Support Vector Machines

Lecture 4 –John Kornak – Epidemiology and Biostatistics

In this lecture:

- History of Support Vector Machines (SVM)
- Maximal Margin Classifier
- Support Vector Classifier
- Support Vector Machine
- Coding in R
- Examples in R
- A few slides on the idea of optimization

History of SVM

- Developed by the computer science community
- Original SVM algorithm was invented by Vladimir N. Vapnik and Alexey Ya. Chervonenkis in 1963 (but the computing had to wait until Vapnik emigrated to US in the 90s – won character recognition competition)
- Extended to nonlinear classifiers via the “kernel trick” by Bernhard E. Boser, Isabelle M. Guyon and Vladimir N. Vapnik in 1992
- Current “soft margin” approach proposed by Corinna Cortes and Vapnik in 1993 and published in 1995
- SVMs gained massive popularity in the 1990s

Motivation

- The *maximal margin classifier*, *support vector classifier* and *support vector machine* are designed for the classification of binary data (supervised learning)
- The idea is to generate boundaries (lines, planes, and hyper-planes) in the dataset that separate the data points in *feature space* i.e. with respect to the predictors, into the original groups/classes.
- When complete separation is not possible we can 1) soften what we mean by *separate* (support vector classifier) and 2) expand the feature space (support vector machine)

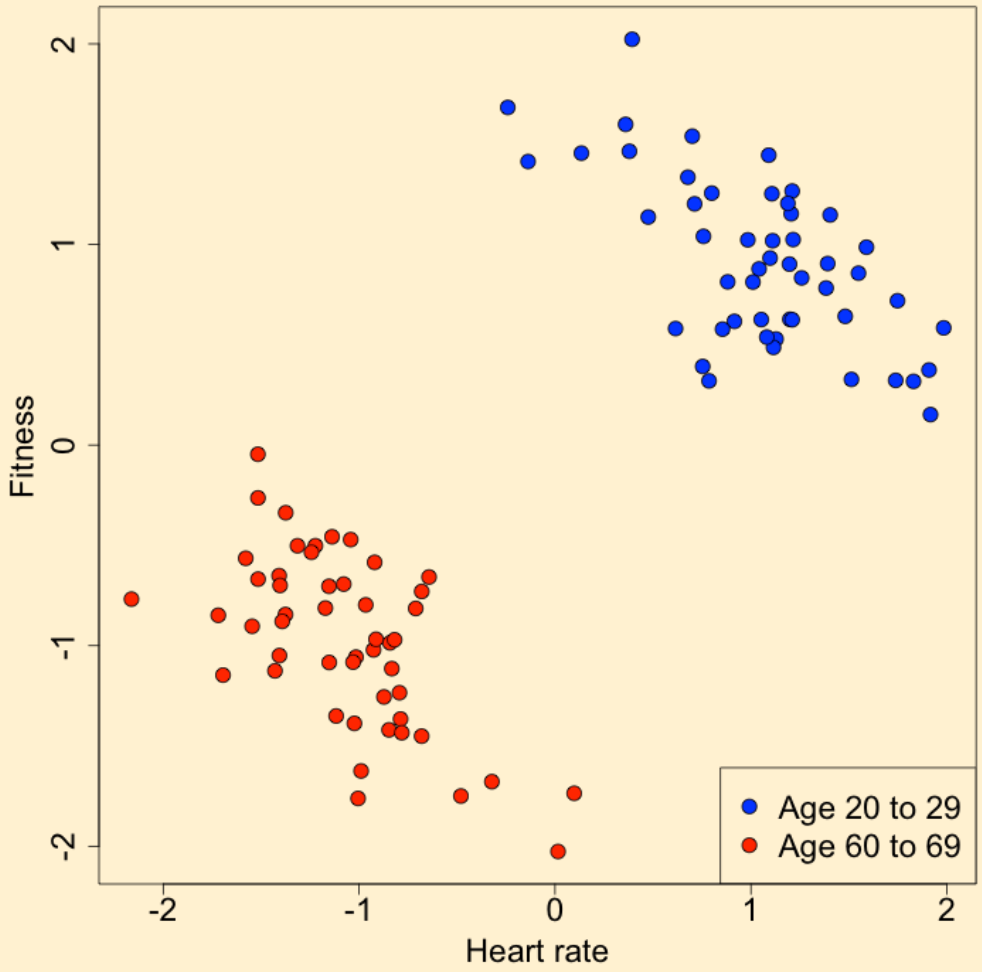
Motivating training dataset

- Simulated training data
- Heart rate (normalized)
- Fitness measure (normalized)
- Age group – 60-69 (group 0) and 20-29 (group 1)
- Data available as "HeartFitnessSim.csv" [use `read.csv("HeartFitness.csv")`] to read into R
- Set ageGrp as a factor using `factor(ageGrp)`
- Data simulated in "svm.R"

```
> dat
      HeartRate     Fitness ageGrp
1   -0.78831193 -1.36587381     -1
2   -0.96613699 -0.79682621     -1
3   -0.68085179 -1.45126330     -1
4   -1.51668789 -0.04591166     -1
5   -1.42939942 -1.12560827     -1
                    ⋮
97  -0.24121523  1.68305725      1
98   1.18869903  1.20473700      1
99   1.51317669  0.32726619      1
100  1.83007684  0.31751328      1
>
```

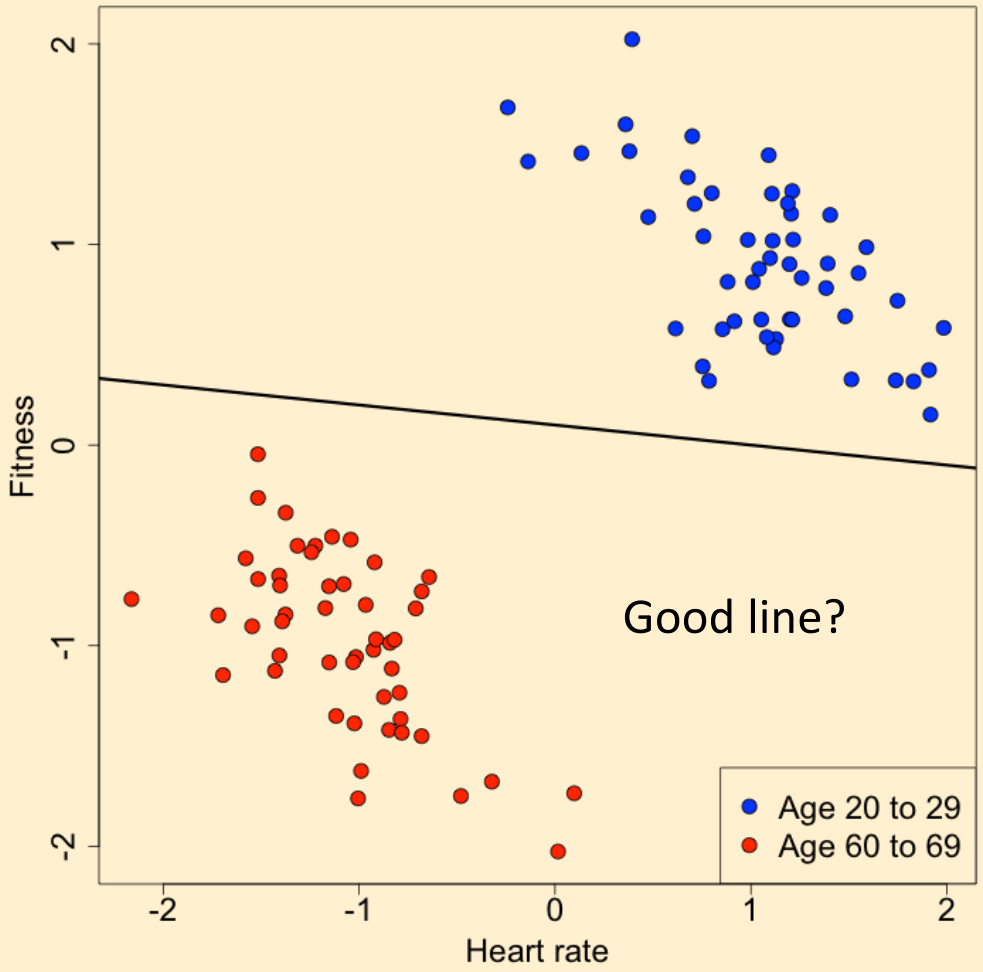
Motivating training dataset

- Simulated data
- Training data plot of fitness vs heart rate (standardized)
- Two groups: young vs. old
- Want a rule to classify future observations
- Draw a line to separate the groups



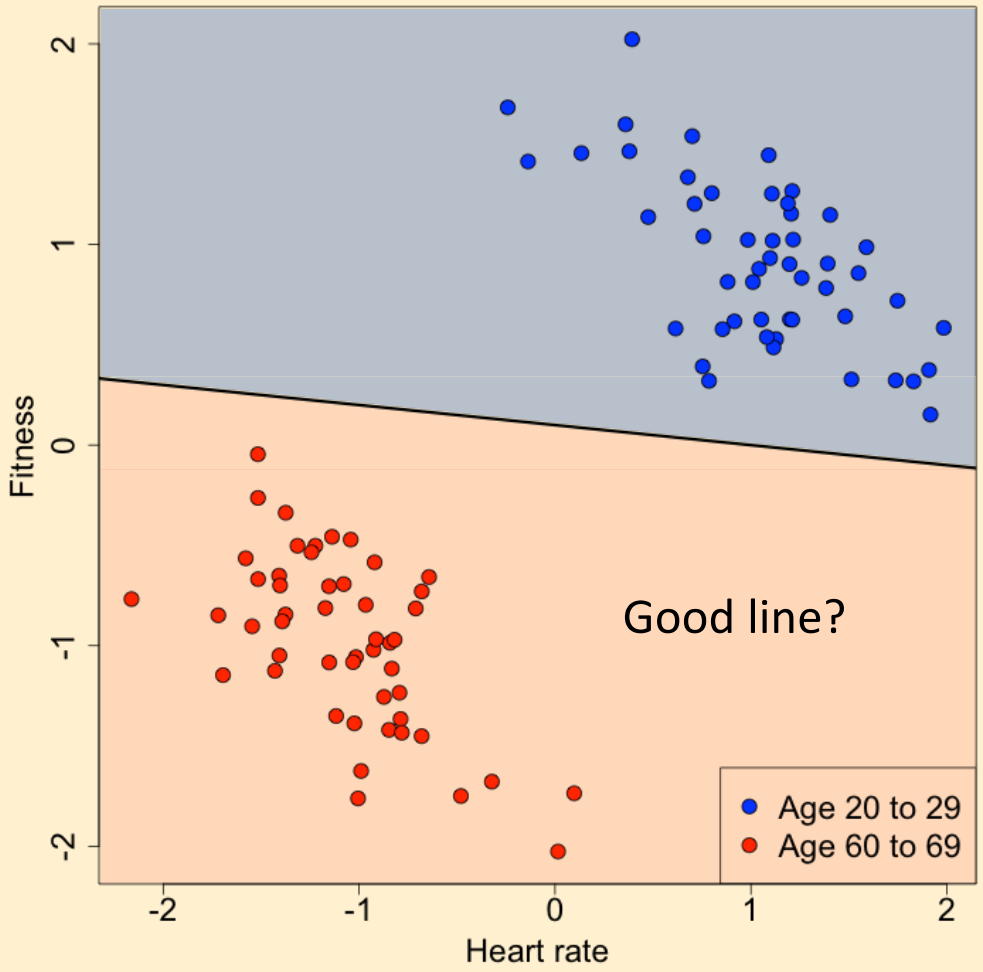
A separating line

- Simulated data
- Training data plot of fitness vs heart rate (standardized)
- Two groups: young vs. old
- Want a rule to classify future observations
- Draw a line to separate the groups



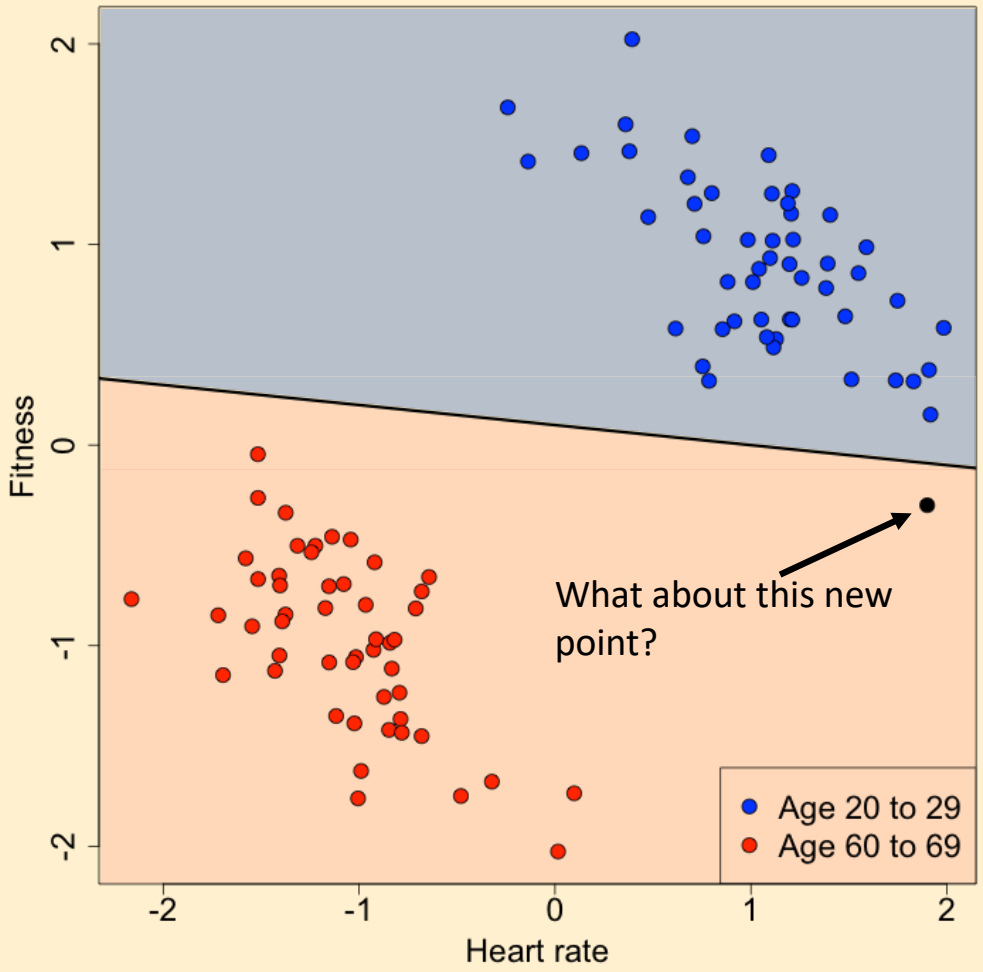
A separating line

- Simulated data
- Training data plot of fitness vs heart rate (standardized)
- Two groups: young vs. old
- Want a rule to classify future observations
- Draw a line to separate the groups



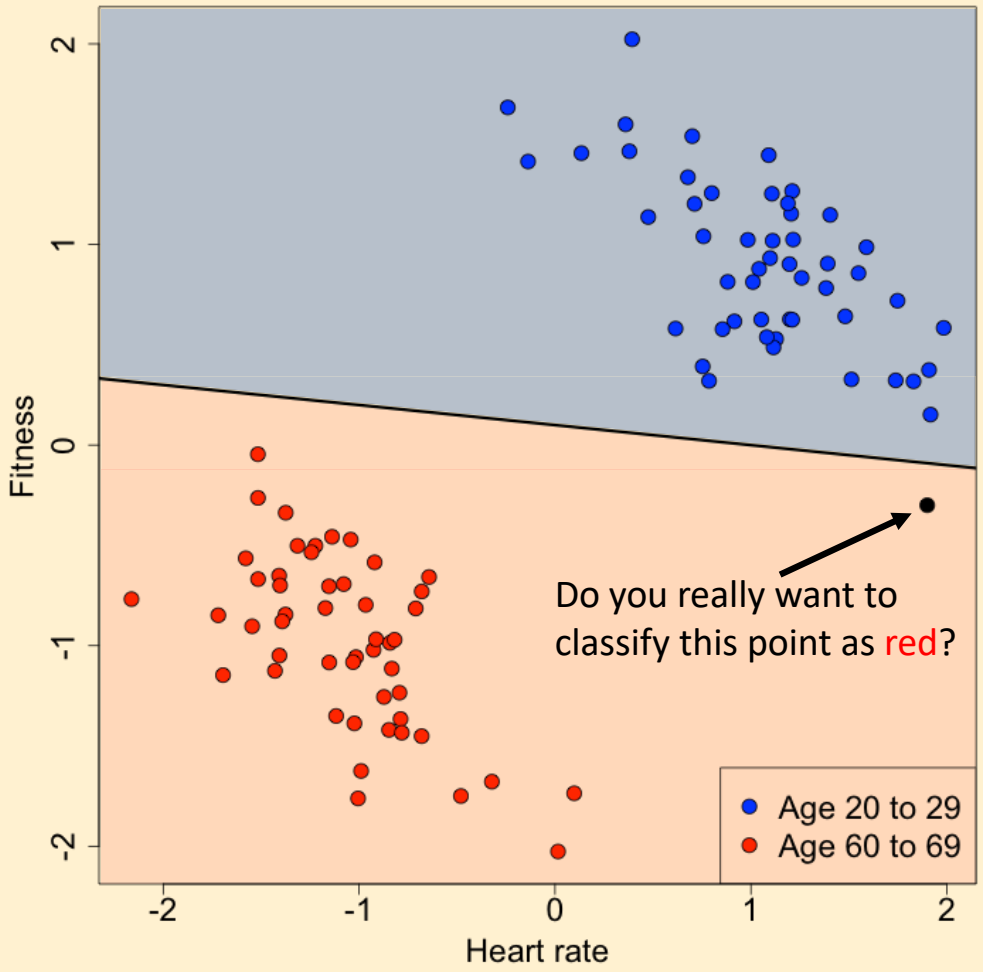
A separating line

- Simulated data
- Training data plot of fitness vs heart rate (standardized)
- Two groups: young vs. old
- Want a rule to classify future observations
- Draw a line to separate the groups



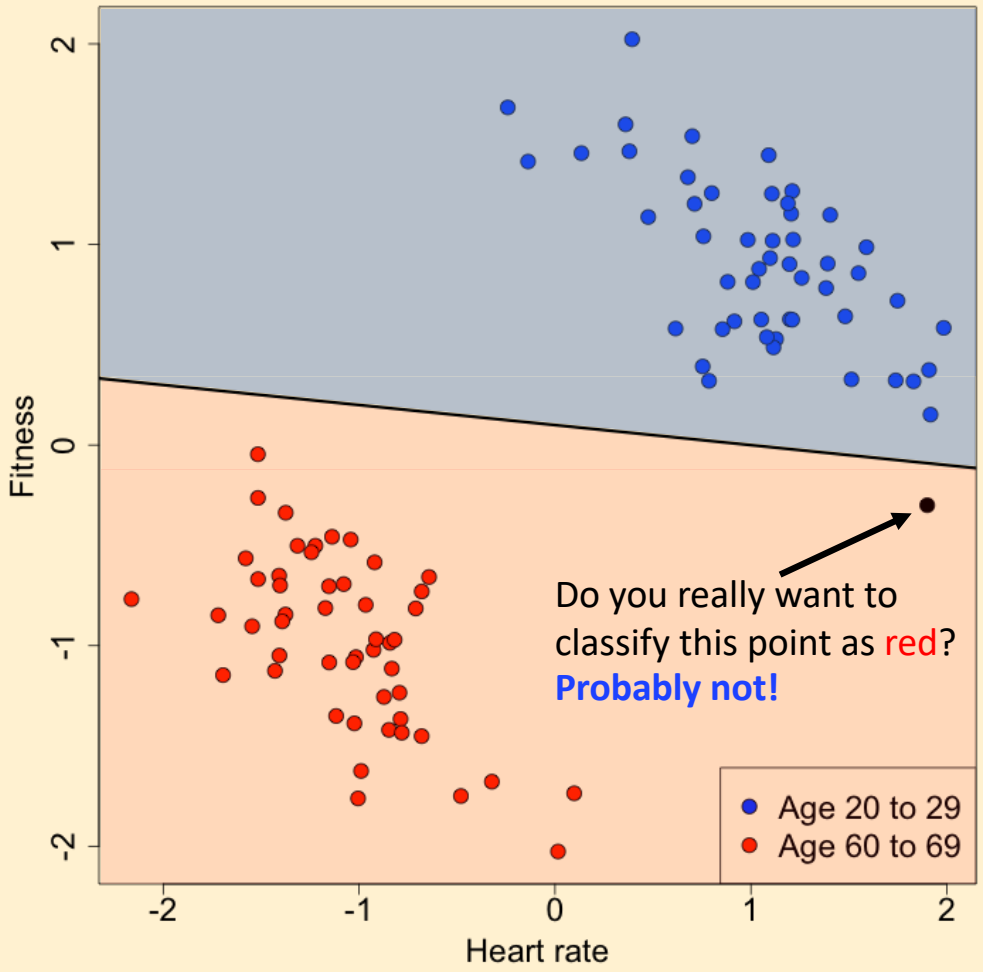
A separating line

- Simulated data
- Training data plot of fitness vs heart rate (standardized)
- Two groups: young vs. old
- Want a rule to classify future observations
- Draw a line to separate the groups



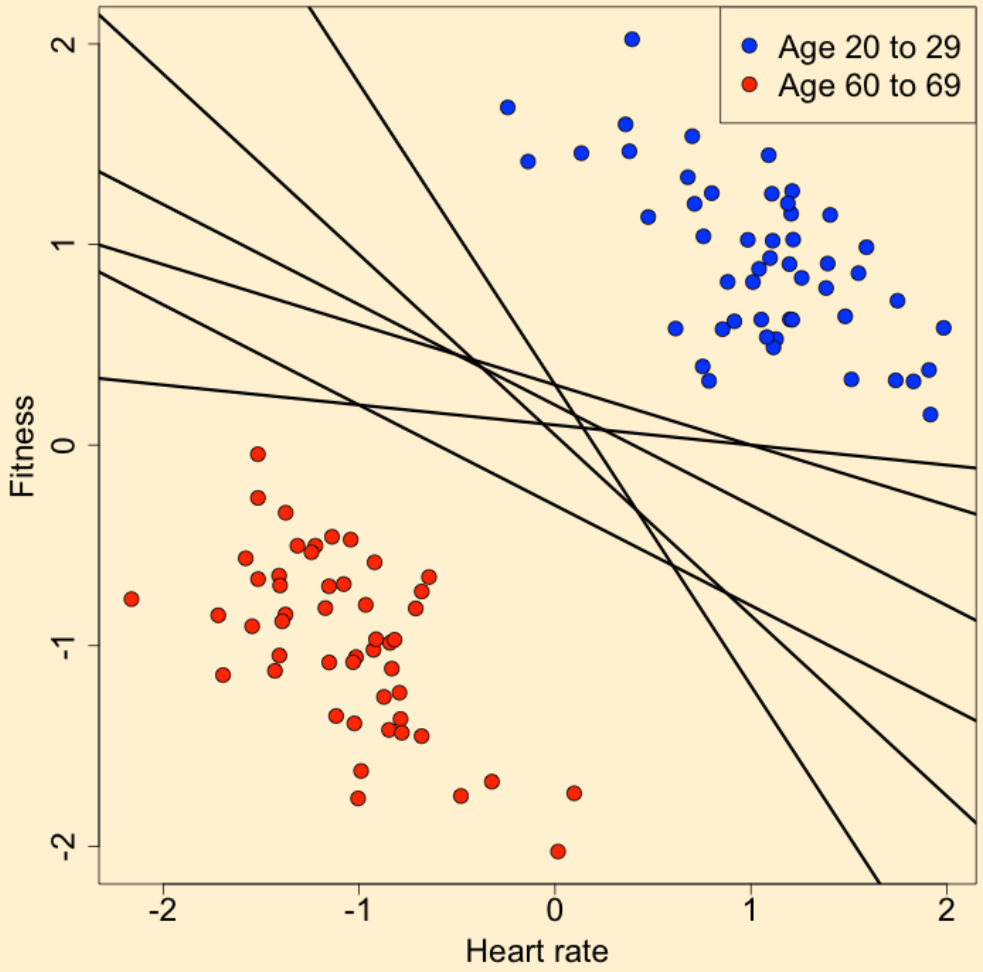
A separating line

- Simulated data
- Training data plot of fitness vs heart rate (standardized)
- Two groups: young vs. old
- Want a rule to classify future observations
- Draw a line to separate the groups



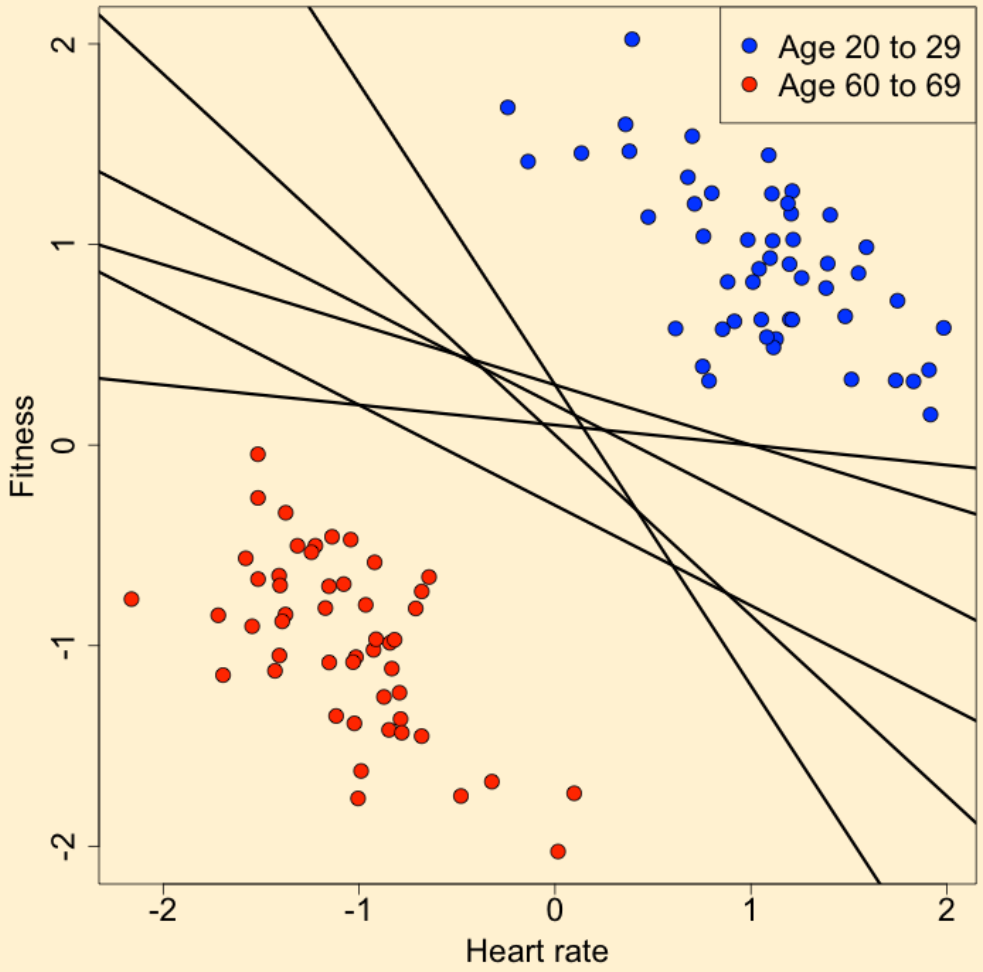
Many separating lines

- Simulated data
- Training data plot of fitness vs heart rate (standardized)
- Two groups: young vs. old
- Want a rule to classify future observations
- Draw a line to separate the groups



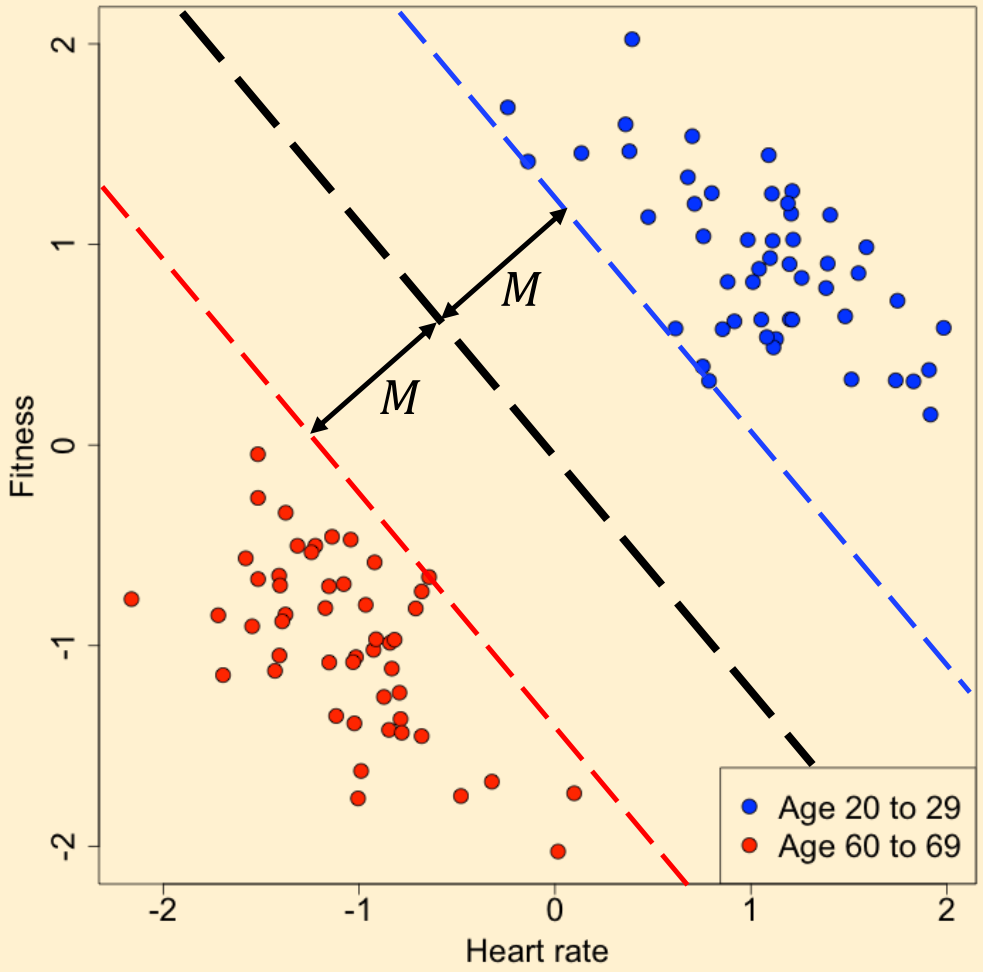
Many separating lines

- So how should we choose a line?



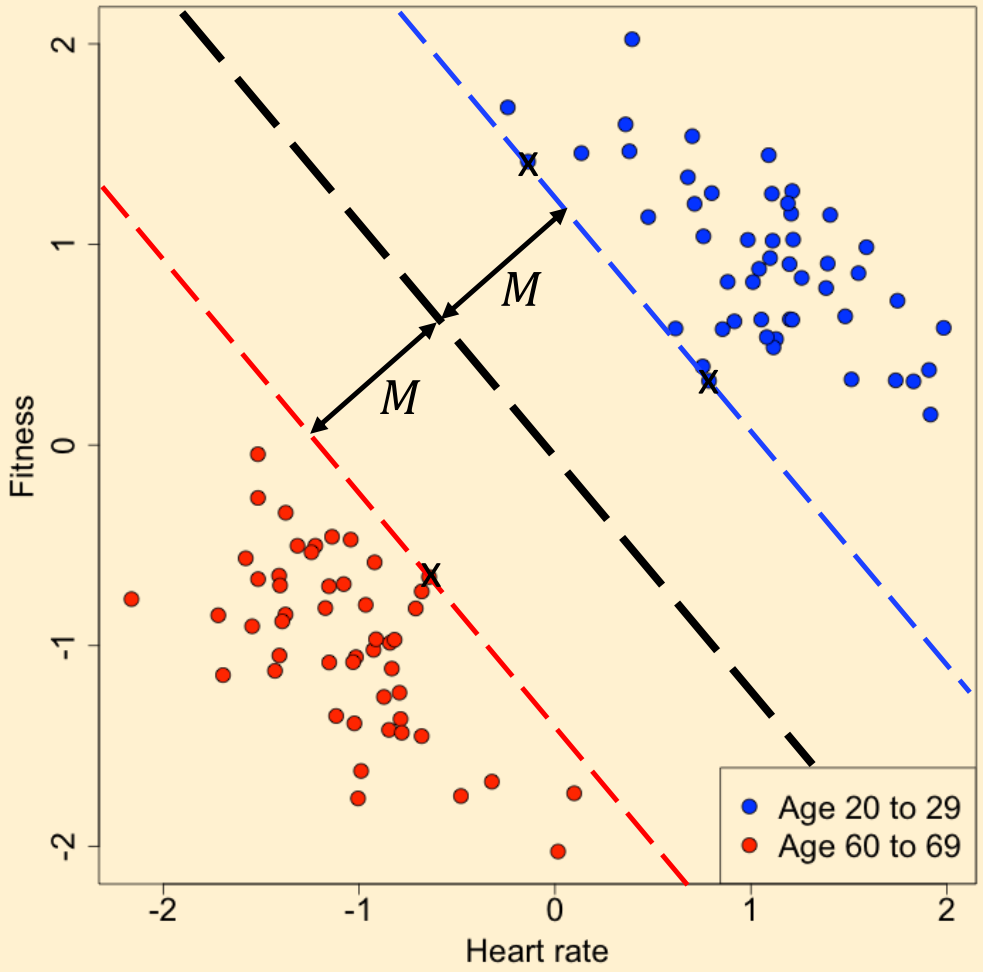
Maximal margin classifier

- So how should we choose a line?
- Maximal margin classifier finds the separating line that is farthest away from any training points (the margin M)



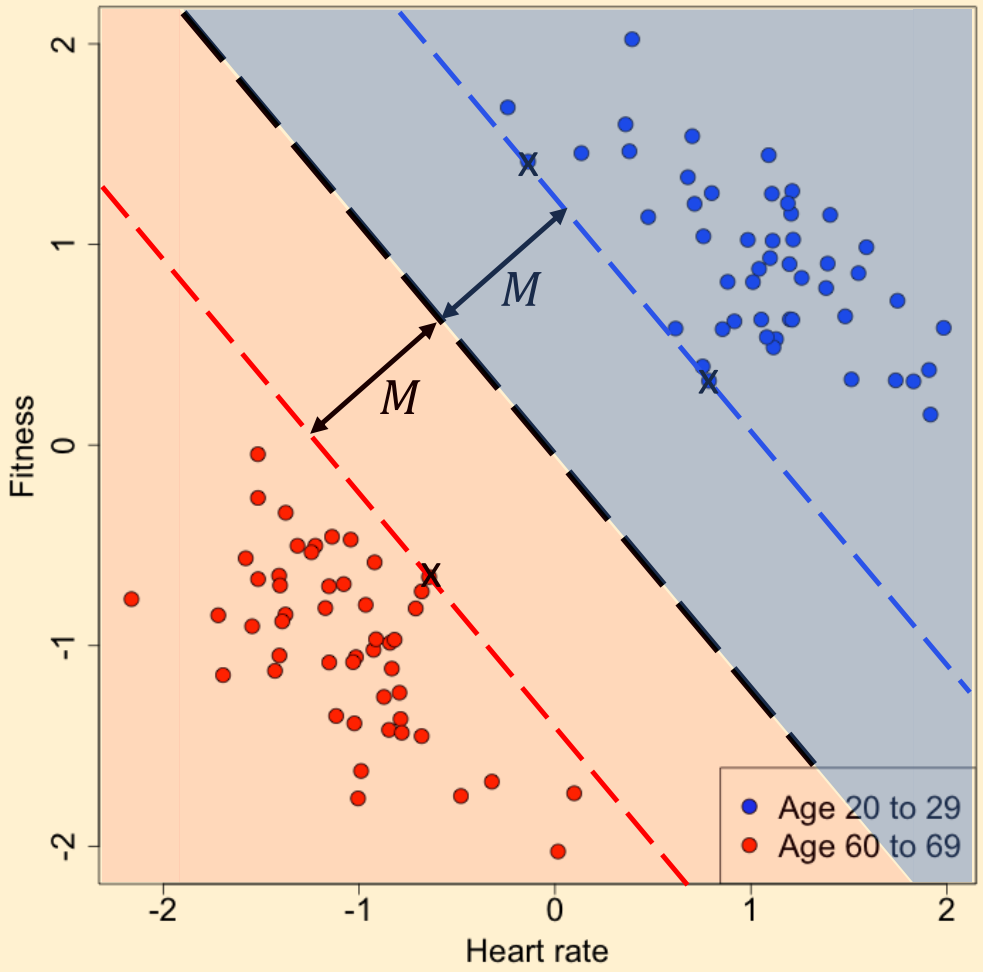
Maximal margin classifier

- So how should we choose a line?
- Maximal margin classifier finds the separating line that is farthest away from any training points (the margin M)
- The 3 points marked by "x" are the *support vectors*
- Only depends on 3 points!!!



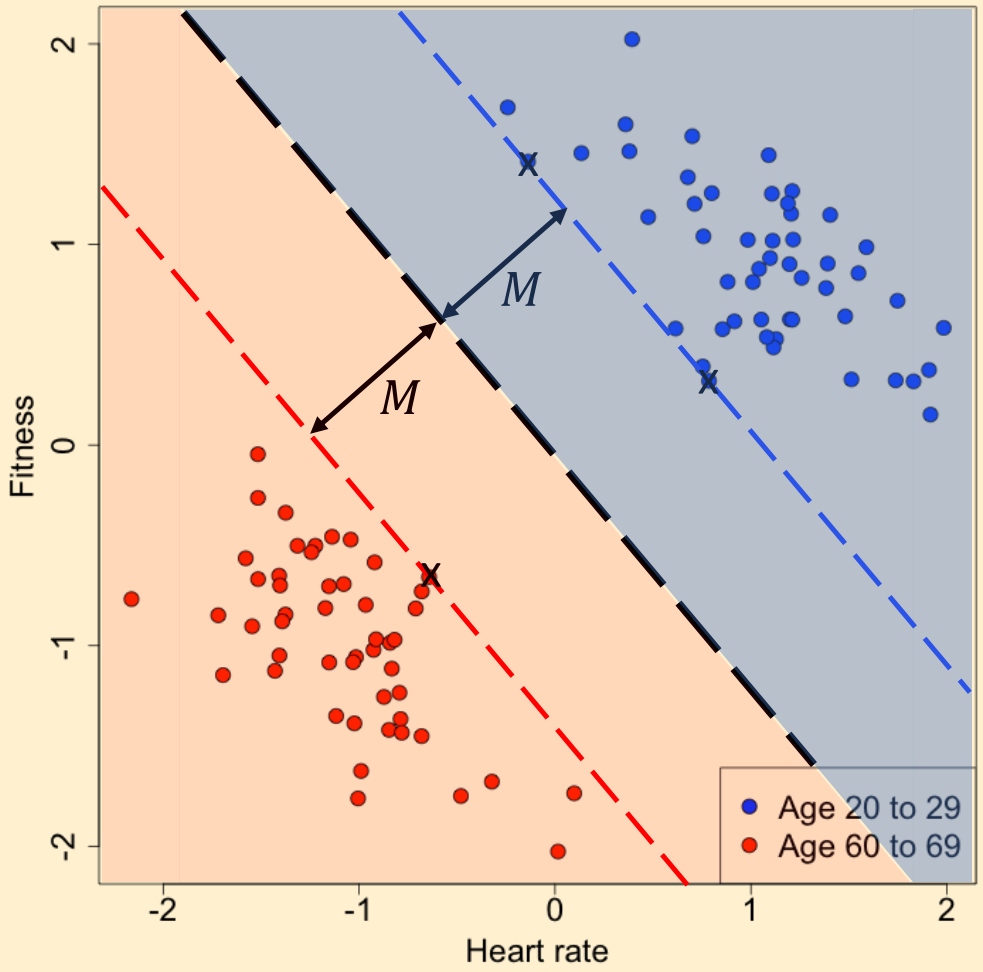
Maximal margin classifier

- So how should we choose a line?
- Maximal margin classifier finds the separating line that is farthest away from any training points (the margin M)
- The 3 points marked by "x" are the *support vectors*
- Only depends on 3 points!!!
- Red and blue areas indicate the *decision rule* made by the classifier for new observations based on the *separating line*
- So a new observation in the red is labeled *old*, and new observation in blue labeled *young*



Maximal margin classifier (math)

- Label blue observations as 1 and red observations -1
- Then separating line has the property that
  $\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} > 0$, if $y_i = 1$,
  and
  $\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} < 0$, if $y_i = -1$



Here, x_{i1} is heart rate for individual i, and x_{i2} is fitness level for individual i

Maximal margin classifier (math)

Maximize the margin, M, by finding the best values of β_0, β_1 and β_2 such that

$\beta_1^2 + \beta_2^2 = 1$

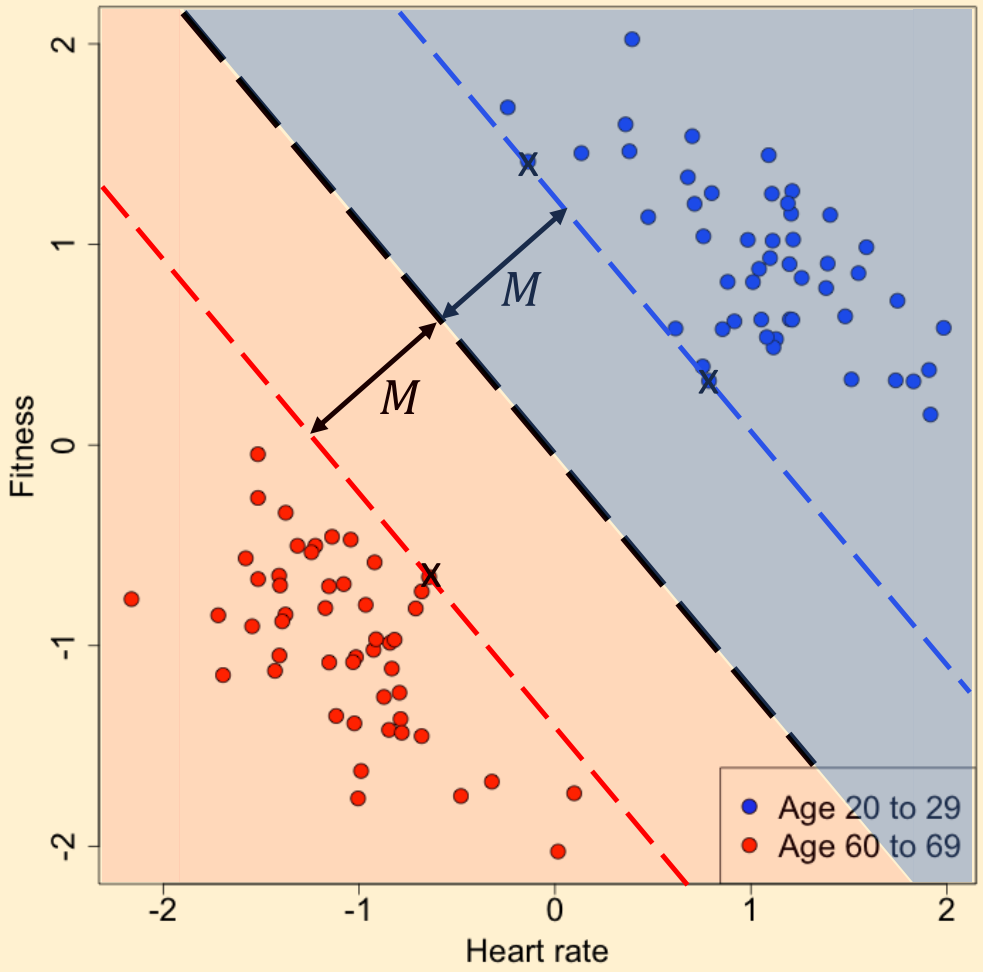
and

$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2}) \geq M$, for all subjects, i

So what is going on here?

If $\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2}$ is positive then $y_i = 1$ and (perpendicular) distance from the dividing line associated with $\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2}$ is larger than M

If $\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2}$ is negative, then $y_i = -1$ and the distance associated with $-(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2})$ is larger than M

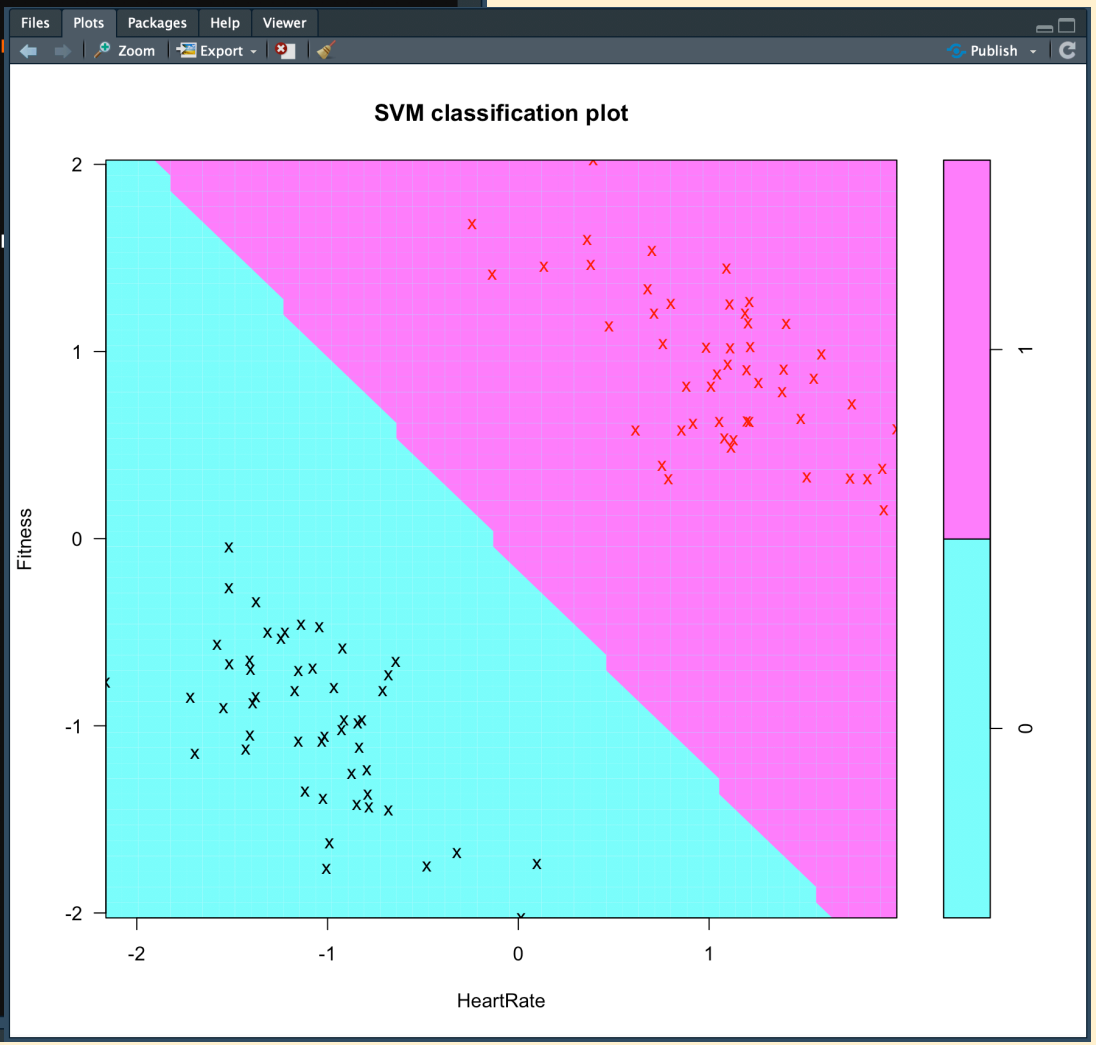


Running this in R

```
> library(e1071)  ### library containing svm function
>
> svmFit <- svm(ageGrp ~ Fitness + HeartRate, data=dat, kernel = "linear", cost = 0.00001,
scale = FALSE)
>
> summary(svmFit)

Call:
svm(formula = ageGrp ~ Fitness + HeartRate, data = dat, kernel = "linear",
    cost = 1e-05, scale = FALSE)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  1e-05
      gamma:  0.5

Number of Support Vectors:  100

 ( 50 50 )


Number of Classes:  2

Levels:
 0 1



> plot(svmFit,dat)
>
```

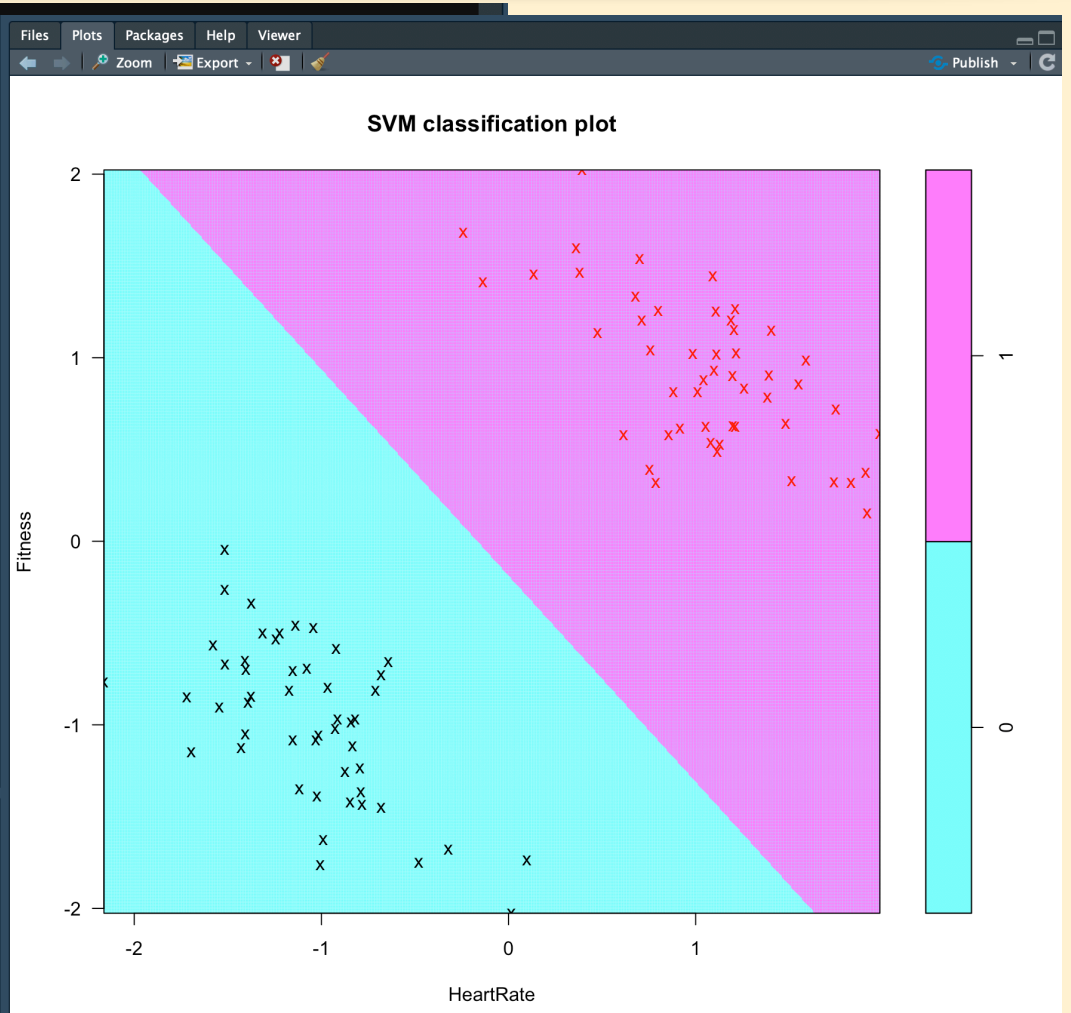
Running this in R

```
> library(e1071)  ### library containing svm function
>
> svmFit <- svm(ageGrp ~ Fitness + HeartRate, data=dat, kerı
scale = FALSE)
>
> summary(svmFit)

Call:
svm(formula = ageGrp ~ Fitness + HeartRate, data = dat, kerı
    cost = 1e-05, scale = FALSE)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  1e-05
      gamma:  0.5

Number of Support Vectors:  100

 ( 50 50 )


Number of Classes:  2

Levels:
 0 1



> plot(svmFit,dat)
>
```



Running this in R

```
> library(e1071)  ### library containing svm function
>
> svmFit <- svm(ageGrp ~ Fitness + HeartRate, data=dat, k
scale = FALSE)
>
> summary(svmFit)

Call:
svm(formula = ageGrp ~ Fitness + HeartRate, data = dat, k
    cost = 1e-05, scale = FALSE)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  1e-05
      gamma:  0.5

Number of Support Vectors:  100

 ( 50 50 )


Number of Classes:  2

Levels:
 0 1



> plot(svmFit,dat)
>
```

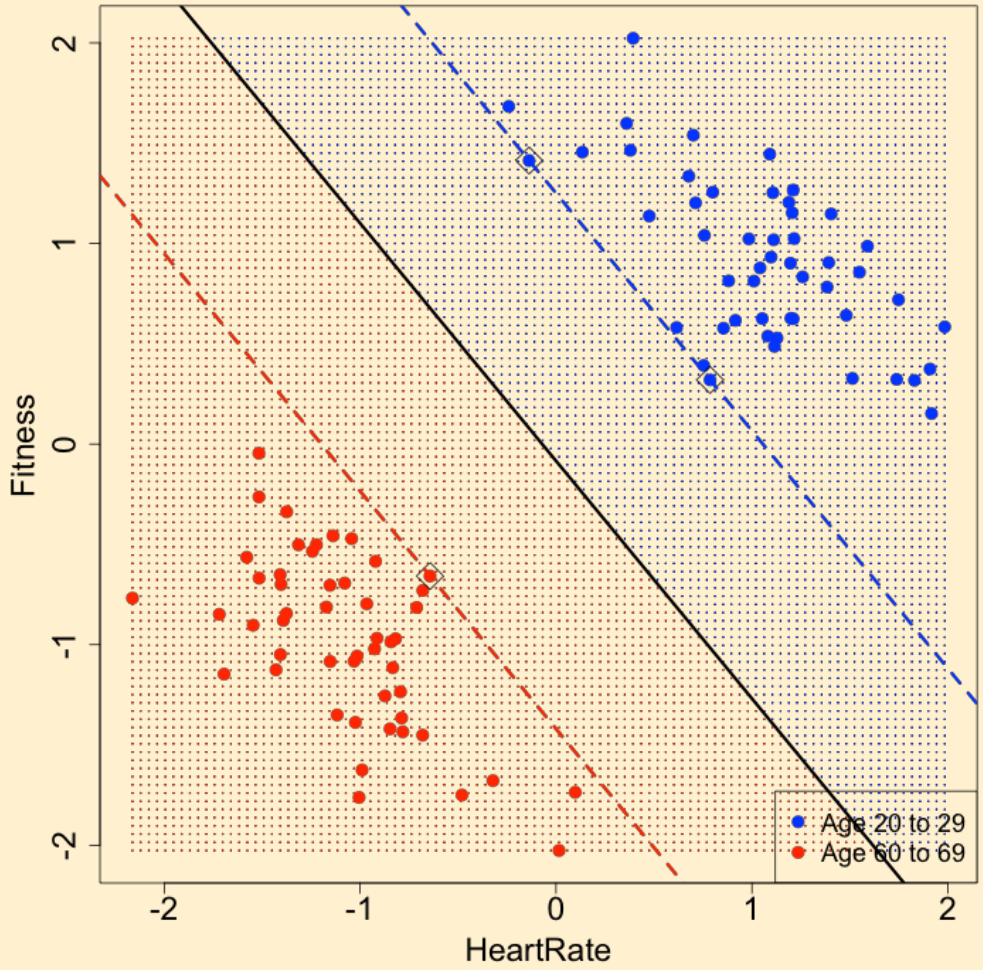



```
> plot(svmFit,grid=500,data = dat)
```

Running this in R

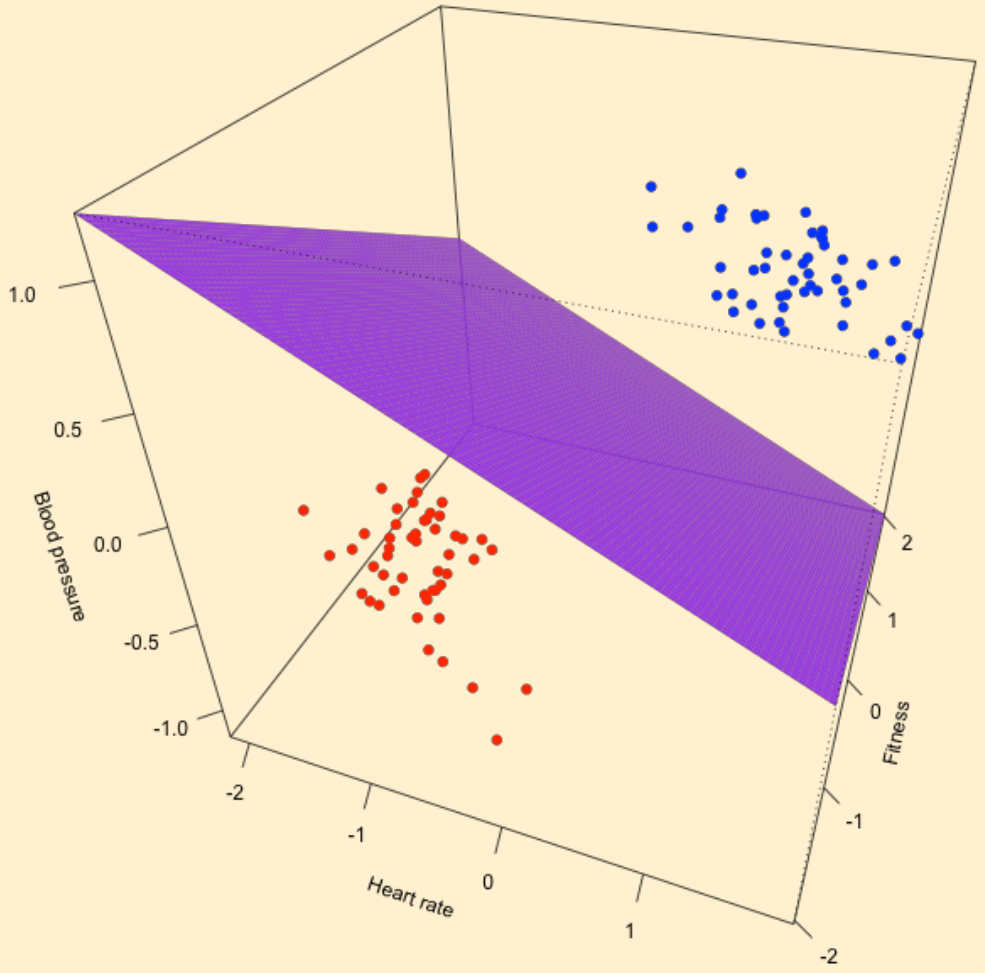
Or use my function `svmPlot`, which was hacked from ISLR materials

```
svmPlot(dat,svmFit)
```



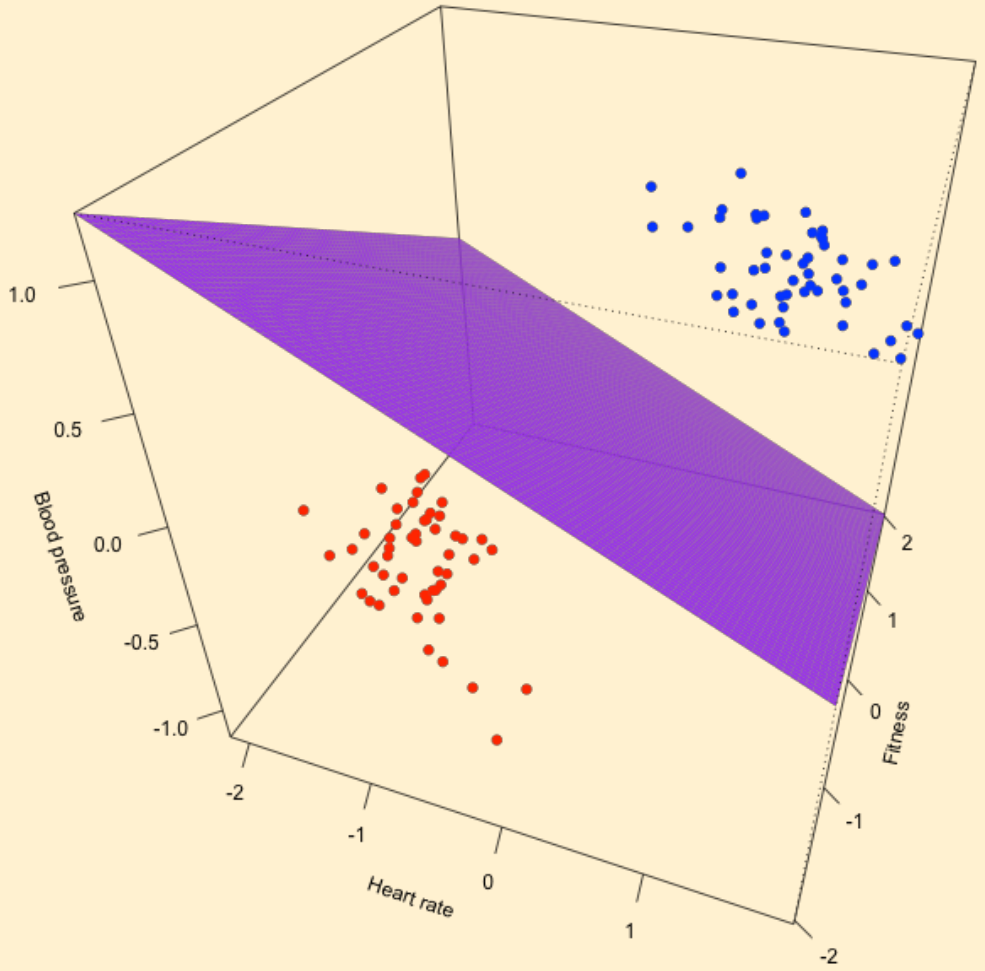
More than 2 predictors

- What about more than 2 predictors, e.g. 3 predictors of fitness, heart rate, and blood pressure?
- Now we need a separating 2D plane
- Separate 3D space with a 2D plane (previously we separated 2D space with a 1D line)



More than 2 predictors

- What about 4 predictors?
- Then we need a 3D *hyperplane* to separate the groups
- In general, for p predictors we need a $p-1$ dimensional hyperplane to separate two classes



More than 2 predictors

- In general, for p predictors we need a $p-1$ dimensional hyperplane to separate two classes
- Extend algorithm such that:

Maximize the margin, M, by finding the best values of β_0, $\beta_1, \ldots, \beta_p$ such that

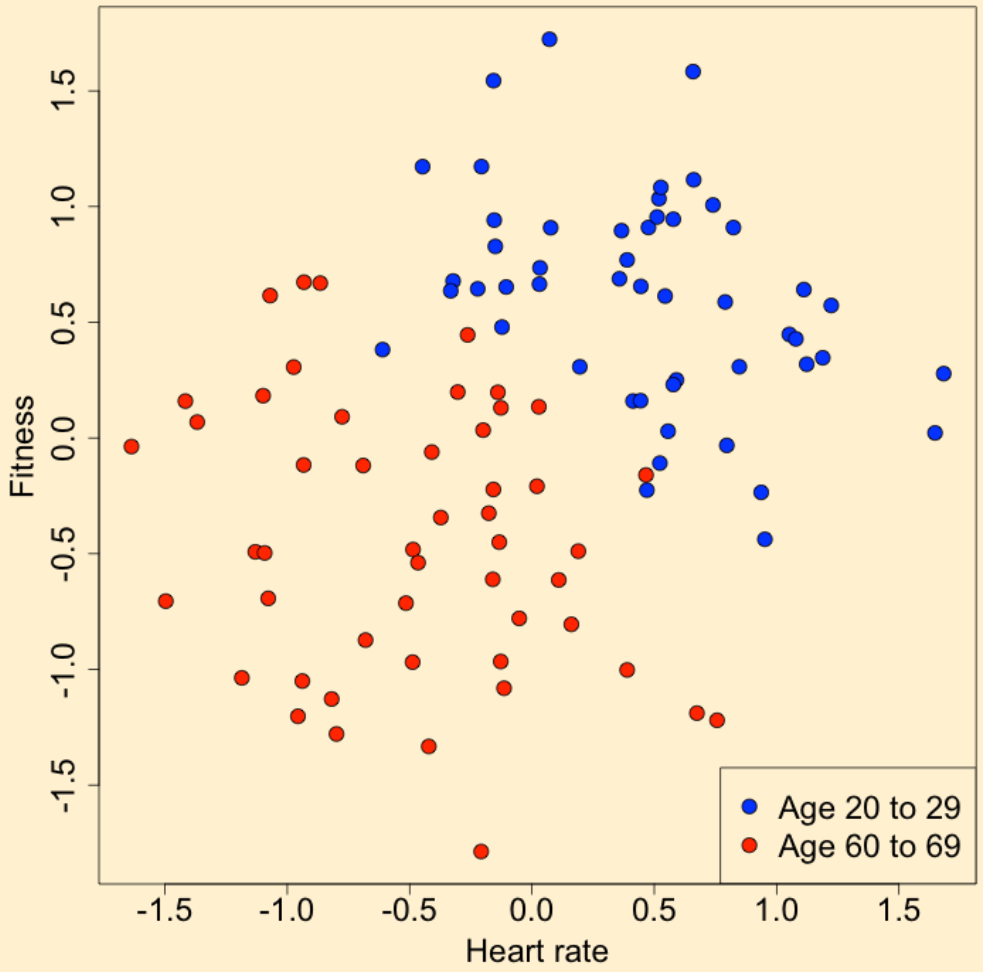
$$\beta_1^2 + \beta_2^2 + \cdots + \beta_p^2 = 1$$

and

$$y_i\left(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip}\right) \geq M, \text{ for all subjects, } i$$

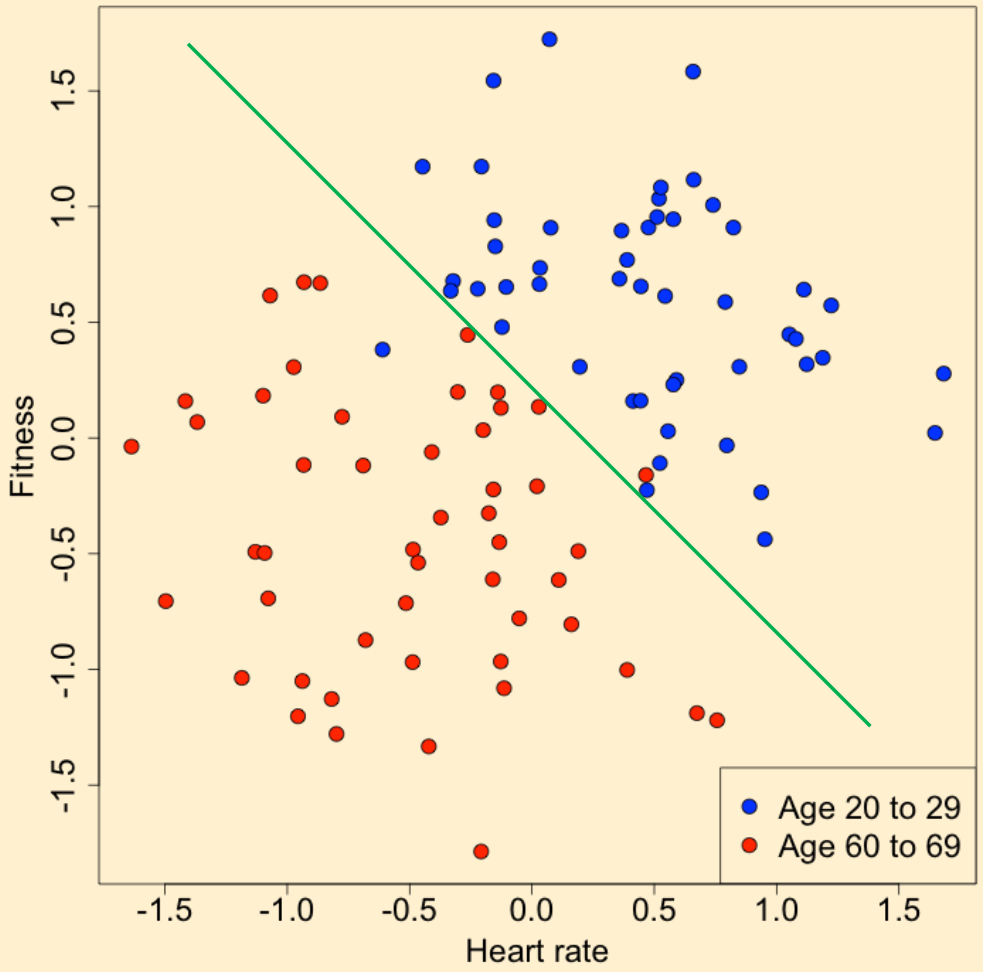
A more realistic dataset

- Simulated data
- Training data plot of fitness vs heart rate (standardized)
- Two groups: young vs. old
- Want a rule to classify future observations
- ***Now there is no clean line (and therefore no margin) to separate the groups***



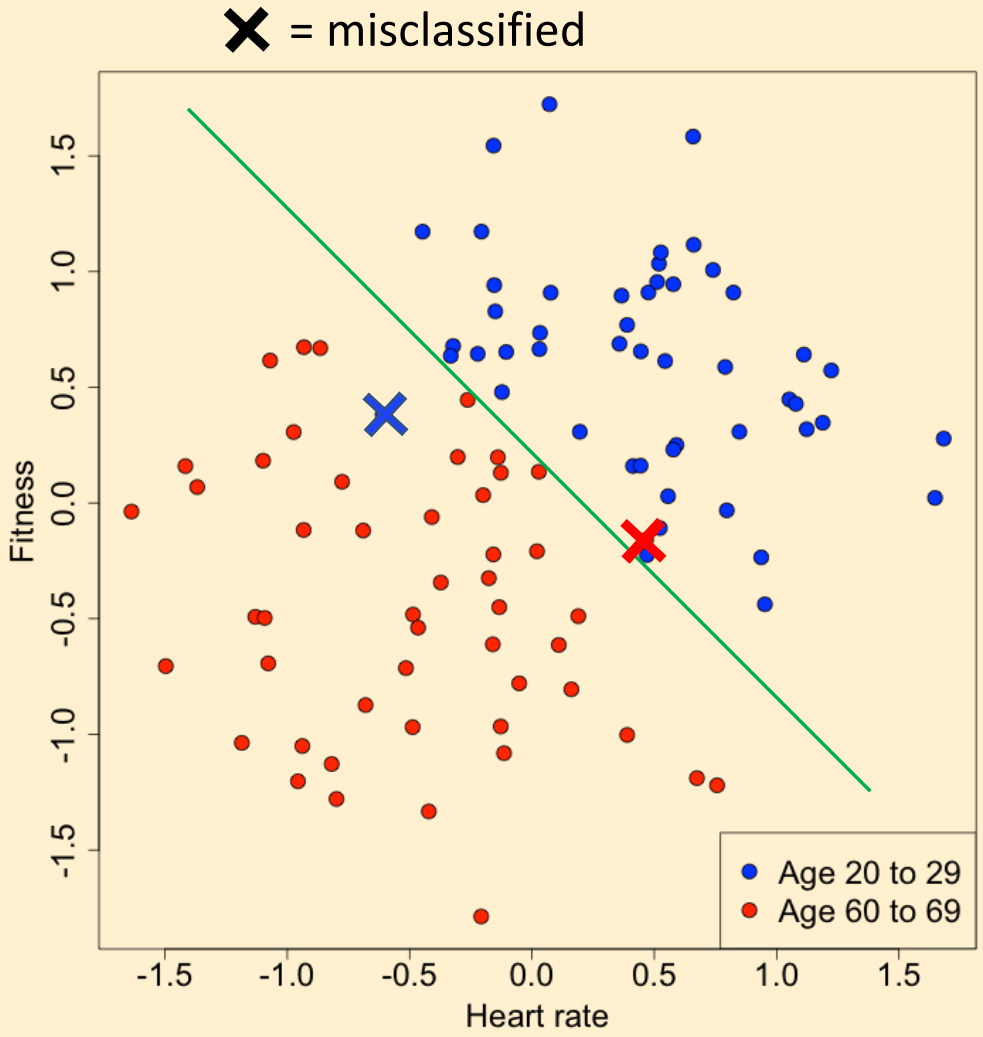
A more realistic dataset

- Simulated data
- Training data plot of fitness vs heart rate (standardized)
- Two groups: young vs. old
- Want a rule to classify future observations
- ***Now there is no clean line (and therefore no margin) to separate the groups***



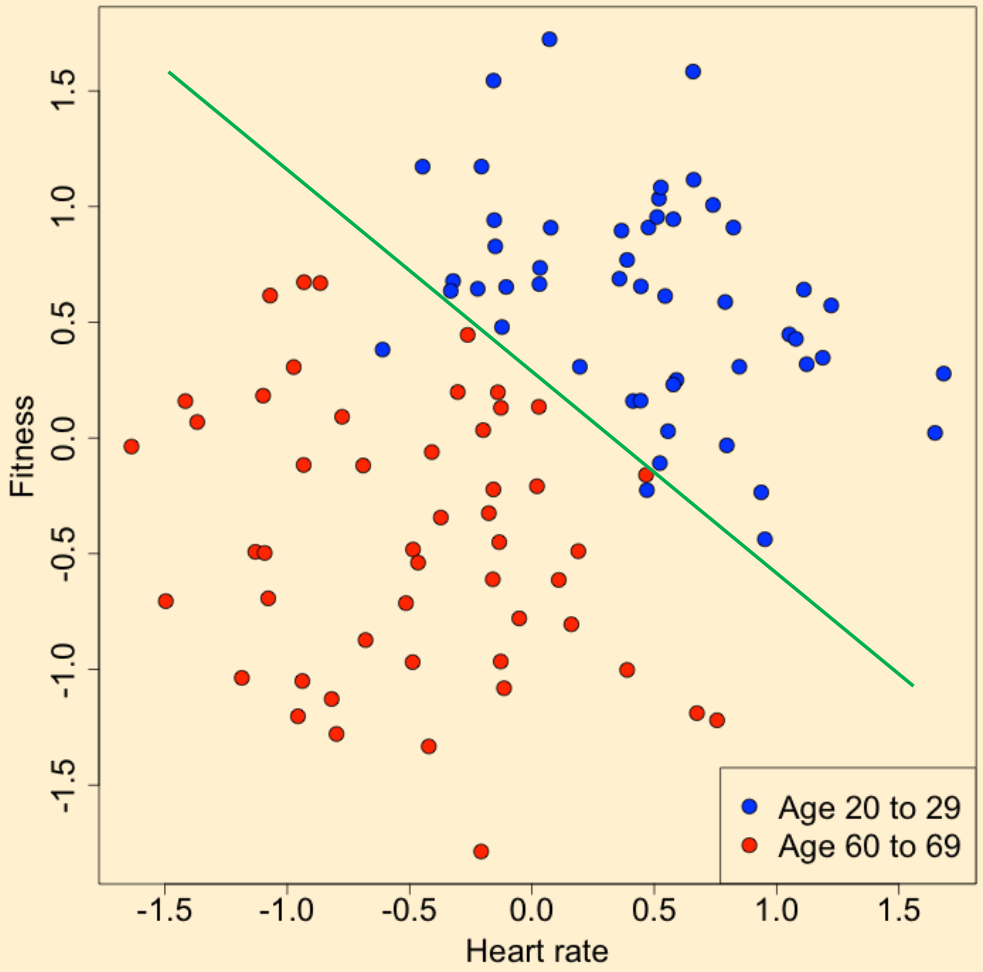
A more realistic dataset

- Simulated data
- Training data plot of fitness vs heart rate (standardized)
- Two groups: young vs. old
- Want a rule to classify future observations
- ***Now there is no clean line (and therefore no margin) to separate the groups***



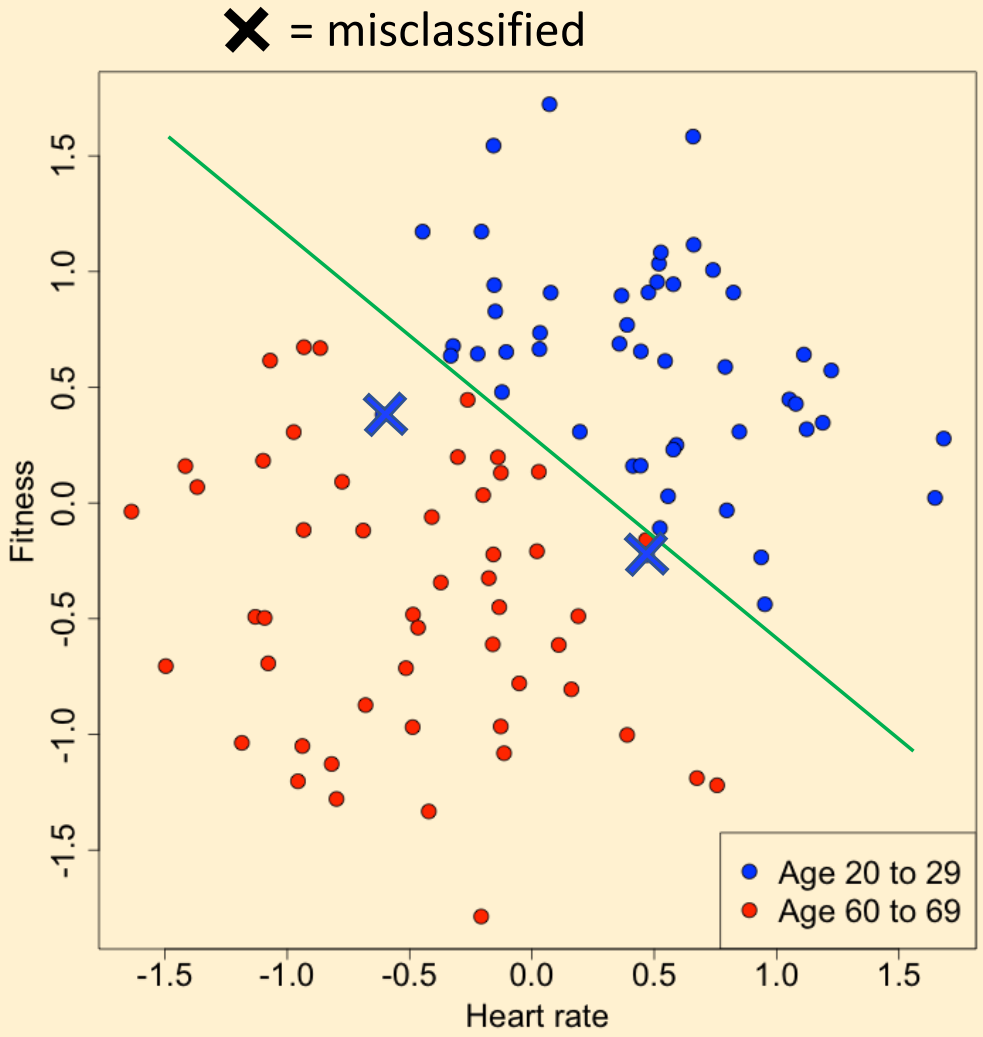
A more realistic dataset

- Simulated data
- Training data plot of fitness vs heart rate (standardized)
- Two groups: young vs. old
- Want a rule to classify future observations
- ***Now there is no clean line (and therefore no margin) to separate the groups***

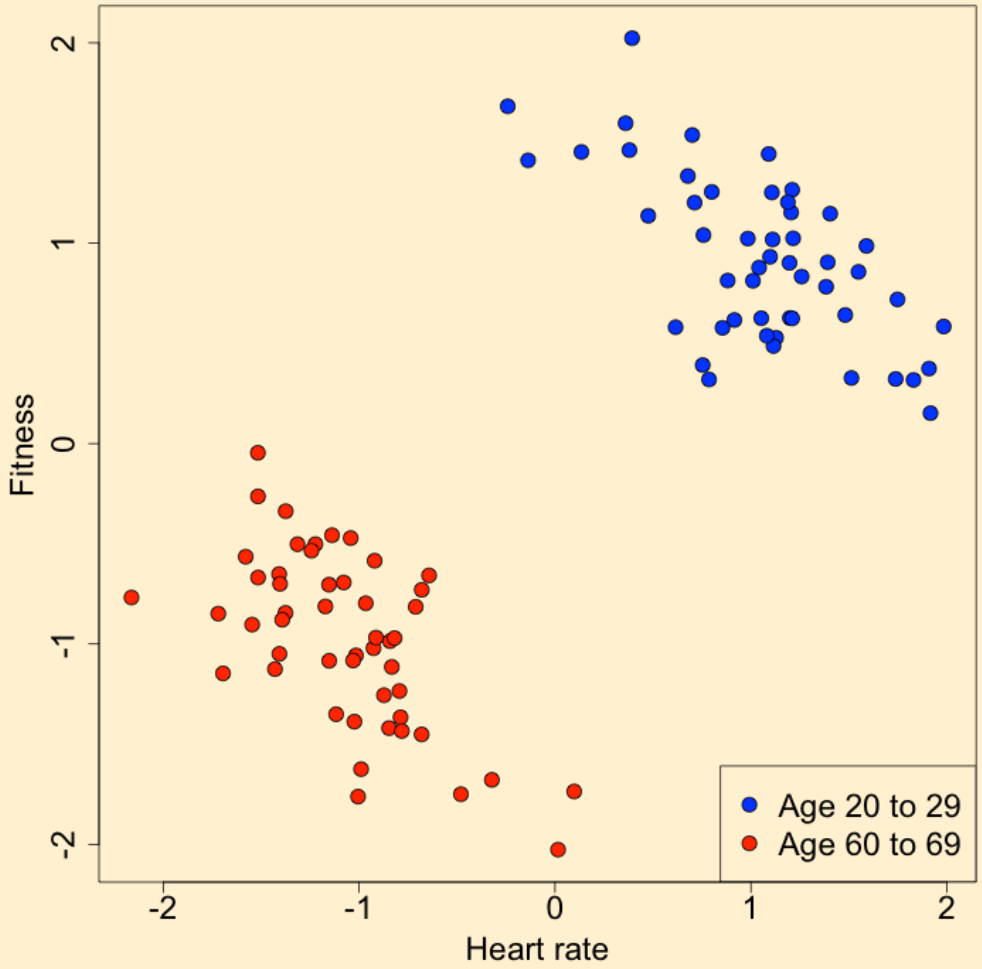


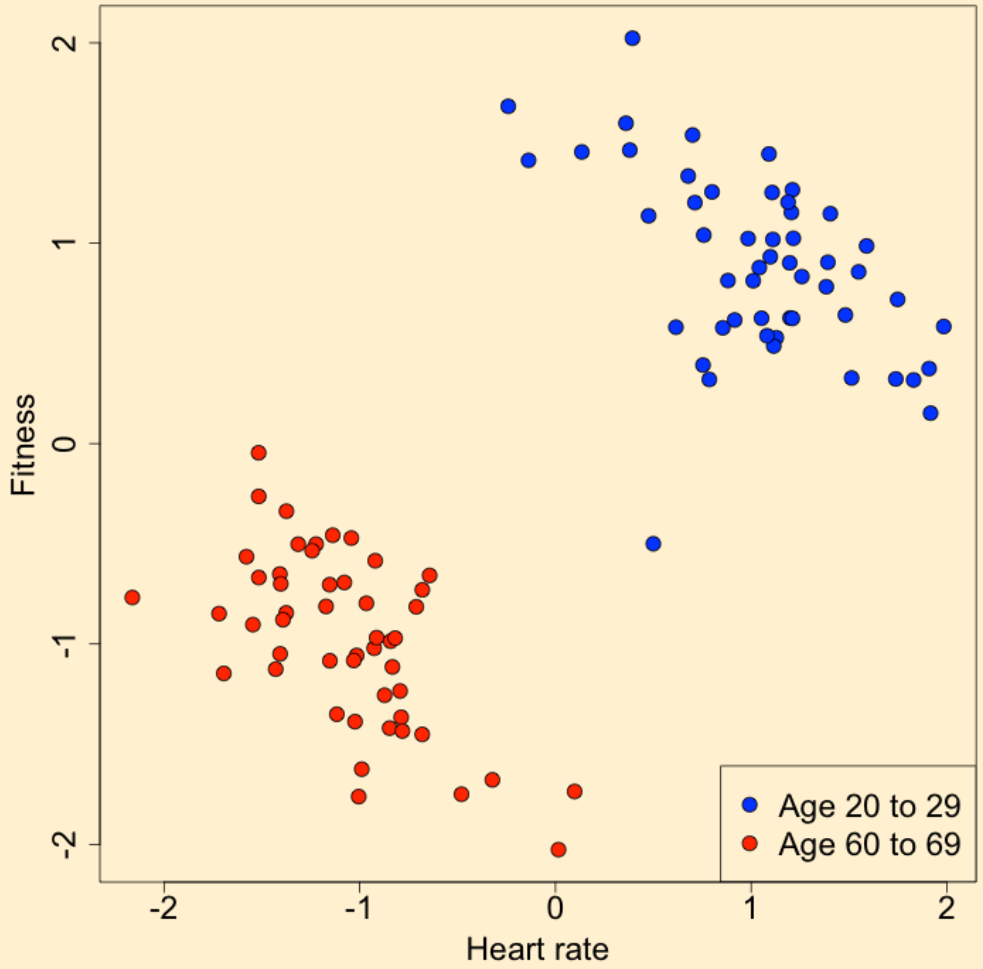
A more realistic dataset

- Simulated data
- Training data plot of fitness vs heart rate (standardized)
- Two groups: young vs. old
- Want a rule to classify future observations
- ***Now there is no clean line (and therefore no margin) to separate the groups***

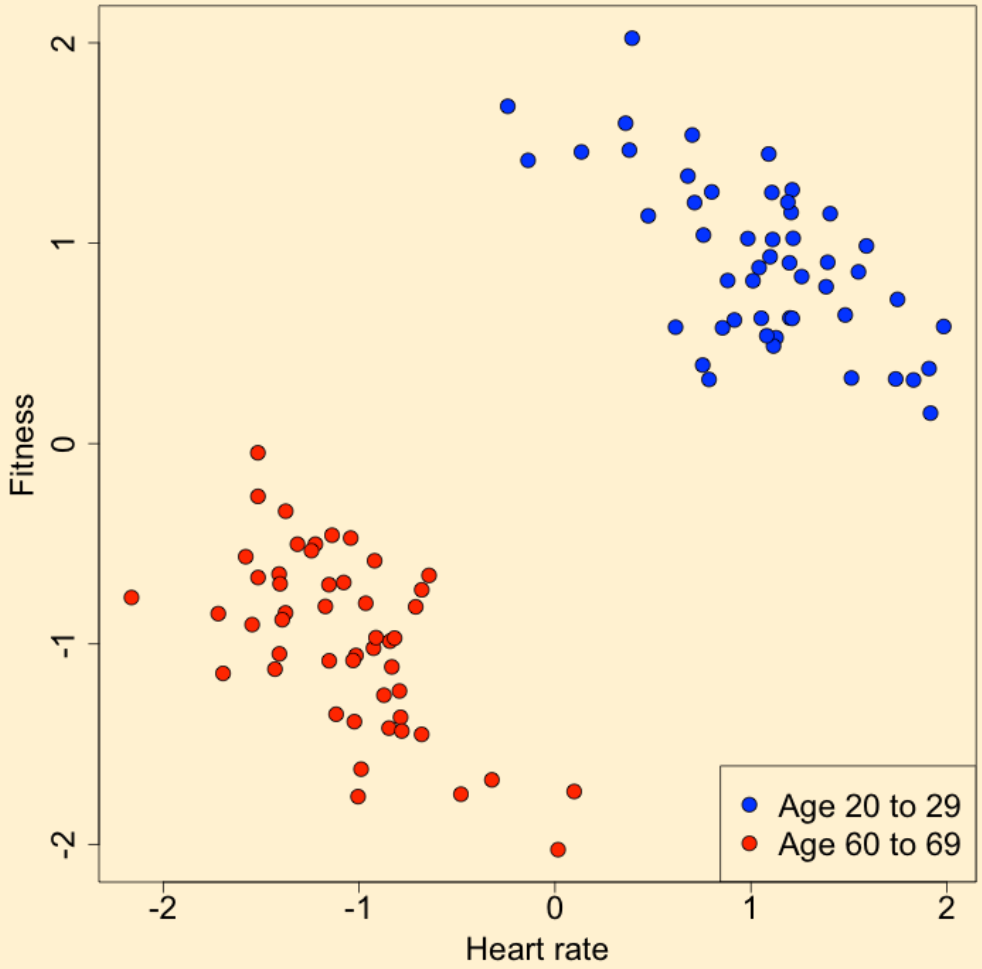


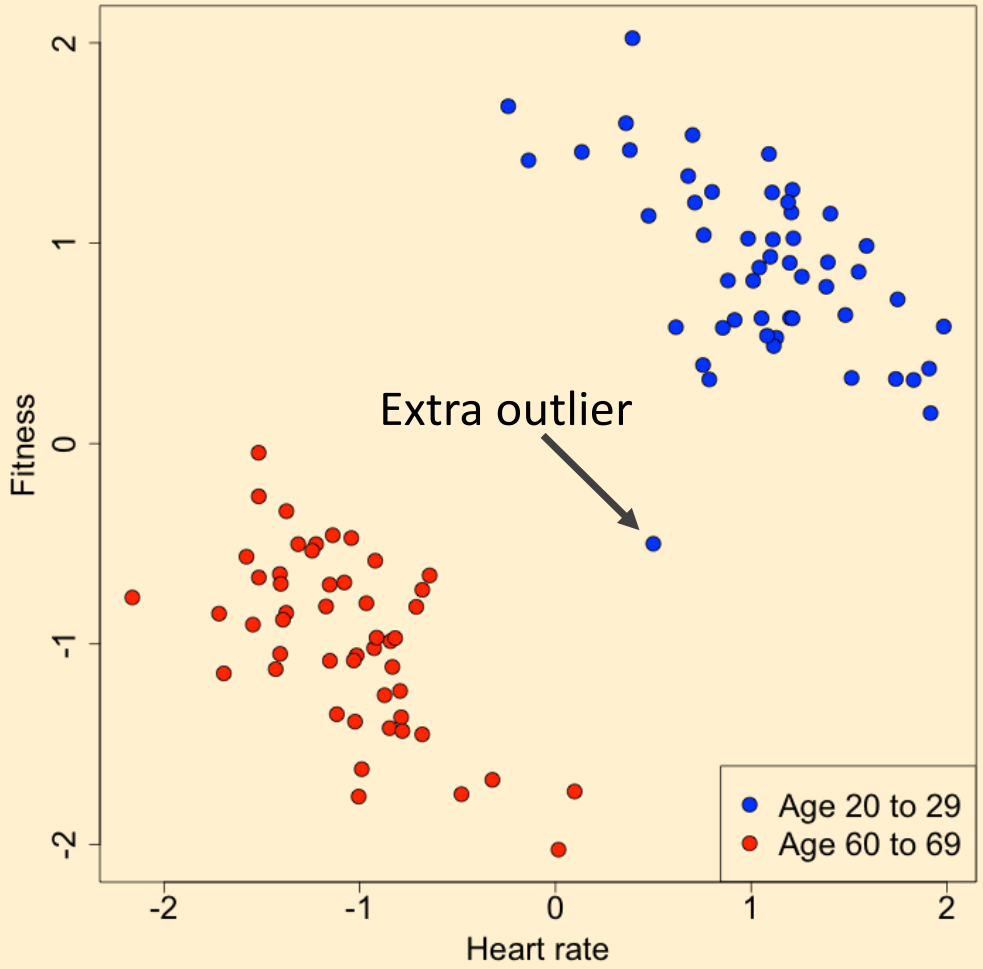
Even with a clean separation…



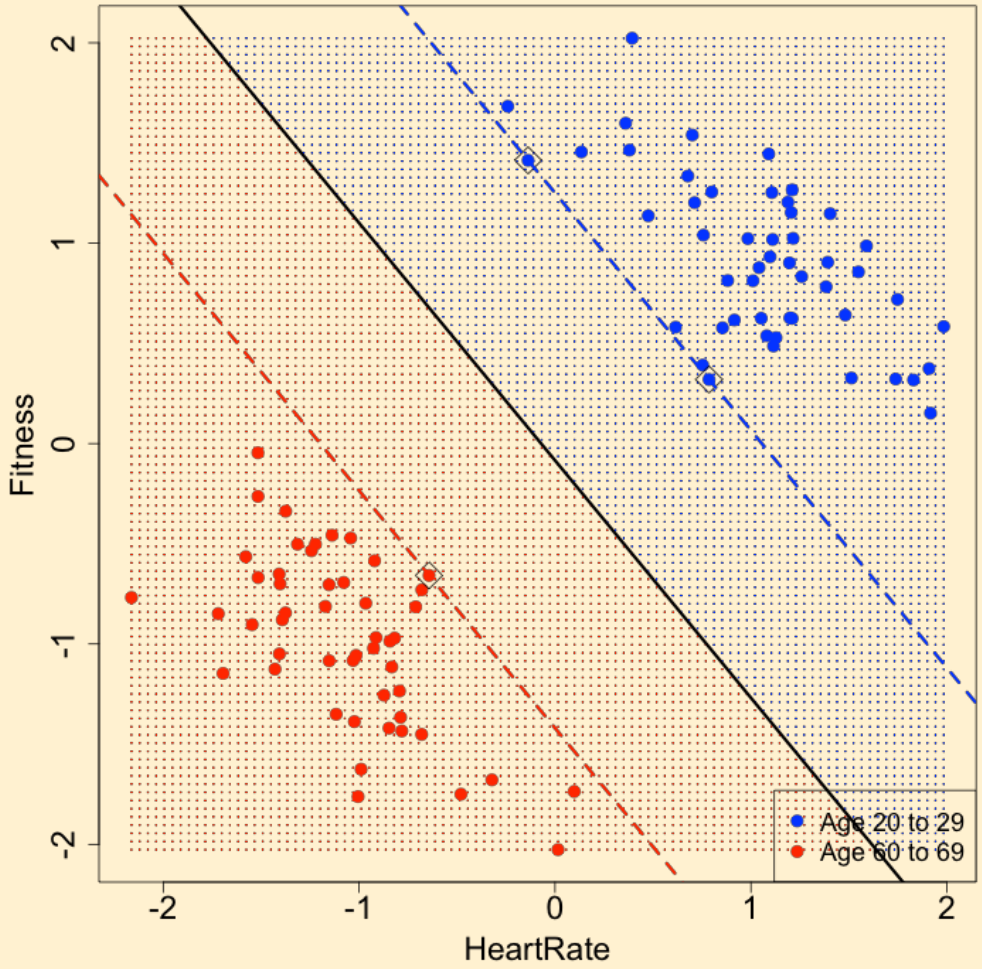


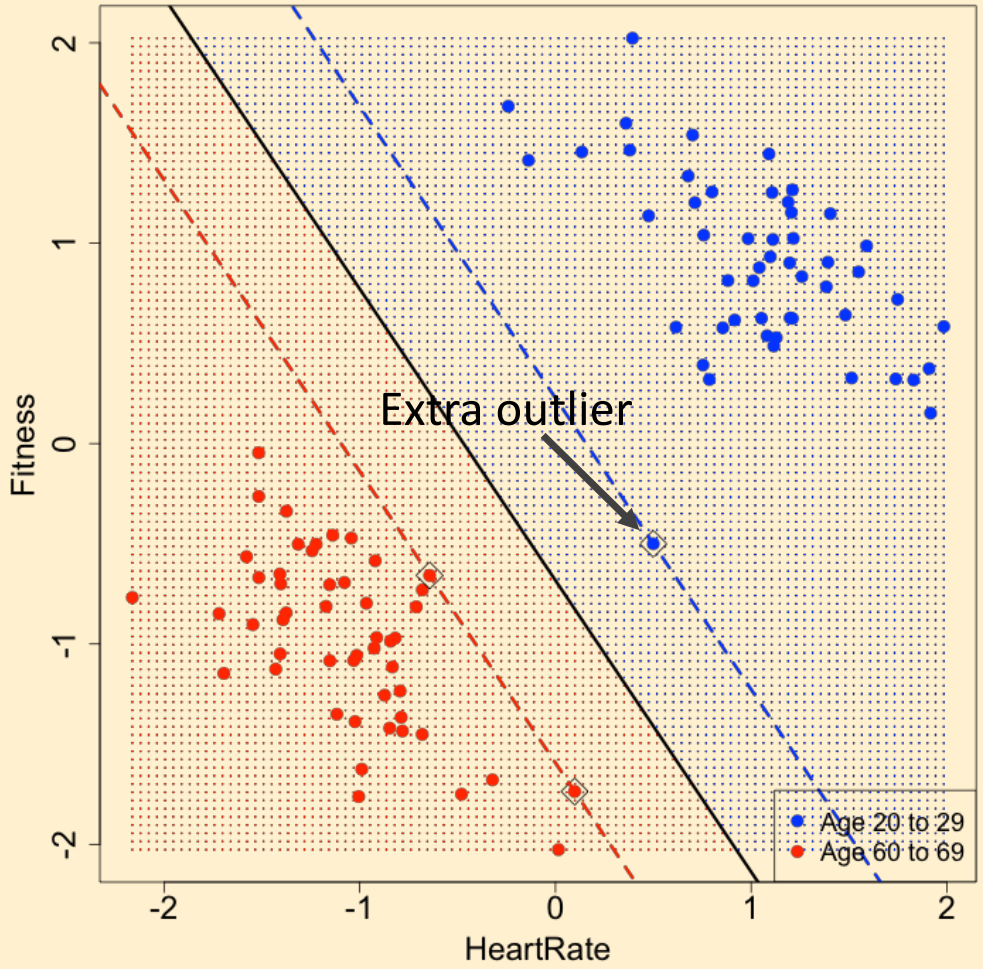
Even with a clean separation…





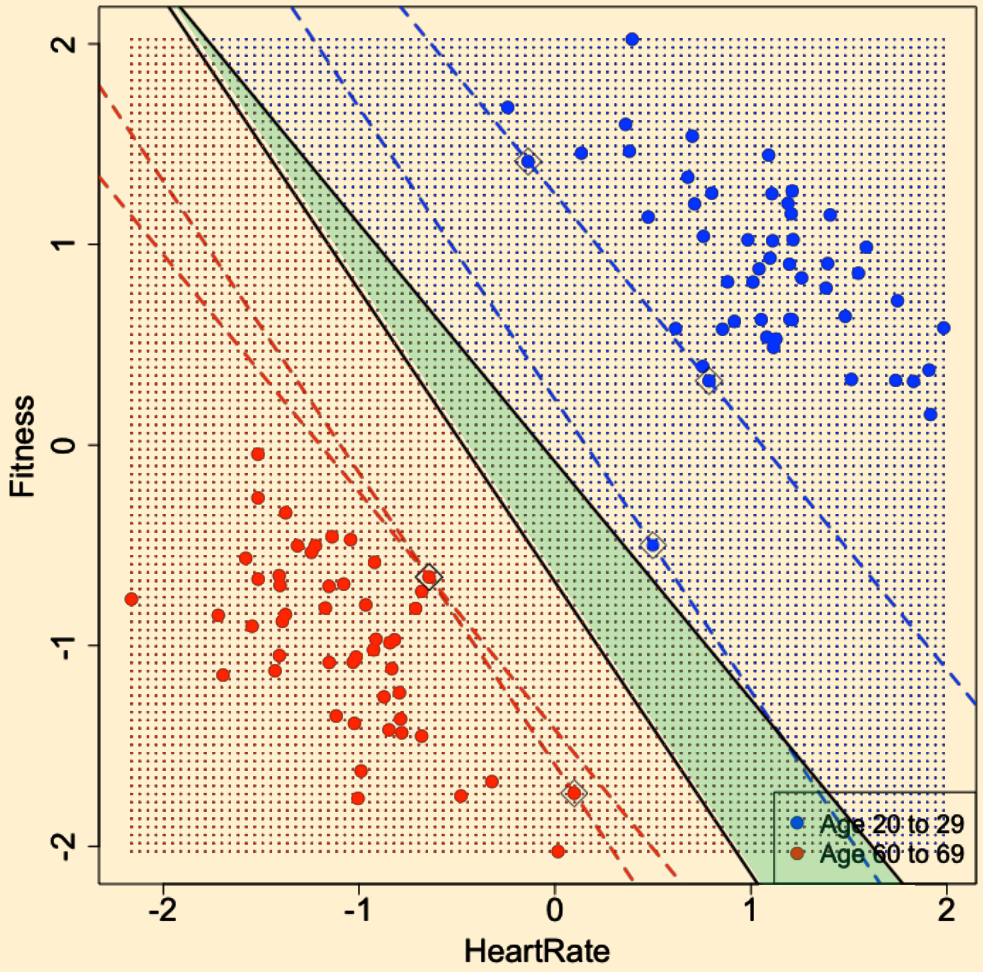
Even with a clean separation…





Even with a clean separation…

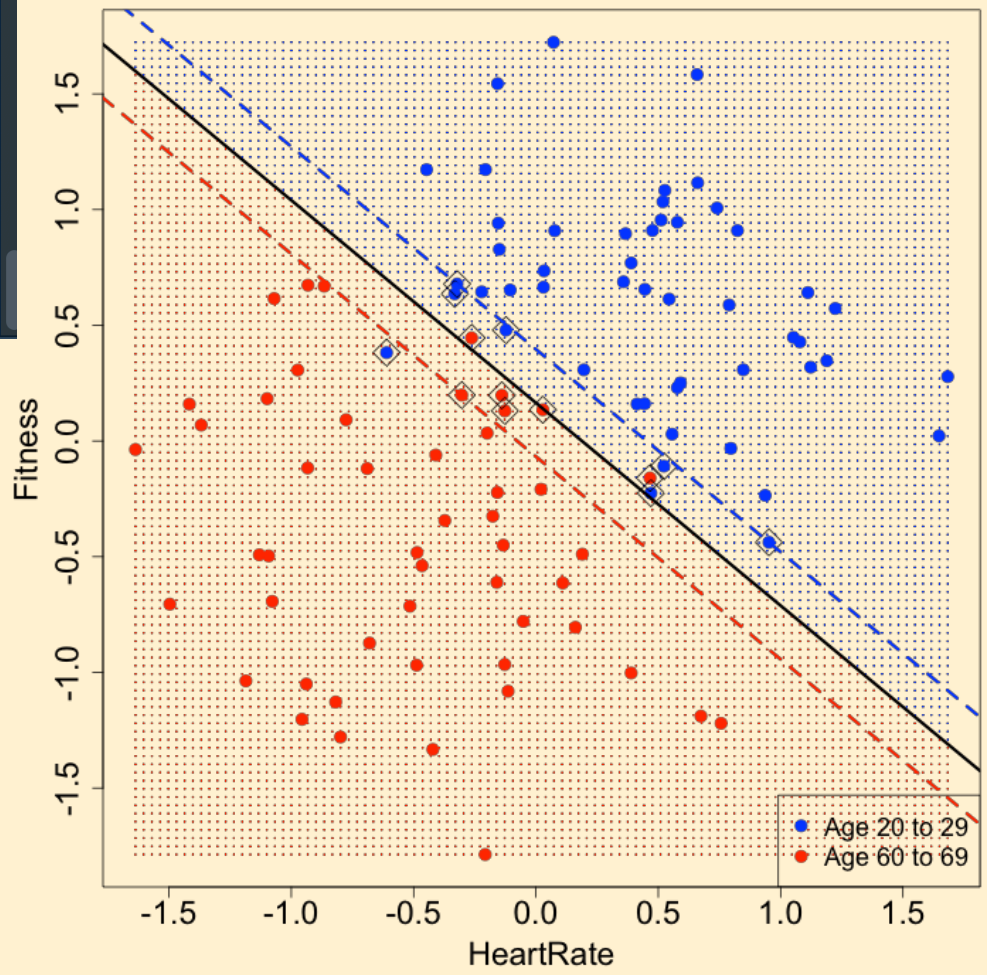
- Maximal margin classifier very sensitive to outliers
- Because driven by only 3 points



Support vector classifier

```
> svmFit <- svm(ageGrp ~ Fitness + HeartRate, data=dat
, kernel="linear", cost=10, scale=FALSE)
>
> svmPlot(dat, svmModel=svmFit, cex.axis=cex.axisval,
cex.lab=cex.labval)
>
> legend(x="bottomright", legend=c("Age 20 to 29","Age
60 to 69"), col=c(youngCol,oldCol), pch=19, cex=1.5)
>
```

- There are points on the wrong side of the decision boundary
- There is no “maximal margin”
- So how does this come up with a separating line?



Support vector classifier – soft margin

- The *support vector classifier* maximizes a *soft margin* as opposed to hard margin
- What does this mean?

Maximize the margin, M, by finding the best values of $\beta_0,\ \beta_1 \ldots, \beta_p$ such that

$$\beta_1^2 + \beta_2^2 + \cdots + \beta_p^2 = 1$$

and

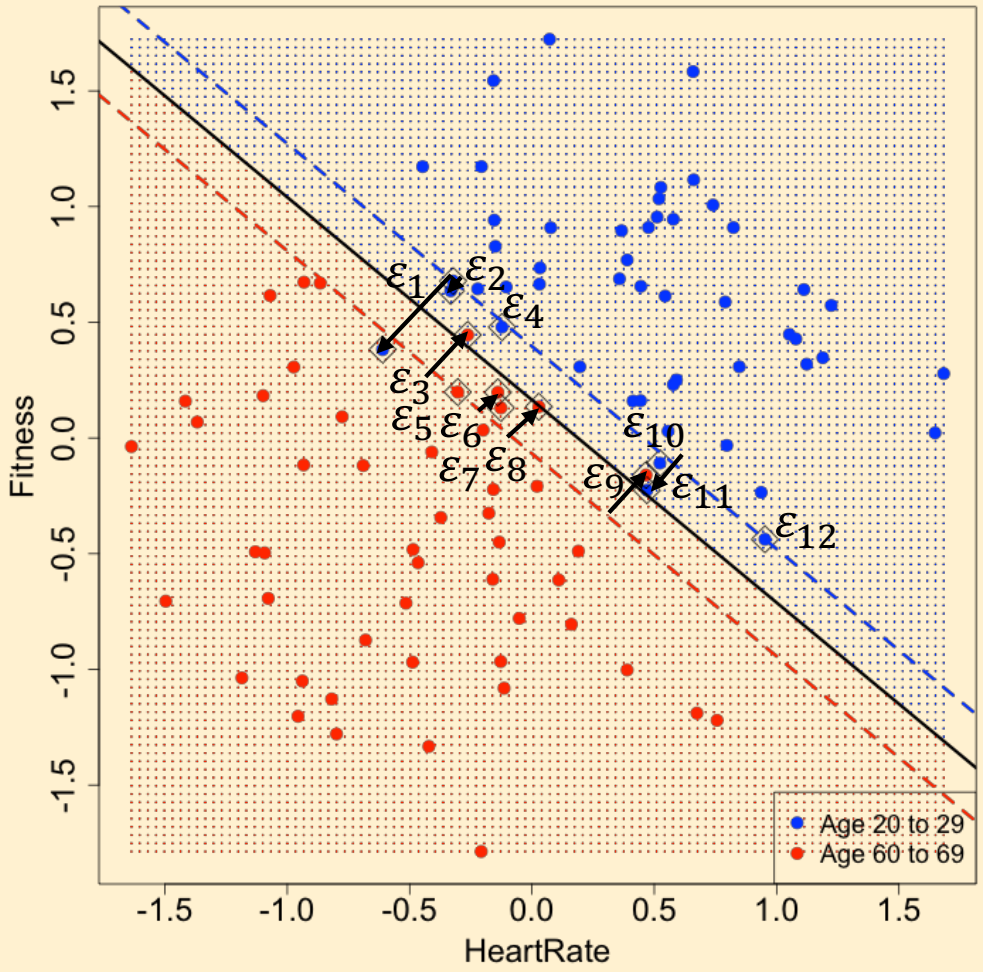
$$y_i\left(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip}\right) \geq M(1 - \varepsilon_i),$$

for all subjects, i

Where *slack variables,* $\varepsilon_i > 0$, and the sum is $\varepsilon_1 + \varepsilon_2 + \cdots + \varepsilon_n \leq C$

C is a tuning parameter (overall budget) that indicates how much decision boundary crossing is allowable: bigger = wider margin.

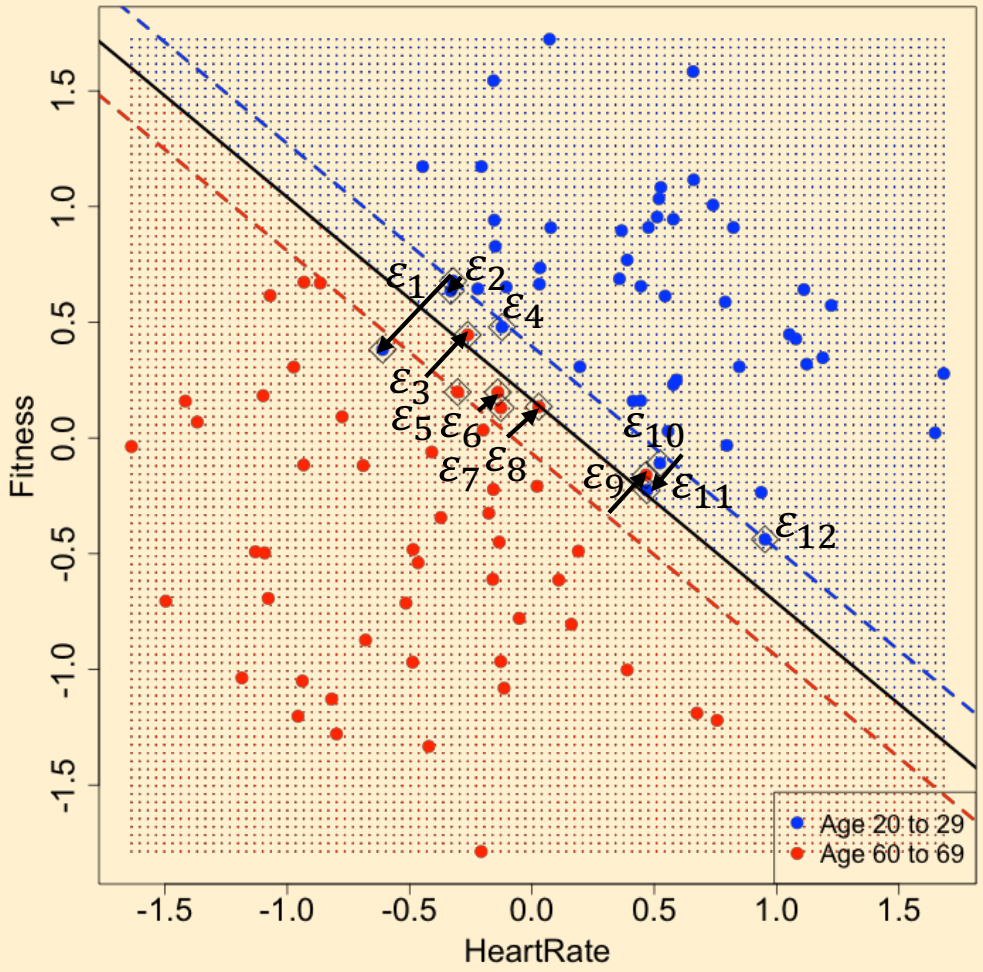
Cost $= 10\ \neq C$, more like $^1/_C$



Support vector classifier – soft margin

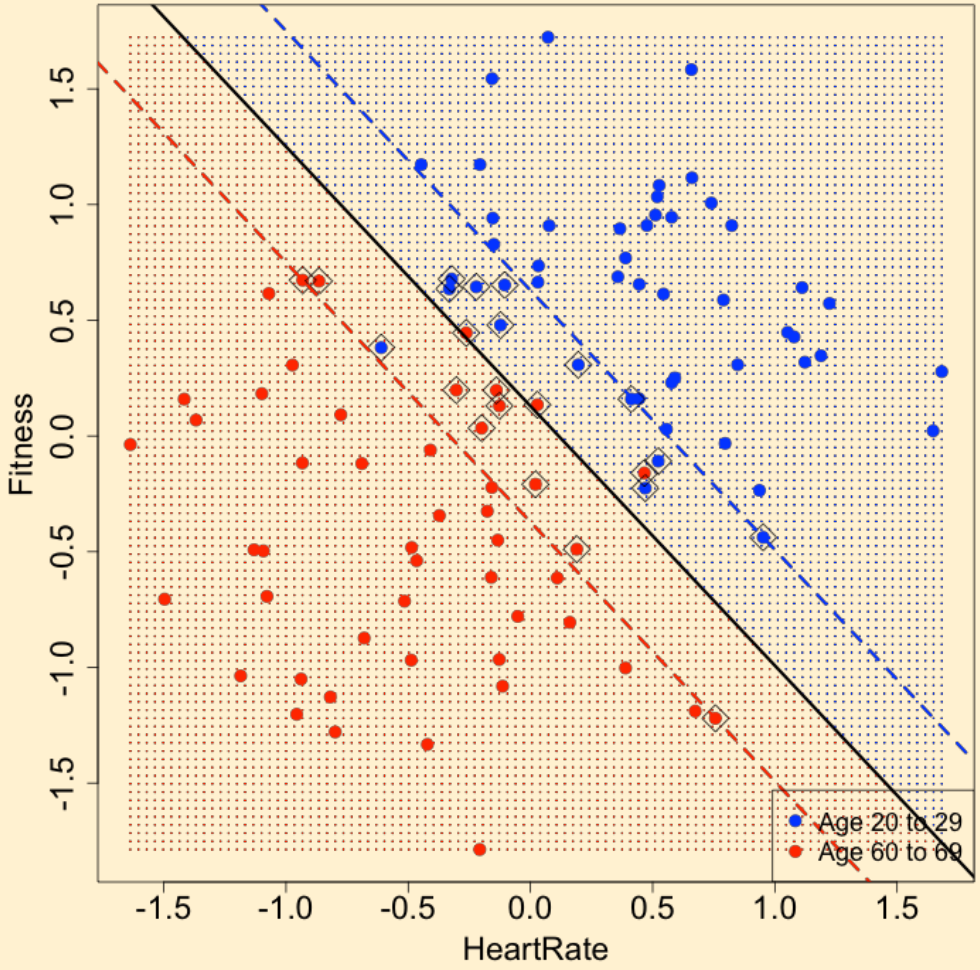
- The slack variable ε_i implies where observation located
- $\varepsilon_i = 0$ means located on correct side of boundary
- $\varepsilon_i > 0$ means located on wrong side of boundary (boundary violated)
- $\varepsilon_i > 1$ means located on wrong side of hyperplane
- C is a tuning parameter (overall budget) that indicates how much decision boundary crossing is allowable: bigger = wider margin.
- Total ε_i's need to be $\leq C$
- Note that in the `svm` function specify a cost (instead of budget): bigger cost = smaller margin, i.e. less crossings

Cost $= 10 \neq C$, more like $^1/_C$

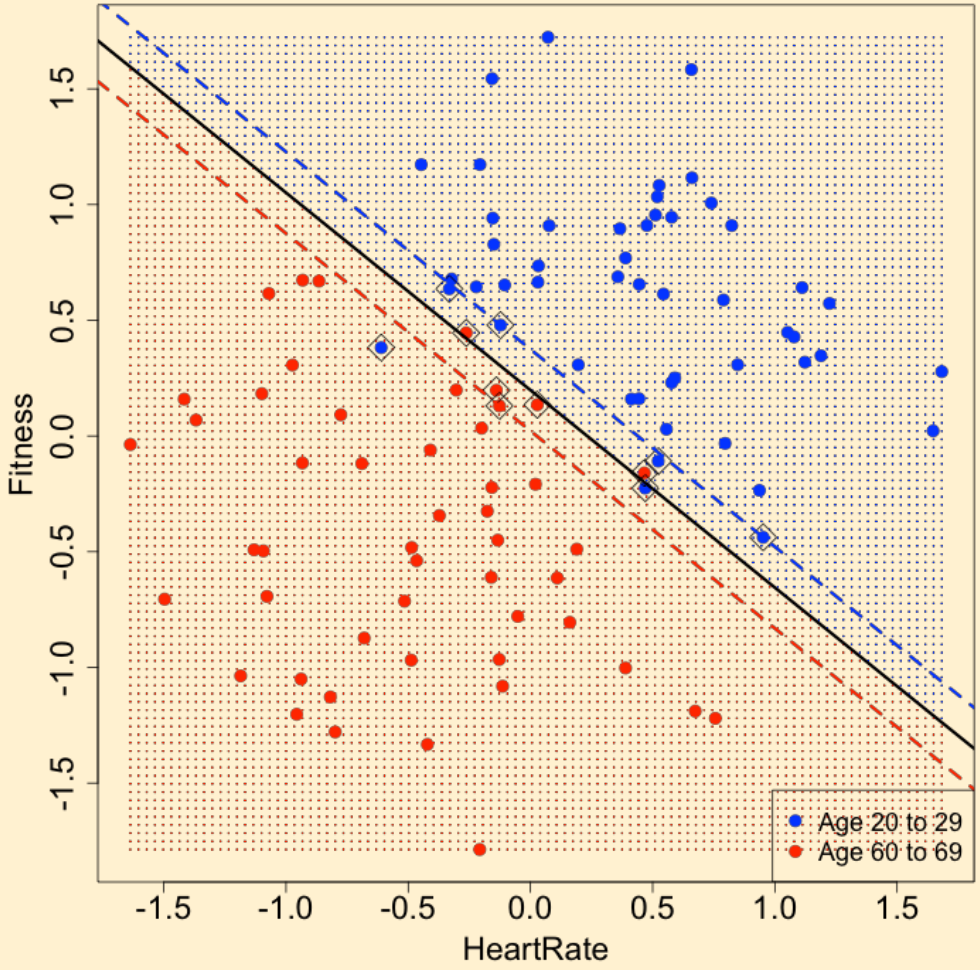


Different costs (and budgets, C)

Cost = 1 (budget C large)



Cost = 100 (budget C small)

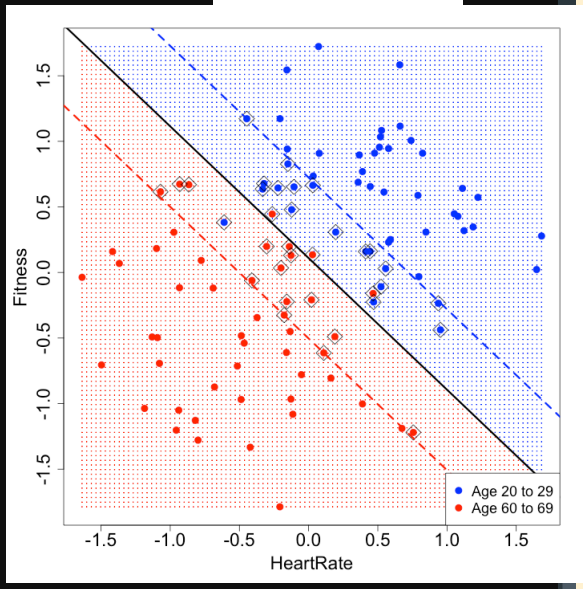


How to find the best cost?

- `e1071` includes function "`tune`" to perform 10-fold cross-validation
- In a larger dataset you might want to refine the search even further
- e.g. `cost <- c(0.3,0.5,0.8)` etc.

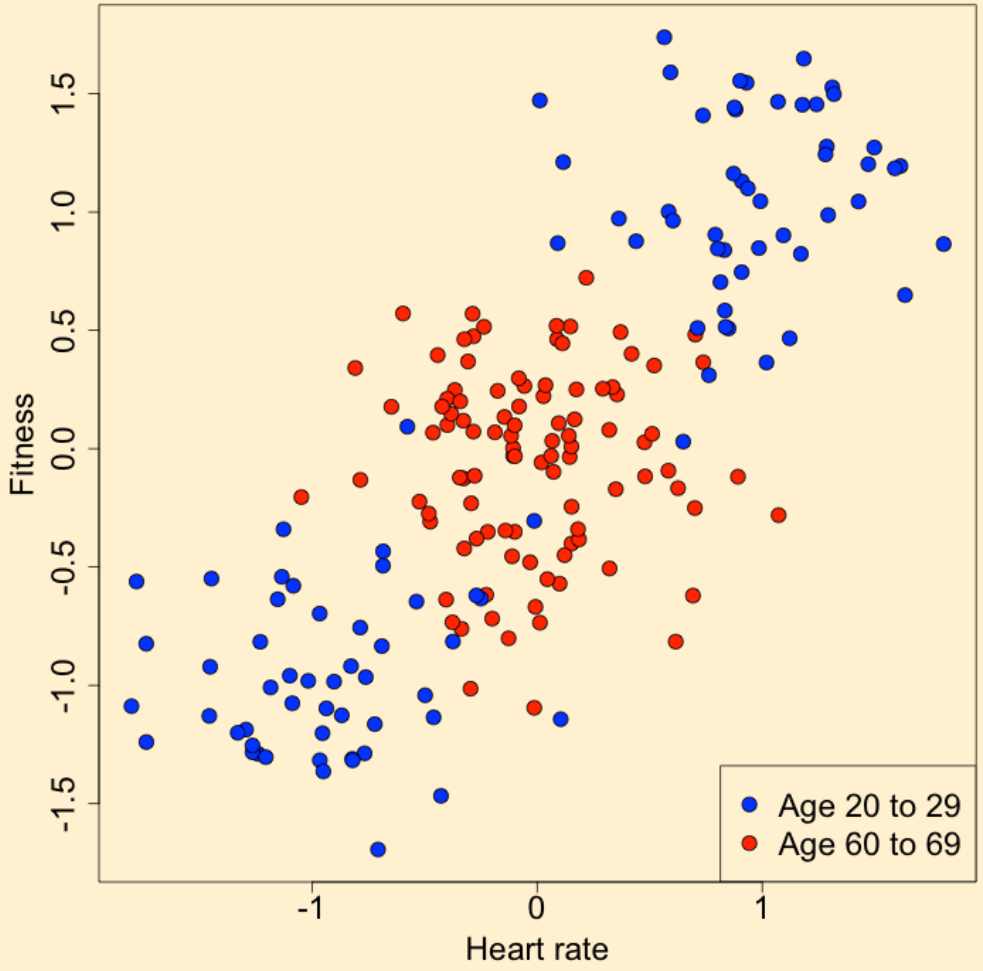
```
>
> ########## cross-validating cost ##################
>
> set.seed(4)
> tune.out <- tune(svm, ageGrp ~ Fitness + HeartRate, data=dat, kernel="lin
ear", scale=FALSE, ranges = list(cost=c(0.001,0.003,0.01,0.03,0.1,0.3,1,3,1
0,33,100)))
> summary(tune.out)

Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost
  0.3

- best performance: 0.04

- Detailed performance results:
      cost error dispersion
1    0.001  0.37 0.29832868
2    0.003  0.37 0.29832868
3    0.010  0.09 0.09944289
4    0.030  0.05 0.07071068
5    0.100  0.05 0.07071068
6    0.300  0.04 0.06992059
7    1.000  0.04 0.06992059
8    3.000  0.04 0.06992059
9   10.000  0.04 0.06992059
10  33.000  0.05 0.07071068
11 100.000  0.05 0.07071068

>
```

Cost = 0.3



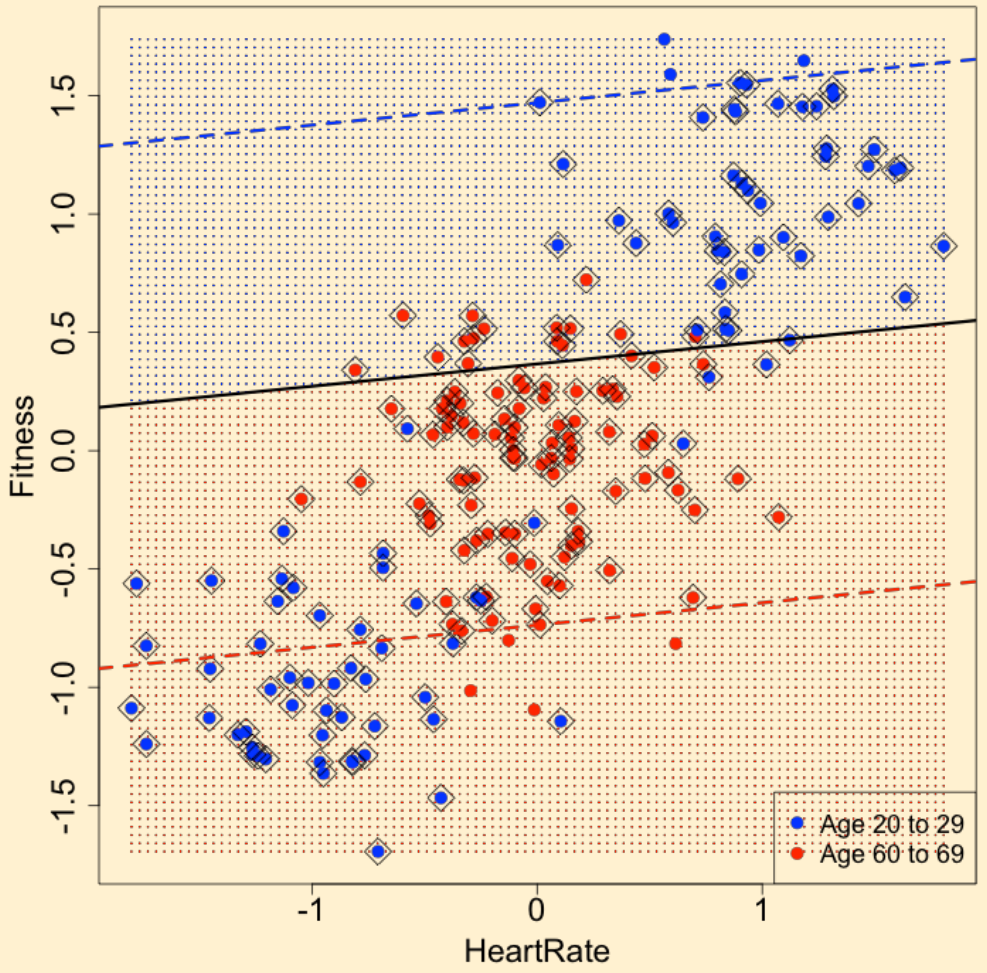
Nonlinear boundary

- Here is a pathological case for a linear decision boundary



Nonlinear boundary

- Here is a pathological case for a linear decision boundary
- Even a really soft margin support vector classifier won't help here (cost 0.1)
- Need nonlinear boundary



Try higher order terms: feature expansion

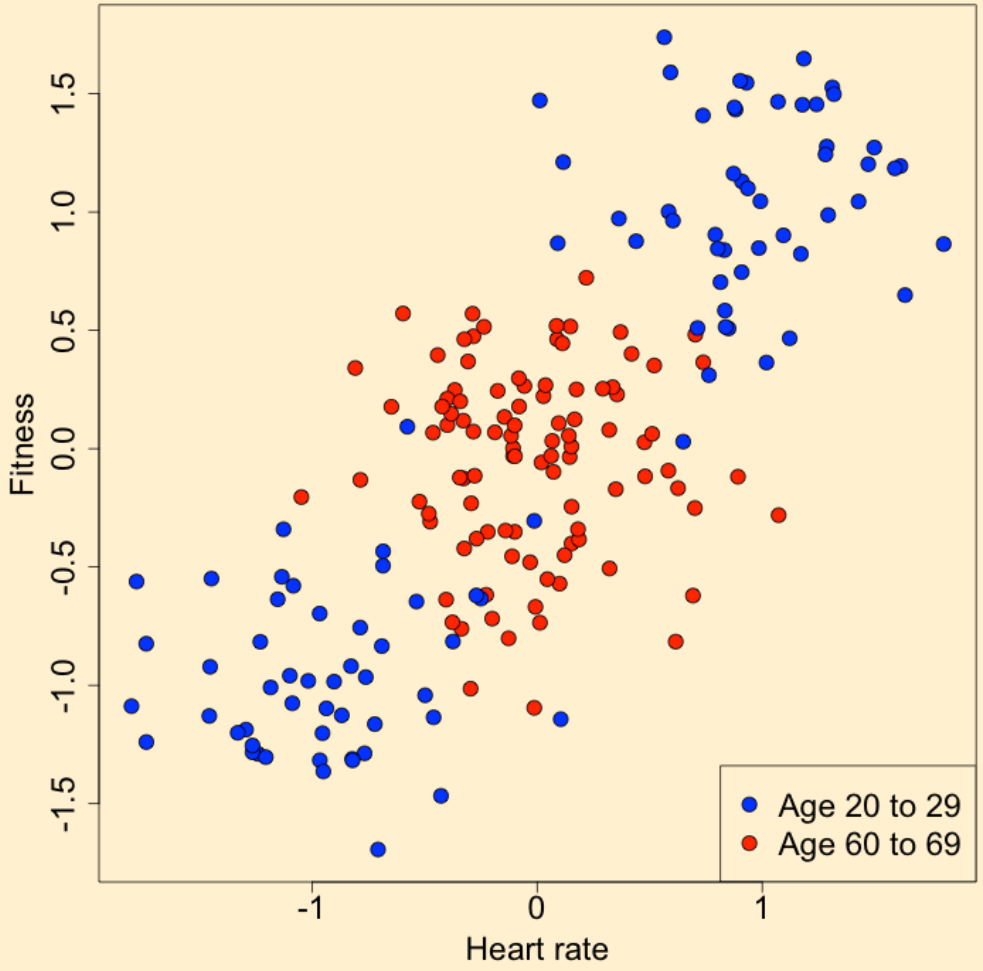
- Enlarge space of features to include transformed variables – e.g. $x_1^2, x_1 x_2, x_2^3, x_1^2 x_2, \ldots$
- Now fit a linear support vector classifier with the enlarged space of features (similar to how we add quadratic and interaction terms in linear or logistic regression)
- This results in a nonlinear boundary when transformed back to the space of x_1 and x_2
- Note that we are not restricted to polynomial terms; we could use pretty much any transformation, e.g. $\exp(x_1)$, $\sin(x_1^2 x_2)$, or even $x_2^{\exp(x_1^{2.5})}$
- The more transformed variables included, the more likely to fit the data well, but also more chance to overfit

Higher order feature expansion: example

- Suppose we use quadratic polynomial expansion $x_1, x_2, x_1^2, x_1 x_2, x_2^2$
- Then the decision boundary will take the form: $\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_1^2 + \beta_4 x_1 x_2 + \beta_5 x_2^2 = 0$
- This is linear in terms of the set $x_1, x_2, x_1^2, x_1 x_2$ and x_2^2
- But is nonlinear with respect to only x_1 and x_2

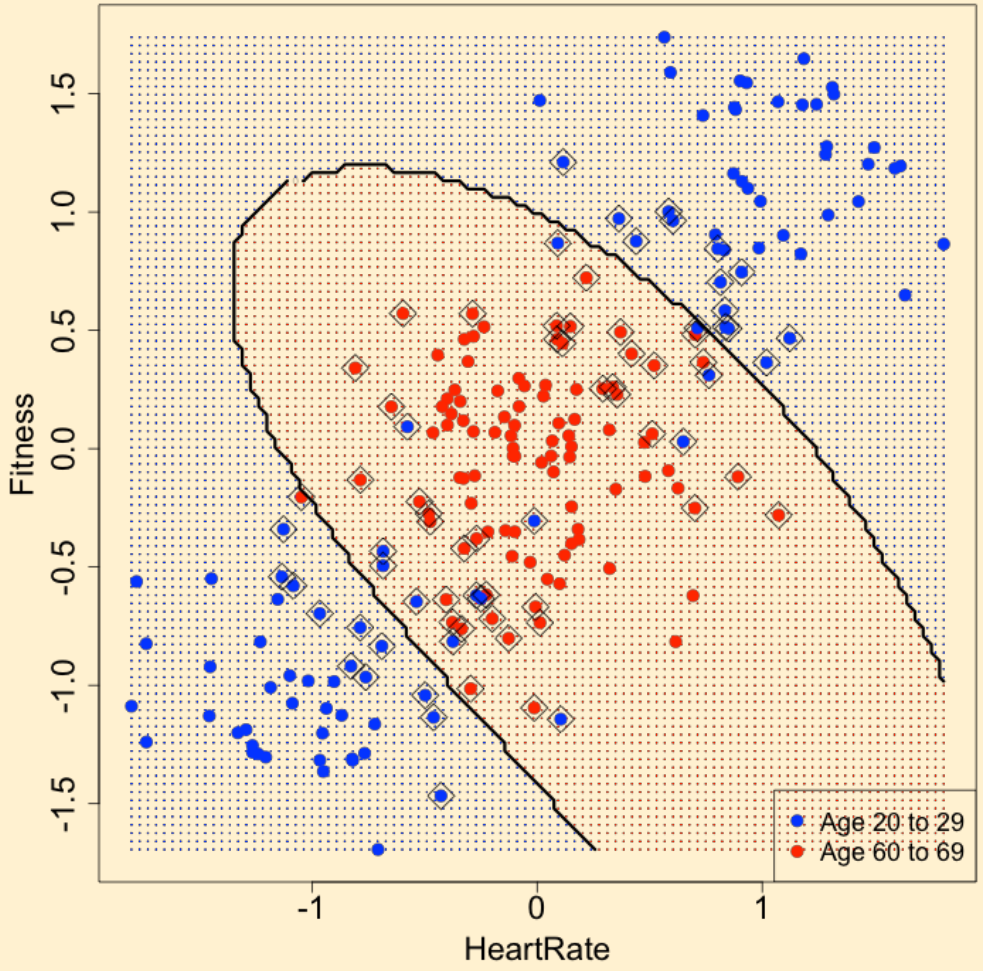
Nonlinear boundary

- Back to our pathological case
- Try a cubic expansion: $x_1, x_2, x_1^2, x_1x_2, x_2^2, x_1^3, x_1^2x_2, x_1x_2^2$, and x_2^3



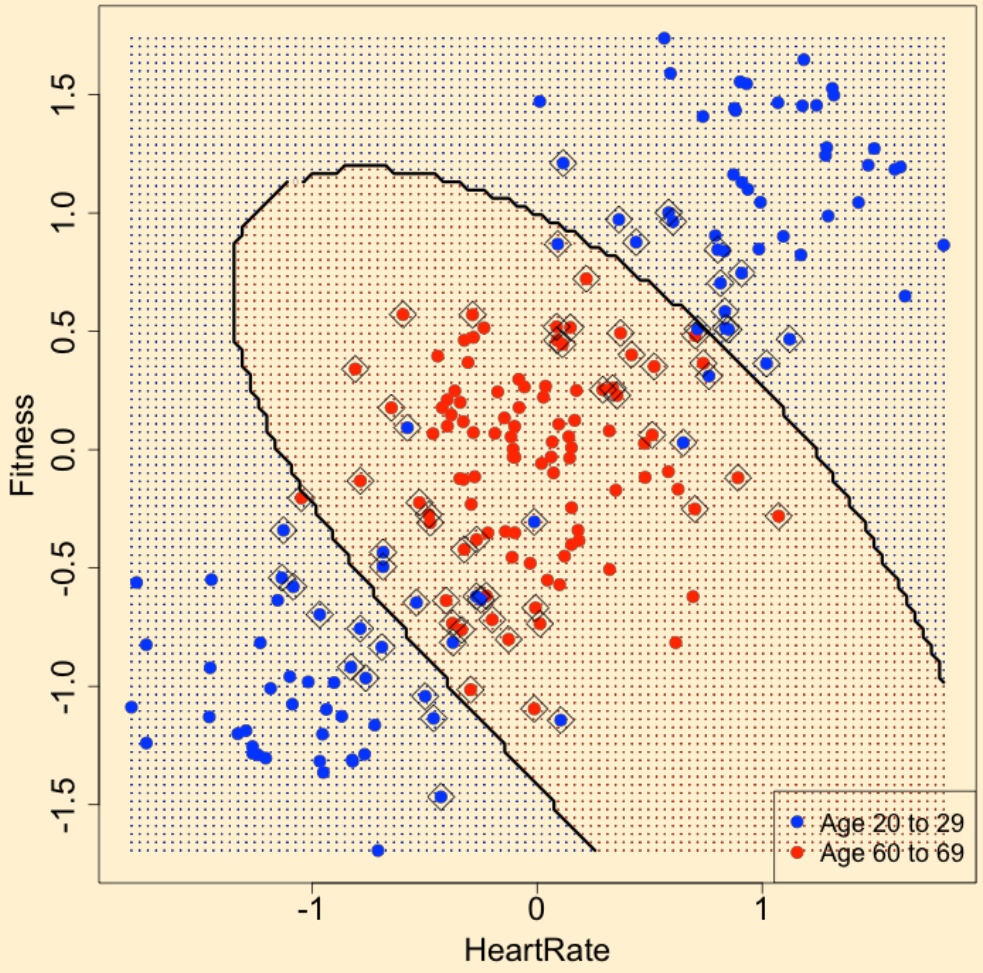
Nonlinear boundary

- Back to our pathological case
- Try a cubic expansion: $x_1, x_2, x_1^2, x_1x_2, x_2^2, x_1^3, x_1^2x_2, x_1x_2^2$, and x_2^3
- Cost = 0.1



Nonlinear boundary

- Back to our pathological case
- Try a cubic expansion: $x_1, x_2, x_1^2, x_1x_2, x_2^2, x_1^3, x_1^2x_2, x_1x_2^2$, and x_2^3
- Cost = 0.1
- But this is a lot of work – especially if try multiple large expansions




```
svmFitEnrich <- svm(ageGrp ~ Fitness + HeartRate + HR2 + HRFT + FT2 + HR3 + HR2FT + HRFT2 +
FT3, data=datEnrich, kernel="linear", cost=0.1, scale=FALSE)
```

Kernel trick!

- Polynomial number of terms gets large really fast if we increase the degree allowable (i.e. increase flexibility)
- So we instead use a *kernel approach (trick)* to deal with nonlinearity
- Will try to give a flavor as to how the kernel trick works but needs a lot of complex math to fully cover so will skate over details
- First step we need to know of the *inner product*

Inner product

- Consider x_i is a set of predictors for an observational unit i
- p is the number of predictors (or features)
- Then the *inner product* between two observations x_i and x_j is: $\langle x_i, x_j \rangle = x_{i1}x_{j1} + x_{i2}x_{j2} + \cdots + x_{ip}x_{jp}$
- The key is then that the linear support vector classifier can be written as:
  $f(x) = \beta_0 + \alpha_1\langle x, x_1 \rangle + \alpha_2\langle x, x_2 \rangle + \cdots + \alpha_n\langle x, x_n \rangle$
- So there are n parameters (the number of observations): α_1 to α_n

Inner product

So we have moved from (for example)

$$f(x) = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_q x_{iq}$$

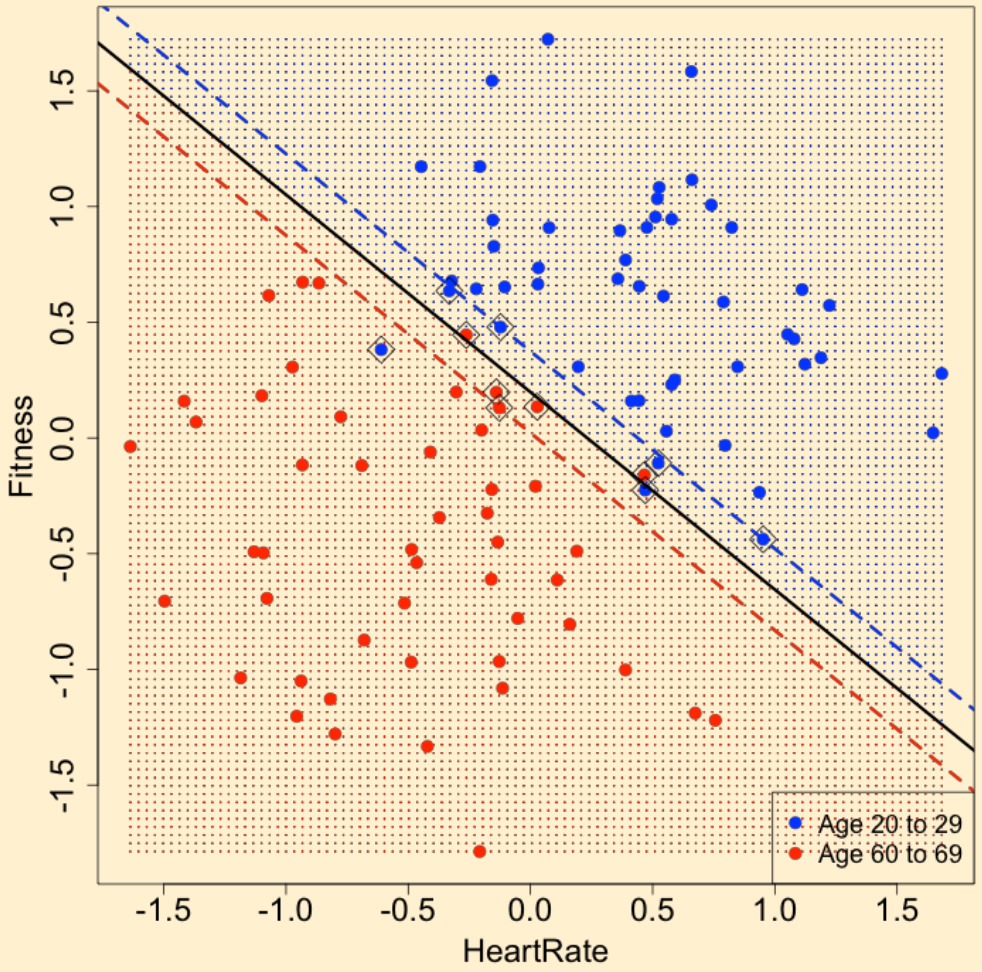
(where $q (\gg p)$, may incorporate a large number of transformations of the original p variables)

to:

$$f(x) = \beta_0 + \alpha_1 \langle x, x_1 \rangle + \alpha_2 \langle x, x_2 \rangle + \cdots + \alpha_n \langle x, x_n \rangle$$

Inner product

- To estimate the n parameters (α_1 to α_n) we need the inner product between every pair of points in the training set
- It turns out that the inner product is zero for all the points *except for* the support vectors (points within, on, or across the margin) -- this is why support vector machines are so computationally attractive!

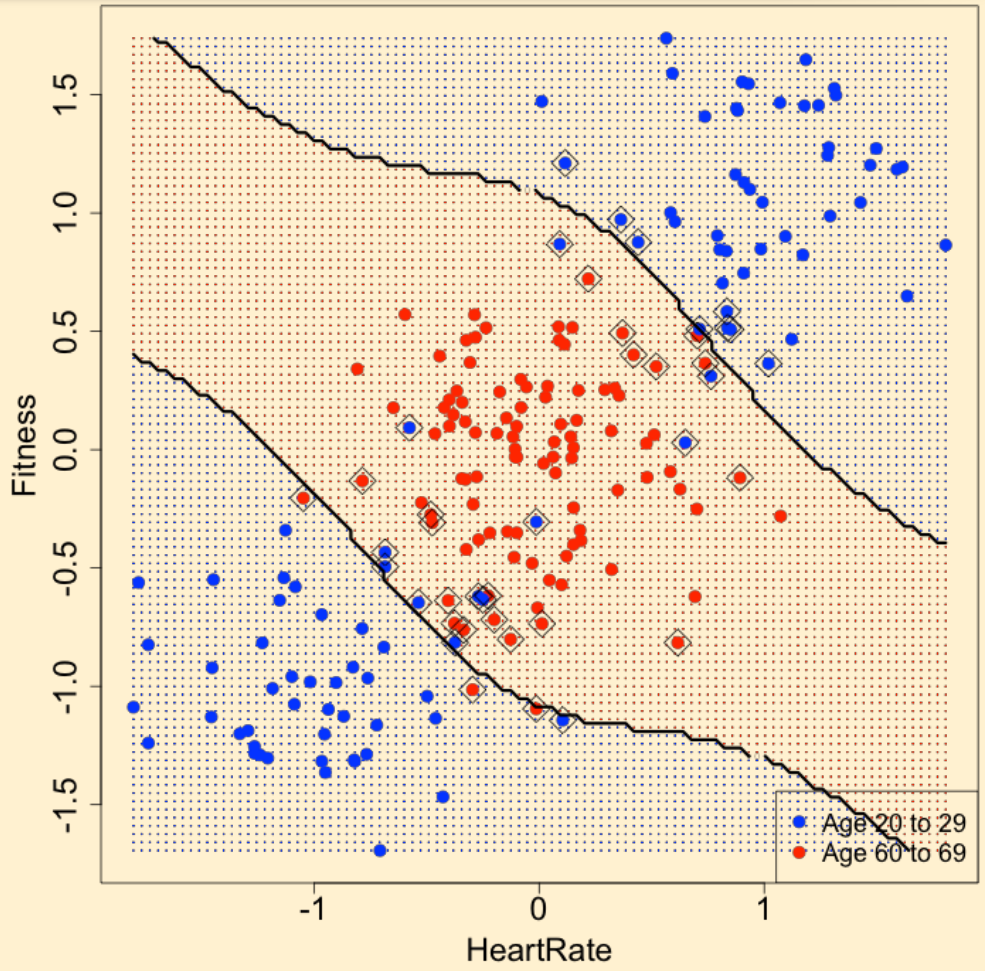


Kernels and support vector machines (gets pretty tough conceptually)

- If we can compute inner-products between observations (and for any new observations) we can fit a support vector classifier
- *Kernel functions* can do this for us, e.g. *polynomial kernel function:*
  $K(x_i, x_j) = (1 + x_{i1}x_{j1} + x_{i2}x_{j2} + \cdots + x_{ip}x_{jp})^d$
  computes inner-products for d-order polynomials
- For large d that means a massive number of transformed features
- But these kernels compute inner products over potentially massive feature space

Nonlinear boundary

- Back to our pathological case
- Use polynomial kernel with $d = 4$
- Cost = 10
- Did not require explicit expansion of feature set




```
svmFitSVM <- svm(ageGrp ~ Fitness + HeartRate, data=dat, kernel="polynomial", degree=4, cost
=10, scale=FALSE)
```

Radial Kernel

- A very popular kernel that we will try out in the lab
- $K(x_i, x_j) = \exp\left(-\gamma\left[(x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2 + \cdots + (x_{ip} - x_{jp})^2\right]\right)$
- γ is an additional parameter that needs to be tuned (controls smoothness: larger value implies more smooth)
- Leads to implicit feature space that is very high dimensional, but controls for overfitting by "squashing down" most dimensions

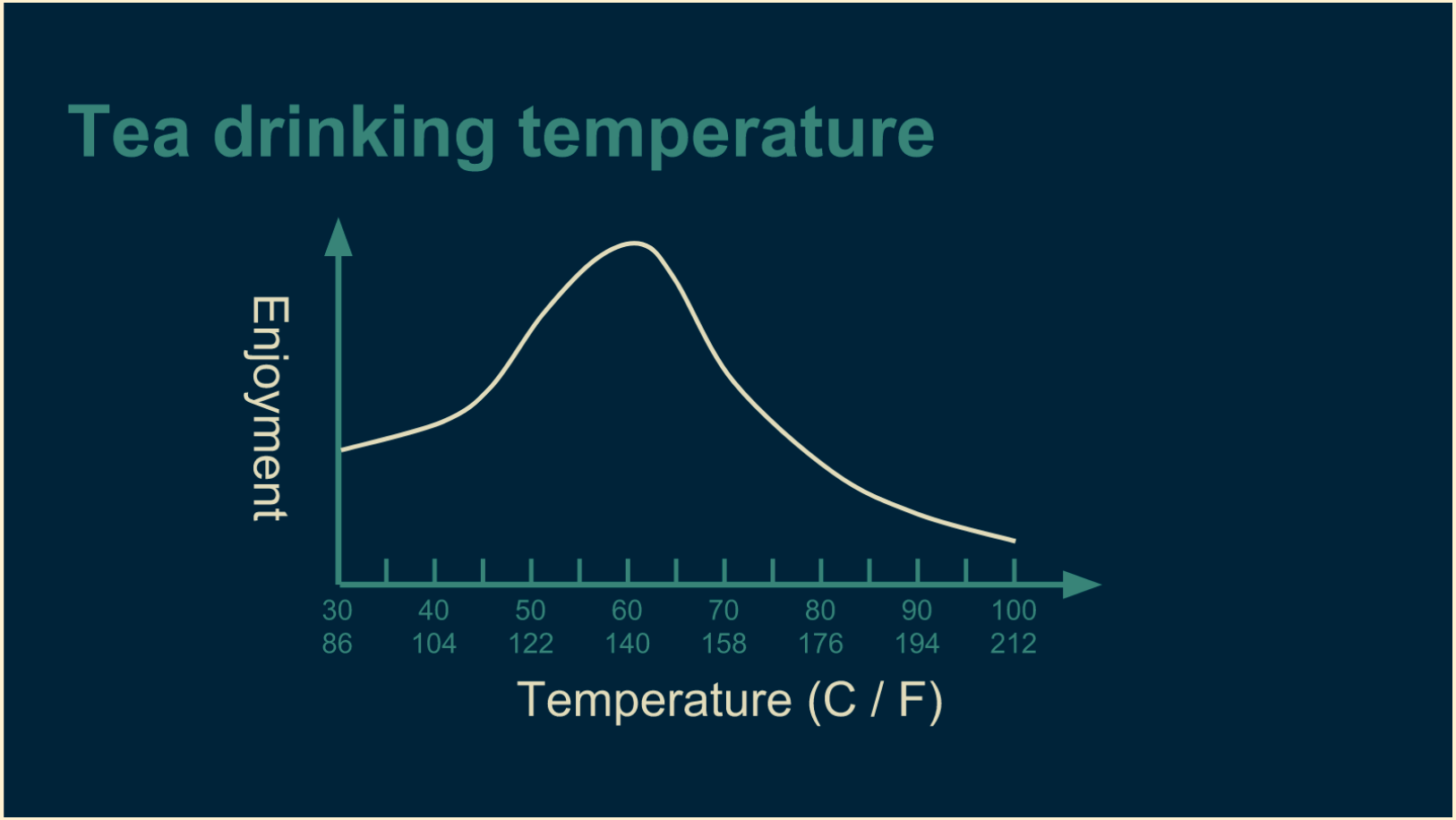
Multiple classes

- SVM is not a natural procedure for more than 2 classes, but it can work well
- Two *ad hoc* approaches
  - OVA = One versus all – fit each class vs. the rest and classify to the one with the largest function value
  - OVO = One versus one – fit each class relative to each other and classify to the one that wins the most pairwise competitions
- OVO usually favored unless large number of classes when OVA used

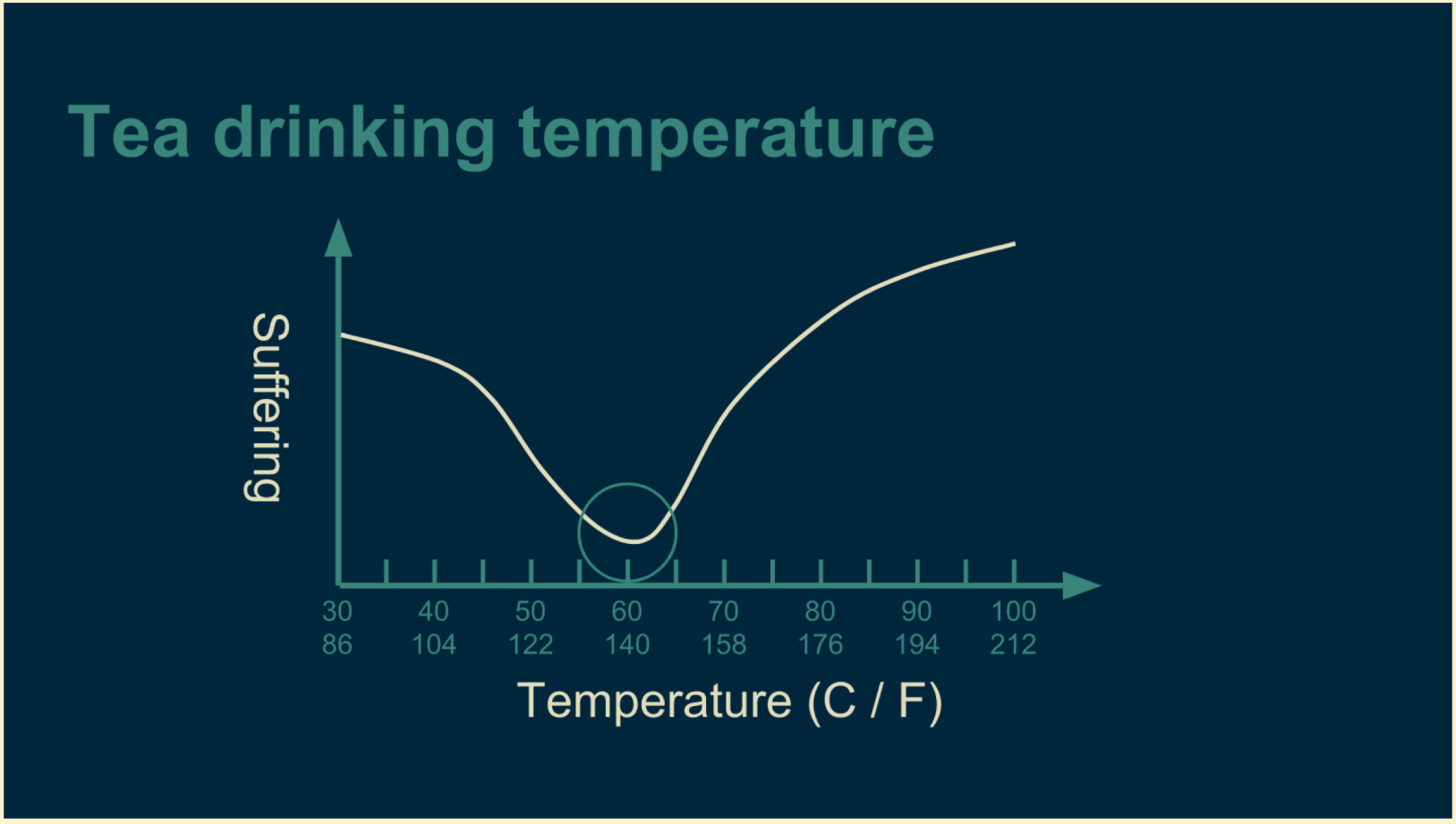
Support Vector Machines

- *Disadvantage:* no interpretable results – close to "black box"
- *Disadvantage:* don't get probabilities of classes naturally
- *Advantage:* gives great results for many types of complex datasets (winner of many big data machine learning competitions)
- *Advantage:* the big deal is that SVM is a "convex optimization" problem (only single optimal solution) and that is why Support Vector Machines overtook neural networks (with its potential for getting stuck in local maxima) – but now neural networks are back on the ascendance with deep learning!
- *For more information:* see a nice lecture on SVMs by Patrick Winston https://www.youtube.com/watch?v=_PwhiWxHK8o that gives more development of the mathematical ideas and trickery behind SVMs

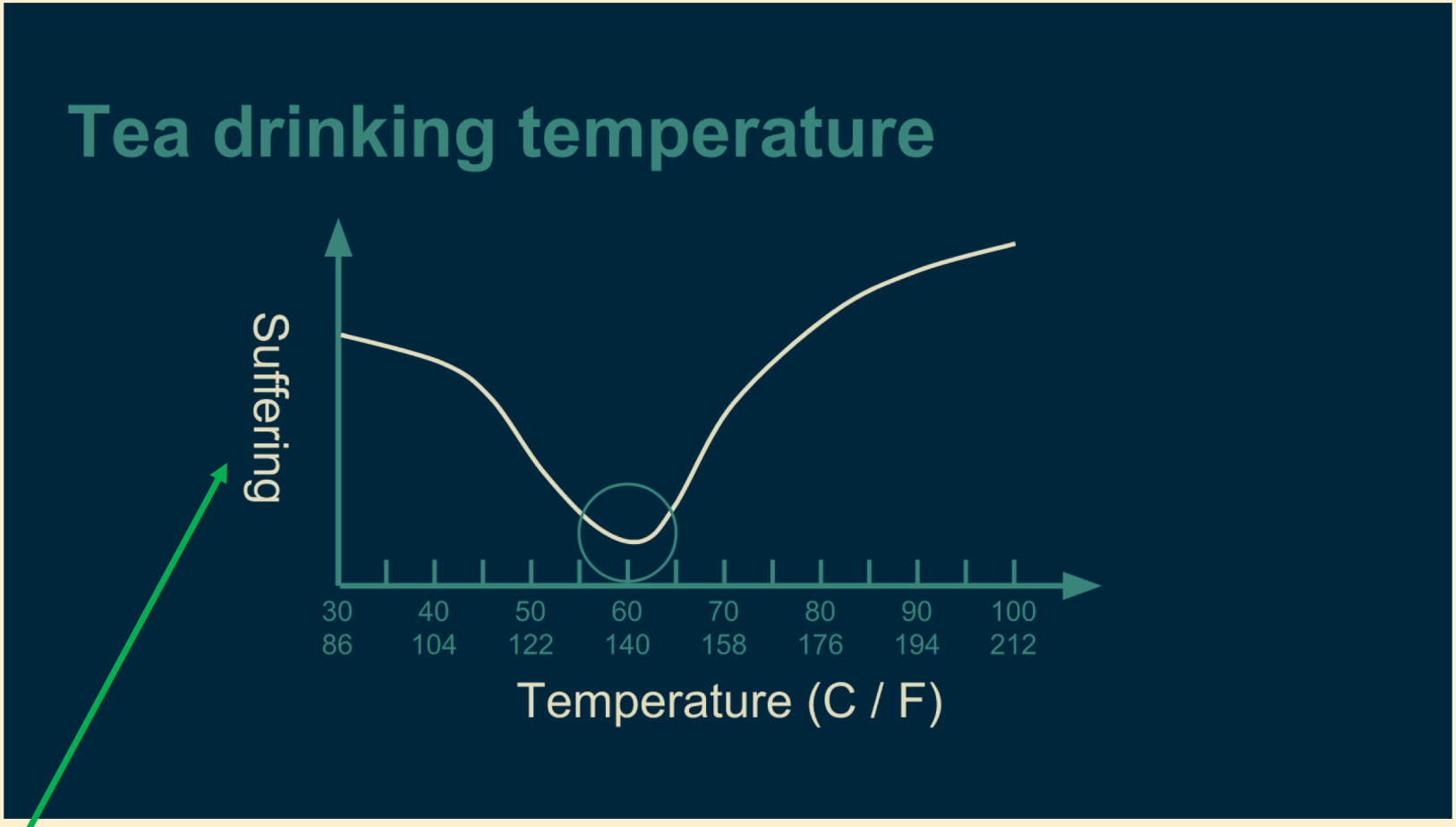
But what is optimization?



But what is optimization?

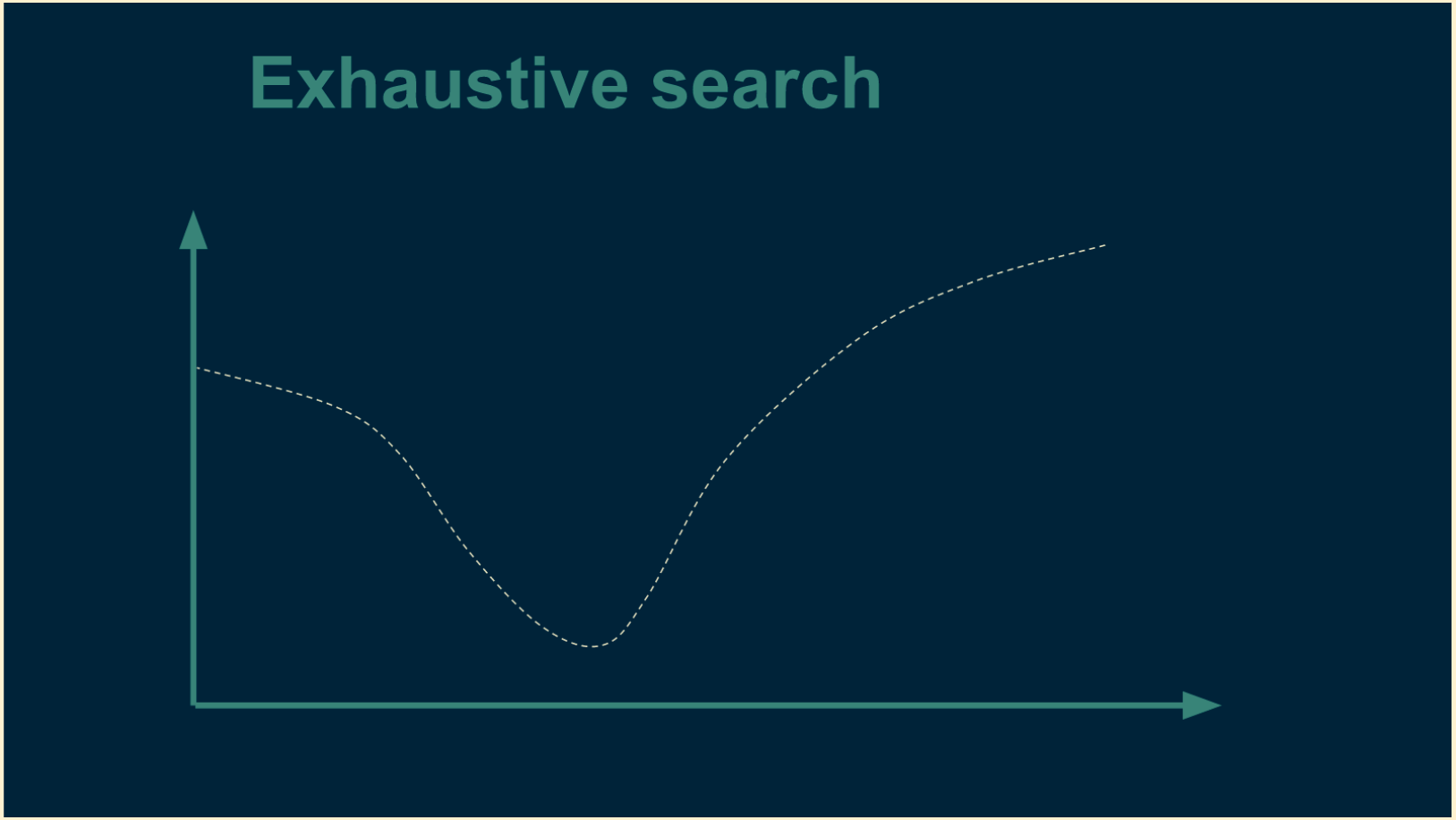


But what is optimization?

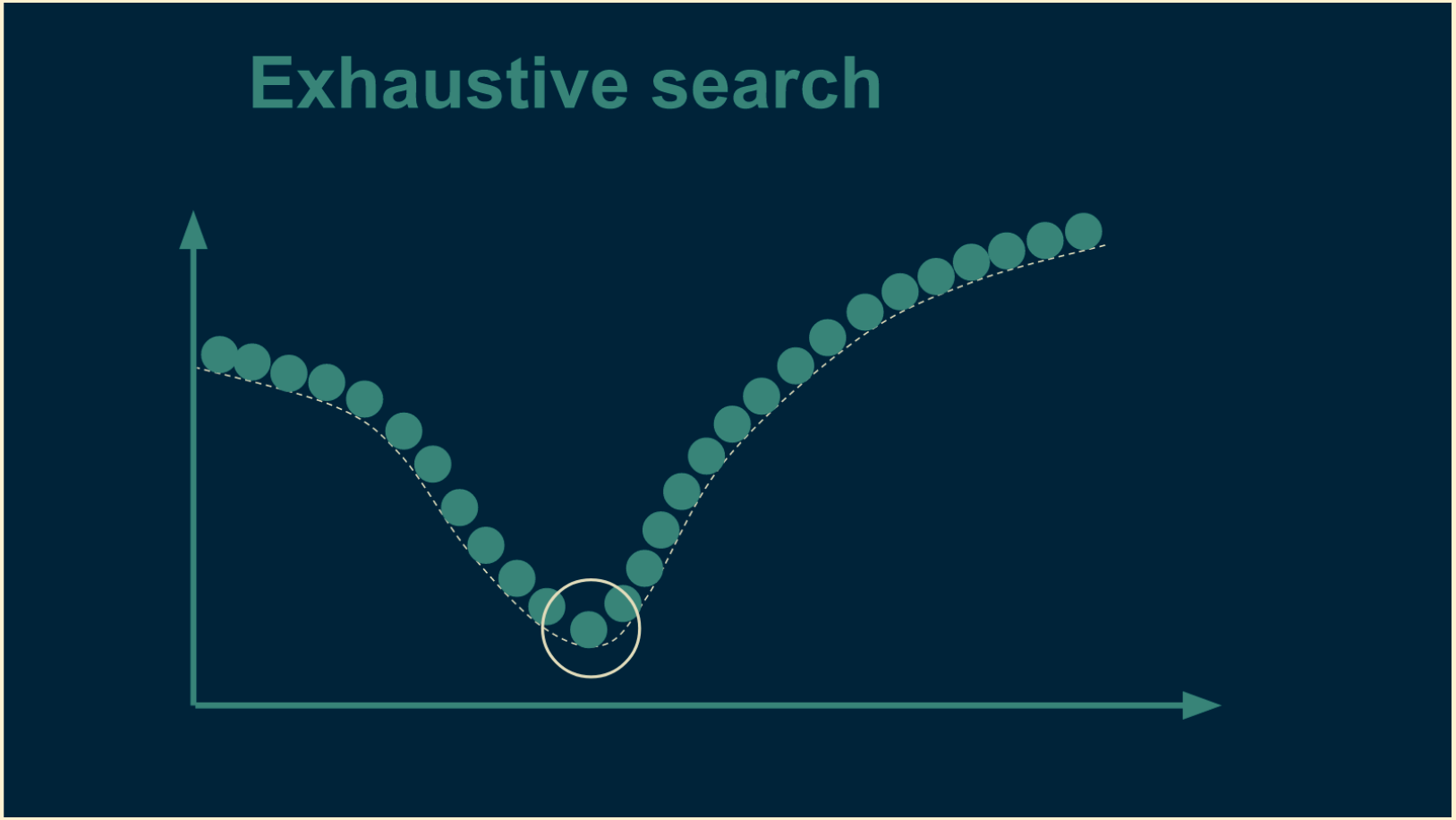


A type of loss function – compare with least squares or misclassification rate

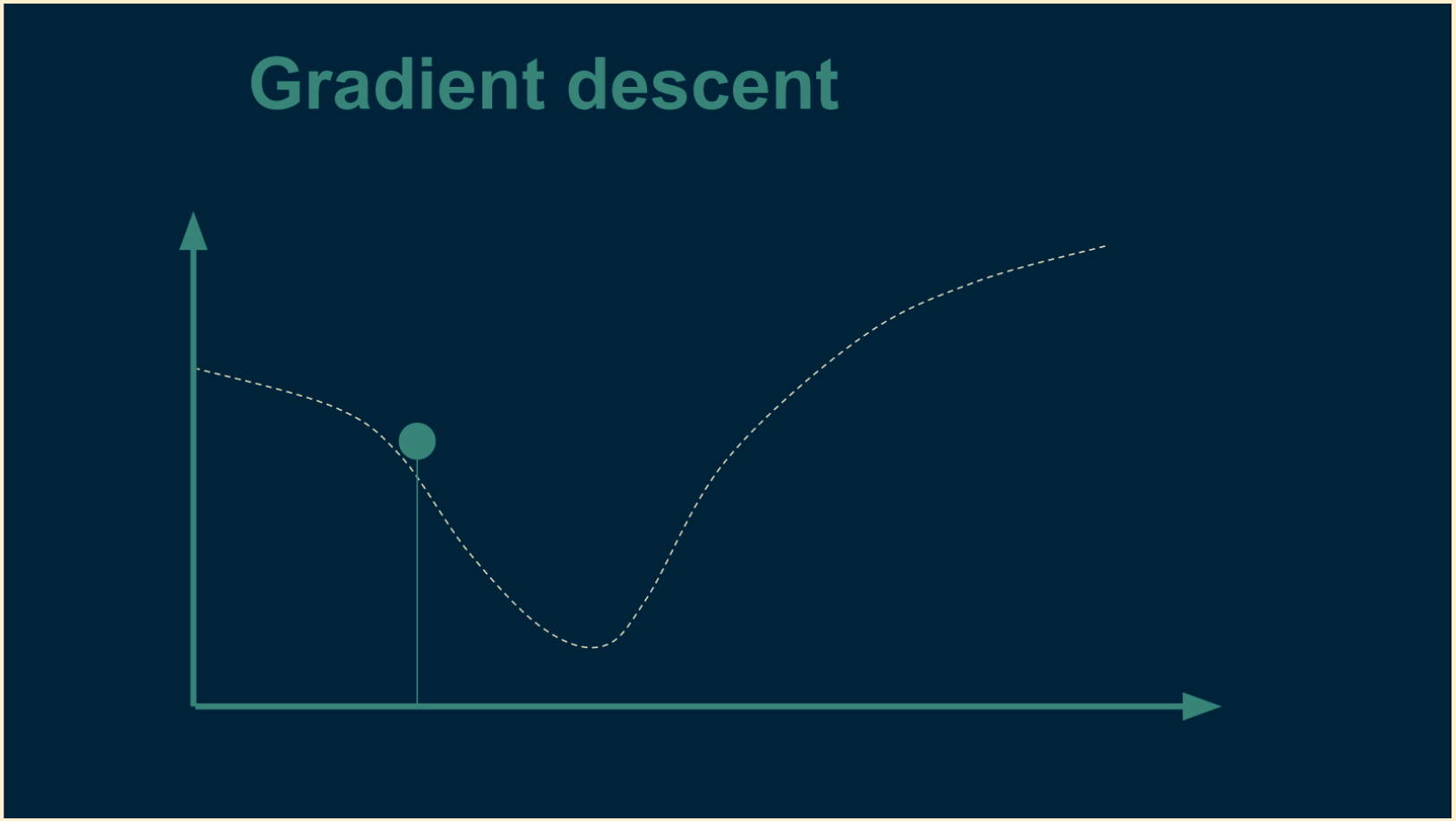
Exhaustive search



Exhaustive search

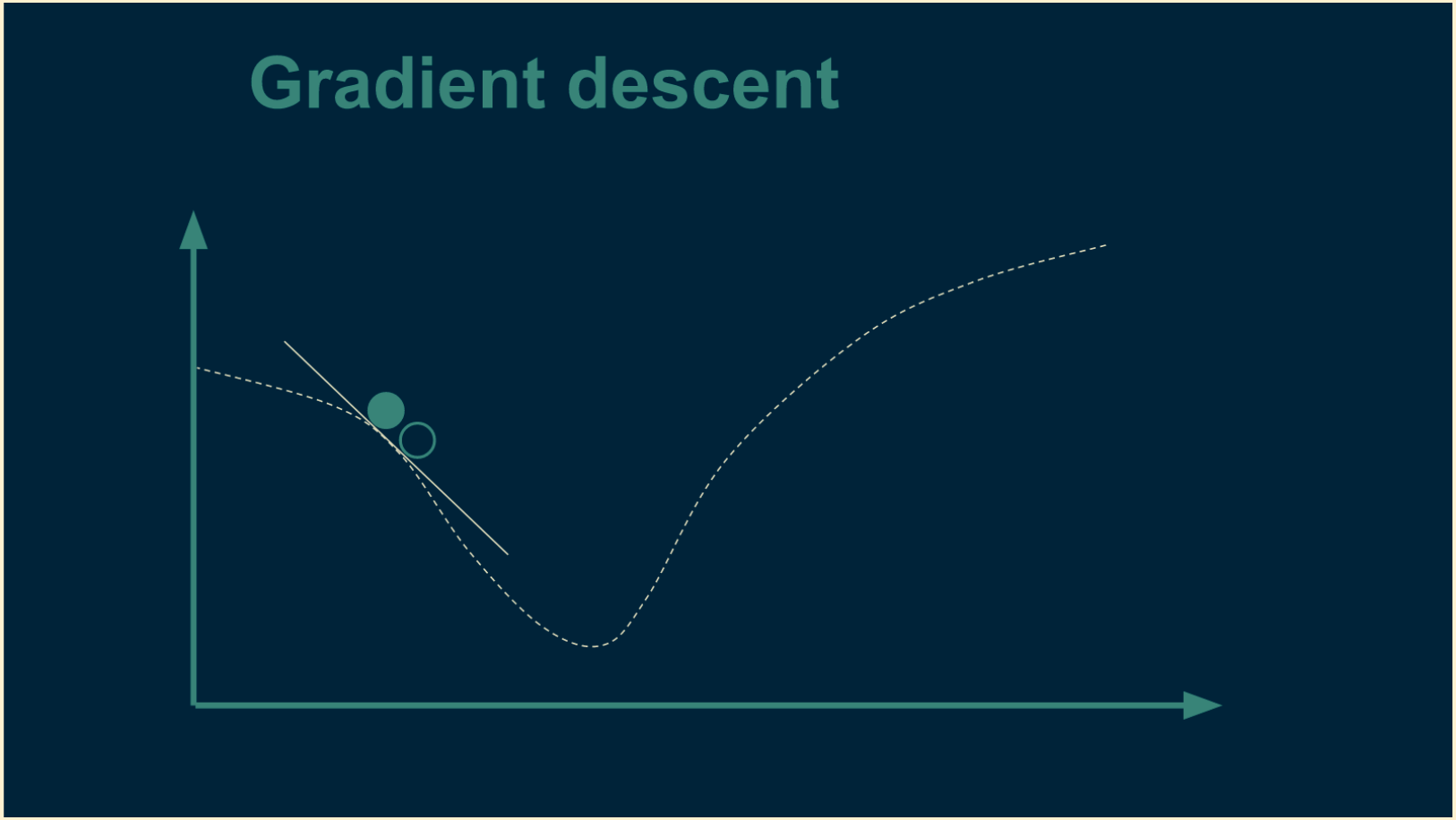


Gradient descent

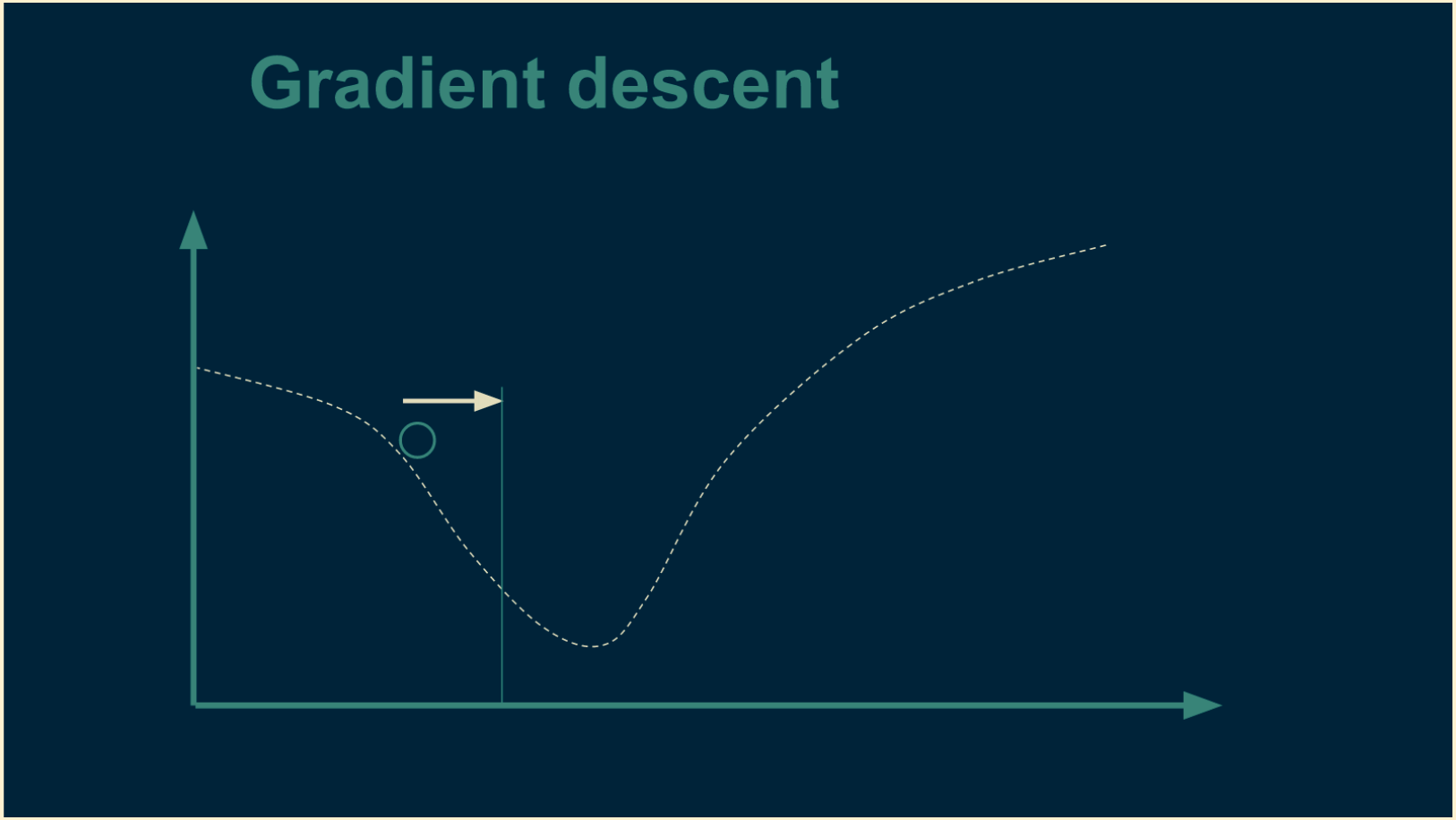


The “bread-and-butter” of optimization for machine learning

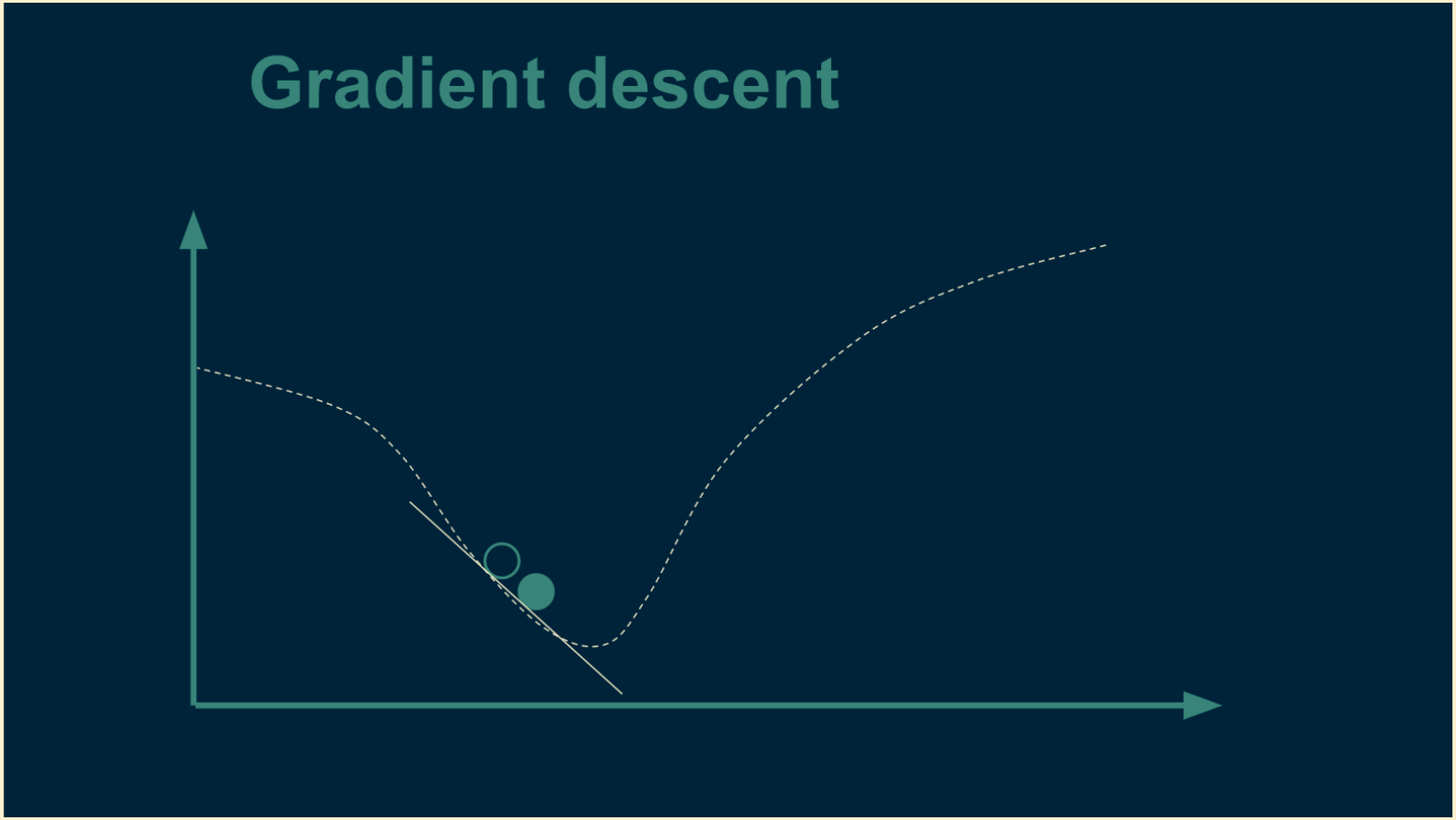
Gradient descent



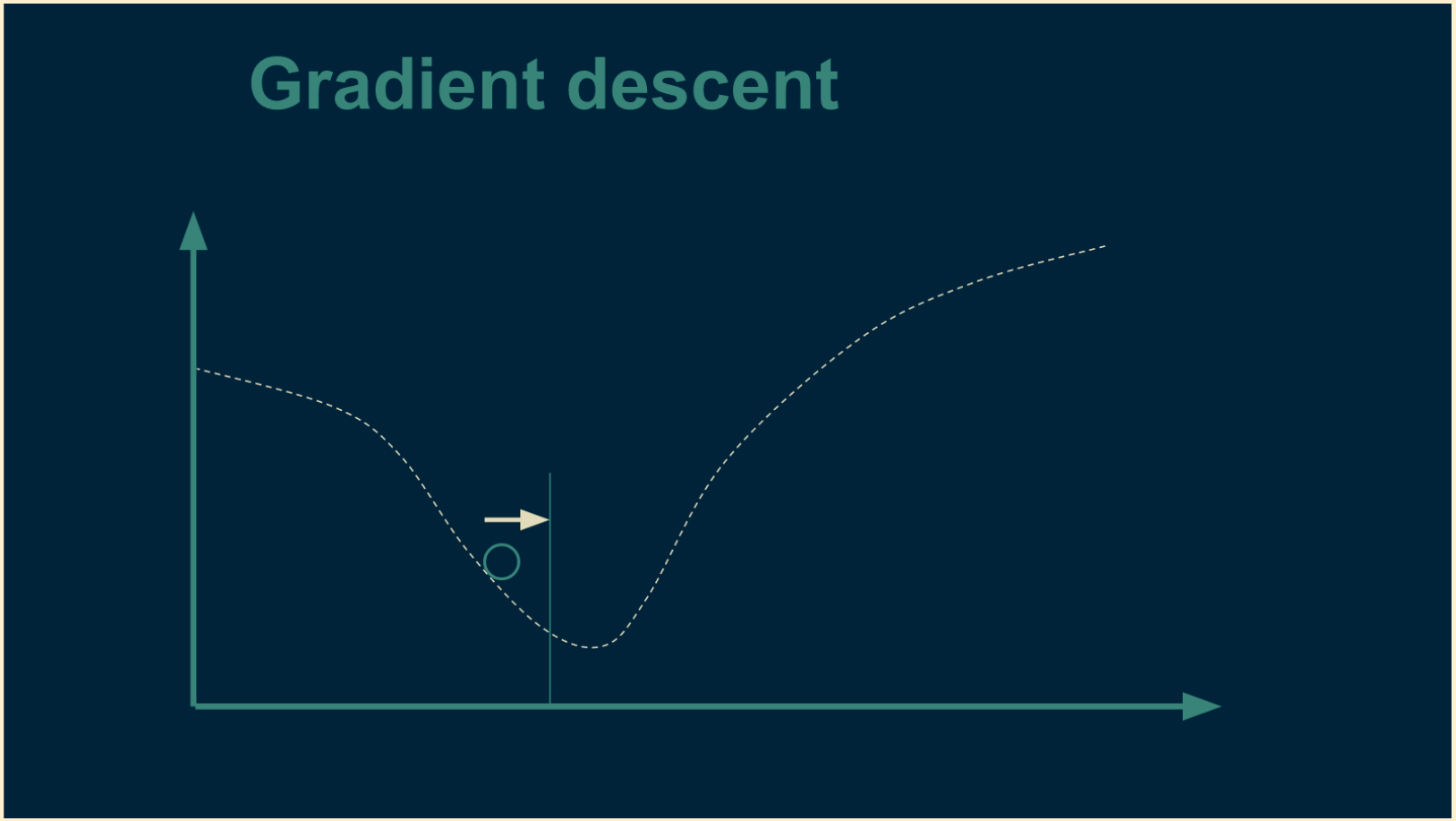
Gradient descent



Gradient descent

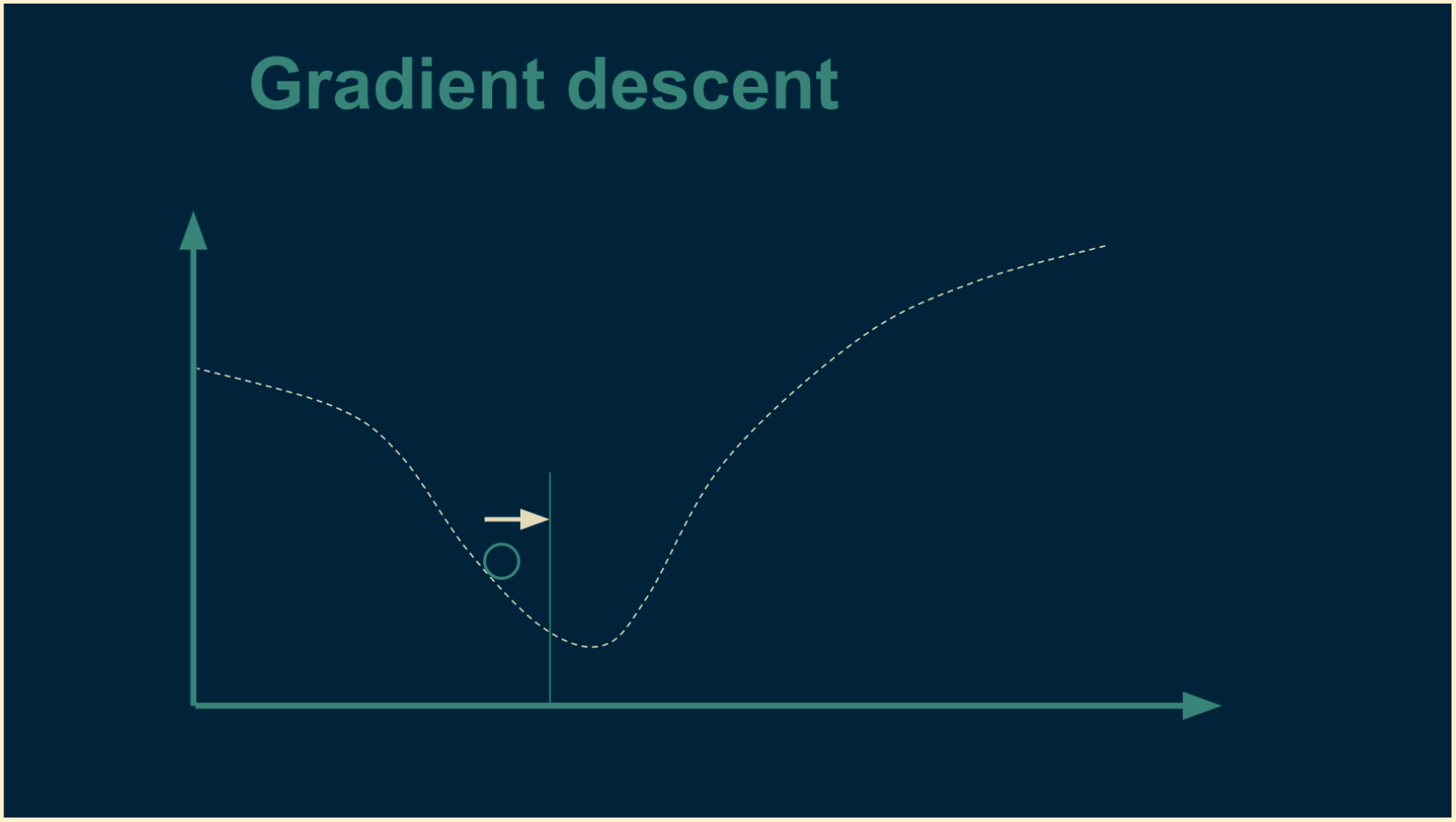


Gradient descent



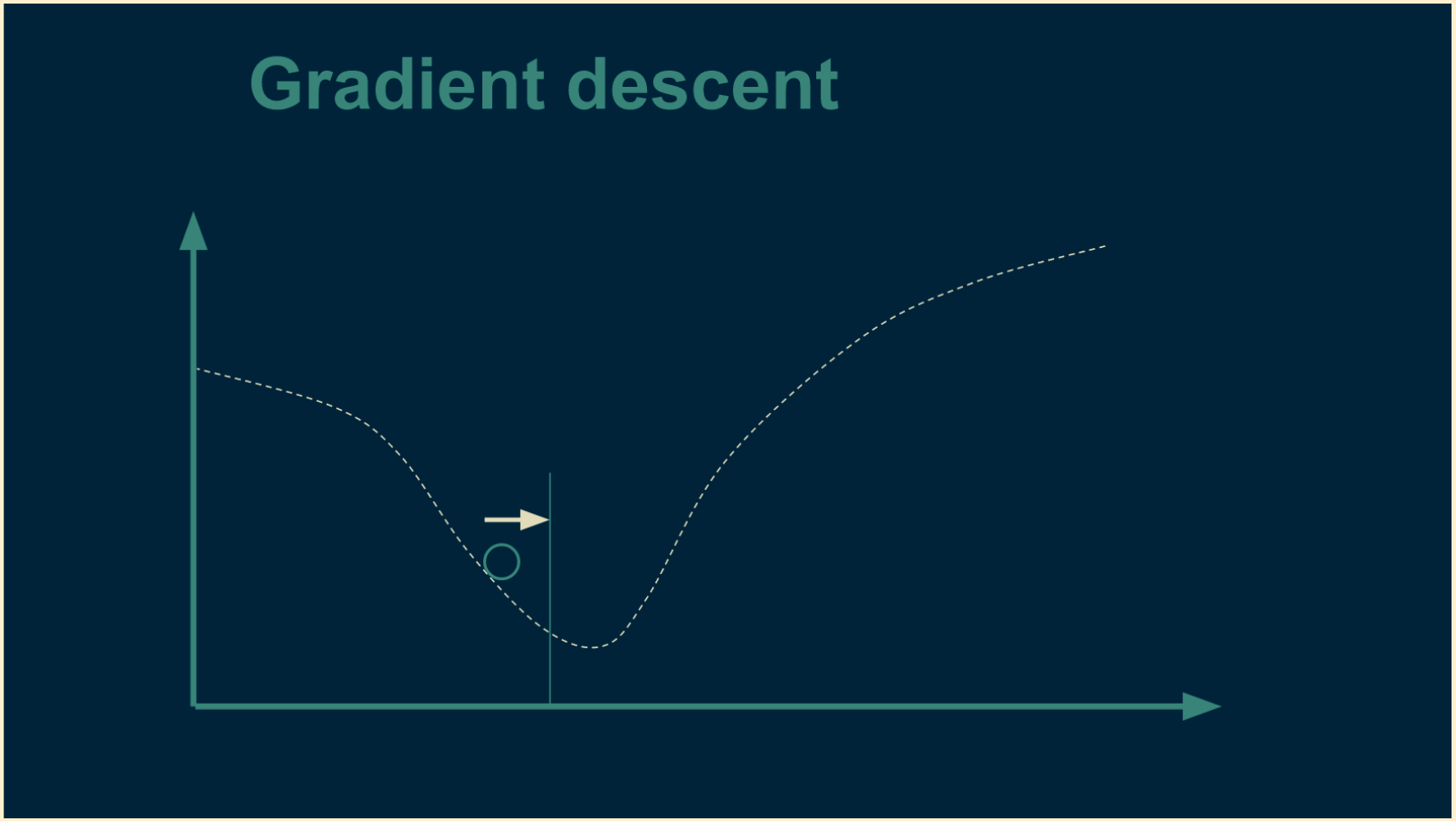
Keep going until there are no further changes

Gradient descent



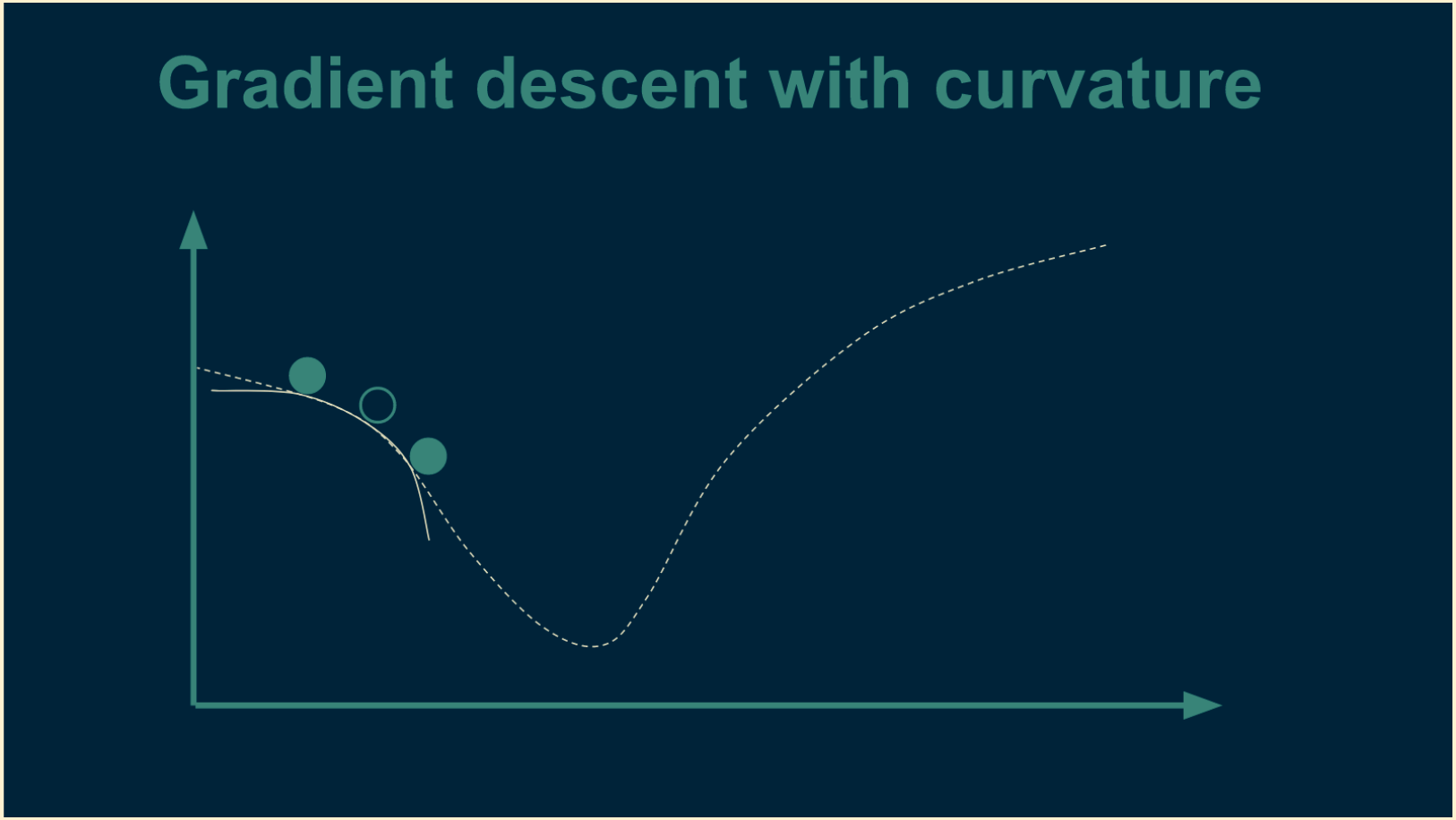
Size of steps controlled by a parameter α (that is multiplied by the gradient) – α too small and gradient descent takes for ever; α too big and can overshoot minimum back-and-forth

Gradient descent

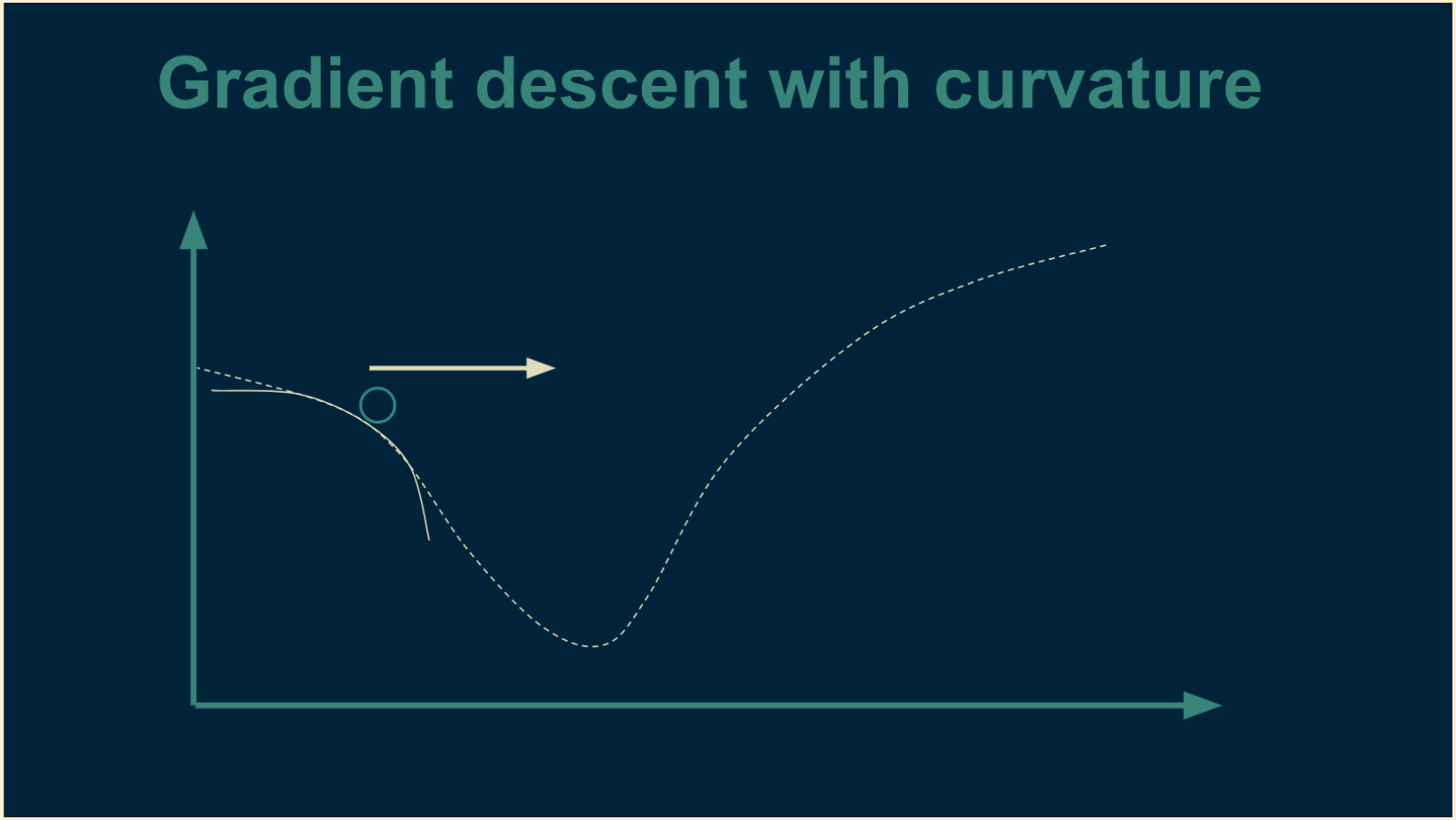


We can extend this to multiple dimensions

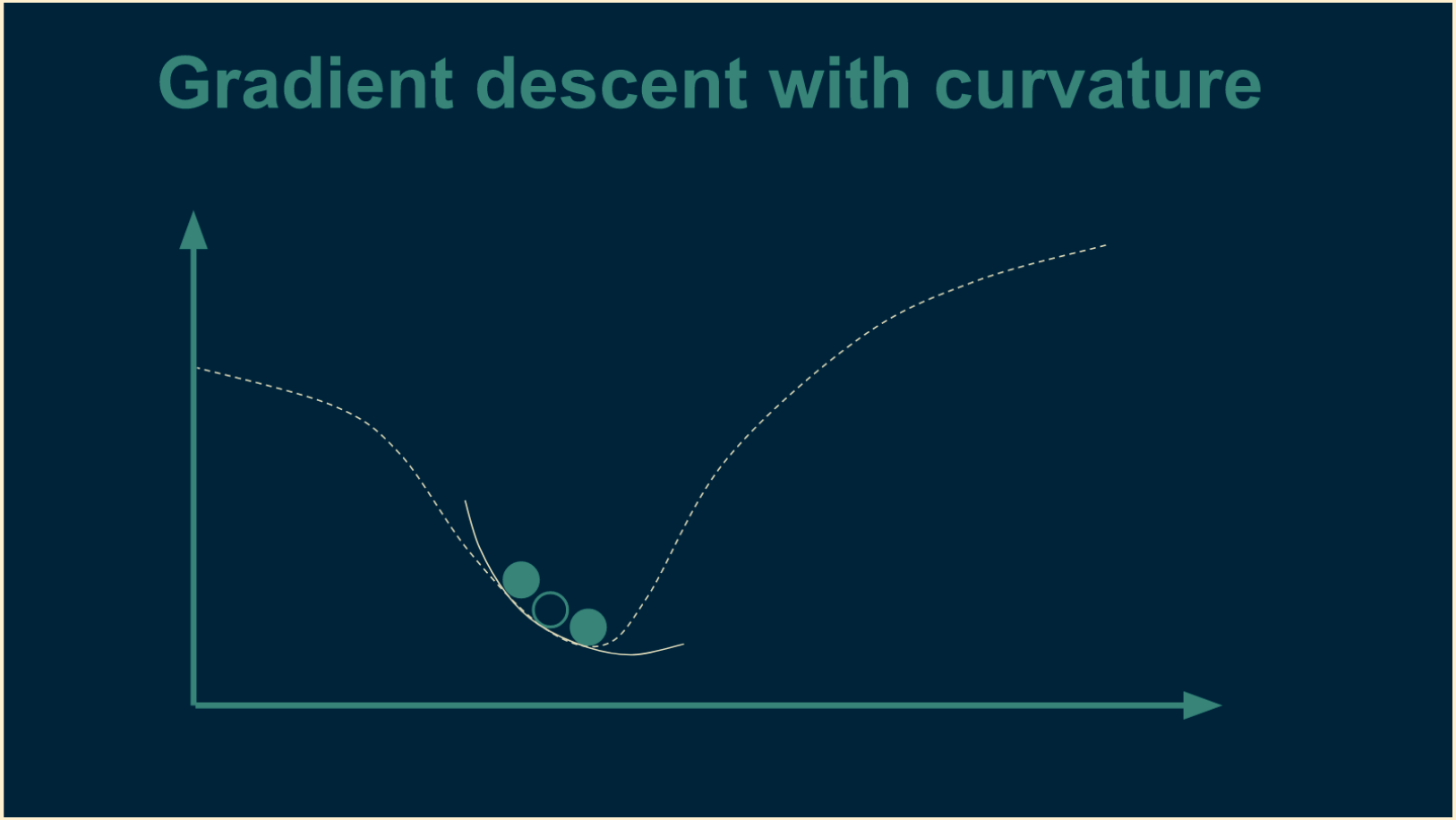
Gradient descent considering curvature



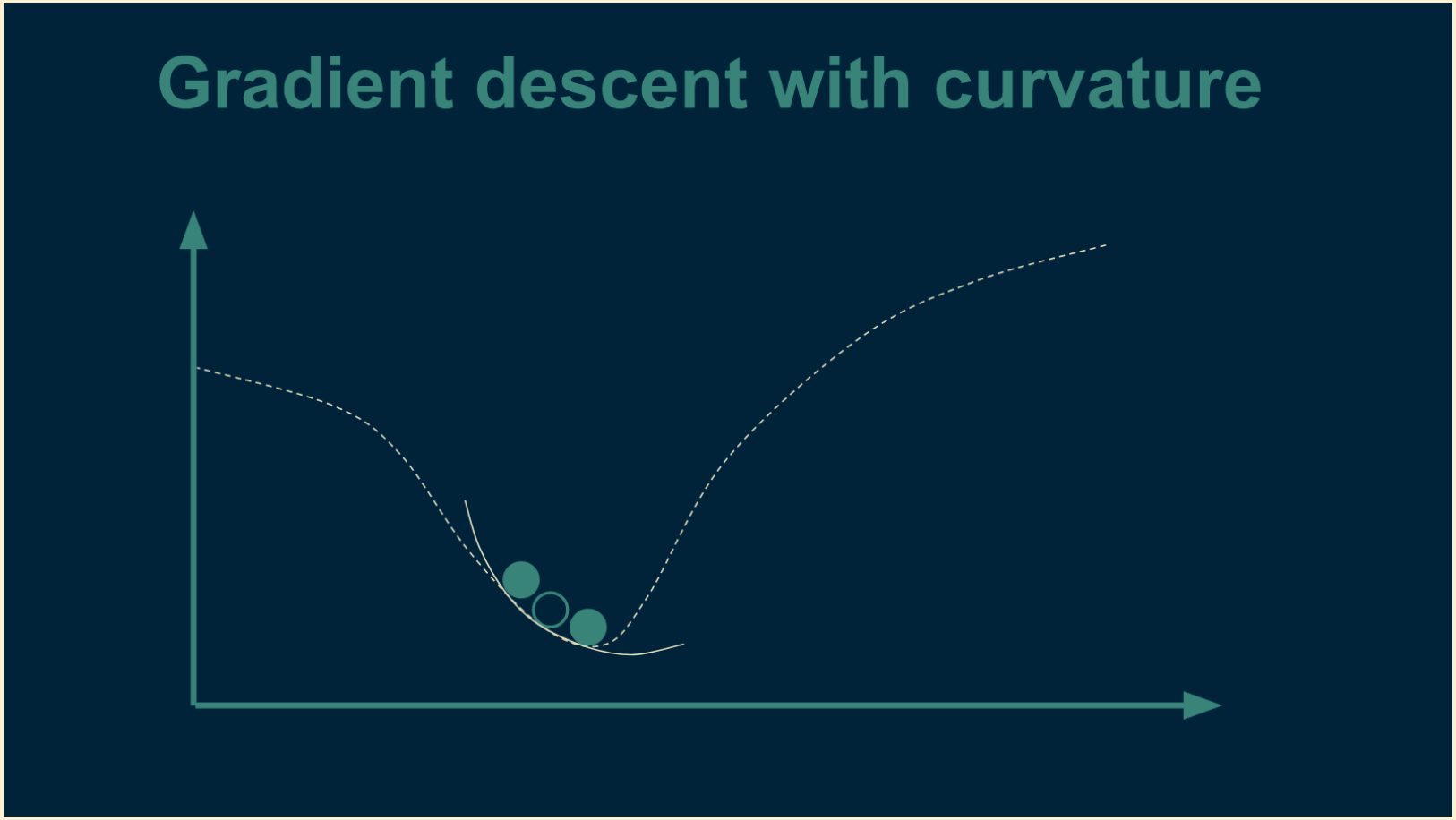
Gradient descent considering curvature



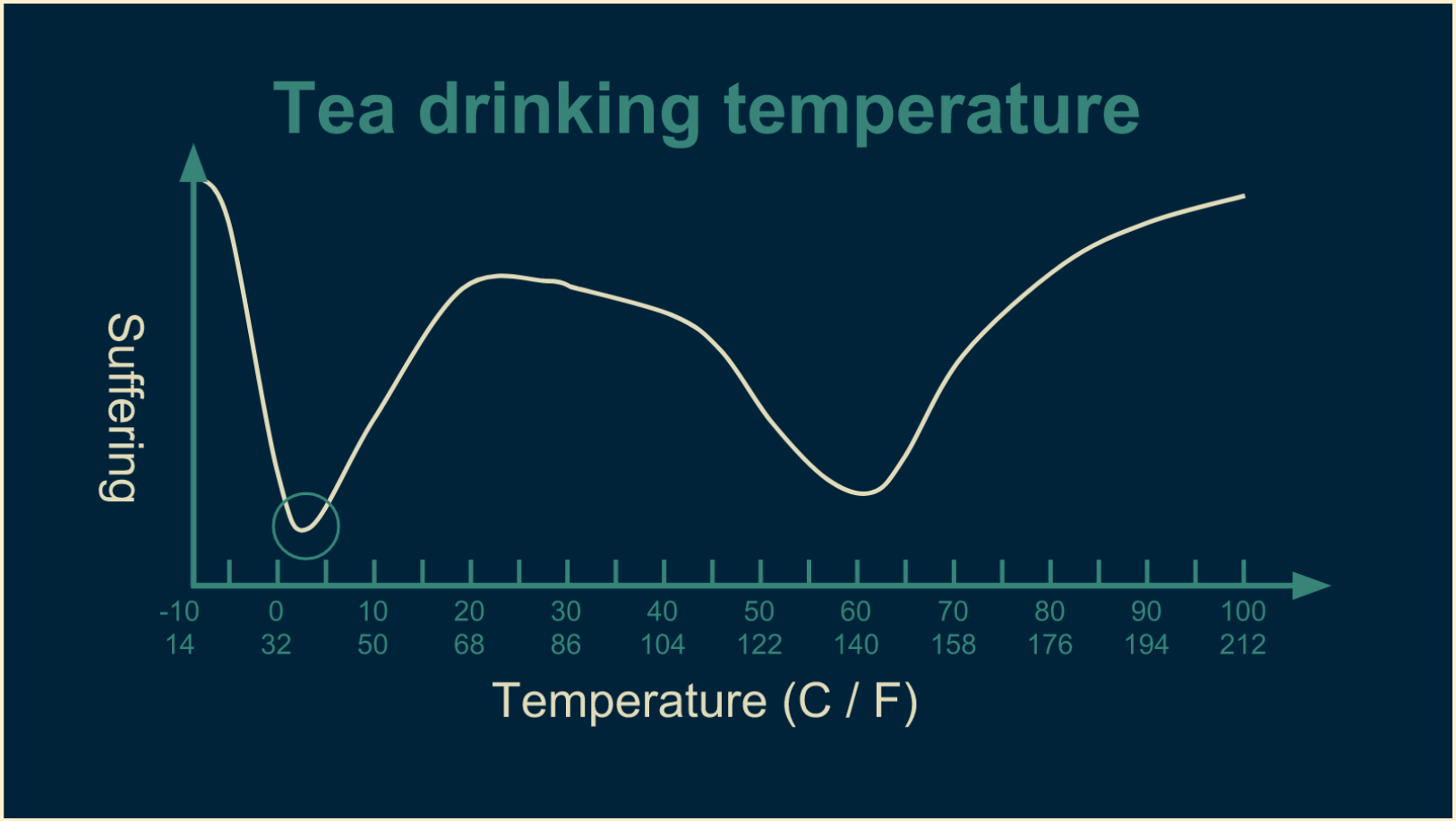
Gradient descent considering curvature



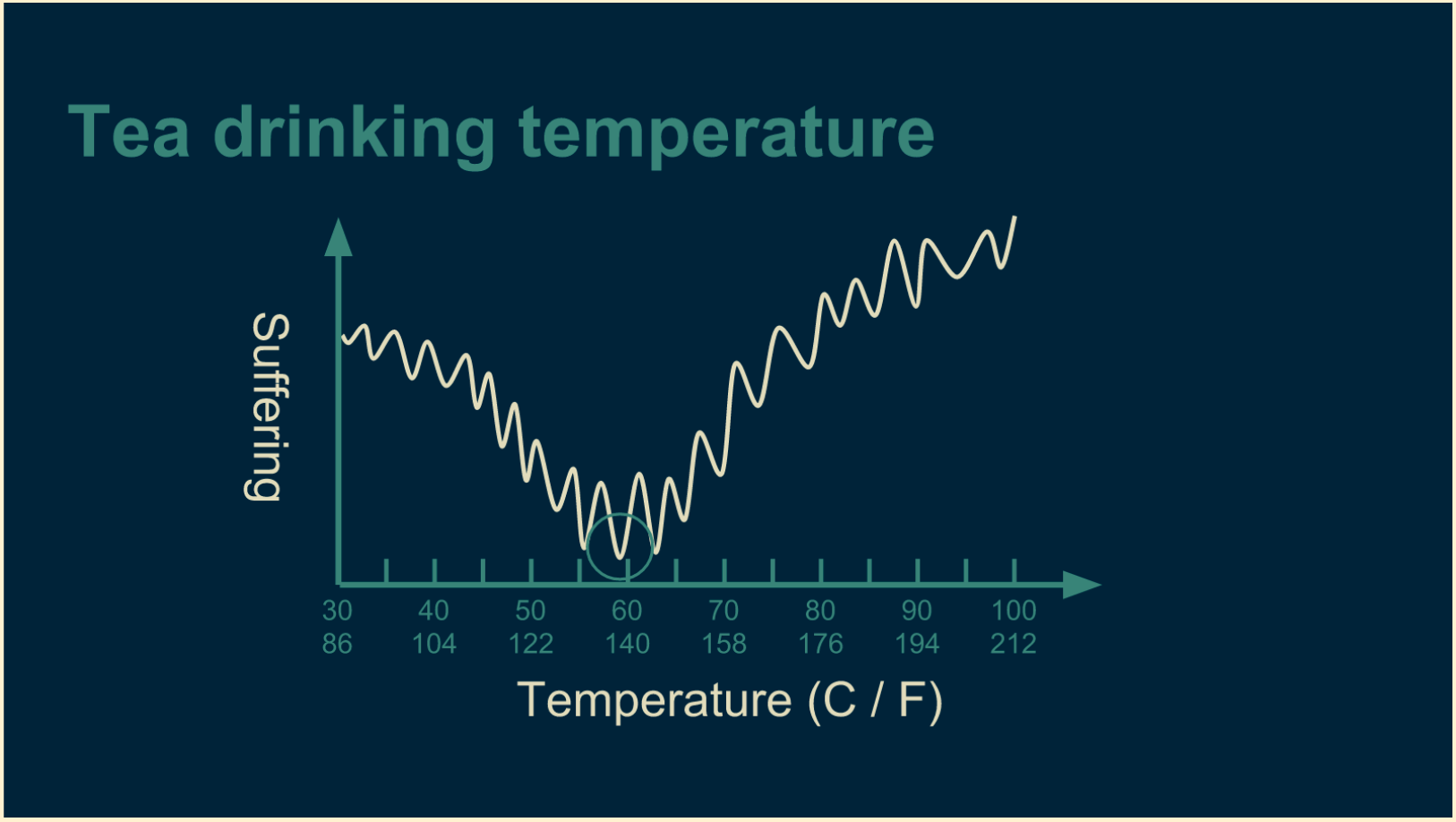
Gradient descent considering curvature



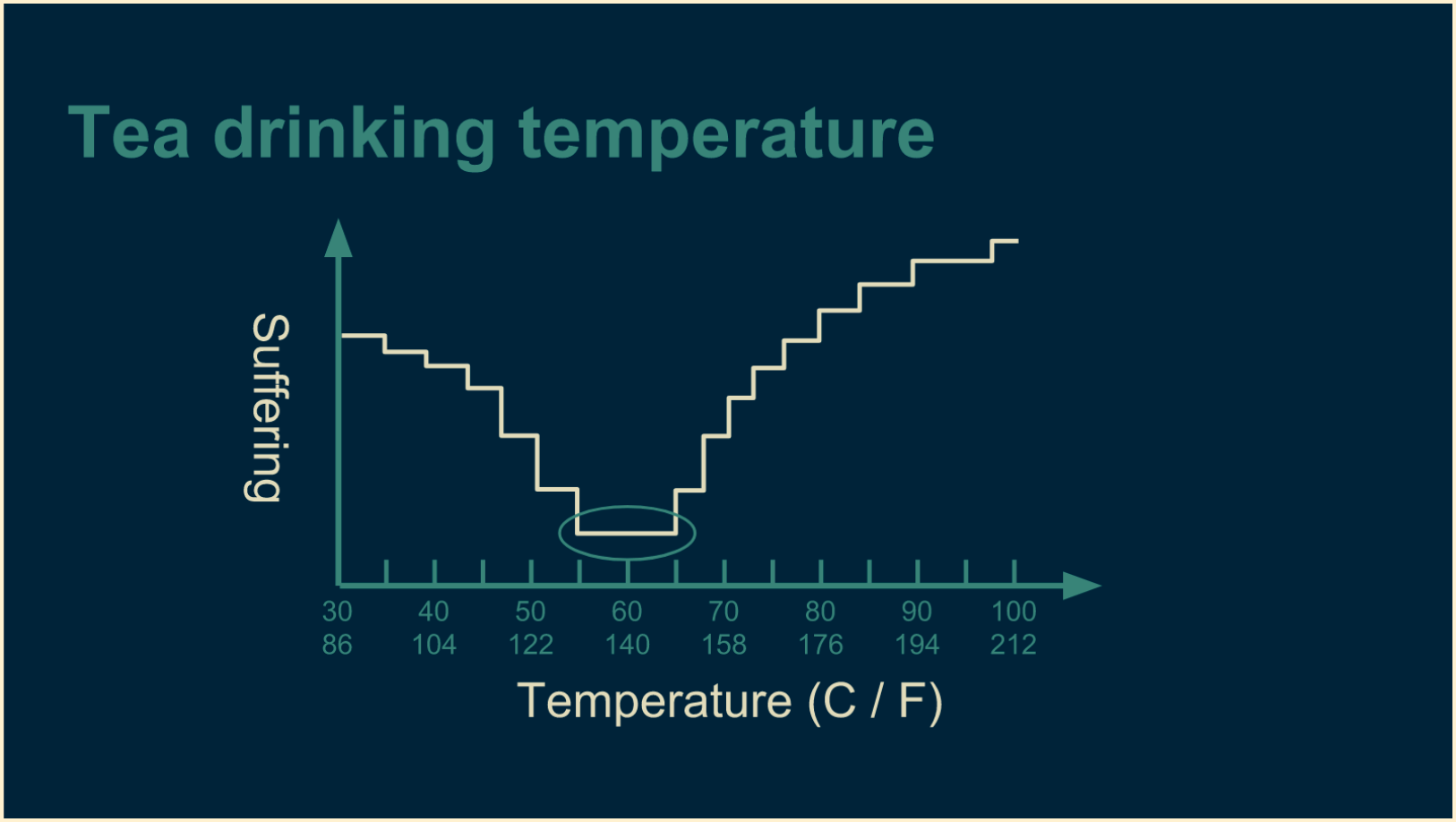
But we need to be careful



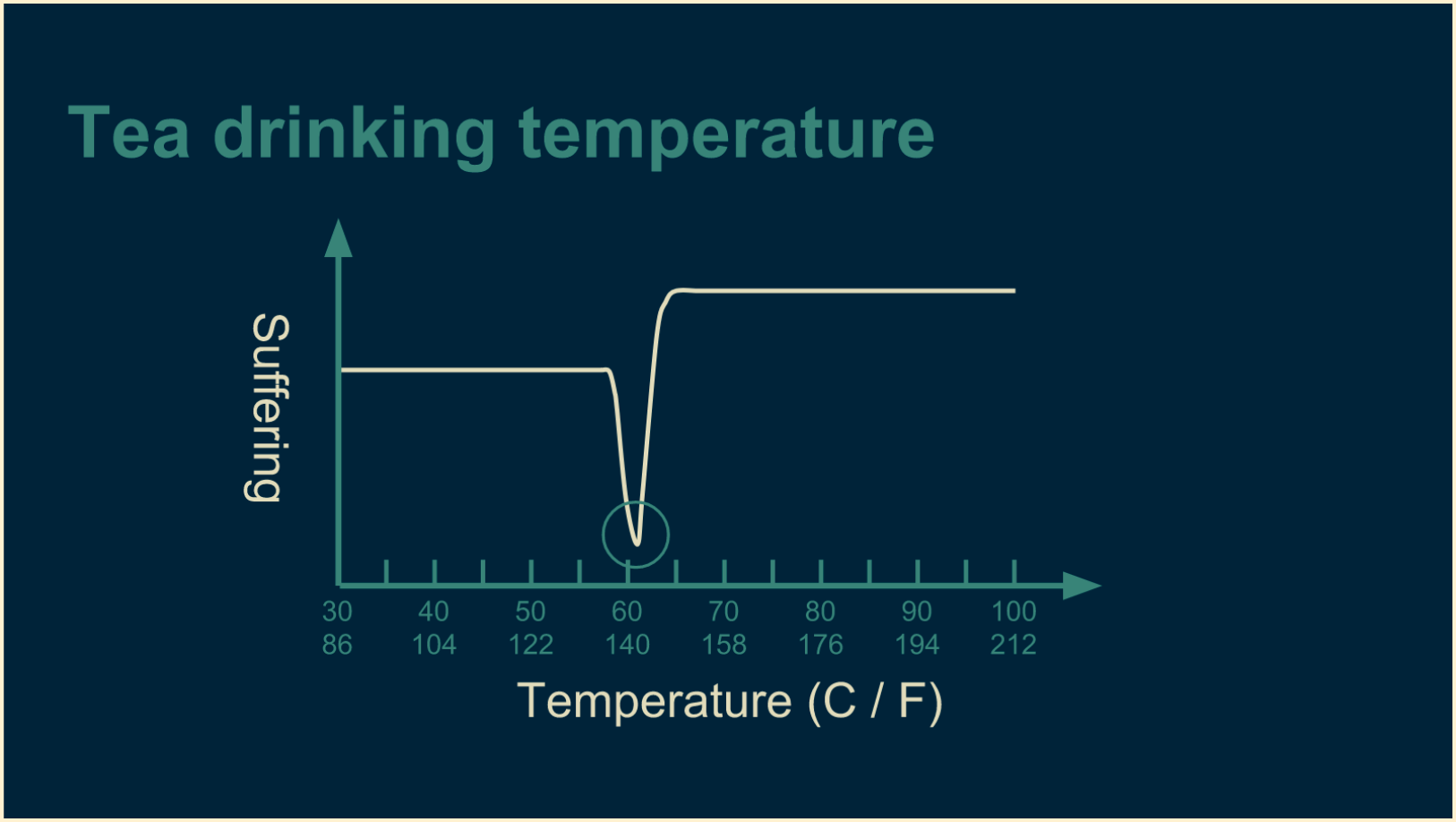
But we need to be careful



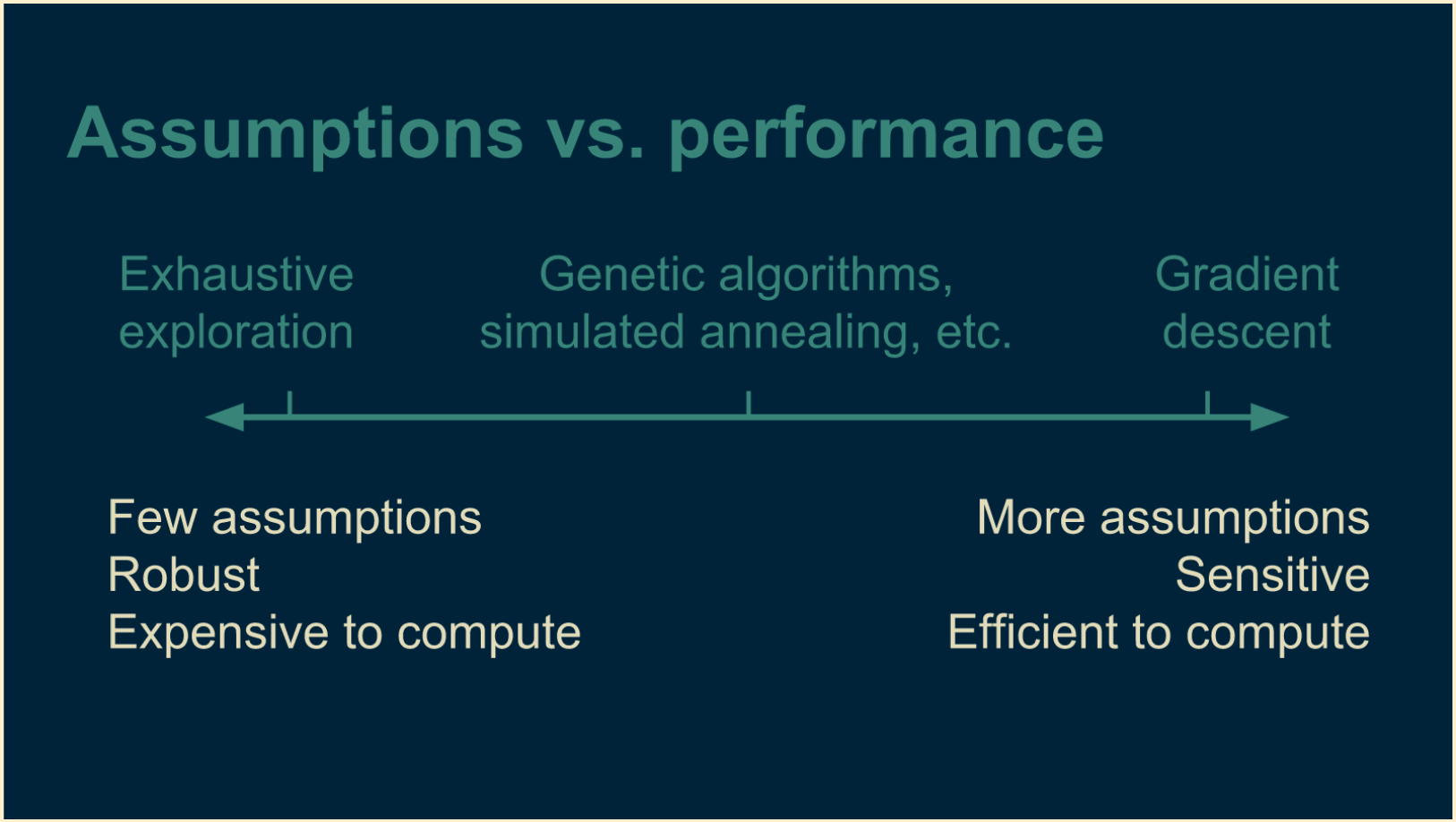
But we need to be careful



But we need to be careful



Robust approaches



Next Lecture – penalized regression

- Dr. Mark Segal will give a lecture on Recursive Partitioning (Regression Trees), bagging, and Random Forests