Support vector machines

1) What is a classifier supposed to do and why is SVM a good one?

2) Establish SVM methodology by walking through a simple case

3) Generalize the SVM to make it more robust

4) A simple trick allows us to create a very complex classifier

5) Practical tips for using SVM

Support vector machines

1) What is a classifier supposed to do and why is SVM a good one?

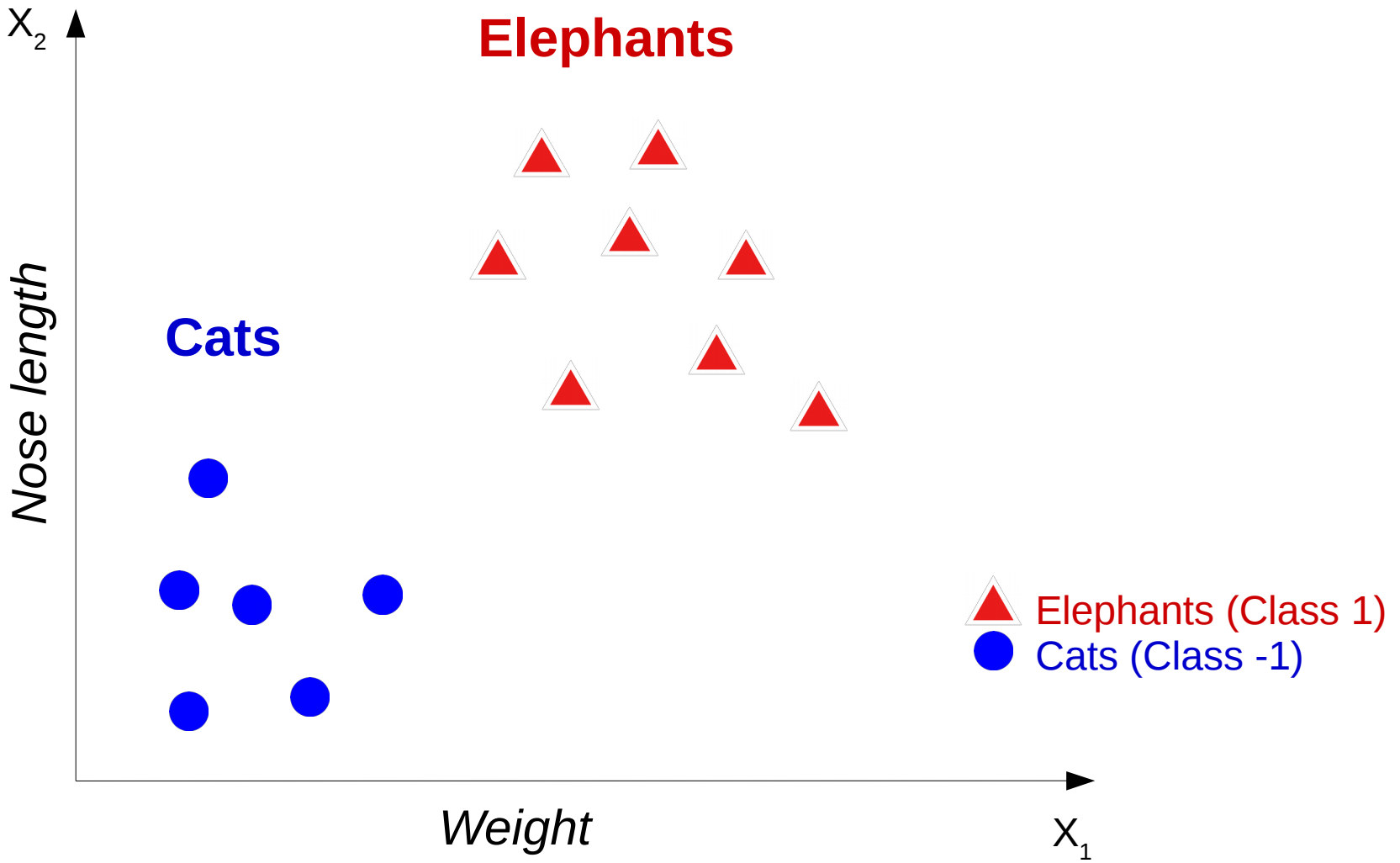
2) Establish SVM methodology by walking through a simple case

3) Generalize the SVM to make it more robust

4) A simple trick allows us to create a very complex classifier

5) Practical tips for using SVM

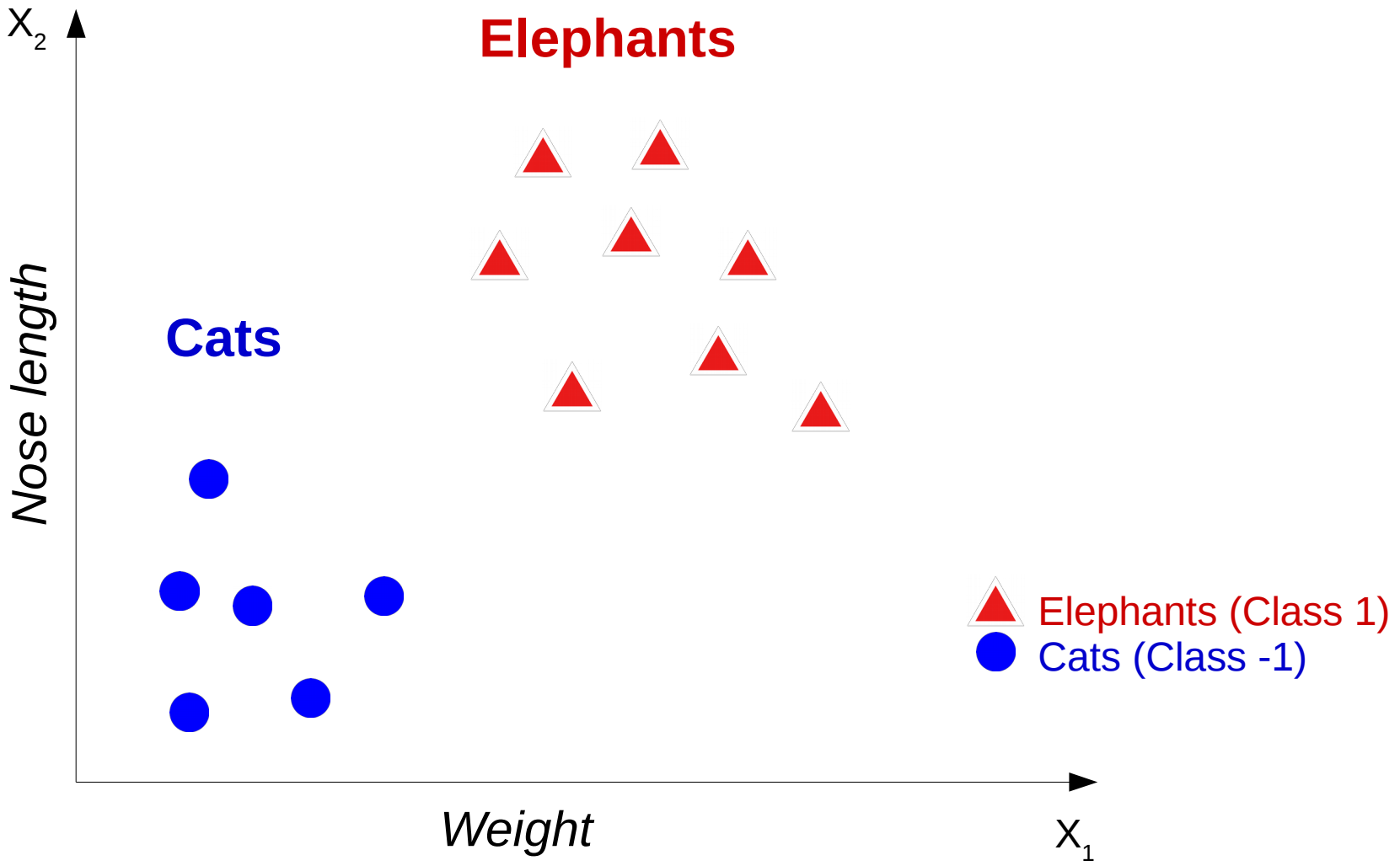
Classification is a categorical supervised learning problem



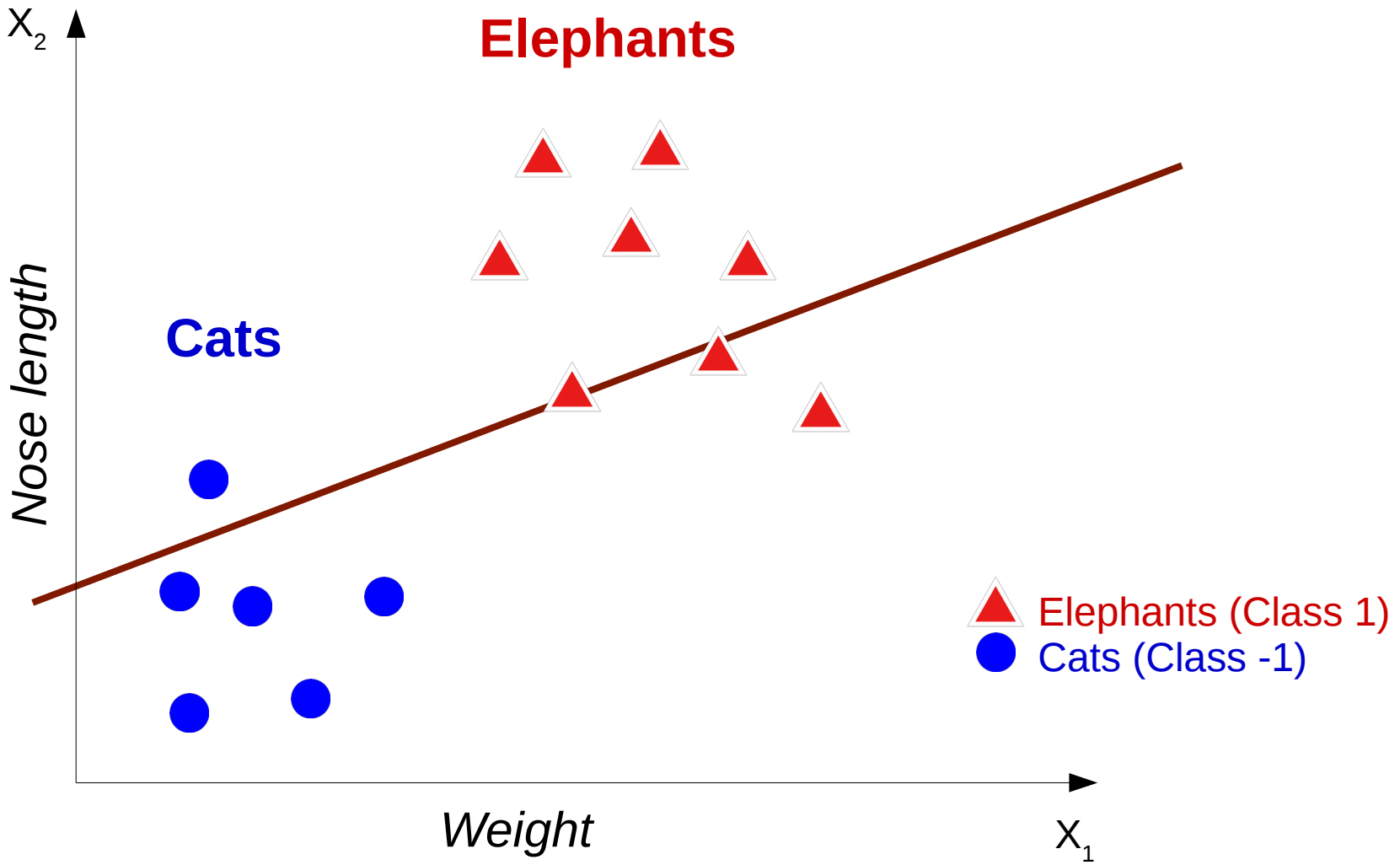
Classifier problem:

Given a new object's weight and nose length, classify ("identify") as elephant or cat

What's a good decision boundary?

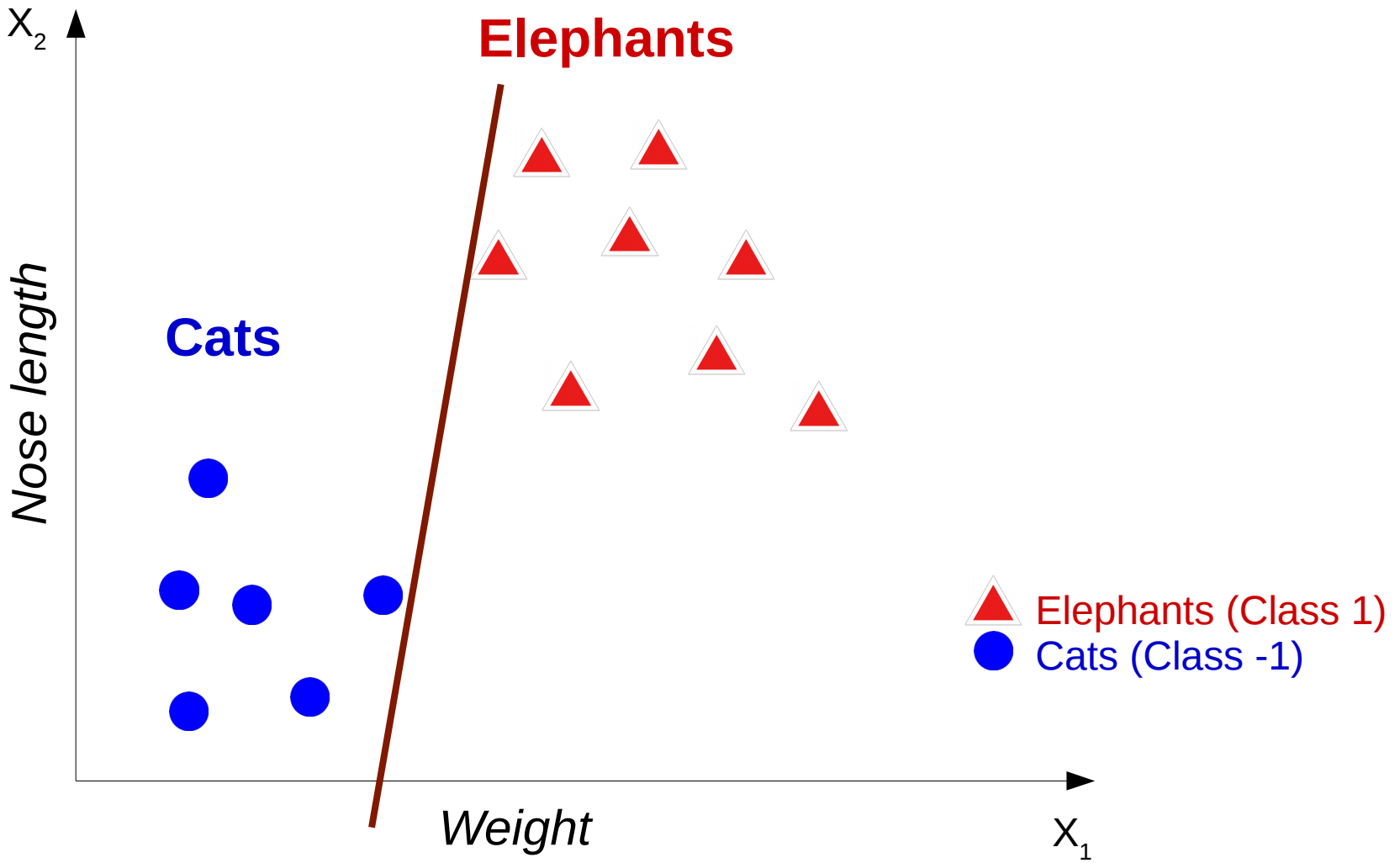


What's a good decision boundary?



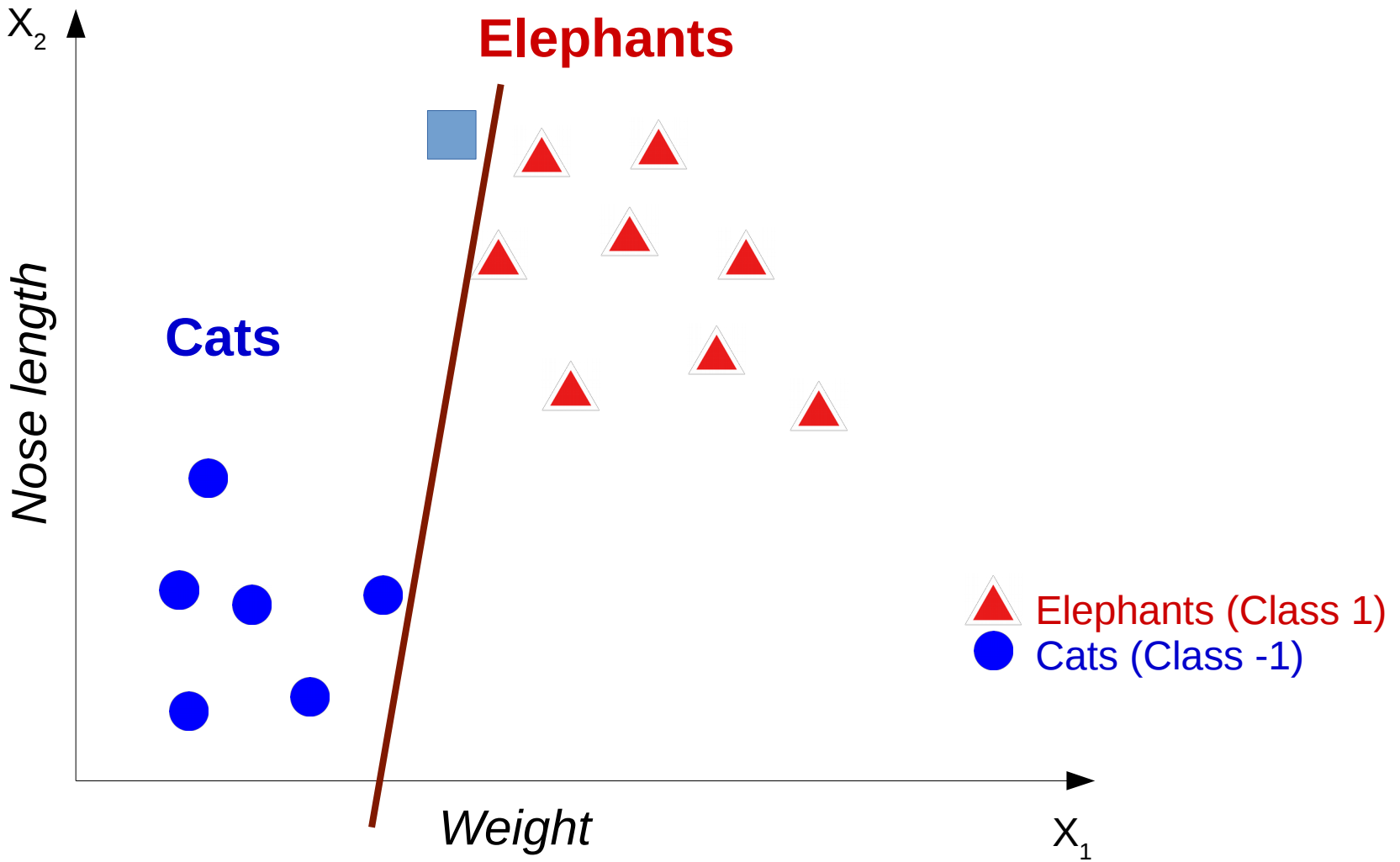
Doesn't separate the classes!

What's a good decision boundary?



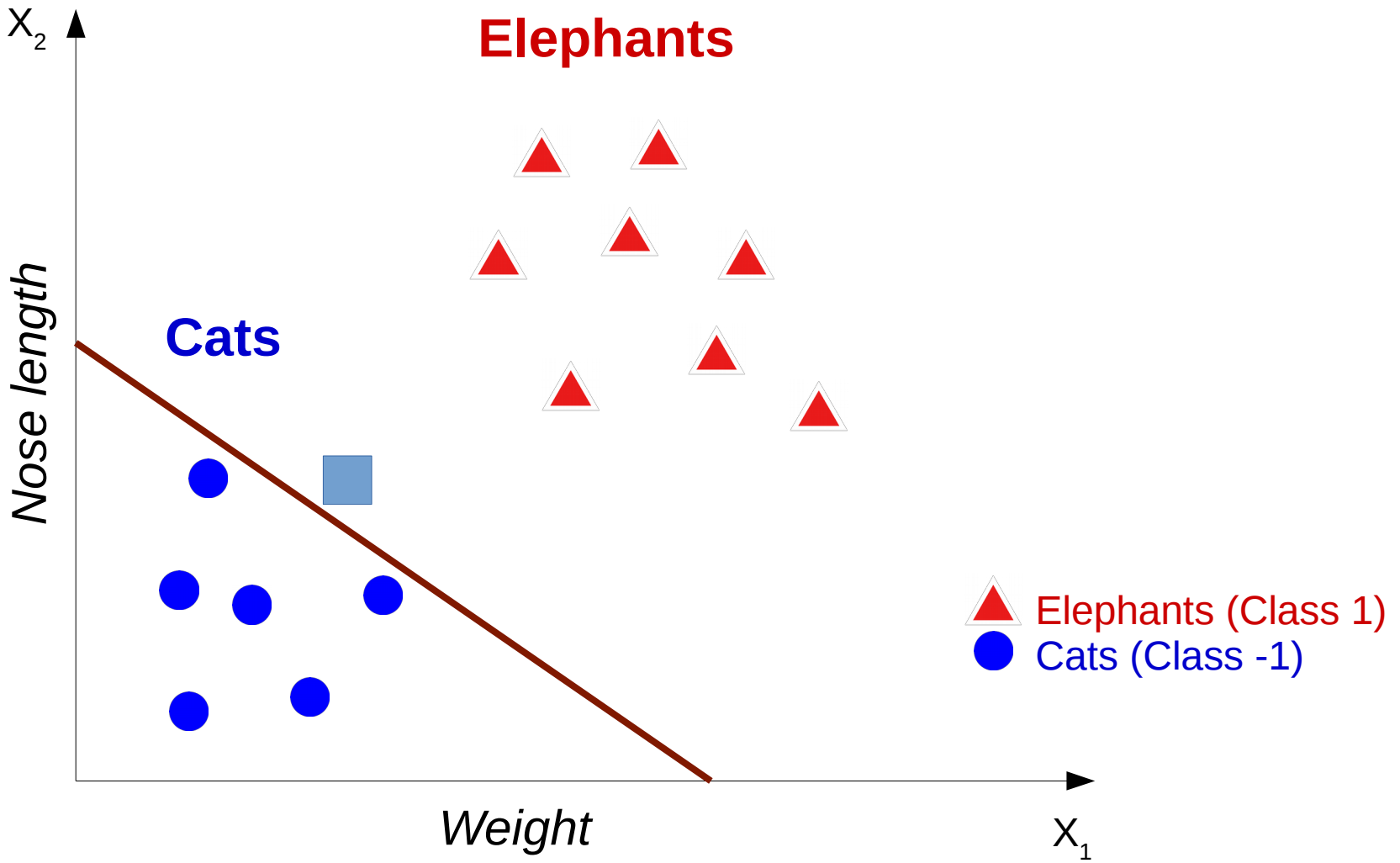
Why is this not optimal?

What's a good decision boundary?



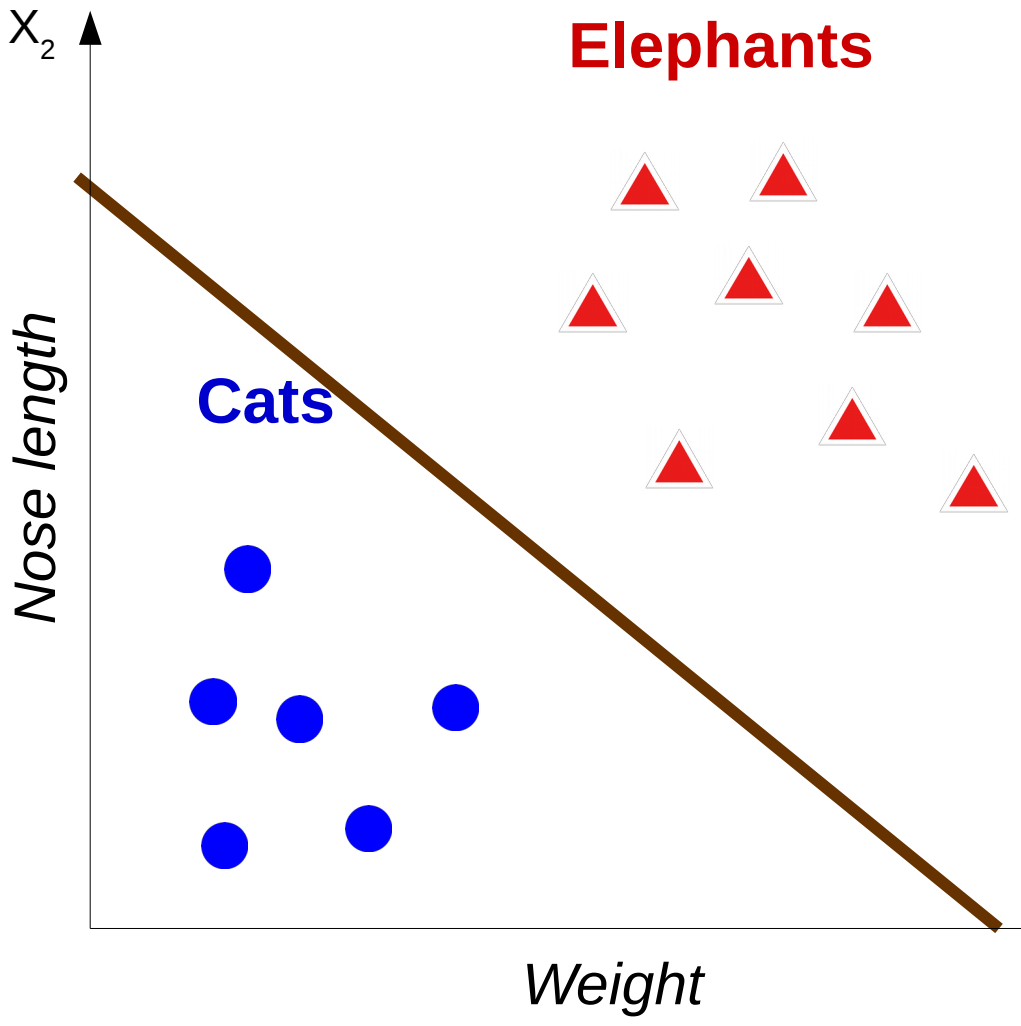
Doesn't treat all points equally, not symmetrical
New point (square) seems like it might be an elephant

What's a good decision boundary?



Same problem. Not symmetrical/fair. Bad generalization error

What's a good decision boundary?

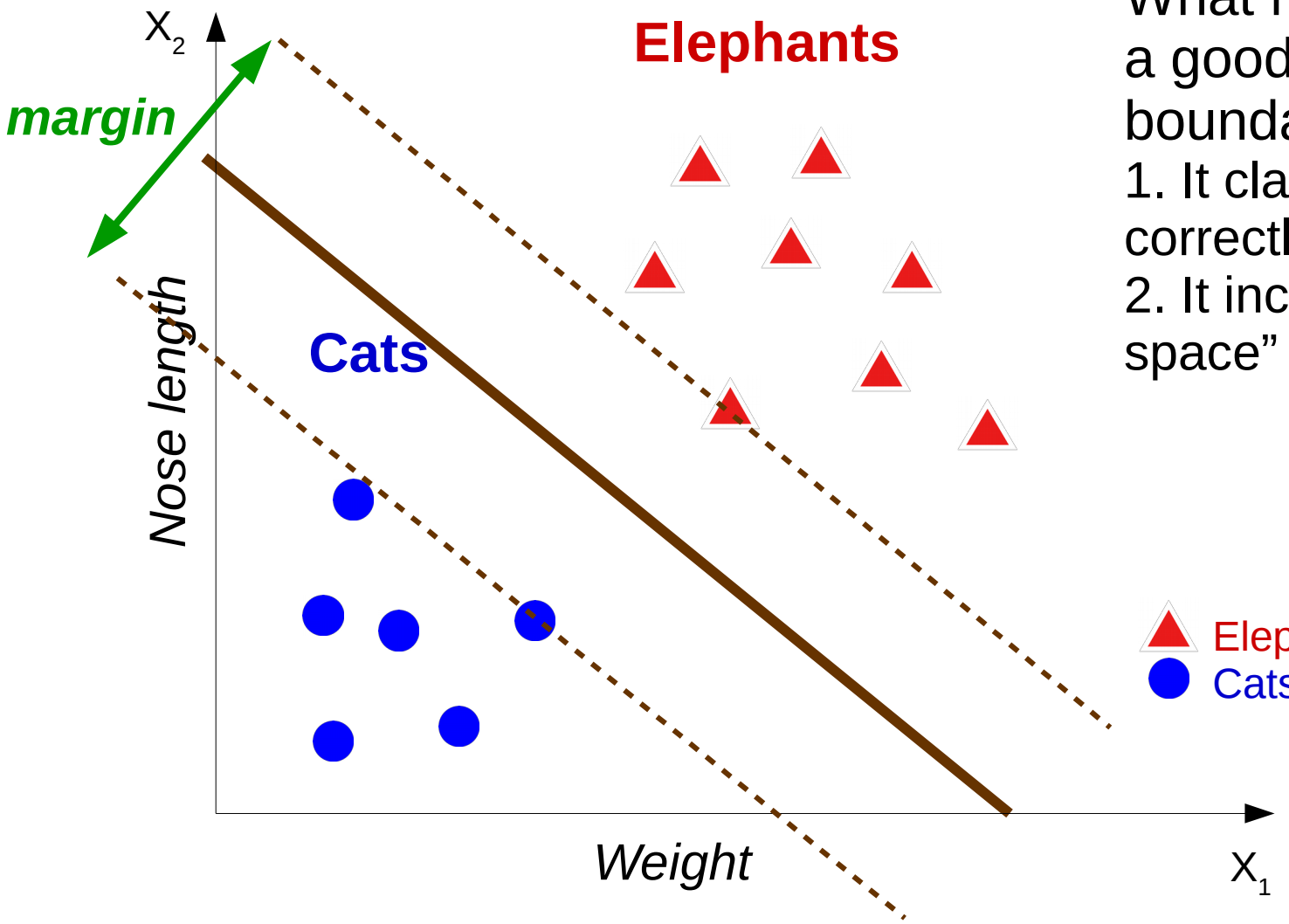


What makes this plane a good decision boundary?

1. It classifies all the data correctly
2. What else?

Elephants (Class 1)
Cats (Class -1)

A good boundary has a large margin



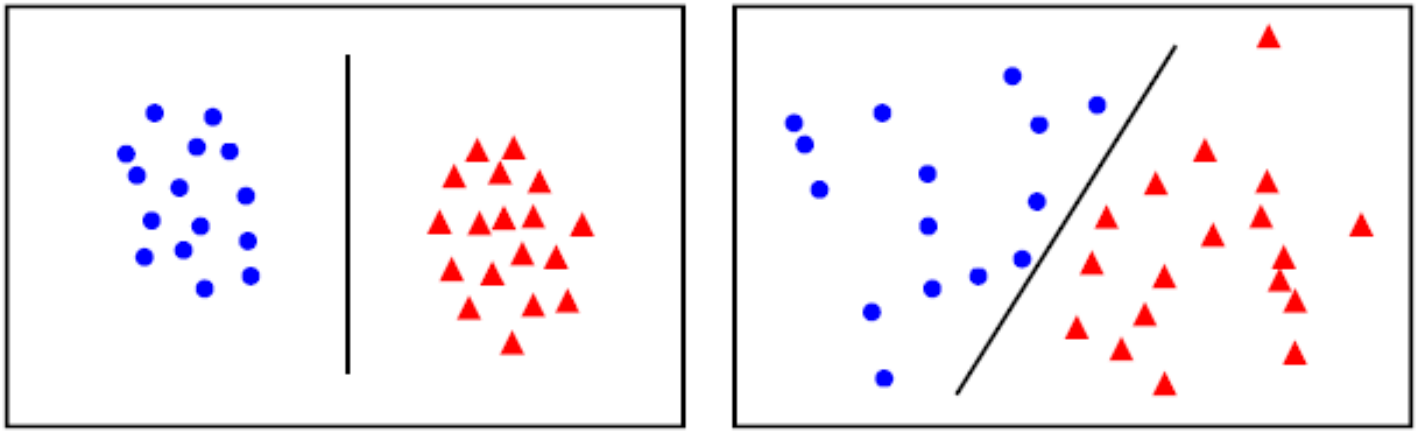
What makes this plane a good decision boundary?

1. It classifies all the data correctly
2. It increases the “empty space” (margin)

Elephants (Class 1)
Cats (Class -1)

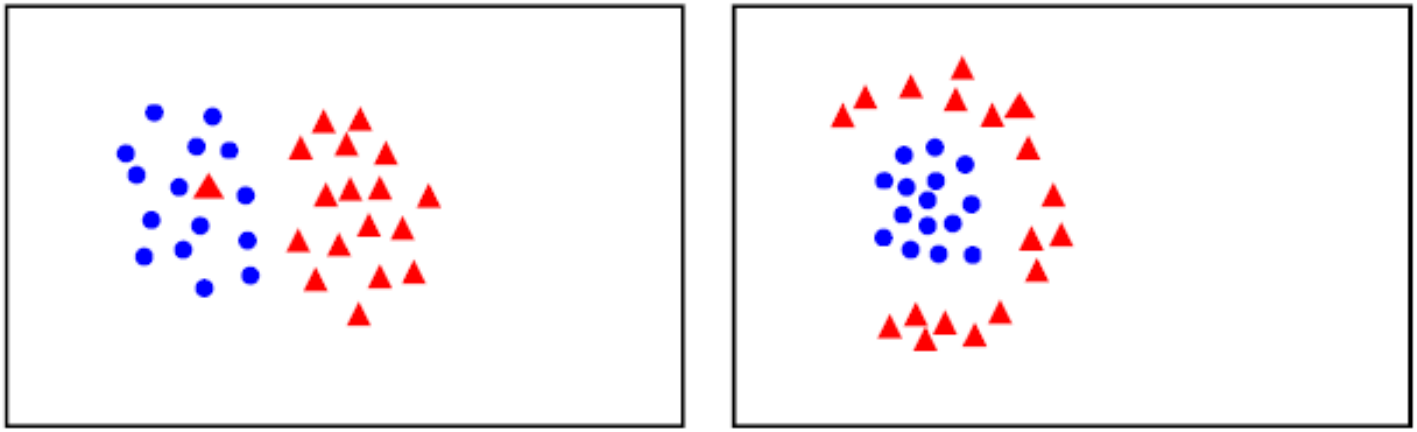
The power of SVM lies in non linearly separable data

linearly separable



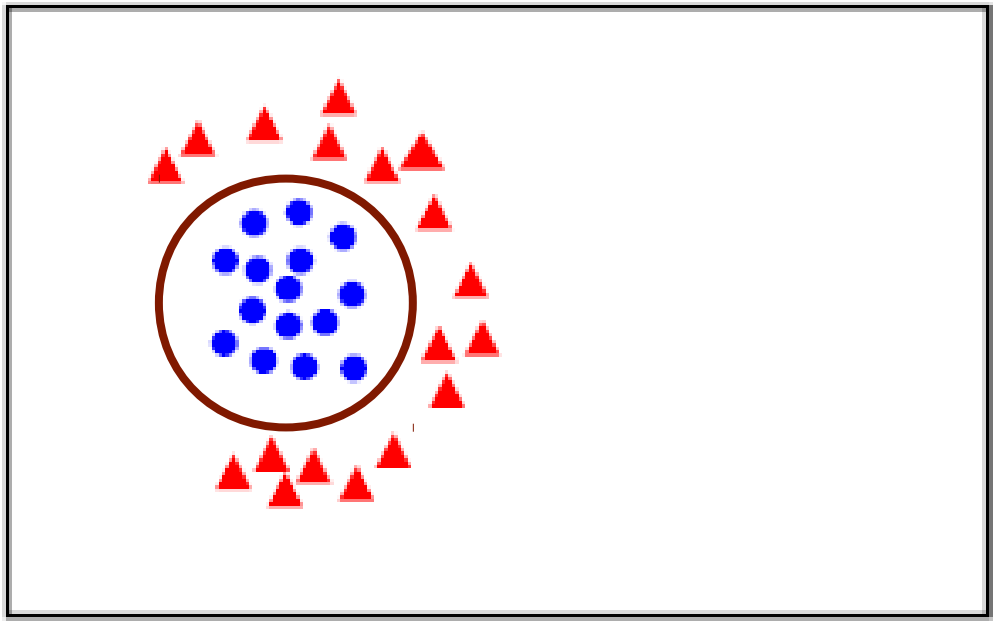
→ *SVM designed to increase the margin, and thus improve generalization error*

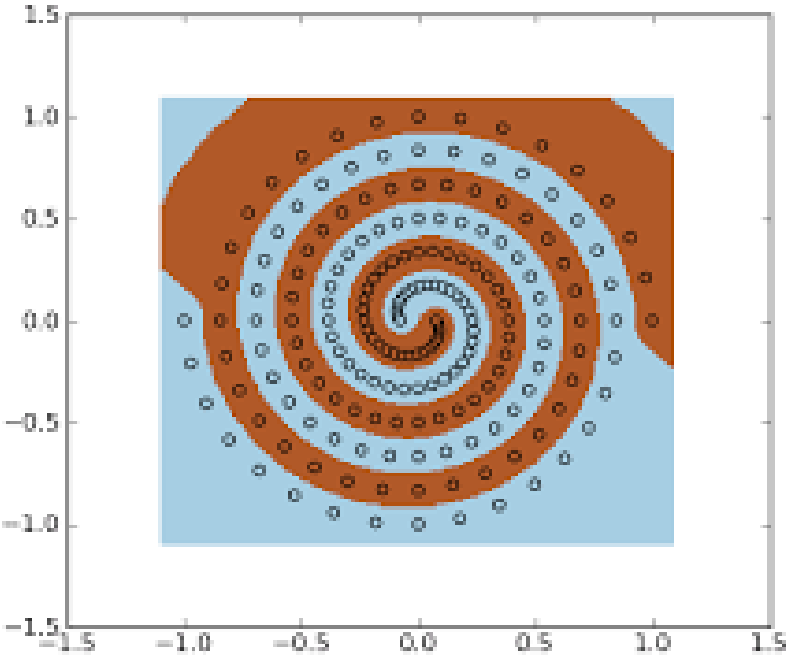
not linearly separable



→ *For these cases SVM really shines! (More on that later)*

SVM finds complex non-linear boundaries very efficiently





Not impossible with Logistic regression but much less efficient

(More on non-linear boundaries later)

Support vector machines

1) What is a classifier supposed to do and why is SVM a good one?

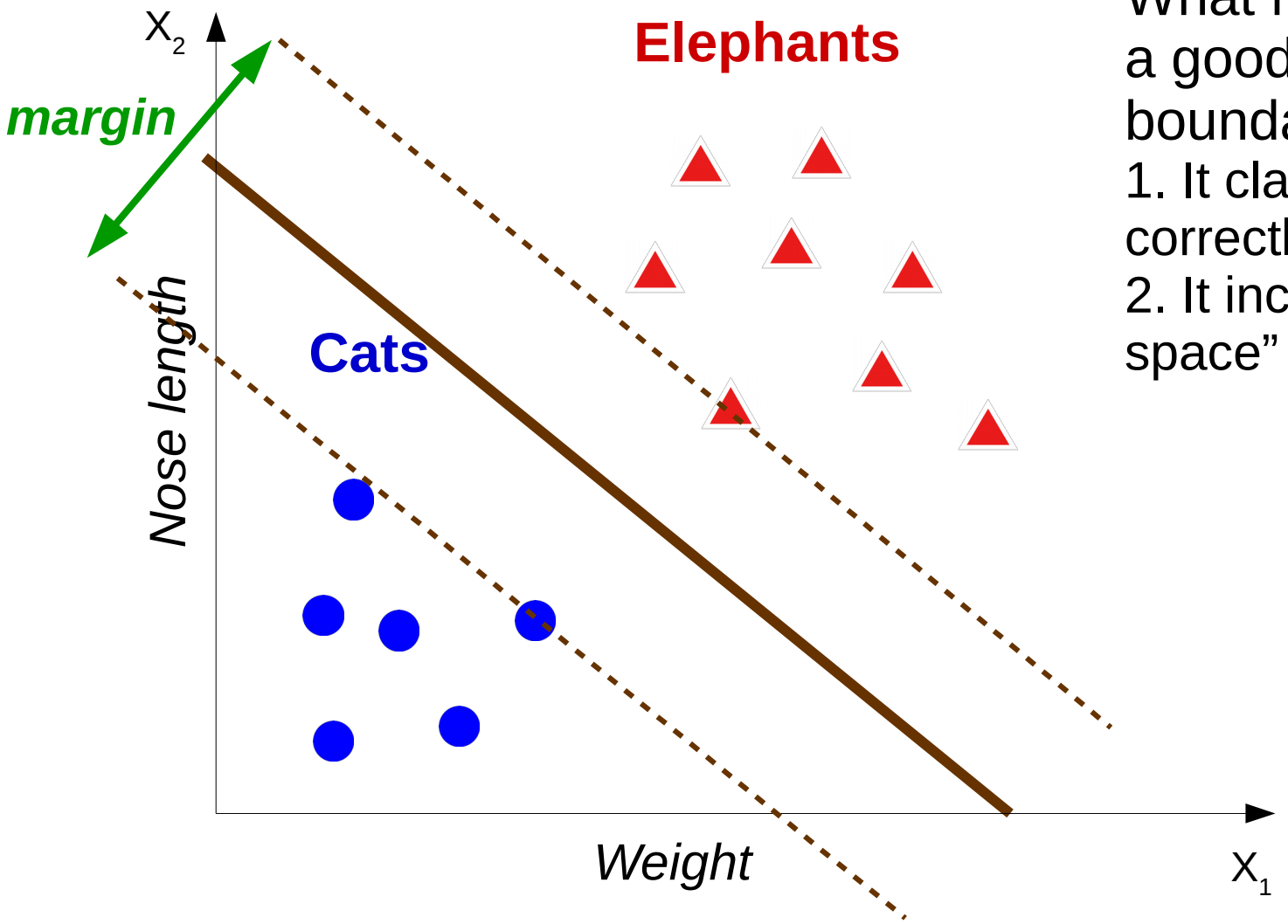
2) Establish SVM methodology by walking through a simple case

3) Generalize the SVM to make it more robust

4) A simple trick allows us to create a very complex classifier

5) Practical tips for using SVM

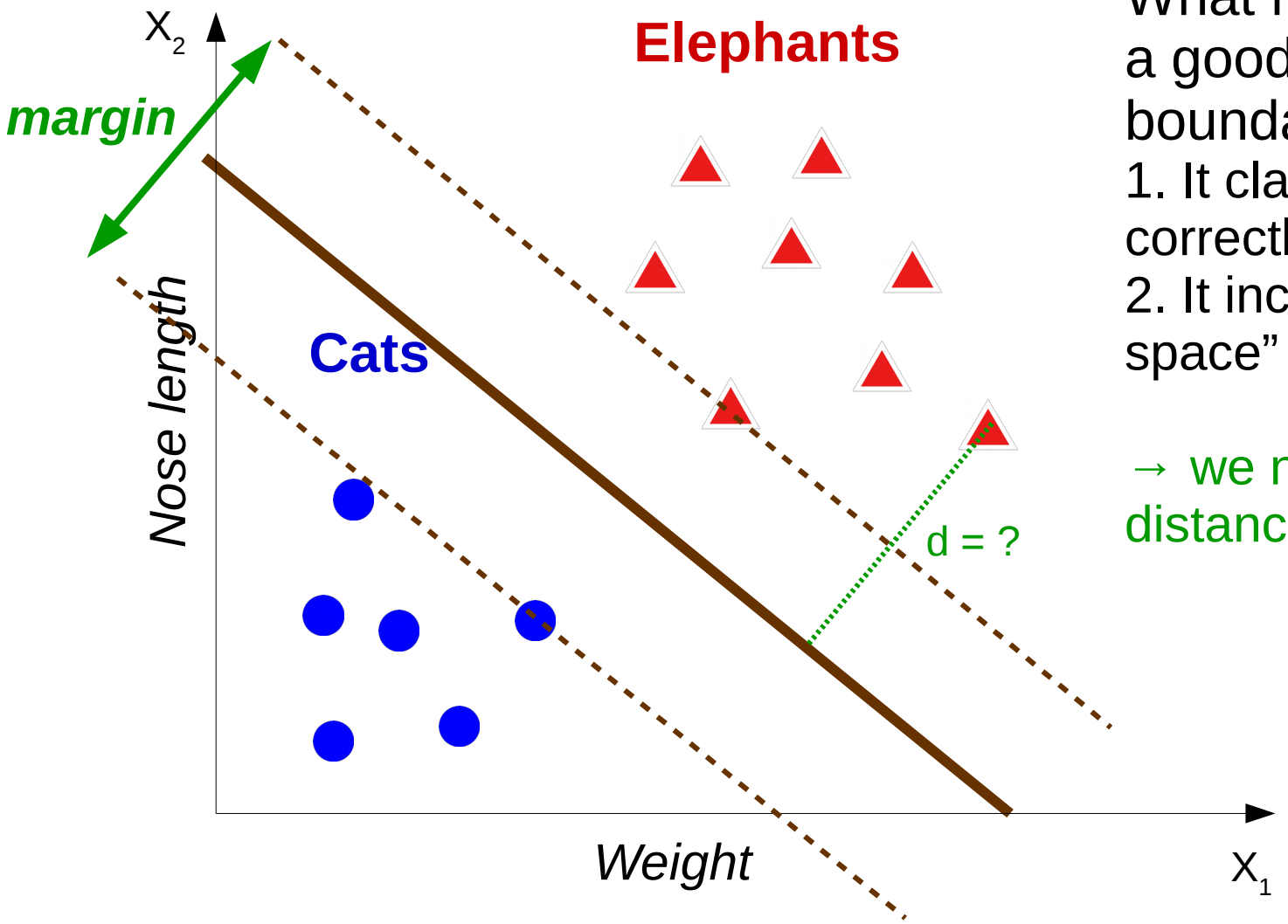
A good boundary has a large margin



What makes this plane a good decision boundary?

1. It classifies all the data correctly
2. It increases the “empty space” (margin)

A good boundary has a large margin

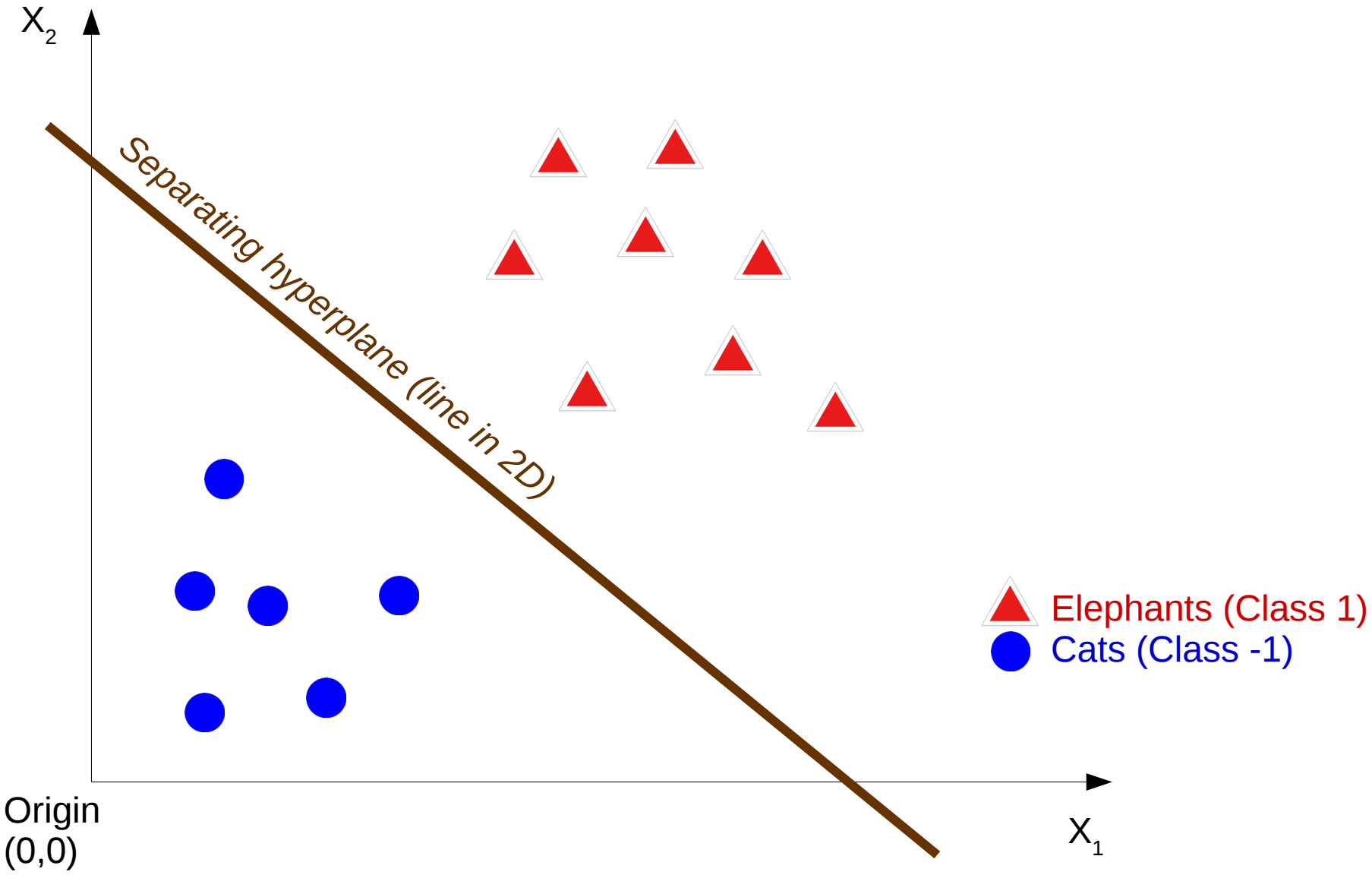


What makes this plane a good decision boundary?

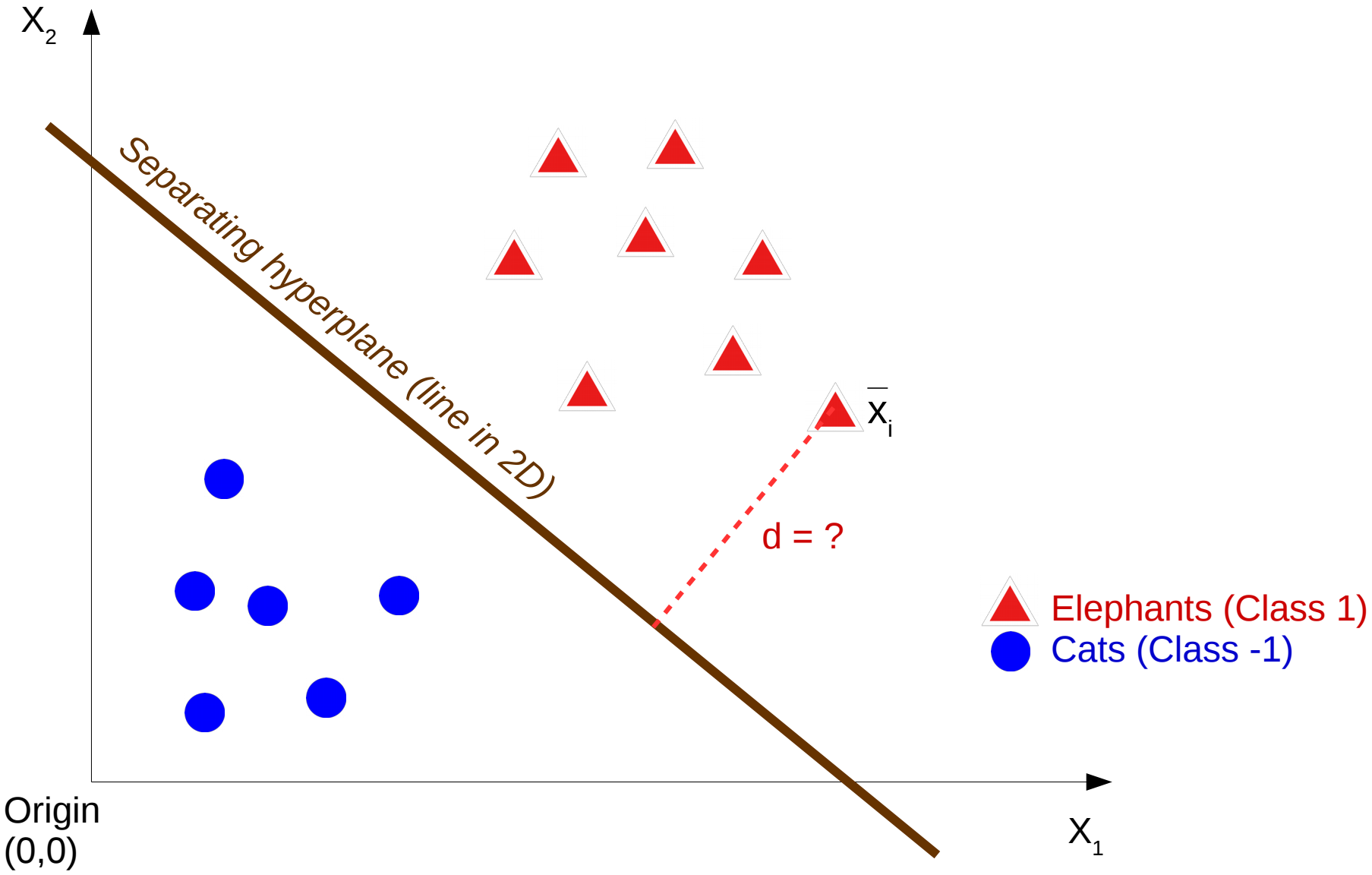
1. It classifies all the data correctly
2. It increases the "empty space" (margin)

→ we need to measure distances from boundary

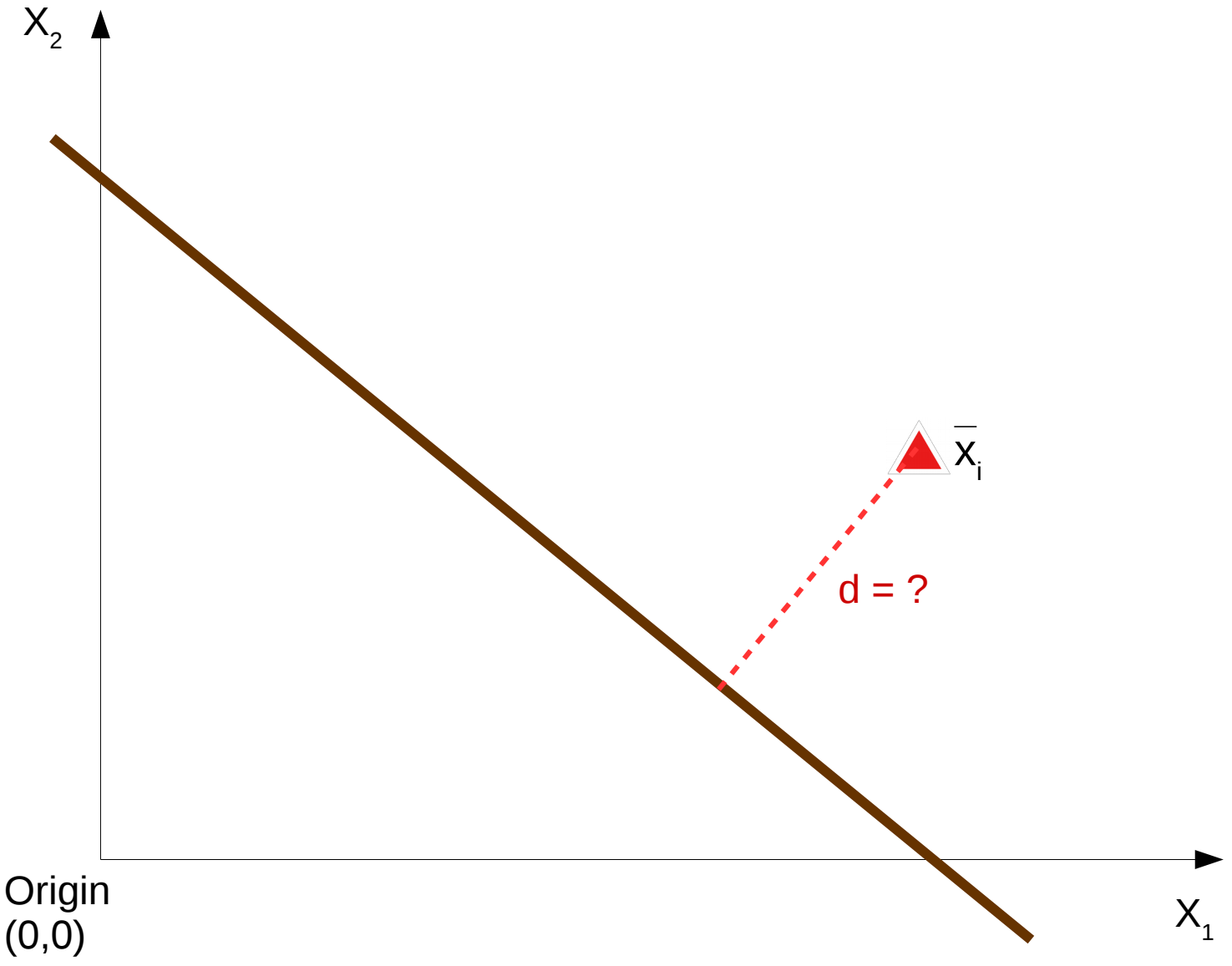
Distance of a point from a plane



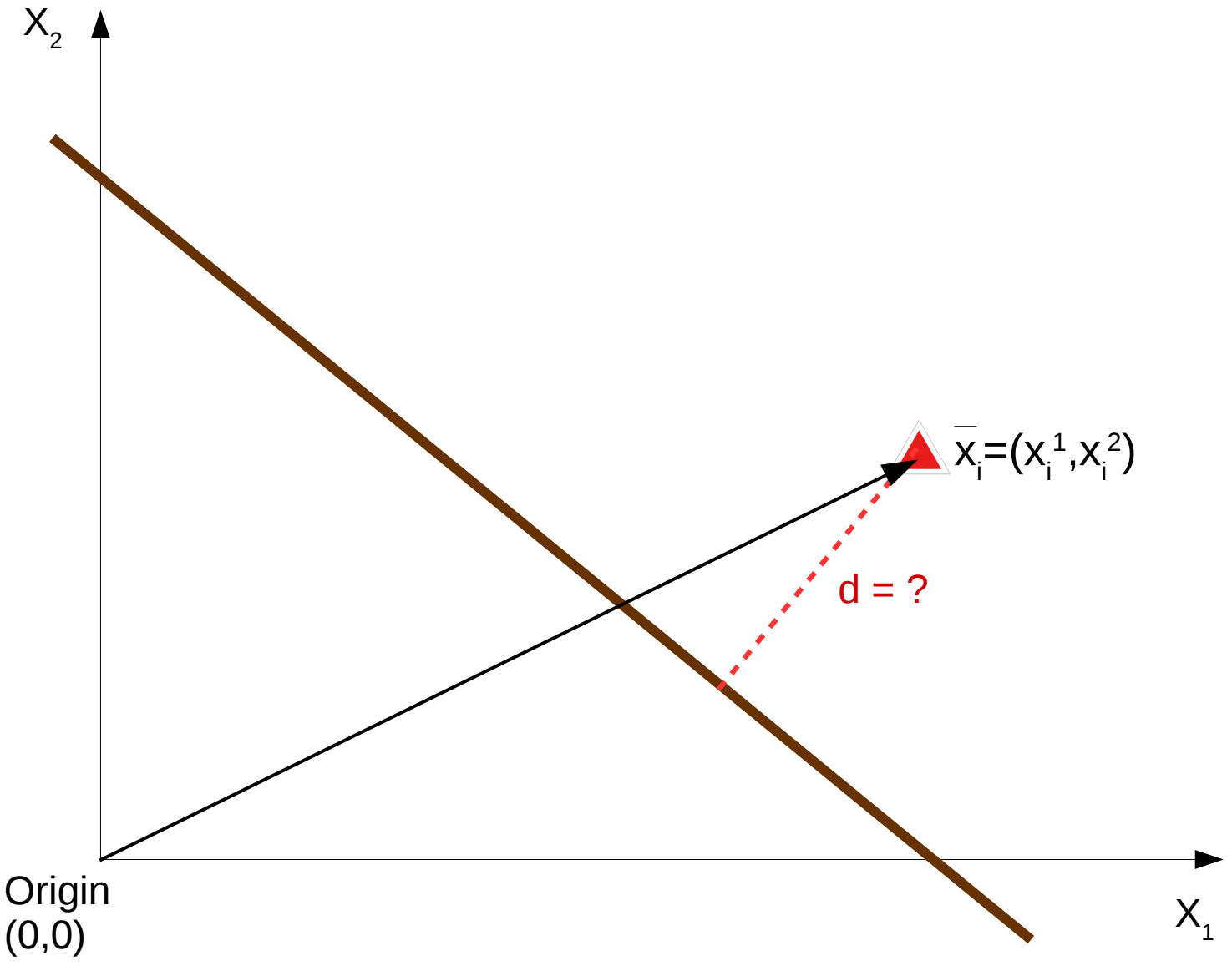
Distance of a point from a plane



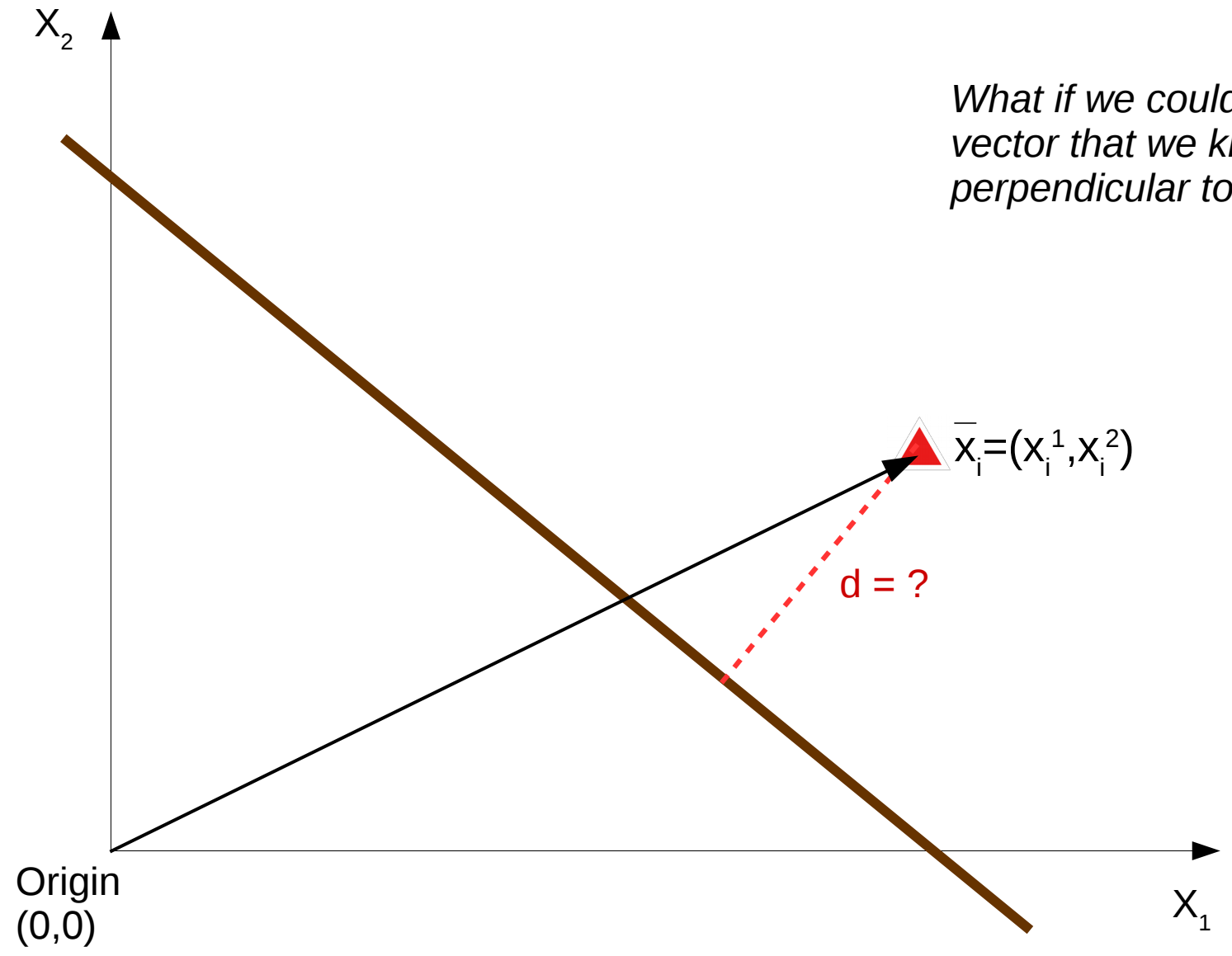
Distance of a point from a plane



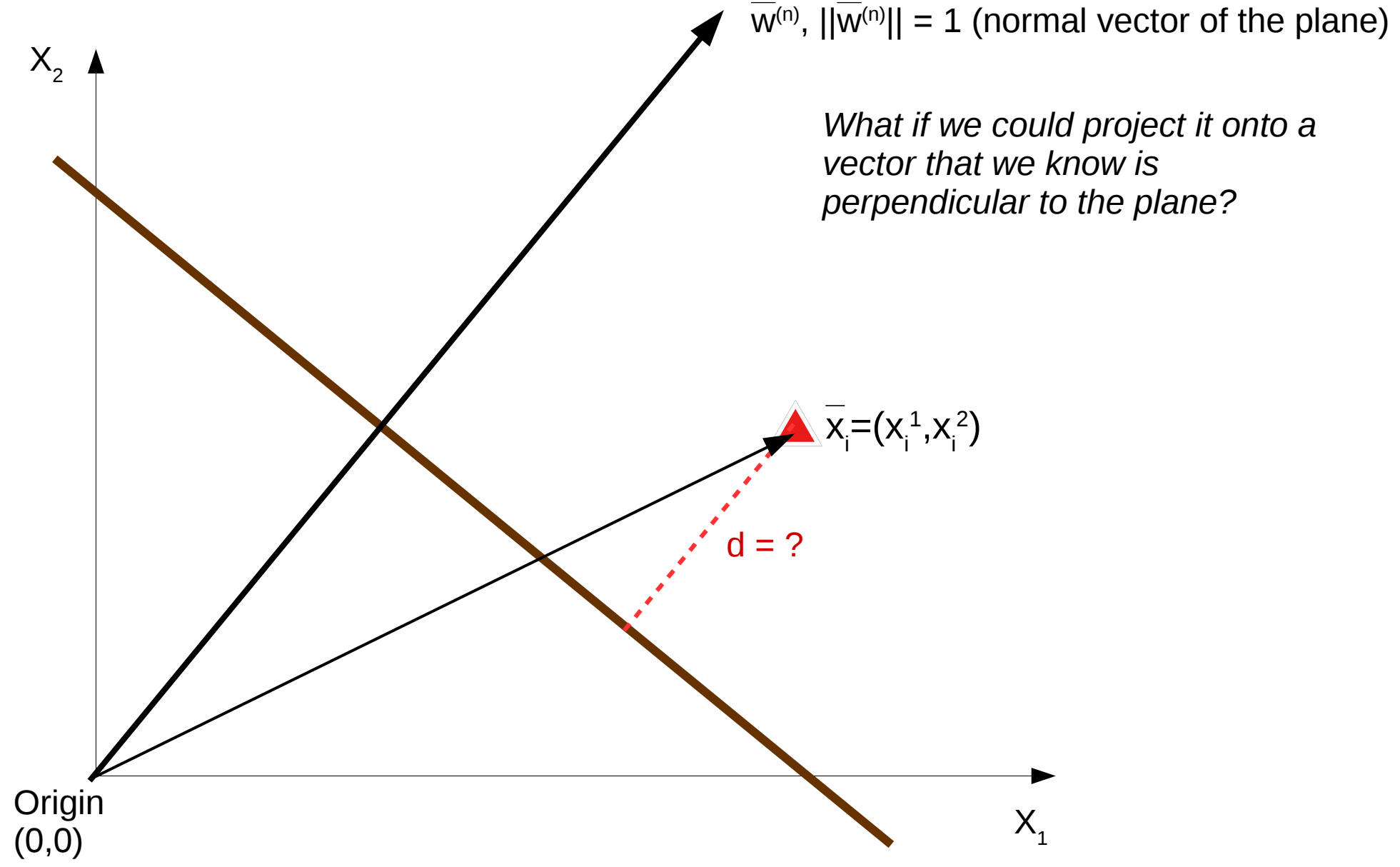
Distance of a point from a plane



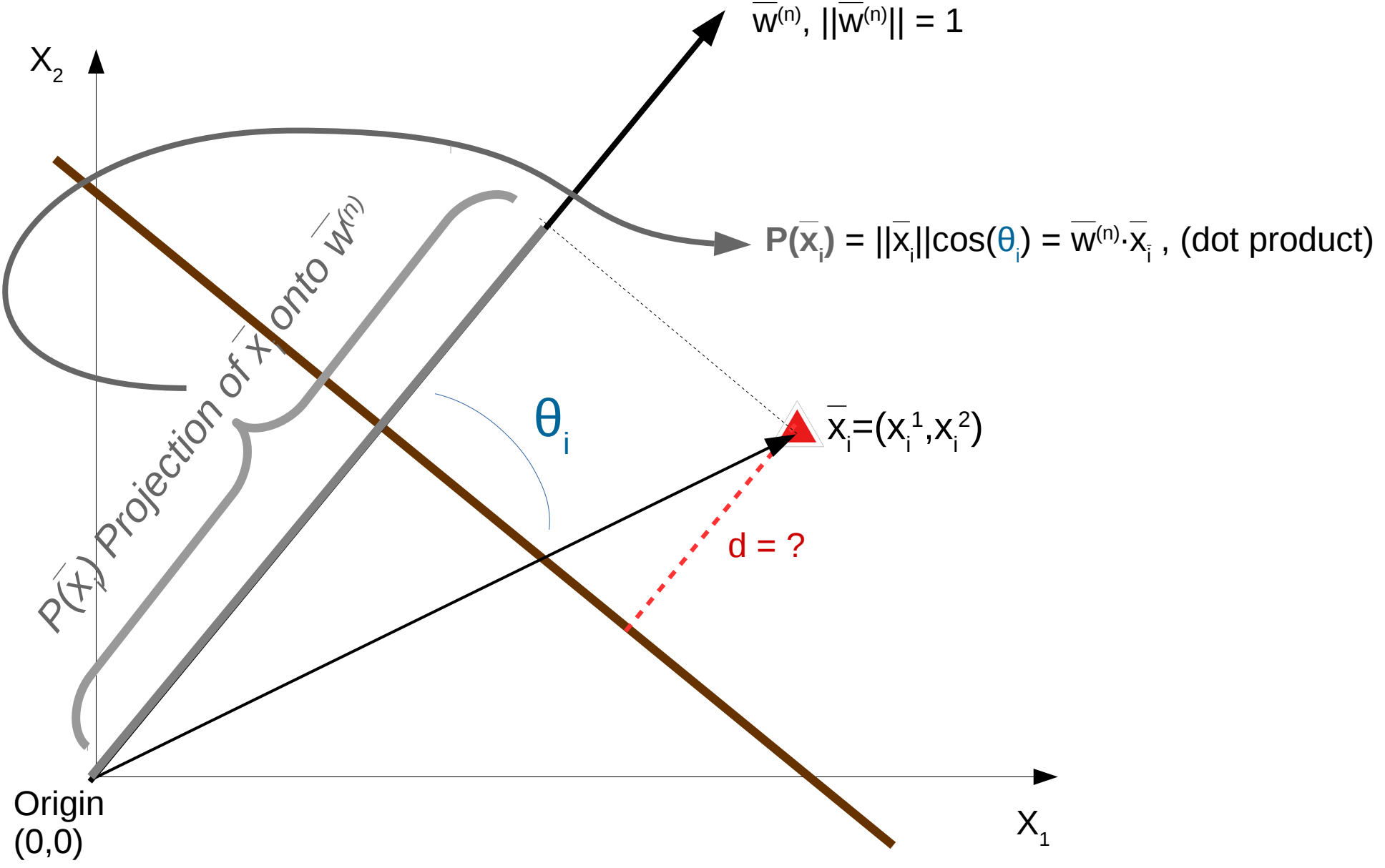
Distance of a point from a plane



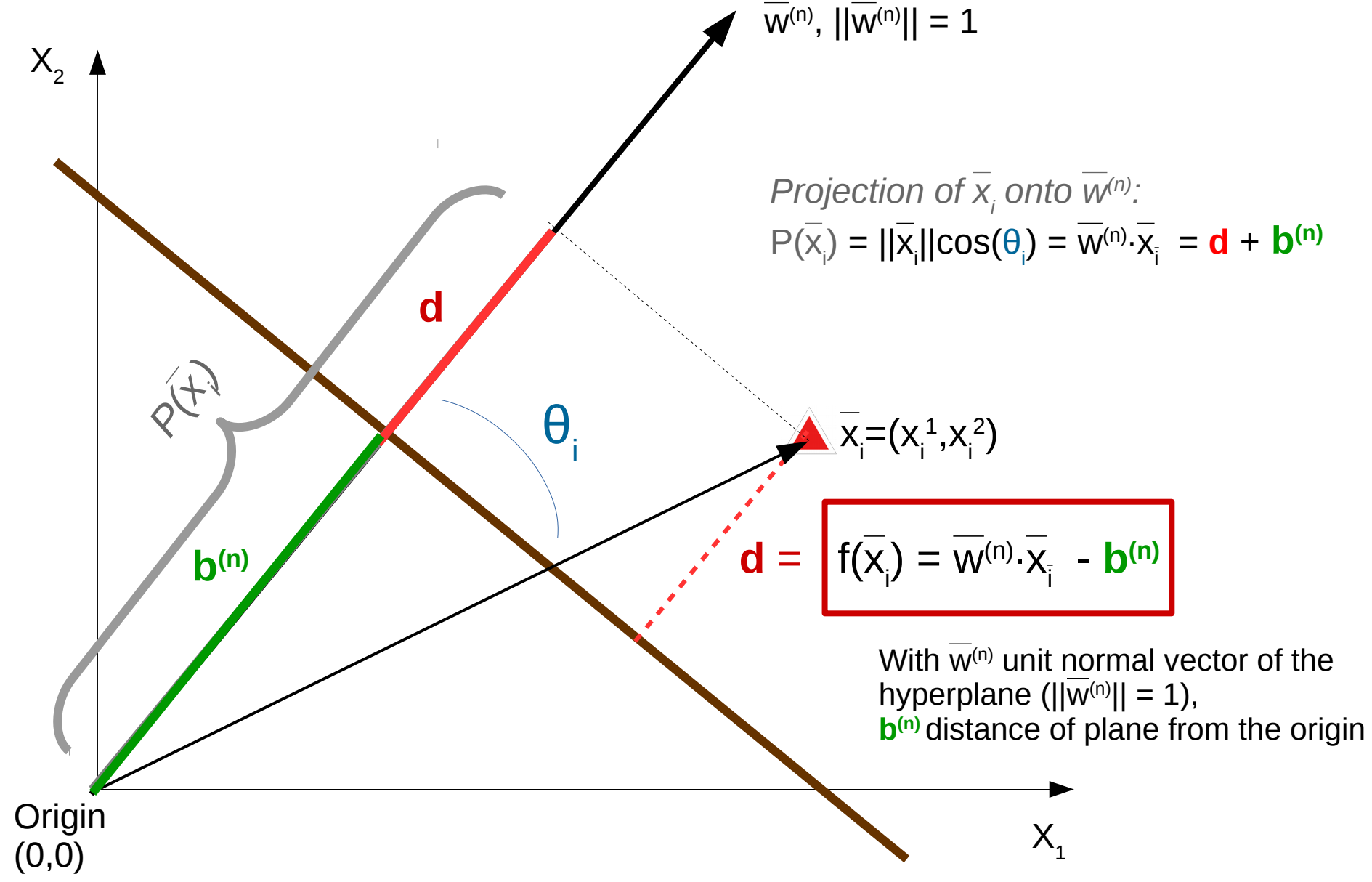
Distance of a point from a plane



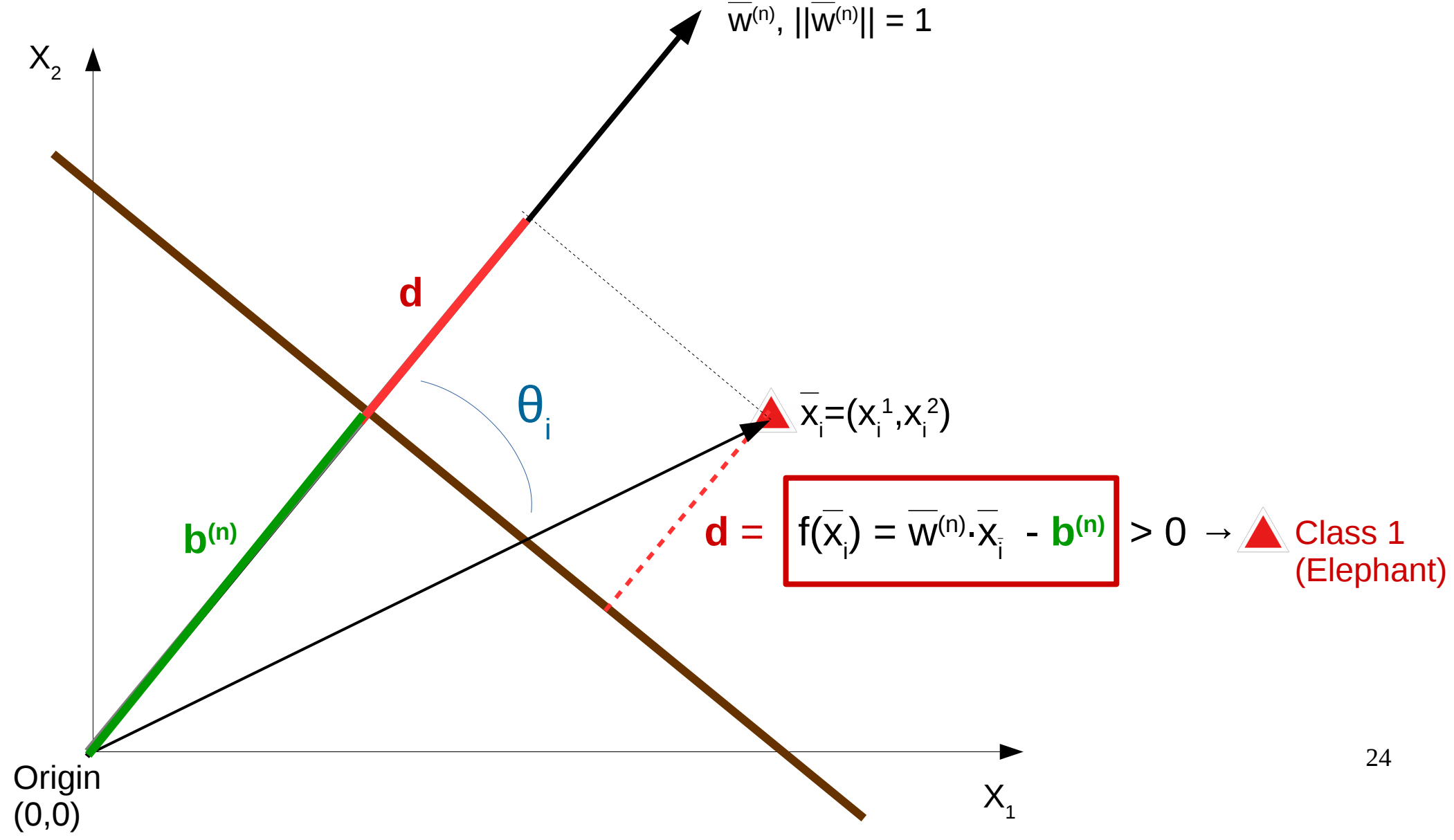
Distance of a point from a plane



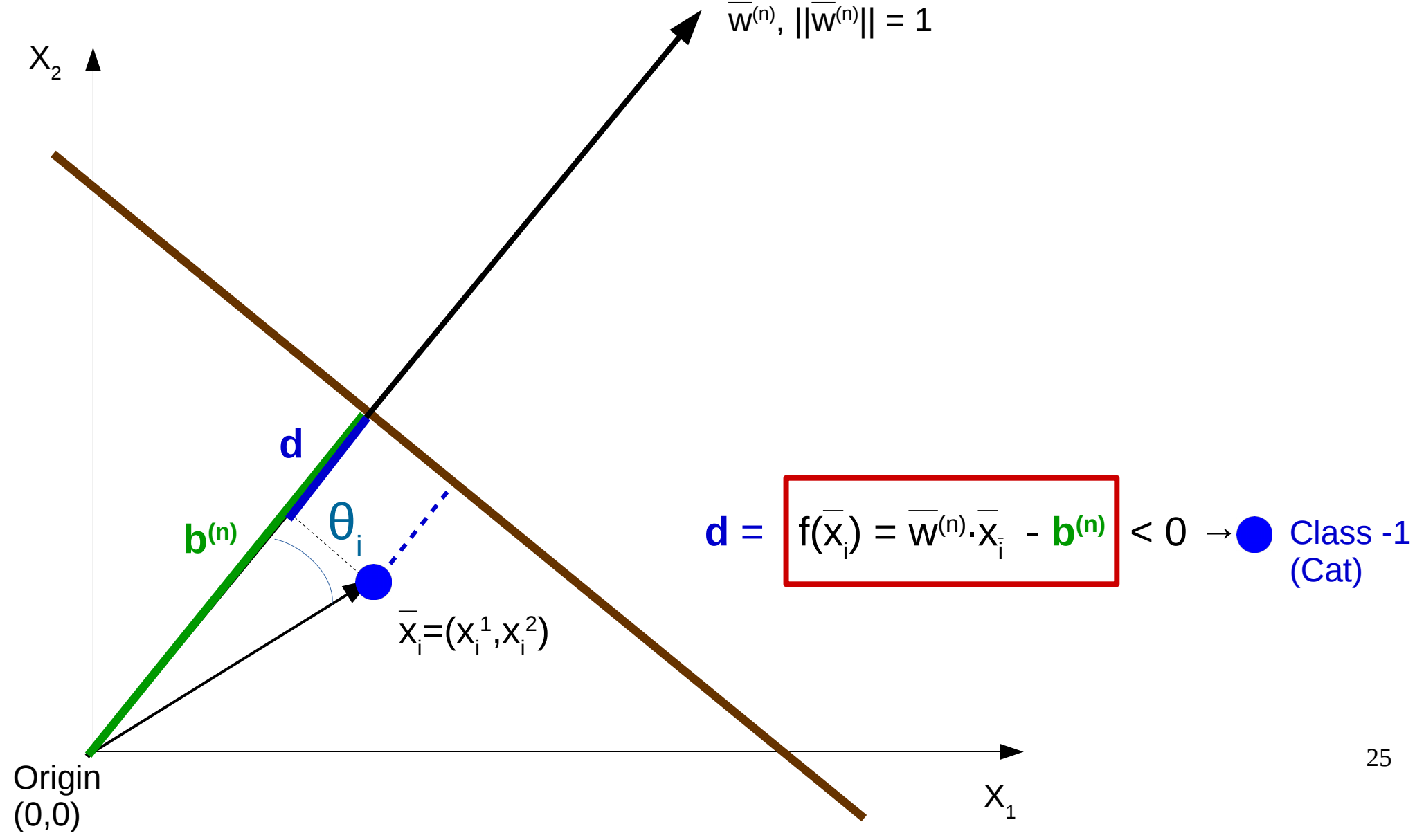
Distance of a point from a plane



Distance of a point from a plane

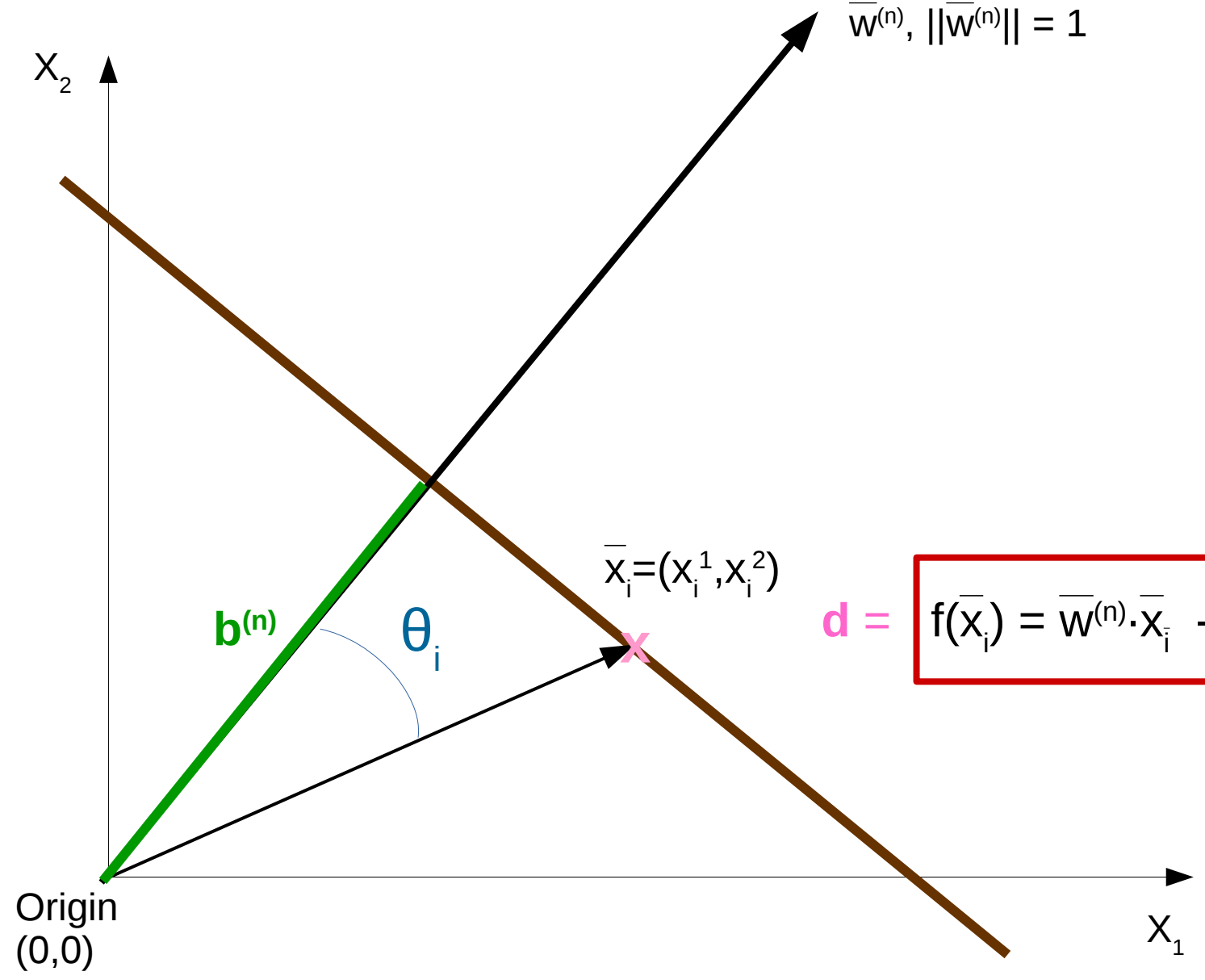


Distance of a point from a plane



d = $f(\bar{x}_i) = \bar{w}^{(n)} \cdot \bar{x}_i - \mathbf{b}^{(n)} < 0 \rightarrow$ Class -1 (Cat)

Distance of a point from a plane

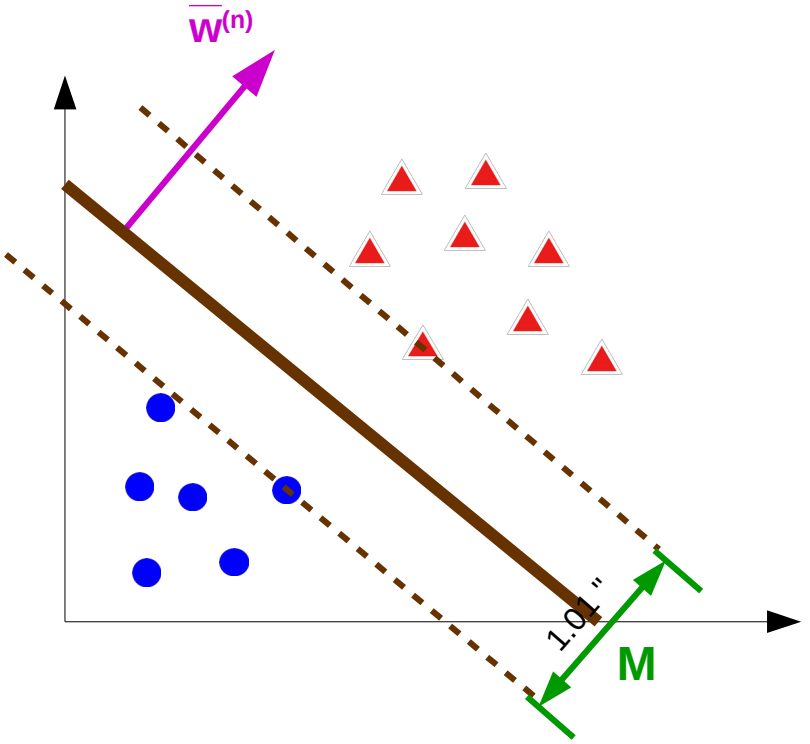


$$\mathbf{d} = \boxed{f(\bar{x}_i) = \bar{w}^{(n)} \cdot \bar{x}_i - \mathbf{b^{(n)}}} = 0 \rightarrow \text{Point lies on plane}$$

How to find the plane: 2 conditions

1. All triangles should be on one side of the *margin*, circles on the other (<u>empty</u> margin)

Classifier $f(\bar{x}_i) = \bar{w}^{(n)} \cdot \bar{x}_i - b^{(n)}$



2. Maximize margin width **M**

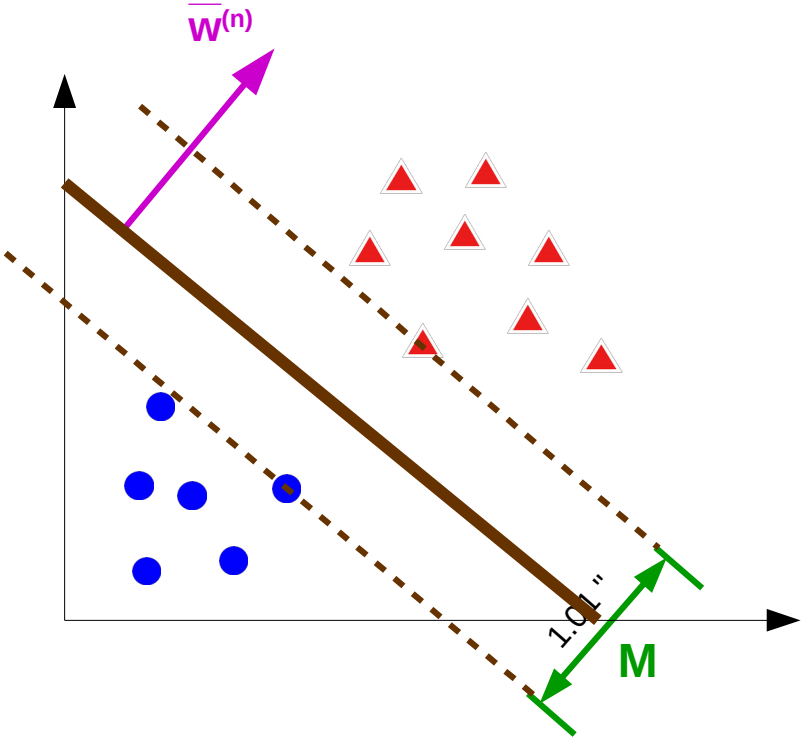
How to find the plane: 2 conditions

1. All triangles should be on one side of the *margin*, circles on the other (<u>empty</u> margin)

Classifier $f(\bar{x}_i) = \bar{w}^{(n)} \cdot \bar{x}_i - b^{(n)}$

a) $f(x_i) = \bar{w}^{(n)} \cdot \bar{x}_i - b^{(n)} \geq$ **M**/2 for all ▲

b) $f(x_i) = \bar{w}^{(n)} \cdot \bar{x}_i - b^{(n)} \leq$ -**M**/2 for all ●



2. Maximize margin width **M**

How to find the plane: 2 conditions

1. All triangles should be on one side of the *margin*, circles on the other (<u>empty</u> margin)

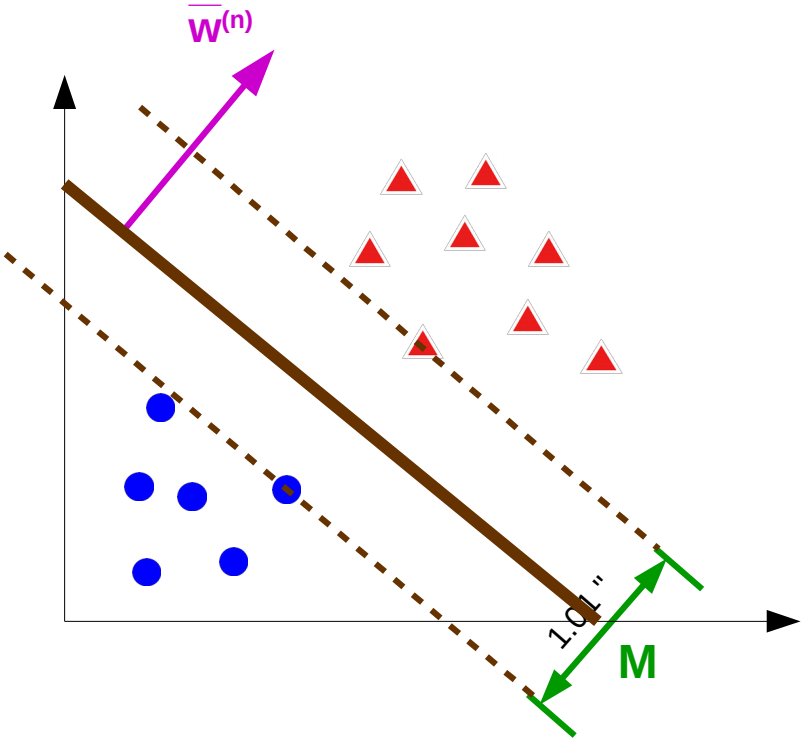
Classifier $f(\bar{x}_i) = \bar{w}^{(n)} \cdot \bar{x}_i - b^{(n)}$

a) $f(x_i) = \bar{w}^{(n)} \cdot \bar{x}_i - b^{(n)} \geq$ **M**/2 for all ▲

b) $f(x_i) = \bar{w}^{(n)} \cdot \bar{x}_i - b^{(n)} \leq$ -**M**/2 for all ●

→ $f^y(x_i) = y_i \cdot (\bar{w}^{(n)} \cdot \bar{x}_i - b^{(n)}) \geq$ **M**/2,

with y_i = +1 for ▲, -1 for ●



2. Maximize margin width **M**

How to find the plane: 2 conditions

1. All triangles should be on one side of the *margin*, circles on the other (empty margin)

a) $f(\bar{x}_i) = \bar{w}^{(n)} \cdot \bar{x}_i - b^{(n)} \geq \mathbf{M}/2$ for all ▲

b) $f(\bar{x}_i) = \bar{w}^{(n)} \cdot \bar{x}_i - b^{(n)} \leq -\mathbf{M}/2$ for all ●

→ $f^y(\bar{x}_i) = y_i \cdot (\bar{w}^{(n)} \cdot \bar{x}_i - b^{(n)}) \geq \mathbf{M}/2$,

with $y_i = +1$ for ▲, -1 for ●

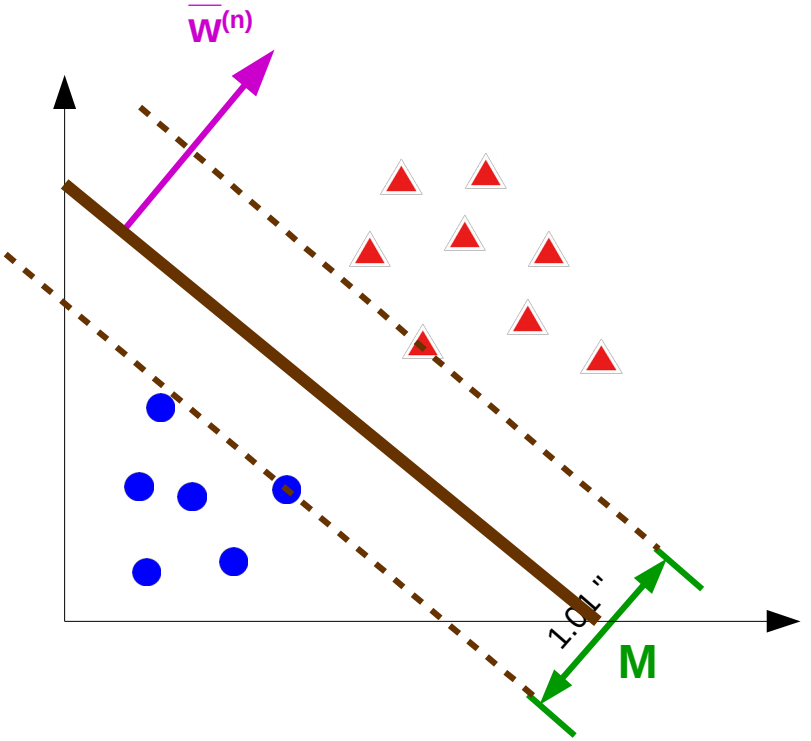
$= y_i \cdot (\bar{w} \cdot \bar{x}_i - b) \geq 1$,

with $\bar{w} = (2/M) \cdot \bar{w}^{(n)}$, $b = (2/M) \cdot b^{(n)}$

2. Maximize margin width **M**

→ Minimize $\bar{w} = (2/\mathbf{M}) \cdot \bar{w}^{(n)}$

Classifier $f(\bar{x}_i) = \bar{w}^{(n)} \cdot \bar{x}_i - b^{(n)}$



We need to solve a quadratic optimization problem

- To resume we want to solve two problems:
  - (A) Find w, b such that $y_i(\vec{w}\cdot\vec{x}_i - b) \geq 1$ for all $\vec{x}_i$
  - (B) Maximize margin M by minimizing $||\vec{w}||$
- This is equivalent to:

$$min_{\vec{w}} \|\vec{w}\|^2, \text{ subject to } y_i \cdot (\vec{w}\cdot\vec{x}_i - b) \geq 1, \text{ for } i = 1, \ldots, N$$

- This is a convex quadratic optimization problem

  (any local optimum is also the global optimum)

The classifier is a linear combination of dot products with the training set

- Solving the optimization problem (...*skipping lots of math here!*):

$$\vec{w} = \sum_{i=1}^{N} \alpha_i \, y_i \, \vec{x}_i, \text{ with weights } \alpha_i \geq 0$$

Linear combination of training set

This all happens during the fitting step

Python: svc.fit(training_data, training_labels)

- → Classifier $f(\vec{x}) = \vec{w} \cdot \vec{x} - b = (\sum_{i=1}^{N} \alpha_i \, y_i \, \vec{x}_i) \cdot \vec{x} - b = \sum_{i=1}^{N} \alpha_i \, y_i (\vec{x}_i \cdot \vec{x}) - b$

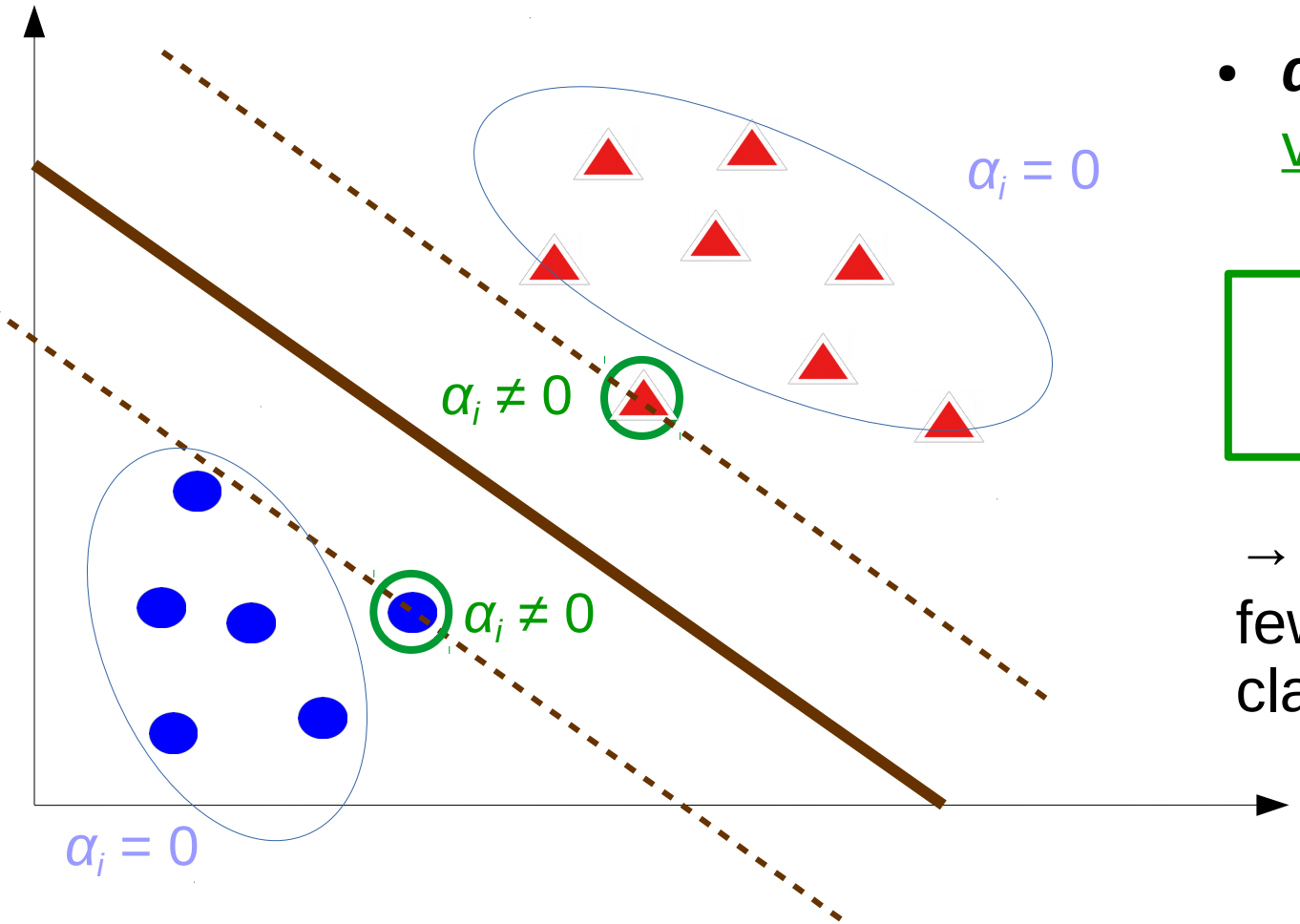
For any new point $\bar{x}$ to classify

→ linear combination of dot products with the training set!

Python: class_prediction = svc.predict(unlabeled_data)

- It gets better! (next slide)

The classifier depends only on support vectors



- $\boldsymbol{\alpha_i = 0}$ except support vectors (*on margin border*)

$$f(\vec{x}) = \sum_{i=1}^{S \ll N} \alpha_i y_i (\vec{x}_i \cdot \vec{x}) - b$$

→ Only dot product with a few points needed to classify a new point

Support vector machines

1) What is a classifier supposed to do and why is SVM a good one?

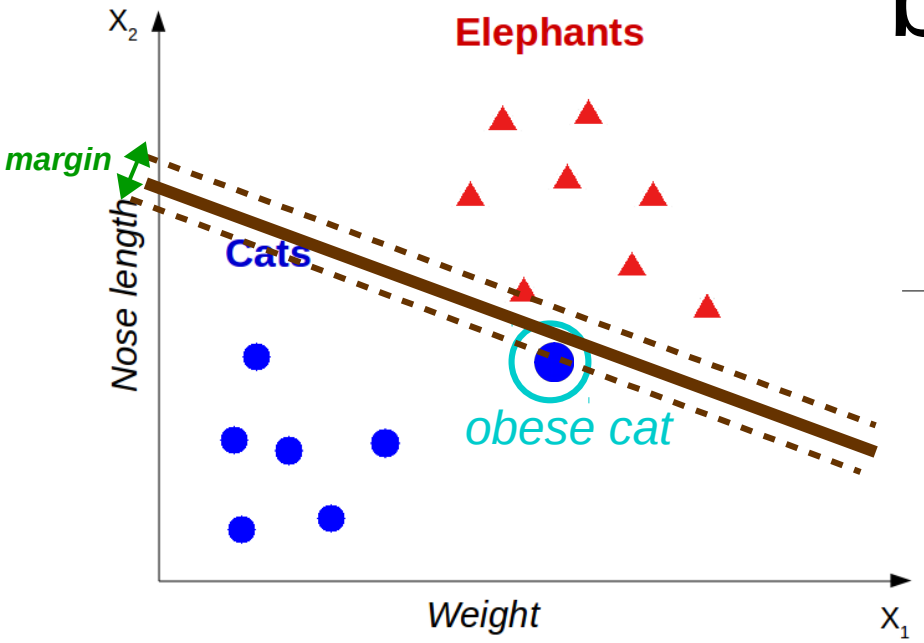
2) Establish SVM methodology by walking through a simple case

3) Generalize the SVM to make it more robust

4) A simple trick allows us to create a very complex classifier

5) Practical tips for using SVM

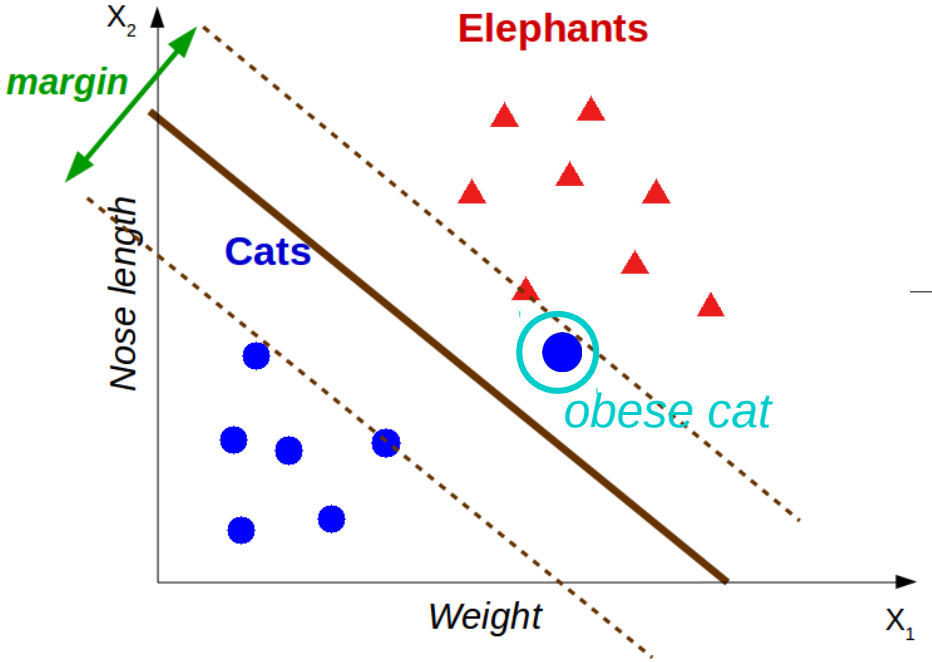
Is a strict (“hard”) margin always the best?



$y_i(\bar{w} \cdot \bar{x}_i - b) \geq 1$ for all $\bar{x}_i$

Satisfied for all training points

BUT <u>margin small</u>!



$y_i(\bar{w} \cdot \bar{x}_i - b) \geq 1$ for all $\bar{x}_i$

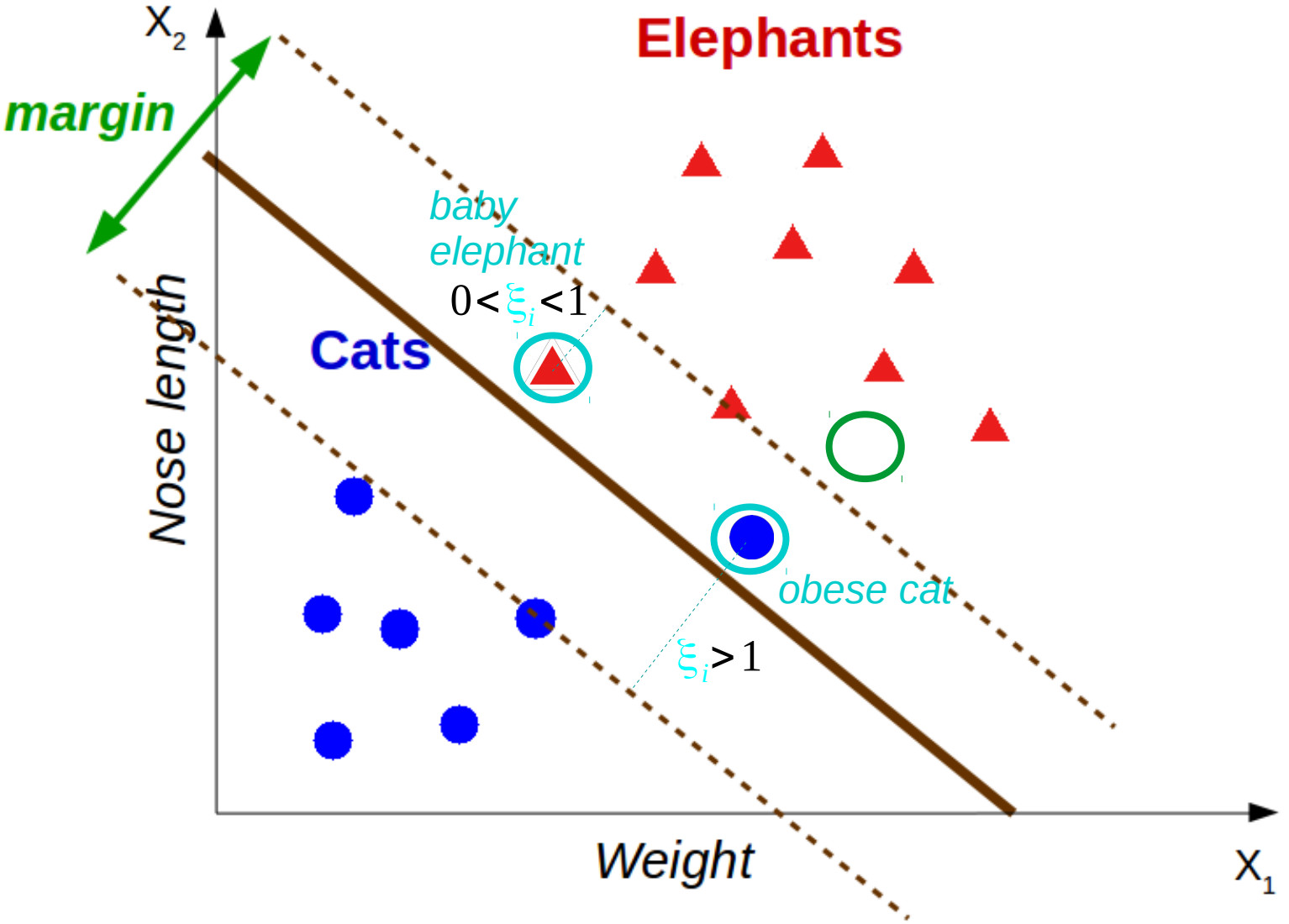
NOT satisfied for all x_i

BUT <u>margin large</u>!

How can we leave some slack?

A “Soft” margin is not completely empty

$$y_i\left(\vec{w}\cdot\vec{x}_i-b\right)\geq 1 \longrightarrow y_i\left(\vec{w}\cdot\vec{x}_i-b\right)\geq 1-\xi_i$$



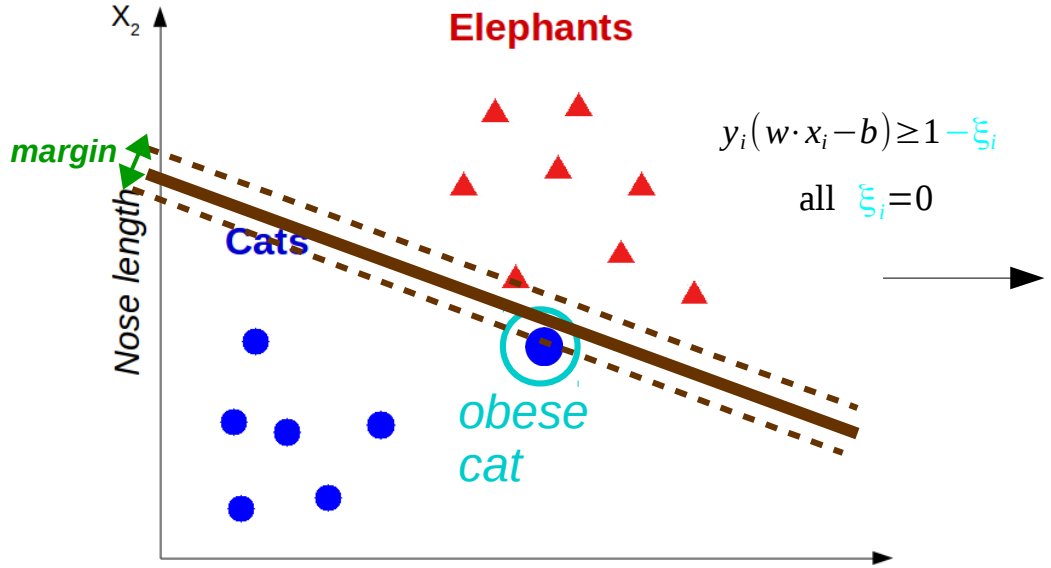
Soft margin

- Optimization becomes:

$$min_{\vec{w},\xi_i} \|\vec{w}\|^2 + C\sum_{i=1}^{N} \xi_i, \text{ subject to } y_i \cdot (\vec{w}\cdot\vec{x}_i - b) \geq 1 - \xi_i, \text{ for } i = 1, \ldots, N$$

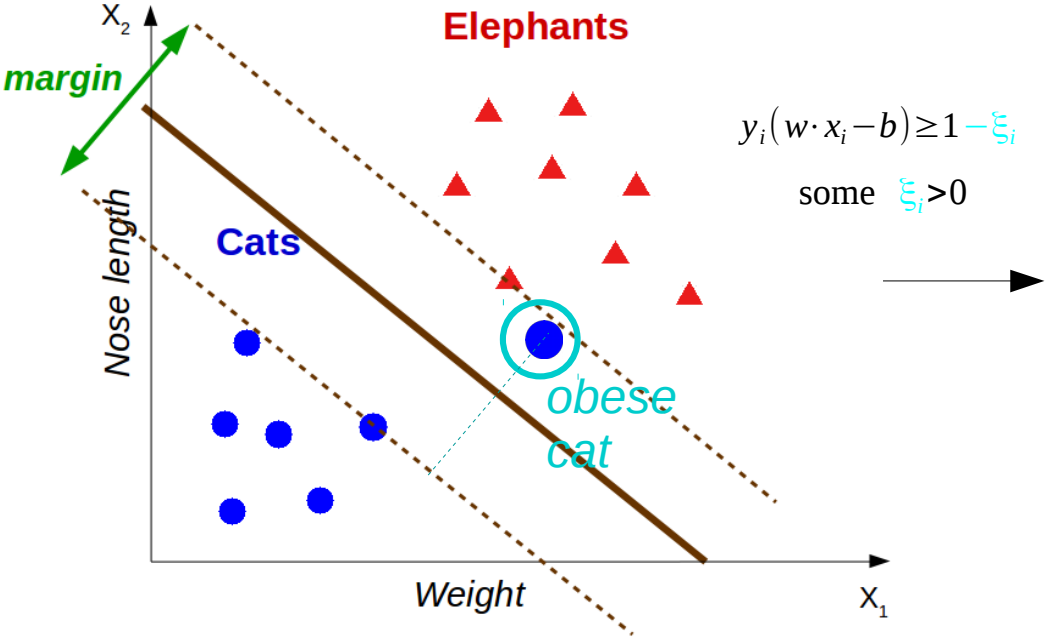
- Still convex → can be solved efficiently
- C controls trade-off between making margin large and keeping margin empty (tuning param)
- C plays the role of $1/\lambda$ from Linear/Logistic regression (Ridge/Lasso)

Regularization parameter C controls bias/variance trade-off



$$min_w \|\vec{w}\| + C \sum_{i=1}^{N} \xi_i$$

C large, points <u>not allowed</u> in margin
(low bias, high variance)
Less robust
(more sensitive to outliers)



C small, points <u>allowed</u> in margin
(high bias, low variance)
More robust
(less sensitive to outliers)

Support vector machines

1) What is a classifier supposed to do and why is SVM a good one?

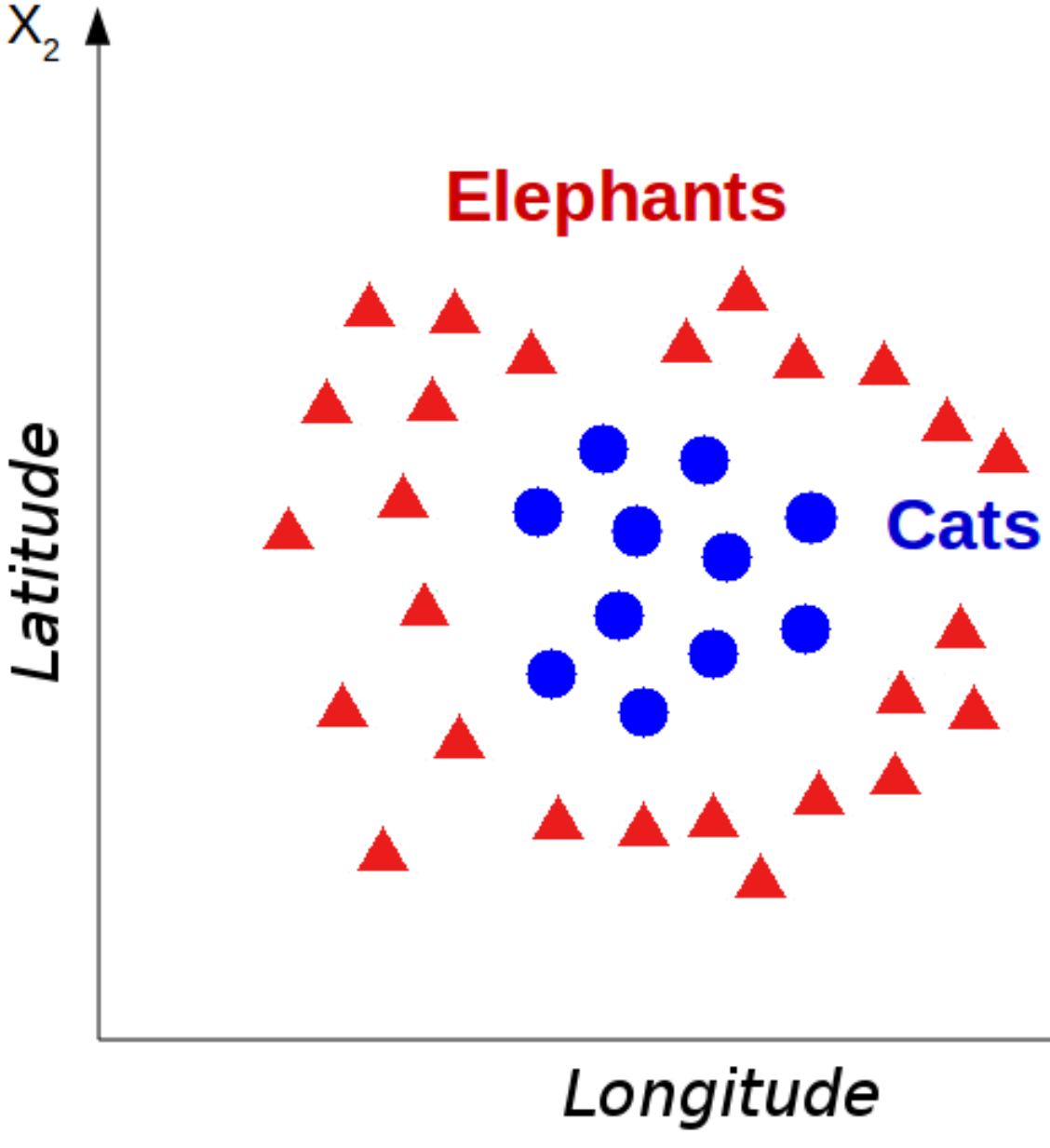
2) Establish SVM methodology by walking through a simple case

3) Generalize the SVM to make it more robust

4) A simple trick allows us to create a very complex classifier

5) Practical tips for using SVM

What if data not linearly separable?

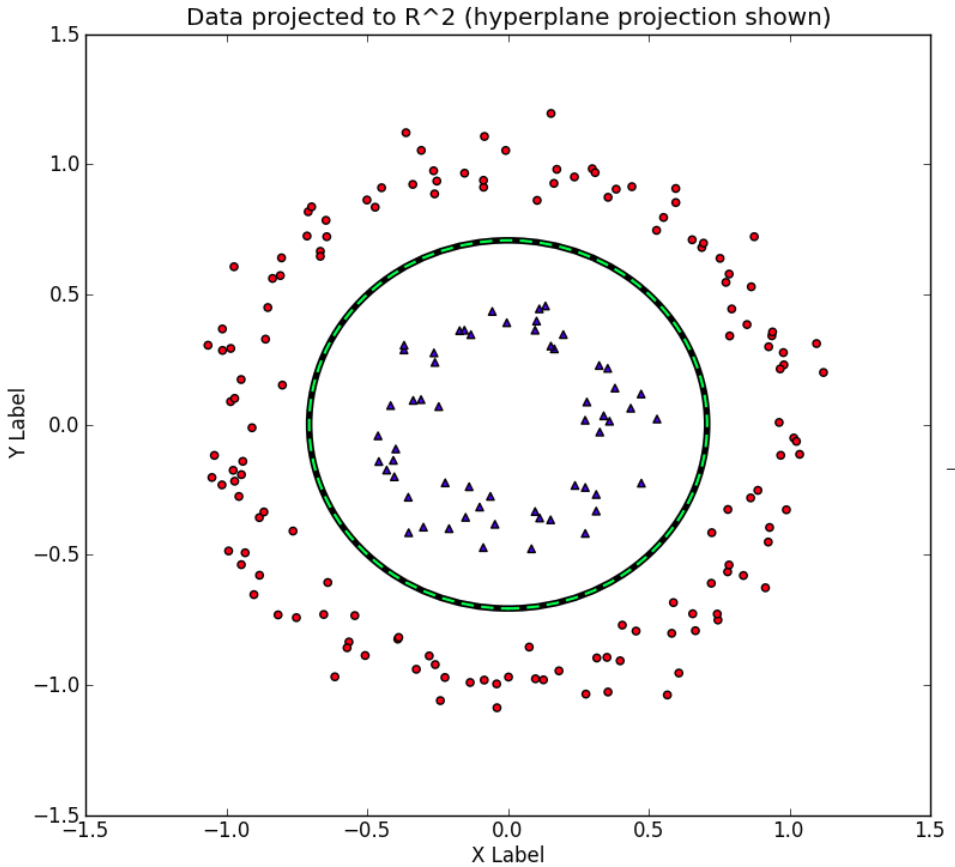


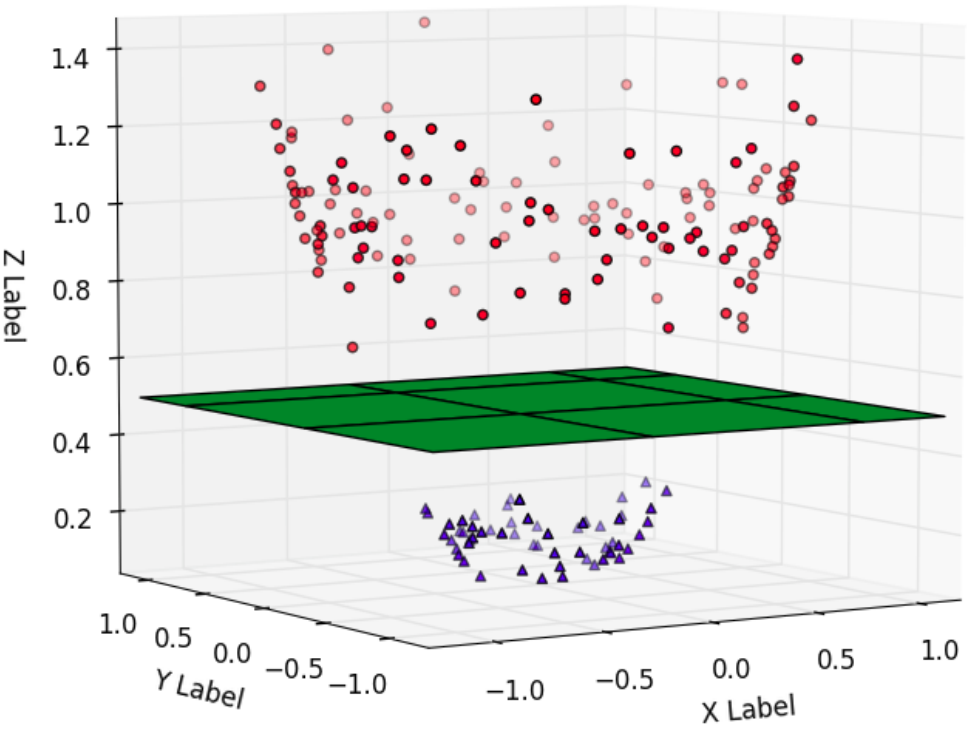
Less elephants live in cities

Would need to collect more data to separate classes (*e.g.* nose lengths), but might not be available!

→ Design new features with original features (*e.g.* $X_3 = X_1^2 + X_2^2$ – distance from city center)

Data will become linearly separable in higher dimension





- Extra dimensions could come from completely new features or features designed from old features (like radius $z = x^2 + y^2$)

Feature mapping to higher dimensions can be expensive

$$Classifier\, f(\vec{x}) = \sum_{i=1}^{N} \alpha_i\, y_i (\vec{x}_i \cdot \vec{x}) - b$$

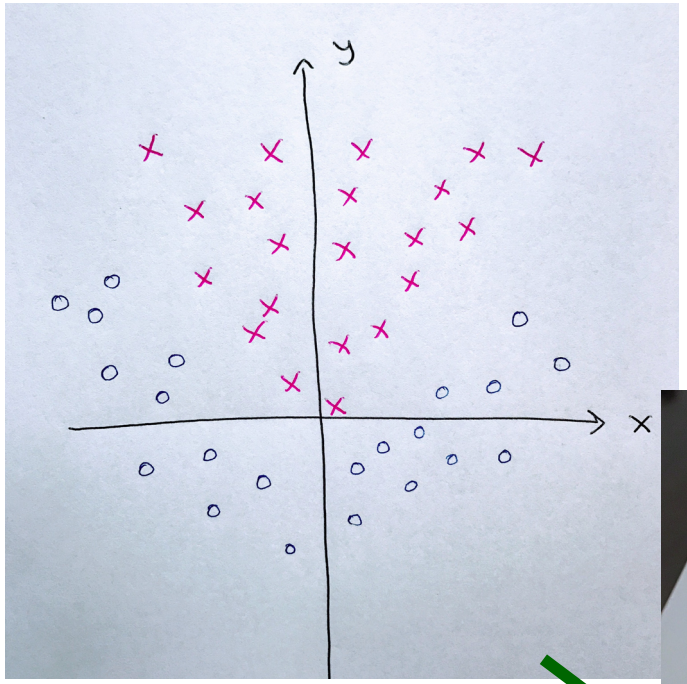
- Add higher order/derivative terms x_1^2, x_2^3, $log(x_3)$ ...
- The classifier becomes

$$f(\vec{x}) = \sum_{i=1}^{N} \alpha_i\, y_i (\phi(\vec{x}_i) \cdot \phi(\vec{x})) - b$$

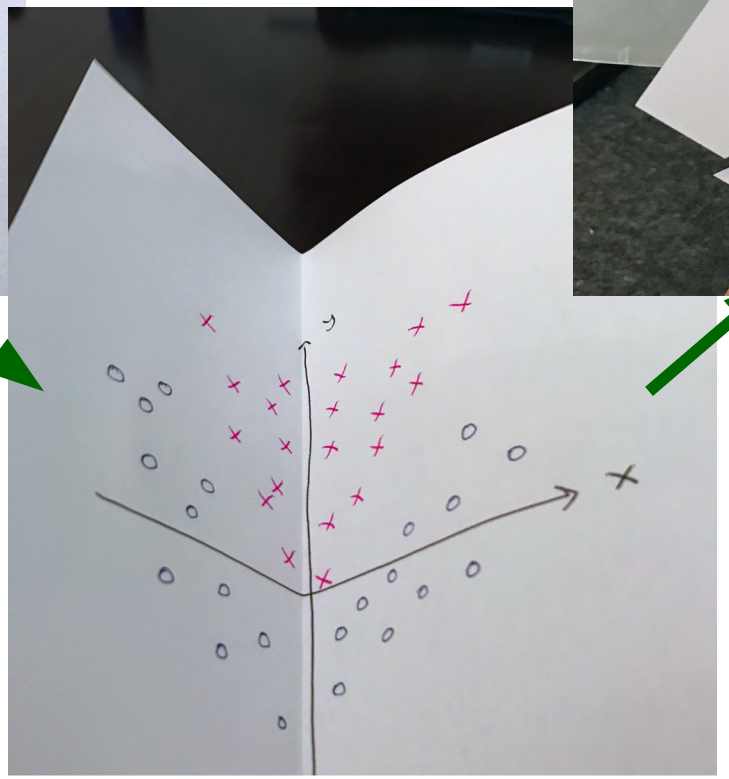
$$e.g.\ \ \phi_{quad}(\vec{x}) = \phi_{quad} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} x_1 x_1 \\ x_1 x_2 \\ x_1 x_3 \\ x_2 x_1 \\ x_2 x_2 \\ x_2 x_3 \\ x_3 x_1 \\ x_3 x_2 \\ x_3 x_3 \end{pmatrix}$$

- With Φ(x) “feature mapping function”
- Manually designing these terms can be tedious
- → Can get very large and computationally expensive

Cool feature mapping demonstration



$\Phi_{3D}(x)$



$\Phi_{2D}(x)$

Linear cut in 3D becomes non-linear cut in 2D

SVM allows us to avoid explicit feature mapping

$$f(x) = \sum_{i=1}^{N} \alpha_i y_i (\phi(\vec{x}_i) \cdot \phi(\vec{x})) - b = \sum_{i=1}^{N} \alpha_i y_i K(\vec{x}_i, \vec{x}) - b$$

- $K(\bar{a}, \bar{b})$ is called the kernel function
- *K* replaces dot product as similarity measure
- Calculating *K* often much less expensive than expanding features through Φ(x)
- Examples on next slide

Example: quadratic kernel equivalent to feature mapping to n^2 dimensions

Classifier $f(x)=\sum_{i=1}^{N}\alpha_i y_i(\phi(\vec{x}_i)\cdot\phi(\vec{x}))-b=\sum_{i=1}^{N}\alpha_i y_i K(\vec{x}_i,\vec{x})-b$

$$e.g.\ K_{quad}(\vec{a},\vec{b})=(\vec{a}\cdot\vec{b})^2=\left(\sum_j a_j b_j\right)\left(\sum_k a_k b_k\right)=\sum_j\sum_k a_j a_k b_j b_k=\sum_{j,k}(a_j a_k)(b_j b_k)$$

Results of $K_{quad}(\bar{a},\bar{b})$ is identical to take dot product of feature expanded vectors!

$$\phi_{quad}(\vec{a})\cdot\phi_{quad}(\vec{b})=\begin{pmatrix}a_1a_1\\a_1a_2\\a_1a_3\\a_2a_1\\a_2a_2\\a_2a_3\\a_3a_1\\a_3a_2\\a_3a_3\end{pmatrix}\cdot\begin{pmatrix}b_1b_1\\b_1b_2\\b_1b_3\\b_2b_1\\b_2b_2\\b_2b_3\\b_3b_1\\b_3b_2\\b_3b_3\end{pmatrix}\longrightarrow K(\vec{a},\vec{b})=(\vec{a}\cdot\vec{b})^2=\left(\begin{pmatrix}a_1\\a_2\\a_3\end{pmatrix}\cdot\begin{pmatrix}b_1\\b_2\\b_3\end{pmatrix}\right)^2$$

Kernel function gets to same result much more efficiently!!!

More general: polynomial kernel implicitly expands to n^d dimensions

$$K_{poly}(\vec{a}, \vec{b}) = (\vec{a} \cdot \vec{b} + c)^d$$

- Kernel function allows us to only care about the original features and efficiently move to higher dimension ***d*** in an implicit way
- Computing polynomial kernel takes O(n) time
- Real feature expansion would take $O(n^{d})$

  (n features, expansion of degree ***d***)
- Polynomial degree ***d*** is a tuning parameter

  (large ***d*** → overfitting)

Gaussian kernel replaces dot product with new similarity metric

$$Classifier\, f(\vec{x}) = \sum_{i=1}^{N} \alpha_i y_i (\vec{x}_i \cdot \vec{x}) - b = \sum_{i=1}^{N} \alpha_i y_i K(\vec{x}_i, \vec{x}) - b$$

- Dot product measures “similarity” of two vectors
- Idea: measure similarity as “smeared” euclidean distance

$$K_{gauss}(\vec{a}, \vec{b}) = \exp\left(-\frac{\|\vec{a} - \vec{b}\|^2}{2\sigma^2}\right), \qquad 0 < K_{gauss}(\vec{a}, \vec{b}) < 1$$

- “How close is new point to all other points of a class” (+1 for each point we are very close to)
- → Predict class with many close neighbors!

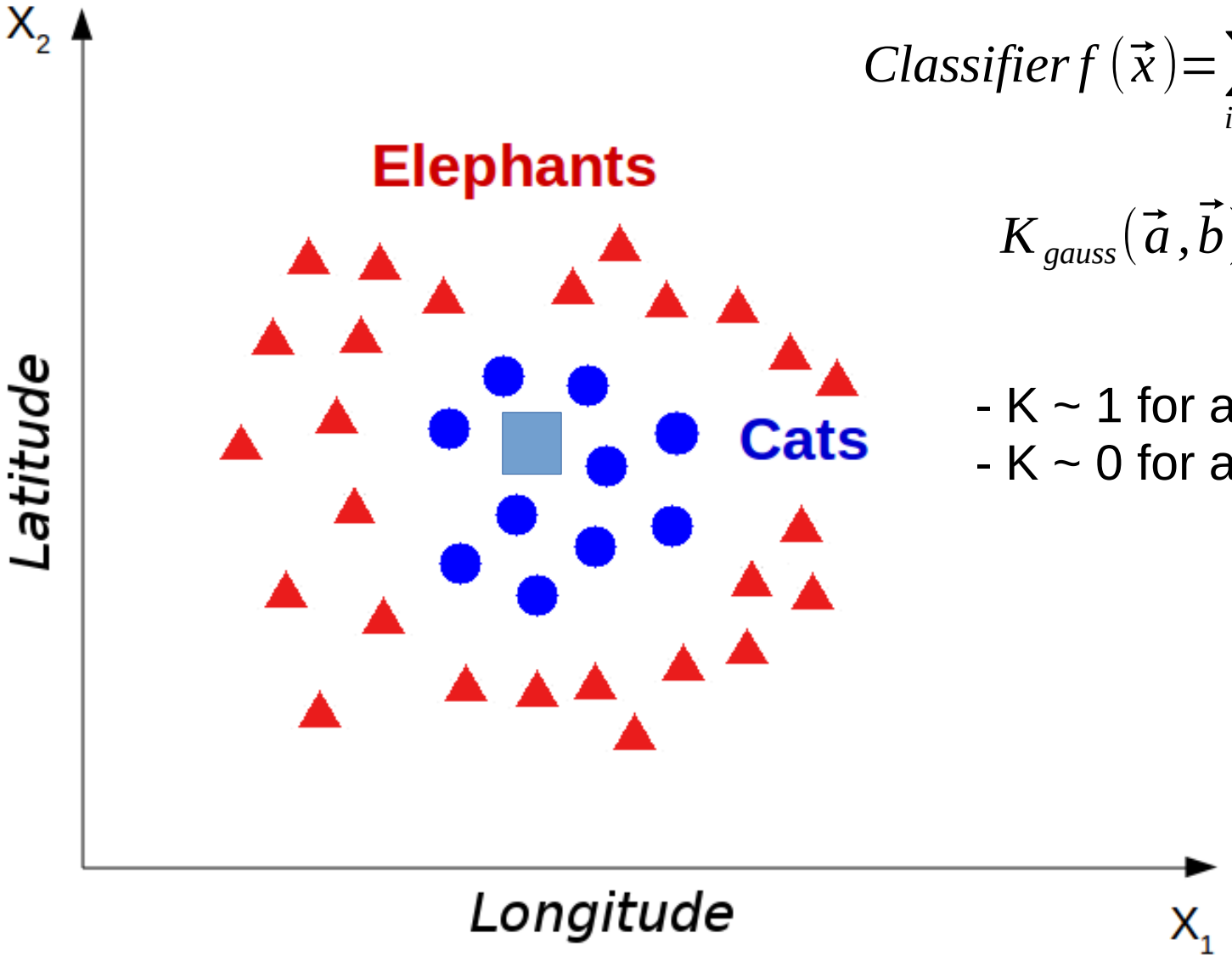
Gaussian kernel visualization: cats in cities

We want to classify new animal

$$Classifier\, f(\vec{x}) = \sum_{i=1}^{N} \alpha_i y_i K(\vec{x}_i, \vec{x}) - b$$

$$K_{gauss}(\vec{a}, \vec{b}) = \exp\left(-\frac{\|\vec{a} - \vec{b}\|^2}{2\sigma^2}\right)$$

- K ~ 1 for animals that are close
- K ~ 0 for animals that are far

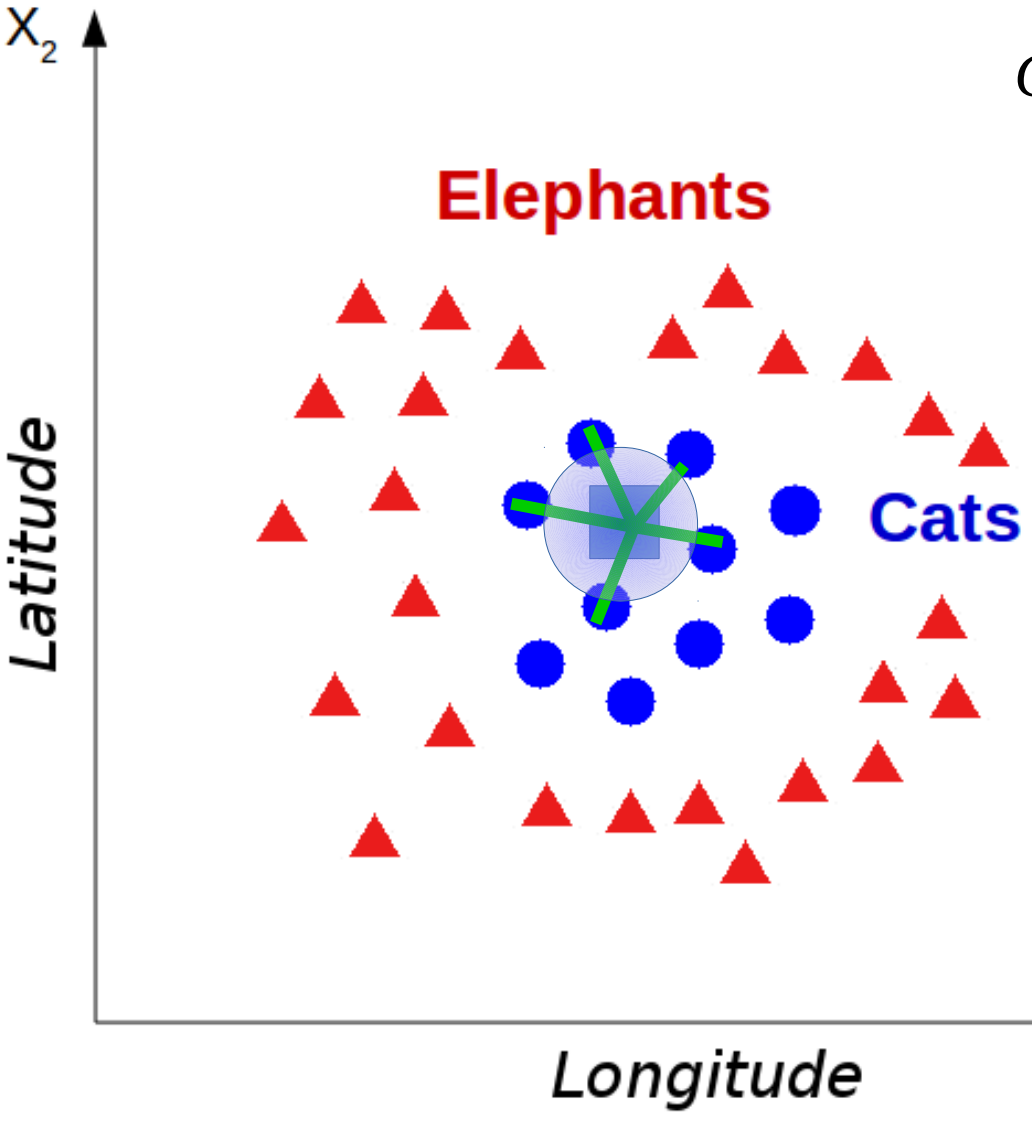


Gaussian kernel visualization: counting neighbors

We want to classify new animal

$$Classifier\, f(\vec{x}) = \sum_{i=1}^{N} \alpha_i y_i K(\vec{x}_i, \vec{x}) - b$$

$$K_{gauss}(\vec{a}, \vec{b}) = \exp\left(-\frac{\|\vec{a} - \vec{b}\|^2}{2\sigma^2}\right)$$



- K ~ 1 for animals that are close
- K ~ 0 for animals that are far

New animal gets score of ~5 for “cat”, score of ~0 for “elephant”
→ classify as “cat”

Represents “smearing” factor σ (how far out to “look”)

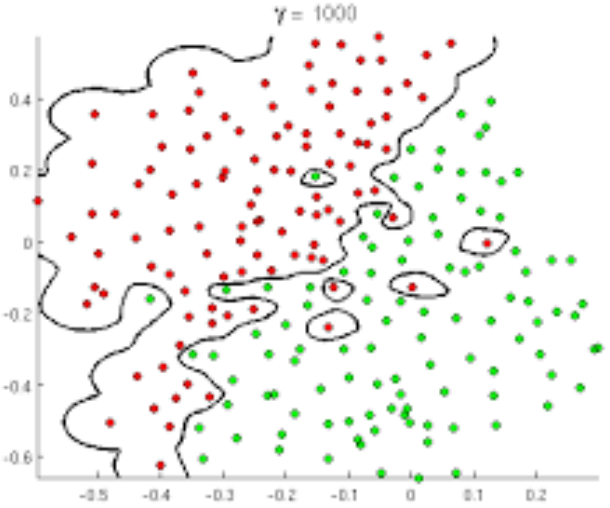
Gaussian kernel behavior

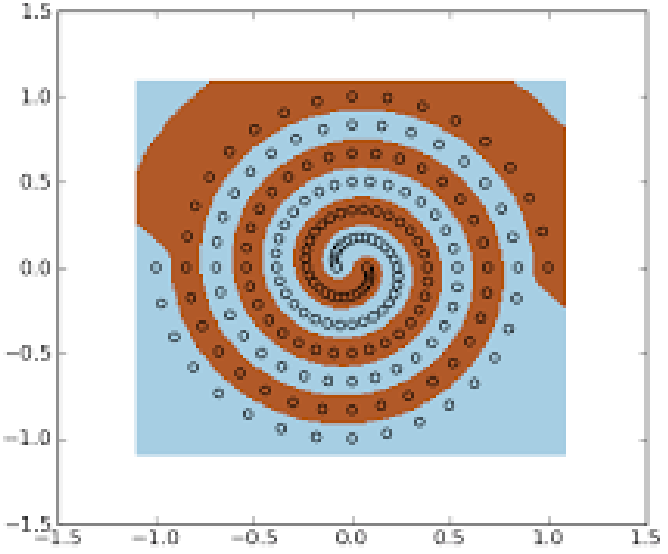
$$f(x) = \sum_{i=1}^{N} \alpha_i y_i K(\vec{x}_i, \vec{x}) - b$$

$$K_{gauss}(\vec{a}, \vec{b}) = \exp\left(-\frac{\|\vec{a} - \vec{b}\|^2}{2\sigma^2}\right) = \exp\left(-\gamma \|\vec{a} - \vec{b}\|^2\right)$$

- Feature space becomes implicitly infinite dimensional (distance to every point now treated as a feature)
- Feature scaling very important: Distance between 2 points dominated by 1 feature → Feature space becomes effectively 1D again
- $\gamma = 1/(2\sigma^2)$ = is a tuning parameter (similar to C or d for poly kernel)

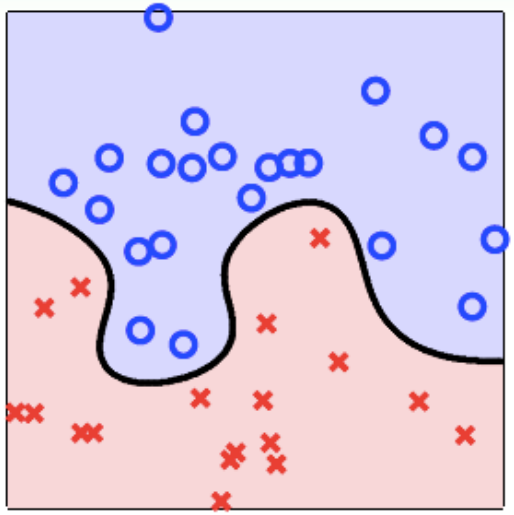
Gaussian kernels are powerful but can overfit



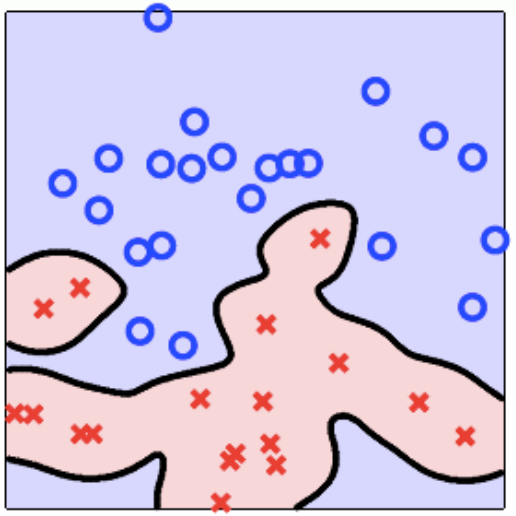


$$K_{gauss}(\vec{a},\vec{b}) = \exp\left(-\gamma \|\vec{a}-\vec{b}\|^2\right)$$

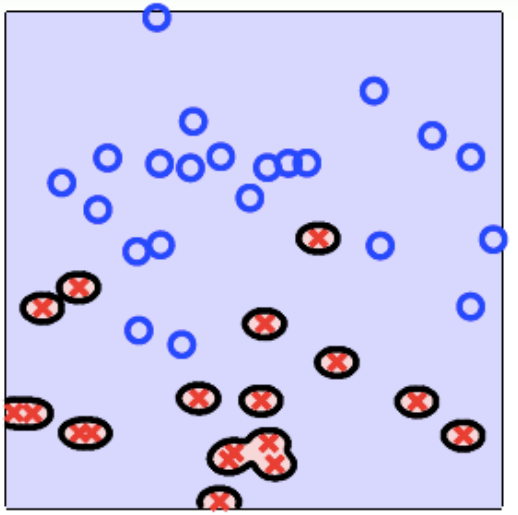
Large γ leads to overfitting!



$\exp(-1\|\mathbf{x}-\mathbf{x}'\|^2)$



$\exp(-10\|\mathbf{x}-\mathbf{x}'\|^2)$



$\exp(-100\|\mathbf{x}-\mathbf{x}'\|^2)$

Support vector machines

1) What is a classifier supposed to do and why is SVM a good one?

2) Establish SVM methodology by walking through a simple case

3) Generalize the SVM to make it more robust

4) A simple trick allows us to create a very complex classifier

5) Practical tips for using SVM

SVM and Logistic Regression are both useful for different cases

Problem type	Logistic Regression	SVM
Data separable	Not Good	Good
Complicated non-linear boundaries	Not Good	Good
Need probabilities	Good	Not Good
Feature reduction/easy interpretability	Good	Not Good
Many features/small training set	Good (or linear SVM)*	Not Good
Small num of features, mid sized training set	Not good	Gaussian kernel (1-1000 features, 10-50k points)
Small num of features, huge training set	Add more features, then use LogReg (or linear SVM)*	Gaussian kernel gets slow

* Linear SVM and Logistic Regression are very comparable

Practical steps

1) Look at your data and plot it!

Practical steps

1) Look at your data and plot it!
2) Do I want to use SVM? (see previous slide)

Practical steps

1) Look at your data and plot it!
2) Do I want to use SVM? (see previous slide)
3) Scale features

Practical steps

1) Look at your data and plot it!
2) Do I want to use SVM? (see previous slide)
3) Scale features
4) Choose kernel (linear, poly, rbf) (“rbf” = Gaussian, usually better than poly for non-linear boundaries)

Practical steps

1) Look at your data and plot it!
2) Do I want to use SVM? (see previous slide)
3) Scale features
4) Choose kernel (linear, poly, rbf) (“rbf” = Gaussian, usually better than poly for non-linear boundaries)
5) Choose regularization parameter C

Practical steps

1) Look at your data and plot it!
2) Do I want to use SVM? (see previous slide)
3) Scale features
4) Choose kernel (linear, poly, rbf) (“rbf” = Gaussian, usually better than poly for non-linear boundaries)
5) Choose regularization parameter C
6) For poly, rbf kernels: choose degree (poly) or gamma (Gaussian)

Practical steps

1) Look at your data and plot it!
2) Do I want to use SVM? (see previous slide)
3) Scale features
4) Choose kernel (linear, poly, rbf) (“rbf” = Gaussian, usually better than poly for non-linear boundaries)
5) Choose regularization parameter C
6) For poly, rbf kernels: choose degree (poly) or gamma (Gaussian)
7) Cross-validation to choose best parameters C, degree, gamma (e.g. grid search for most accurate classifier)

Practical steps

1) Look at your data and plot it!
2) Do I want to use SVM? (see previous slide)
3) Scale features
4) Choose kernel (linear, poly, rbf) (“rbf” = Gaussian, usually better than poly for non-linear boundaries)
5) Choose regularization parameter C
6) For poly, rbf kernels: choose degree (poly) or gamma (Gaussian)
7) Cross-validation to choose best parameters C, degree, gamma (e.g. grid search for most accurate classifier)
8) Keep looking at your data and plotting it! (Do boundaries make sense?)

Practical steps

1) Look at your data and plot it!
2) Do I want to use SVM? (see previous slide)
3) Scale features
4) Choose kernel (linear, poly, rbf) (“rbf” = Gaussian, usually better than poly for non-linear boundaries)
5) Choose regularization parameter C
6) For poly, rbf kernels: choose degree (poly) or gamma (Gaussian)
7) Cross-validation to choose best parameters C, degree, gamma (e.g. grid search for most accurate classifier)
8) Keep looking at your data and plotting it! (Do boundaries make sense?)

##Python:
svc=SVC(C=0.01|1|100, kernel='linear|poly|rbf', gamma=0.1|1|10, degree = 1|2|3)
svc.fit(training_data, training_labels)
class_prediction = svc.predict(unlabeled_data)