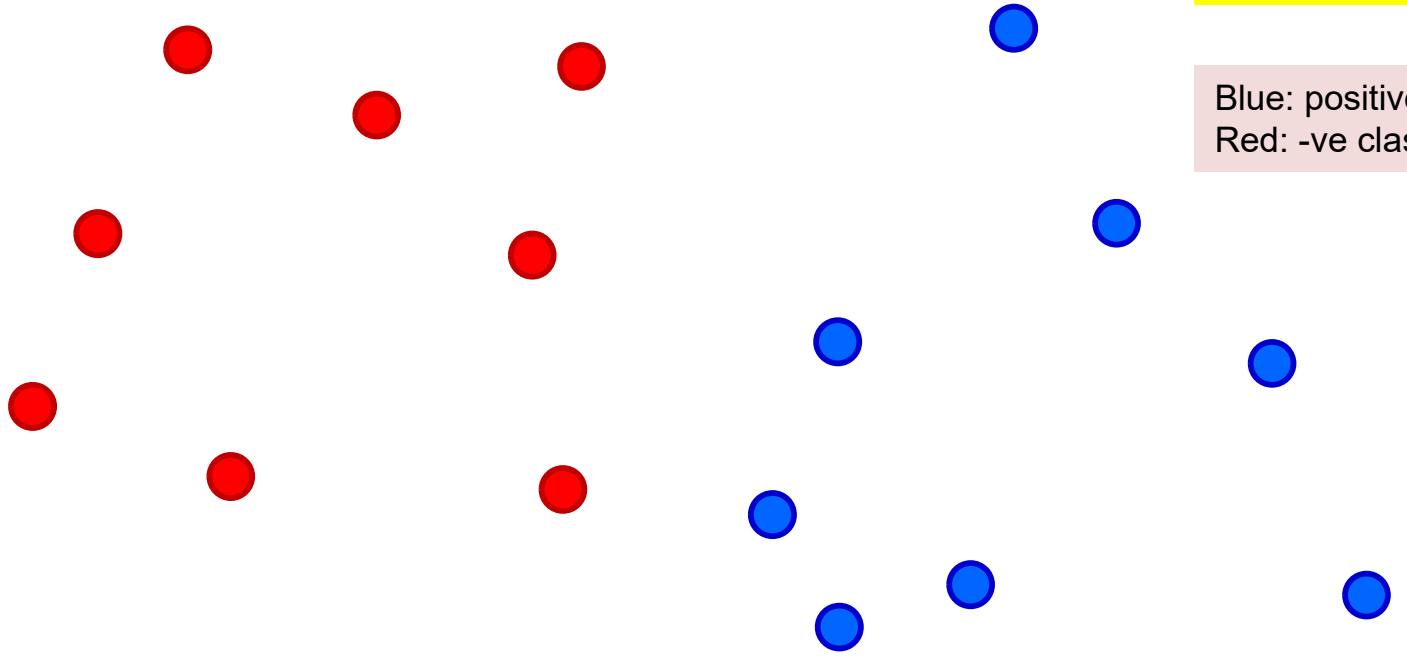
Linear Classifiers:
SVM

Classification

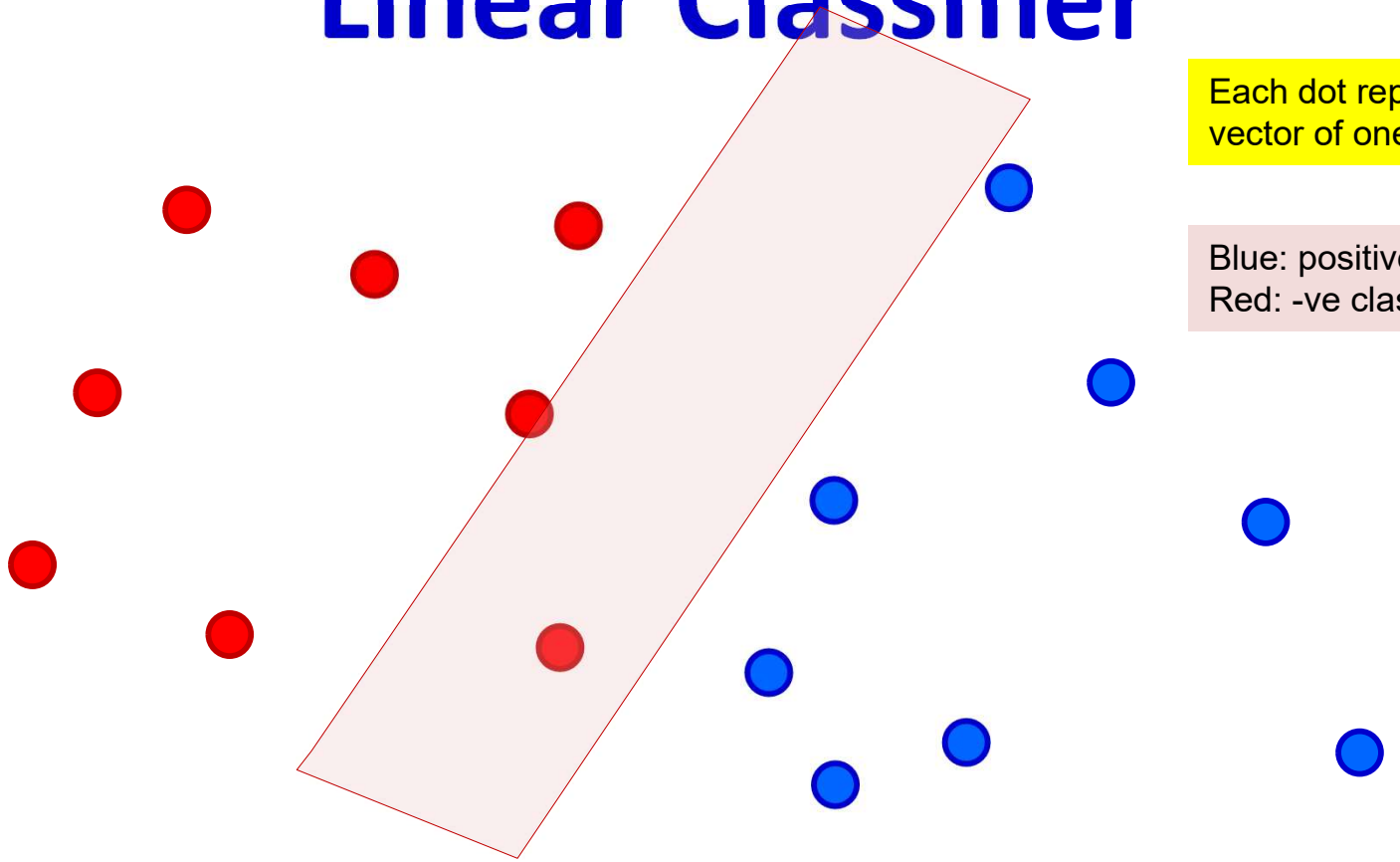


Each dot represents a feature vector of one training instance

Blue: positive class,
Red: -ve class

- Given a bunch of +ve and –ve training instances
  - Find a rule that correctly assigns a new test instance to one of the two classes

Linear Classifier

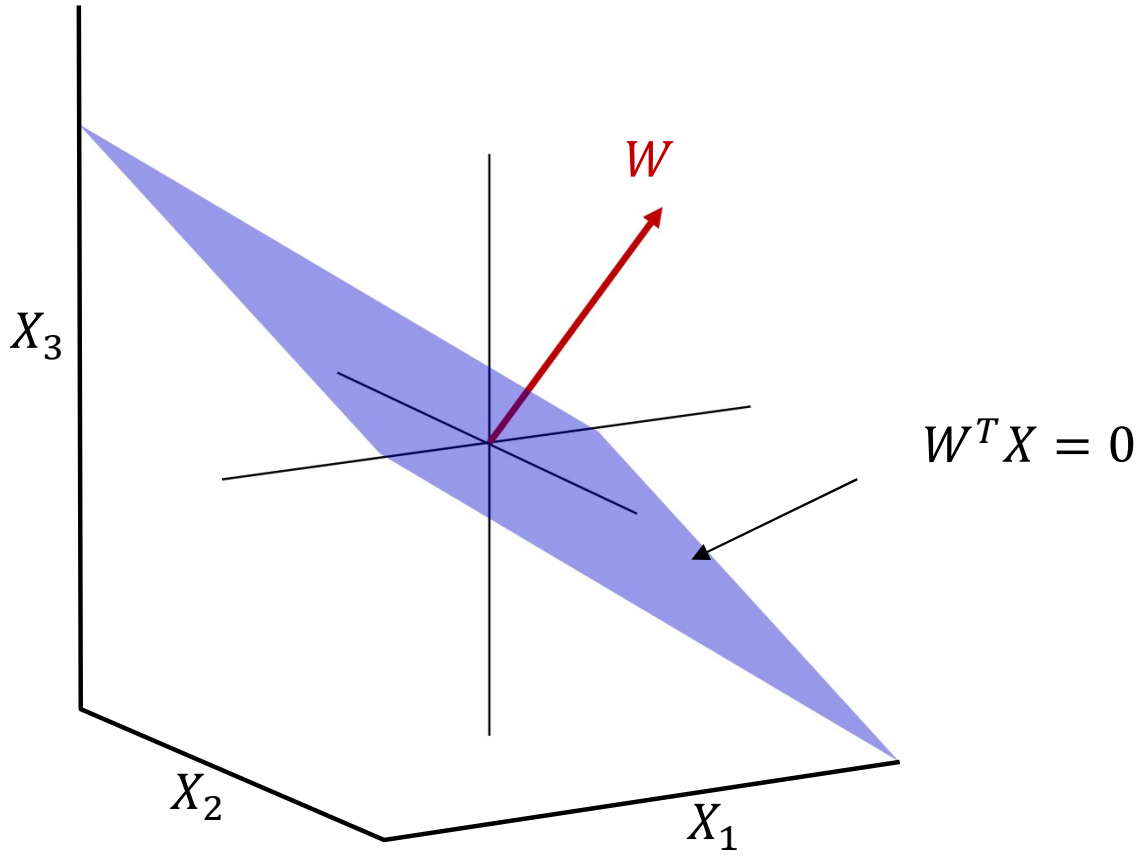


Each dot represents a feature vector of one training instance

Blue: positive class,
Red: -ve class

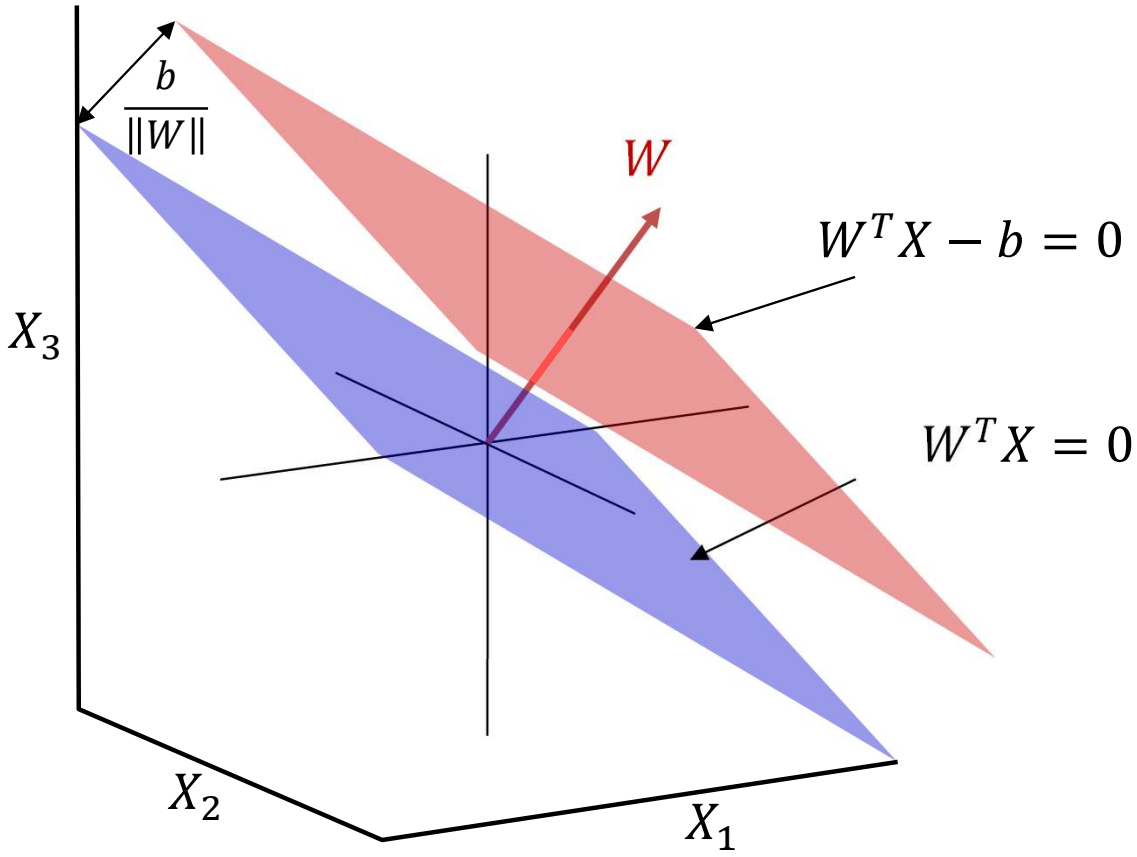
- Initially assume that the classes are separable by a *hyperplane*
  - A *linear* classifier
- Also that the training data are *perfectly* separable by the hyperplane
- We will fix these assumptions later

The equation for a hyperplane



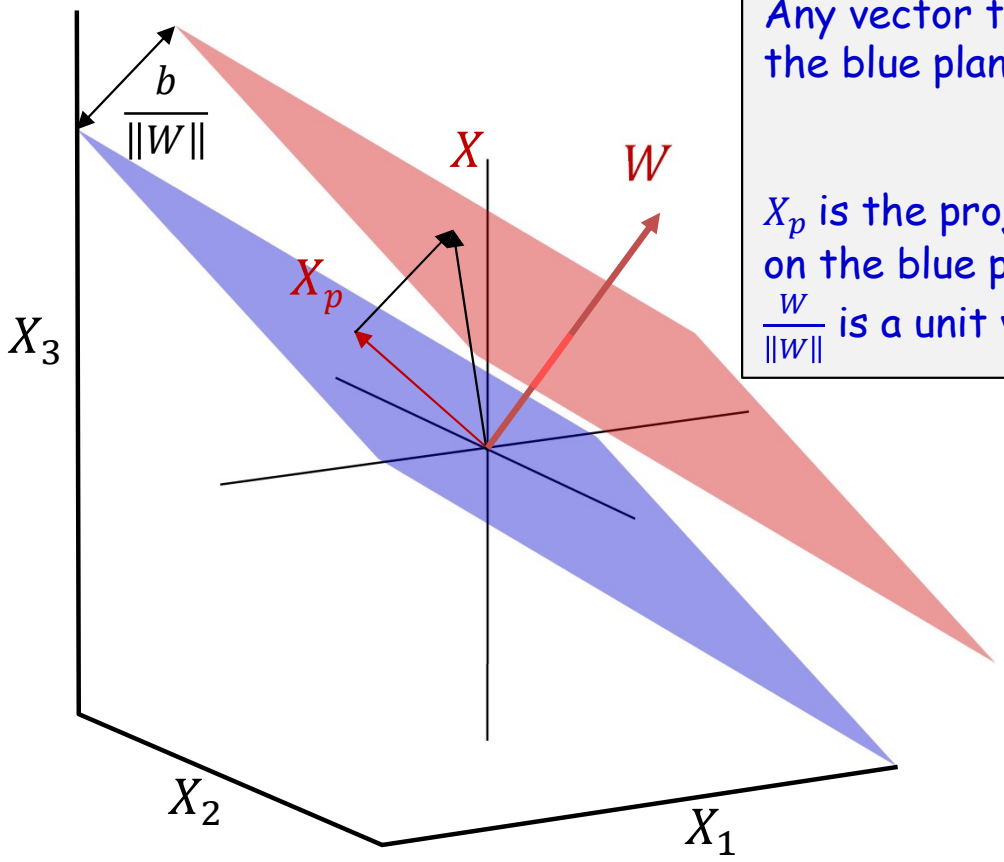
- $W^T X = 0$ is the equation representing the set of all vectors that are orthogonal to W

The equation for a hyperplane



- $W^T X - b = 0$ is the equation representing plane that is orthogonal to W and a distance $\frac{b}{\|W\|}$ from origin
  - The set of all vectors that are a distance $\frac{b}{\|W\|}$ from the blue plane

The equation for a hyperplane



Any vector that is a distance d from the blue plane can be written as

$$X = X_p + d\frac{W}{\|W\|}$$

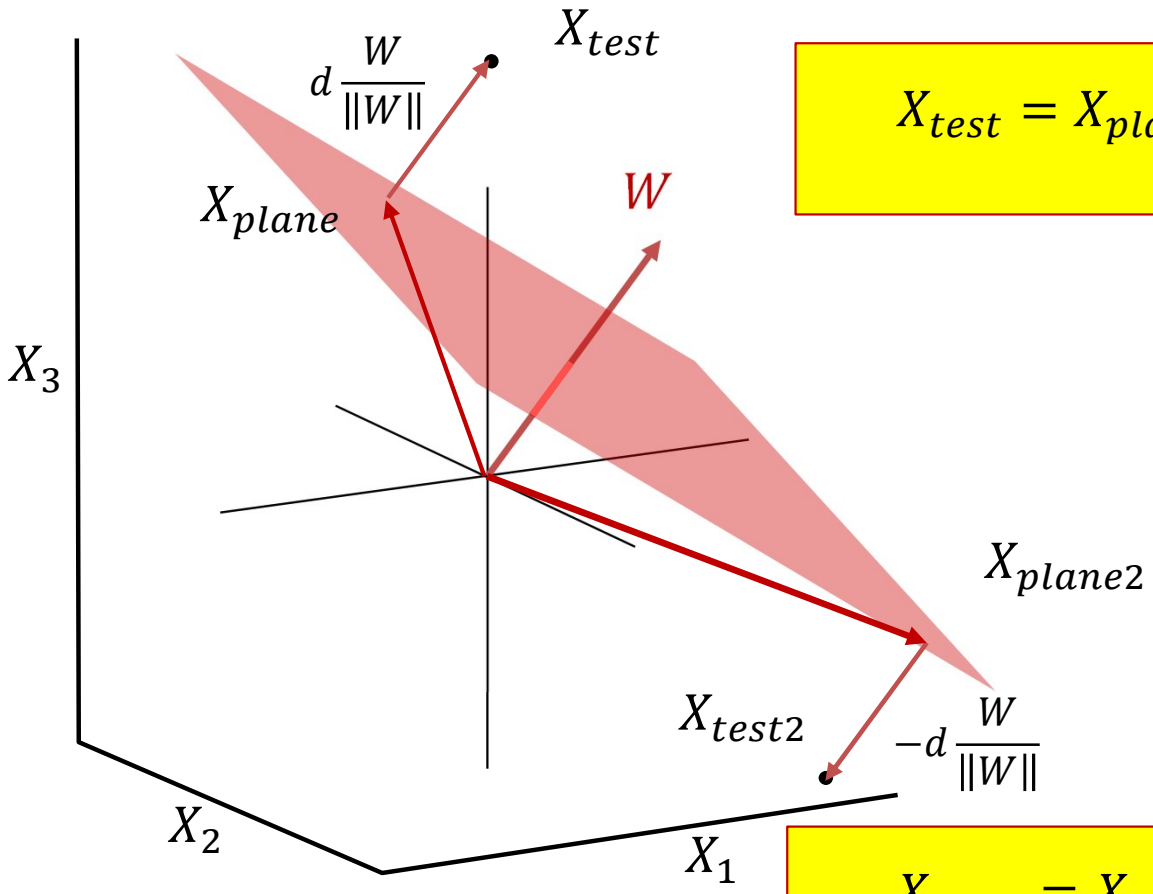
X_p is the projection of the vector on the blue plane

$\frac{W}{\|W\|}$ is a unit vector in the direction of W

Trivial proof:

- On the red plane any $X = X_p + \left(\frac{b}{\|W\|}\right)\frac{W}{\|W\|}$
- $W^T X = W^T X_p + b\frac{W^T W}{\|W\|^2} = b$

Distance from a hyperplane



$$X_{test} = X_{plane} + d\frac{W}{\|W\|}$$

$$X_{test} = X_{plane2} - d\frac{W}{\|W\|}$$

- The distance of any X_{test} from the plane $W^T X - b = 0$ is $d = \frac{W^T X_{test} - b}{\|W\|}$
- This can be positive (in the direction of W) or negative (opposite to W)

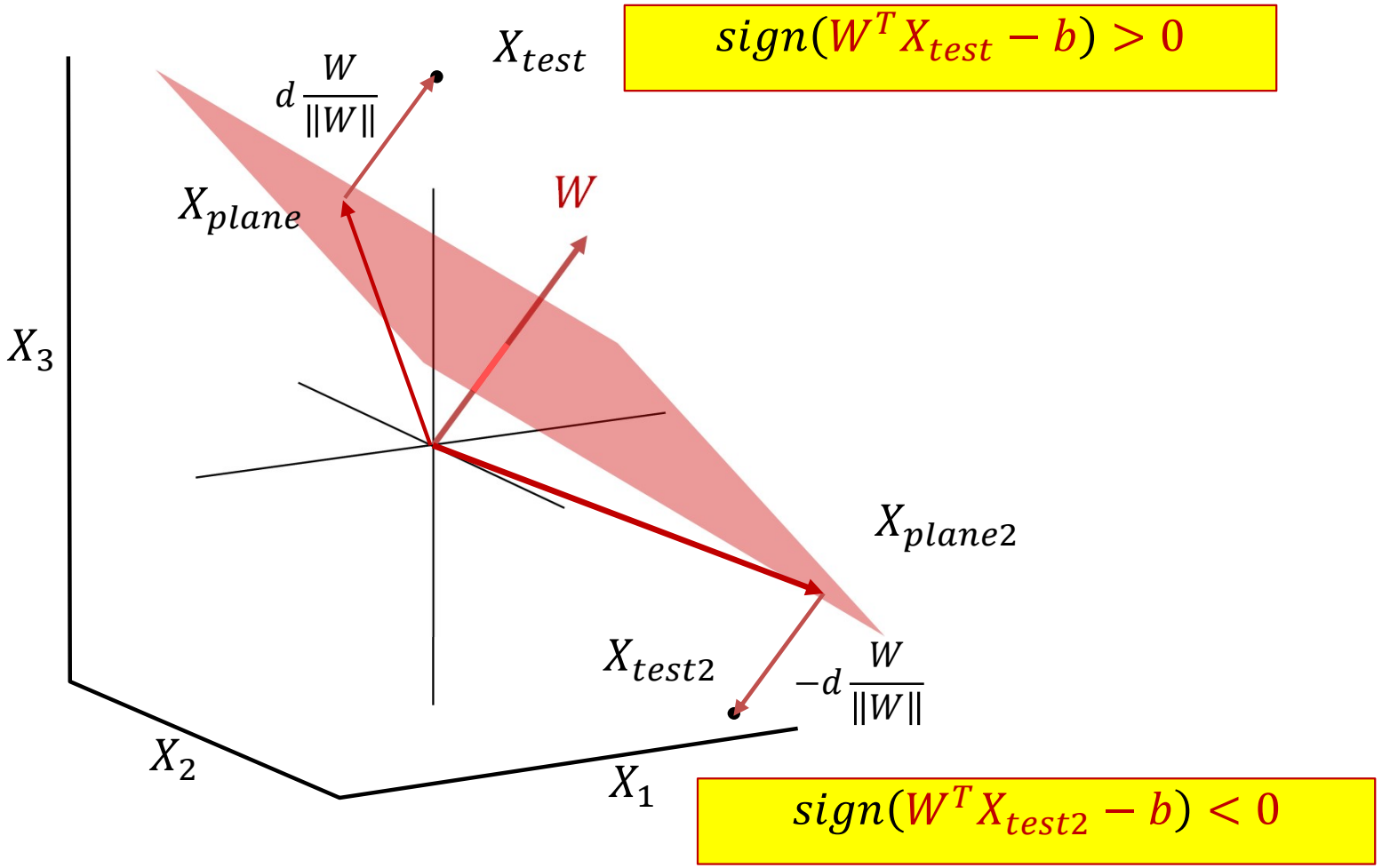
Poll 1

- The plane $W^T X - b = 0$ is the equation of a plane orthongal to W and a distance b from the origin
  - True
  - False

Poll 1

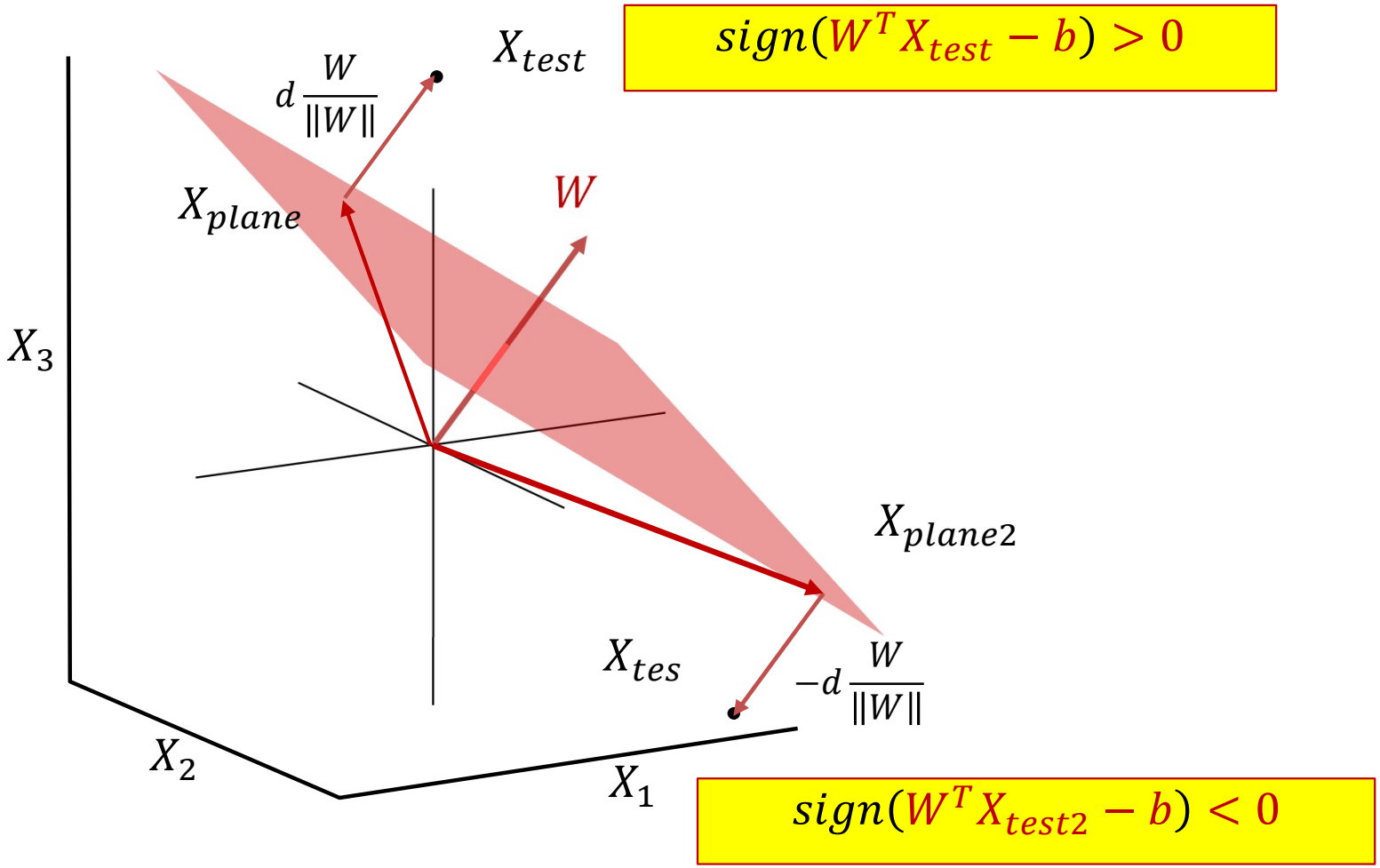
- The plane $W^T X - b = 0$ is the equation of a plane orthongal to W and a distance b from the origin
  - True
  - **False**

Sign of distance from hyperplane



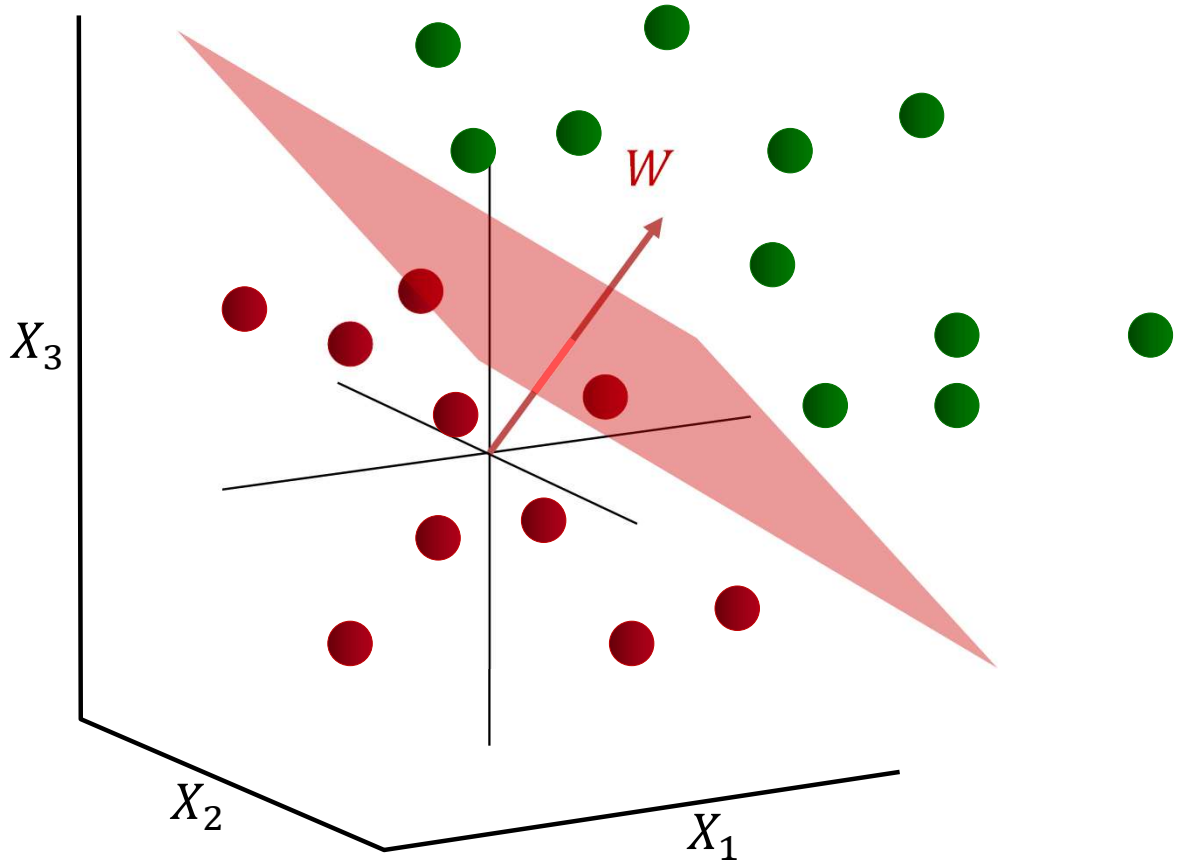
- The sign of $W^T X - b$ signifies which side of the plane the point X is on

Linear Classifier



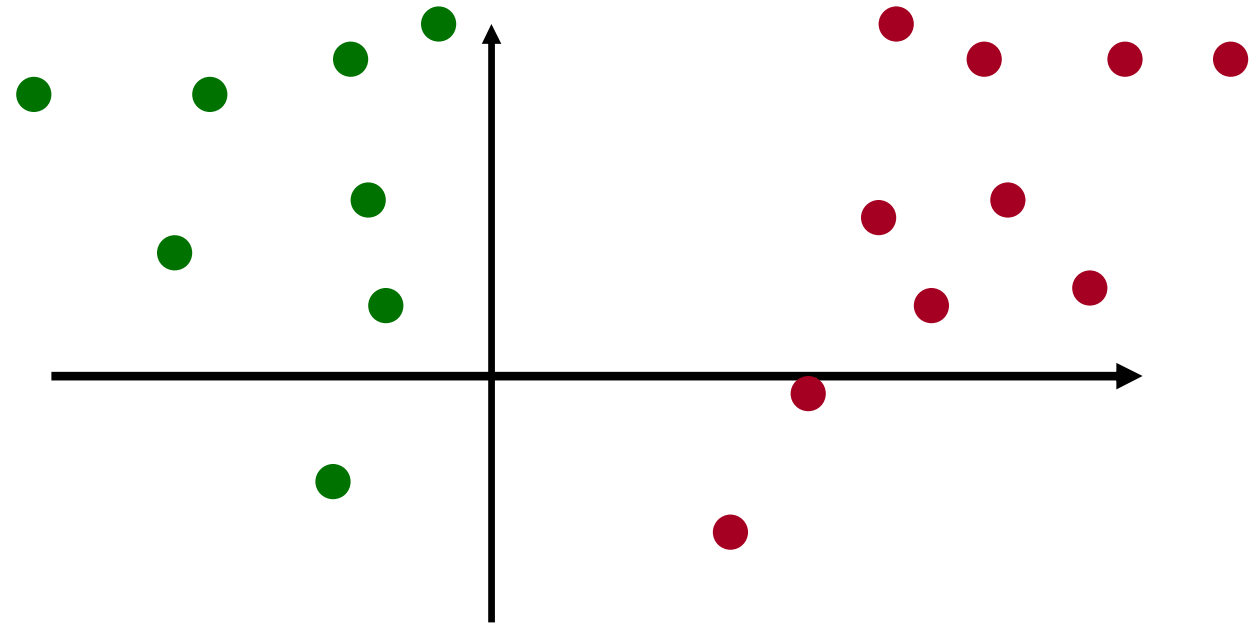
- The plane $W^T X - b$ is a linear classifier
  - The class is given by $sign(W^T X_{test} - b)$

Linearly separable data



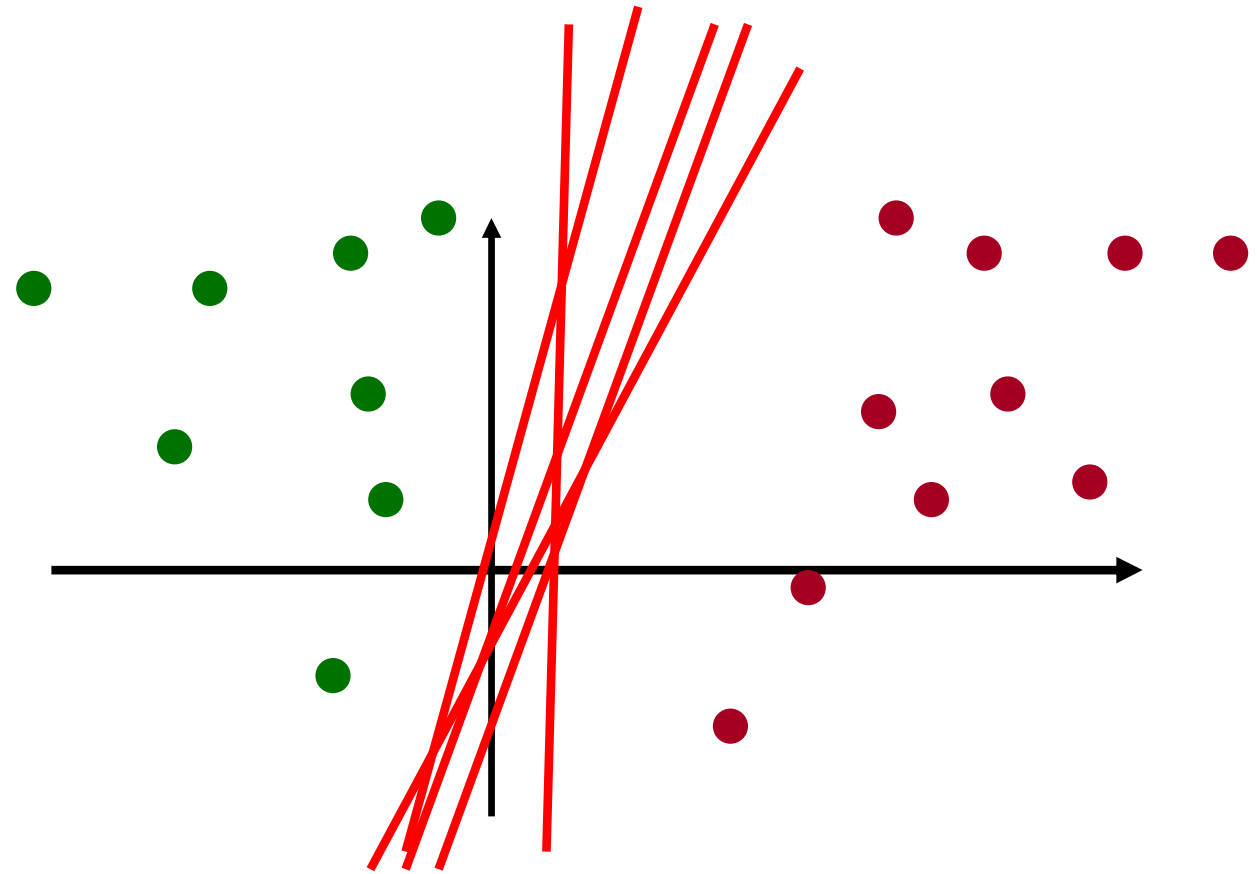
- Data where the two classes are separated by a hyperplane
  - And classification can be performed by $sign(W^T X_{test} - b)$ for any separating hyperplane

2D illustration, linearly separable data



- Classes are linearly separable
- Dots represent “training” instances
- **Training problem**: Given these training instances find a separating hyperplane

The separating hyperplane

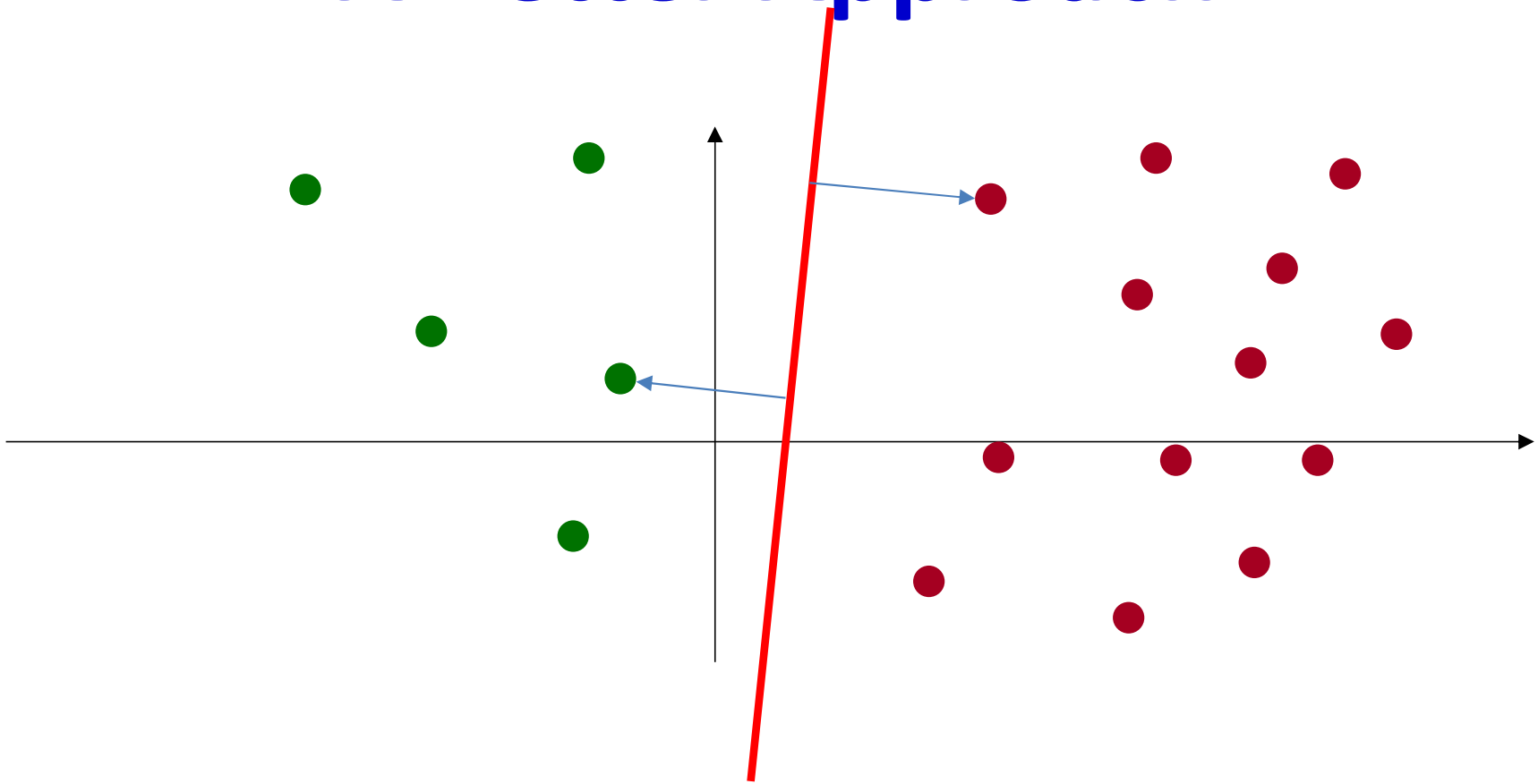


- Problem: Given these training instances find a separating hyperplane
- **Many ways of finding this hyperplane**
  - Any number of solution algorithms are possible

Enter: Support Vector Machines

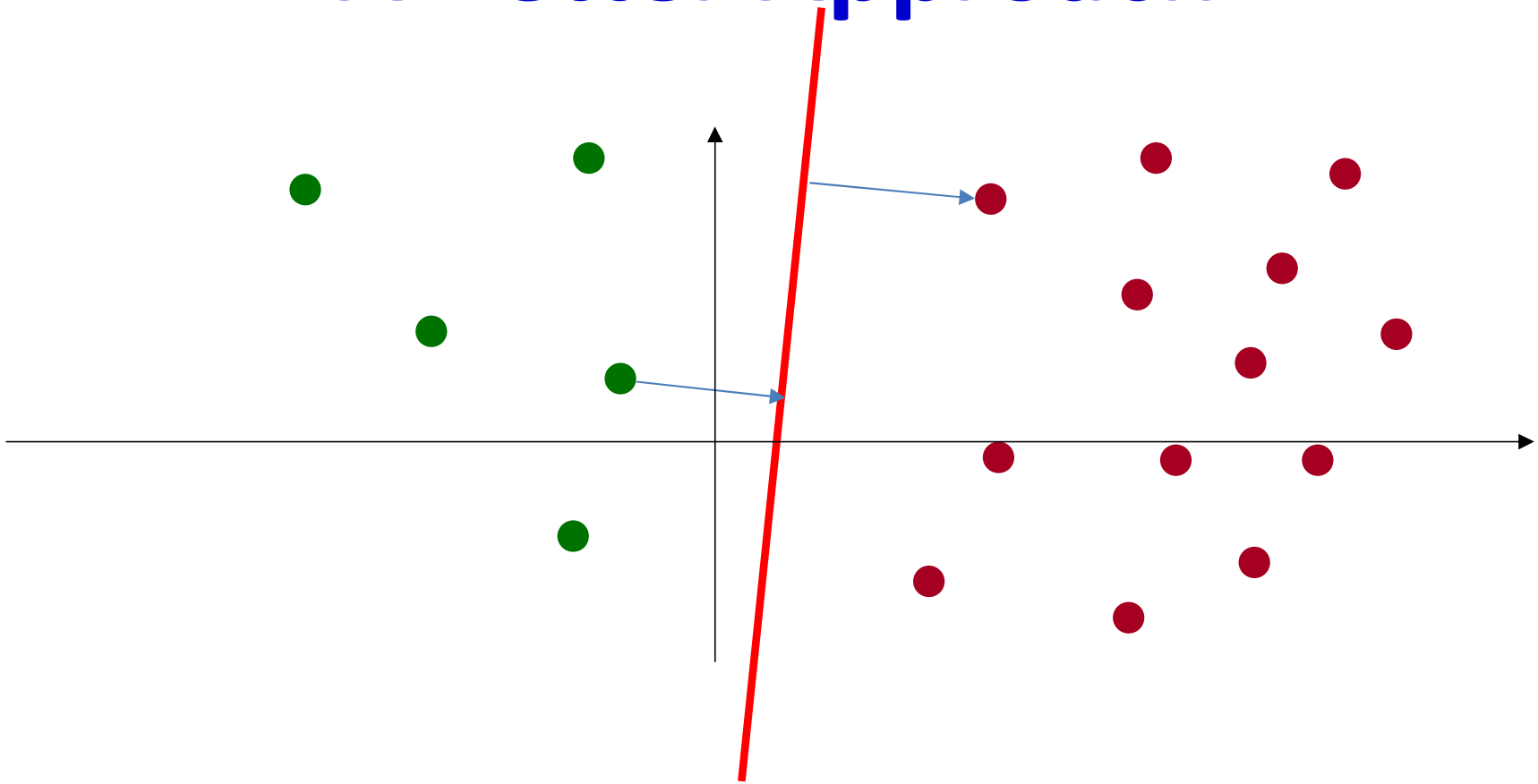
- Find a classifier that is maximally distant from the *closest* instances from either class

A Better Approach



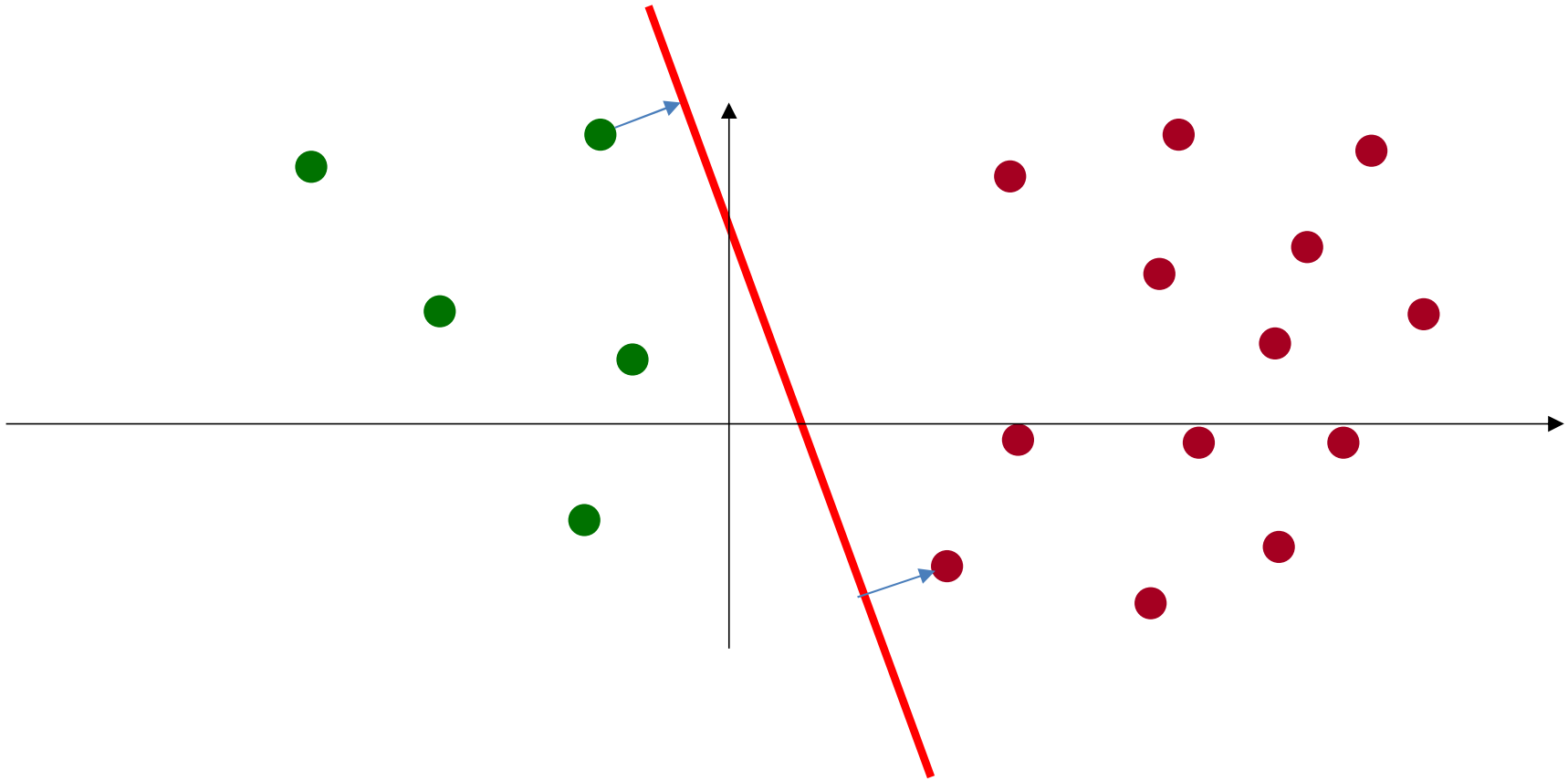
- Any linear classifier has some *closest* instances
- These instances will be at some distance from the boundary
- Changing the classifier will change both, the closest instance, and their distance from the boundary

A Better Approach



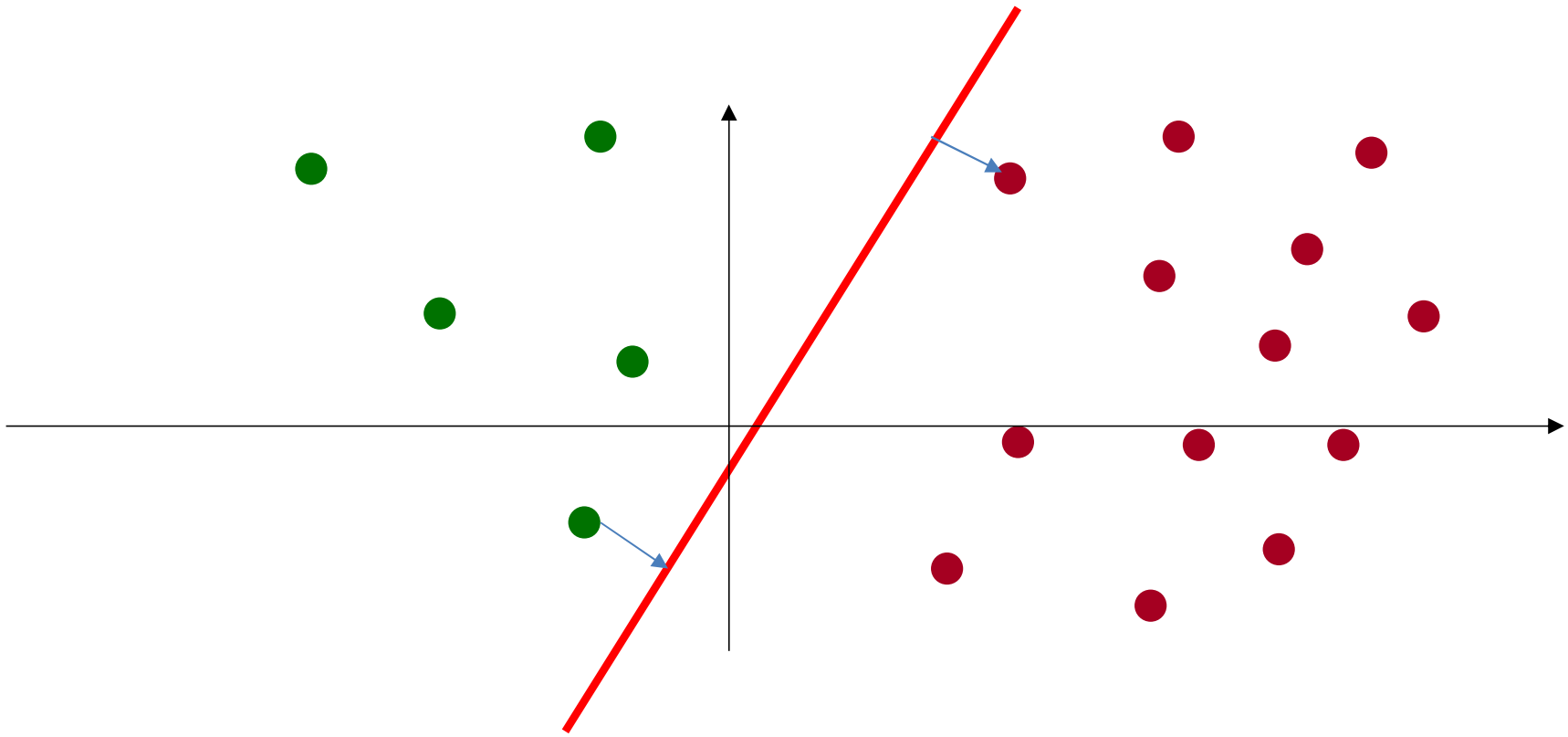
- Search through all classifiers such that the distance to the closest points is maximized
  - Very conservative
  - Focuses on *worst-case* scenario
  - Maximizes the chance that the classifier will work well on new unseen data

A Better Approach



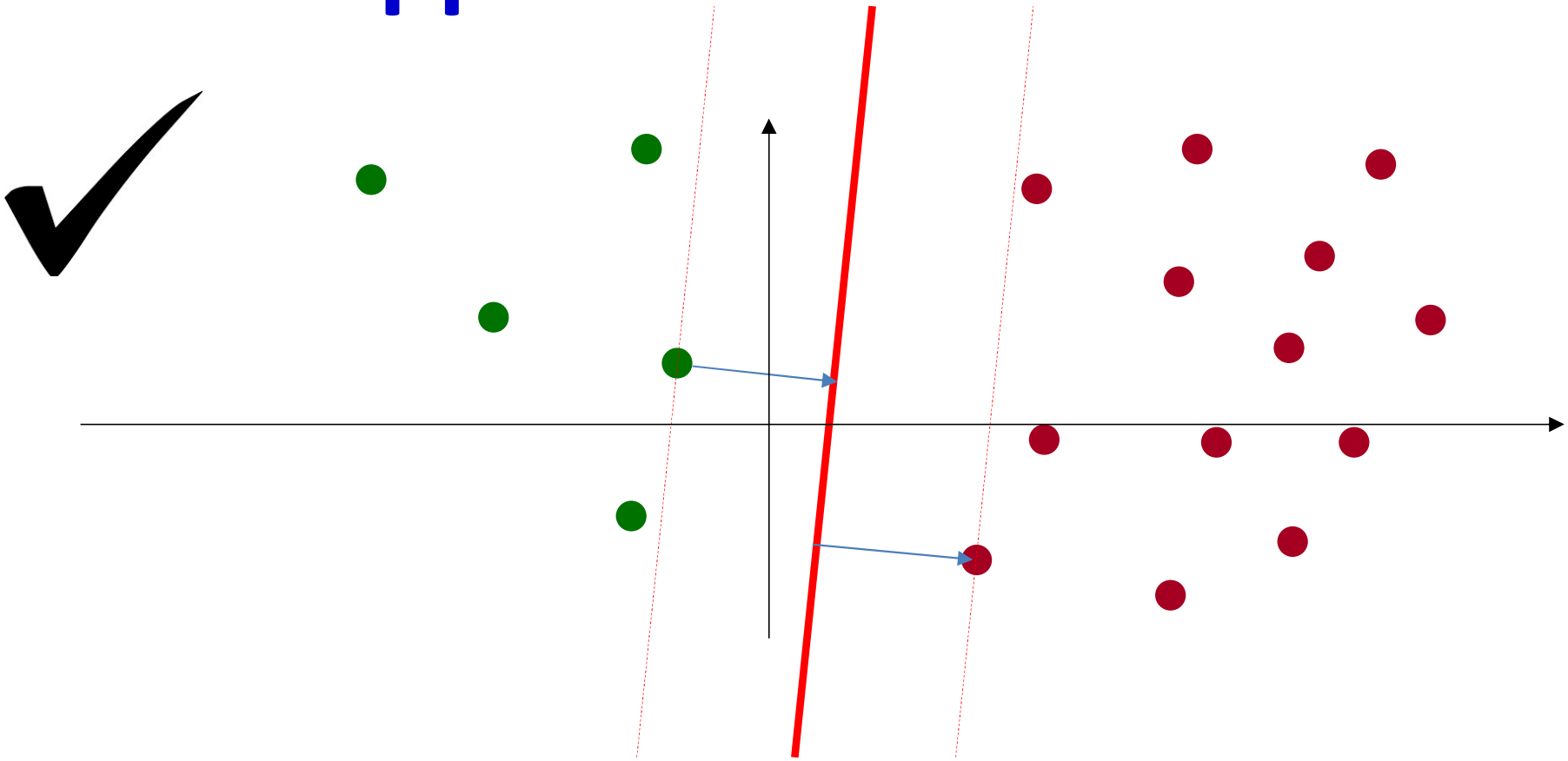
- Search through all classifiers such that the distance to the closest points is maximized
  - Very conservative
  - Focuses on *worst-case* scenario
  - Maximizes the chance that the classifier will work well on new unseen data

A Conservative Approach



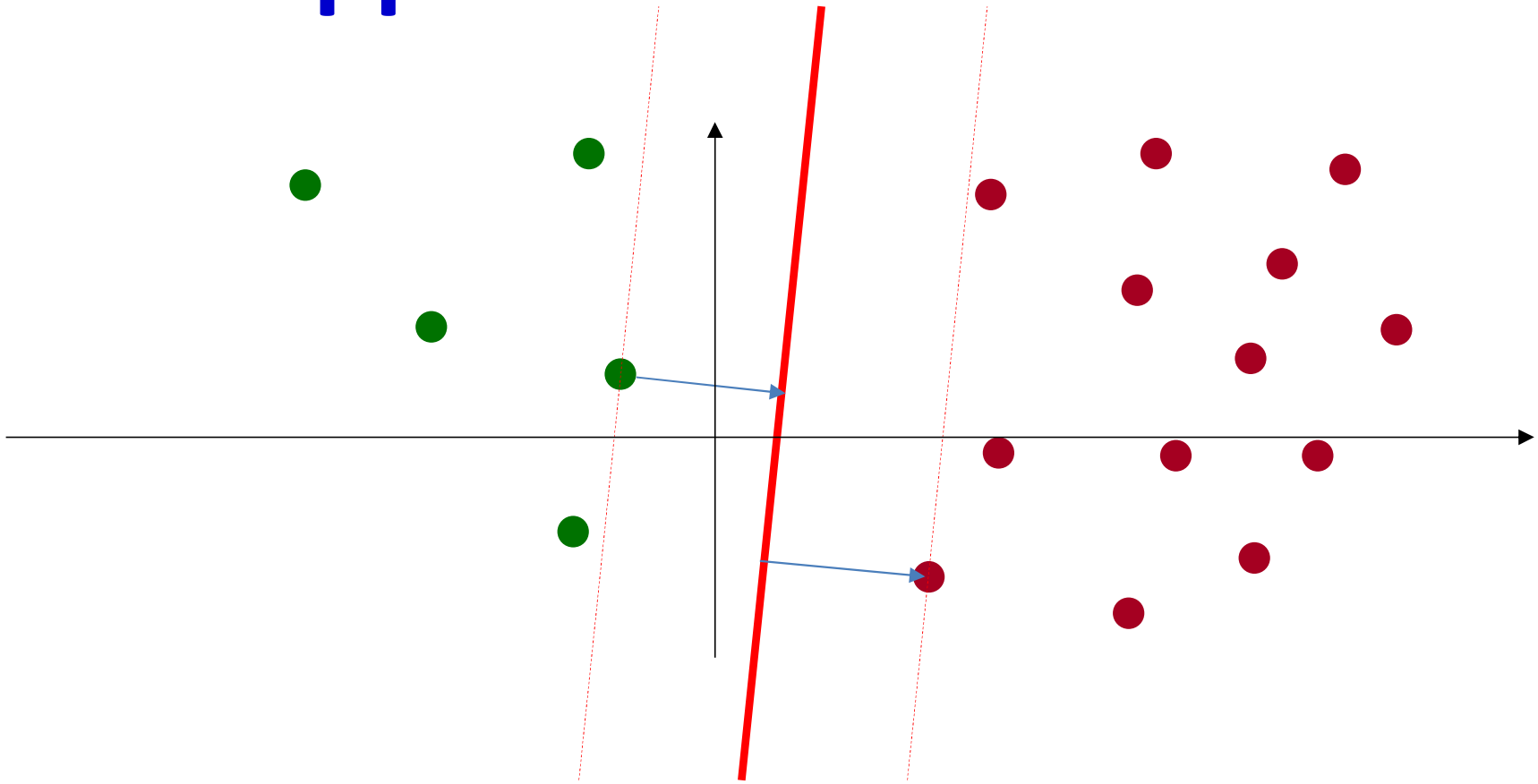
- Search through all classifiers such that the distance to the closest points is maximized
  - Very conservative
  - Focuses on *worst-case* scenario
  - Maximizes the chance that the classifier will work well on new unseen data

Support Vector Machine



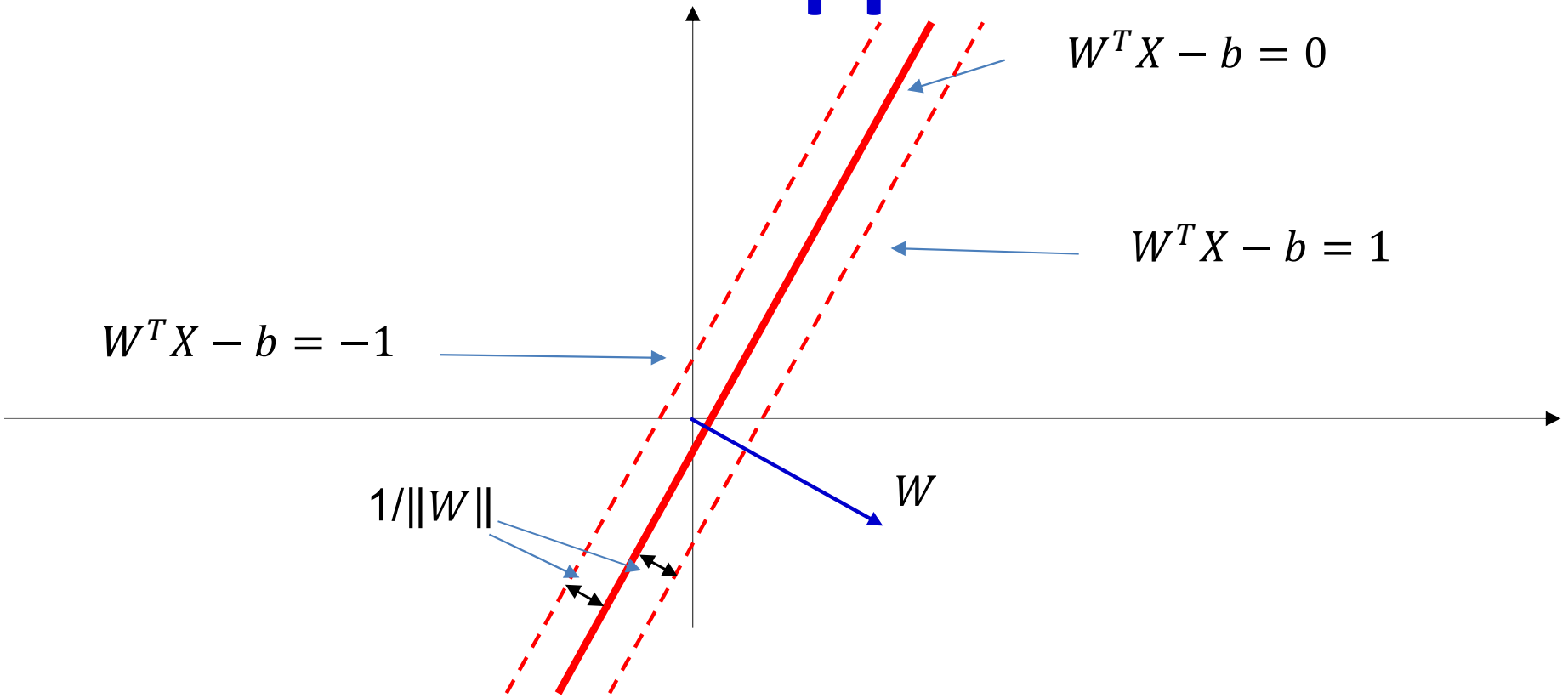
- Search through all classifiers such that the distance to the closest points is maximized
  - Very conservative
  - Focuses on *worst-case* scenario
  - Maximizes the chance that the classifier will work well on new unseen data

Support Vector Machine



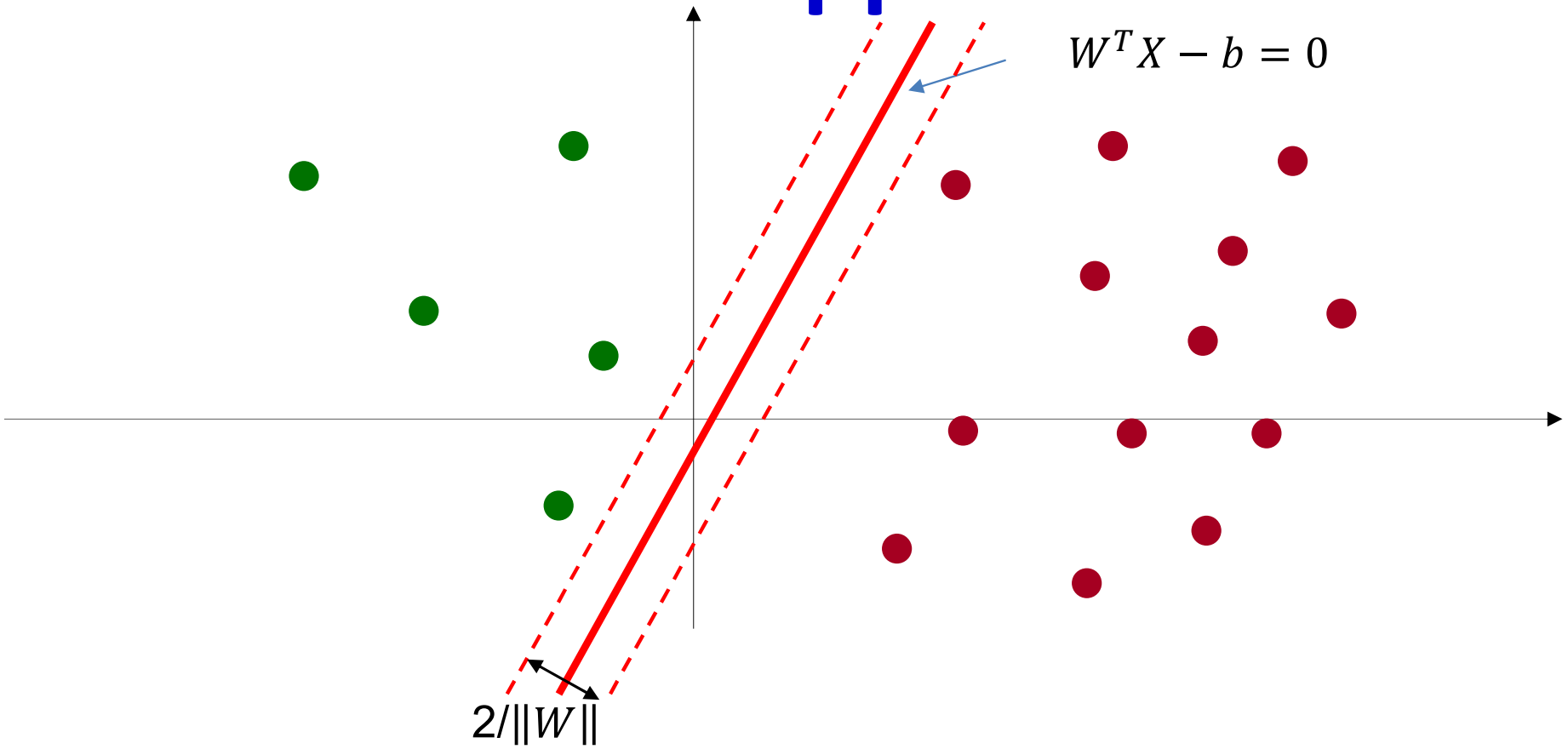
- Find the classifier such that the distance to the *closest* points is maximized
- I.e. solve *two* problems: find the closest points, and the classifier, such that the distance is maximum
  - Position the classifier in the *middle* so that the distance to the closest green = distance to the closest red
- Is this a combinatorial optimization problem??

Solution Approach



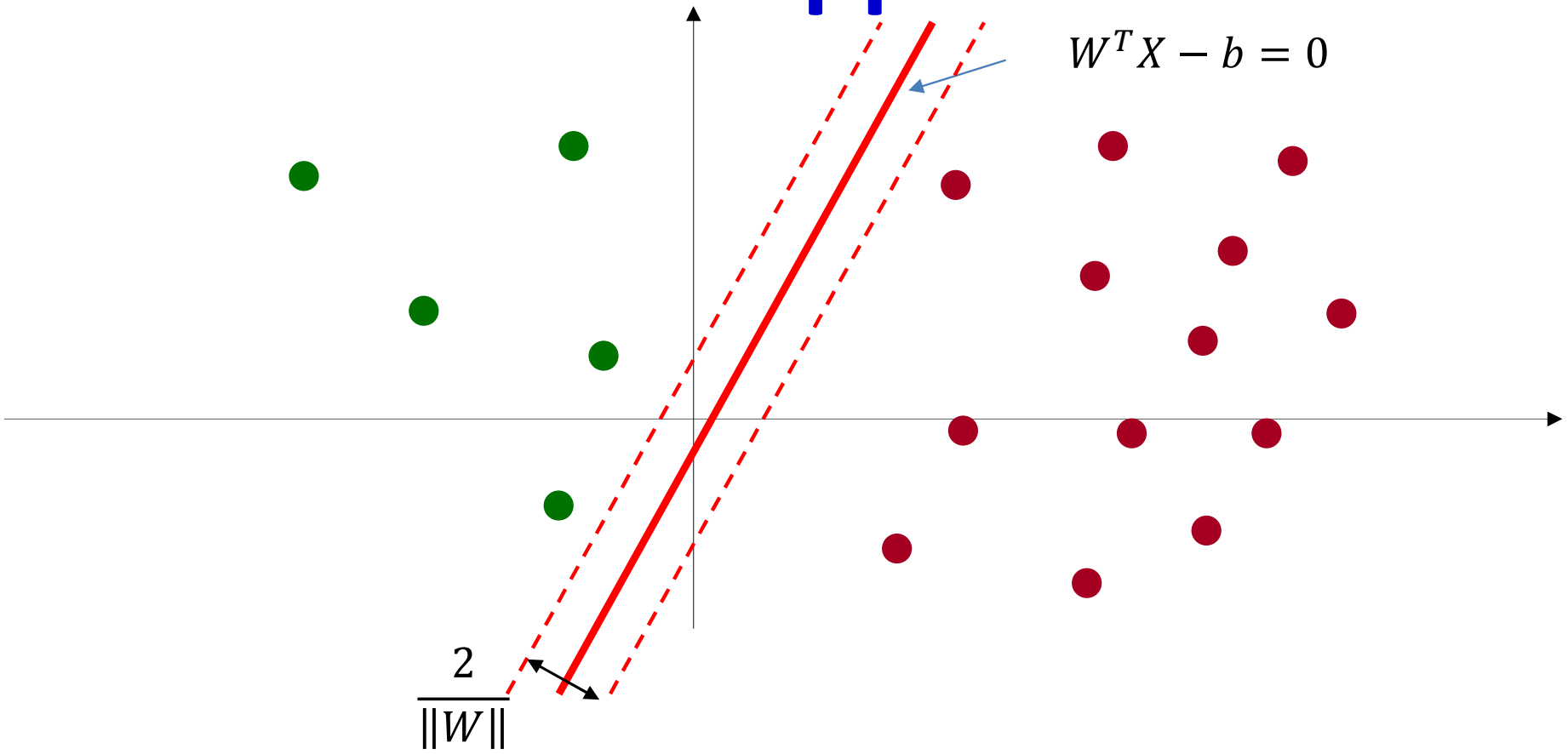
- For any hyperplane (linear classifier) $W^T X - b = 0$
- Choose two hyperplanes $W^T X - b = 1$ and $W^T X - b = -1$
  - The distance of these hyperplanes from the classifier is $1/\|W\|$
  - The total distance between the hyperplanes is $2/\|W\|$

Solution Approach



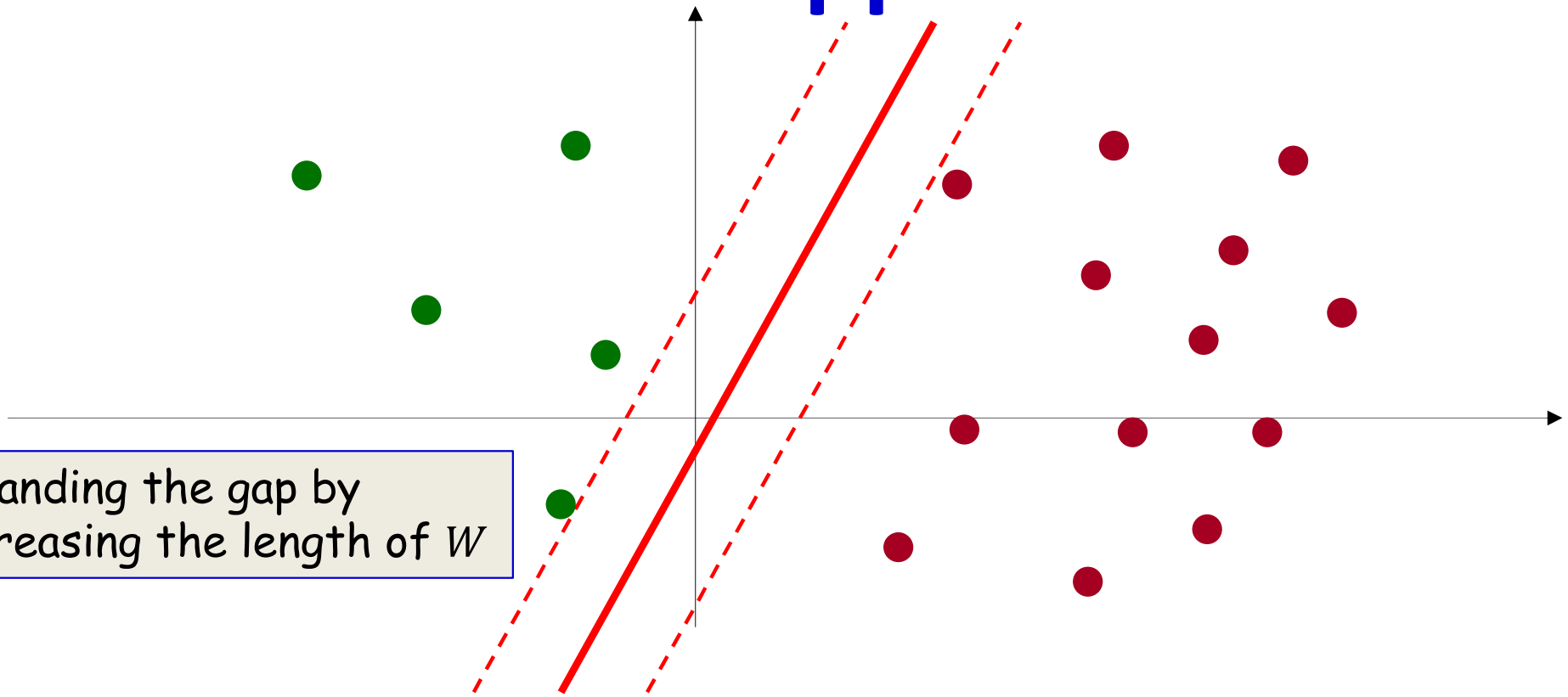
- Constraint: Perfect classification with a margin
- Choose the hyperplanes such that
  - All positive points are on the positive side of the positive hyperplane
  - All negative points are on the negative side of the negative hyperplane

Solution Approach



- The distance between the hyperplanes is $\frac{2}{\|W\|}$
- Find the W (and b) such that this is maximized, while maintaining the constraint that all training points are on the "outside" of the appropriate hyperplane

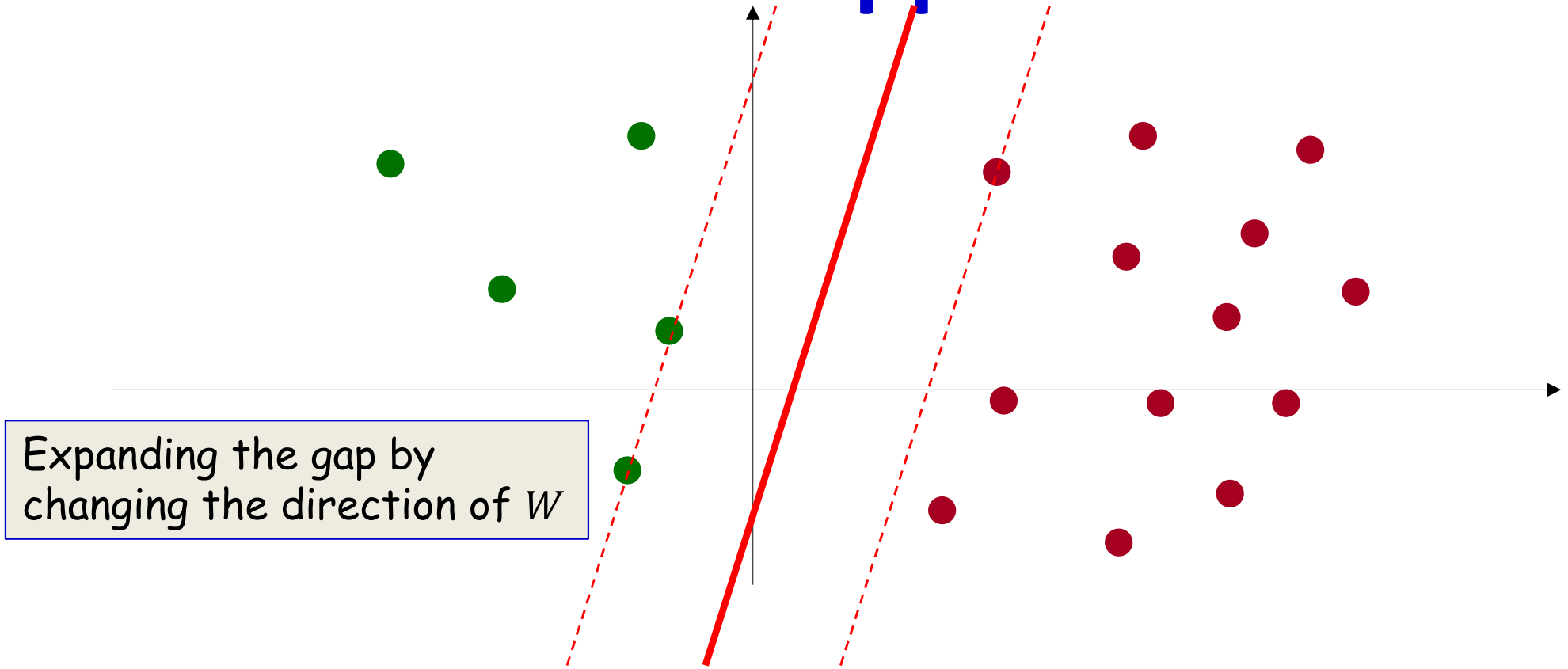
Solution Approach



Expanding the gap by decreasing the length of W

- The distance between the hyperplanes is $\frac{2}{\|W\|}$
- Find the W (and b) such that this is maximized, while maintaining the constraint that all training points are on the "outside" of the appropriate hyperplane

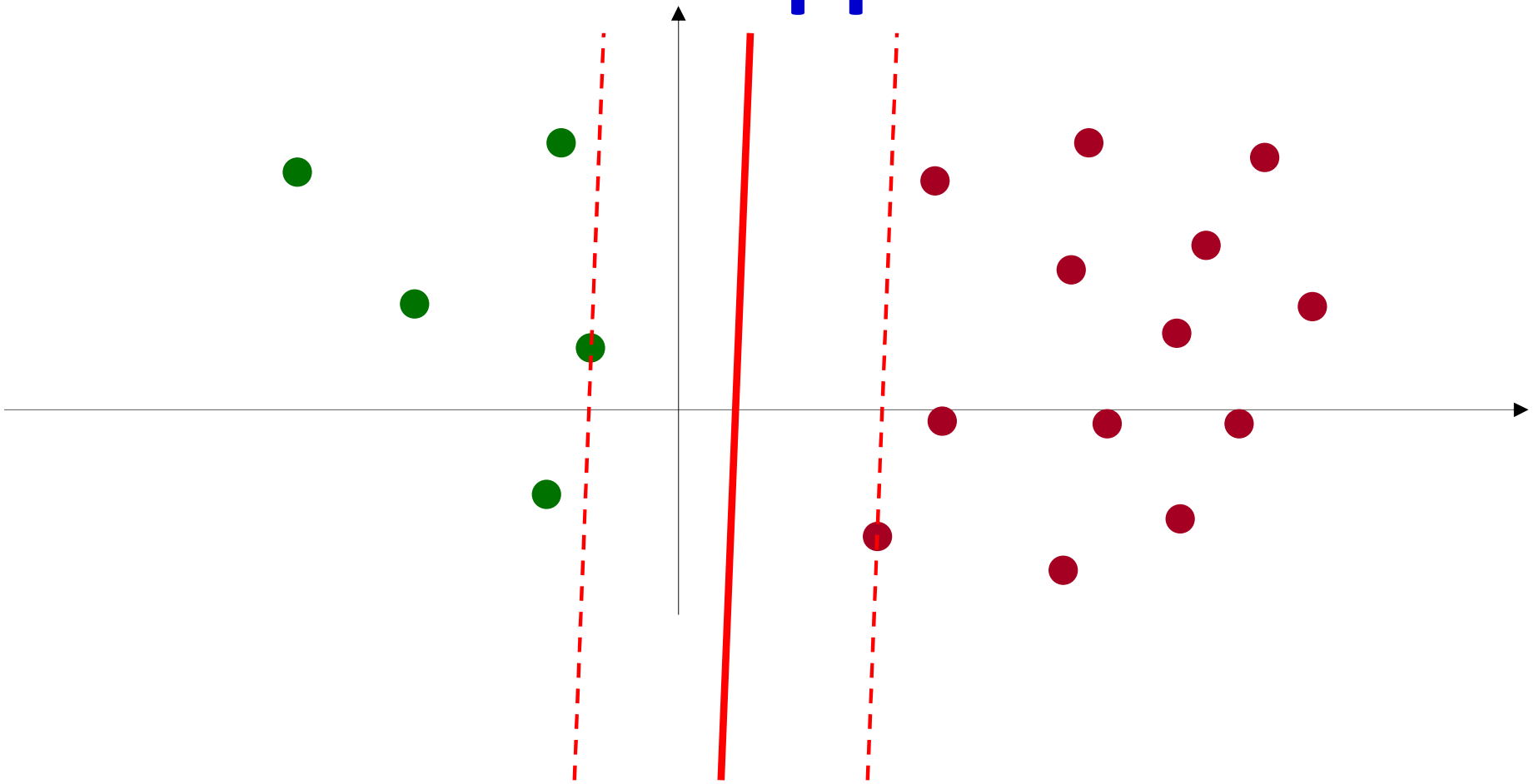
Solution Approach



Expanding the gap by changing the direction of W

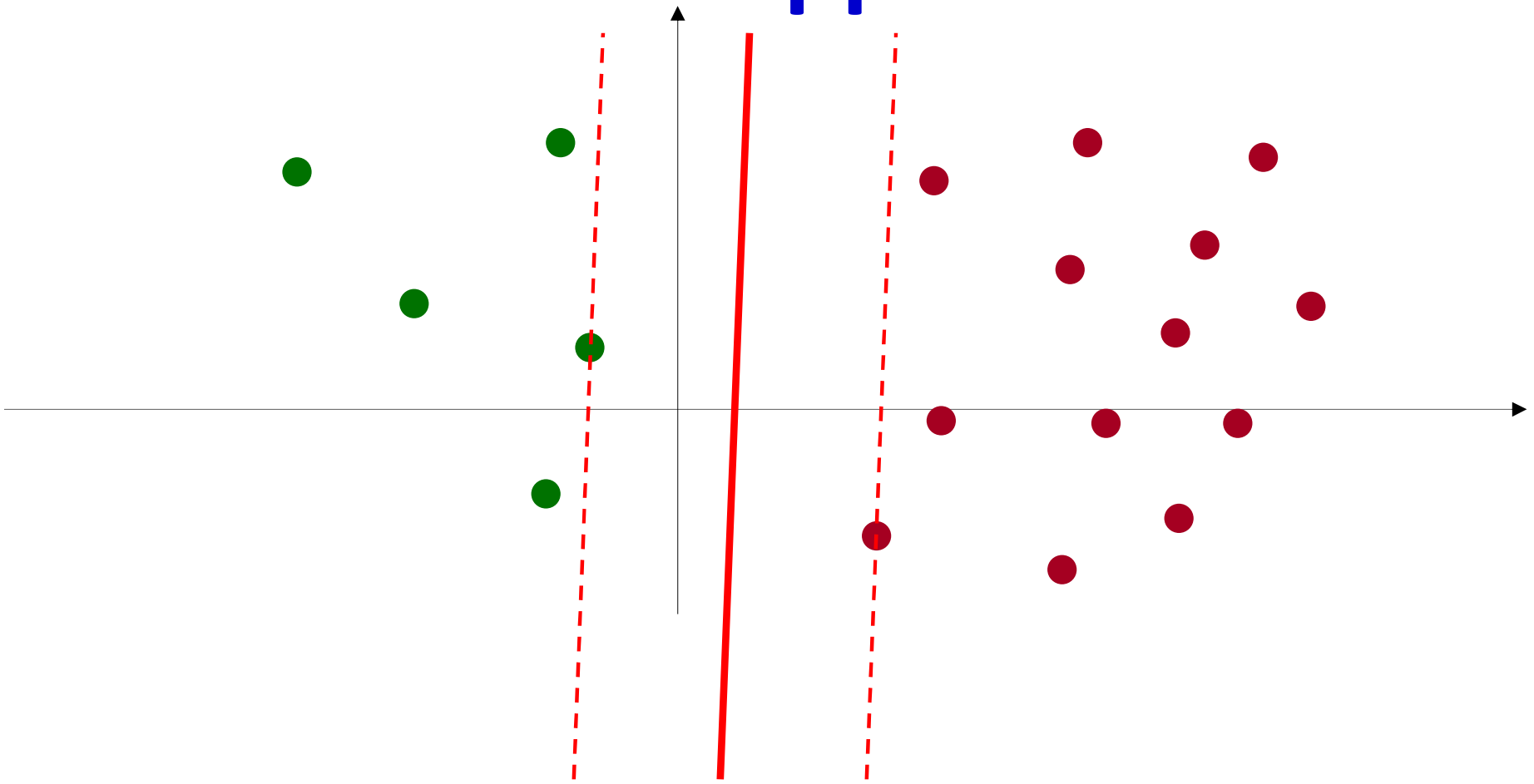
- The distance between the hyperplanes is $\frac{2}{\|W\|}$
- Find the W (and b) such that this is maximized, while maintaining the constraint that all training points are on the "outside" of the appropriate hyperplane

Solution Approach



- The distance between the hyperplanes is $\frac{2}{\|W\|}$
- Find the W (and b) such that this is maximized, while maintaining the constraint that all training points are on the "outside" of the appropriate hyperplane

Solution Approach



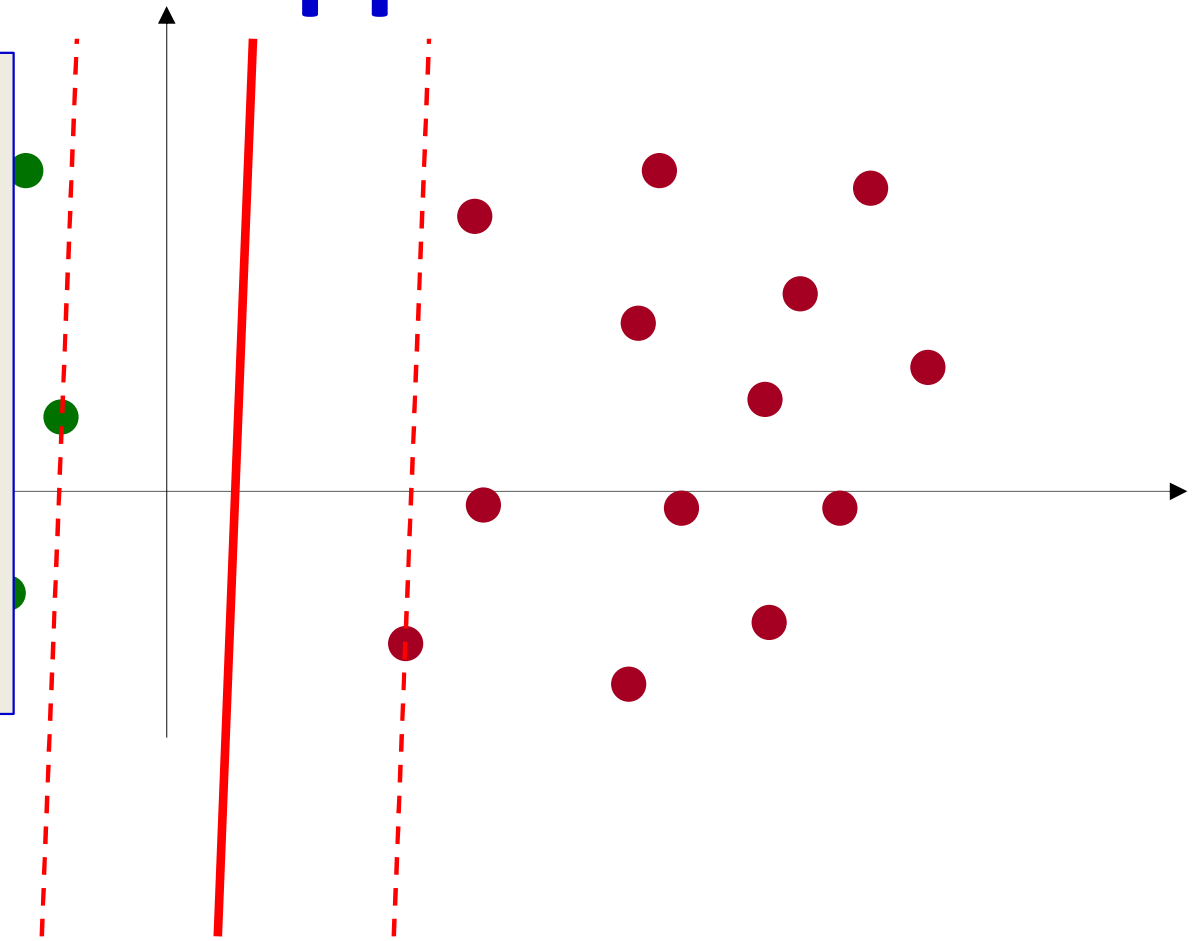
- The distance between the hyperplanes is $\frac{2}{\|W\|}$
- Maximize this distance. I.e. ..
- **Minimize $\|W\|$ such that**
  - all training points are on the "outside" of the appropriate hyperplane

Solution Approach

Decreasing the length of W will expand the gap between the boundary planes

Rotating it will *also* provide scope to increase this gap

Must find a formalism that explores both options simultaneously



- The distance between the hyperplanes is $\frac{2}{\|W\|}$
- Maximize this distance. I.e. ..
- **Minimize $\|W\|^2$ such that**
  - all training points are on the "outside" of the appropriate hyperplane

Let's formalize this

- Constraint: Ensuring that all training instances are on the proper side of their respective hyperplanes
- For positive training instances X_i:

$$W^T X_i - b \geq 1$$

- For negative instances

$$W^T X_i - b \leq -1$$

- Generically stated, for all instances we want

$$Y_i(W^T X_i - b) \geq 1$$

Solution Formalism

- Minimize $\|W\|$ such that
- For all training instances

$$Y_i(W^T X_i - b) \geq 1$$

- Formally

$$\widehat{W} = \underset{W,b}{\operatorname{argmin}} \|W\|^2$$
$$s.t.\ \forall\ i \quad Y_i(W^T X_i - b) \geq 1$$

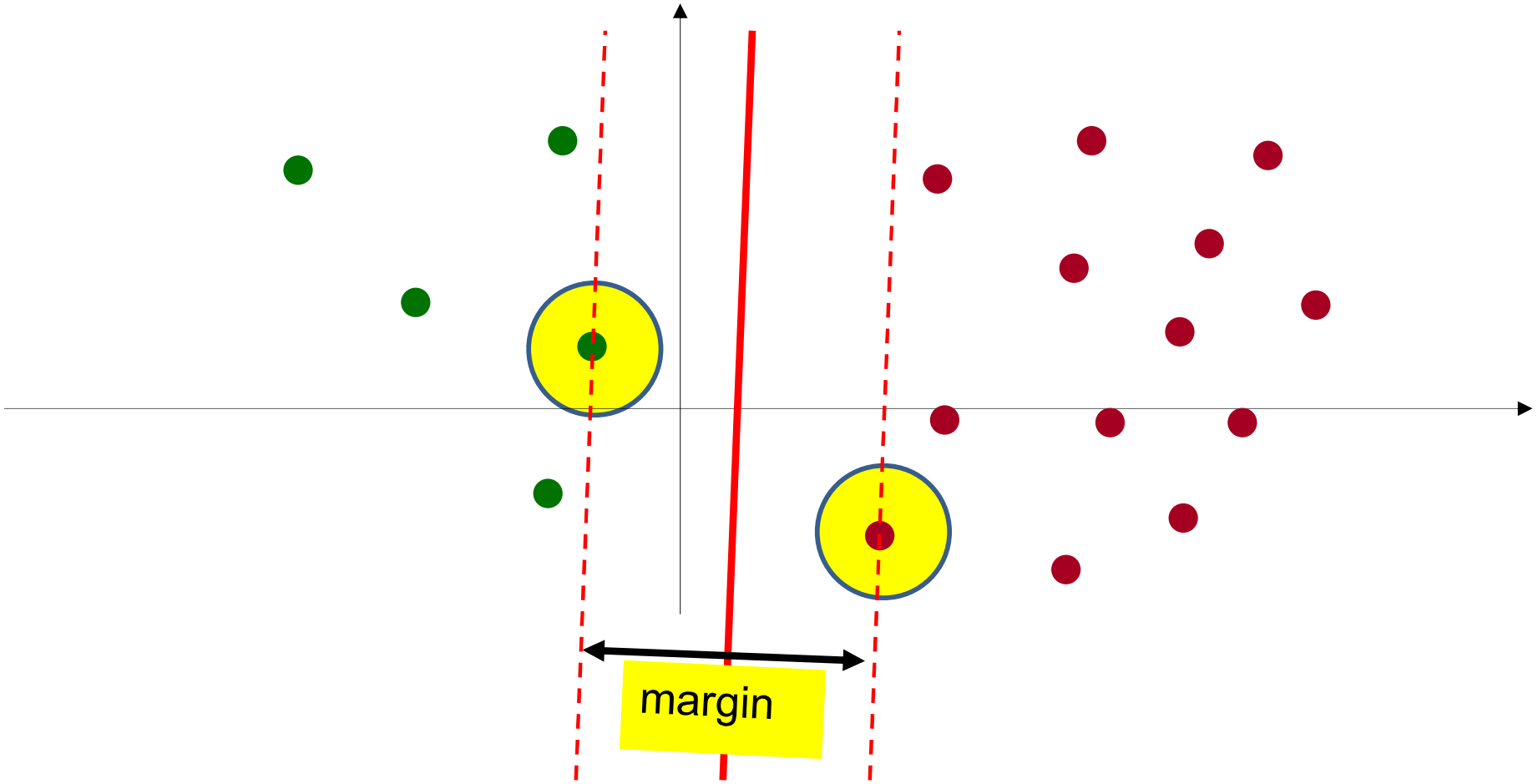
Solving the optimization

- This is a quadratic programming problem!

$$\widehat{W} = \underset{W,b}{\operatorname{argmin}} \|W\|^2$$
$$s.t.\ \forall\ i \quad Y_i(W^T X_i - b) \geq 1$$

- A variety of techniques can be applied
  - Interior point methods, active set methods, gradient descent, conjugate gradient
  - The objective function is convex, QP *will* find the (near) optimal solution
- Most useful solution is based on *Lagrangian duals*
  - Later..

The solution



- Maximizes the *margin*
- This is a *max-margin* classifier
- The boundary samples are called support vectors
  - All the information about the classifier is in these support vectors

Poll 2

- An SVM can only handle data that are linearly separable, i.e. data perfectly separable by some hyperplane
  - True
  - False

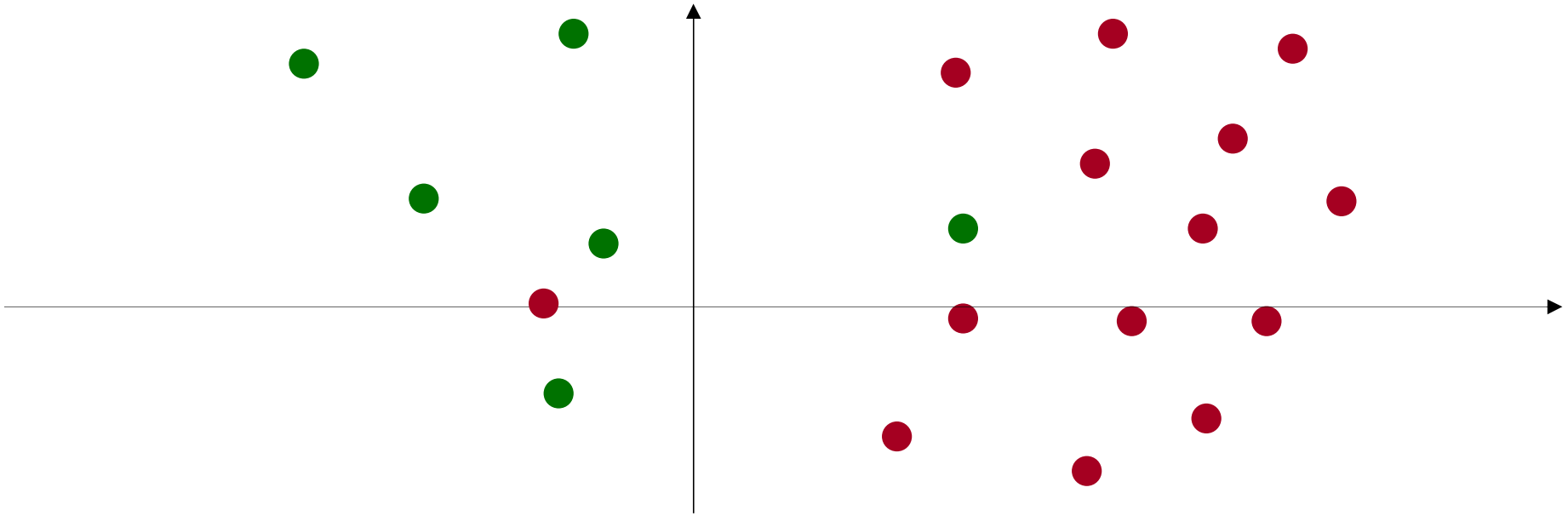
Poll 2

- An SVM can only handle data that are linearly separable, i.e. data perfectly separable by some hyperplane
  - True
  - **False**

Challenges

- What if the classes are not linearly *separable*
- What if the classes are not *linearly* separable?
- What if the classes are not *linearly separable*?

What if they are not separable?



- What if the data are not separable?

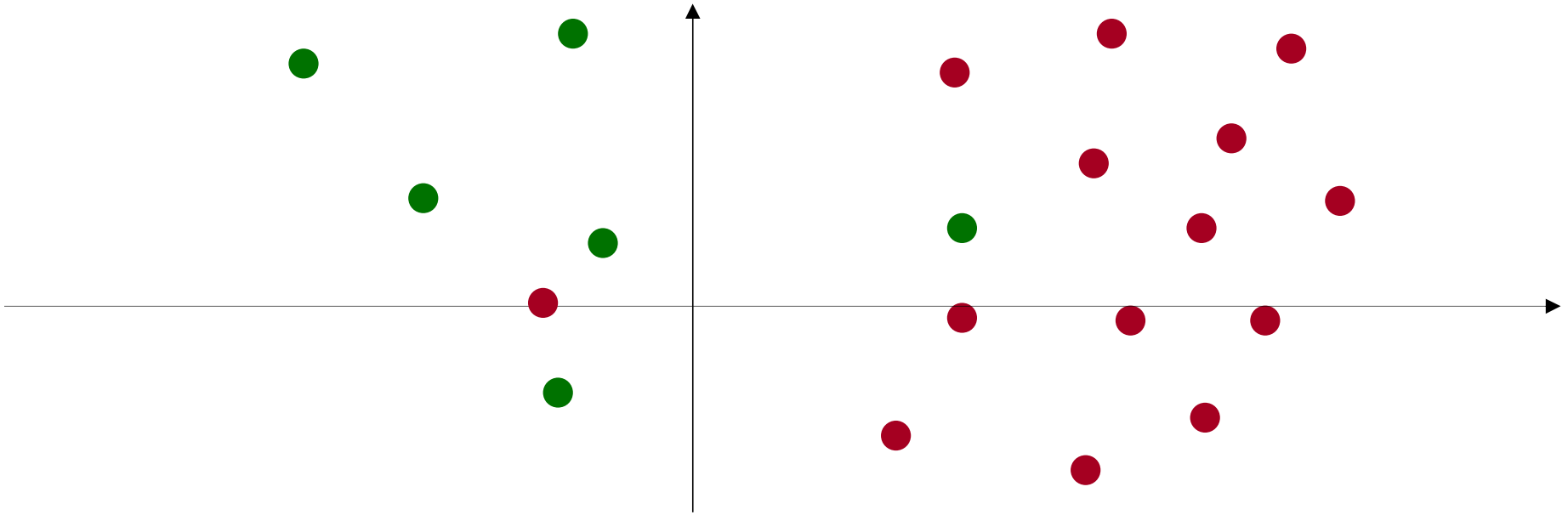
Original Problem

- This is a quadratic programming problem!

$$\widehat{W} = \underset{W}{\operatorname{argmin}} \|W\|^2$$
$$s.t.\ \forall\ i \quad Y_i(W^T X_i - b) \geq 1$$

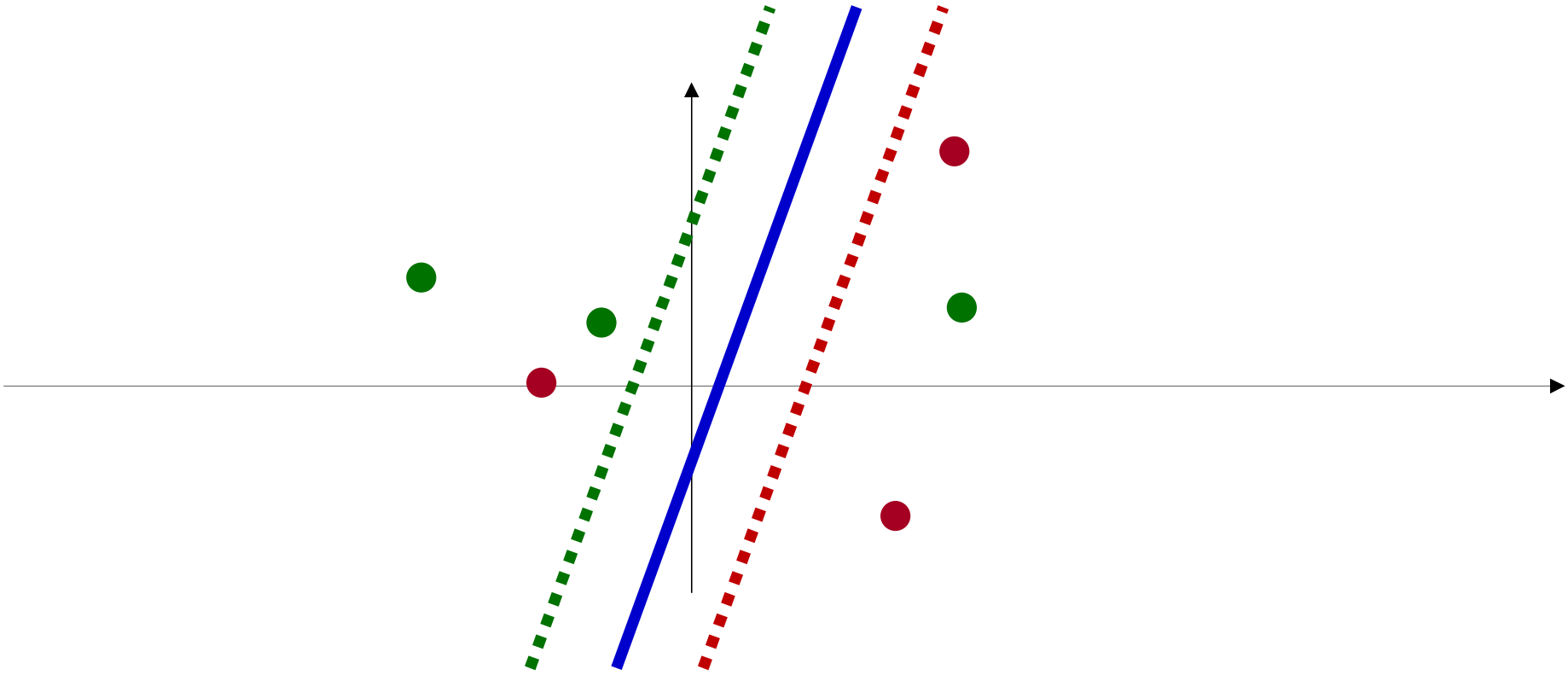
- Maximize the distance between the planes
- Subject to the constraint that all training data instances are on the "correct" side of the plane
- When data are not linearly separable, this constraint can never be satisfied

Introducing the *slack* variable



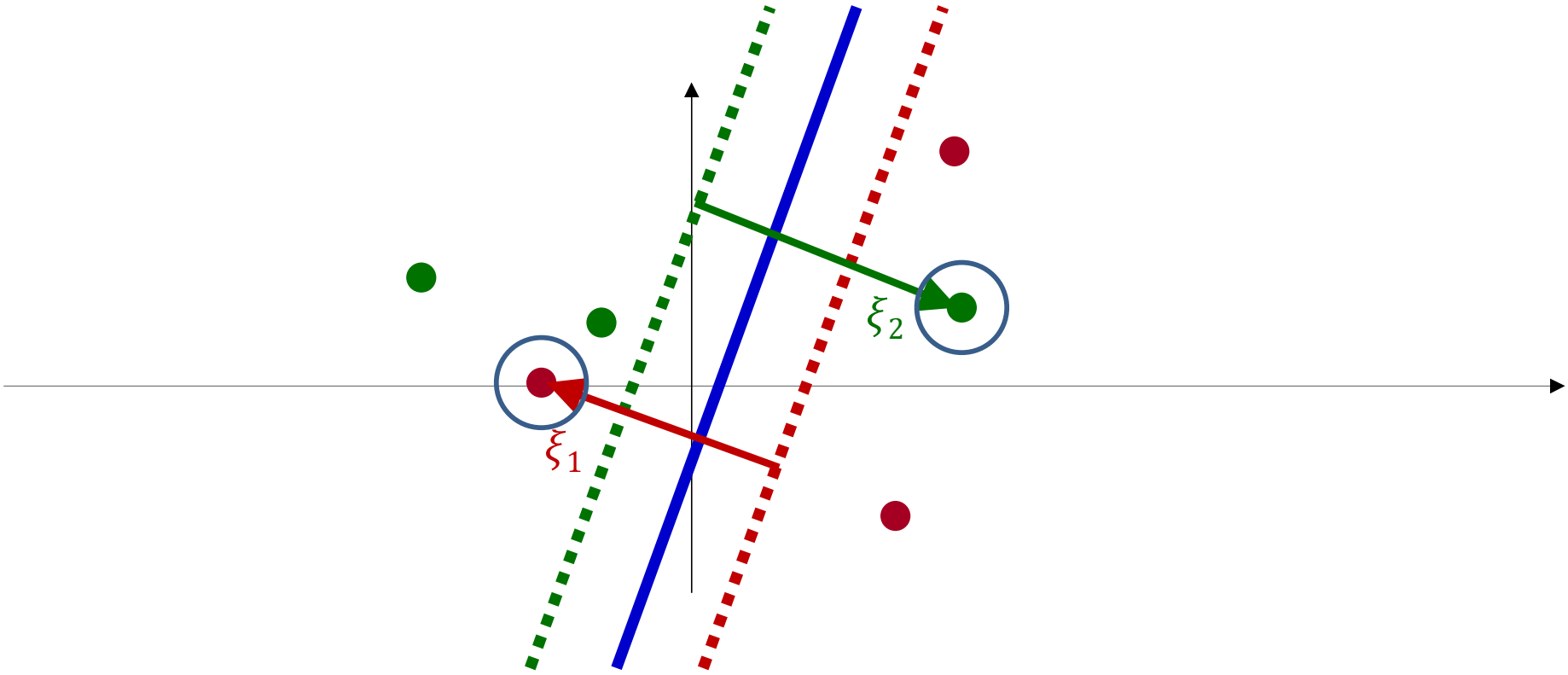
- What if the data are not separable?

Introducing the *slack* variable



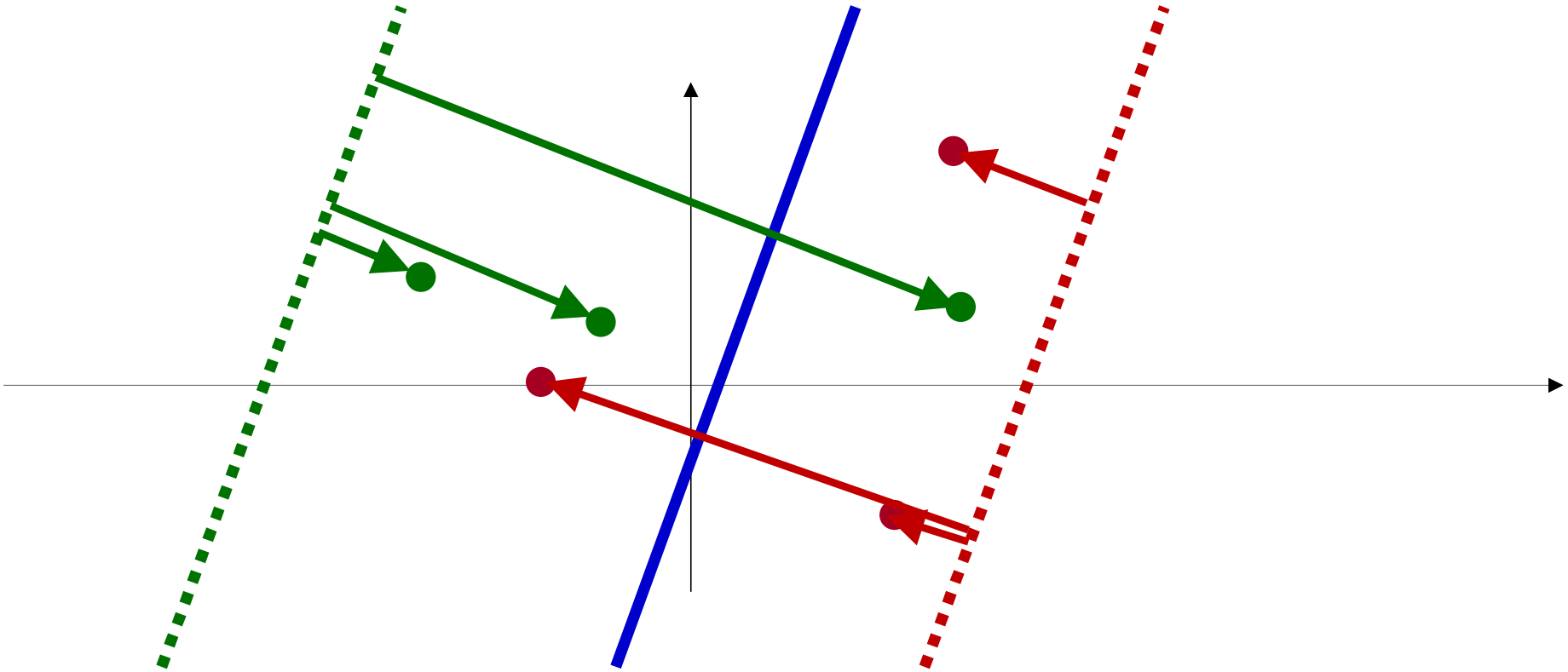
- For every training instance, introduce a *slack* variable ξ
- The slack variable is the maximum distance you have to *shift* the boundary plane to move the point to the "correct" side

Introducing the *slack* variable



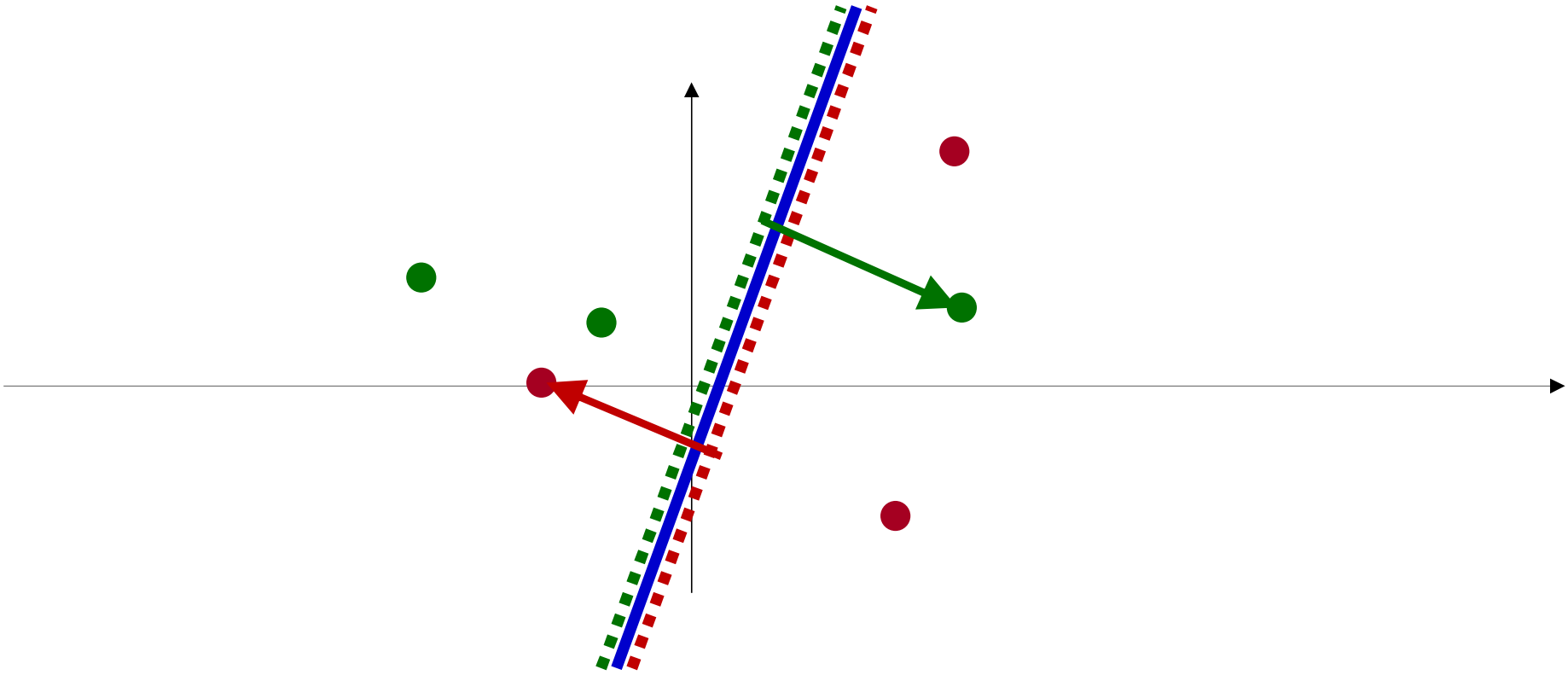
- For every training instance, introduce a *slack* variable ξ
- The slack variable is the *reverse* distance from the *margin* plane of the training instance
  - This will be non-zero only for some instances
  - **Ideally this should be minimum**

Introducing the *slack* variable



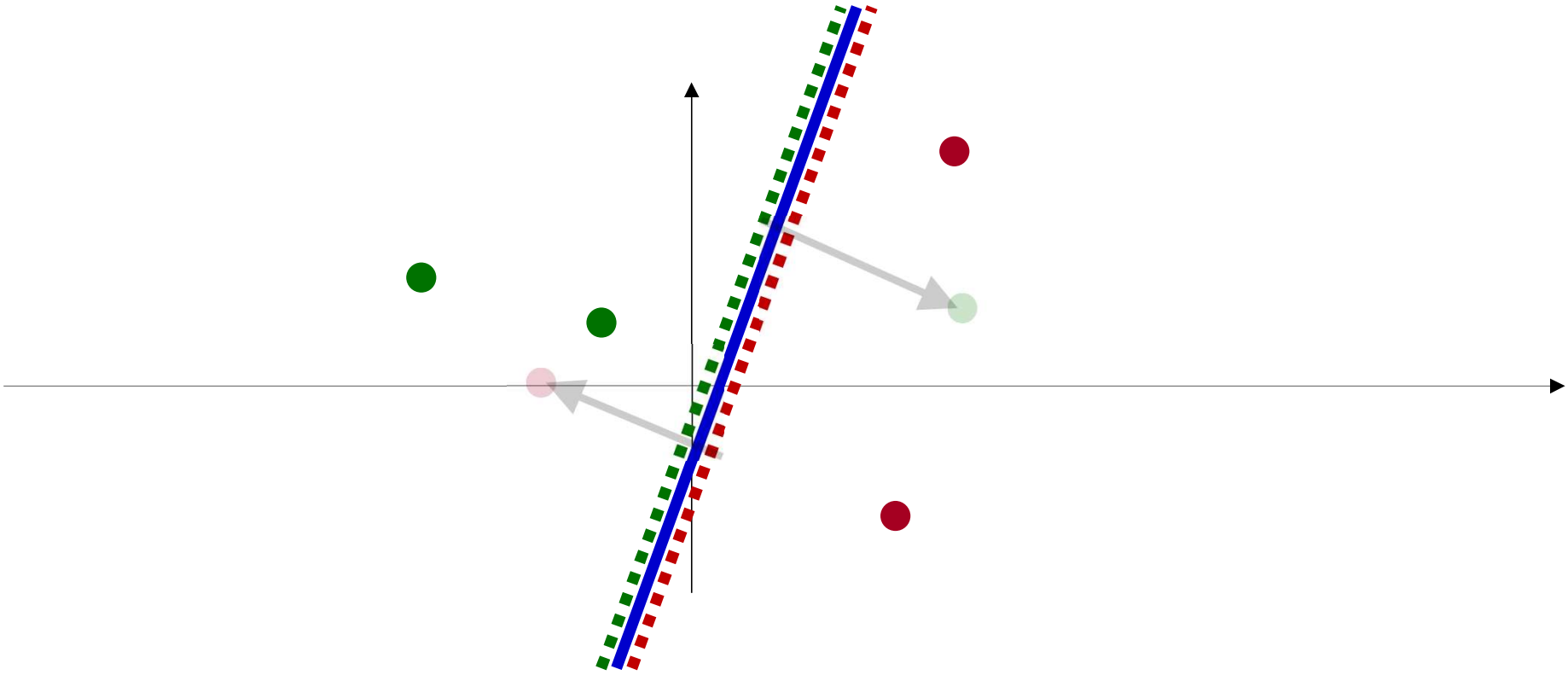
- The total length of slack variables varies with the boundary
- If you push the boundaries too far you will have a greater length of slack variable
  - Which contradicts our desire that they should be minimum

Introducing the *slack* variable



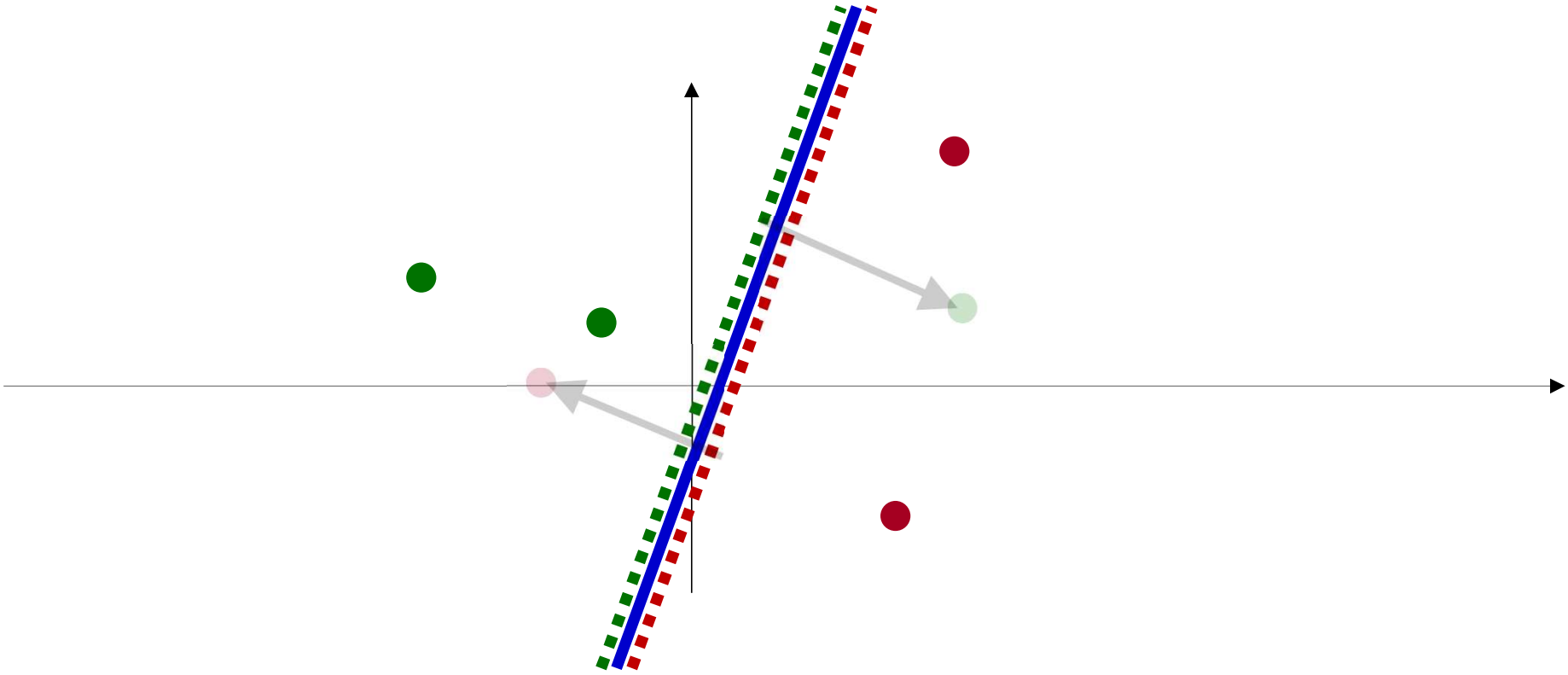
- If they are very close, only the *inseparable points* will have non-zero slack variable
  - The minimum slack value is when the margin planes coincide with the linear classifier

Introducing the *slack* variable



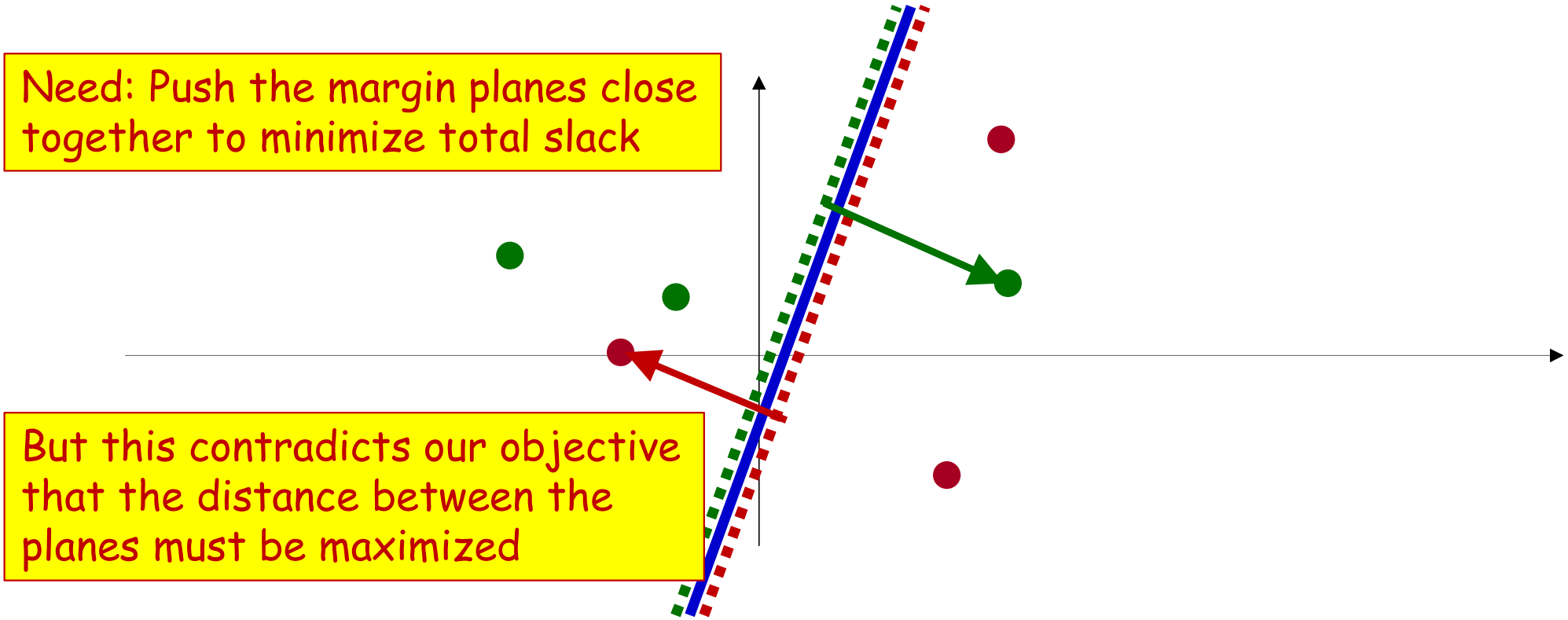
- If they are very close, only the *inseparable points* will have non-zero slack variable
  - The minimum slack value is when the margin planes coincide with the linear classifier
- For linearly separable classes, if the boundary planes are close enough, the total slack length will be 0

Introducing the *slack* variable



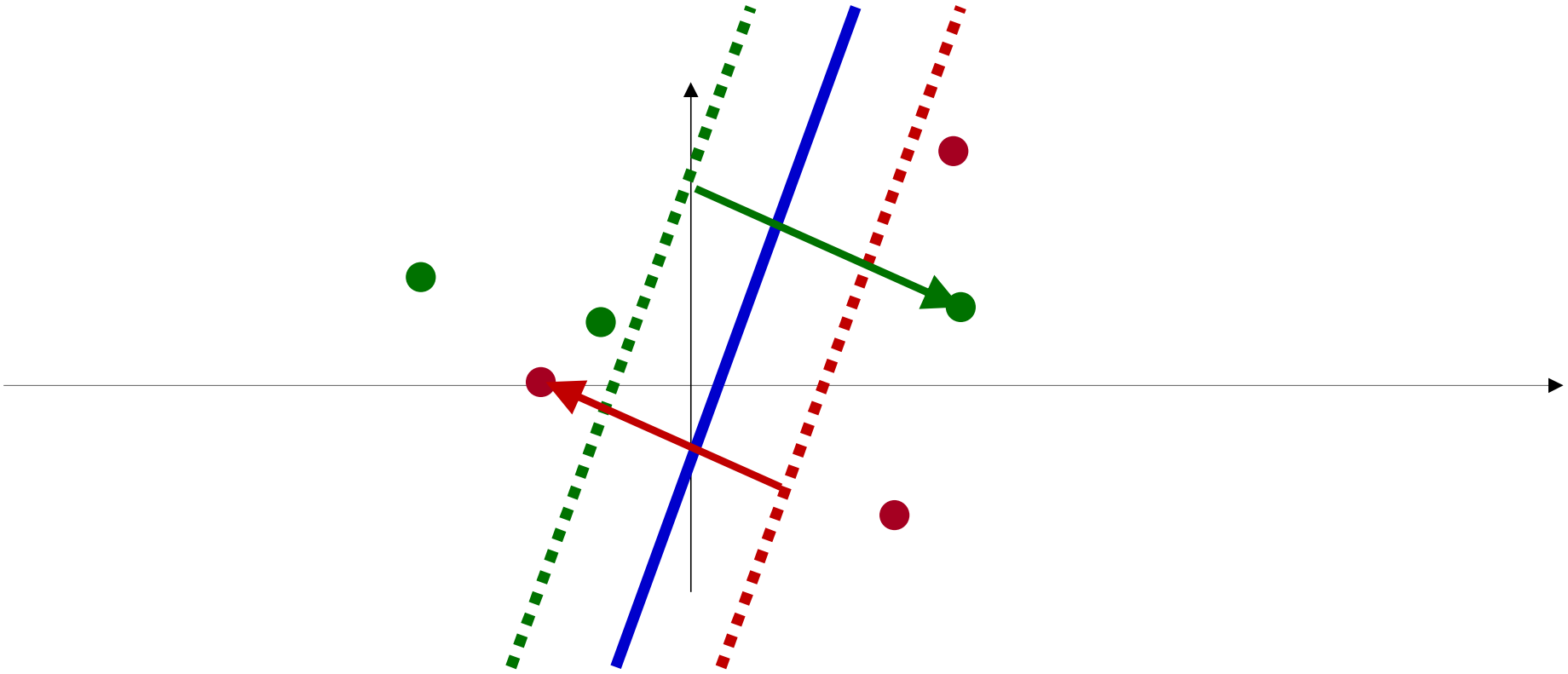
- Problem: If they are too close, the planes violate our desire to *maximize* the margin

Introducing the *slack* variable



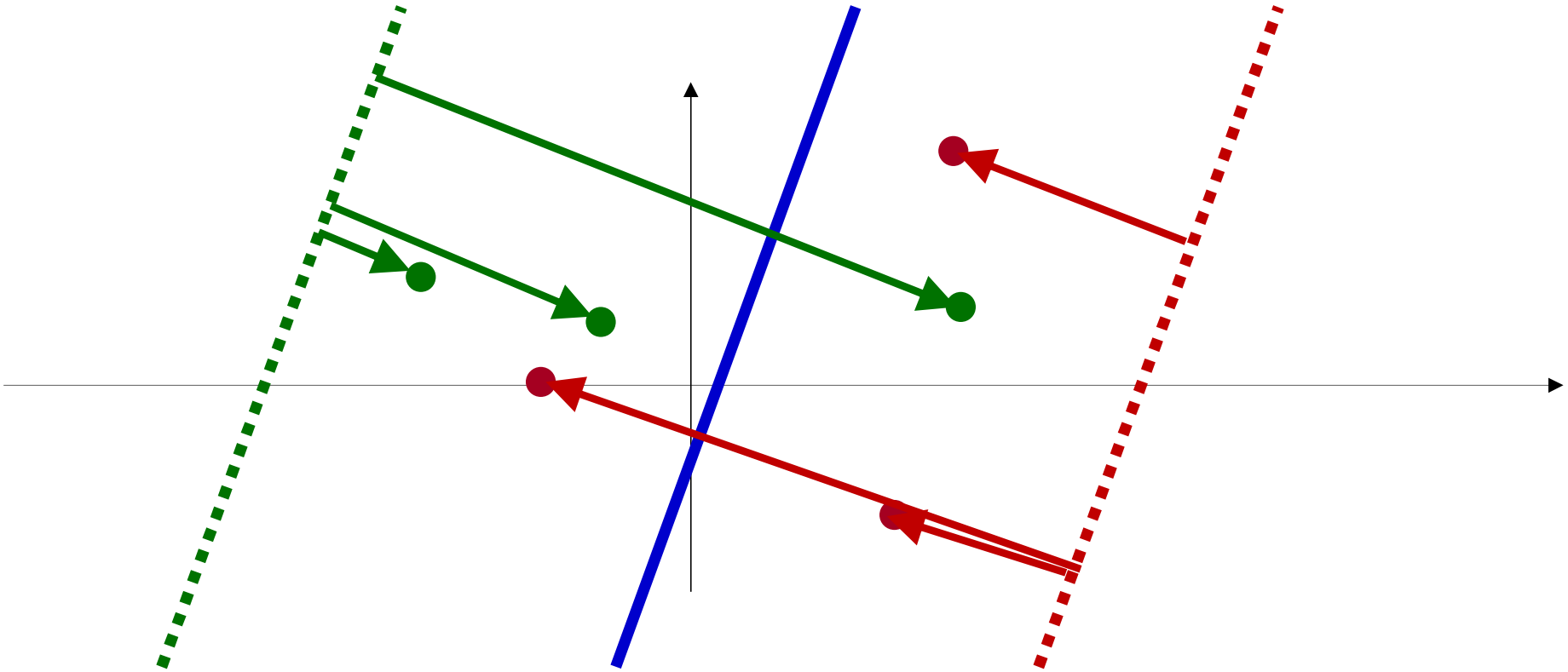
- Contradicting requirements..

New Objective



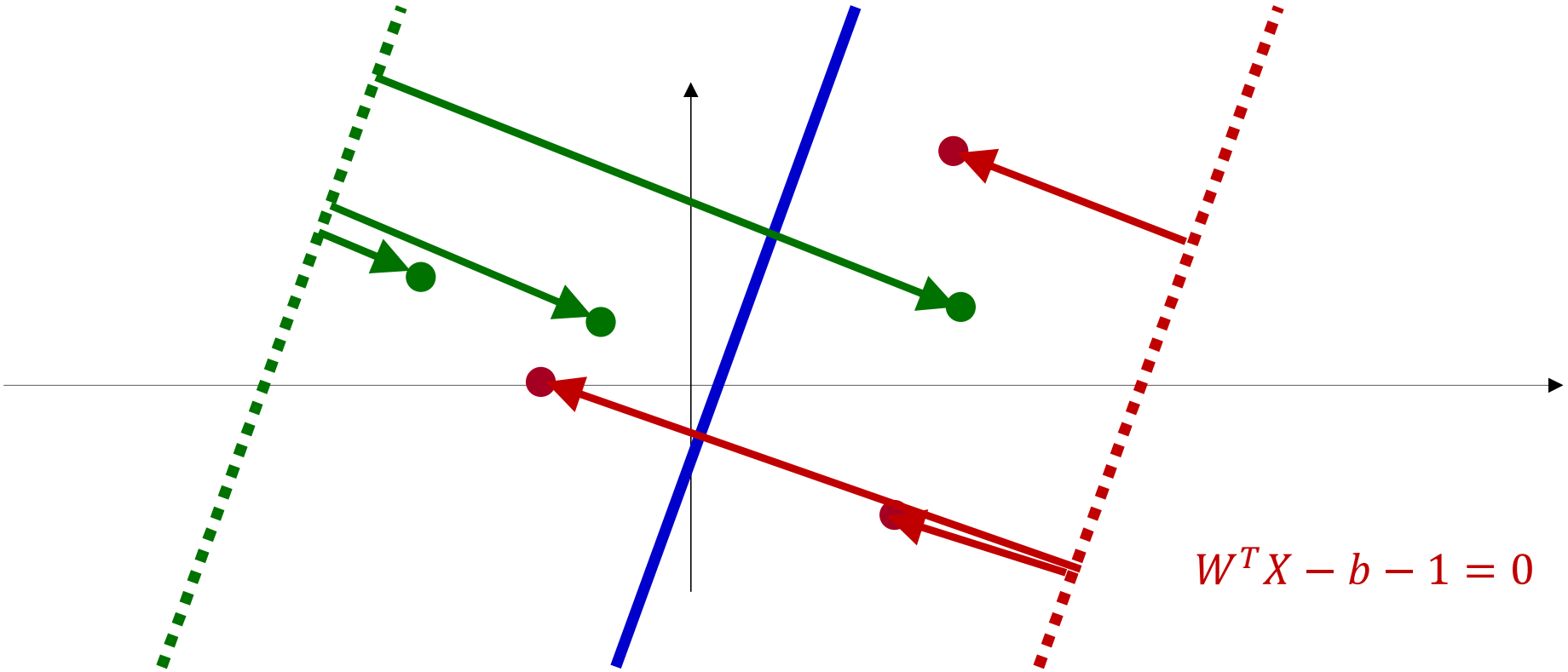
- Simultaneously
  - Maximize distance between planes
  - Minimize total slack length

Quantifying Slack Length



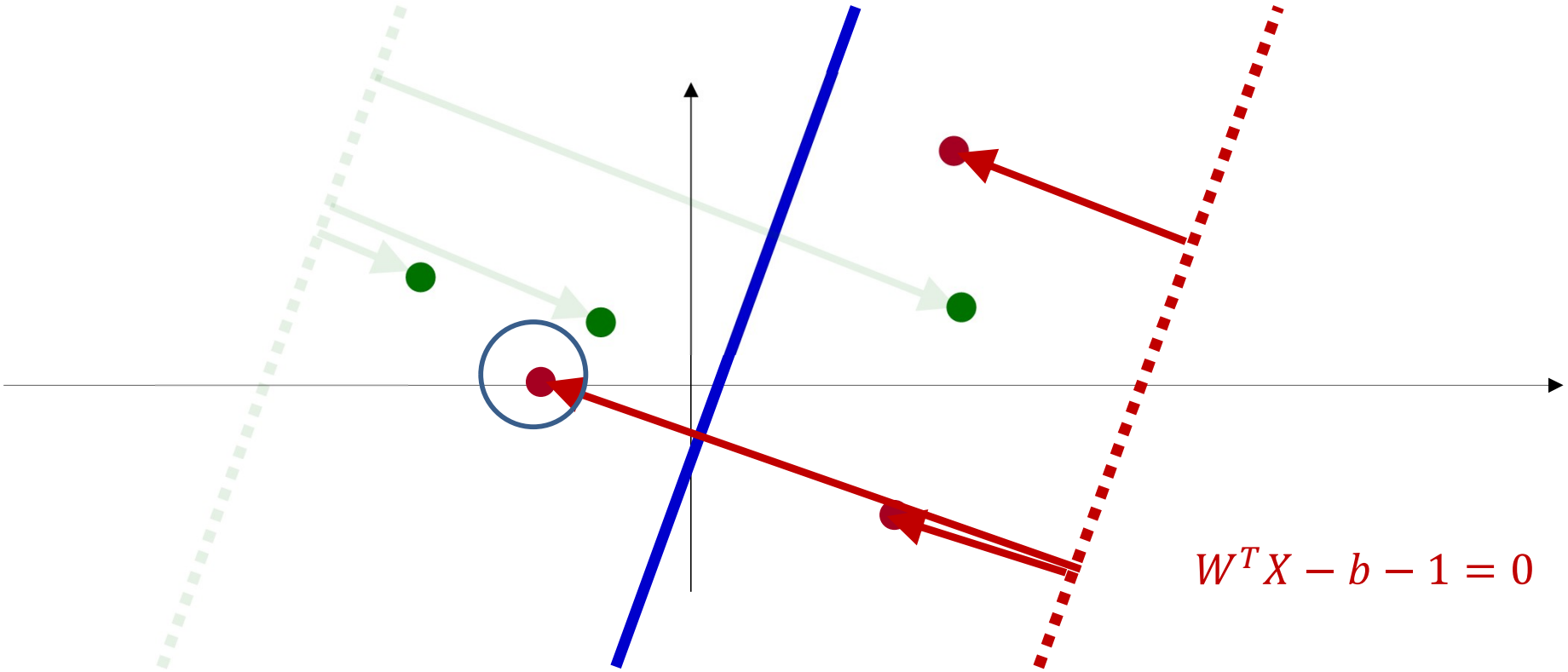
- We need a formula for the total slack length first..

Quantifying Slack Length



- The *positive* margin plane is given by
- $W^T X - b - 1 = 0$
- This plane is at a distance is $\frac{1}{\|W\|}$ from the decision boundary on the *positive* side of the decision plane (in the direction of W)
  - Ideally all positive training points would be to the right of it

Quantifying Slack Length

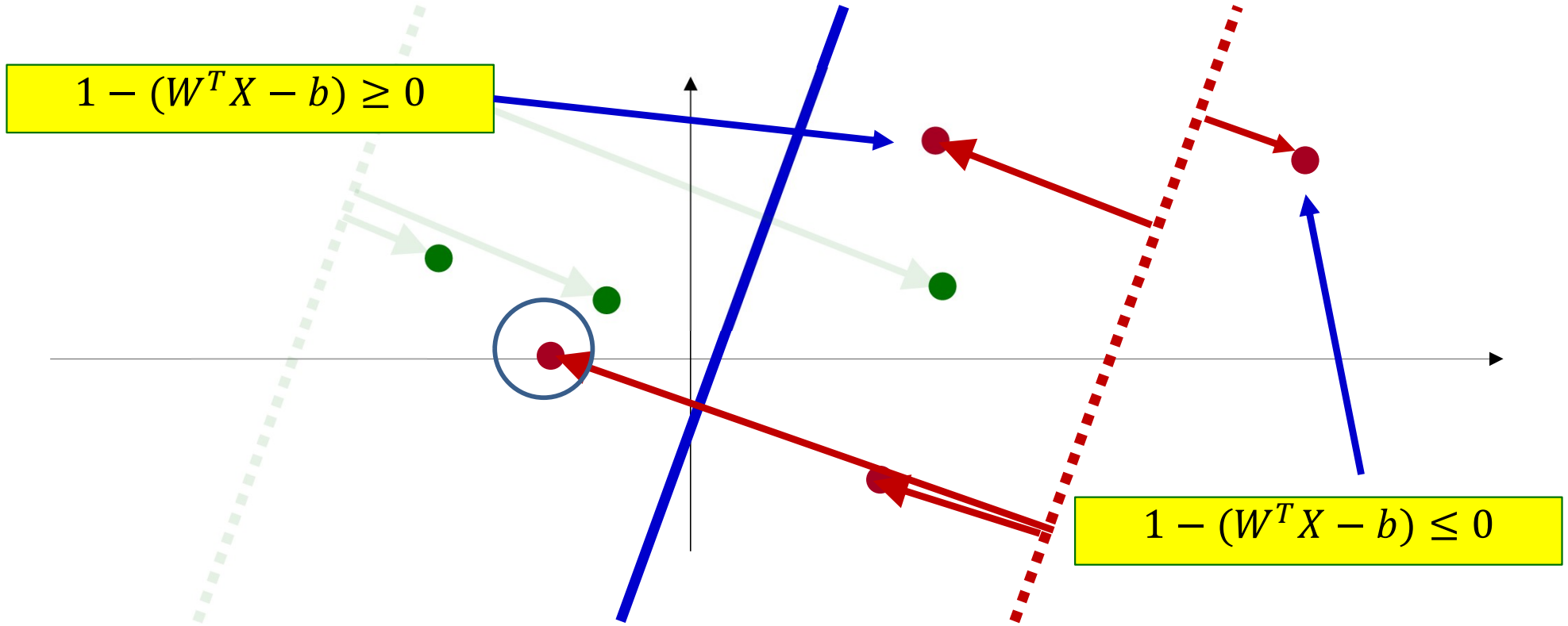


- The (unnormalized) distance of any X from this plane

$$W^T X - b - 1$$

- This will be negative for instances on the "wrong" side (in the direction away from W), but positive for those on the "right" side

Quantifying Slack Length

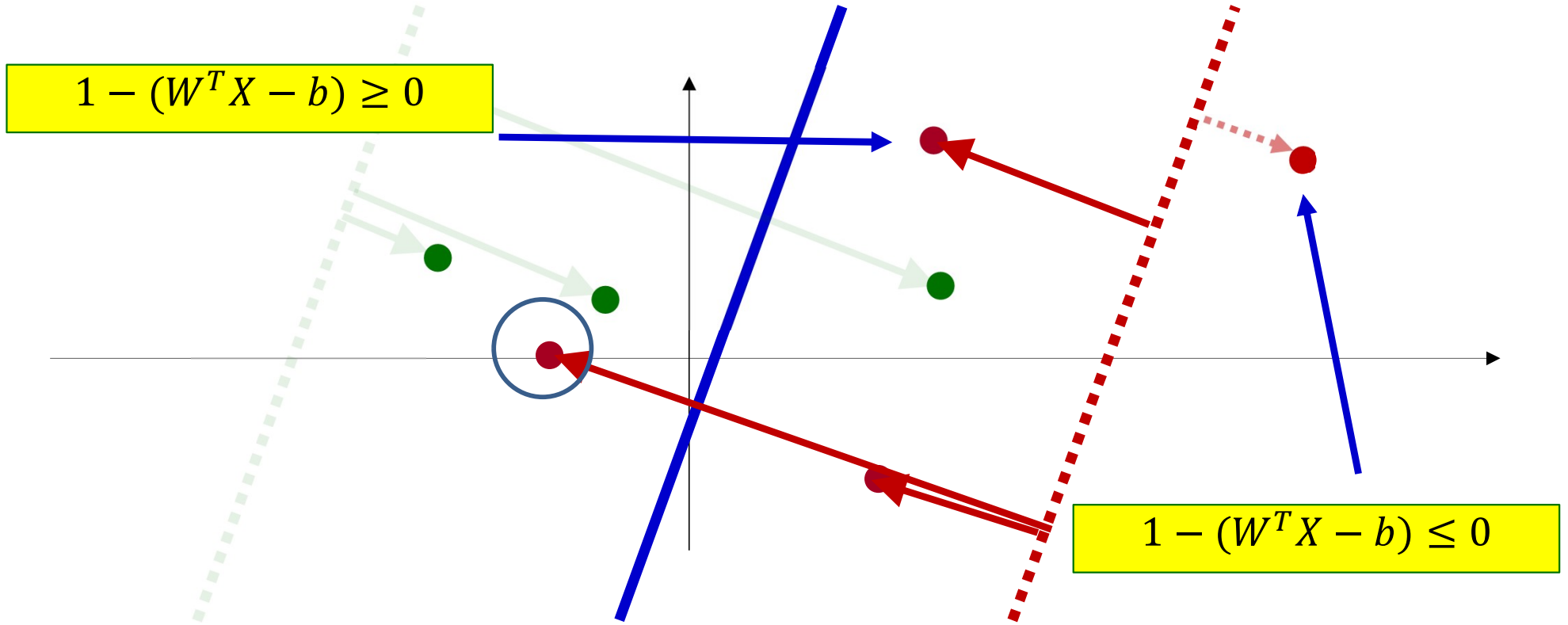


- The *negated* (unnormalized) distance of any X from this plane

$$1 - (W^T X - b)$$

- This will be positive for instances on the wrong side of the margin plane, but negative for instances on the right side of it

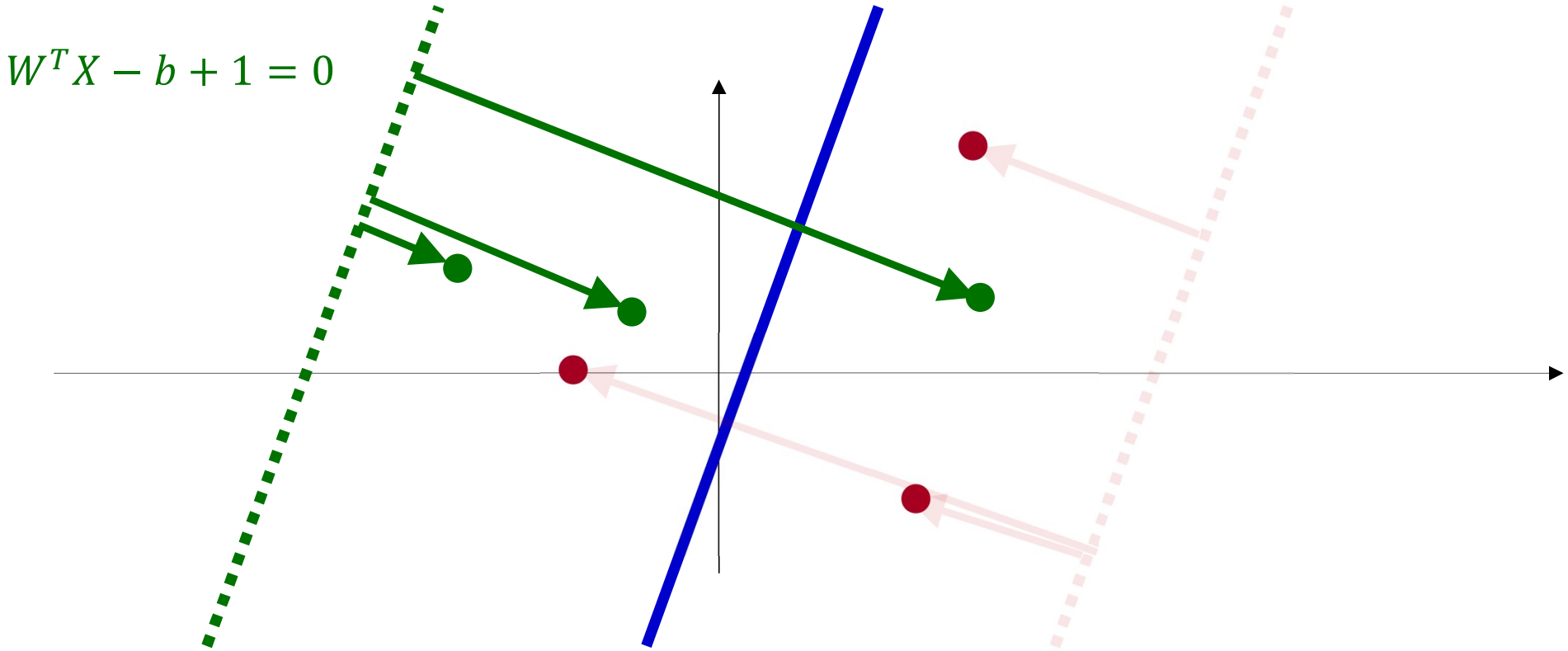
Quantifying Slack Length



- We do not care about the actual distance of instances to the *right* of the plane
- So the slack value of any point is

$$\max(0, 1 - (W^T X - b))$$

Quantifying Slack Length

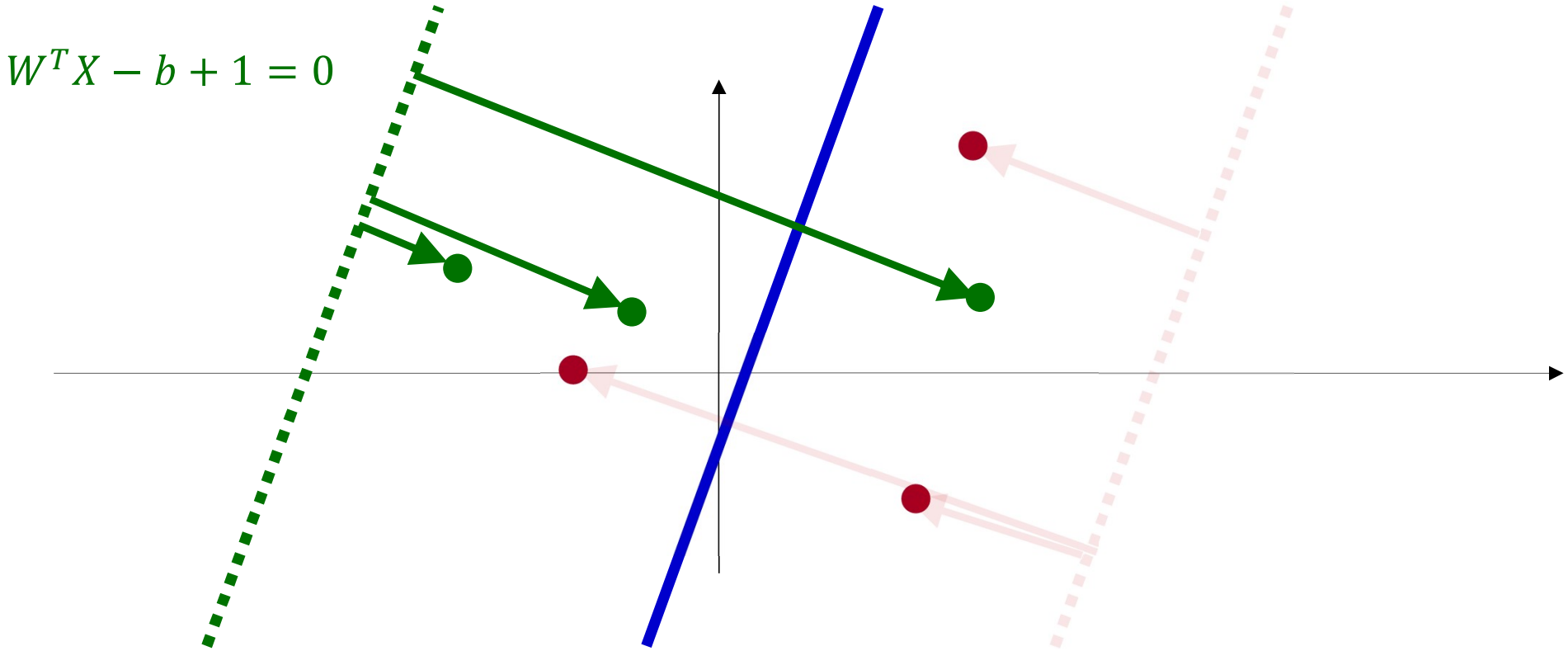


- The *negative* margin plane is given by

$$W^T X - b + 1 = 0$$

  - Ideally all negative training points would be to the left of it

Quantifying Slack Length

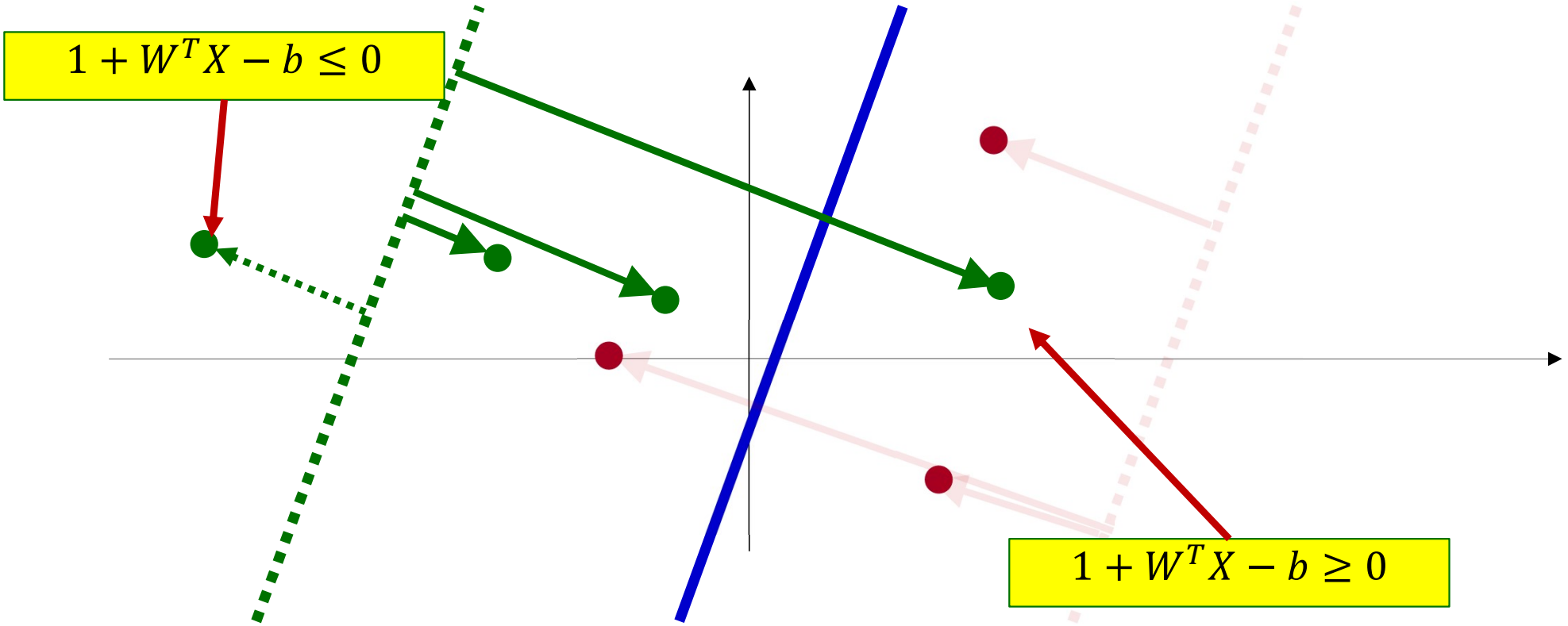


- The (unnormalized) distance of any X from this plane

$$W^T X - b + 1 = 1 + W^T X - b$$

- This will be positive for vectors on the "wrong" side, but negative for vectors on the right side

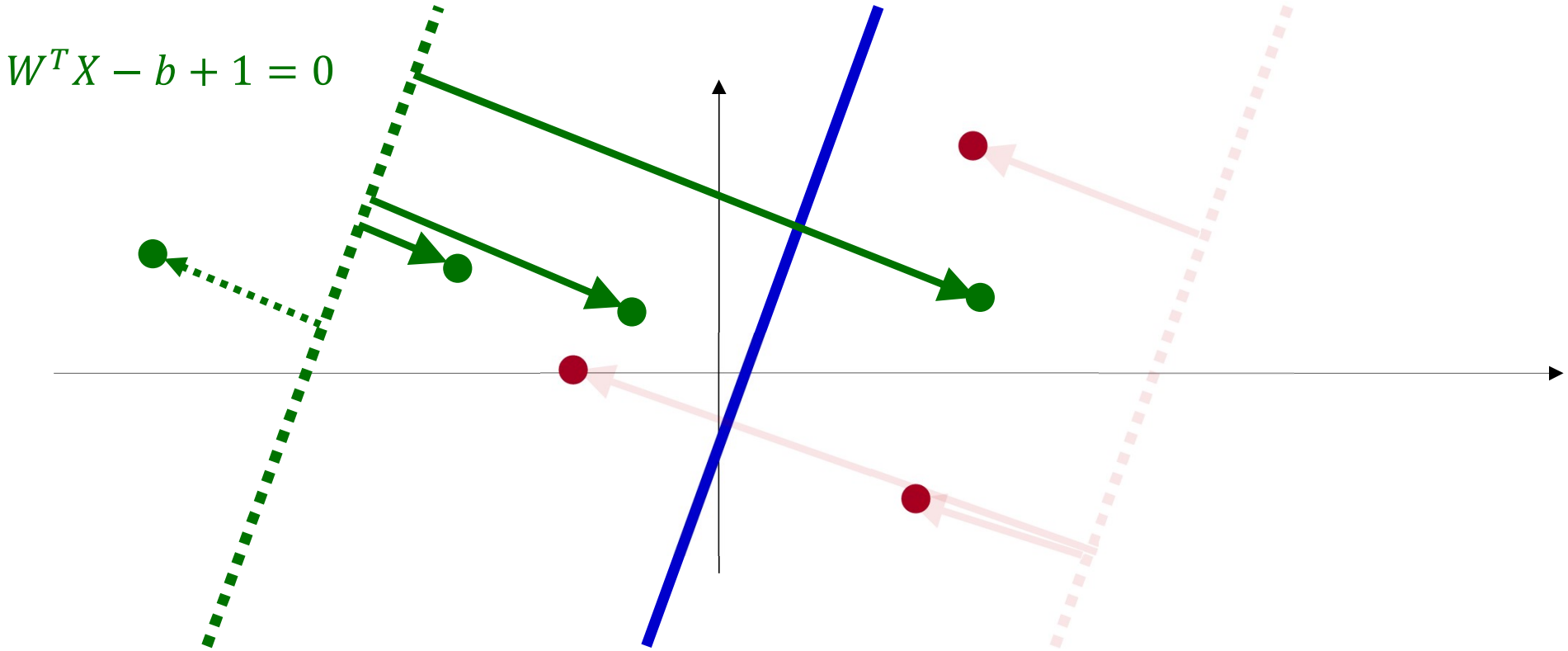
Quantifying Slack Length



- We do not care about the actual distance of instances to the *left* of the plane
- So the slack value of any point is

$$\max(0, 1 + W^T X - b)$$

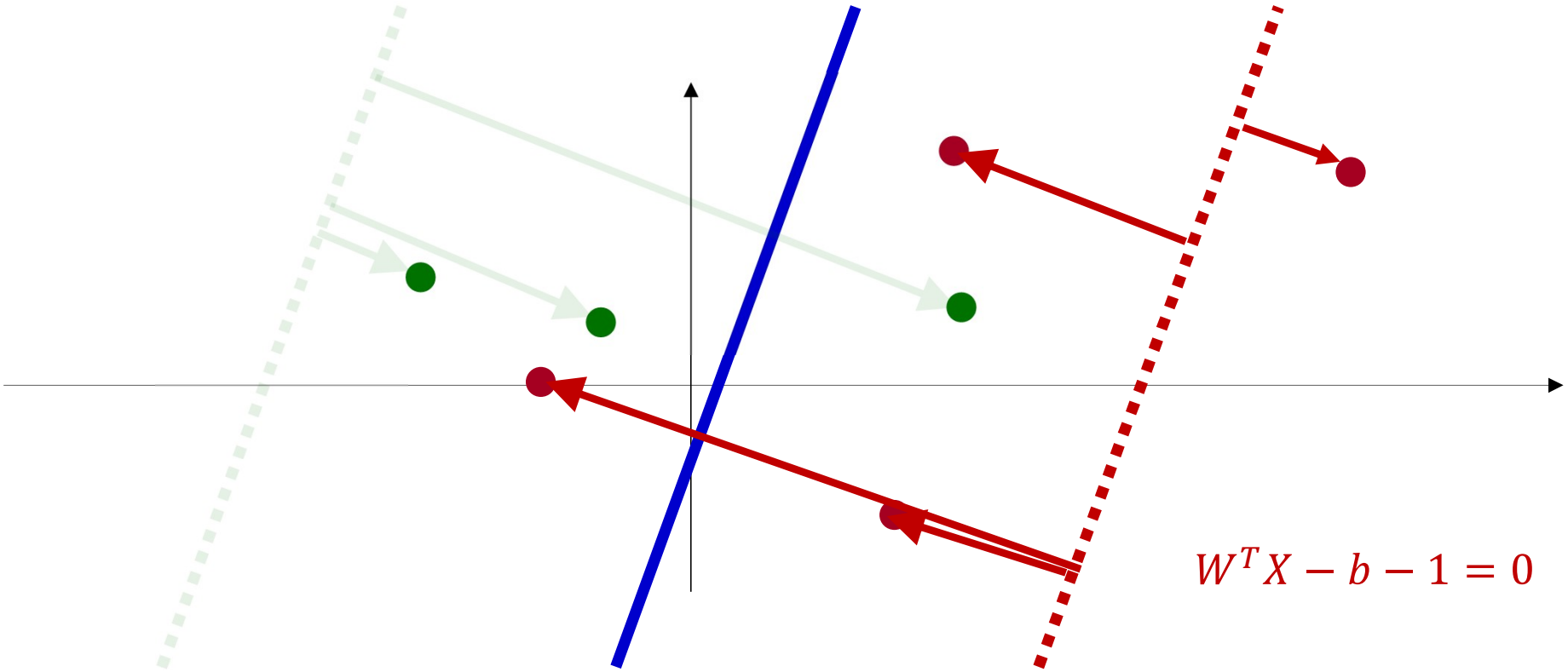
Quantifying Slack Length



- Combining the following for negative instances

$$\max(0, 1 + (W^T X - b))$$

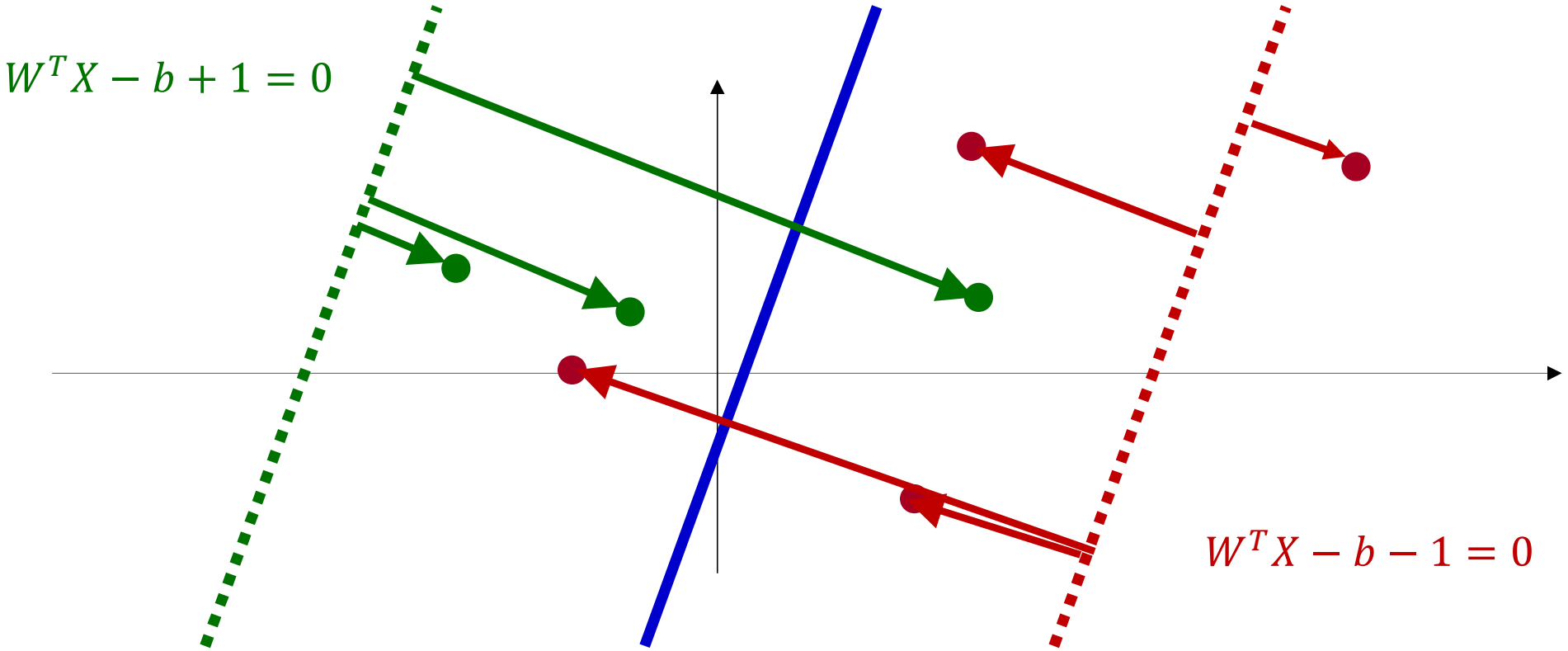
Quantifying Slack Length



- And the following for positive instances

$$\max(0, 1 - (W^T X - b))$$

Quantifying Slack Length



- Generic Slack length for any point

$$\max(0, 1 - y(W^T X - b))$$

- This is also called a *hinge loss*

Total Slack Length

- Total slack length for *all* training instances

$$\sum_i \max(0, 1 - y(W^T X - b))$$

- This must be minimized

Overall Optimization

- Minimize $\|W\|^2$to maximize the distance between margin planes
- Minimize total slack length to minimize the distance of *misclassified* instances to margin planes

$$\sum_i \max(0, 1 - y(W^T X - b))$$

  - This will make the margin planes *closer*
- The two objectives must be traded off..

Support Vector Machine for Inseparable data

- Minimize

$$\underset{W,b}{\operatorname{argmin}} \frac{1}{N} \sum_{i} \max(0, 1 - y(W^T X - b)) \quad + \lambda \|W\|^2$$

- λ is a "regularization" parameter that decides the relative importance of the two terms

- This is just a regular optimization problem that can be solved through gradient descent

Support Vector Machine for Inseparable data

- λ is typically set using *held-out* training data
  - Train the classifier for various values of λ
  - Test each of these classifiers on some held-out portion of the training data that was not included in training the SVM
  - Pick the λ for which the classifier gave best performance
  - Retrain the SVM using the entire training data and this λ

- Frequently, instead of a single held-out set, λ is set through K-fold cross validation

Equivalent Slack Formalism

$$\underset{W,b}{\operatorname{argmin}} \ \|W\|^2 + C \sum_i \xi_i$$

- Subject to

$$Y_i(W^T X_i - b) \geq 1 - \xi_i$$

- This is a quadratic programming problem
- Slack parameter C is determined through held-out data as earlier (or through K-fold cross-validation)

Poll 3

- Mark all that are true
  - We must use the information from all the vectors in the training samples to calculate the support vectors to get the hyperplane for classification
  - A classifier can be represented by sign(w^T x – b) for any separating hyperplane
  - SVM can find the classifier that has the maximal distance from the closest instances of either class, so that it may maximize the chance that the classifier will work on new, unseen data
  - We can turn the problem of finding w^T x – b = 0 to a quadratic programming problem, so that we can use all kinds of convex optimization algorithms on constrained sets, to solve it

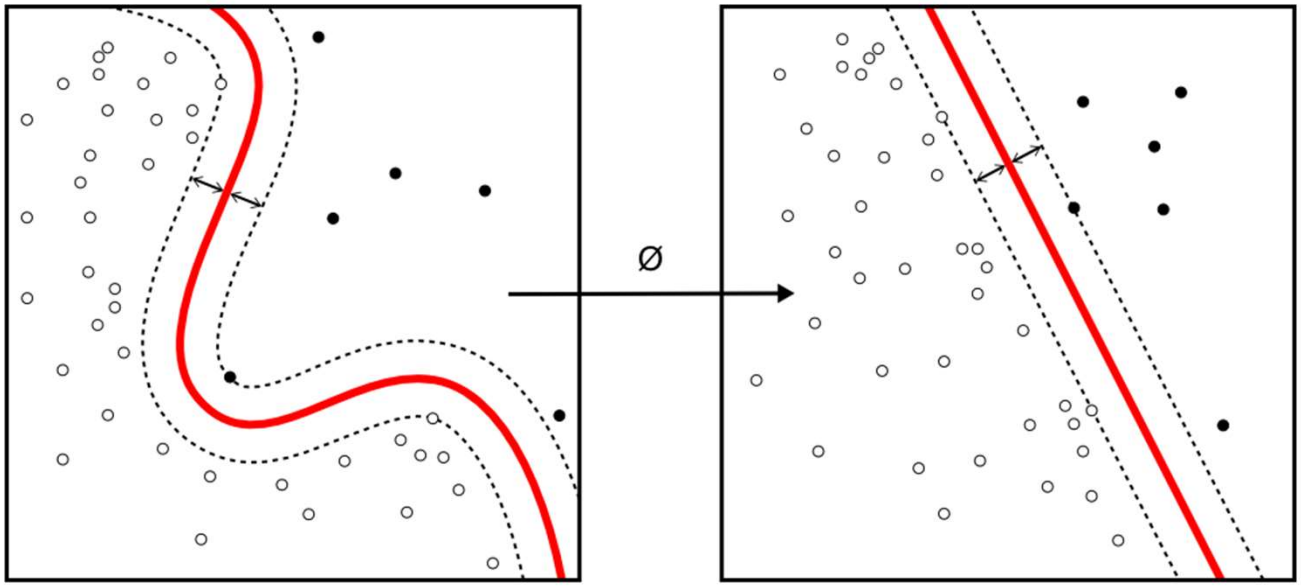
- Mark all that are true
  - We can use SVM with slack variables for the case where the data are linearly separable except for some outlier instances
  - We need to maximize the margin, as well as the slack variables in this case
  - We can also use convex optimization methods such as Lagrangian duals to solve it, even when introducing slack variables
  - We can use K-fold cross-validation to determine the parameter C which is the parameter for the slack variable in the quadratic programming loss function

Poll 3

- Mark all that are true
  - We must use the information from all the vectors in the training samples to calculate the support vectors to get the hyperplane for classification
  - A classifier can be represented by sign(w^T x – b) for any separating hyperplane
  - SVM can find the classifier that has the maximal distance from the closest instances of either class, so that it may maximize the chance that the classifier will work on new, unseen data
  - We can turn the problem of finding w^T x – b = 0 to a quadratic programming problem, so that we can use all kinds of convex optimization algorithms on constrained sets, to solve it

- Mark all that are true
  - We can use SVM with slack variables for the case where the data are linearly separable except for some outlier instances
  - We need to maximize the margin, as well as the slack variables in this case
  - We can also use convex optimization methods such as Lagrangian duals to solve it, even when introducing slack variables
  - We can use K-fold cross-validation to determine the parameter C which is the parameter for the slack variable in the quadratic programming loss function

How to deal with *non-linear* boundaries?



- First some math..

Recall: The Lagrange Method

- Optimize $f(x, y)$ subject to $g(x, y) = c$

$$L(x, y, \lambda) = f(x, y) - \lambda(g(x, y) - c)$$

to maximize $f(x, y)$: $\max_{x,y} \left(\min_{\lambda} L(x, y, \lambda) \right)$

to minimize $f(x, y)$: $\min_{x,y} \left(\max_{\lambda} L(x, y, \lambda) \right)$

Optimization with inequality constraints

- Optimization problem with constraints

$$\min_x f(x)$$
$$s.t.\ \ g_i(x) \le 0,\ \ i = \{1,\ldots,k\}$$
$$h_j(x) = 0,\ \ j = \{1,\ldots,l\}$$

- Lagrange multipliers $\lambda_i \ge 0, \nu \in \Re$

$$L(x,\lambda,\nu) = f(x) + \sum_{i=1}^{k} \lambda_i g_i(x) + \sum_{j=1}^{l} \nu_j h_j(x)$$

- The optimization problem

$$\operatorname*{argmin}_x \max_{\lambda,\nu} L(x,\lambda,\nu)$$

Revisiting the *linearly separable* case

- This is a quadratic programming problem!

$$\widehat{W} = \underset{W}{\operatorname{argmin}} \|W\|^2$$
$$s.t.\ \forall\, i \quad Y_i(W^T X_i - b) \geq 1$$

- Can be stated using Lagrangians as

$$\underset{W,b}{\operatorname{argmin}} \max_{\alpha>0} \frac{1}{2}\|W\|^2 + \sum_i \alpha_i\big(1 - Y_i(W^T X_i - b)\big)$$

For convenience

Constraint: must be -ve

Linearly separable case: Lagrangian formalism

- Can be stated using Lagrangians as

$$\underset{W,b}{\operatorname{argmin}} \ \max_{\alpha>0} \frac{1}{2}\|W\|^2 + \sum_i \alpha_i\left(1 - Y_i(W^T X_i - b)\right)$$

- The optimum satisfies the *Karush Kuhn-Tucker* conditions, hence we can rewrite it as

$$\underset{\alpha>0}{\operatorname{argmax}} \ \min_{W,b} \frac{1}{2}\|W\|^2 + \sum_i \alpha_i\left(1 - Y_i(W^T X_i - b)\right)$$

Linearly separable case: Lagrangian formalism

- Under the KKT conditions

$$\underset{\alpha>0}{\operatorname{argmax}} \min_{W,b} \frac{1}{2}\|W\|^2 + \sum_i \alpha_i \left(1 - Y_i(W^T X_i - b)\right)$$

- Taking the deriviative w.r.t W and setting to 0, we get

$$W = \sum_i \alpha_i Y_i X_i$$

Linearly separable case: Lagrangian formalism

- Under the KKT conditions

$$\underset{\alpha>0}{\operatorname{argmax}} \min_{W,b} \frac{1}{2}\|W\|^2 + \sum_i \alpha_i \left(1 - Y_i(W^T X_i - b)\right)$$

- Taking the deriviative w.r.t b and setting to 0, we get

$$0 = \sum_i \alpha_i Y_i$$

Linearly separable case:

- Restating (and ignoring the factor of 2)

$$\underset{\alpha>0}{\operatorname{argmax}} \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j Y_i Y_j X_i^T X_j - b\sum_i \alpha_i Y_i$$

- Since the last term is 0

$$\underset{\alpha}{\operatorname{argmax}} \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j Y_i Y_j X_i^T X_j$$

$$s.t.\ \alpha_i \geq 0$$

$$\textstyle\sum_i \alpha_i Y_i = 0$$

Large margin linear classifier

- Solve for α_i

$$\underset{\alpha}{\text{argmax}} \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j Y_i Y_j X_i^T X_j$$

$$s.t.\ \alpha_i \geq 0$$

$$\sum_i \alpha_i Y_i = 0$$

- α_i will turn out to be non-zero only for the support vectors (and 1 for the support vectors)

Large margin linear classifier with slack

- Solve for α_i

$$\underset{\alpha}{\text{argmax}} \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j Y_i Y_j X_i^T X_j$$

$$s.t. \; C \geq \alpha_i \geq 0$$

$$\sum_i \alpha_i Y_i = 0$$

- Note upper bound on α_i
- α_i will turn out to be non-zero only for the support vectors (and 1 for the support vectors)

The usual simple SVM can also be solved through the ugly form

$$\underset{\alpha}{\text{argmax}} \sum_i \alpha_i - \sum_{i,j} \alpha_i \alpha_j Y_i Y_j X_i^T X_j$$

$$s.t. \; C \geq \alpha_i \geq 0$$

$$\sum_i \alpha_i Y_i = 0$$

- This is for the linear case. Note that the optimization is in terms of $X_i^T X_j$
- Also $W = \sum_i \alpha_i Y_i X_i$
- So the classifier on any test instance has the form:

$$sign\left(\sum_i \alpha_i Y_i X_{test}^T X_i - b\right)$$

The SVM as KNN classification

$$\underset{\alpha}{\text{argmax}} \sum_i \alpha_i - \sum_{i,j} \alpha_i \alpha_j Y_i Y_j X_i^T X_j$$

$$s.t. \; C \geq \alpha_i \geq 0$$

$$\sum_i \alpha_i Y_i = 0$$

Weighted-nearest neighbor classifier

- This is for the linear case. Note that the optimization is in terms of $X_i^T X_j$
- Also $W = \sum_i \alpha_i Y_i X_i$
- So the classifier on any test instance has the form:

$$sign\left(\sum_i \alpha_i Y_i X_{test}^T X_i - b\right)$$

The SVM as KNN classification

$$\underset{\alpha}{\text{argmax}} \sum_i \alpha_i - \sum_{i,j} \alpha_i \alpha_j Y_i Y_j X_i^T X_j$$

L1 norm of α

$s.t. C \geq \alpha_i \geq 0$

Total weighted accuracy on training data

Weighted-nearest neighbor classifier

- This is for the linear case. Note that the optimization is in terms of $X_i^T X_j$
- Also $W = \sum_i \alpha_i Y_i X_i$
- So the classifier on any test instance has the form:

$$sign\left(\sum_i \alpha_i Y_i X_{test}^T X_i - b\right)$$

The Kernel Trick

$$\underset{\alpha}{\text{argmax}} \sum_i \alpha_i - \sum_{i,j} \alpha_i \alpha_j Y_i Y_j X_i^T X_j$$

$$s.t. C \geq \alpha_i \geq 0$$

$$\sum_i \alpha_i Y_i = 0$$

- This is for the linear case. Note that the optimization is in terms of $X_i^T X_j$
- Also $W = \sum_i \alpha_i Y_i X_i$
- So the classifier on any test instance has the form:

$$sign\left(\sum_i \alpha_i Y_i X_{test}^T X_i - b\right)$$

The Kernel Trick

$$\underset{\alpha}{\text{argmax}} \sum_i \alpha_i - \sum_{i,j} \alpha_i \alpha_j Y_i Y_j K(X_i, X_j)$$

$$s.t. C \geq \alpha_i \geq 0$$

$$\sum_i \alpha_i Y_i = 0$$

- For classification:

$$sign\left(\sum_i \alpha_i Y_i K(X_i, X_{test}) - b\right)$$

The Kernel Trick

$$\underset{\alpha}{\operatorname{argmax}} \sum_i \alpha_i - \sum_{i,j} \alpha_i \alpha_j Y_i Y_j K(X_i, X_j)$$

$$s.t. \; C \geq \alpha_i \geq 0$$

$$\sum_i \alpha_i Y_i = 0$$

This is a quadratic programming problem

- For classification:

$$sign\left(\sum_i \alpha_i Y_i K(X_i, X_{test}) - b\right)$$

Poll 4

- We can deal with non-linear decision boundaries using Kernel functions
  - True
  - False

- The KKT condition is an extension of the Lagrange multiplier method. We can apply it to the Lagrangian dual of any primal optimization problem to obtain the modified equation that the optimal solution must satisfy (if KKT is satisfied)
  - True
  - False

Poll 4

- We can deal with non-linear decision boundaries using Kernel functions
  - **True**
  - False

- The KKT condition is an extension of the Lagrange multiplier method. We can apply it to the Lagrangian dual of any primal optimization problem to obtain the modified equation that the optimal solution must satisfy (if KKT is satisfied)
  - **True**
  - False

Nonlinear SVMs: The Kernel Trick

- Examples of commonly-used kernel functions:
  - Linear kernel: $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$
  - Polynomial kernel: $K(\mathbf{x}_i, \mathbf{x}_j) = (1 + \mathbf{x}_i^T \mathbf{x}_j)^p$
  - Gaussian (Radial-Basis Function (RBF)) kernel:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\left\|\mathbf{x}_i - \mathbf{x}_j\right\|^2}{2\sigma^2}\right)$$

  - Sigmoid:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\beta_0 \mathbf{x}_i^T \mathbf{x}_j + \beta_1)$$

- In general, functions that satisfy *Mercer's condition* can be kernel functions.

Nonlinear SVM: Optimization

- Formulation: (Lagrangian Dual Problem)

$$\text{maximize} \quad \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$$

such that

$$0 \le \alpha_i \le C$$

$$\sum_{i=1}^{n} \alpha_i y_i = 0$$

- The solution of the discriminant function is

$$g(\mathbf{x}) = \sum_{i \in \text{SV}} \alpha_i K(\mathbf{x}_i, \mathbf{x}) + b$$

- The optimization technique is the same.

Support Vector Machine: Algorithm

- 1. Choose a kernel function
- 2. Choose a value for *C*
- 3. Solve the quadratic programming problem (many software packages available)
- 4. Construct the discriminant function from the support vectors

Some Issues

- Choice of kernel
  - Gaussian or polynomial kernel is default
  - if ineffective, more elaborate kernels are needed
  - domain experts can give assistance in formulating appropriate similarity measures

- Choice of kernel parameters
  - e.g. σ in Gaussian kernel
  - σ is the distance between closest points with different classifications
  - In the absence of reliable criteria, applications rely on the use of a validation set or cross-validation to set such parameters.

- Optimization criterion – Hard margin v.s. Soft margin
  - a lengthy series of experiments in which various parameters are tested

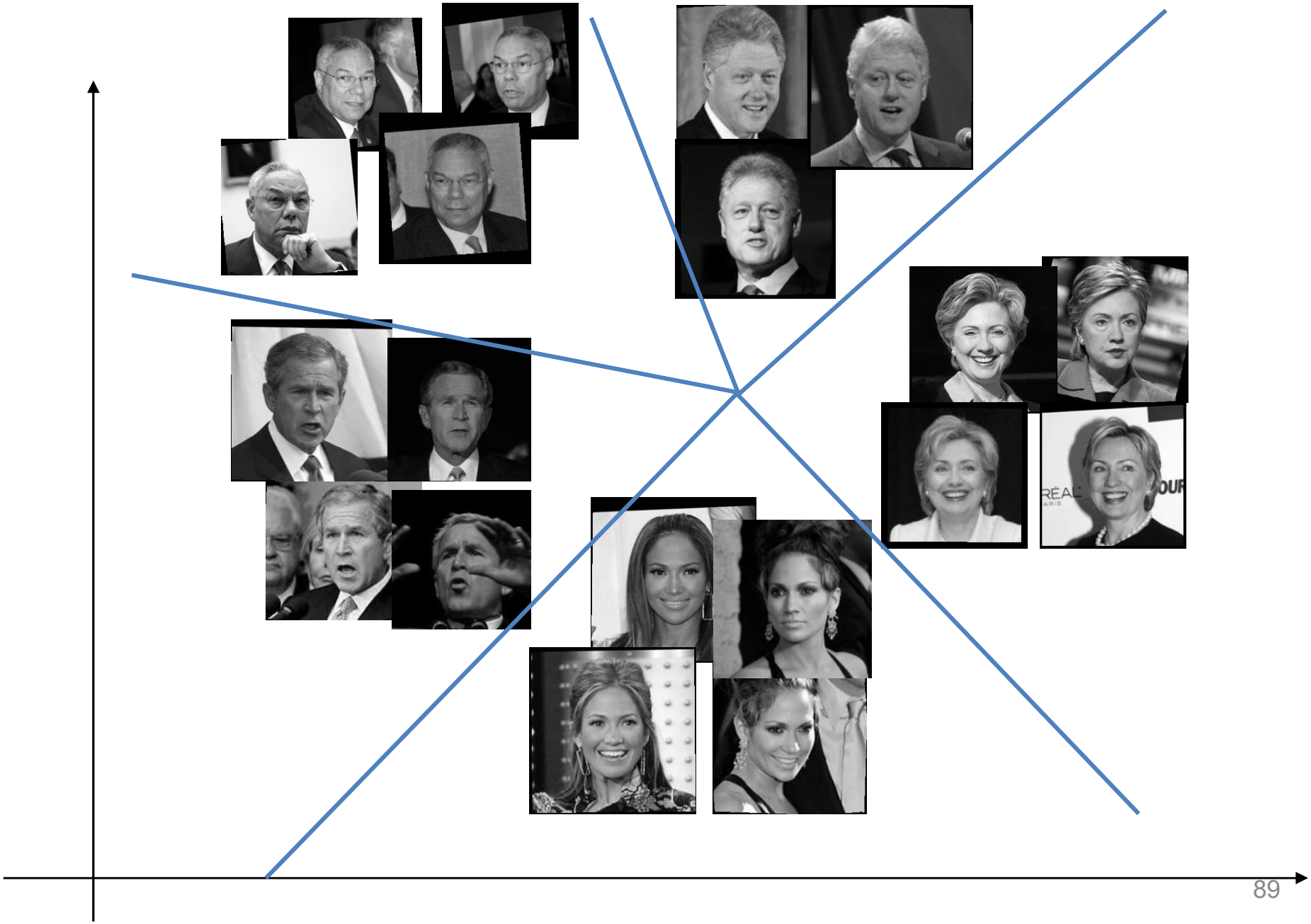
This slide is courtesy of *www.iro.umontreal.ca/~pift6080/documents/papers/**svm**_tutorial.**ppt***

Summary: Support Vector Machine

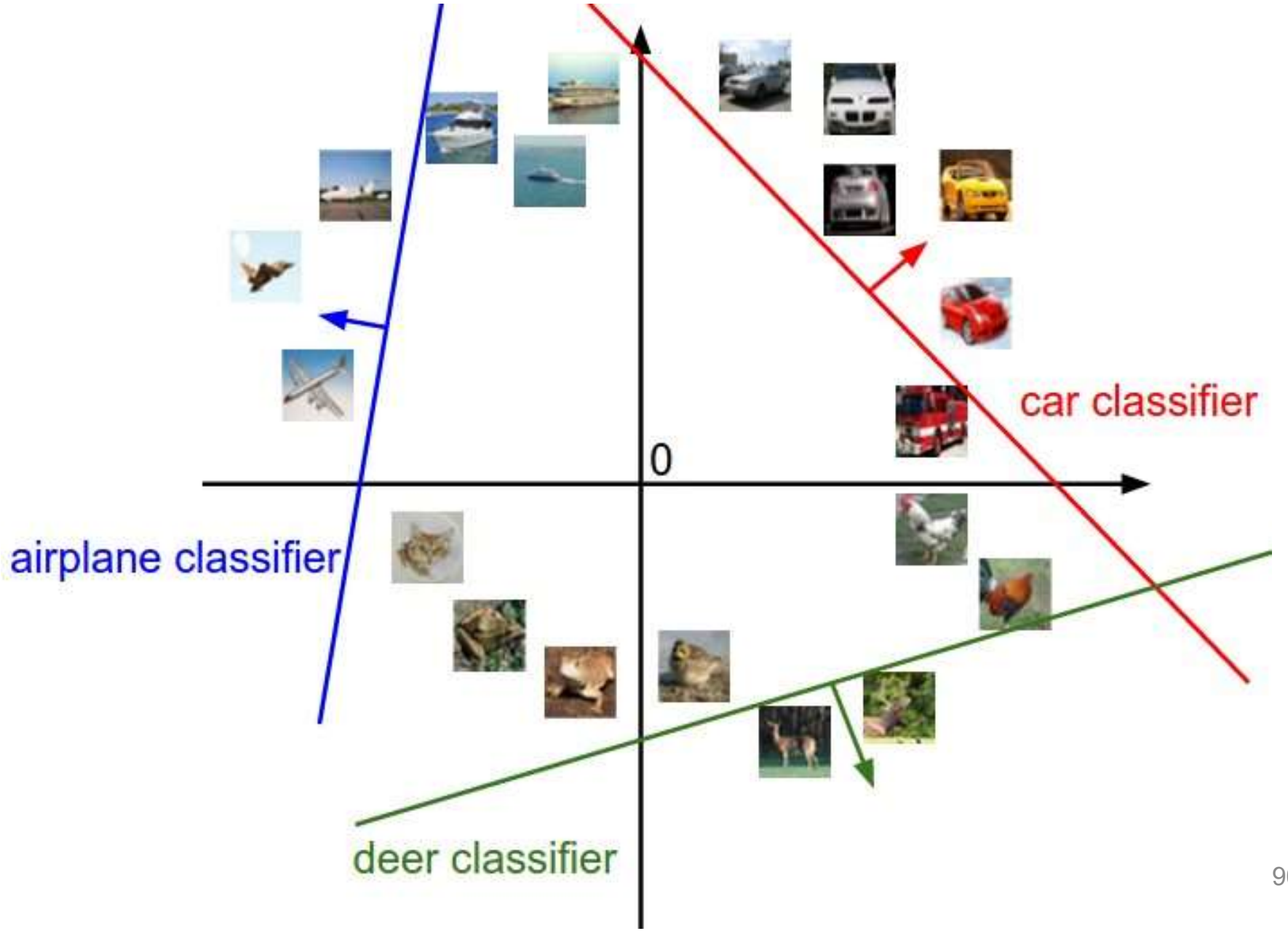
- 1. Large Margin Classifier
  - Better generalization ability & less over-fitting

- 2. The Kernel Trick
  - Map data points to higher dimensional space in order to make them linearly separable.
  - Since only dot product is used, we do not need to represent the mapping explicitly.

Multi-class generalization Pairwise



Multi-class generalization One-vs-all



Linear Classifiers: Conclusion

- Simple linear classifiers can be surprisingly effective
  - Particularly when trained to maximize a margin
    - Whereupon the "simple" arithmetic magically becomes complicated
- Kernel trick enables classification of even non-linear problems
- Most commonly used classifier, still