argument is the probability that crossover will occur during reproduction.

A common mutation rule for feature selection is that any included feature can mutate to any other potential feature. This rule is implemented by the following function, assuming that `context` is a data frame with one row per feature.

```
> selection.mutate <- function(allele, context) {
+     context <- as.data.frame(context)
+     sample(1:nrow(context), 1)
+ }
```

4 The Tour de France 2009 Fantasy Cycling Challenge

To illustrate the use of the feature-selection genetic algorithm, we turn from the world of genes and proteins to the world of professional cycling. As part of its coverage of the 2009 Tour de France, the Versus broadcasting network ran a "fantasy cycling challenge" on its web site at `http://www.versus.com/tdfgames`. The (simplified) rules of the challenge were to select nine riders for a fantasy team. Each player was given a fixed budget to work with; each rider "cost" a certain amount to include on the fantasy team. Players could change their selections during the first four stages of the tour. During stages 5 through 21, riders finishing in the top 15 positions were awarded points, with higher finishes garnering more points. Riders in the three "leaders jerseys" were awarded bonus points. We have put together a data frame containing the cost and scores of all riders who scored points for this contest during the 2009 tour. The data set can be loaded with the following command; Table 1 lists a few of the riders, their cost, and total score.

```
> data(tourData09)
```

	Cost	Scores	JerseyBonus	Total
Albert Timmer (Ned) Skil-Shimano	13	46	0	46
Alberto Contador Velasco (Spa) Astana	95	440	175	615
Alessandro Ballan (Ita) Lampre - NGC	41	92	0	92
Alexandre Botcharov (Rus) Team Katusha	8	30	0	30
Amaël Moinard (Fra) Cofidis, Le Credit en Ligne	6	59	0	59
Amets Txurruka (Spa) Euskaltel - Euskadi	8	91	0	91
Andreas Klöden (Ger) Astana	49	250	0	250
Andy Schleck (Lux) Team Saxo Bank	68	358	0	358
Angelo Furlan (Ita) Lampre - NGC	21	30	0	30
Bradley Wiggins (GBr) Garmin - Slipstream	16	325	0	325
Brice Feillu (Fra) Agritubel	5	198	5	203
Cadel Evans (Aus) Silence - Lotto	78	153	0	153
Carlos Sastre Candil (Spa) Cervelo TestTeam	81	59	0	59
Christian Knees (Ger) Team Milram	7	41	0	41
Christian Vande Velde (USA) Garmin - Slipstream	57	40	0	40

Table 1:

The specific challenge is to select nine riders, at a total cost of at most 470, who achieve the maximum total score. (Our task is easier than that of the participants in the contest, since they had to make their selections before knowing the outcome. With hindsight, we are trying to figure out what would have been the best choice.) Thus, we can define the *objective function* (or *fitness function*) for the genetic algorithm:

```
> scores.fitness <- function(arow, context) {
+     ifelse(sum(context$Cost[arow]) > 470, 0, sum(context$Total[arow]))
+ }
```

This objective function, `scores.fitness`, illustrates some conventions of the `GenAlg` class. First, the fitness function is always called with two arguments. The first argument, `arow`, is a vector of integers representing indices into the list of features being selected. In the present case, these represent indices into the `tourData09` data frame, and thus correspond to cyclists being considered for inclusion on the ultimate fantasy team. The second argument, `context`, is a list (or data frame) containing whatever auxiliary data is needed to

evaluate the fitness of this candidate team. When we actually initialize the `GenAlg` function, we will pass the `tourData09` data frame in as the `context` argument. In our problem, any team that costs more than 470 is invalid, and so it is given a fitness of 0. Otherwise, the fitness of the team is the cumulative total score that its riders achieved in the 2009 Tour de France.

Now we can initialize a starting population for the genetic algorithm. We will (arbitrarily) start with a population of 200 individual candidate solutions.

```
> set.seed(821813)
> n.individuals <- 200
> n.features <- 9
> y <- matrix(0, n.individuals, n.features)
> for (i in 1:n.individuals) {
+     y[i, ] <- sample(1:nrow(tourData09), n.features)
+ }
```

We are finally ready to initialize the genetic algorithm:

```
> my.ga <- GenAlg(y, scores.fitness, selection.mutate, tourData09,
+     0.001, 0.75)
```

The `summary` method reports the distribution of "fitness" scores:

```
> summary(my.ga)

An object representing generation 1 in a genetic algorithm.
Population size: 200
Mutation probability: 0.001
Crossover probability: 0.75
Fitness distribution:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  446.0   915.8  1164.0  1197.0  1428.0  2276.0
```

Now we can advance to the second generation:

```
> my.ga <- newGeneration(my.ga)
> summary(my.ga)

An object representing generation 2 in a genetic algorithm.
Population size: 200
Mutation probability: 0.001
Crossover probability: 0.75
Fitness distribution:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    468    1077    1304    1320    1537    2143
```

Notice that the mean, but not the maximum fitness, has increased.

Now we can advance through 100 generations:

```
> for (i in 1:100) {
+     my.ga <- newGeneration(my.ga)
+ }
> summary(my.ga)
```

```
An object representing generation 102 in a genetic algorithm.
Population size: 200
Mutation probability: 0.001
Crossover probability: 0.75
Fitness distribution:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   2055    2549    2682    2687    2831    3260
```

We can access the "best" fitness or the complete fitness vector from appropriate slots in the `GenAlg` object, `my.ga`.

```
> my.ga@best.fit
```

```
[1] 3260
```

```
> mean(my.ga@fitness)
```

```
[1] 2687.135
```

Since we also know which "individual" feature set gives the best score in this generation, we can retrieve it and get a list of cyclists for a pretty good fantasy team (Table 2).

```
> bestFound <- tourData09[my.ga@best.individual, ]
> bestFound <- bestFound[rev(order(bestFound$Total)), ]
```

	Cost	Scores	JerseyBonus	Total
Thor Hushovd (Nor) Cervelo Test Team	68	563	60	623
Mark Cavendish (GBr) Team Columbia - HTC	100	528	25	553
Jose Joaquin Rojas Gil (Spa) Caisse d'Epargne	12	399	0	399
Franco Pellizotti (Ita) Liquigas	54	337	45	382
Andy Schleck (Lux) Team Saxo Bank	68	358	0	358
Tyler Farrar (USA) Garmin - Slipstream	48	329	0	329
Bradley Wiggins (GBr) Garmin - Slipstream	16	325	0	325
Vincenzo Nibali (Ita) Liquigas	33	245	0	245
José Ivan Gutierrez Palacios (Spa) Caisse d'Epargne	6	46	0	46

Table 2: Best team found by a small genetic algorithm

5 Convergence

How can we tell if the answer found by the genetic algorithm is really the best possible? Actually, in the present instance, we know that the answer found in our limited application of the genetic algorithm is **not** the best possible. The winner of the Versus fantasy cycling challenge in 2009 scored $3,515$ points, which is more than the best solution we have found so far.

To explore the solution space more completely, we re-ran the genetic algorithm using a starting population of 1000 individuals (instead of the 200 used above). We let the solutions evolve through 1100 generations (instead of just 100). In order to allow this vignette to be produced in a timely fashion, we write down (but do not evaluate) the code to run the genetic algorithm in this way:

```
> set.seed(274355)
> n.individuals <- 1000
> n.features <- 9
> y <- matrix(0, n.individuals, n.features)
> for (i in 1:n.individuals) {
+     y[i, ] <- sample(1:nrow(tourData09), n.features)
+ }
> my.ga <- GenAlg(y, scores.fitness, selection.mutate, tourData09,
+     0.001, 0.75)
> output <- "Generations"
> if (!file.exists(output)) dir.create(output)
> recurse <- my.ga
> filename <- file.path(output, "gen0000.txt")
> assign("junk", as.data.frame(recurse), env = .GlobalEnv, immediate = T)
> write.csv(junk, file = filename)
> n.generations <- 1100
> diversity <- meanfit <- fitter <- rep(NA, n.generations)
> for (i in 1:n.generations) {
+     base <- ""
+     if (i < 1000) {
+         base <- "0"
+     }
+     if (i < 100) {
+         base <- "00"
+     }
+     if (i < 10) {
+         base <- "000"
+     }
+     filename <- file.path(output, paste("gen", base, i, ".txt",
+         sep = ""))
+     recurse <- newGeneration(recurse)
+     fitter[i] <- recurse@best.fit
+     meanfit[i] <- mean(recurse@fitness)
+     diversity[i] <- popDiversity(recurse)
+     cat(paste(filename, "\n"))
+     assign("junk", as.data.frame(recurse), env = .GlobalEnv,
+         immediate = T)
+     write.csv(junk, file = filename)
+ }
> save(fitter, meanfit, recurse, diversity, file = "gaTourResults.rda")
```

Instead, we load the saved results.

```
> data(gaTourResults)
```

In Figure 1, we plot the best fitness and the mean fitness as a function of the number of generations through which the genetic algorithm has been allowed to evolve. The maximum score that we ever achieve is 3978, and the fantasy cycling team that gives this score is shown in Table 3.

```
> newBest <- tourData09[recurse@best.individual, ]
> newBest <- newBest[rev(order(newBest$Total)), ]
```

	Cost	Scores	JerseyBonus	Total
Thor Hushovd (Nor) Cervelo Test Team	68	563	60	623
Alberto Contador Velasco (Spa) Astana	95	440	175	615
Mark Cavendish (GBr) Team Columbia - HTC	100	528	25	553
Gerald Ciolek (Ger) Team Milram	46	413	0	413
Jose Joaquin Rojas Gil (Spa) Caisse d'Epargne	12	399	0	399
Franco Pellizotti (Ita) Liquigas	54	337	45	382
Rinaldo Nocentini (Ita) AG2R La Mondiale	16	139	200	339
Tyler Farrar (USA) Garmin - Slipstream	48	329	0	329
Bradley Wiggins (GBr) Garmin - Slipstream	16	325	0	325

Table 3: Best team found by an extensive genetic algorithm

In this case, we can *prove* that this team yields the optimal score. The selected team includes 9 of the 11 highest scoring individual cyclists in the 2009 Tour de France. The two omitted cyclists are Oscar Freire (cost = 82; total = 369) and Andy Schleck (cost = 68; total = 358. Replacing one of the three cyclists on the selected team with lower scores than these two riders would increase the cost above the cap of 470, and so no single change can improve the score. We also consider the possibilty of replacing the two lowest scoring cyclists on the best selected team with one of these high scorers and one other cyclist. Since the two lowest scoring cyclists have a combined cost of $48 + 16 = 64$ and the total cost for the best team is $455 = 470 - 15$, we would need to find two replacements whose combined cost is at most $64 + 15 = 79$. Since the cost to include Oscar Freire already exceeds this cap, we must find another cyclist to pair with Andy Schleck. So, the cost is bounded by $79 - 68 = 11$, while the total score must exceed $329 + 325 - 358 = 296$. But there are no riders in the database meeting these conditions.

5.1 Lessons for Fantasy Cylists

The "ultimate" fantasy cycling team for the 2009 Tour de France, as shown in Table 3, leads to several conclusions for how to pick a team for this competition. First, the team should be heavily weighted toward sprinters. Most of the 20 teams in the 2009 tour had one sprinter among their nine riders; the fantasy team has five sprinters (Hushovd, Cavendish, Ciolek, Rojas, and Farrar) out of nine. Second, riders who wear leaders jerseys are valuable. During the 2009 tour, three riders wore the "yellow jersey" for the overall leade: Fabian Cancellara, Rinaldo Nocentini, and Alberto Contador; two of those three are on the best fantasy team. The 'green jersey" for most consistent rider was shared by Mark Cavendish and Thor Hushovd, both on the best fantasy team. The "polka dot jersey" for best rider in the mountains was worn for nine stages by Franco Pellizotti. The final lesson is to avoid "breakaway" specialists in favor of riders who can win sprints or place high in the "general classification" for overlall best time. Riders who win breakaways are not likely to score well in more than one stage in a tour; that does not provide enough points to put them among the overall leaders. (The exception here is Rinaldo Nocentini, whose one breakaway win put him in the yellow jersey lead for eight days.)

5.2 Lessons for Convergence of Genetic Algorithms

Based on Figure 1, the optimal score is first achieved around generation 400. However, the population temporarily evolves away from this score at about generation 630, but then comes back. Interestingly, however, the mean fitness decreases for a period even after the maximum appears to stabilize. Taken as a whole, this figure suggests that neither tracking the maximum nor the mean fitness, by themsleves, gives a reliable measure of the convergence of the genetic algorithm. Note, by the way, that simple mutations (which

replace one feature with another) can have a large effect on the fitness, and thus can have a strong influence on the mean.

One possibility is to look at smoothed changes in the mean and maximum fitness. The next commands use the `loess` function to fit a smooth approximation to the changing values of the mean and maximum fitness per generation. A scatter plot of these smooth curves is displayed in Figure 2. This figure suggests that, at about generation 800, the best fitness stops changing, while the mean fitness continues to increase slowly. It also suggests that most of the gain in finding the optimal solution occured in the first 300 to 400 generations.

```
> n.generations <- length(meanfit)
> index <- 1:n.generations
> lo <- loess(meanfit ~ index, span = 0.08)
> lof <- loess(fitter ~ index, span = 0.08)
```

An alternative is to measure the "diversity" of the population. We define the "distance" between two individuals in the population to be the number of alleles that are different. In other words, we count the number of different features that the two candidate solutions would select. This measure of distance defines an *ultrametric* on the space of candidate feature selection solutions, analogous to the Hamming distance between binary strings. We define the diversity of the population to be the average of the distance between all pairs of individuals. This measure of diversity is implemented in the `popDiversity` function in the *ClassPrediction* package; we used this function above when evolving the population with the genetic algorithm.

Figure 3 overlays the diversity on a plot of the best and mean fitnesses. We note first that the average diversity appears to be negatively correlated with the mean fitness.:

```
> cor(diversity, meanfit)

[1] -0.9863062

> cor(diversity, meanfit, method = "spearman")

[1] -0.9935457
```

This strong correlation suggests that we might be able to use the mean fitness to draw conclusions about whether the algorithm has converged. One reason to prefer the diversity, however, is that it is more interpretable. For instance, we note that the diversity starts in genration zero at a value between eight and nine. This observation makes sense; two sets of nine riders selected randomly from among the 102 riders who scored points in the 2009 tour challenge should almost never have more than one overlap. At the point where we first reach the best solution, the average diversity falls to about 2, showing that most pairs of solutions have seven riders in common. Howevre, the diversity is near 1.5 during the period when the best solution temporarily drifts away from the optimum. During the final phase, however, the diversity drops below 1.25 and stays near that level, suggesting that this might provide a reasonable criterion for convergence. An additonal measure that might be applied when the diversity starts to fall into this range is to compute the number of individuals in the population that are identical to the best solution:

```
> temp <- apply(recurse@data, 1, function(x, y) {
+     all(sort(x) == sort(y))
+ }, recurse@best.individual)
> sum(temp)

[1] 297
```

In this case, rhe best solution is represented by 297 of the 1000 individual candidates.

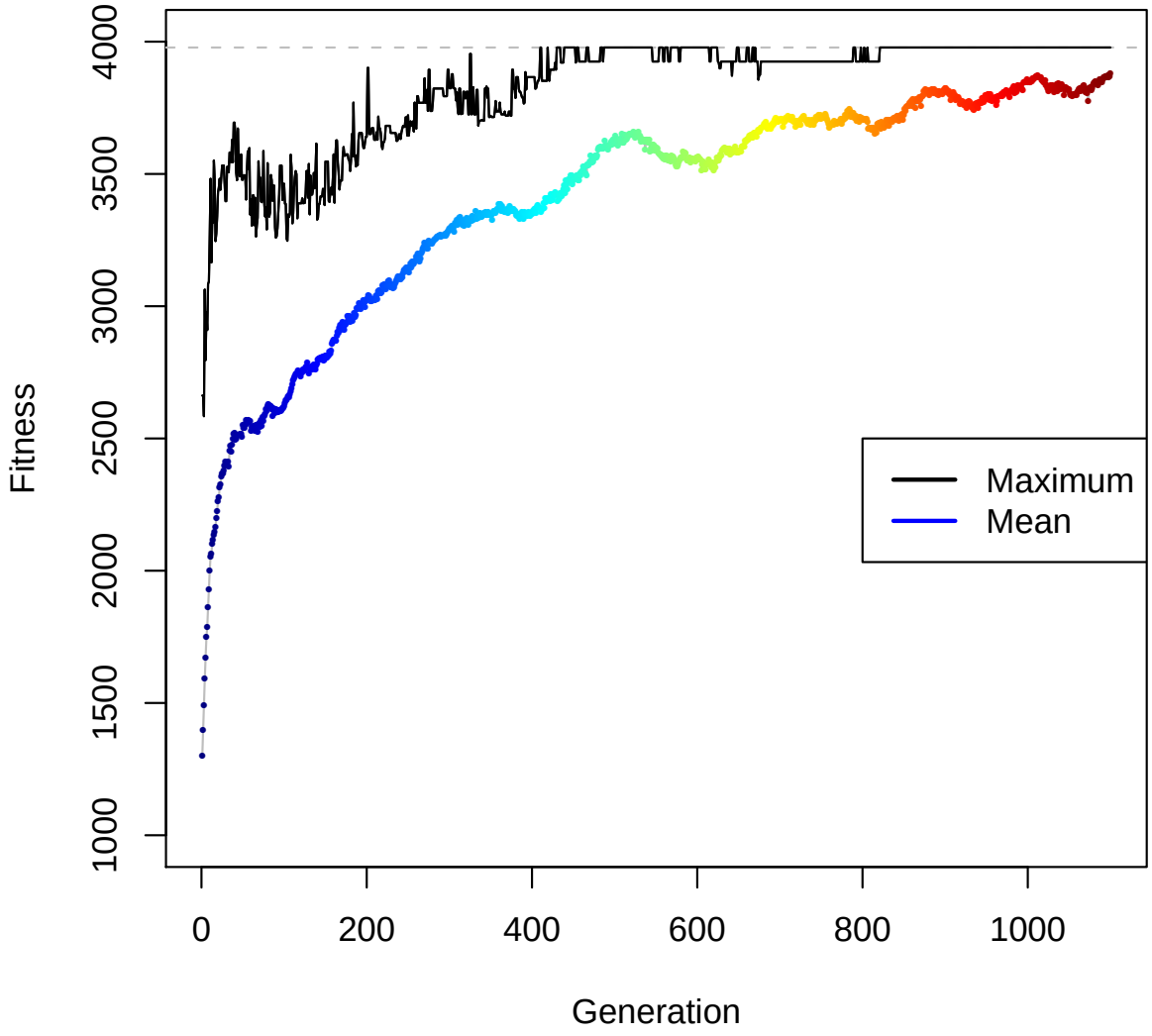

Figure 1: Maximum and mean fitness found in each generation.

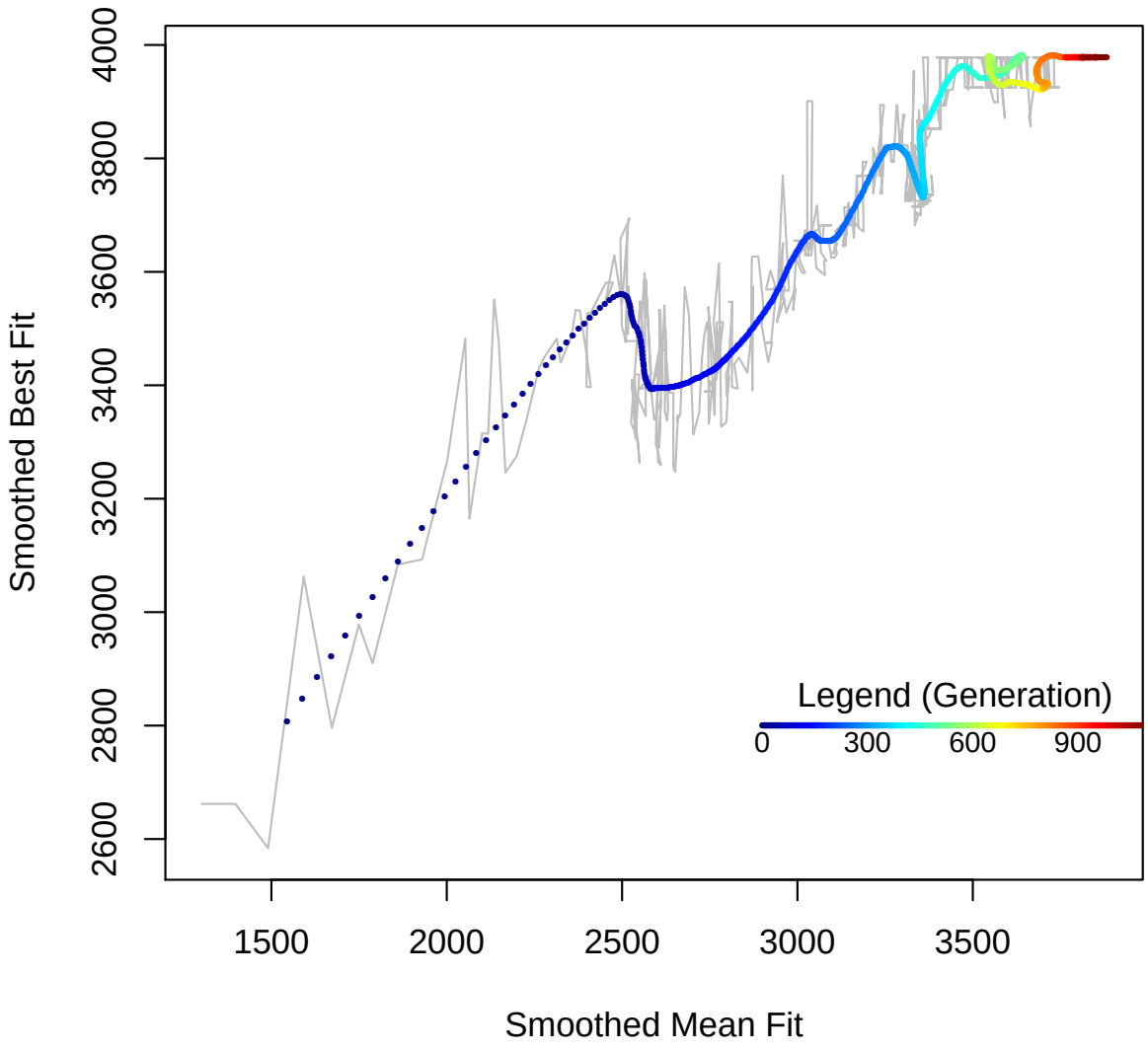

Figure 2: Relationship between the mean and maximum fitness in a population of potential solutions as the geneerations evolve. Gray curve tracks the complete fluctuations; colored dots follow the smoothed curves, with colors the same as in Figure 1.

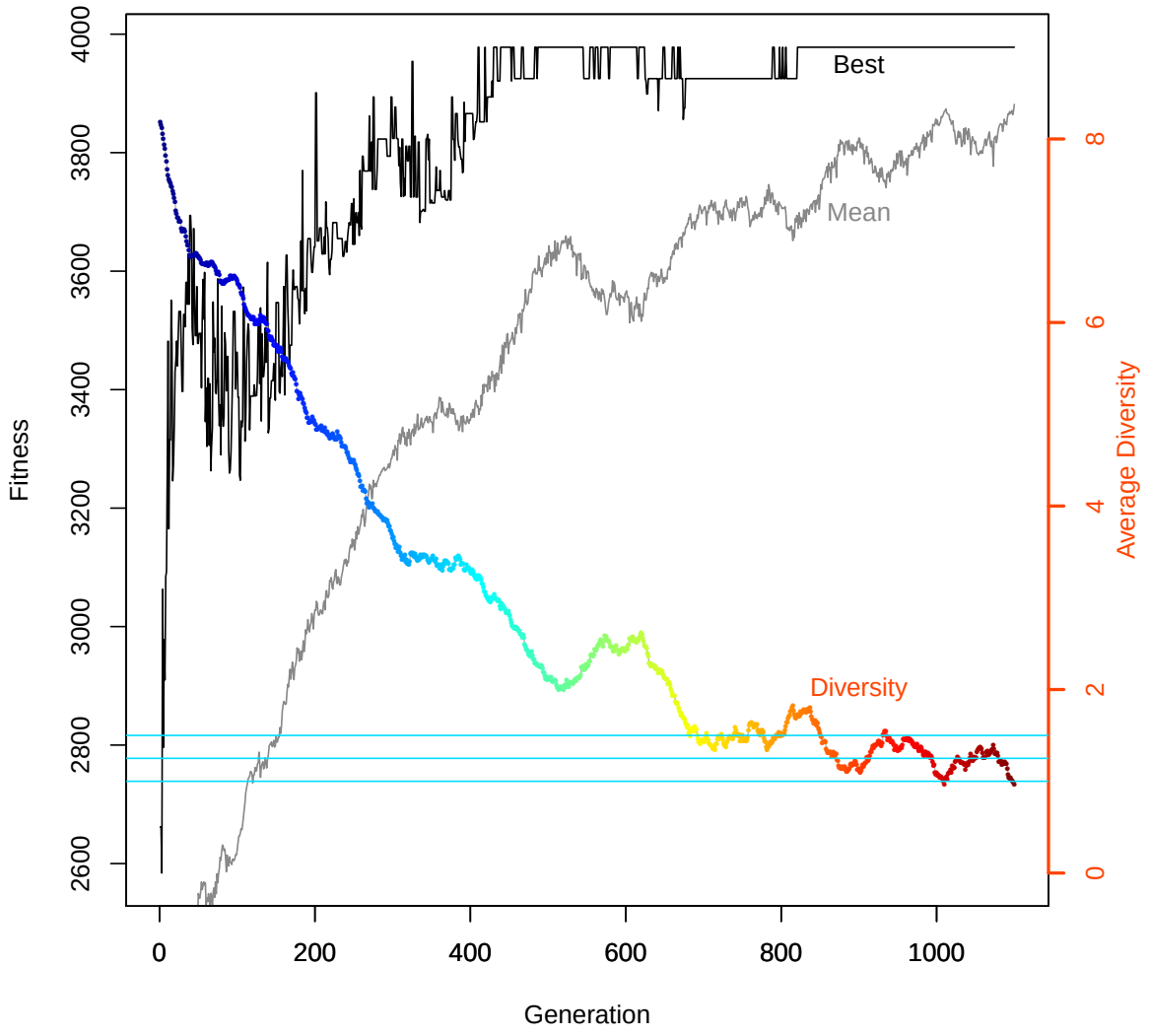

Figure 3: Overaly of the average diversity (color) on plots of the best (black) and mean (gray) fitness through the generations of the genetic algorithm. Horizontal (cyan) lines are locatyed at a diversity of 1, 1.25, and 1.5.

6 Implications for Gene Expression Signatures

The example presented here has some important implications for applying feature selection to develop gene expression signatures to predict useful clinical outcomes. The most significant issues relate to the problem of convergence. The cycling example is likley to be easier than the gen epxression problem, since there are only 102 "cycling-features" (i.e., riders) while there may be several thousand gene-features. In order to search adequately through the sample space, we needed a population of 1000 individuals evolved through about 1000 generations. (Even though that means we evaluated up to $1,000,000$ candidate solutions, this is still a small number compared to the 2×10^{12} ways to choose nine riders from a list of 102.) Unless some preliminary filtering ir performed to reduce the number of genes, it will probably take a much larger population and many more generations to search through gene-space for useful predicitve features.

We do expect, however, that the "diversity" wil prove useful to monitor convergence even in the gene expression setting. As mentioned earlier, diversity has the advantage fo providing an interpretable condition for converence, which doe s not need to know the "fitness" of the optimal solution in advance. By contrast, we know that the fitness function must will be different in the gene epxression case from the fitness function used for the fantasy cycling challenge. The underlying difficulty is that we do not have as clear an objective score. Instead, we can start by considering the problem of selecting features in order to classify samples into two different groups. If we use something like linear discriminant analysis (LDA), or an equivalent linear model, to separate the groups, then a natural measure of the fitness of a set of gene-features is the distance between the centers of the two groups in the resulting multivariate space. This measure is known as the Mahalanobis distance, and is implemented by the `maha` function in the *ClassPrediction* package. A fitness function that is adequate for use in the constructor of a `GenAlg` object is then provided by the function:

```
> mahaFitness <- function(arow, context) {
+     maha(t(context$dataset[arow, ]), context$gps, method = "var")
+ }
```

where the appropriate `context` object consists of a list containing the gene expression dataset and a vector identifying the two groups.