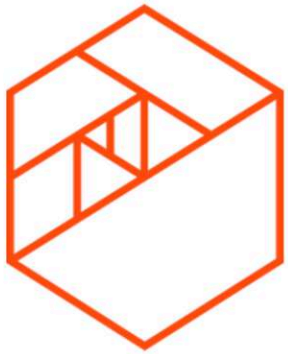
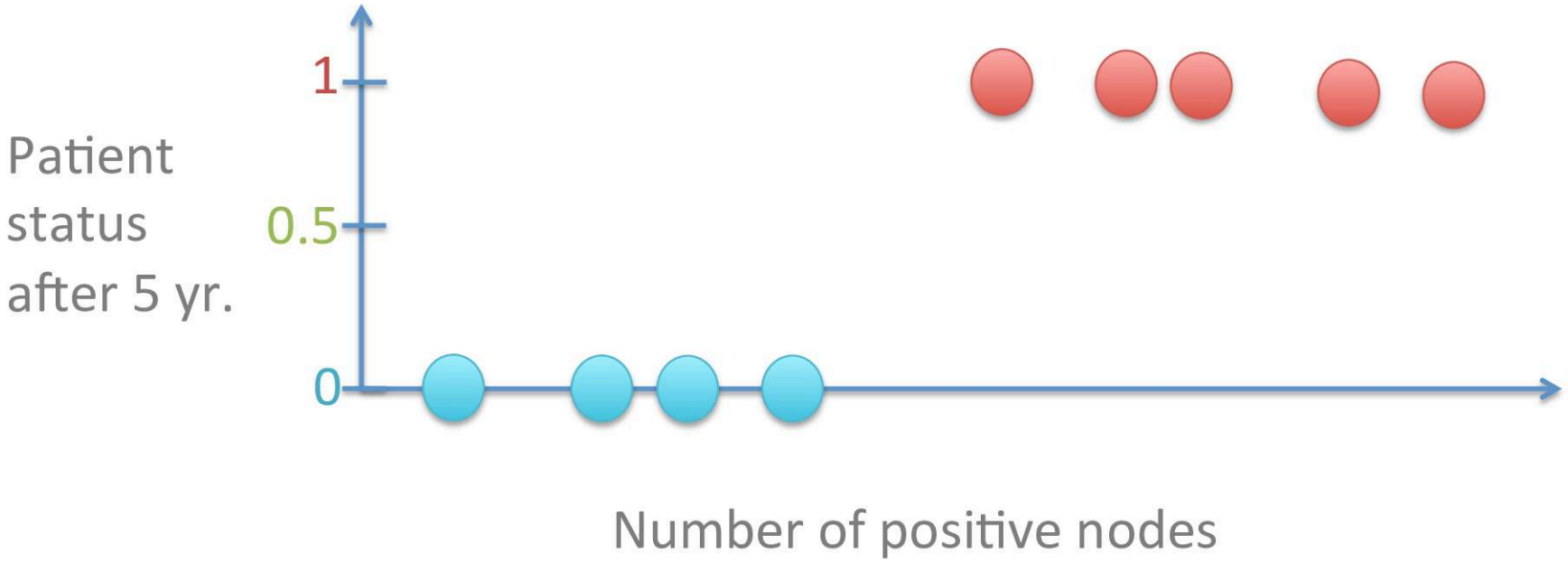
Support Vector Machines



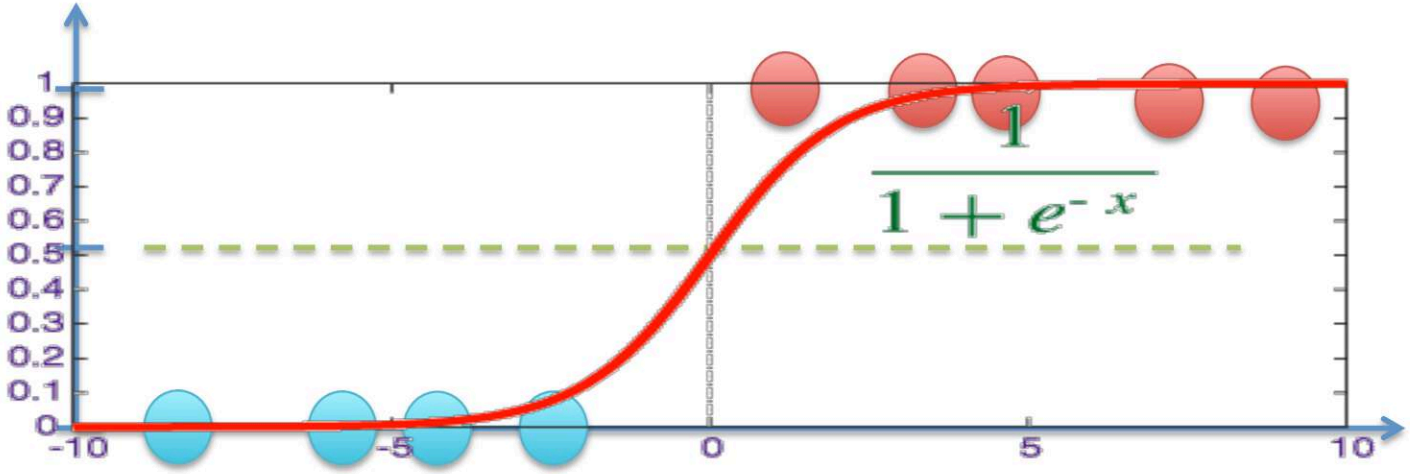
DATA SCIENCE BOOTCAMP

1 feature (num. nodes), 2 labels (survived/not)



Logistic Regression

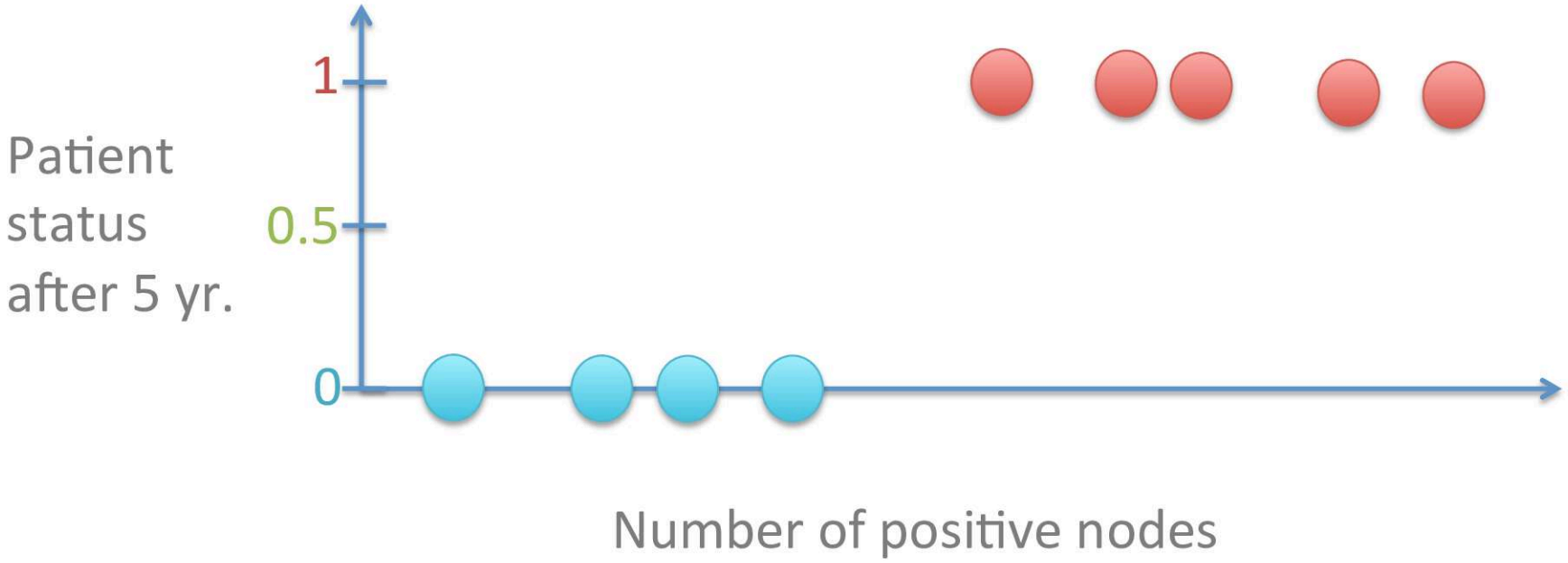
Patient status after 5 yr.



Number of positive nodes

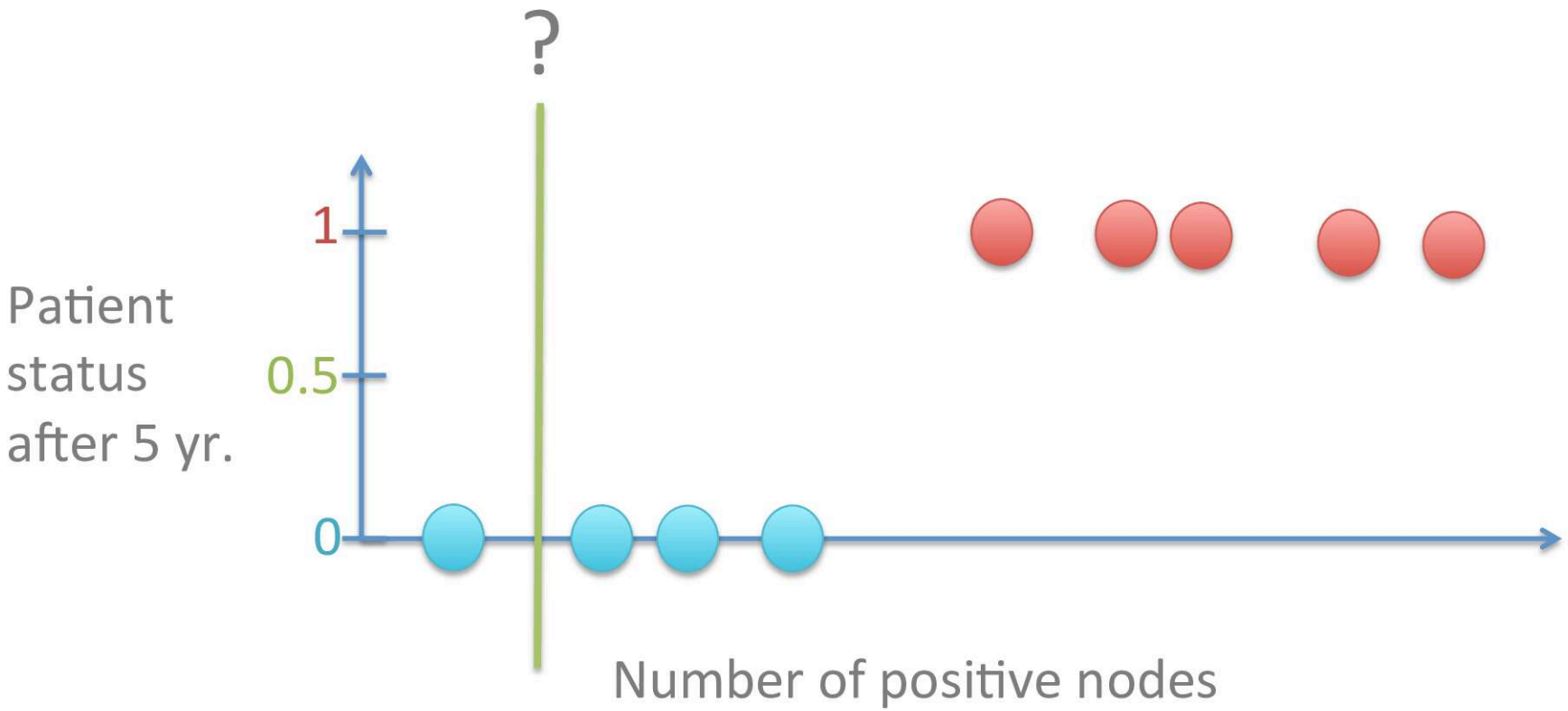
$$y_\beta(x) = \frac{1}{1+e^{-(\beta_0+\beta_1 x+\varepsilon)}}$$

Support Vector Machine (SVM)



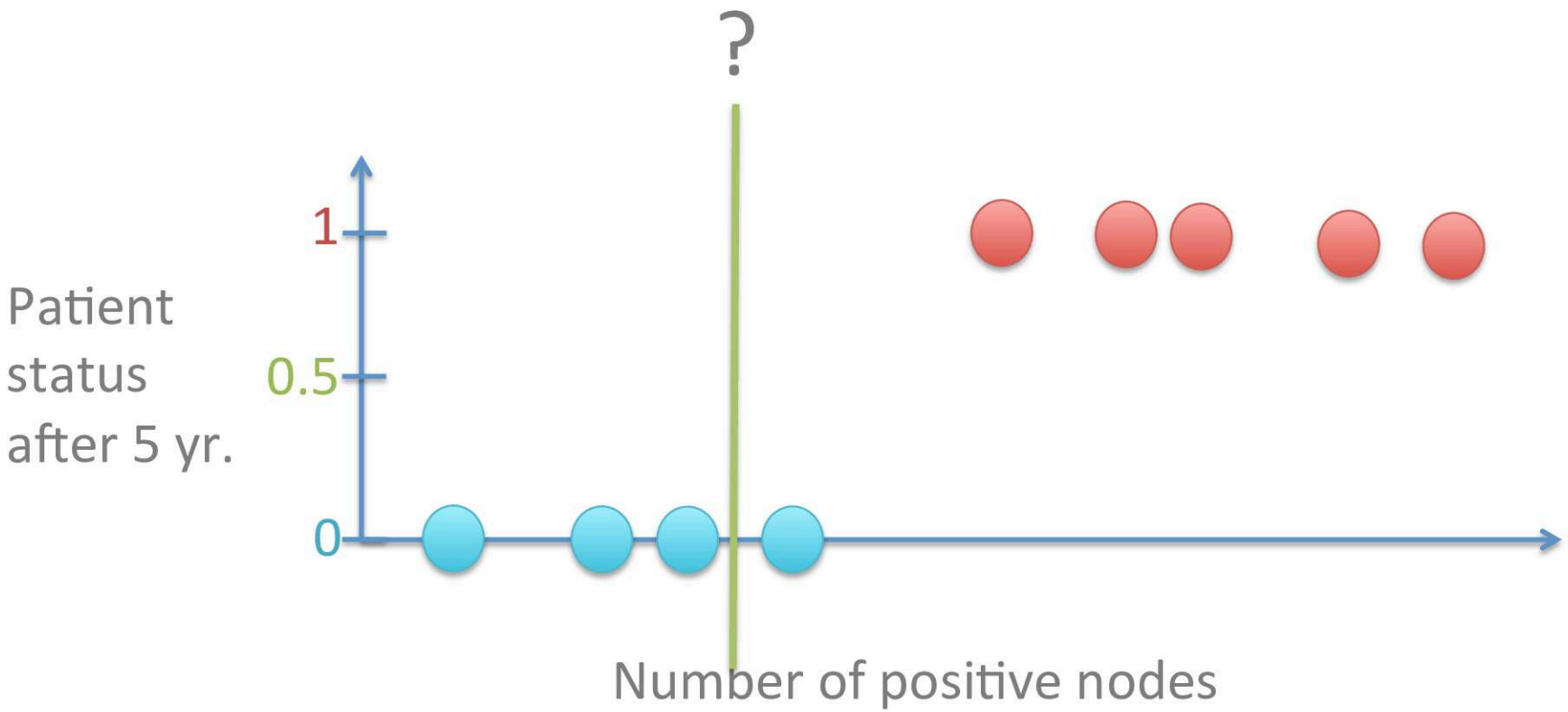
Find the best boundary that separates two classes

Support Vector Machine (SVM)



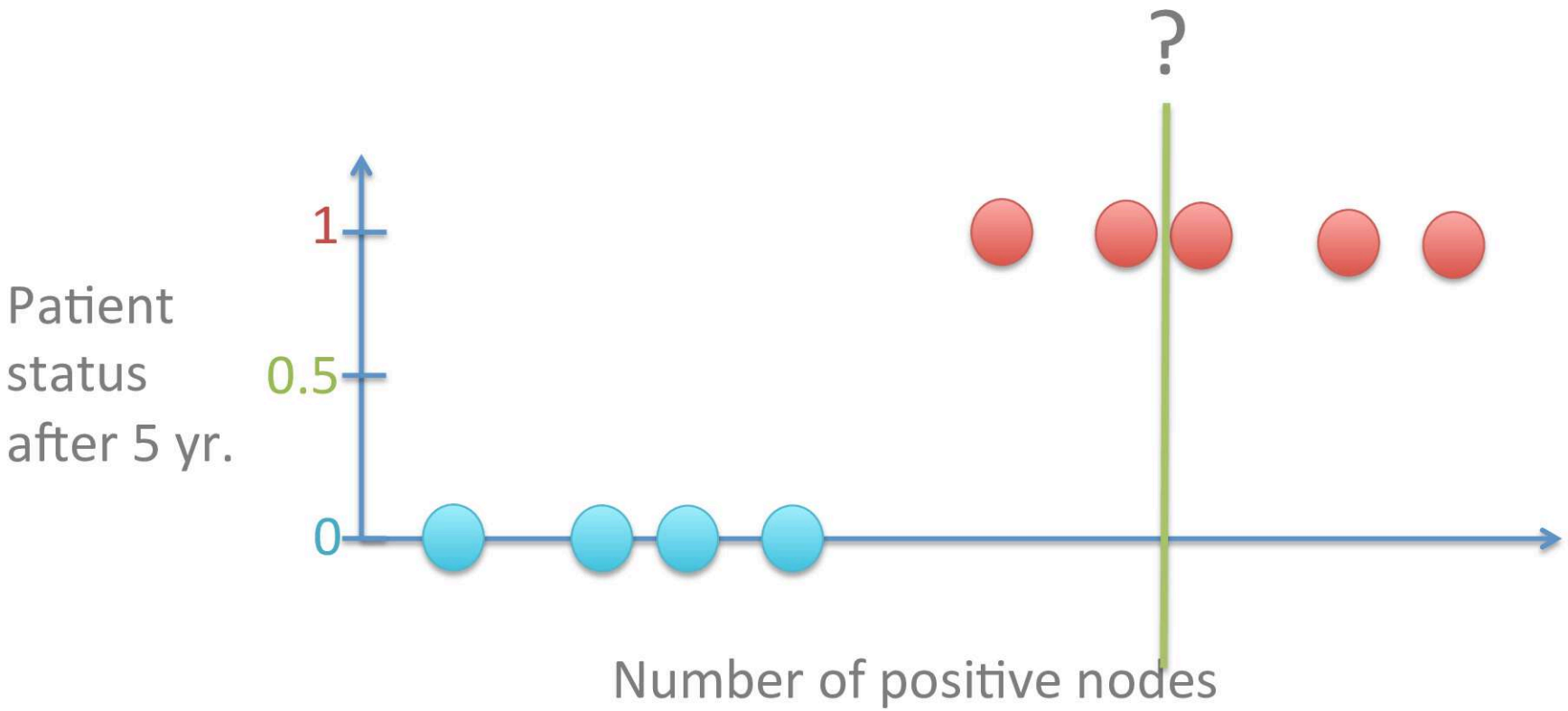
Bad: 3 misclassifications, accuracy 67%

Support Vector Machine (SVM)



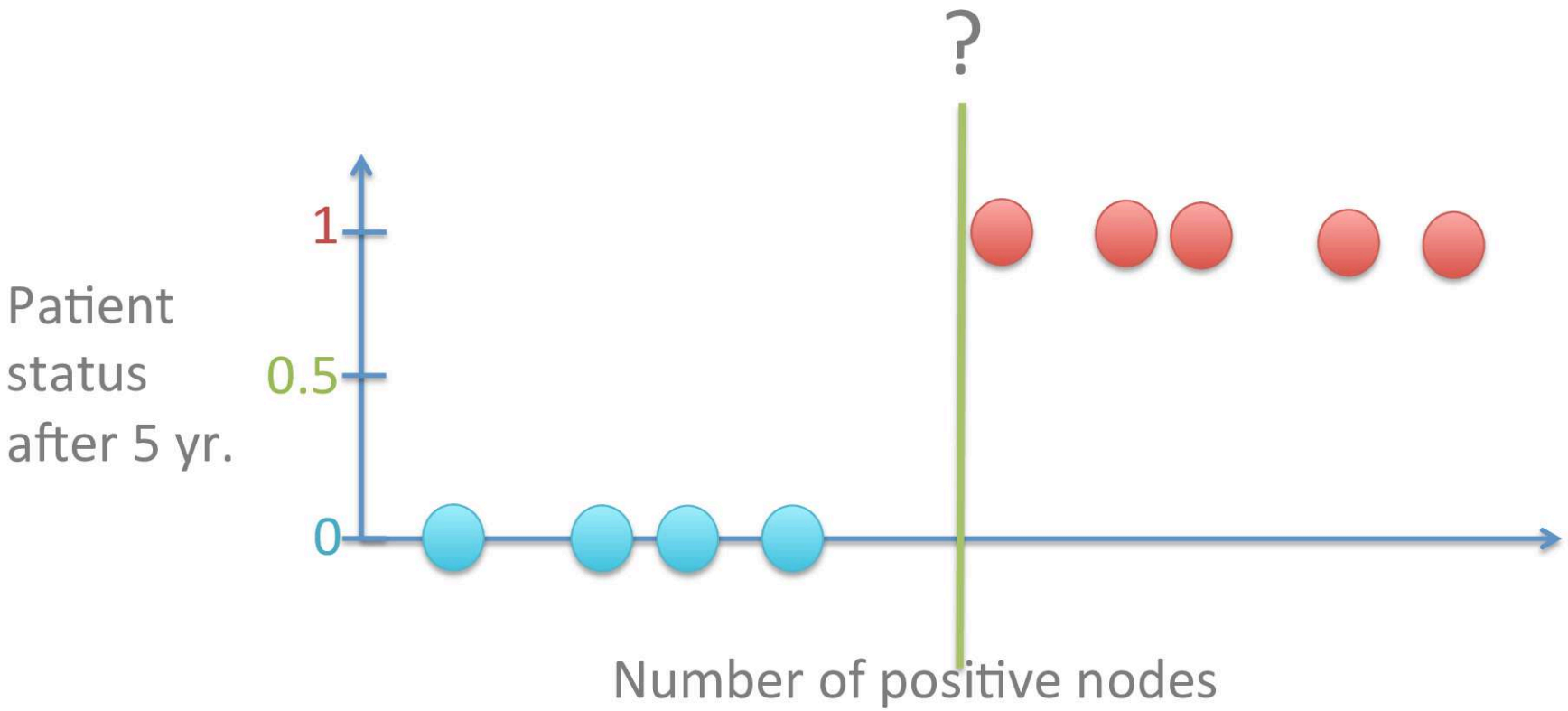
One misclassification, accuracy 89%

Support Vector Machine (SVM)



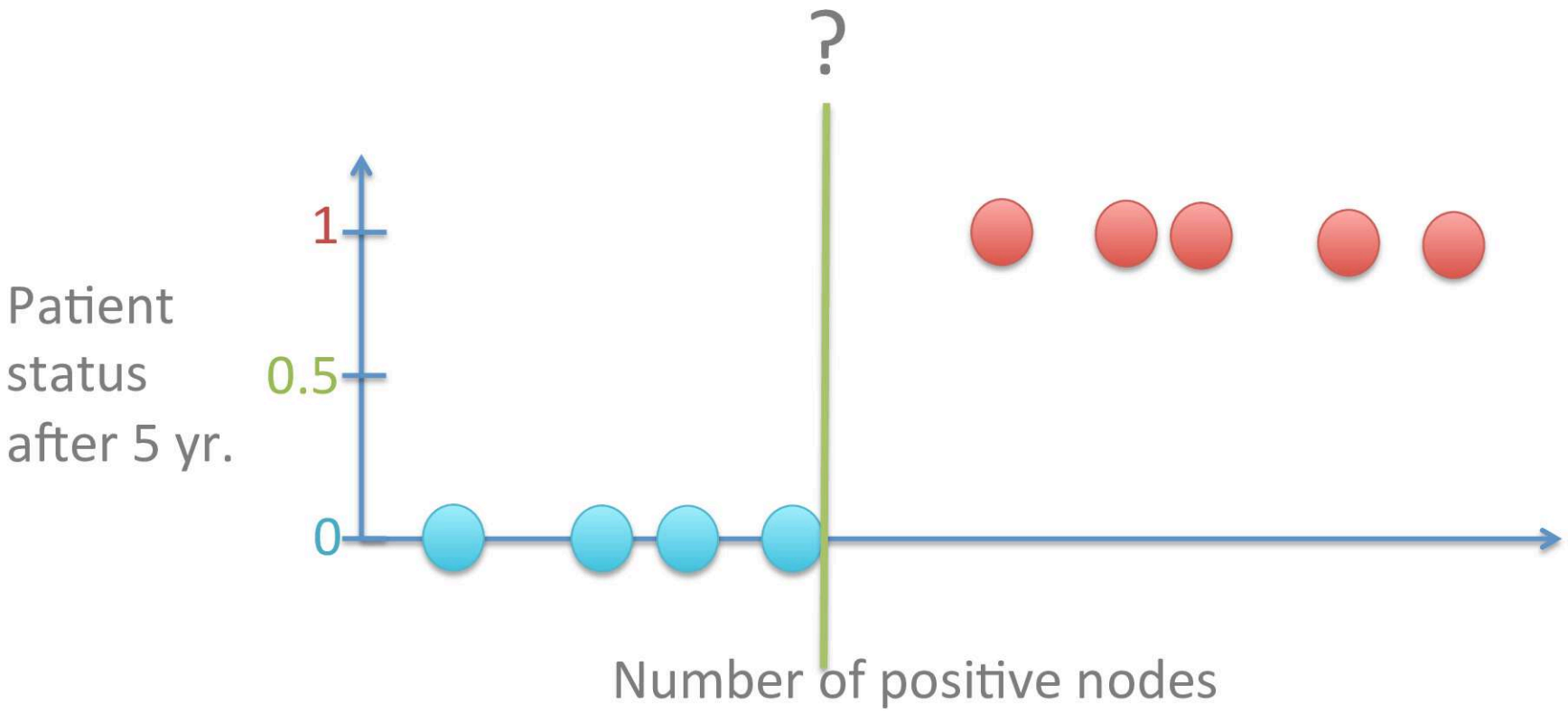
Accuracy: 78%

Support Vector Machine (SVM)



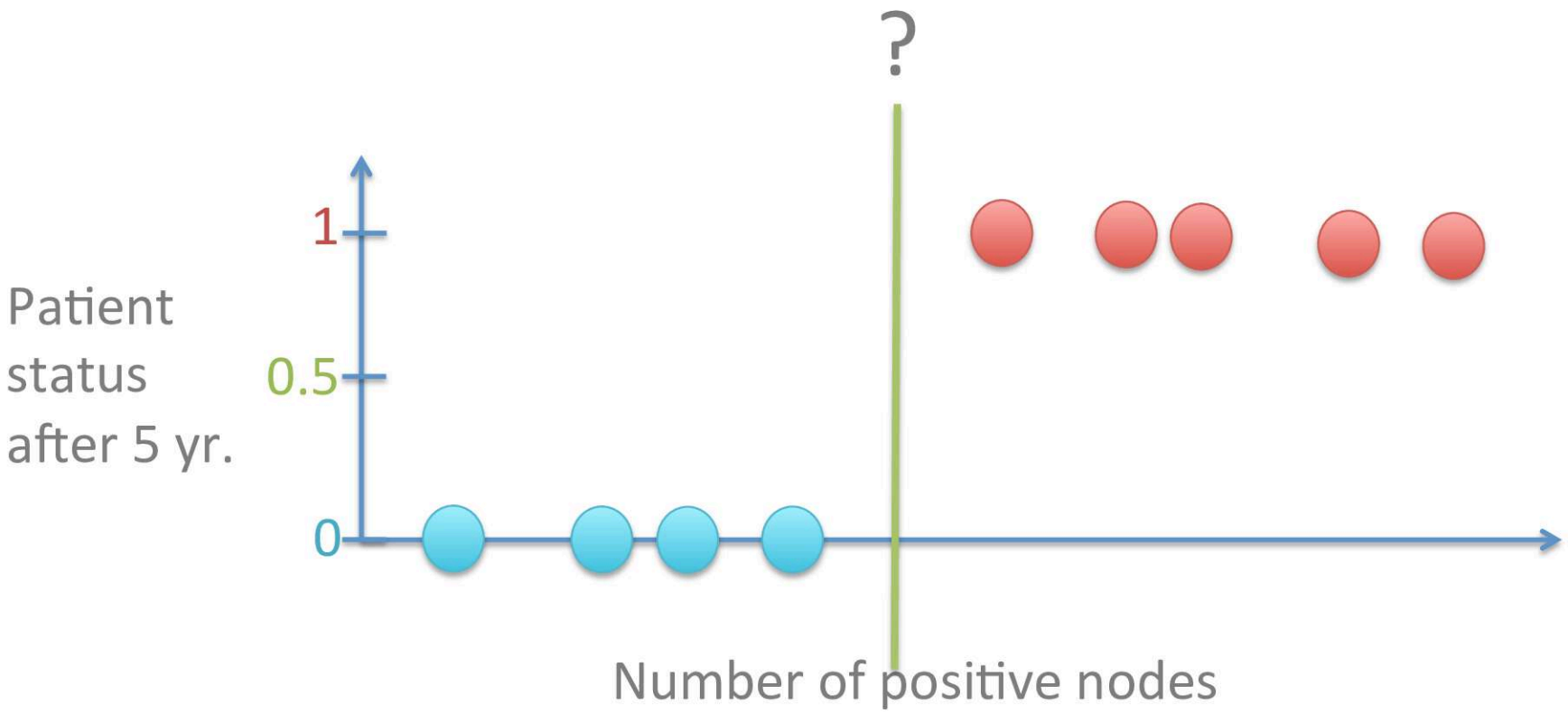
Accuracy: 100%

Support Vector Machine (SVM)



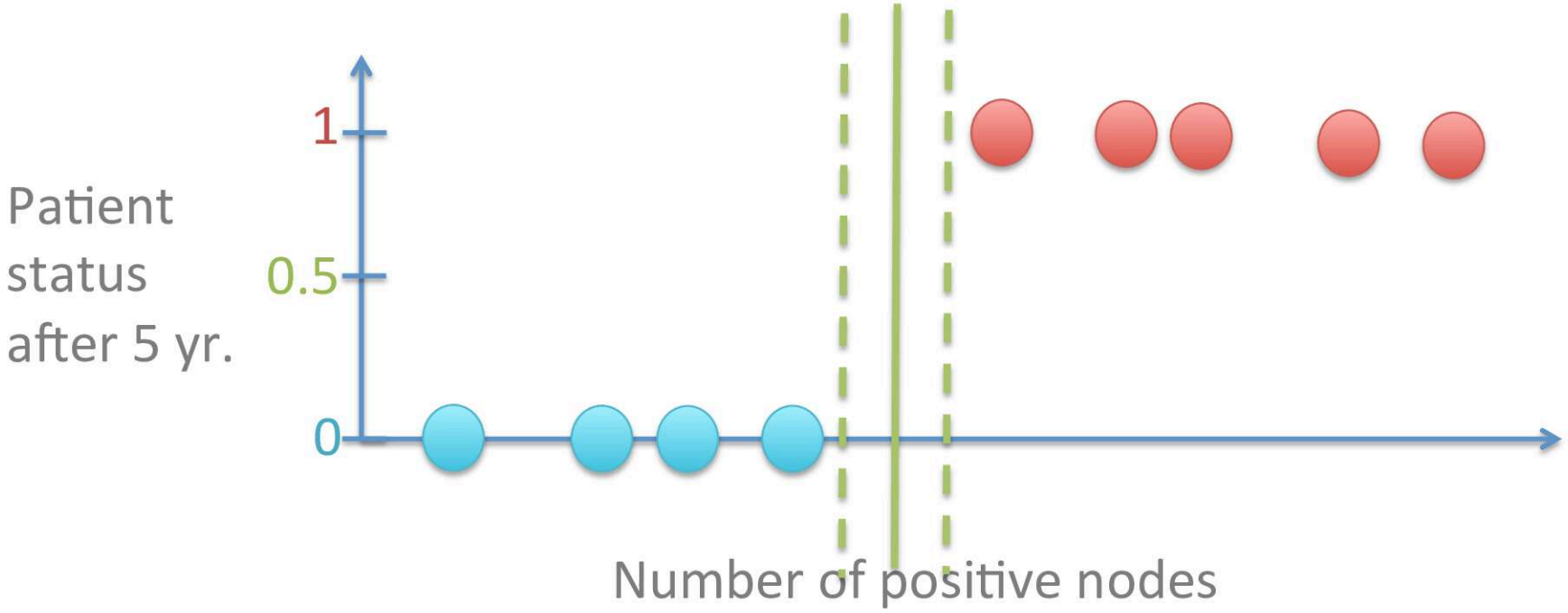
Accuracy: 100%

Support Vector Machine (SVM)



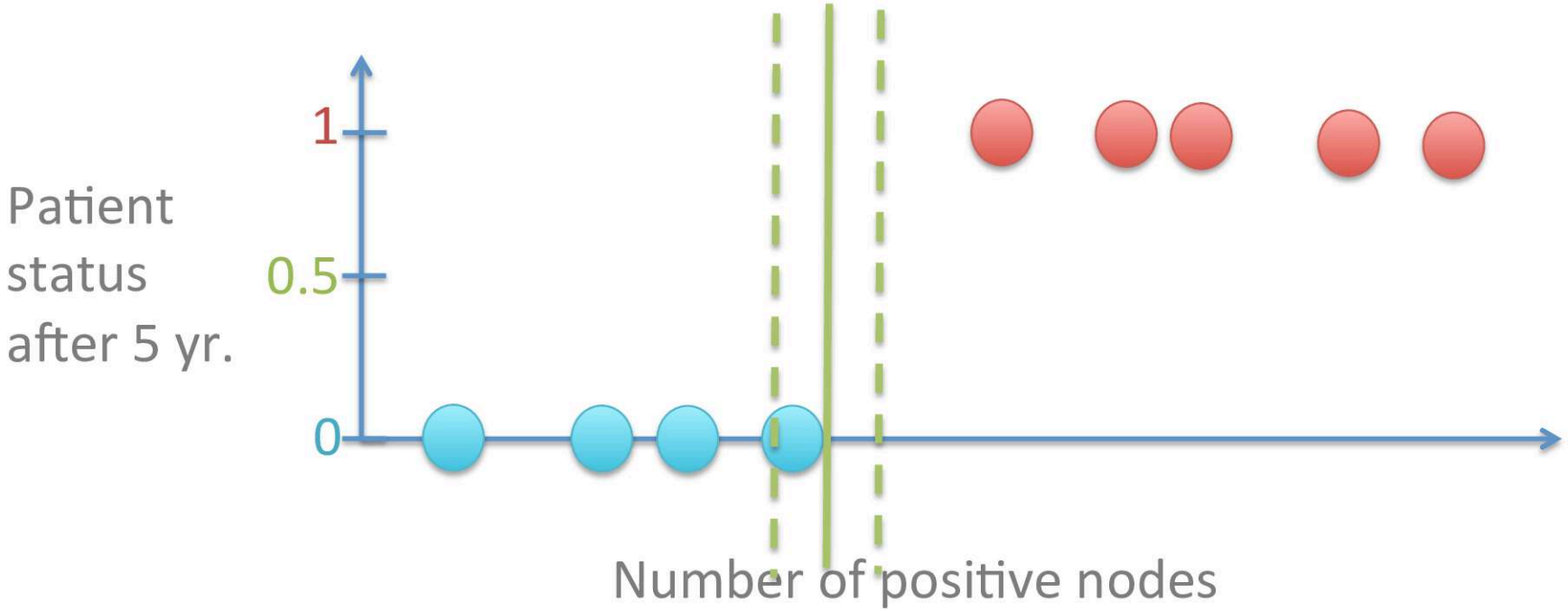
Accuracy: 100%

Support Vector Machine (SVM)



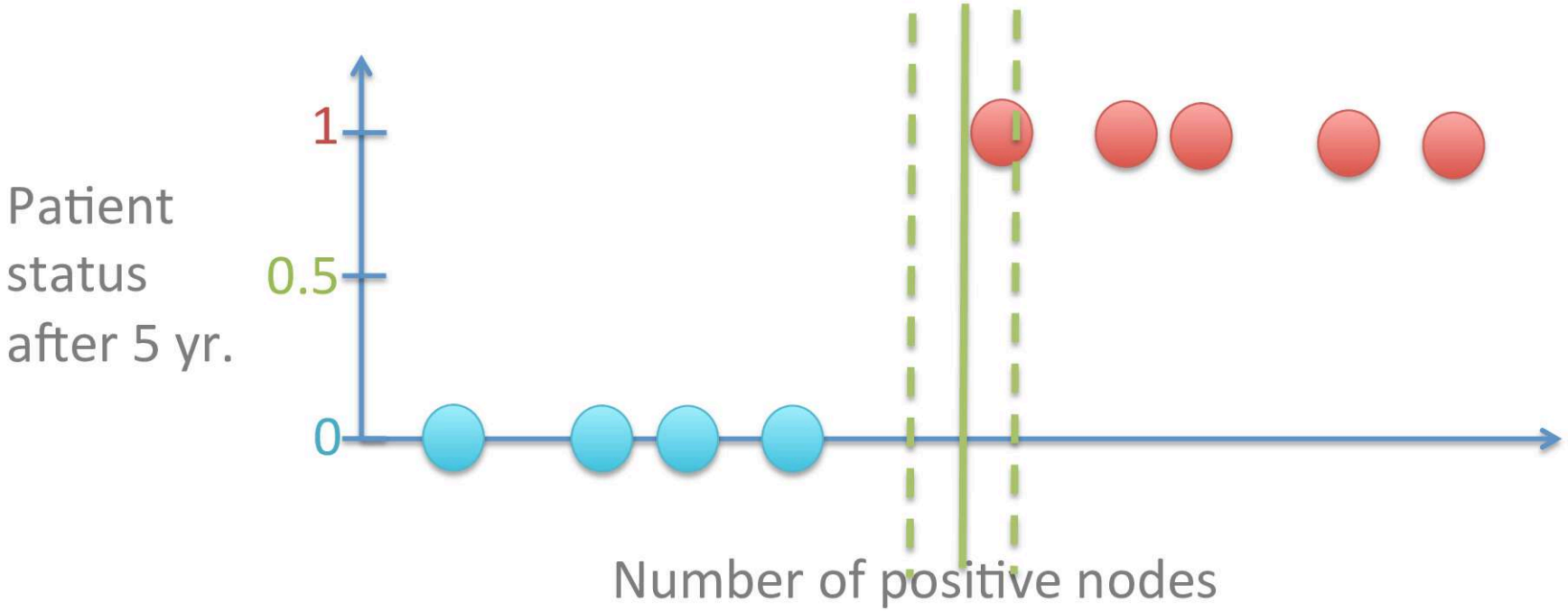
The margin: No man's land

Support Vector Machine (SVM)



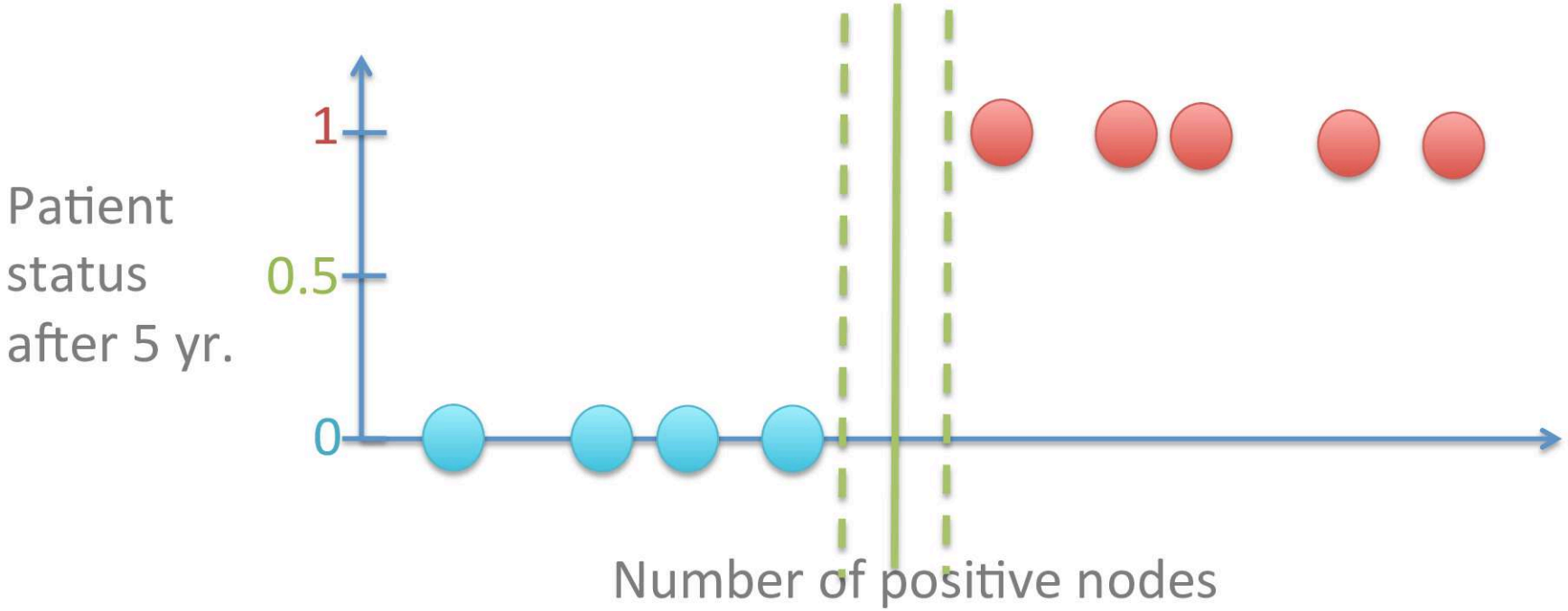
Extra error due to point in margin

Support Vector Machine (SVM)



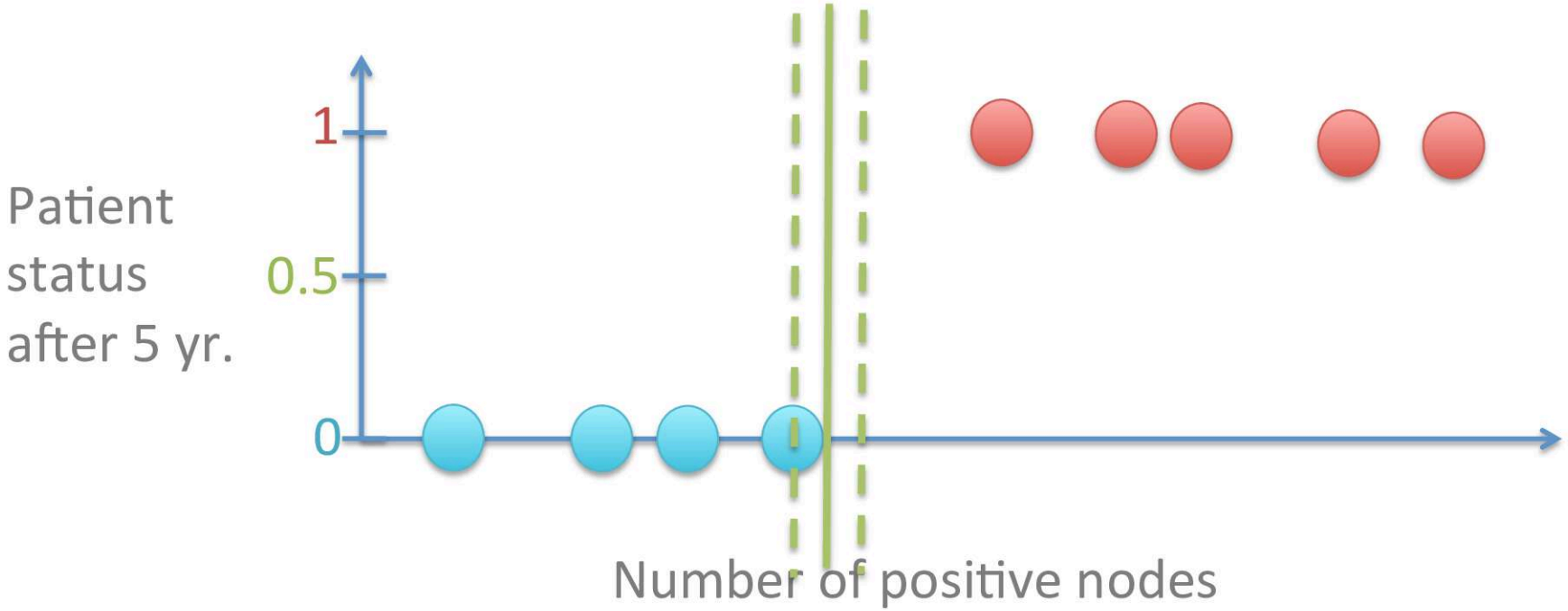
Extra error due to point in margin

Support Vector Machine (SVM)



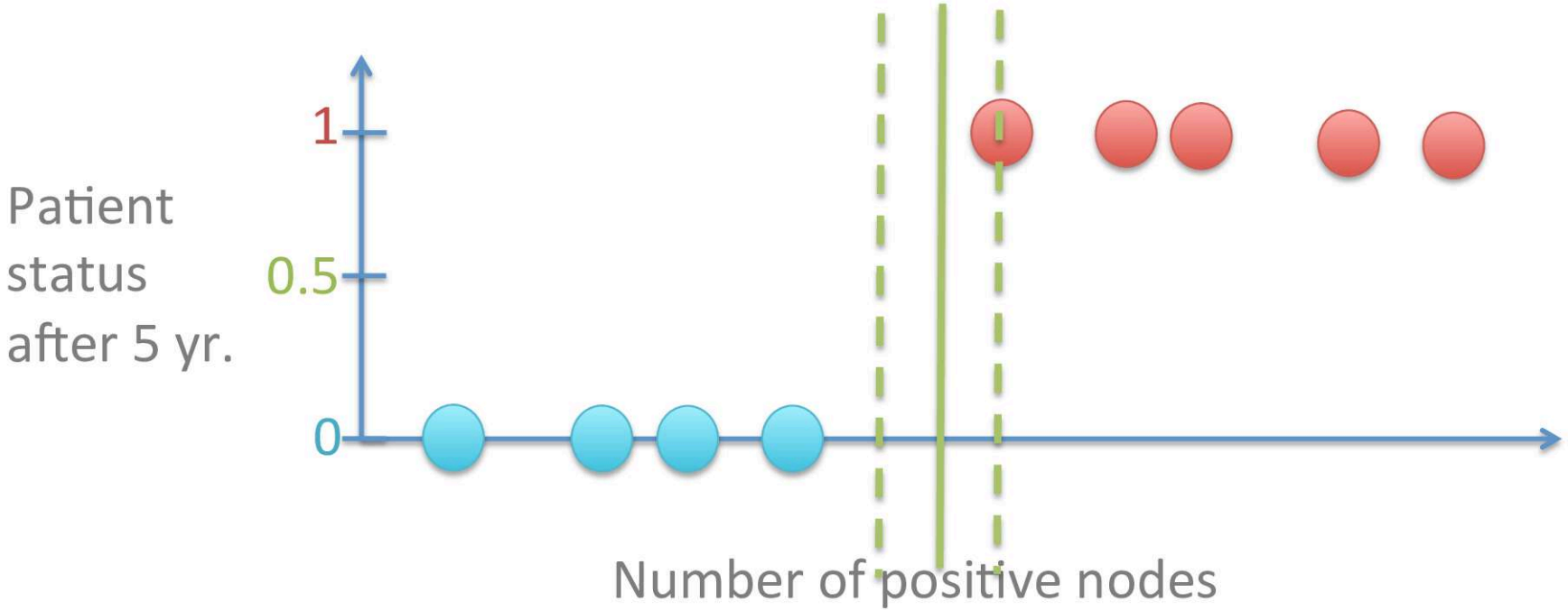
Best boundary

Support Vector Machine (SVM)



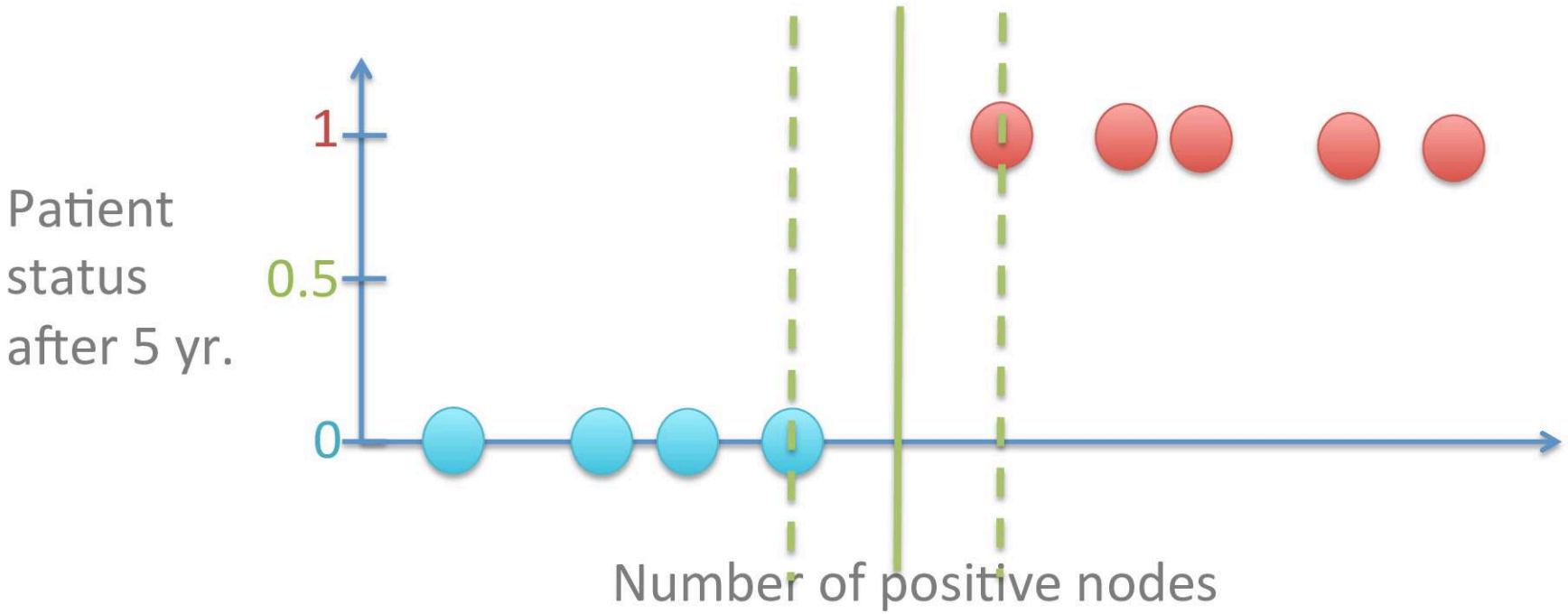
An even better way of doing this:
Find the boundary with the largest margin

Support Vector Machine (SVM)



Extra error due to point in margin

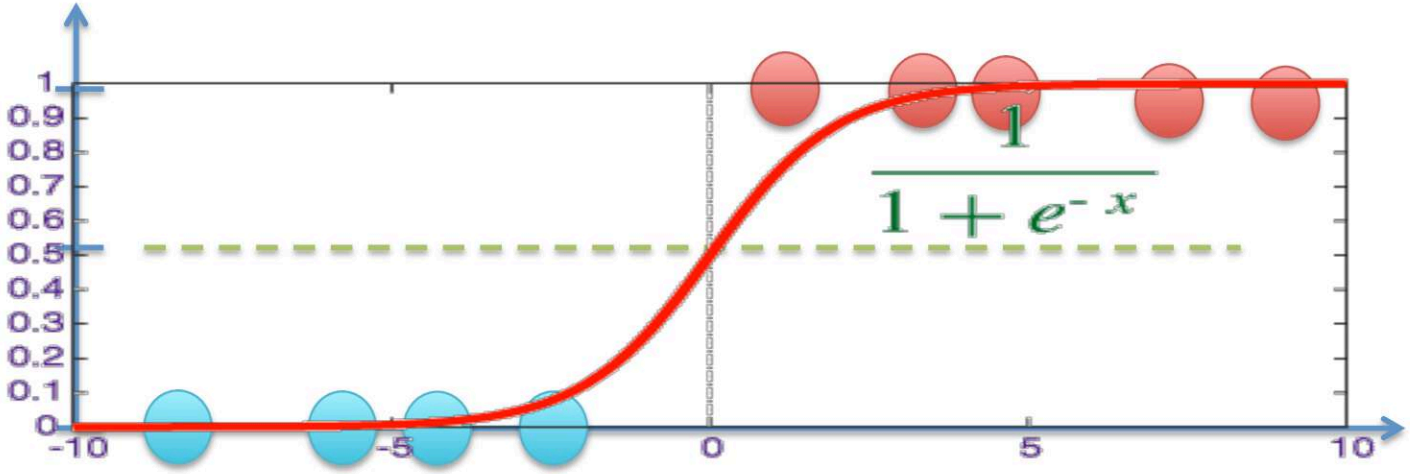
Support Vector Machine (SVM)



Best boundary

Logistic Regression

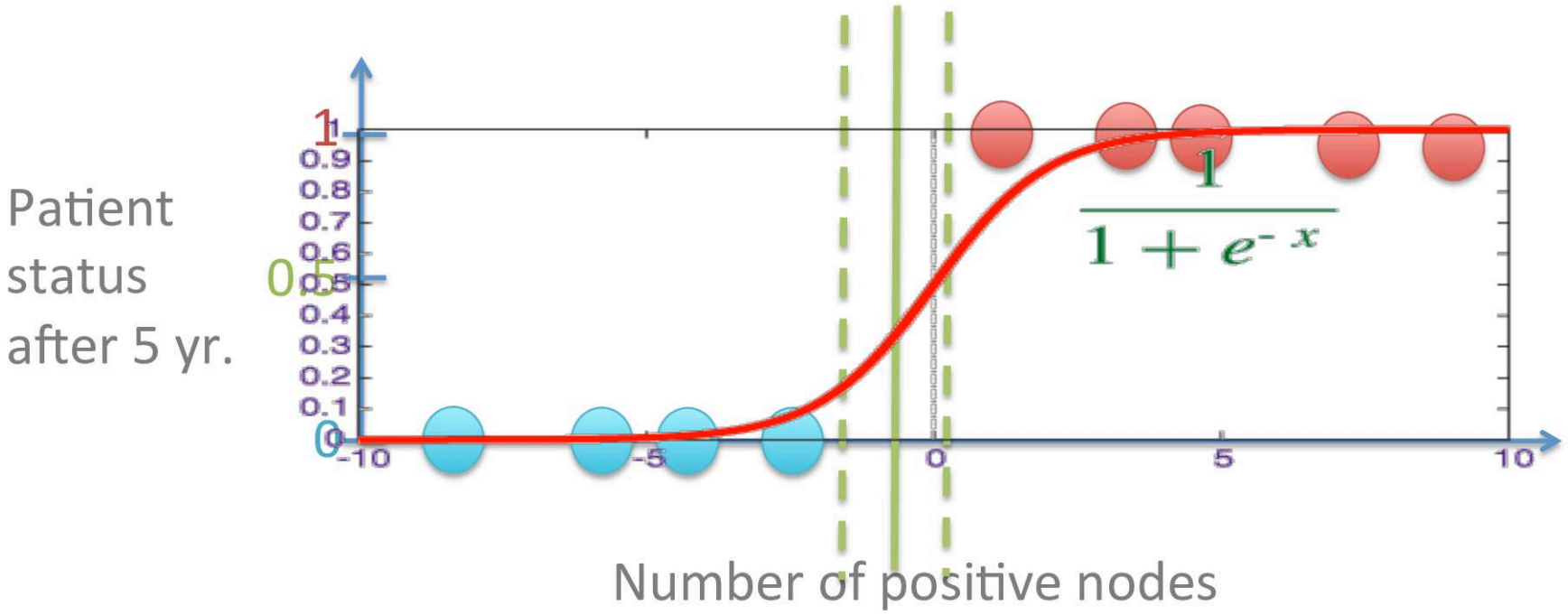
Patient status after 5 yr.



Number of positive nodes

$$y_\beta(x) = \frac{1}{1+e^{-(\beta_0+\beta_1 x+\varepsilon)}}$$

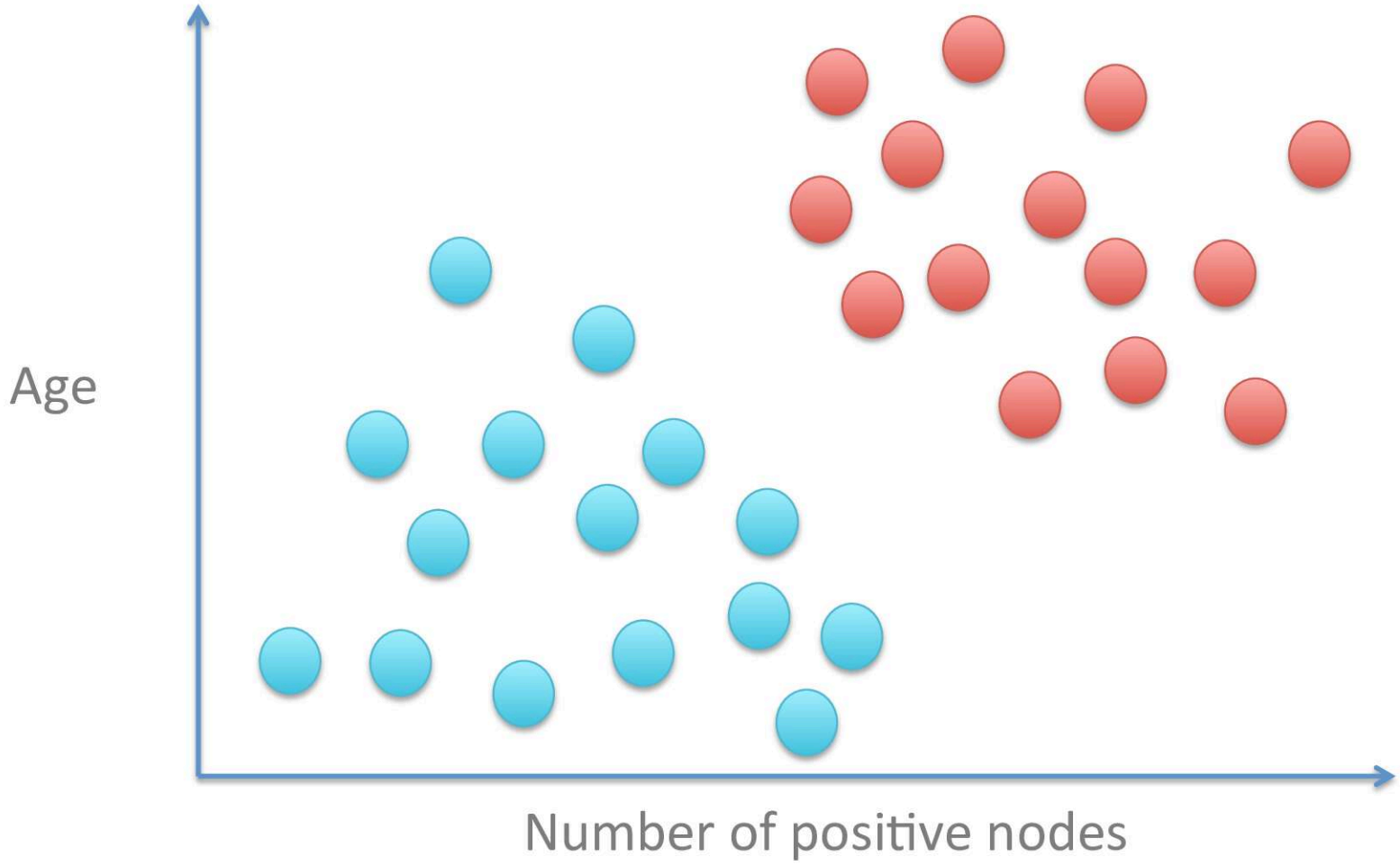
Logistic Regression and SVM are not very different



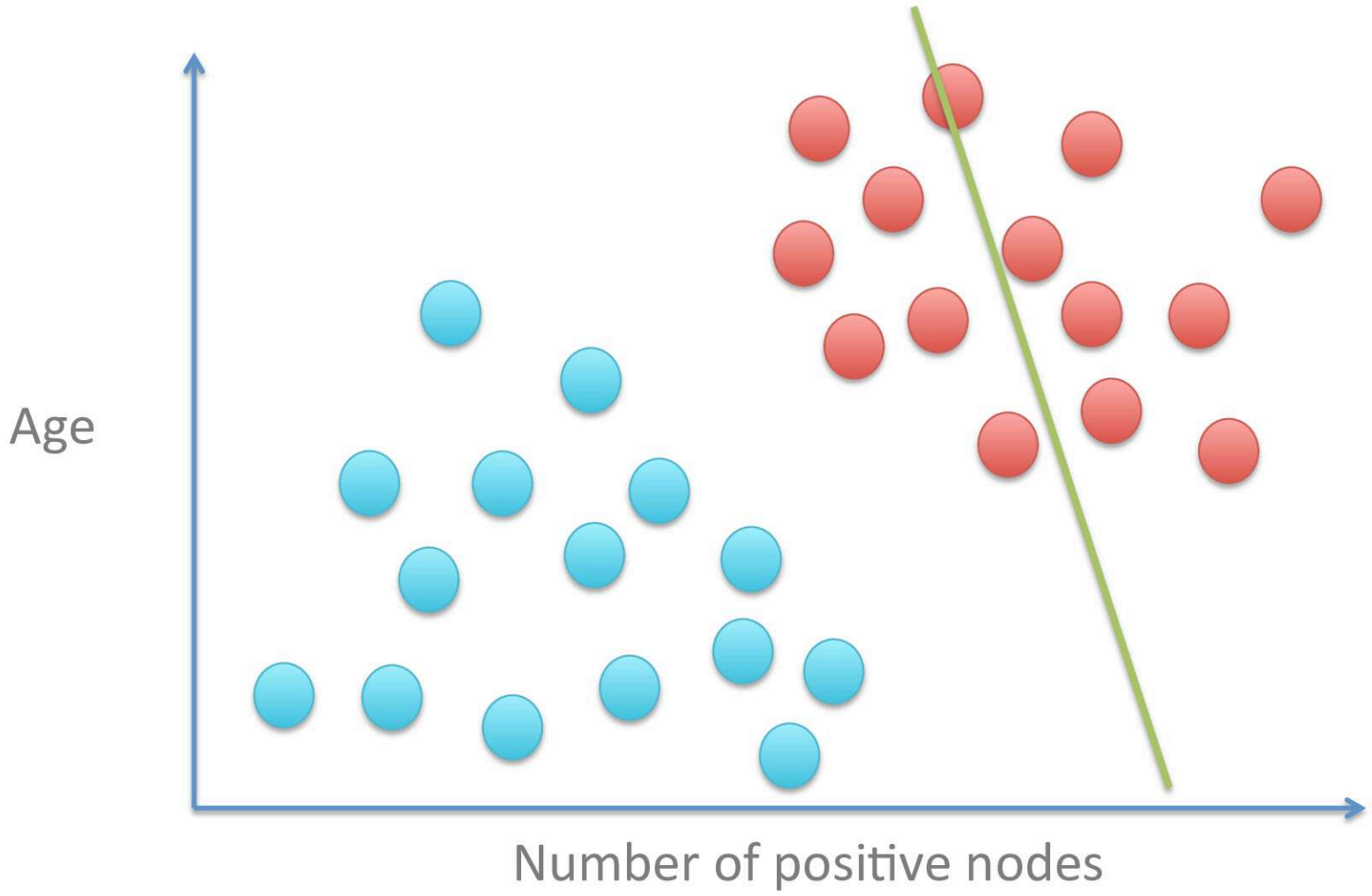
Best boundary

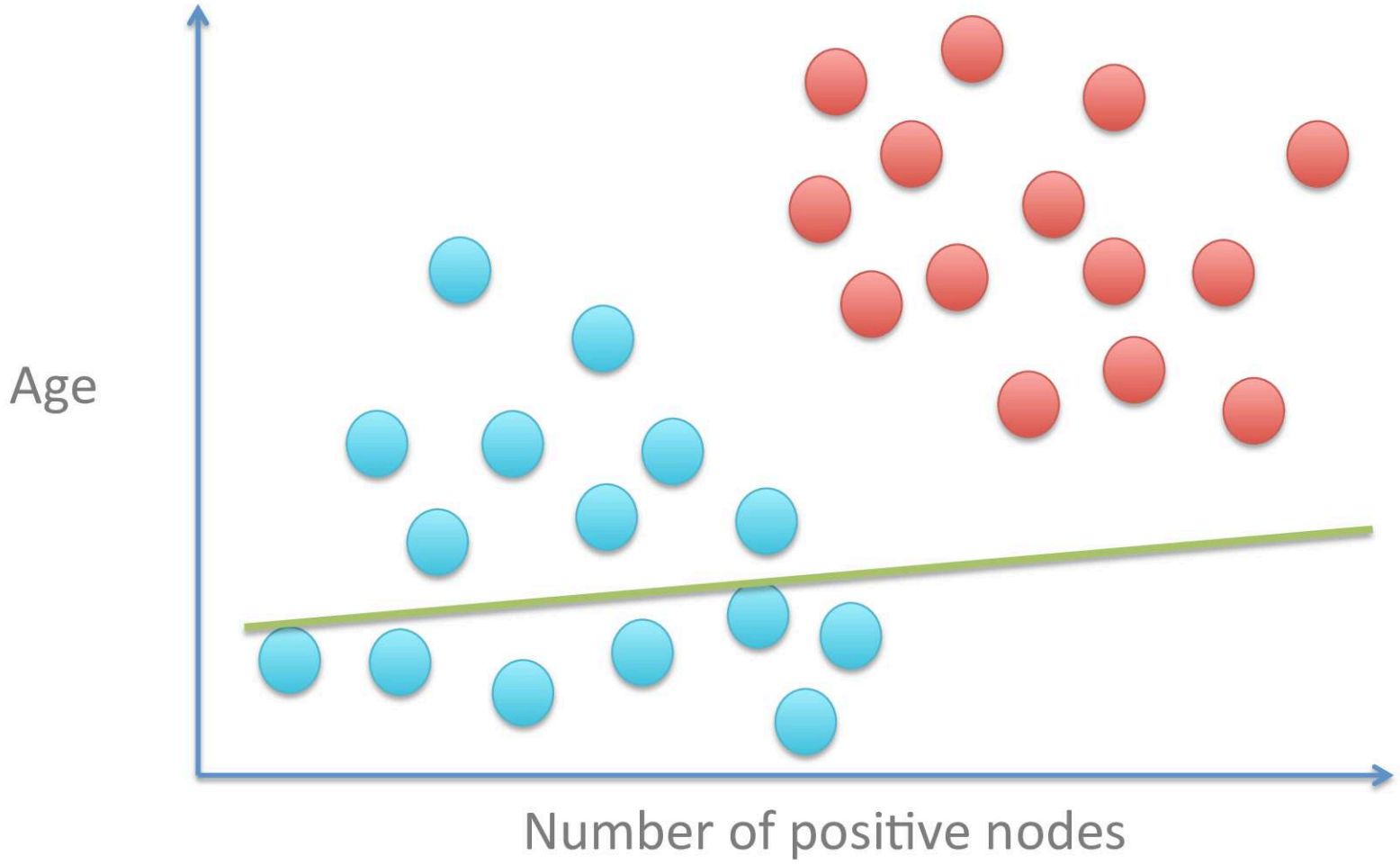
2 Features: Number of + nodes, Age

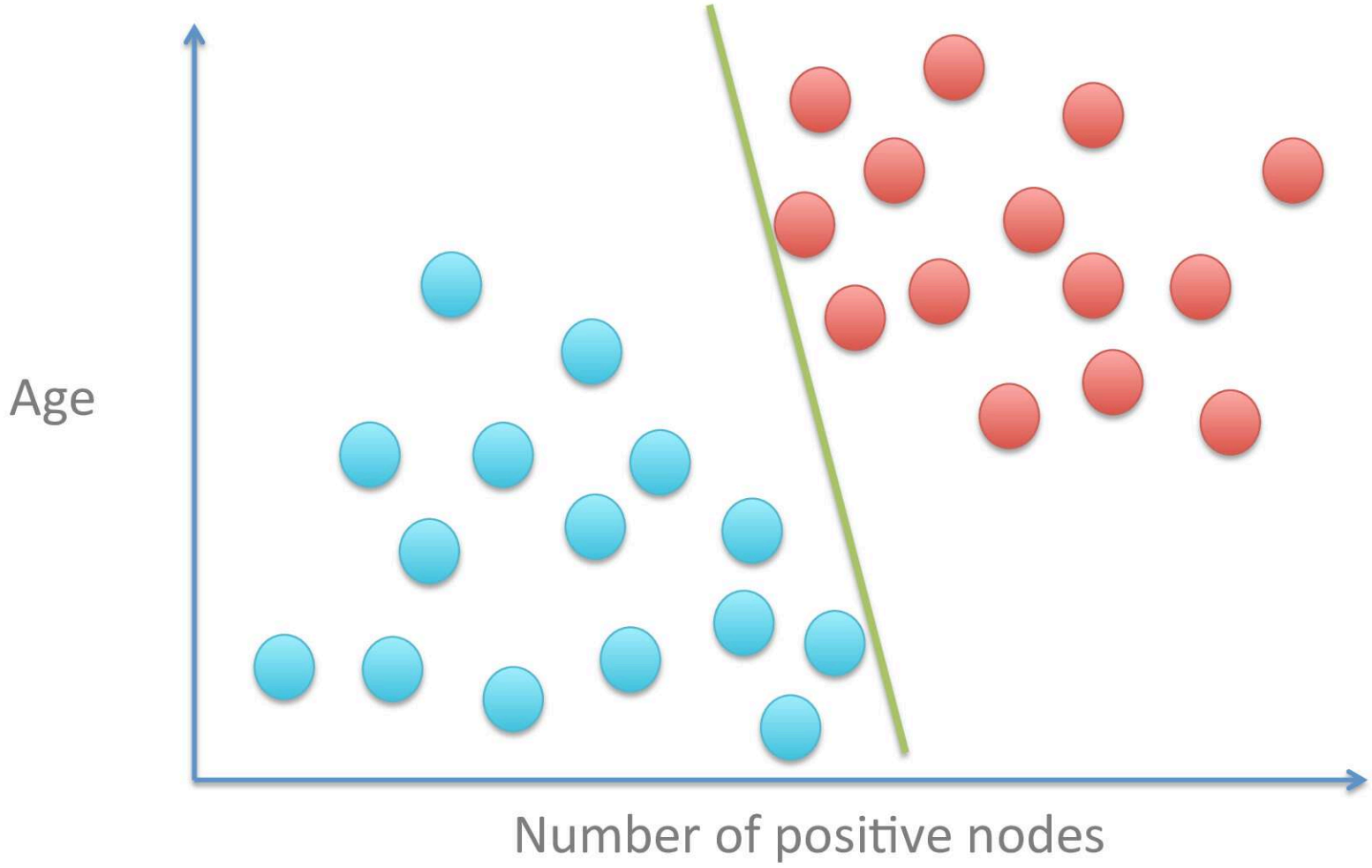
2 Labels: Survived / Lost

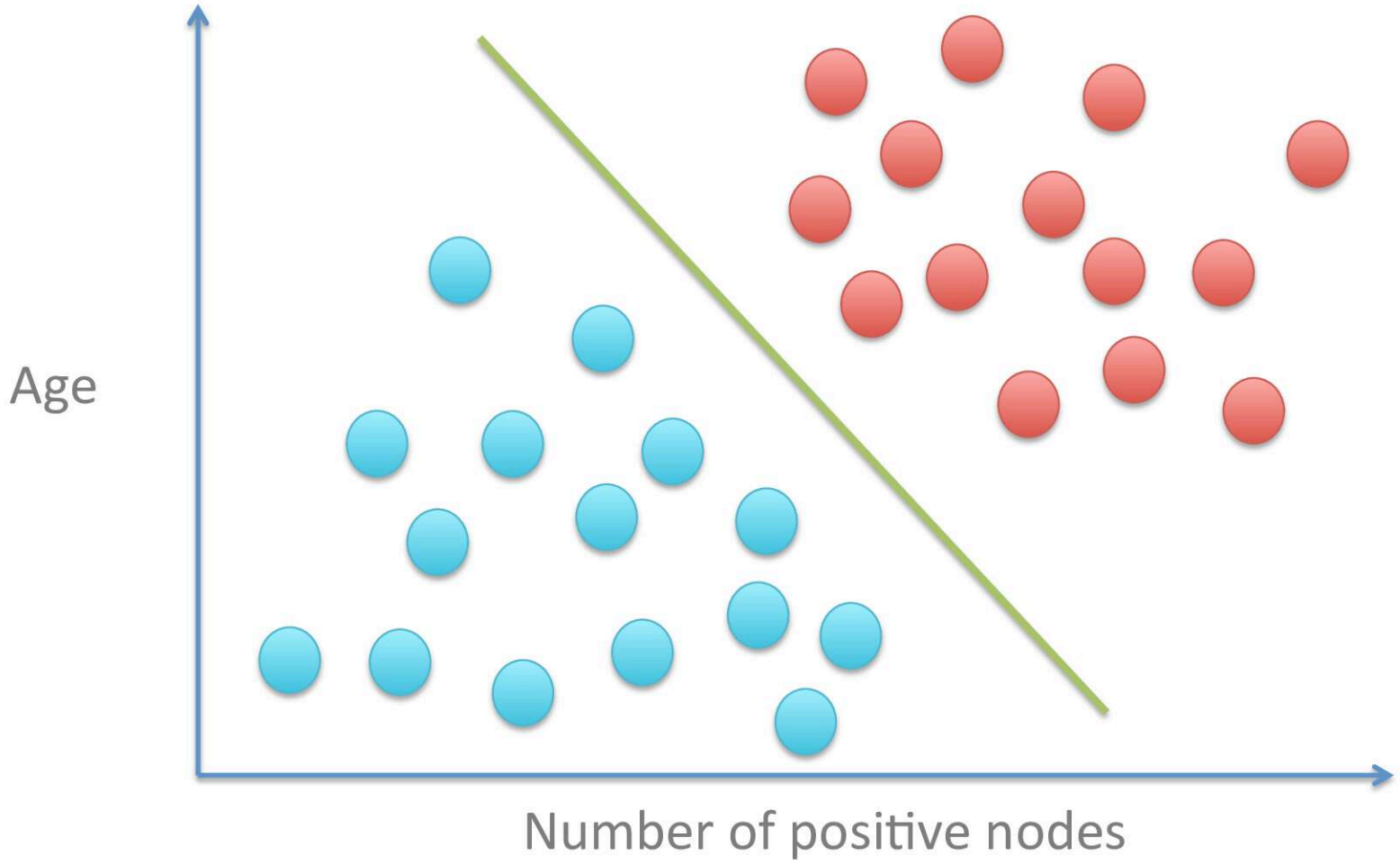


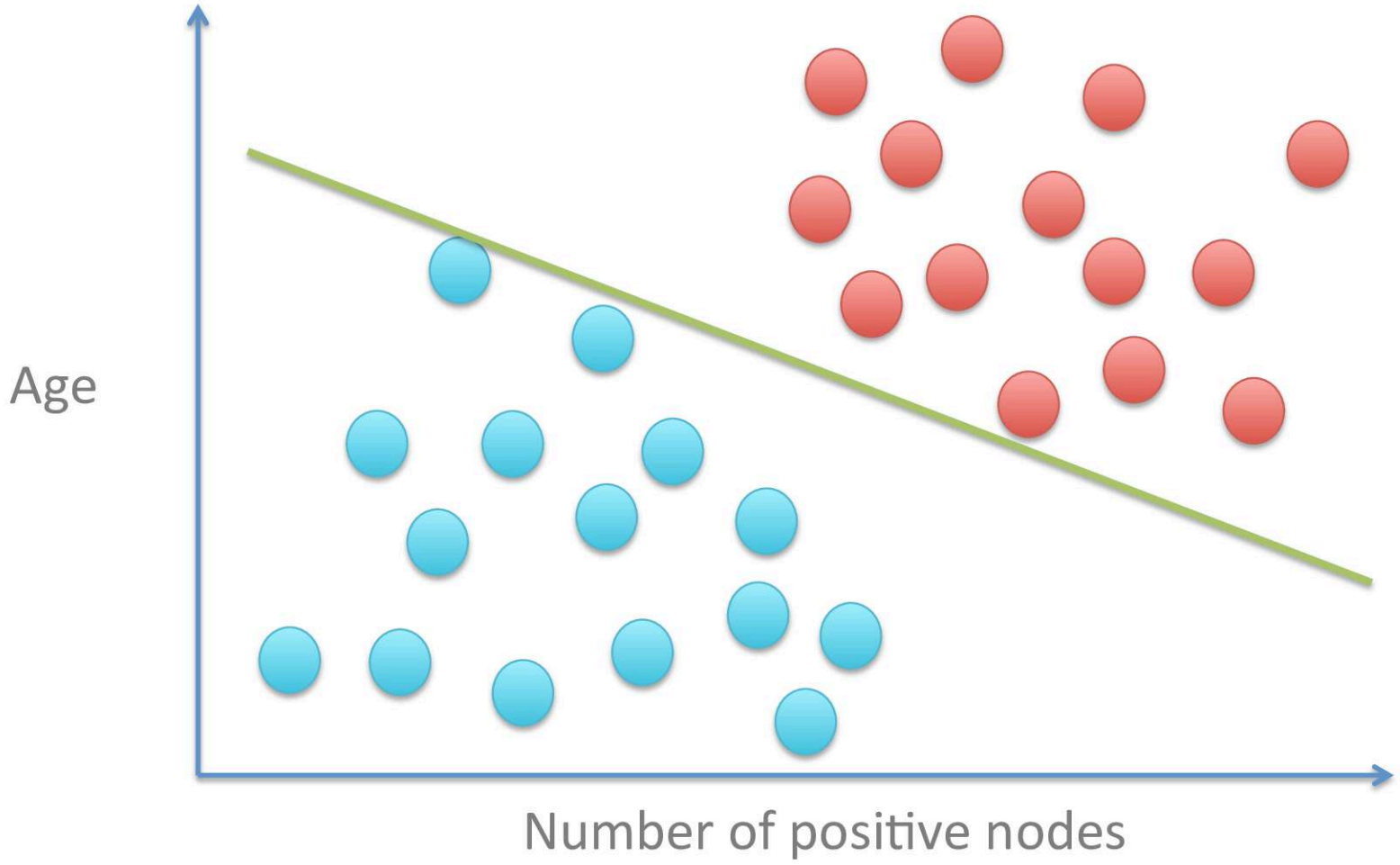
Find the line that separates the classes best

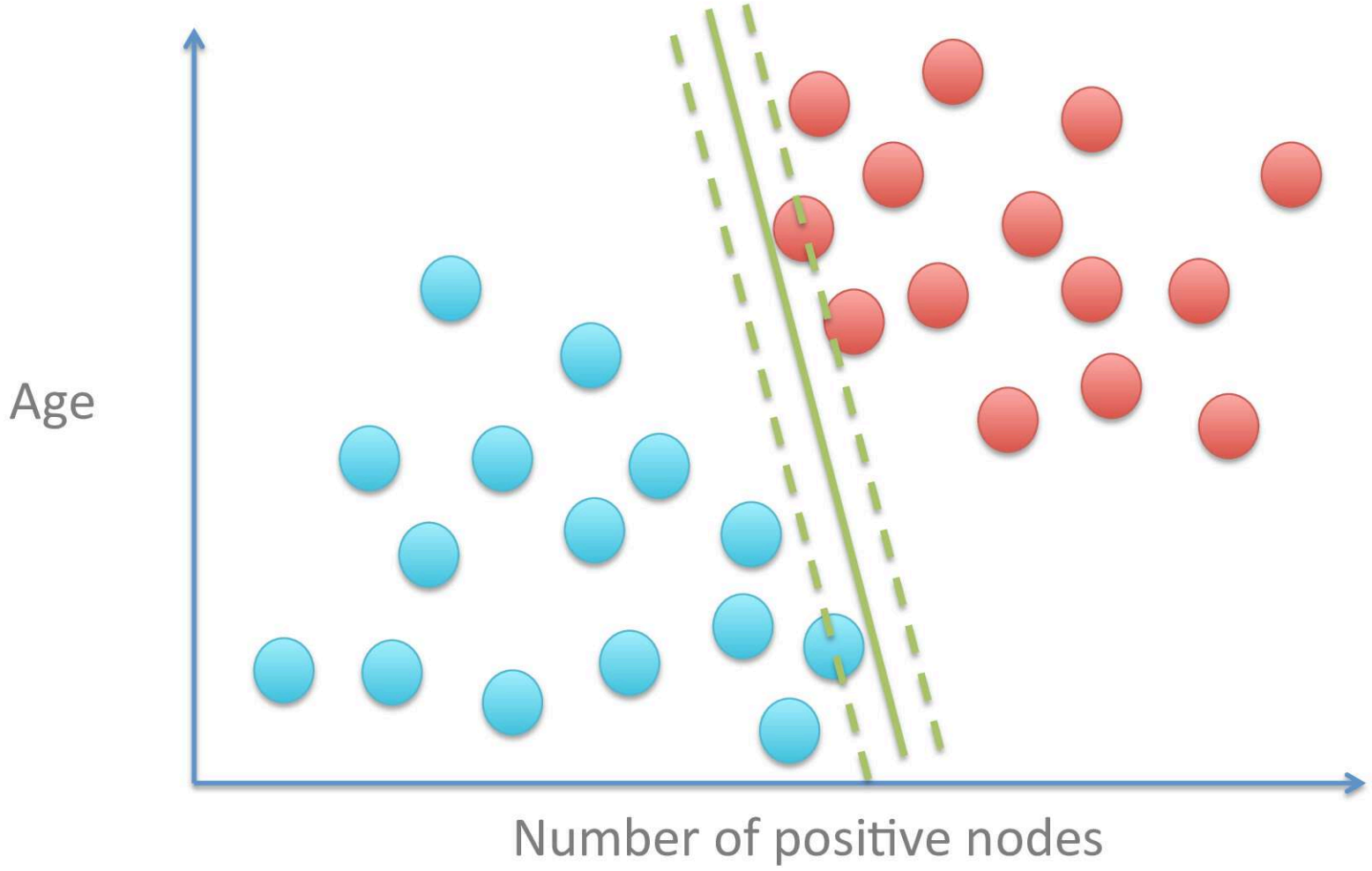


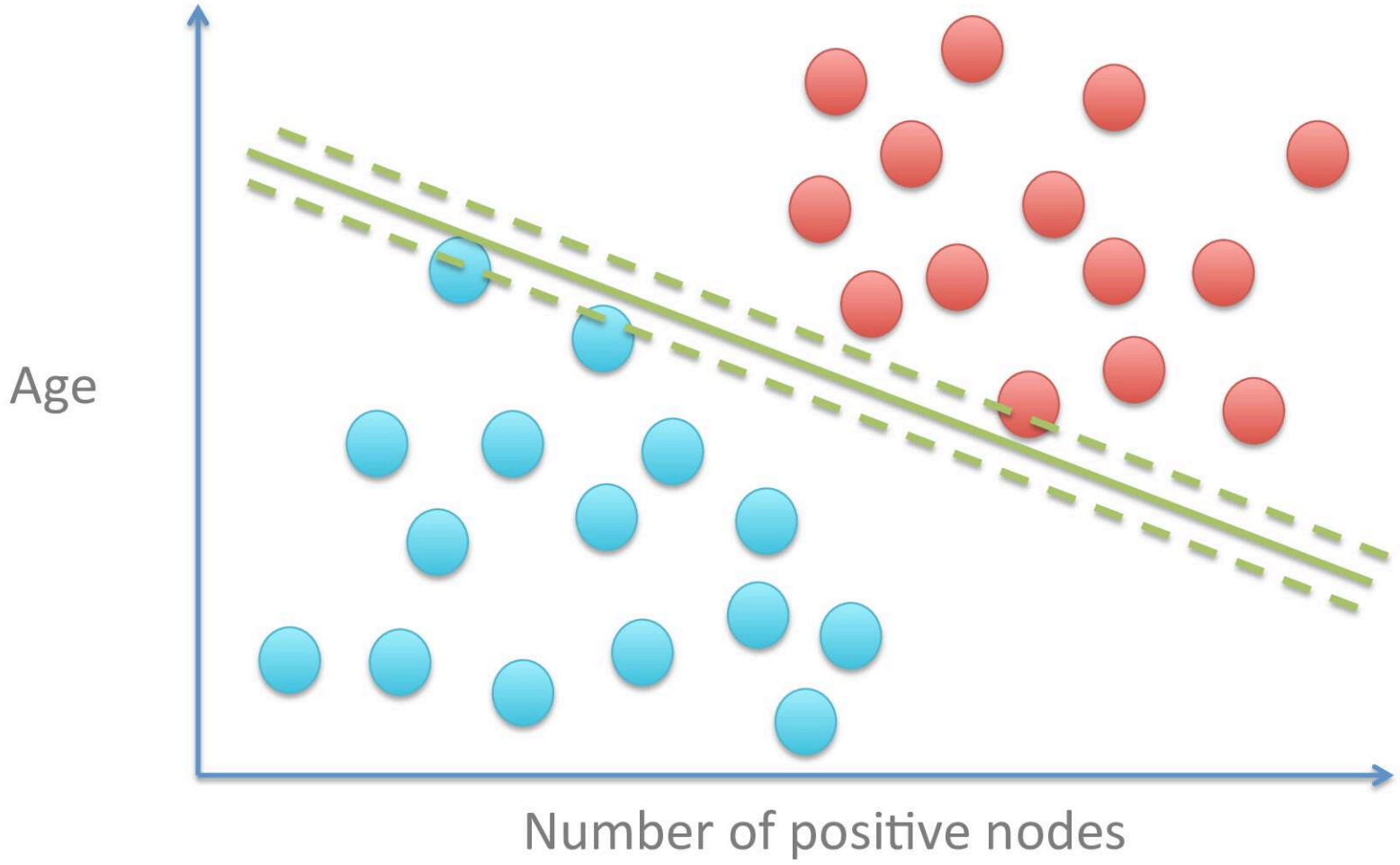

Age
Number of positive nodes


Age
Number of positive nodes

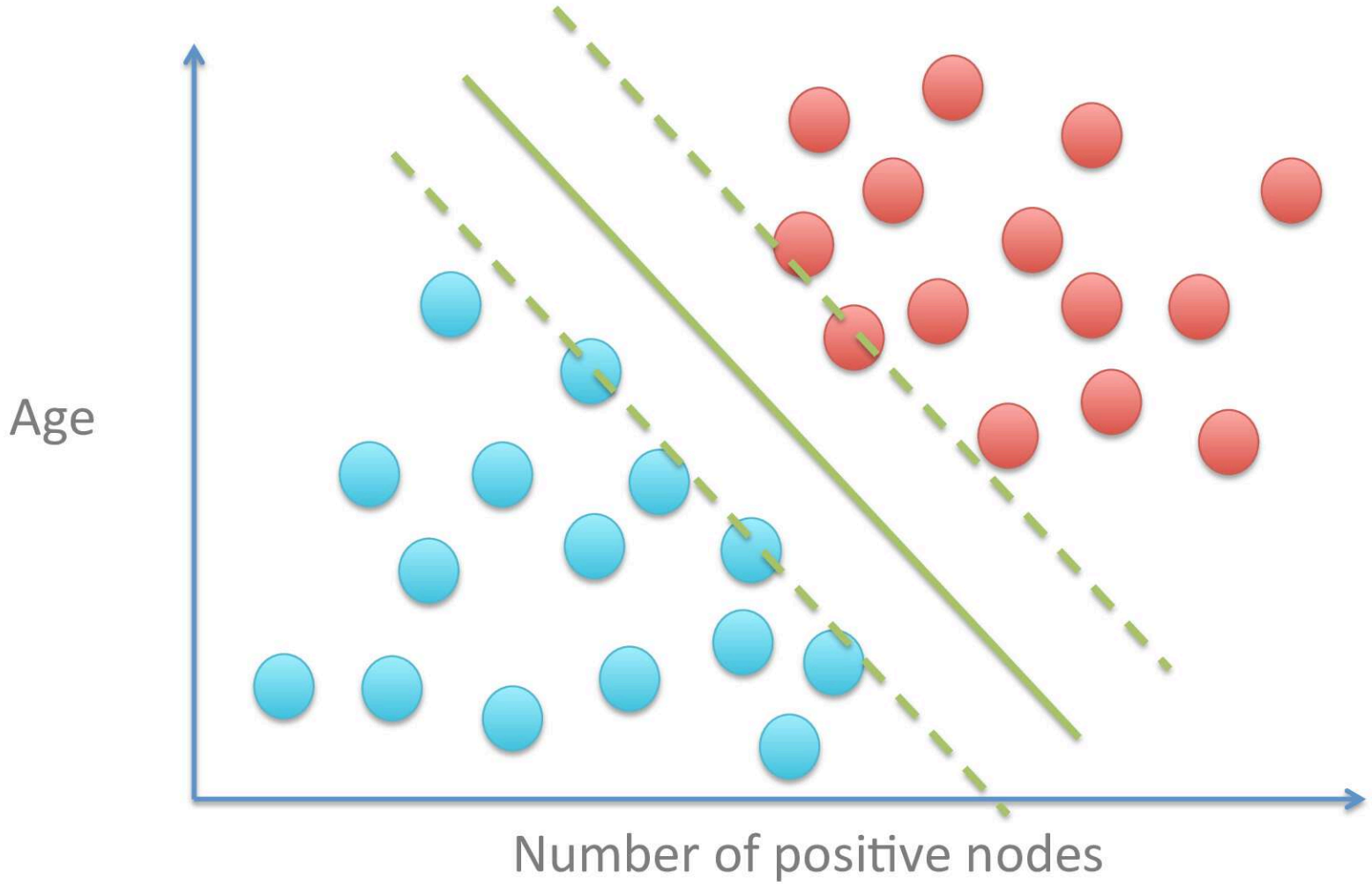

Age
Number of positive nodes


Age
Number of positive nodes


Age
Number of positive nodes

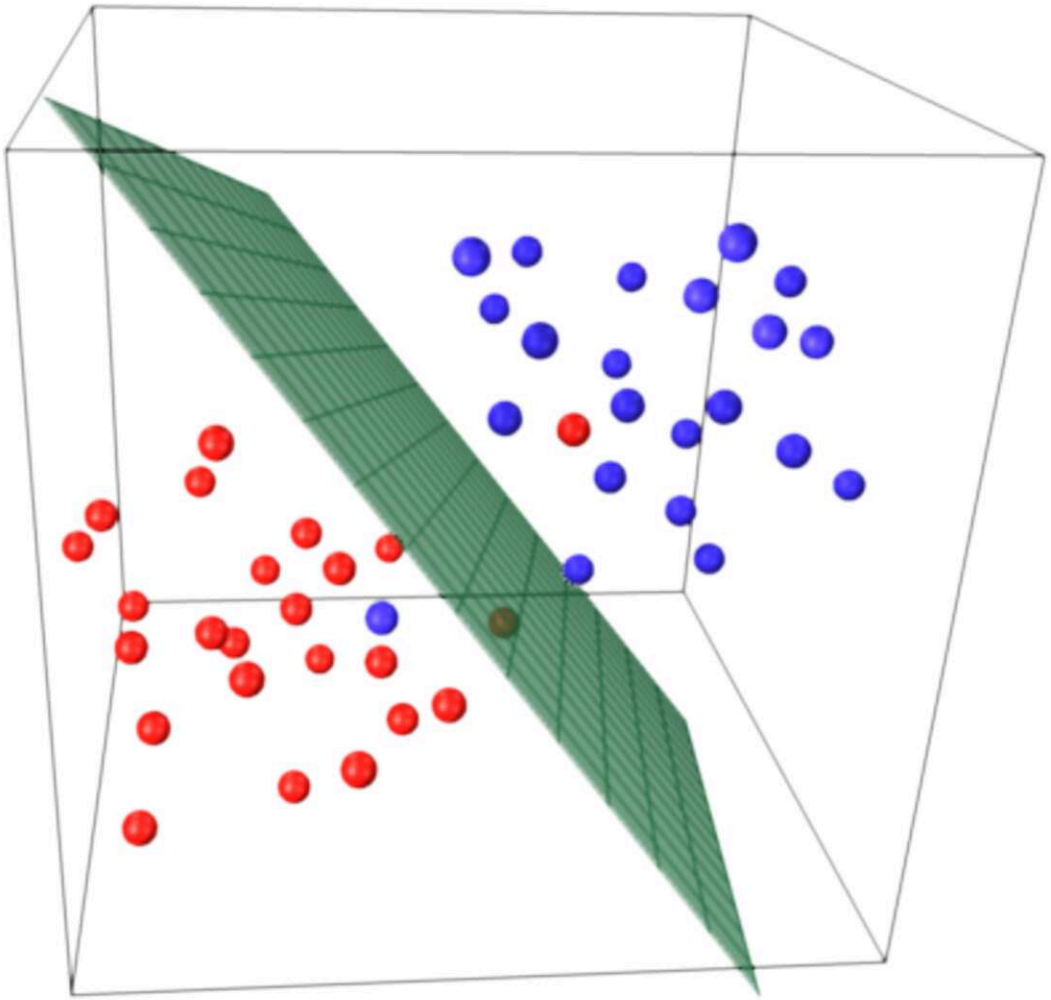

Age
Number of positive nodes

Best boundary



3 features: Find the best boundary plane

(More features: hyperplane)



Logistic Regression Cost Function

$$y_{pred}(x) = \frac{1}{1+e^{-z}} = \frac{1}{1+e^{-(\beta_0+\beta_1 x+\ldots+\varepsilon)}}$$

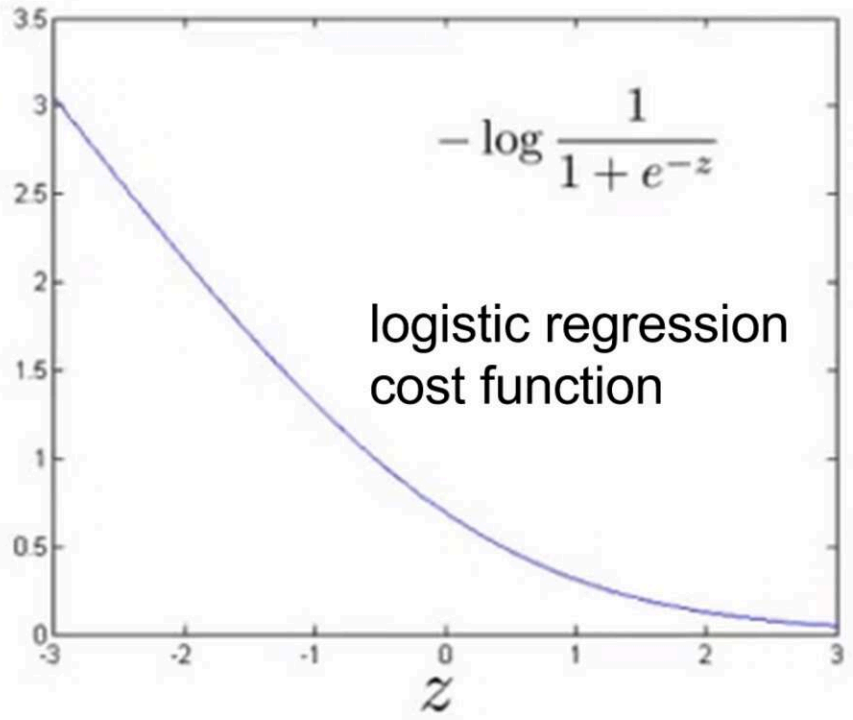
Logistic Regression Cost Function

$$y_{pred}(x) = \frac{1}{1+e^{-z}} = \frac{1}{1+e^{-(\beta_0+\beta_1 x+\ldots+\varepsilon)}}$$

$$Cost(y_{pred}, y_{true}) = \begin{cases} -\log(y_{pred}) & if \quad y_{true} = 1 \\ -\log(1-y_{pred}) & if \quad y_{true} = 0 \end{cases}$$

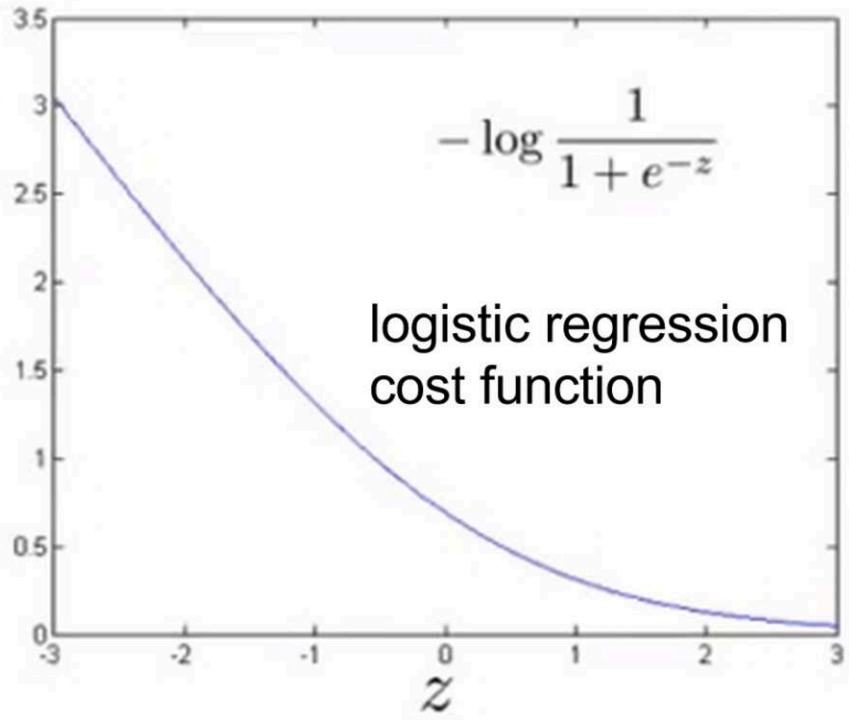
Logistic Regression Cost Function

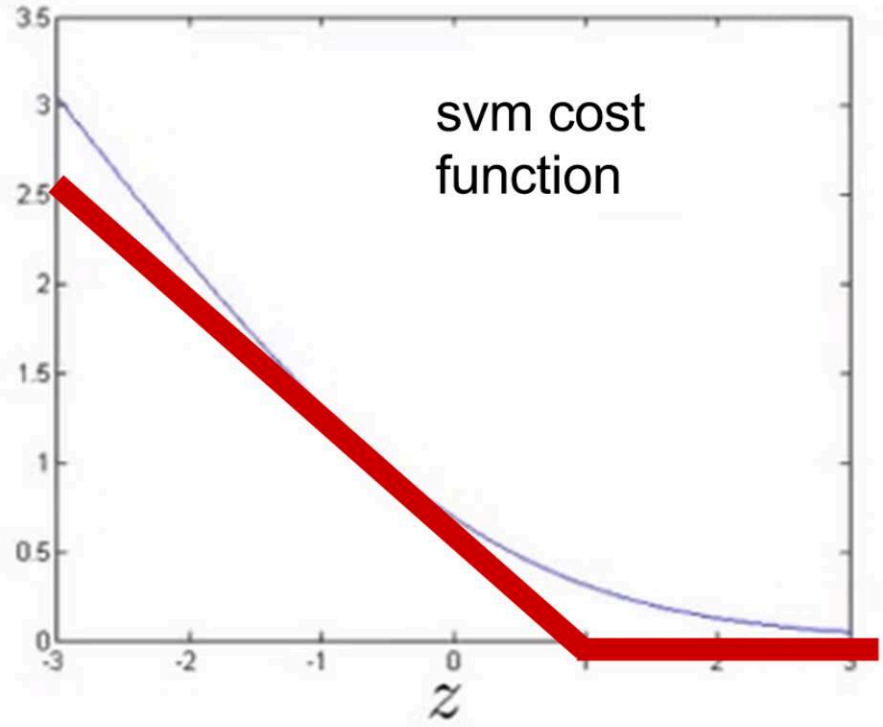
$$Cost(y_{pred}, y_{true}) = \begin{cases} -\log(y_{pred}) & if \quad y_{true} = 1 \\ -\log(1 - y_{pred}) & if \quad y_{true} = 0 \end{cases}$$



Logistic Regression Cost Function

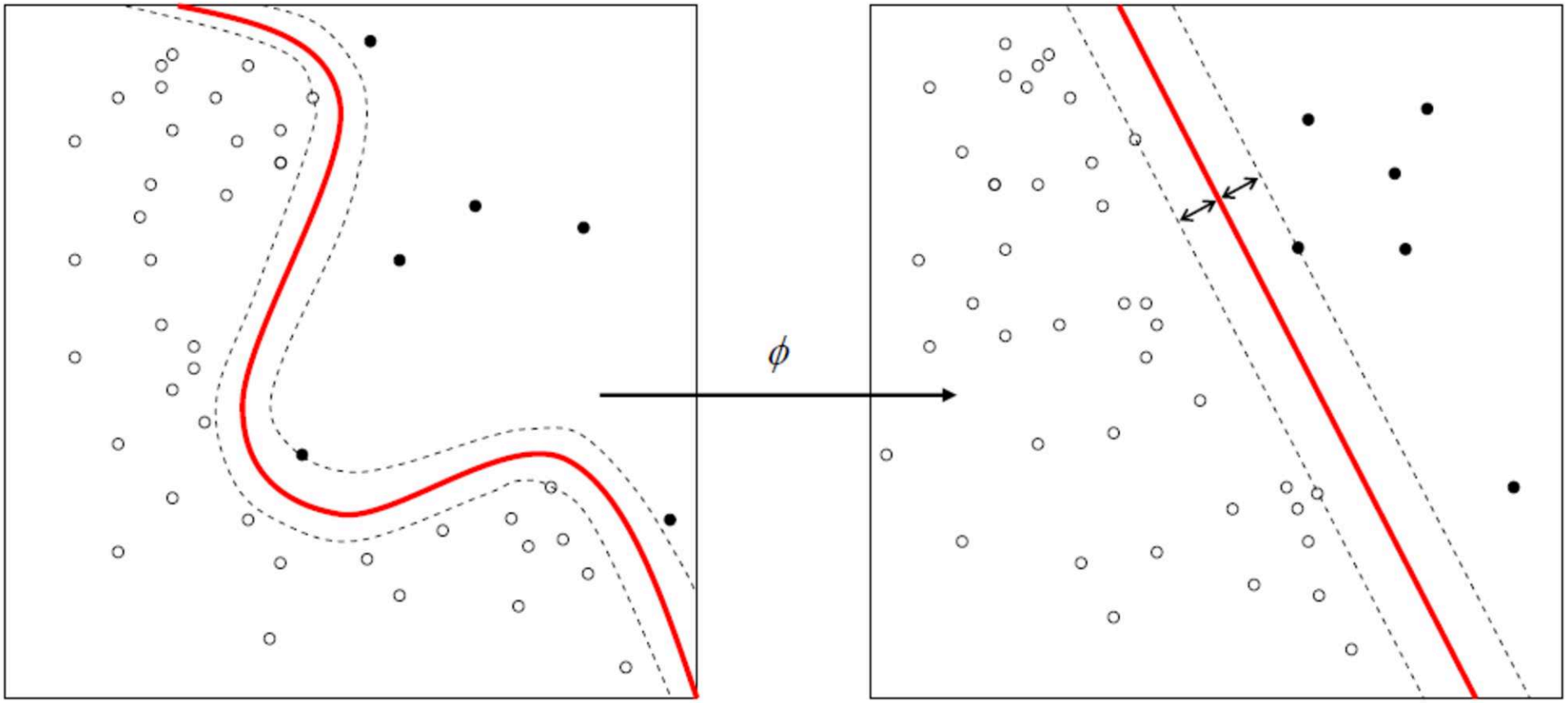
$$Cost(y_{pred}, y_{true}) = \begin{cases} -\log(y_{pred}) & if \quad y_{true} = 1 \\ -\log(1 - y_{pred}) & if \quad y_{true} = 0 \end{cases}$$





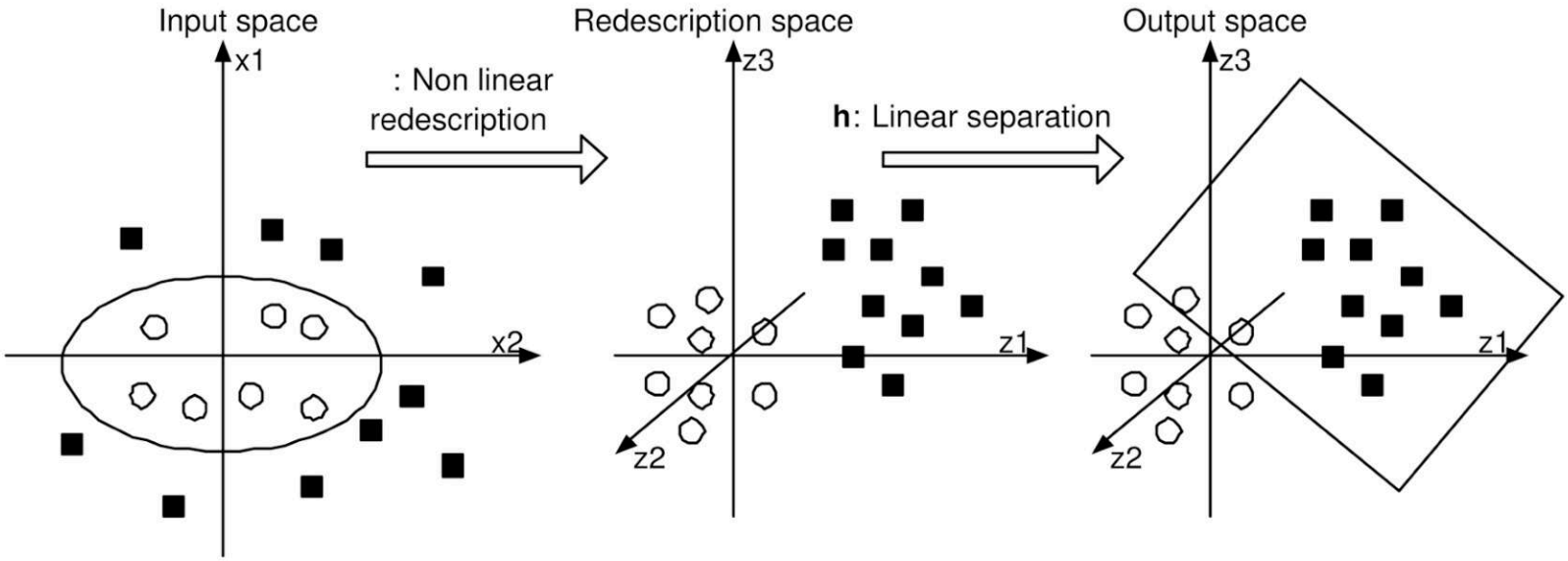
Nonlinear SVMs

Nonlinear data is linear in higher dimensions



Kernel Trick

Transform data so it is linearly separable



Different Kernels

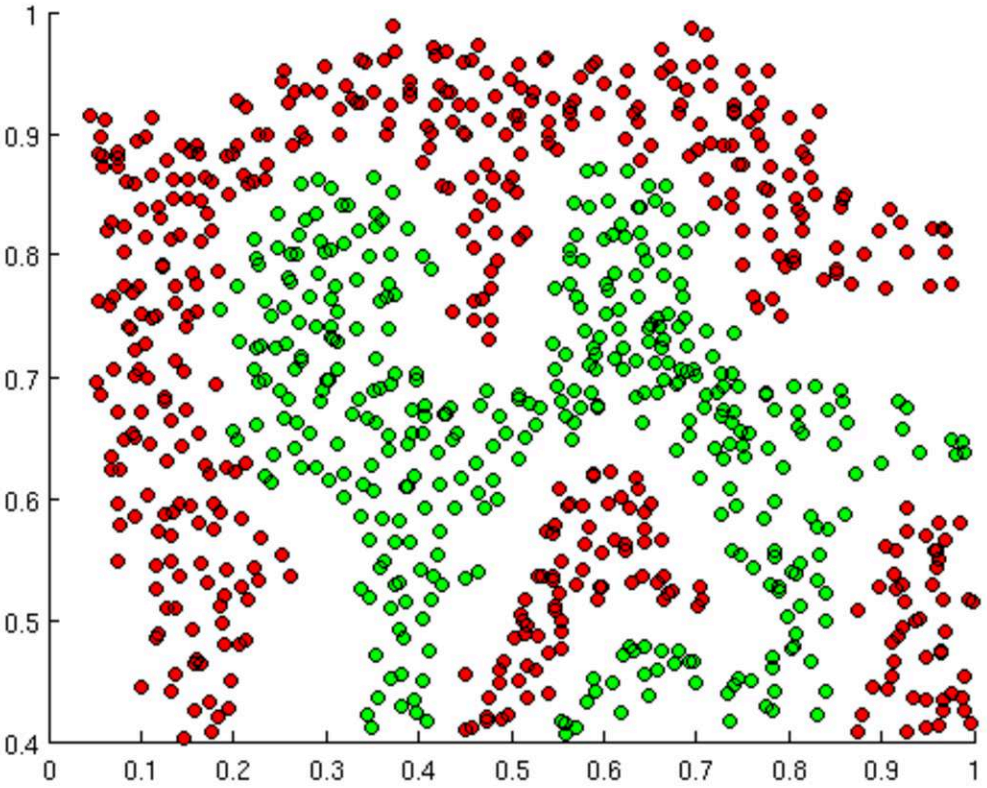
Linear kernel: No transformation

Type of Support Vector Machine	Inner Product Kernel $K(x,x_i)$, I = 1, 2, …, N	Comments
Polynomial learning machine	$(x^T x_i + 1)^p$	Power p is specified apriori by the user
Radial-basis function network	$\exp(1/(2\sigma^2)\|\|x-x_i\|\|^2)$	The width σ^2 is specified apriori
Two layer perceptron	$\tanh(\beta_0 x^T x_i + \beta_1)$	Mercer's theorem is satisfied only for some values of β_0 and β_1

Different Kernels

Type of Support Vector Machine	Inner Product Kernel $K(x,x_i)$, I = 1, 2, …, N	Comments
Polynomial learning machine	$(x^T x_i + 1)^p$	Power p is specified apriori by the user
Radial-basis function network	$\exp(1/(2\sigma^2)\|\|x-x_i\|\|^2)$	The width σ^2 is specified apriori
Two layer perceptron	$\tanh(\beta_0 x^T x_i + \beta_1)$	Mercer's theorem is satisfied only for some values of β_0 and β_1

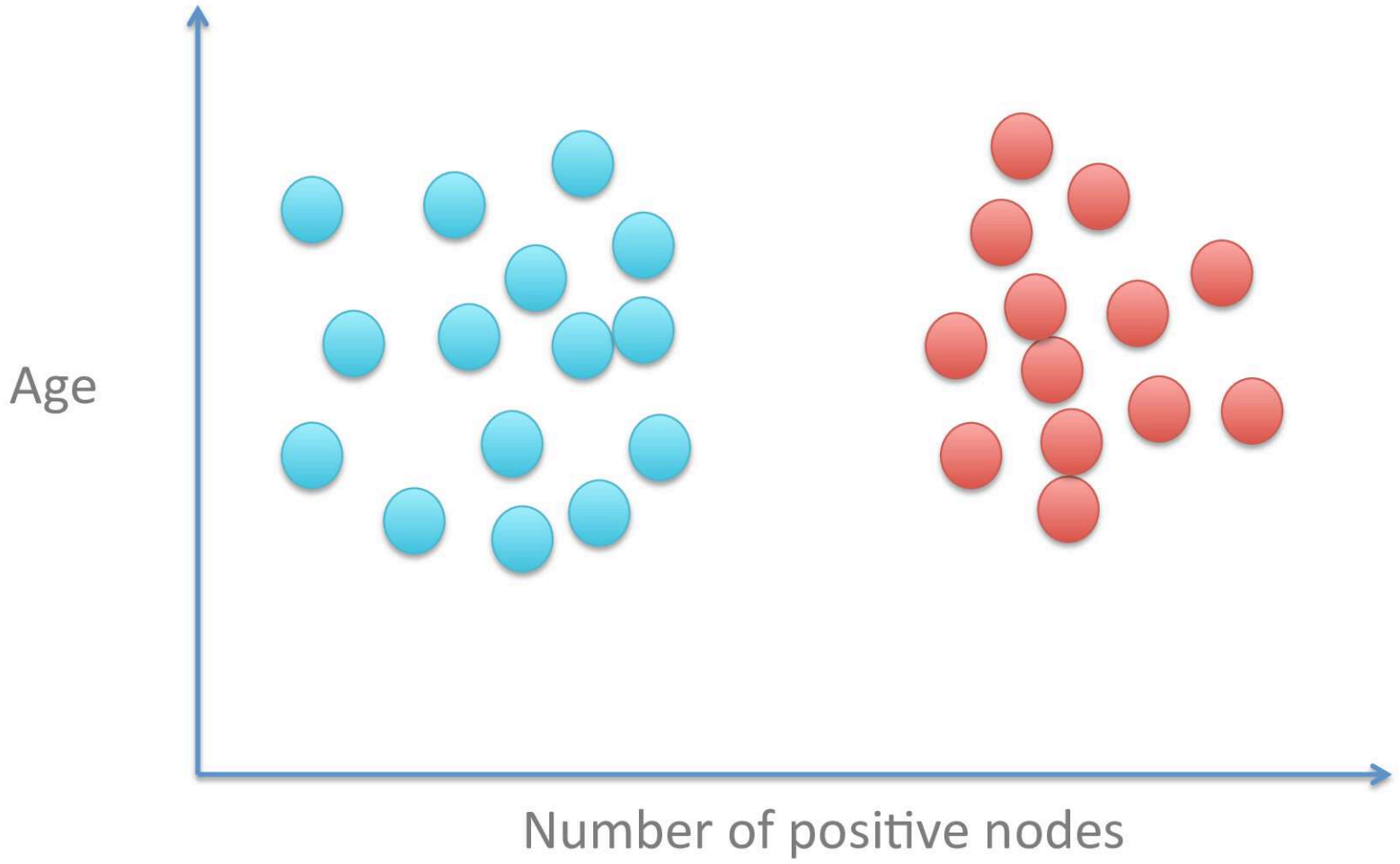
SVMs can fit intricate boundaries with the Gaussian (RBF) kernel



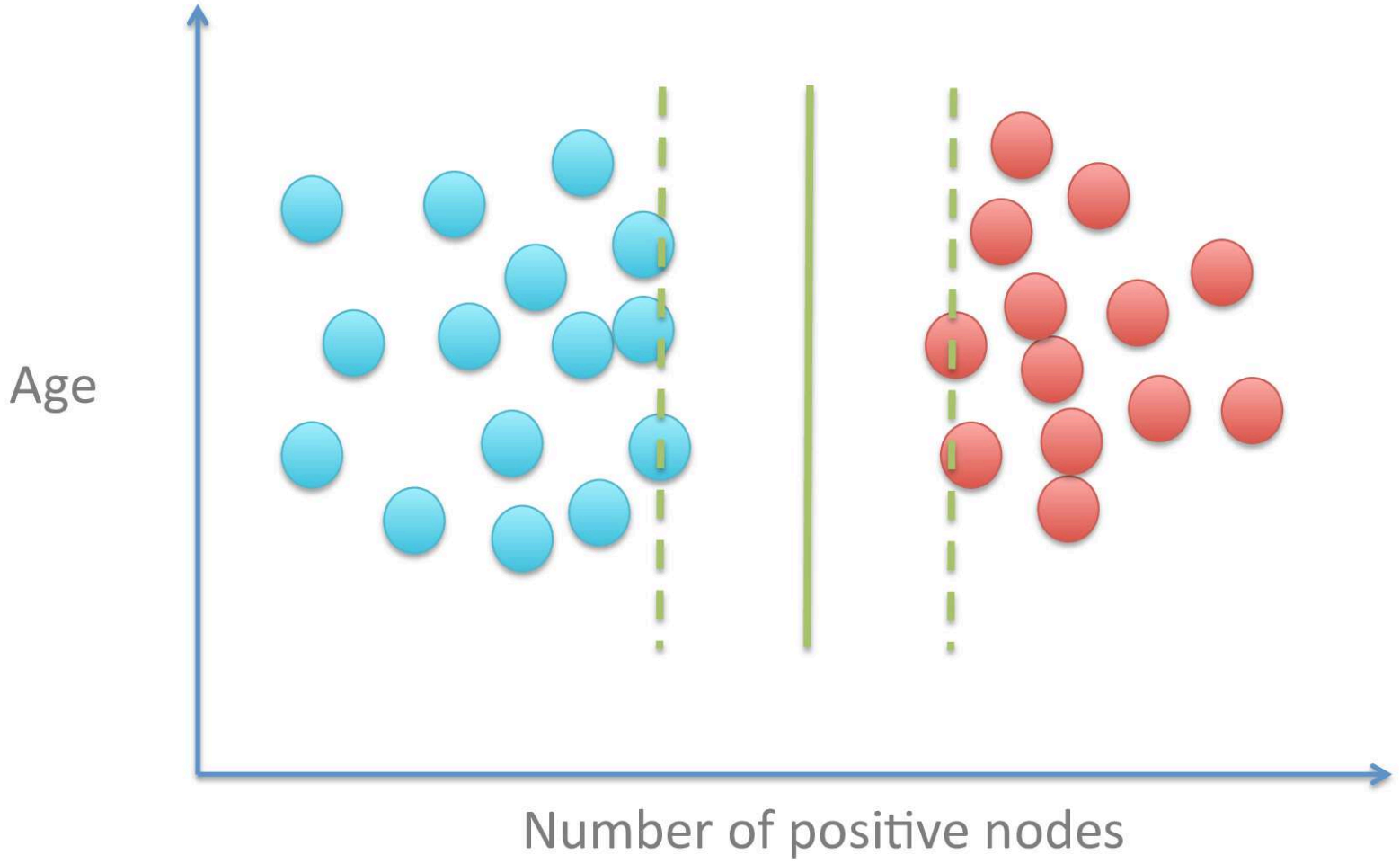
```
from sklearn.svm import SVC

from sklearn.svm import LinearSVC
```

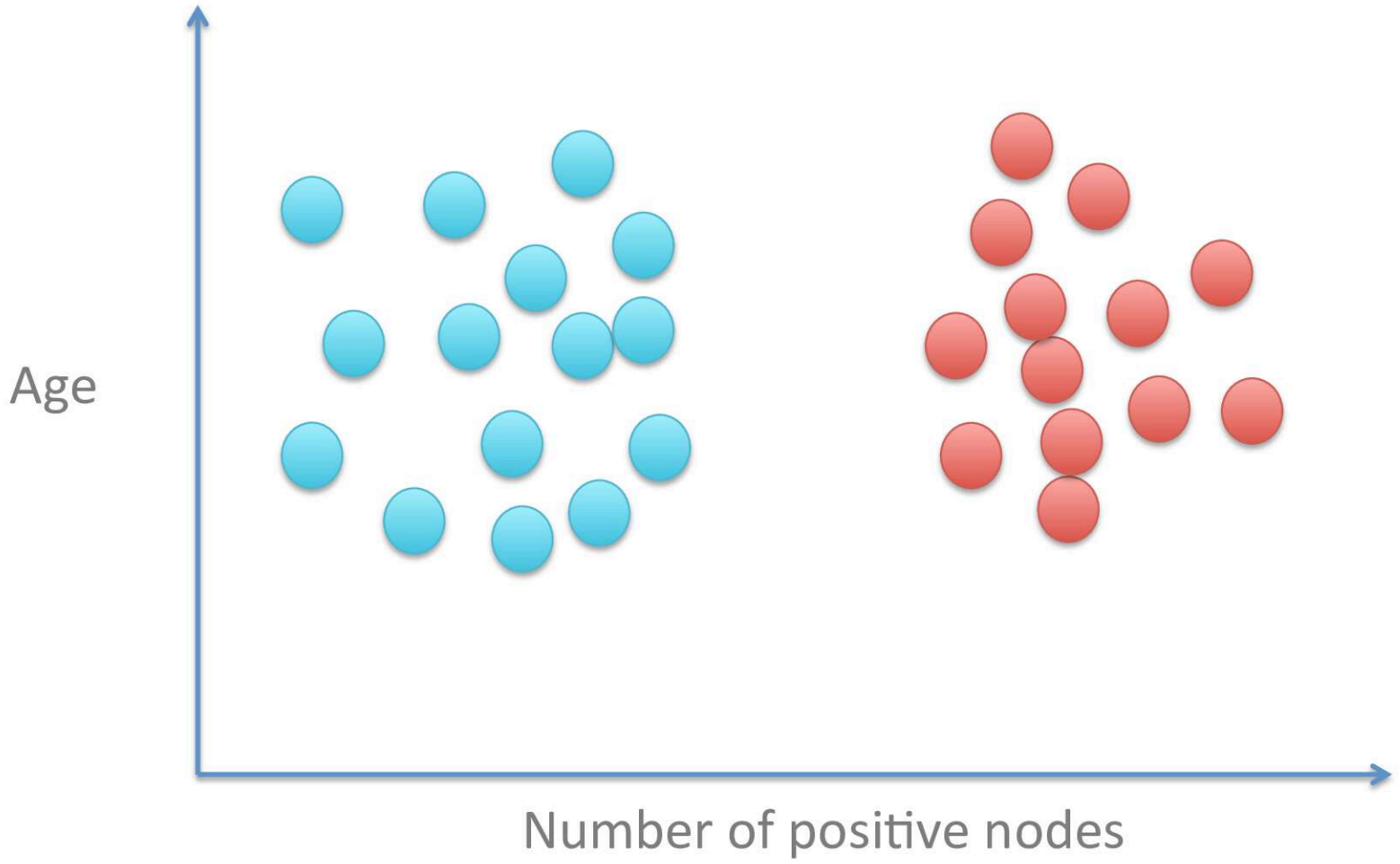
Outlier problems



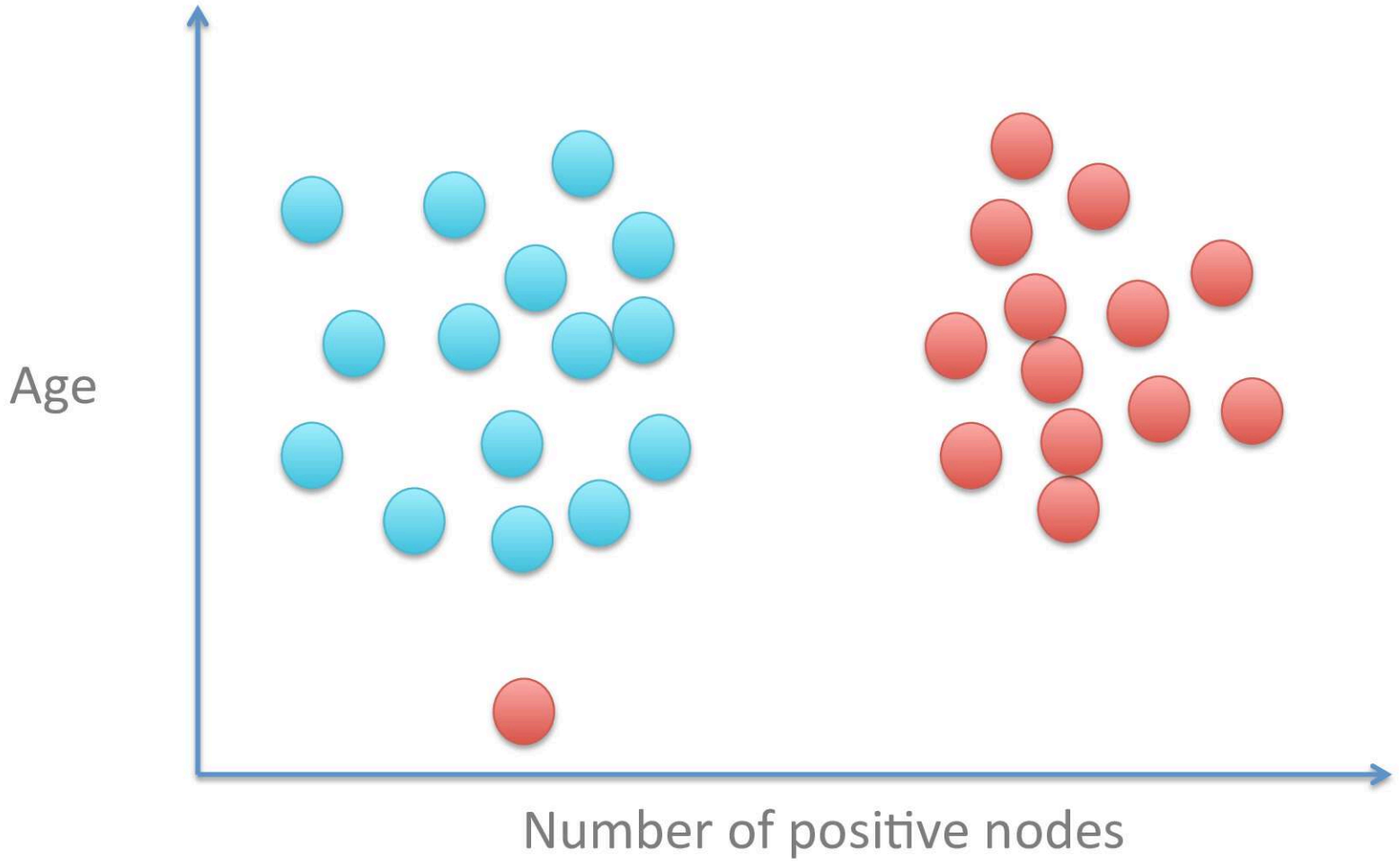
Outlier problems



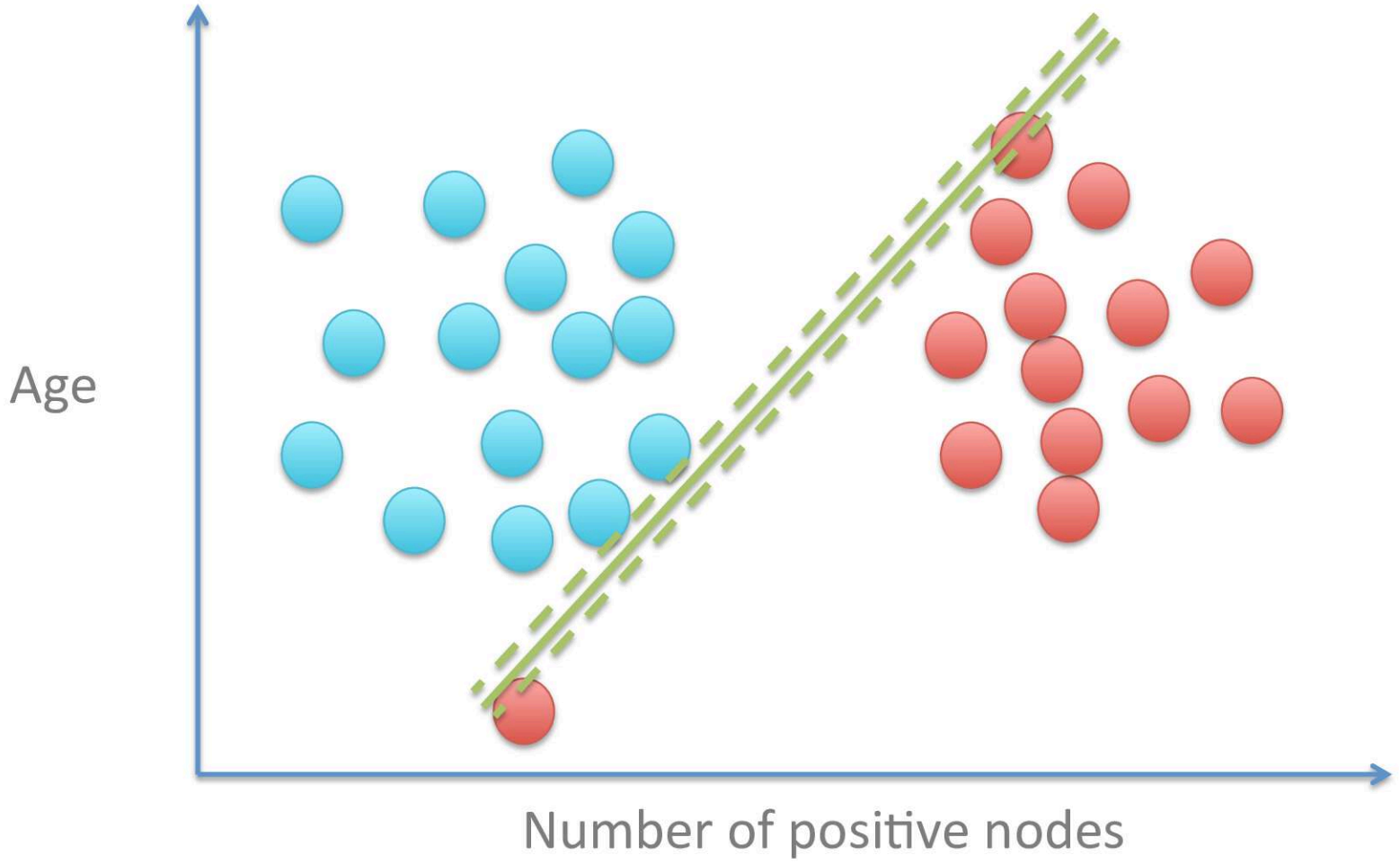
Outlier problems



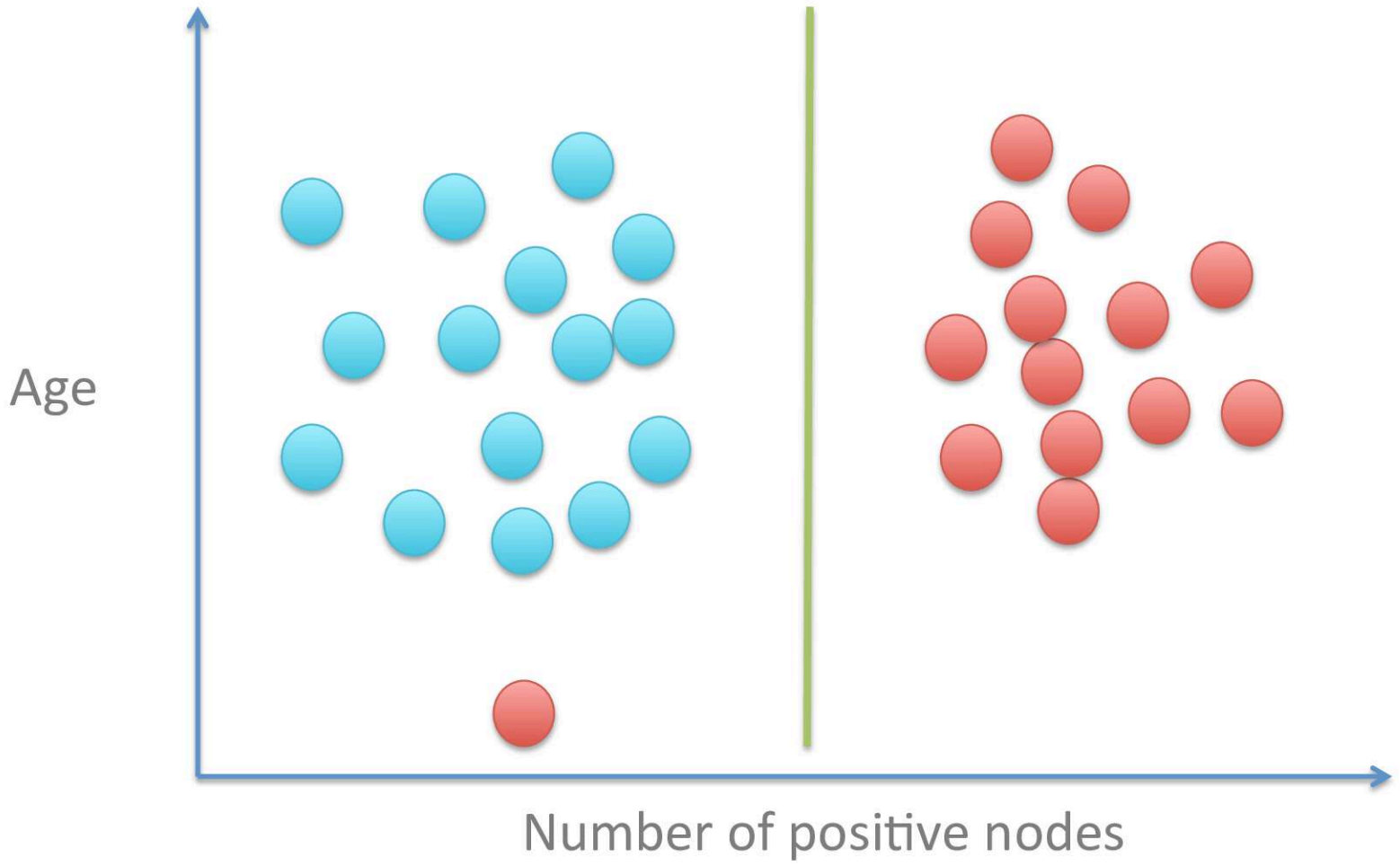
Outlier problems



Outlier problems

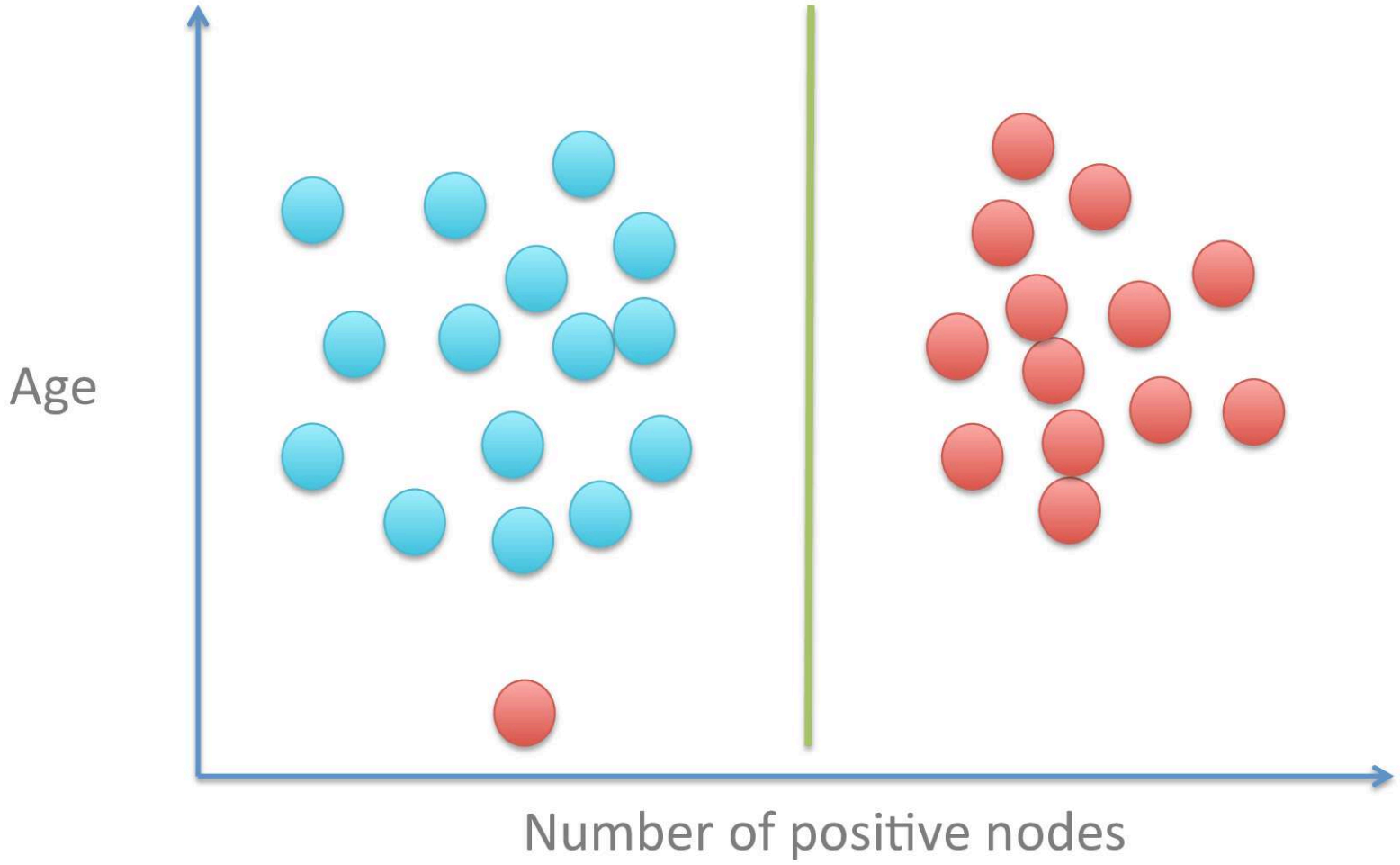


This is probably still the best boundary



Regularization in the Cost Function handles this

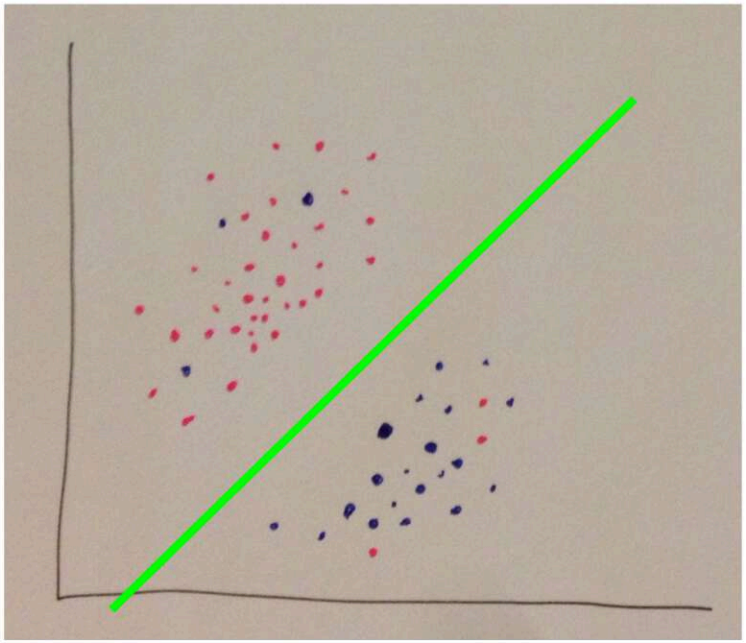
$$Cost(y_{pred}, y_{true}) = SVMCost(\beta_i) + \frac{1}{C}\sum_i \beta_i^2$$



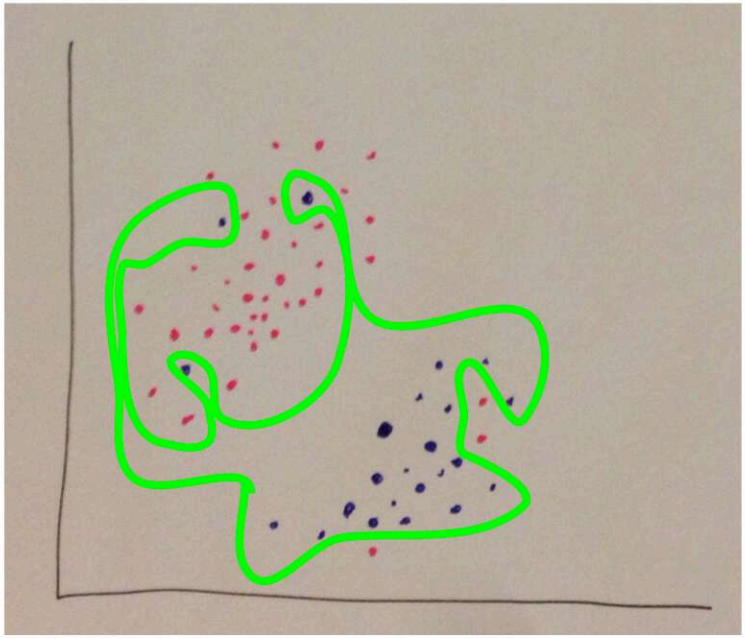
Tune regularization with C

$$Cost(y_{pred}, y_{true}) = SVMCost(\beta_i) + \frac{1}{C}\sum_i \beta_i^2$$

Low C makes the decision surface smooth



High C aims at classifying all training examples correctly.

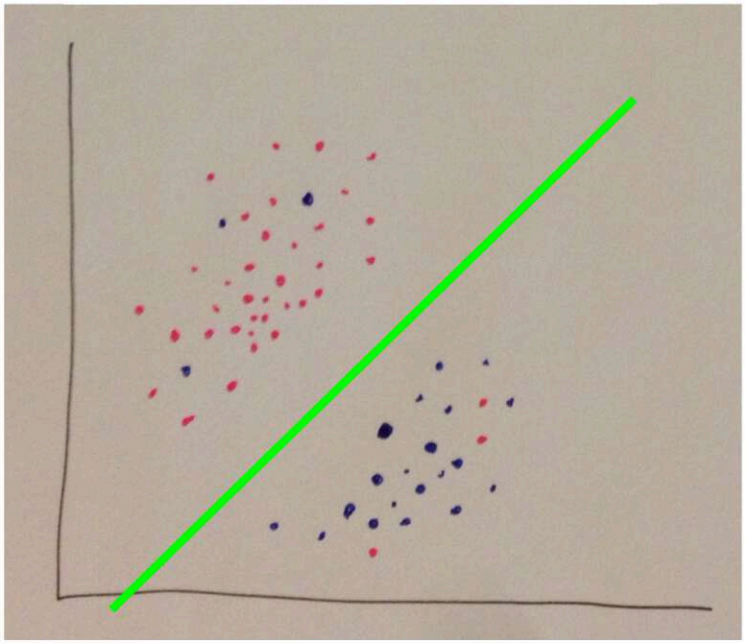


Tune regularization with C

$$Cost(y_{pred}, y_{true}) = SVMCost(\beta_i) + \frac{1}{C}\sum_i \beta_i^2$$

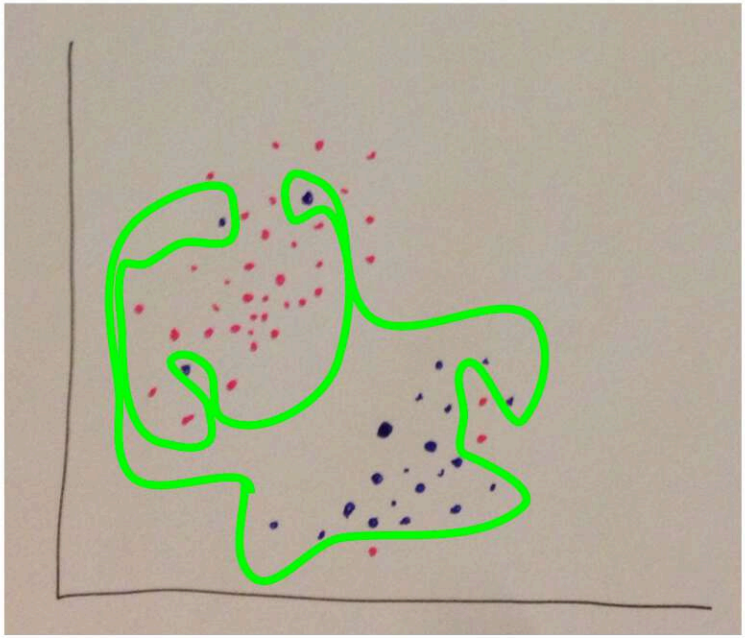
Low C

underfit



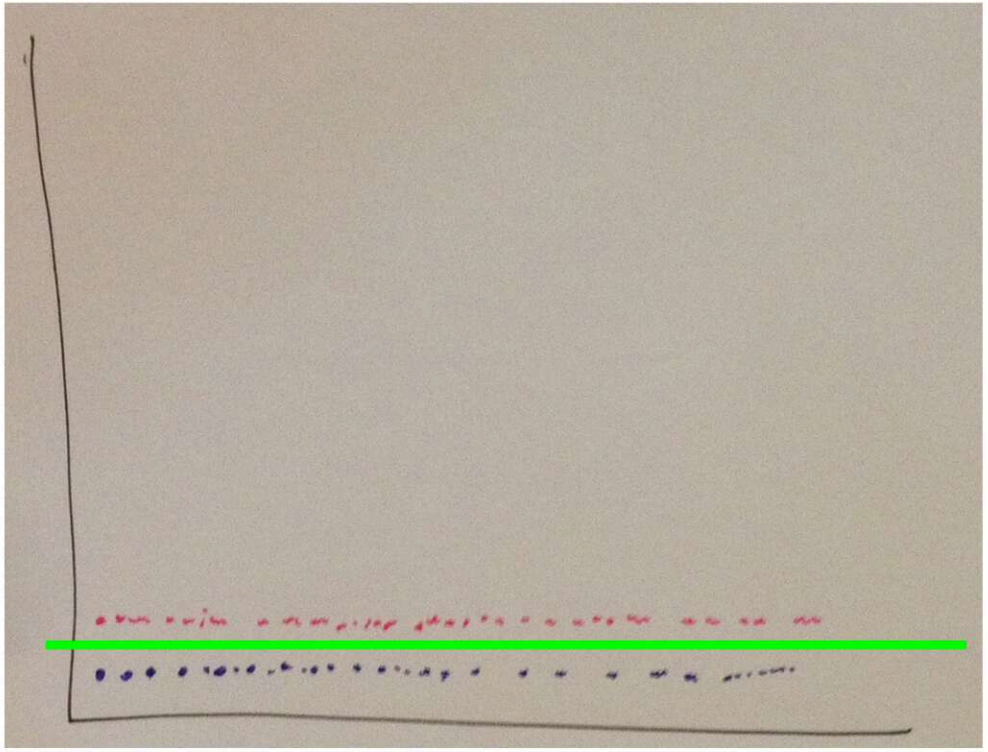
High C

overfit

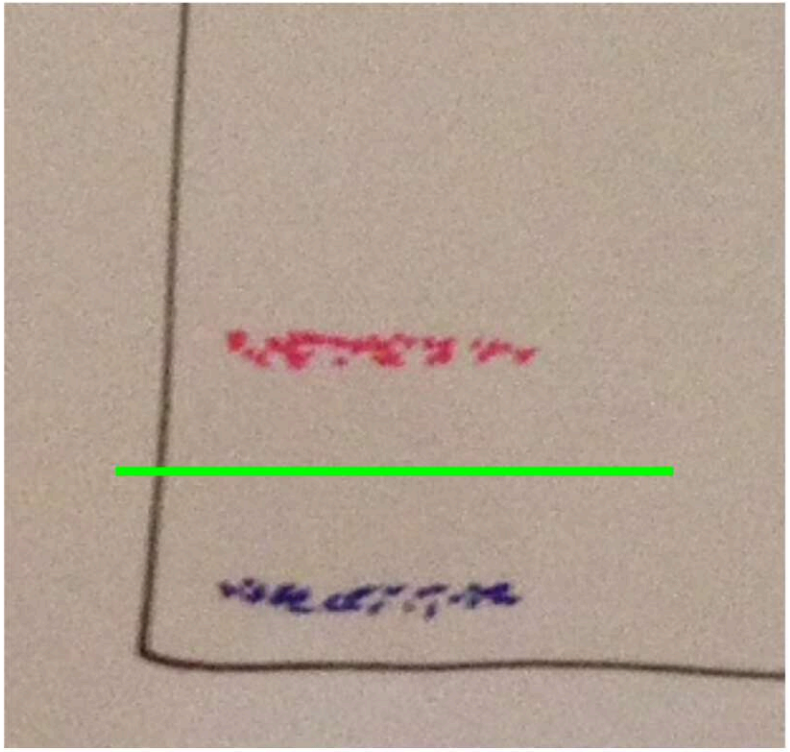


SVMs are not scale-invariant.

Scale matters



harder to fit



easier to fit

```
from sklearn import preprocessing

X_scaled = preprocessing.scale(X)
```

When to use SVMs, Linear Regression, etc.

A lot of features, but small data
(10K features, 1000 training examples)

Best you can do is a simple model, go for Linear Regression or LinearSVC

When to use SVMs, Linear Regression, etc.

Few features, decent data
(5-100 features, 10K training examples)

Go for a Gaussian Kernel SVC, rock the hell out of it.

When to use SVMs, Linear Regression, etc.

Few features, lots of data
(5-100 features, 100K training examples)

With that much data, you can extract more from weaker features. Add more features. Use Linear Regression or LinearSVC

(Gaussian kernel SVC is too slow with large data)