CS 453X: Class 11

Jacob Whitehill

More on constrained optimization

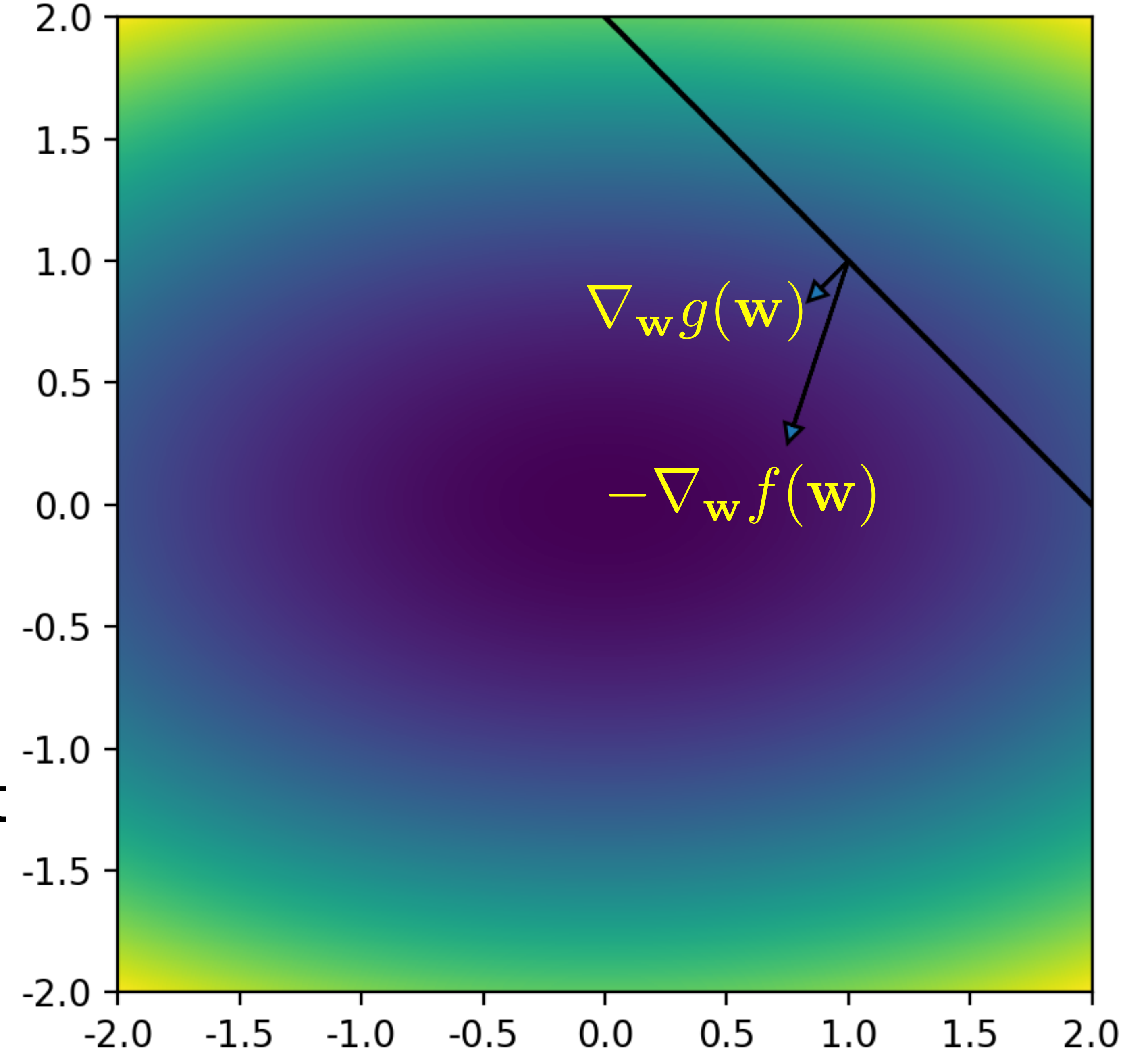
Why do Lagrange multipliers work?

- Where does the Lagrangian function come from?

- How does finding *its* critical point yield the constrained optimal solution?

- We need to restrict our "search" for optimal **w** to the **feasible set** — points where *g*(**w**) = 0.

- Imagine we are doing a gradient descent search starting from some point **w** such that *g*(**w**) = 0.

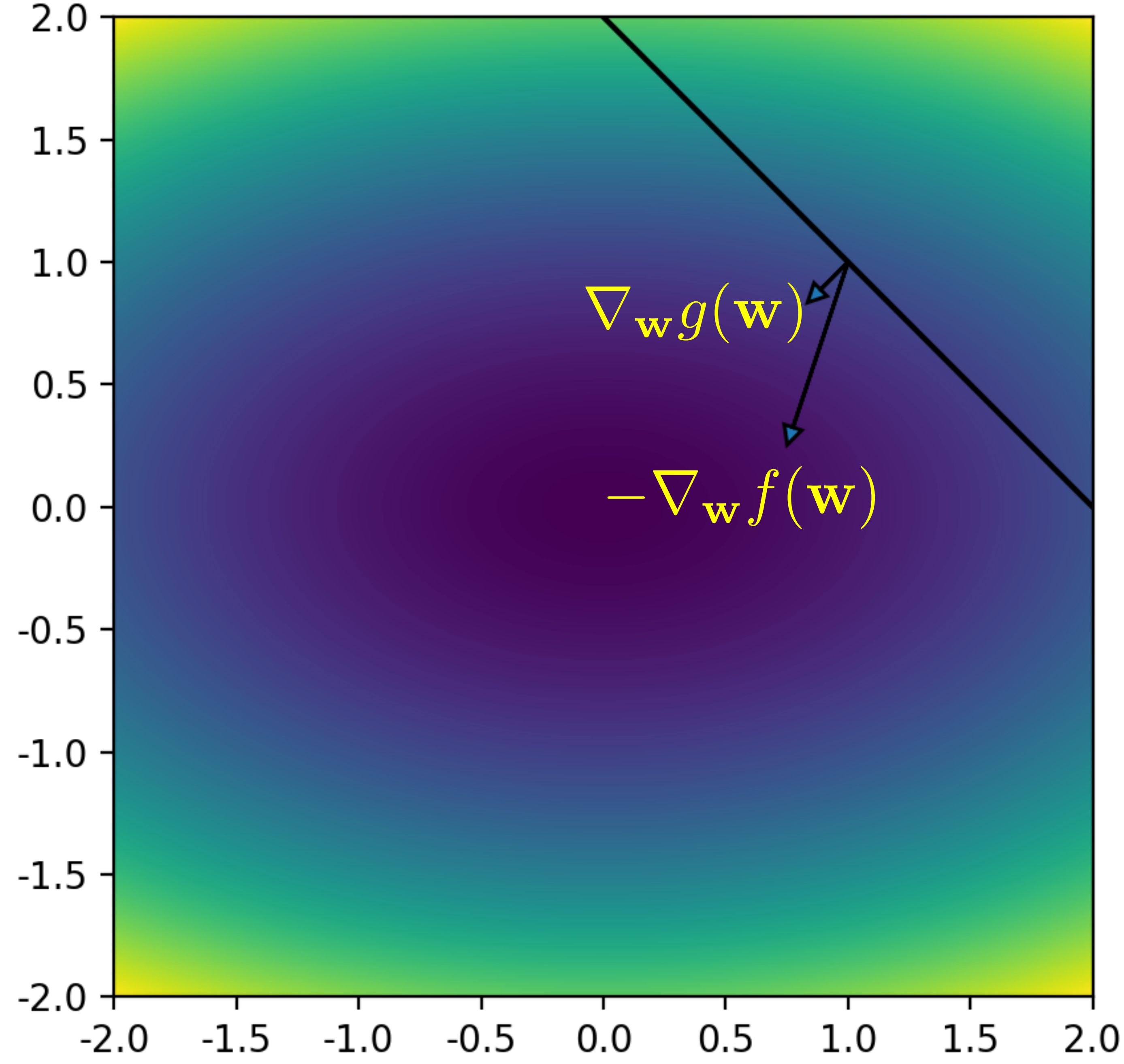


Why do Lagrange multipliers work?

- We want to “move” **w** to reduce *f*, but we need to keep **w** in the feasible set.

- Hence, we can move **w** along *only* the components of $\nabla_{\mathbf{w}} f(\mathbf{w})$ that *do not change g*(**w**).

- The directions that change *g*(**w**) are those aligned with $\nabla_{\mathbf{w}} g(\mathbf{w})$.

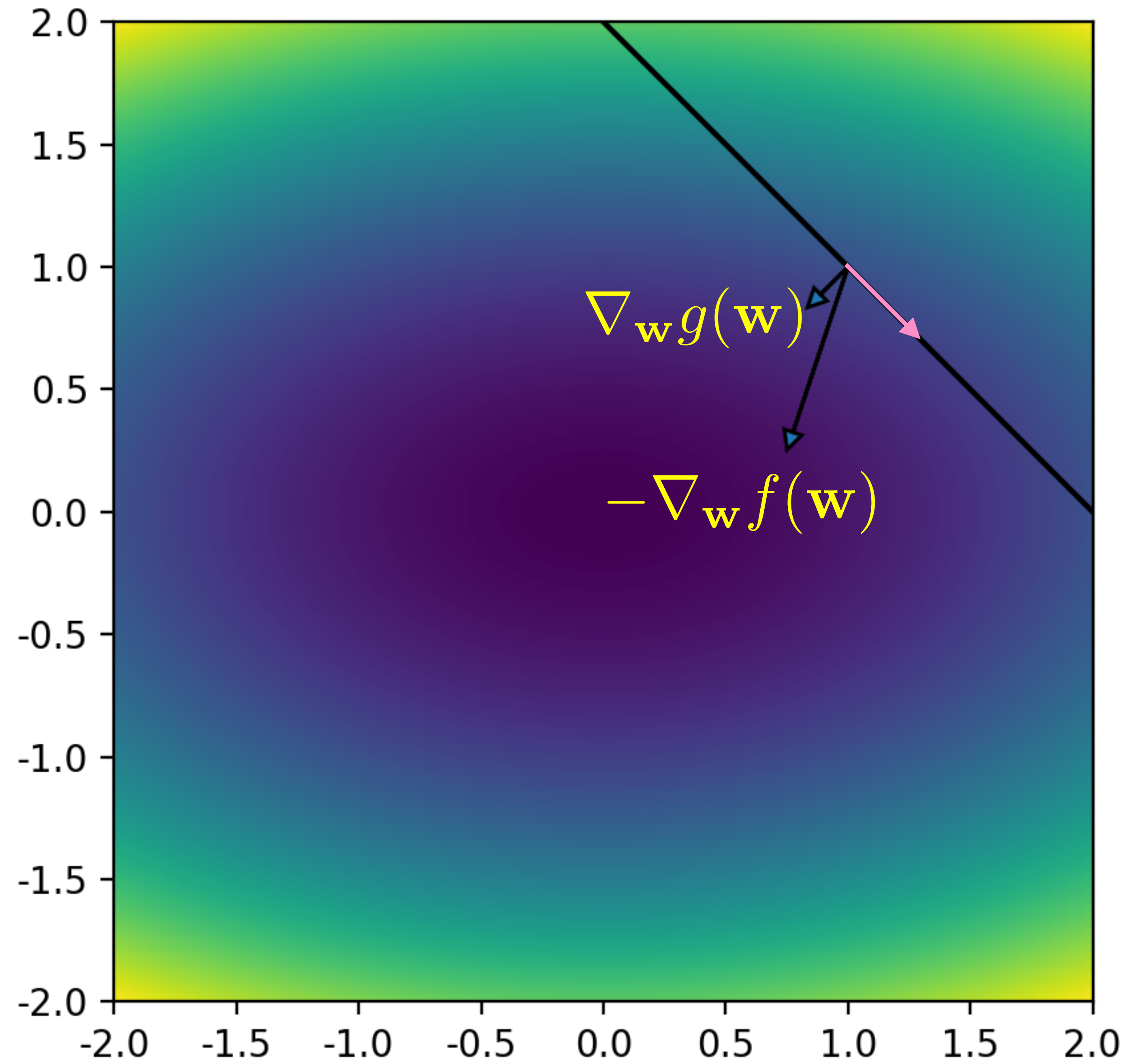


Why do Lagrange multipliers work?

- We want to "move" **w** to reduce *f*, but we need to keep **w** in the feasible set.

- Hence, we will subtract from $\nabla_{\mathbf{w}} f(\mathbf{w})$ those components that are parallel to $\nabla_{\mathbf{w}} g(\mathbf{w})$.

- In particular, we will find a "restricted gradient" vector: $\nabla_{\mathbf{w}} f(\mathbf{w}) - \alpha \nabla_{\mathbf{w}} g(\mathbf{w})$ for some value *α*.



Why do Lagrange multipliers work?

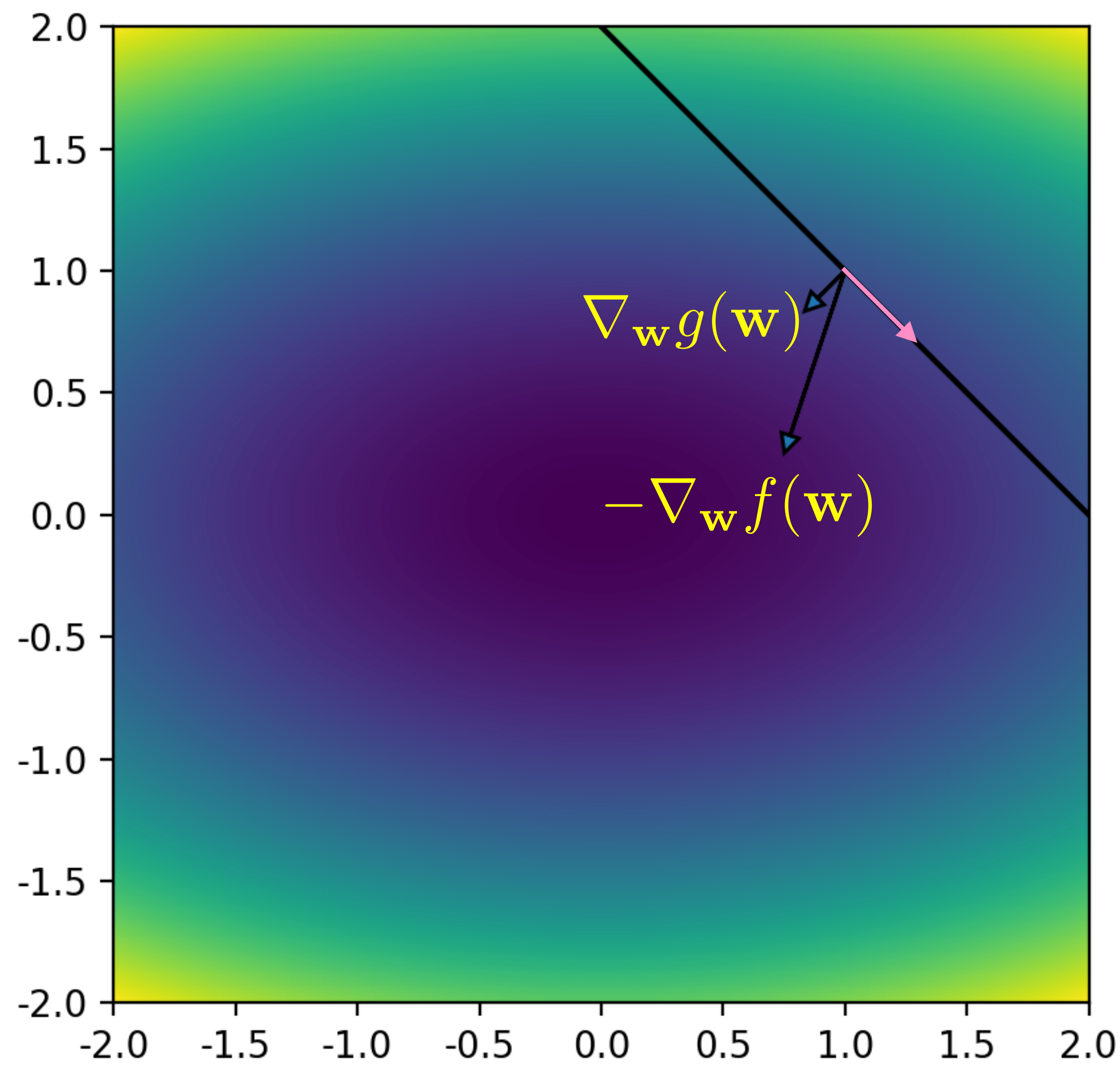
- We want to “move” **w** to reduce *f*, but we need to keep **w** in the feasible set.

- We will reach a constrained minimum when the “restricted gradient” is 0.

- We can compute the “restricted gradient” vector as the gradient of the **Lagrangian**:

$$L(\mathbf{w}, \alpha) = f(\mathbf{w}) - \alpha g(\mathbf{w})$$

- We then differentiate, set to 0, and solve.



Lagrange multipliers

- Note that either of the following Lagrangian formulations will work (since the value of α can compensate):

$$L(\mathbf{w}, \alpha) = f(\mathbf{w}) - \alpha g(\mathbf{w})$$
$$L(\mathbf{w}, \alpha) = f(\mathbf{w}) + \alpha g(\mathbf{w})$$

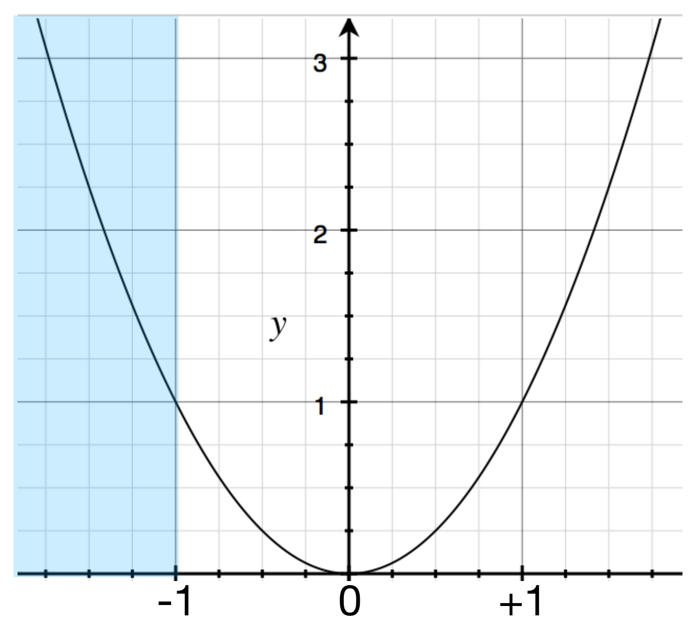
- However, with SVMs, the convention is:

$$L(\mathbf{w}, \alpha) = f(\mathbf{w}) - \alpha g(\mathbf{w})$$

Karush-Kuhn-Tucker (KKT) conditions

- A generalization of Lagrange multipliers to handle both equality and inequality constraints are the Karush-Kuhn-Tucker (KKT) conditions.

- Suppose we wish to minimize f subject to $g(x) \leq 0$:

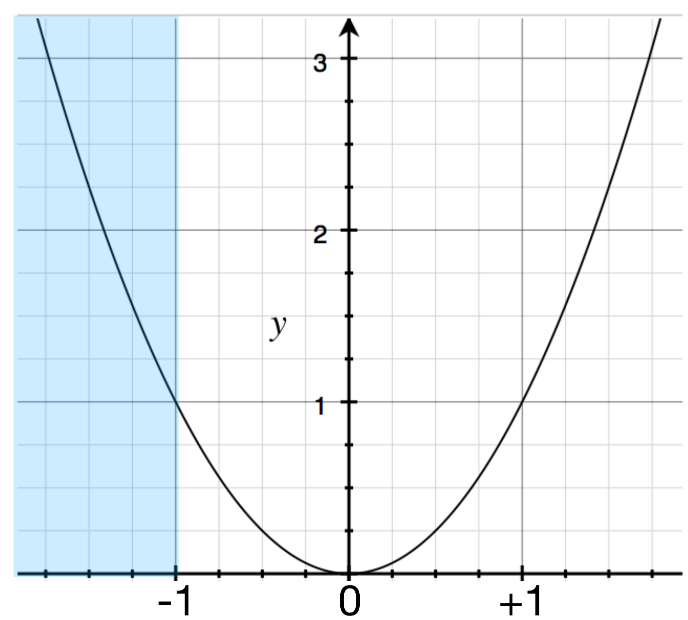


$g(x)=$?

Karush-Kuhn-Tucker (KKT) conditions

- A generalization of Lagrange multipliers to handle both equality and inequality constraints are the Karush-Kuhn-Tucker (KKT) conditions.

- Suppose we wish to minimize f subject to $g(x) \leq 0$:

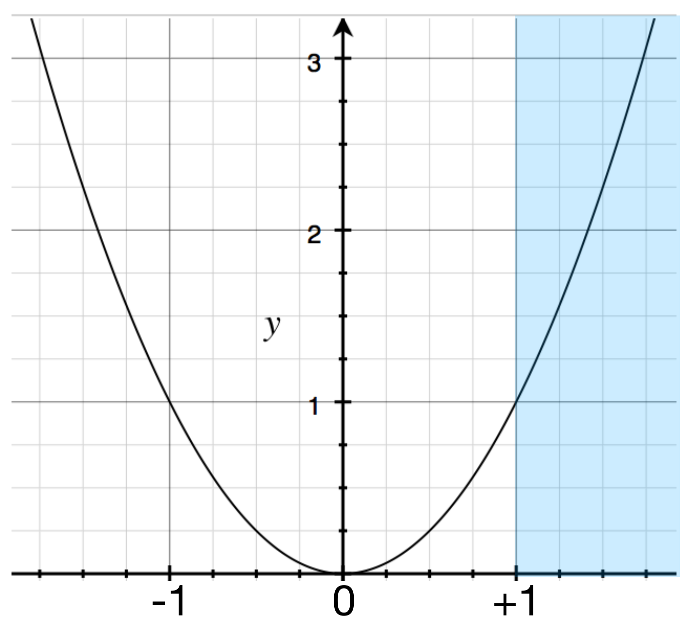


$g(x)=x+1$

Karush-Kuhn-Tucker (KKT) conditions

- A generalization of Lagrange multipliers to handle both equality and inequality constraints are the Karush-Kuhn-Tucker (KKT) conditions.

- Suppose we wish to minimize f subject to $g(x) \leq 0$:

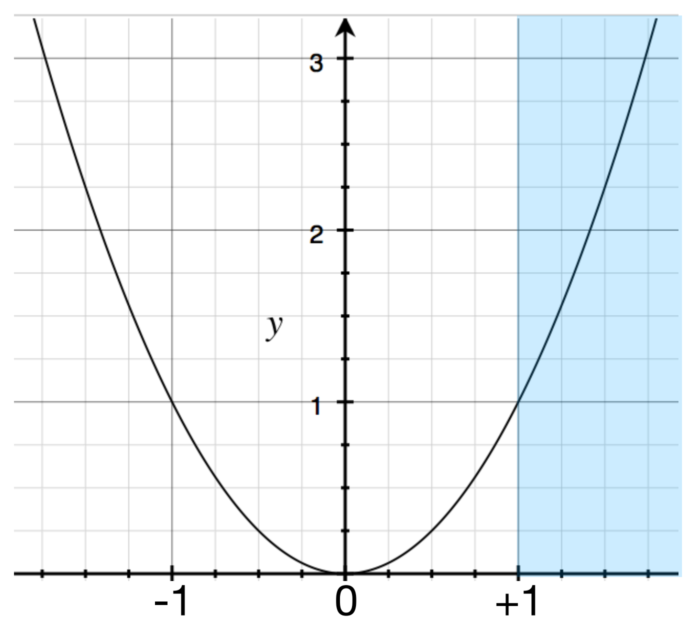


$g(x)=$?

Karush-Kuhn-Tucker (KKT) conditions

- A generalization of Lagrange multipliers to handle both equality and inequality constraints are the Karush-Kuhn-Tucker (KKT) conditions.

- Suppose we wish to minimize *f* subject to $g(x) \leq 0$:

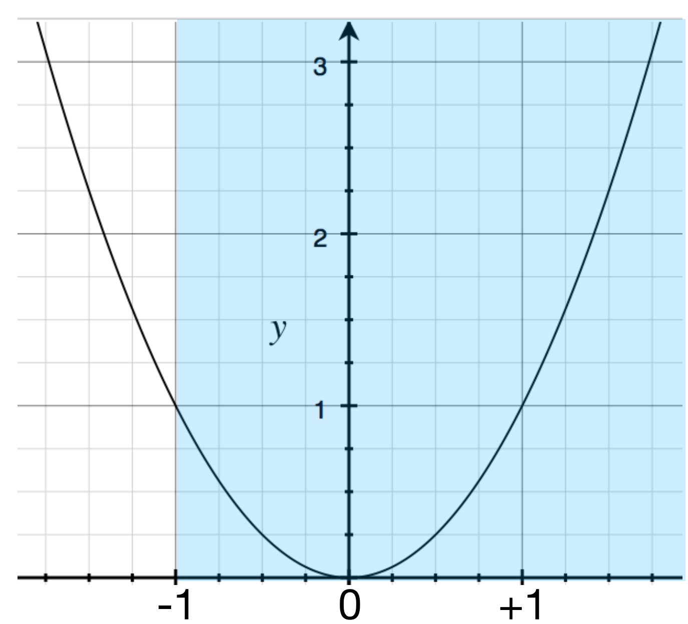


$g(x)=1-x$

Karush-Kuhn-Tucker (KKT) conditions

- A generalization of Lagrange multipliers to handle both equality and inequality constraints are the Karush-Kuhn-Tucker (KKT) conditions.

- Suppose we wish to minimize *f* subject to $g(x) \leq 0$:

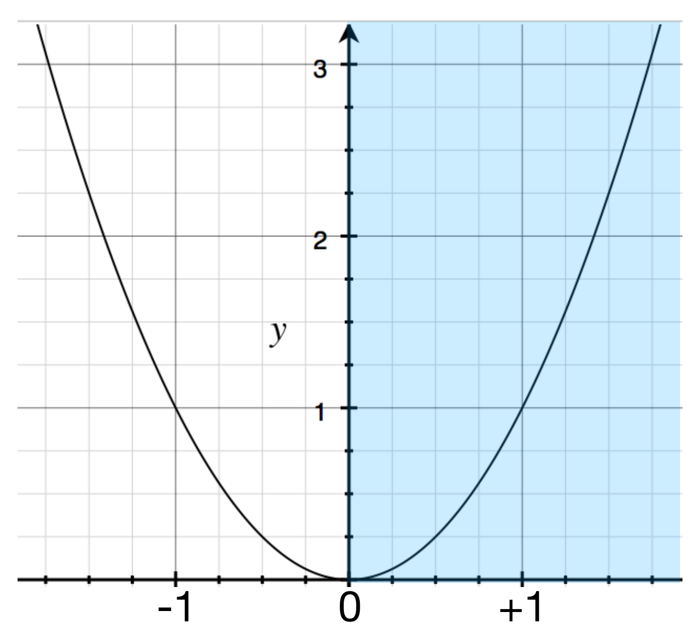


$g(x) = -1 - x$

Karush-Kuhn-Tucker (KKT) conditions

- A generalization of Lagrange multipliers to handle both equality and inequality constraints are the Karush-Kuhn-Tucker (KKT) conditions.

- Suppose we wish to minimize f subject to $g(x) \leq 0$:



$g(x) = -x$

Karush-Kuhn-Tucker (KKT) conditions

- Similarly as with Lagrange multipliers, with KKT conditions we also use a set of “multipliers” α (one for each constraint), sometimes known as **dual variables.**

$$L(\mathbf{w}, \alpha) = f(\mathbf{w}) - \sum_{i=1}^{n} \alpha_i g_i(\mathbf{w})$$

Karush-Kuhn-Tucker (KKT) conditions

- Similarly as with Lagrange multipliers, with KKT conditions we also use a set of "multipliers" α (one for each constraint), sometimes known as **dual variables**.

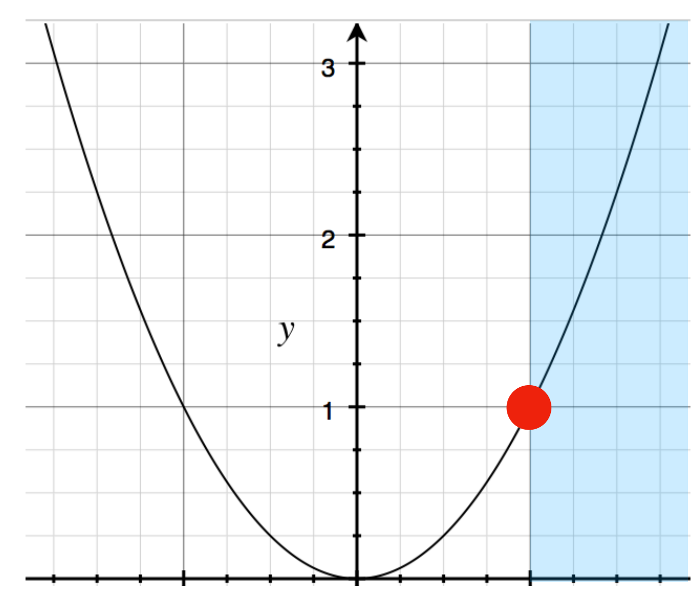
$$L(\mathbf{w}, \alpha) = f(\mathbf{w}) - \sum_{i=1}^{n} \alpha_i g_i(\mathbf{w})$$

- Key points:

1. With *inequality* constraints, we require that each $\alpha_i \geq 0$.

2. At optimal solution:

   - $\alpha_i > 0$ if the constraint is **active**.



$g(x)=x$

Karush-Kuhn-Tucker (KKT) conditions

- Similarly as with Lagrange multipliers, with KKT conditions we also use a set of "multipliers" α (one for each constraint), sometimes known as **dual variables.**

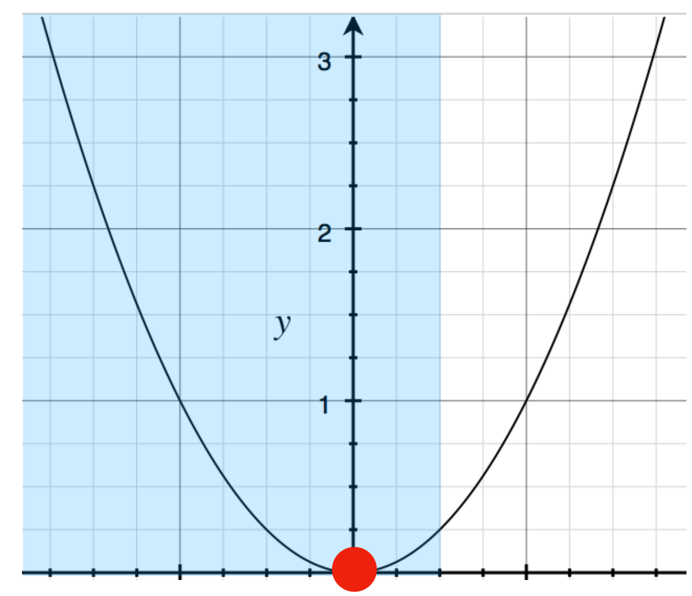
$$L(\mathbf{w}, \alpha) = f(\mathbf{w}) - \sum_{i=1}^{n} \alpha_i g_i(\mathbf{w})$$

- Key points:

1. With *inequality* constraints, we require that each $\alpha_i \geq 0$.

2. At optimal solution:

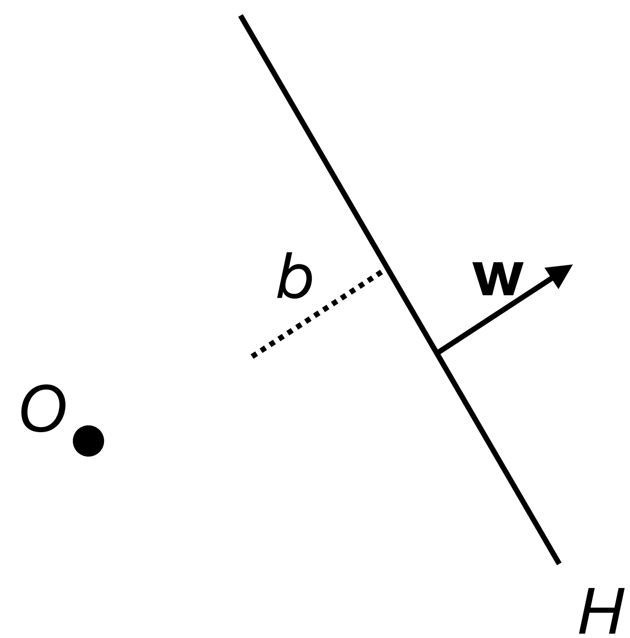
    - $\alpha_i > 0$ if the constraint is **active**.
    - $\alpha_i = 0$ if the constraint is **inactive**.



$g(x)=x-1/2$

Hyperplanes

Defining a hyperplane



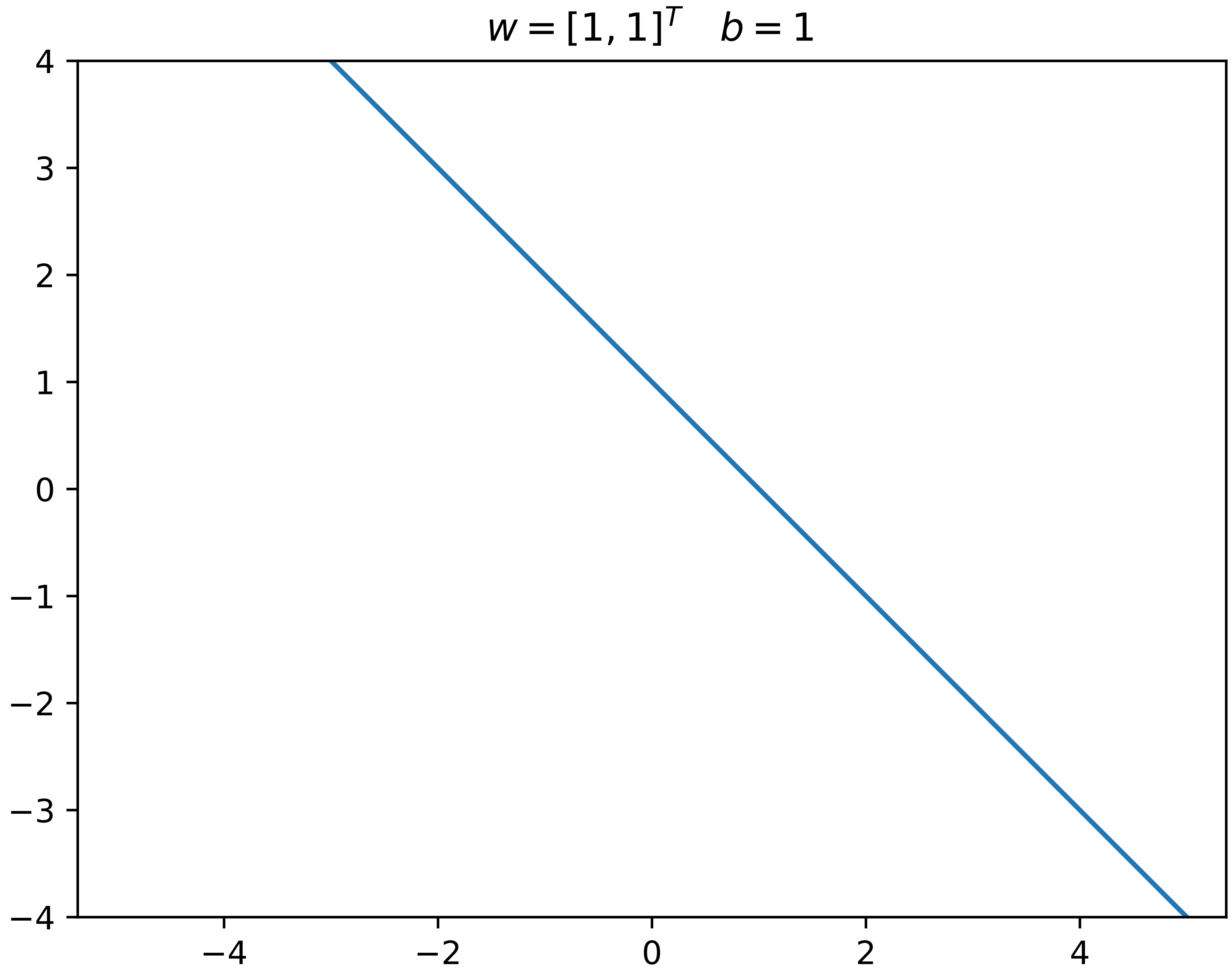
- A **hyperplane** is defined by a normal vector **w** ($\perp$ to *H*) and a bias *b* that is proportional to the distance to the origin.

- The points on hyperplane *H* are those values of **x** that satisfy:

$$\mathbf{x}^\top \mathbf{w} + b = 0$$

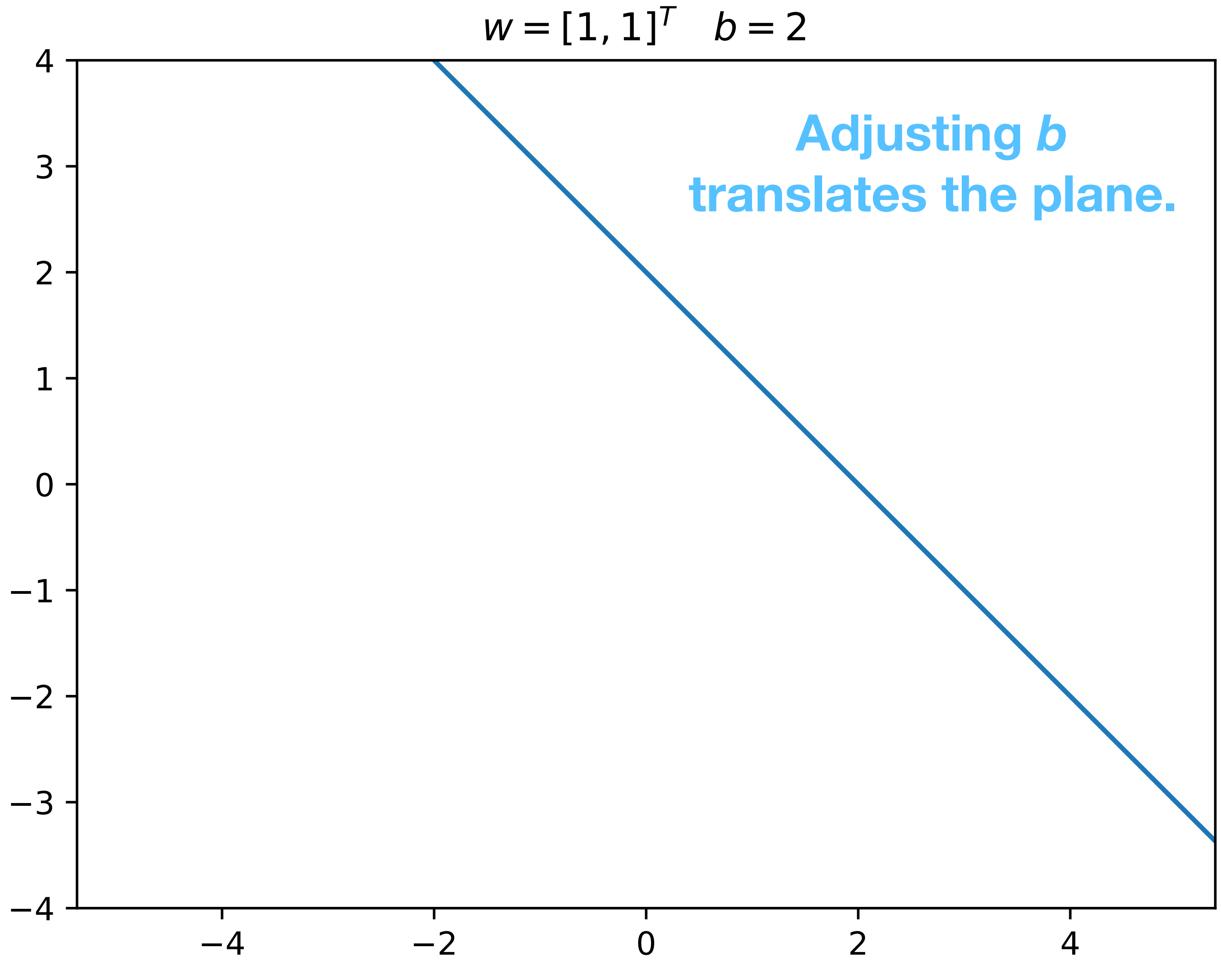
Hyperplane examples

$$H = \{\mathbf{x} \in \mathbb{R}^m : \mathbf{x}^\top \mathbf{w} + b = 0\}$$



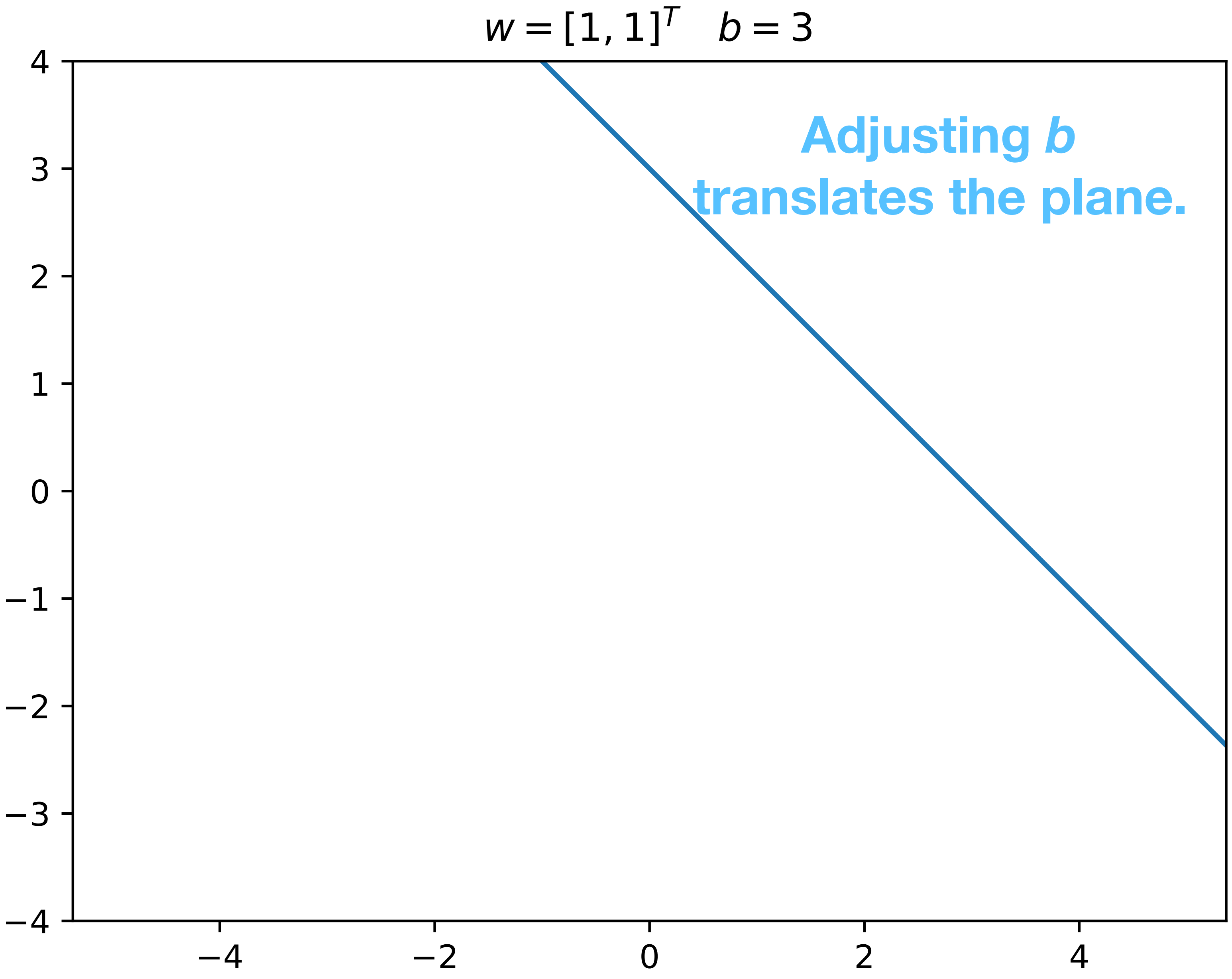
Hyperplane examples

$$H = \{\mathbf{x} \in \mathbb{R}^m : \mathbf{x}^\top \mathbf{w} + b = 0\}$$



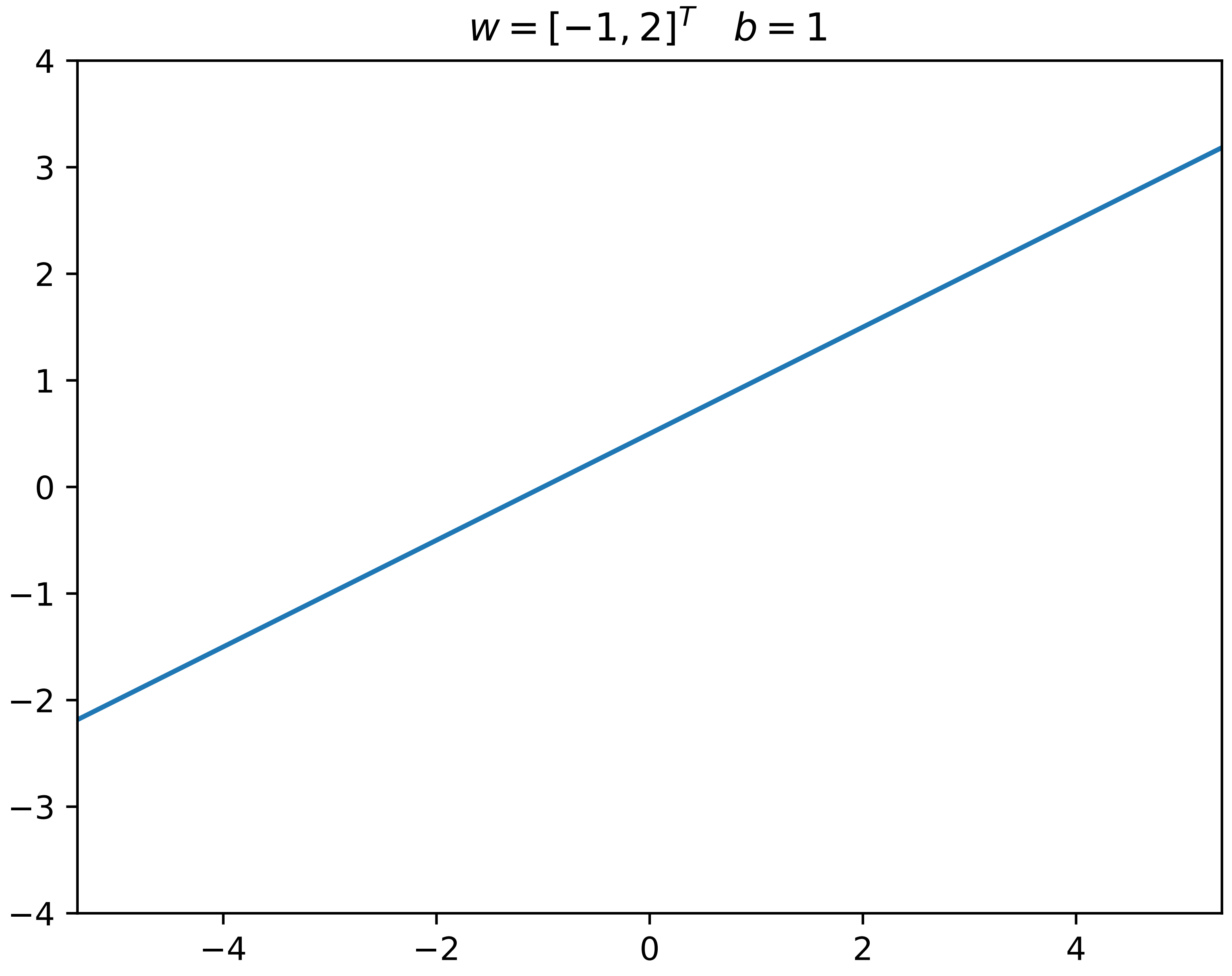
Hyperplane examples

$$H = \{\mathbf{x} \in \mathbb{R}^m : \mathbf{x}^\top \mathbf{w} + b = 0\}$$



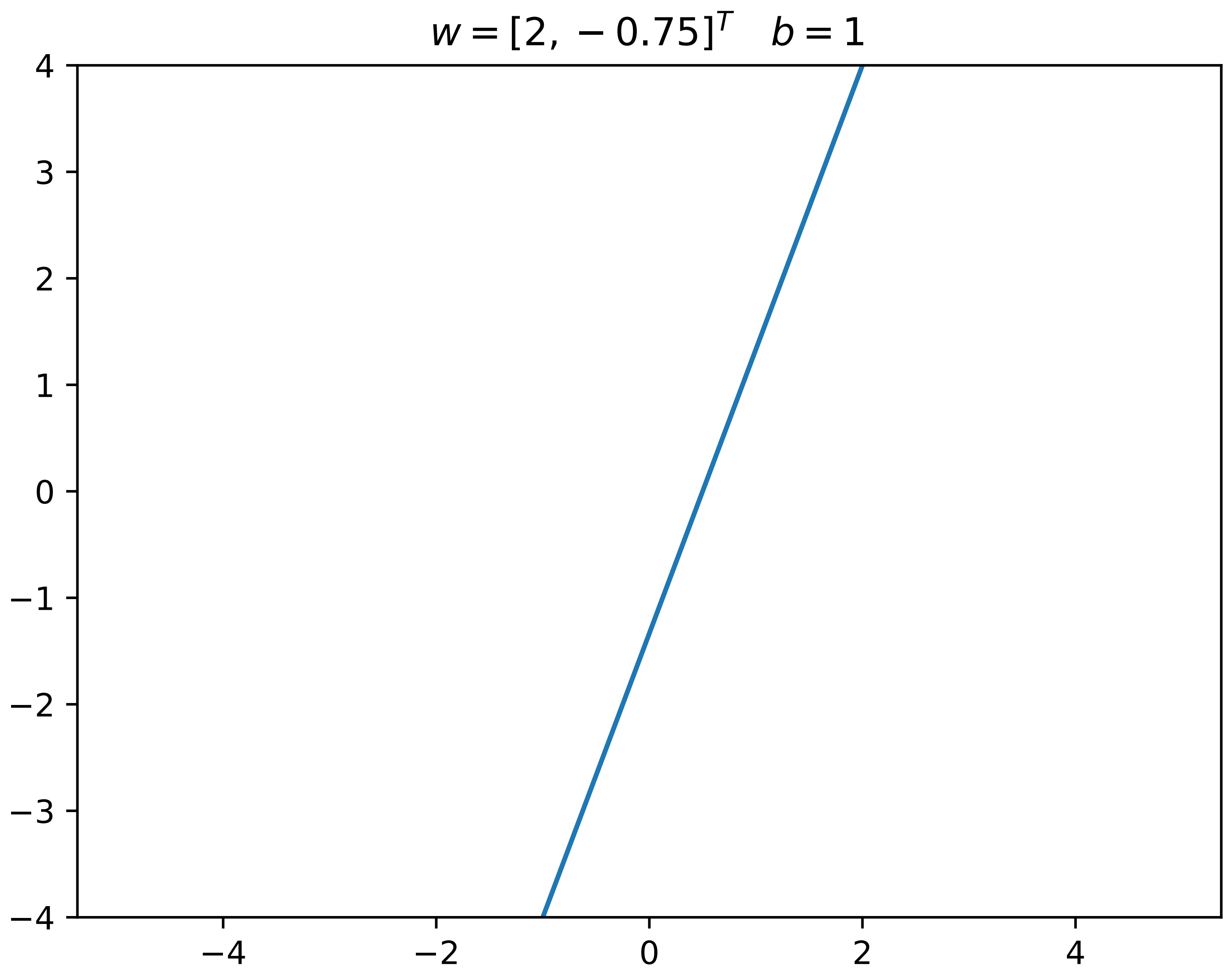
Hyperplane examples

$$H = \{\mathbf{x} \in \mathbb{R}^m : \mathbf{x}^\top \mathbf{w} + b = 0\}$$



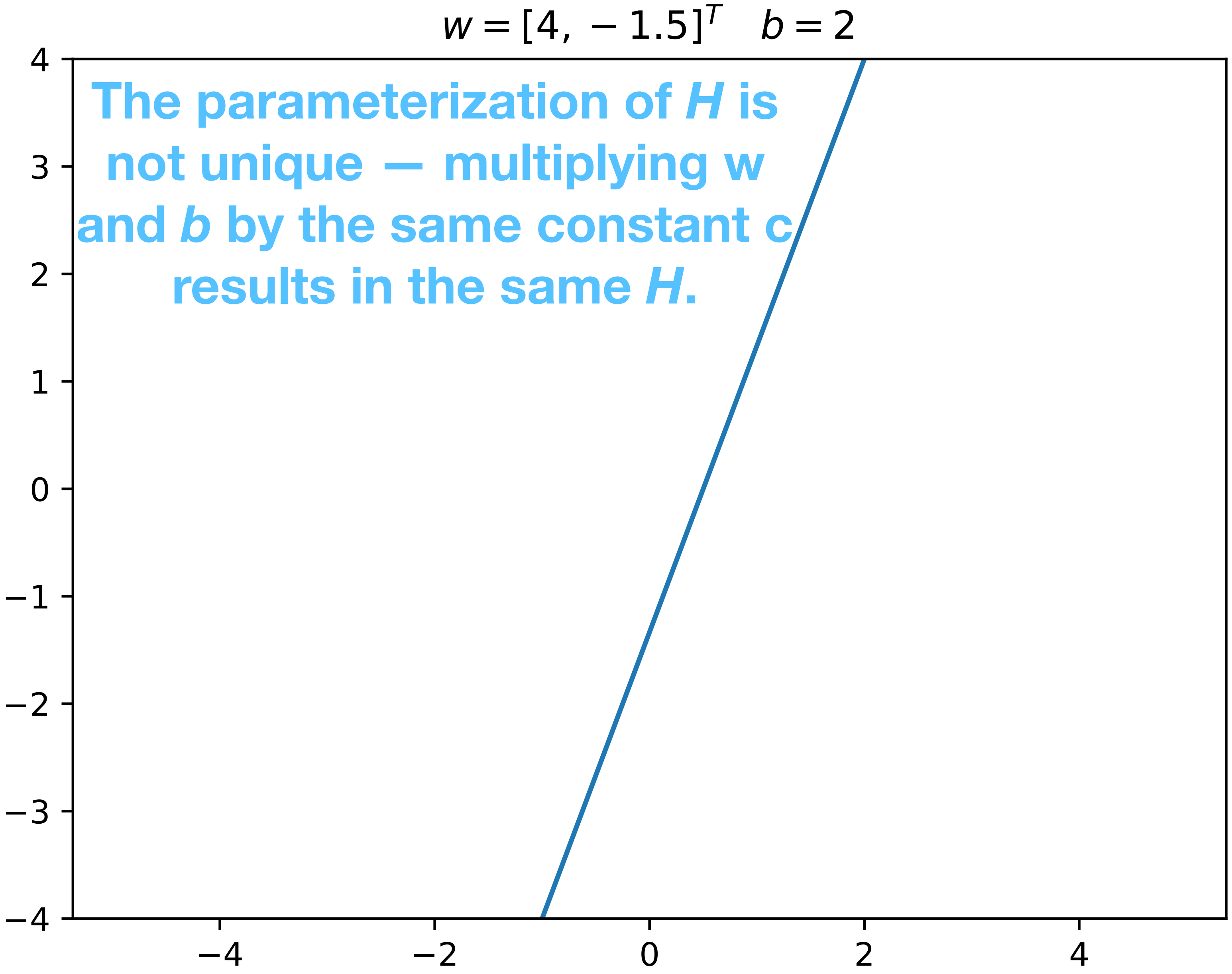
Hyperplane examples

$$H = \{\mathbf{x} \in \mathbb{R}^m : \mathbf{x}^\top \mathbf{w} + b = 0\}$$

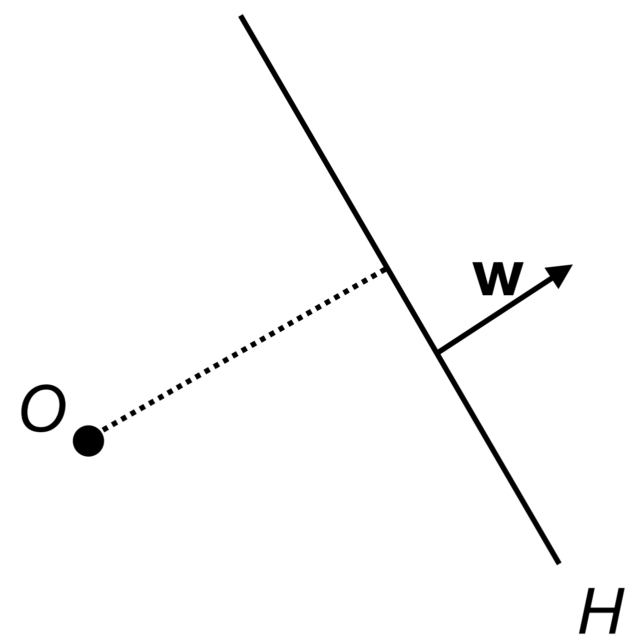


Hyperplane examples

$$H = \{\mathbf{x} \in \mathbb{R}^m : \mathbf{x}^\top \mathbf{w} + b = 0\}$$

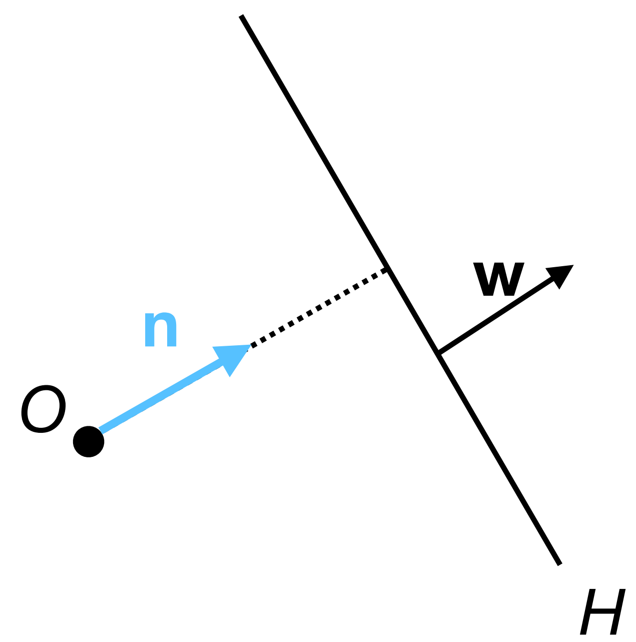


Distance from *O* to *H*



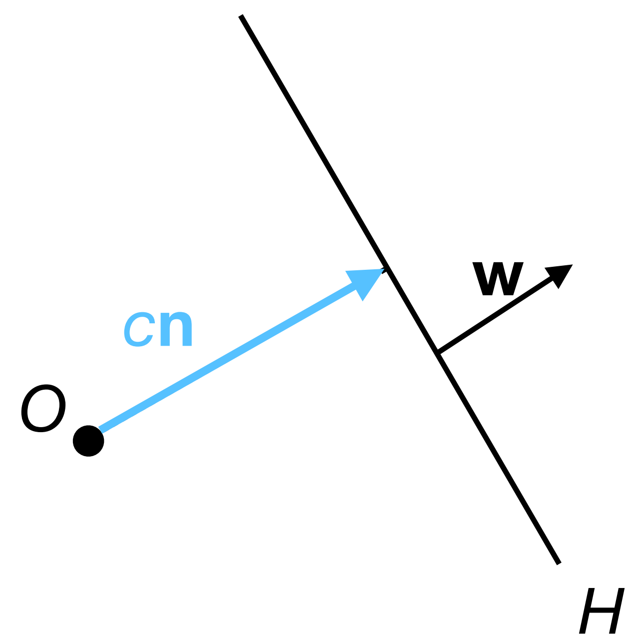
- To find the shortest (perpendicular) distance between the origin *O* and the hyperplane *H*:

Distance from O to H



- To find the shortest (perpendicular) distance between the origin O and the hyperplane H:
  - Define a *unit* vector **n** with same direction as **w**: $\mathbf{n} = \frac{\mathbf{w}}{|\mathbf{w}|}$

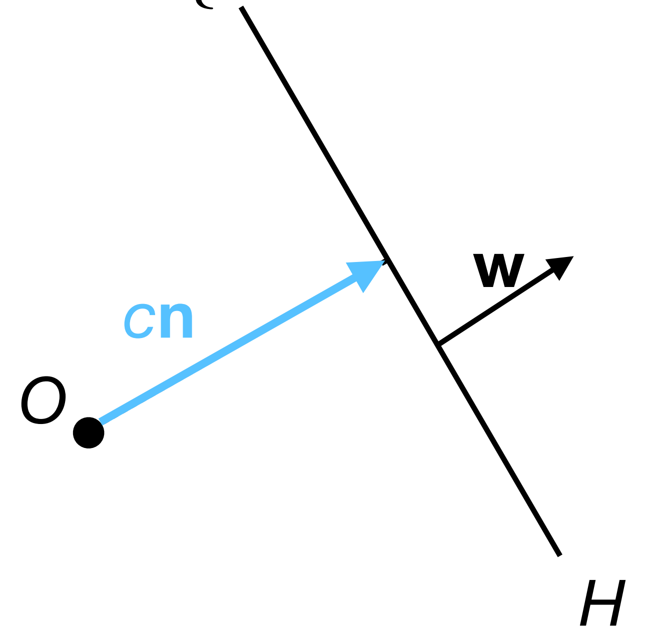
Distance from *O* to *H*



- To find the shortest (perpendicular) distance between the origin *O* and the hyperplane *H*:
  - Define a *unit* vector **n** with same direction as **w**: $\mathbf{n} = \frac{\mathbf{w}}{|\mathbf{w}|}$
  - The shortest line from *O* to *H* ends at *c***n**, for some distance *c*.

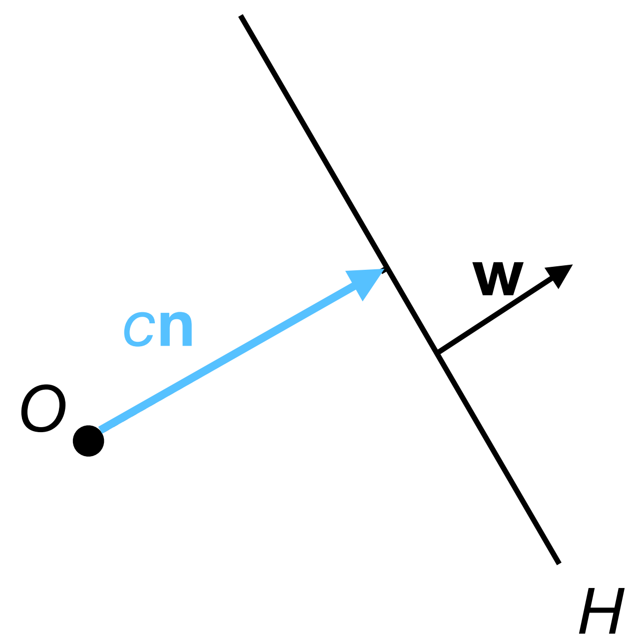
Distance from *O* to *H*

$$H = \{\mathbf{x} \in \mathbb{R}^m : \mathbf{x}^\top \mathbf{w} + b = 0\}$$



- Since *c***n** is within *H*, we have: $c\mathbf{n}^\top \mathbf{w} + b \;=\; 0$

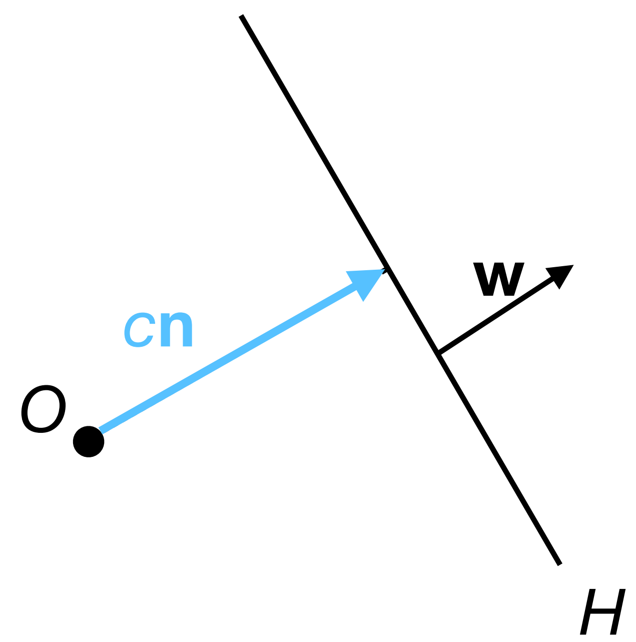
Distance from *O* to *H*



- Since *c***n** is within *H*, we have: $$c\mathbf{n}^\top \mathbf{w} + b \;=\; 0$$

- We can then solve for *c* (distance from *O* to *H*):

Distance from O to H

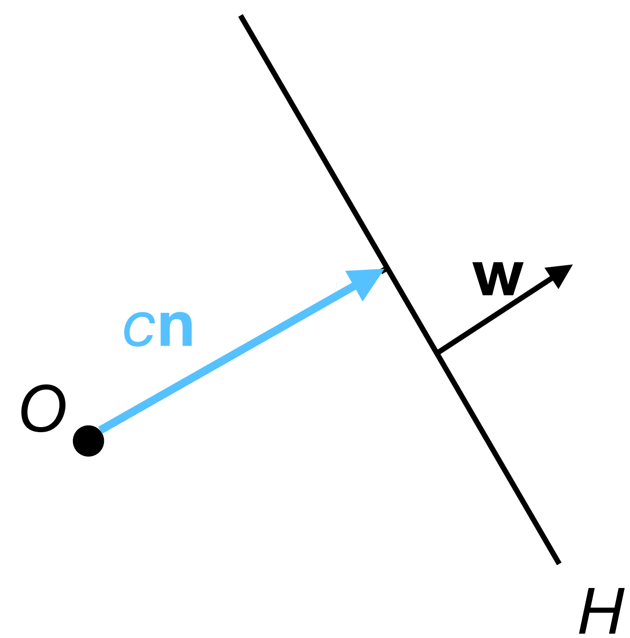


- Since $c\mathbf{n}$ is within H, we have:

- We can then solve for c (distance from O to H):

$$
\begin{aligned}
c\mathbf{n}^\top \mathbf{w} + b &= 0 \\
c\left(\frac{\mathbf{w}}{|\mathbf{w}|}\right)^\top \mathbf{w} &= -b
\end{aligned}
$$

Distance from O to H

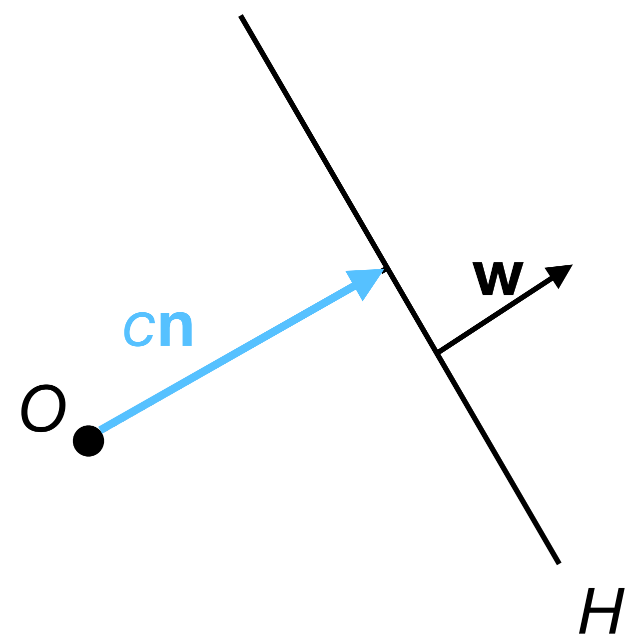


- Since $c\mathbf{n}$ is within H, we have:

- We can then solve for c (distance from O to H):

$$
\begin{aligned}
c\mathbf{n}^\top \mathbf{w} + b &= 0 \\
c\left(\frac{\mathbf{w}}{|\mathbf{w}|}\right)^\top \mathbf{w} &= -b \\
\frac{c}{|\mathbf{w}|}\mathbf{w}^\top \mathbf{w} &= -b
\end{aligned}
$$

Distance from O to H

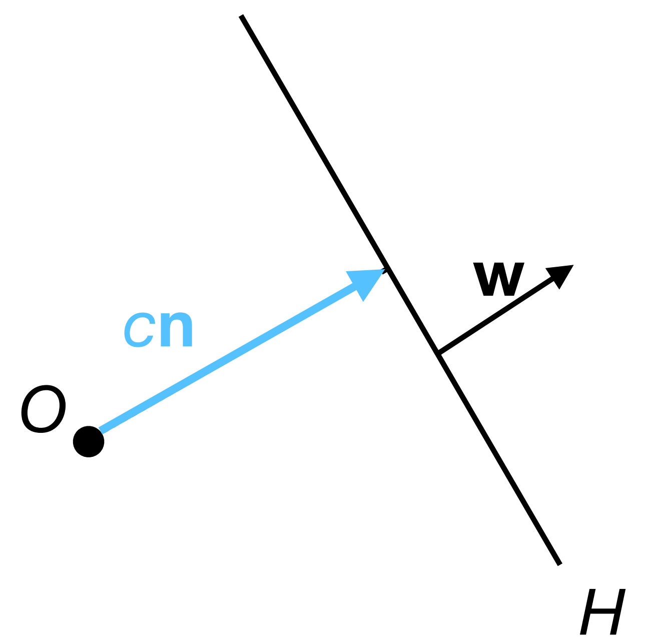


- Since $c\mathbf{n}$ is within H, we have:

- We can then solve for c (distance from O to H):

$$
\begin{aligned}
c\mathbf{n}^\top \mathbf{w} + b &= 0 \\
c\left(\frac{\mathbf{w}}{|\mathbf{w}|}\right)^\top \mathbf{w} &= -b \\
\frac{c}{|\mathbf{w}|}\mathbf{w}^\top \mathbf{w} &= -b \\
\frac{c}{|\mathbf{w}|}|\mathbf{w}|^2 &= -b
\end{aligned}
$$

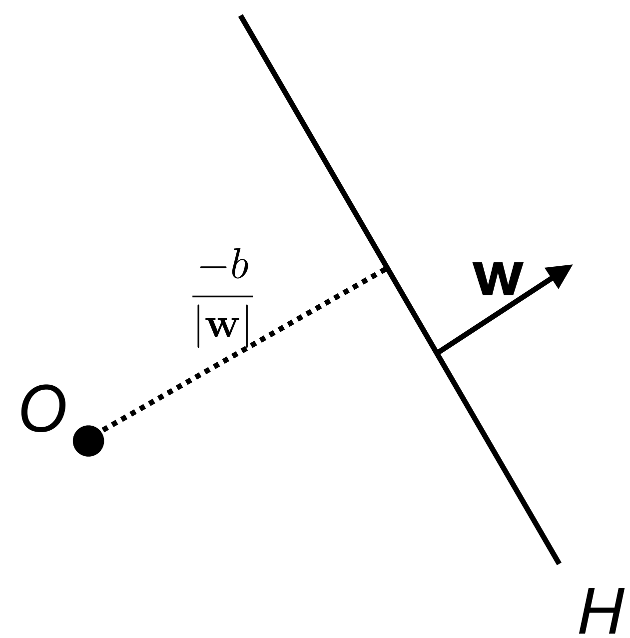
Distance from O to H



- Since $c\mathbf{n}$ is within H, we have:
- We can then solve for c (distance from O to H):

$$
\begin{aligned}
c\mathbf{n}^\top \mathbf{w} + b &= 0 \\
c\left(\frac{\mathbf{w}}{|\mathbf{w}|}\right)^\top \mathbf{w} &= -b \\
\frac{c}{|\mathbf{w}|}\mathbf{w}^\top \mathbf{w} &= -b \\
\frac{c}{|\mathbf{w}|}|\mathbf{w}|^2 &= -b \\
c|\mathbf{w}| &= -b \\
c &= \frac{-b}{|\mathbf{w}|}
\end{aligned}
$$

Distance from *O* to *H*



- Therefore, the shortest distance between the origin *O* and the hyperplane *H* is:

$$\frac{-b}{|\mathbf{w}|}$$

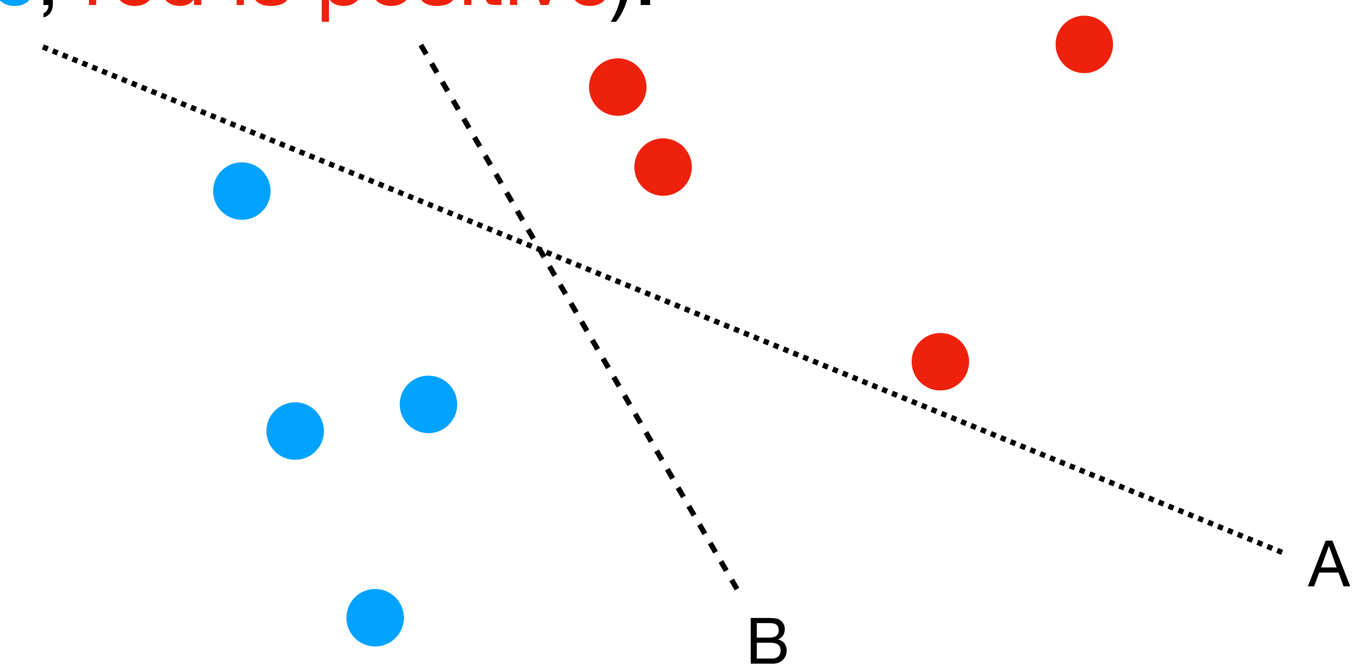
Support vector machines

Support vector machines

- **Support vector machines (SVMs)** are a ML model for binary classification.
- SVMs are optimized using **constrained optimization** rather than unconstrained optimization (e.g., for logistic regression).

Support vector machines

- Suppose we have the following set of training data (blue is negative, red is positive):

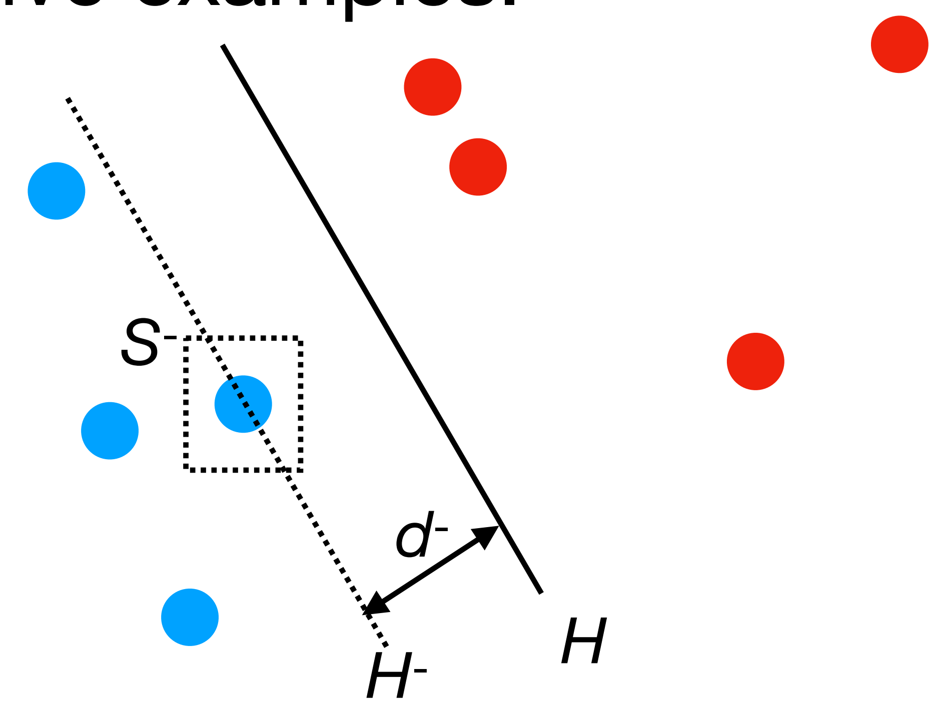


- Examples above the line will be classified as positive; examples below the line will be classified as negative.

- Which line (or **hyperplane** in higher dimensions) would likely perform better on *testing* data, and why?

Support vector machines

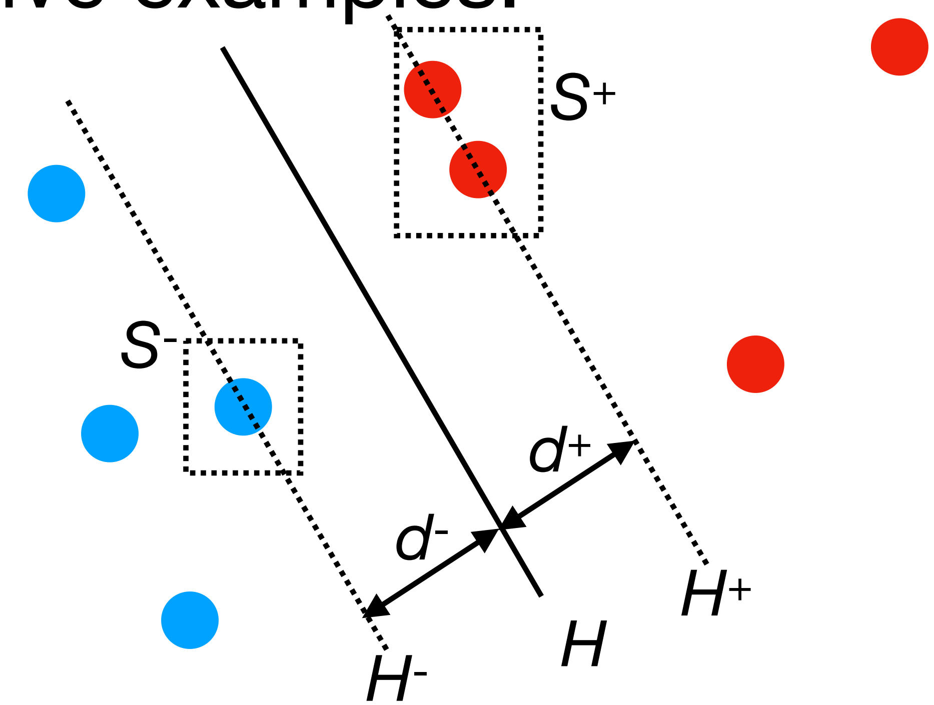
- For any hyperplane H that perfectly separates the positive from the negative examples:



- Find the subset S^- of - examples that lie closest to H.
- The points in S^- lie in a hyperplane H^- parallel to H.
- Denote the shortest distance between H^- and H as d^-.

Support vector machines

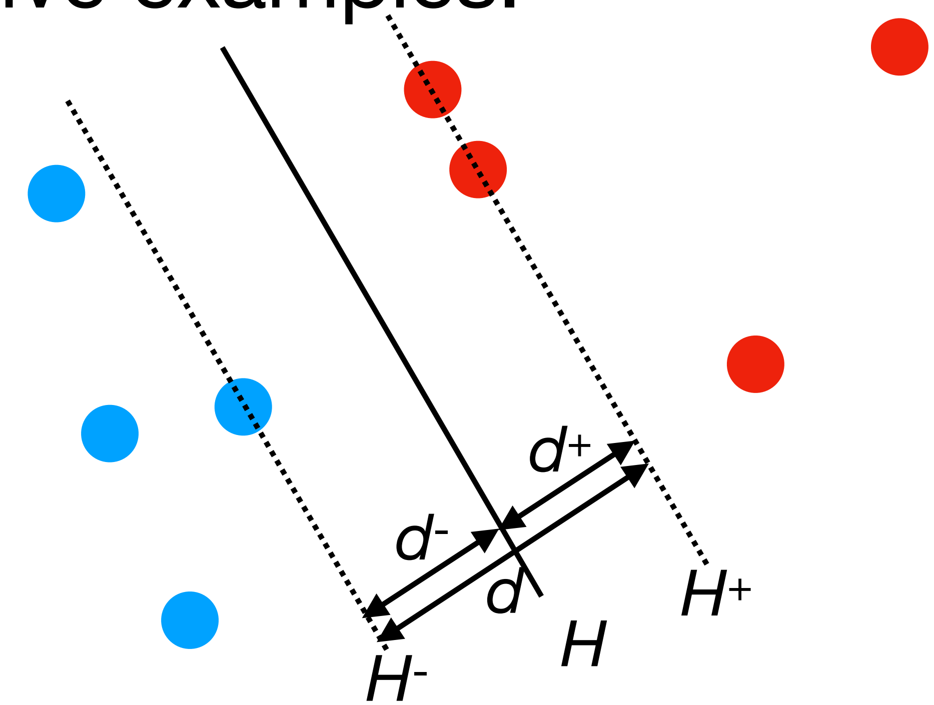
- For any hyperplane H that perfectly separates the positive from the negative examples:



- Find the subset S^+ of + examples that lie closest to H.
- The points in S^+ lie in a hyperplane H^+ parallel to H.
- Denote the shortest distance between H^+ and H as d^+.

Support vector machines

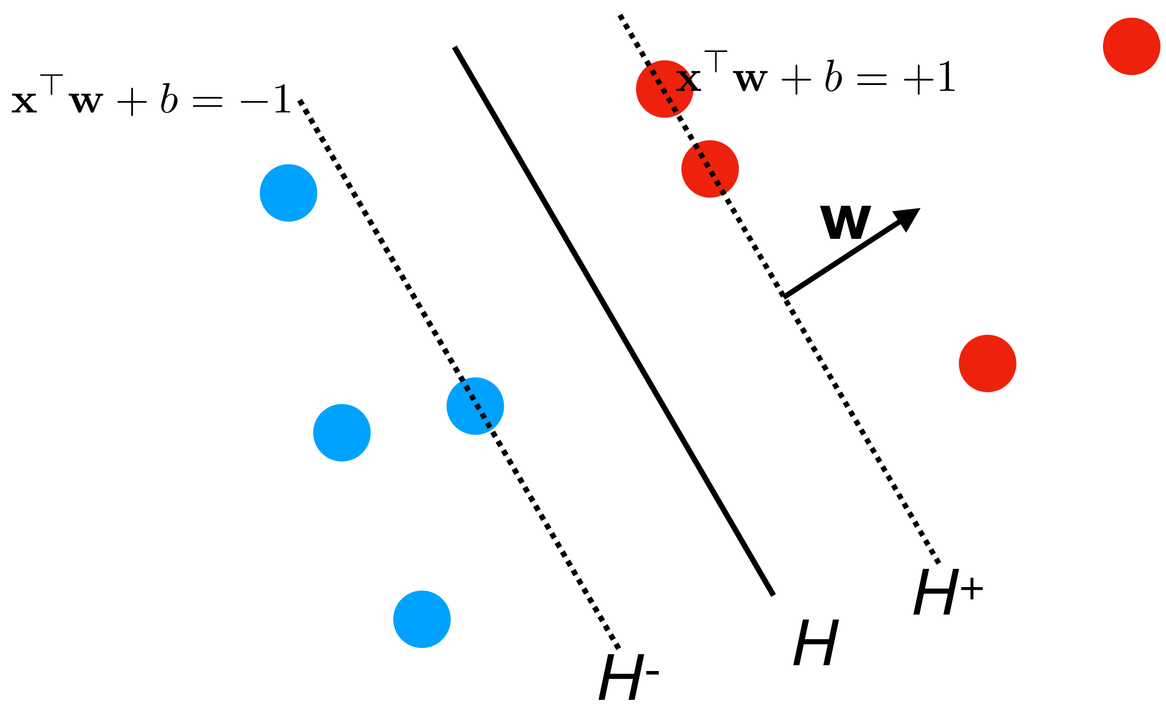
- For any hyperplane H that perfectly separates the positive from the negative examples:



- Let d denote the **margin** — the sum of d^+ and d^-.
- The optimization objective of SVMs is to find a separating hyperplane H that maximizes d.

Support vector machines

- Recall that $H \parallel H^{+} \parallel H^{-}$. Then they can share the same **w**.

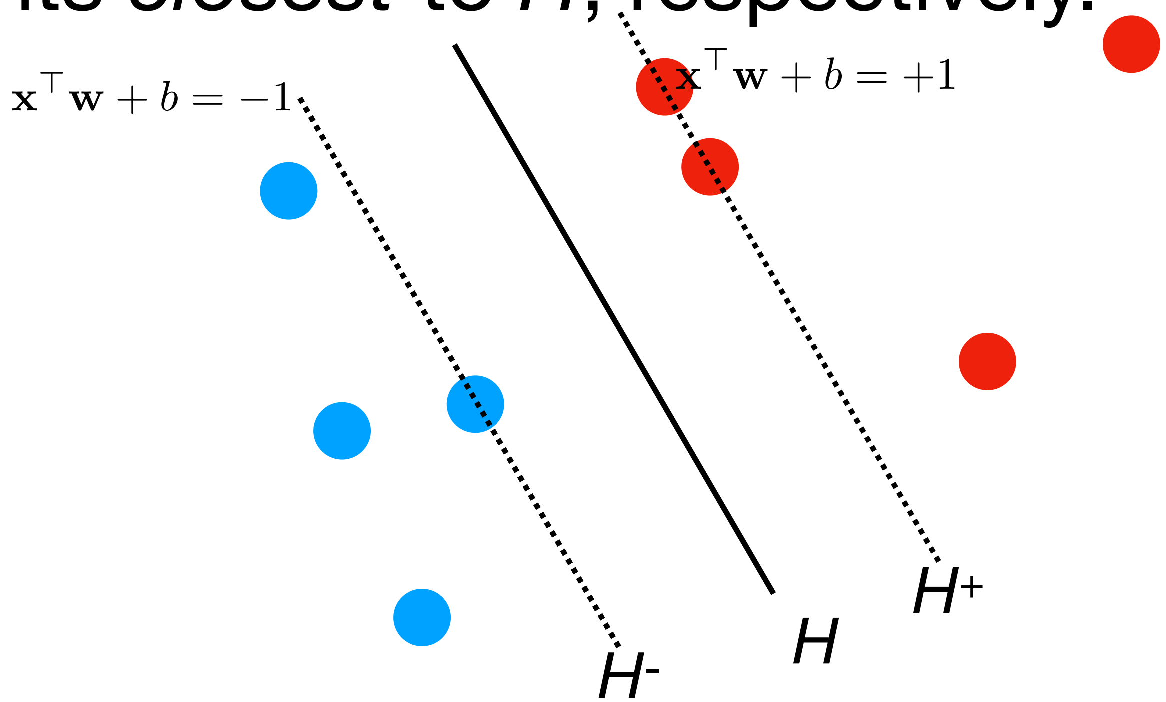


- We can thus scale **w** and *b* such that:

$$
\begin{aligned}
H^{-}: &\qquad \mathbf{x}^\top\mathbf{w} + b = -1 \\
H: &\qquad \mathbf{x}^\top\mathbf{w} + b = 0 \\
H^{+}: &\qquad \mathbf{x}^\top\mathbf{w} + b = +1
\end{aligned}
$$

Support vector machines

- H^- and H^+ intersect the negatively and positively labeled data points *closest* to H, respectively.

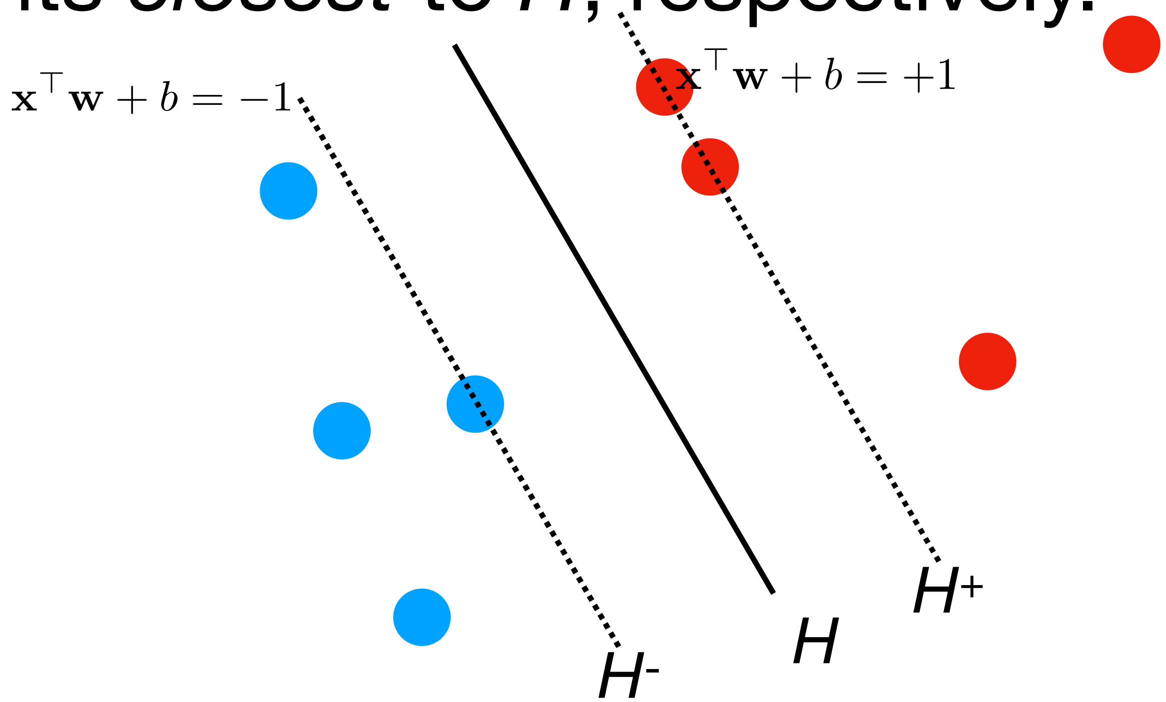


- Since all data points not in H^+ or H^- must lie even farther from H, we require that:

$$y^{(i)} = +1 \quad \Longrightarrow \quad {\mathbf{x}^{(i)}}^\top \mathbf{w} + b \geq +1$$
$$y^{(i)} = -1 \quad \Longrightarrow \quad {\mathbf{x}^{(i)}}^\top \mathbf{w} + b \leq -1$$

Support vector machines

- H^- and H^+ intersect the negatively and positively labeled data points *closest* to H, respectively.



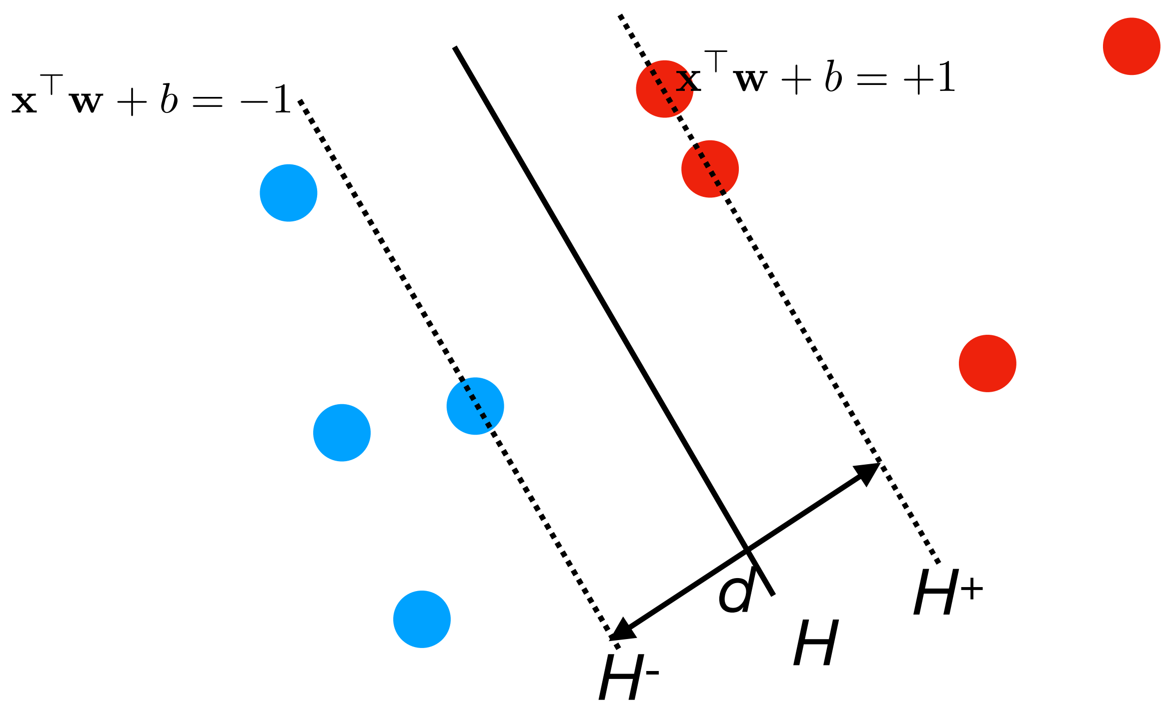
- These two sets of **constraints** can be unified:

$$y^{(i)}(\mathbf{x}^{(i)\top}\mathbf{w} + b) \geq 1 \quad \forall i$$

Inequality constraints

Maximizing the margin

- How do we maximize the margin *d*?

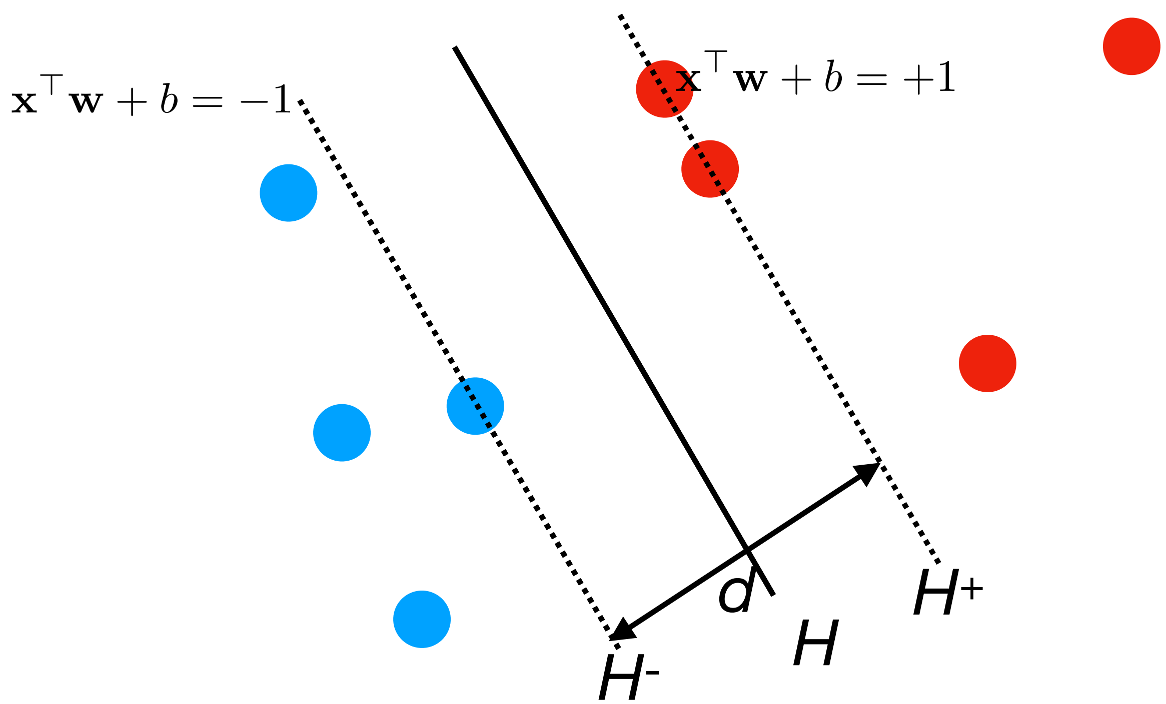


- Since H^- is (-1-*b*)/|**w**| from the origin and H^+ is (1-*b*)/|**w**| from the origin, then the margin must be:

$$d = \frac{1-b}{|\mathbf{w}|} - \frac{-1-b}{|\mathbf{w}|} = \frac{2}{|\mathbf{w}|}$$

Maximizing the margin

- How do we maximize the margin *d*?



- To *maximize d*=2/|**w**|, we can thus *minimize* |**w**|/2 or (equivalently) minimize:

$$\frac{1}{2}\mathbf{w}^\top \mathbf{w}$$

Optimization objective (cost function)

SVM optimization problem

- Putting the parts together, we wish to:
  - Minimize: $\frac{1}{2}\mathbf{w}^\top \mathbf{w}$
  - Subject to: $y^{(i)}(\mathbf{x}^{(i)\top}\mathbf{w} + b) \geq 1 \quad \forall i$

SVM optimization problem

- Putting the parts together, we wish to:
  - Minimize: $\frac{1}{2}\mathbf{w}^\top \mathbf{w}$
  - Subject to: $y^{(i)}(\mathbf{x}^{(i)\top}\mathbf{w} + b) \geq 1 \quad \forall i$
- This is a **quadratic programming** problem: quadratic objective with linear inequality (and/or equality) constraints.
- There are many efficient solvers for quadratic programs.

SVM optimization problem

- However, we can get some intuition by doing some analytical simplification.

- Similar as with Lagrange multipliers, with KKT conditions we also define a function *L* of the optimization variables (**w**) and the dual variables (*α*):

$$L(\mathbf{w}, b, \alpha) \quad = \quad \frac{1}{2}\mathbf{w}^\top\mathbf{w} - \sum_{i=1}^{n} \alpha^{(i)} \left(y^{(i)} \left({\mathbf{x}^{(i)}}^\top \mathbf{w} + b - 1 \right) \right)$$

Objective **Inequality constraints**

SVM optimization problem

- However, we can get some intuition by doing some analytical simplification.

- Similar as with Lagrange multipliers, with KKT conditions we also define a function *L* of the optimization variables (**w**) and the dual variables (*α*):

$$L(\mathbf{w}, b, \alpha) \quad = \quad \frac{1}{2}\mathbf{w}^\top \mathbf{w} - \sum_{i=1}^{n} \alpha^{(i)} \left(y^{(i)} \left(\mathbf{x}^{(i)\top} \mathbf{w} + b - 1 \right) \right)$$

Objective Inequality constraints

- We then compute the gradient of *L*, set to 0, and solve (numerically)…

SVM optimization problem

- As shown below, an optimal **w** will always be a **linear combination** of the data points $\mathbf{x}^{(i)}$, weighted by the $\alpha^{(i)}$.

$$
\begin{aligned}
L(\mathbf{w}, b, \alpha) &= \frac{1}{2}\mathbf{w}^\top \mathbf{w} - \sum_{i=1}^{n} \alpha^{(i)} \left(y^{(i)} \left(\mathbf{x}^{(i)\top} \mathbf{w} + b - 1 \right) \right) \\
\frac{\partial L}{\partial \mathbf{w}} &= \mathbf{w} - \sum_{i=1}^{n} \alpha^{(i)} y^{(i)} \mathbf{x}^{(i)} \\
\implies \mathbf{w} &= \sum_{i=1}^{n} \alpha^{(i)} y^{(i)} \mathbf{x}^{(i)}
\end{aligned}
$$

SVM optimization problem

- As shown below, an optimal **w** will always be a **linear combination** of the data points $\mathbf{x}^{(i)}$, weighted by the $\alpha^{(i)}$.

- As mentioned earlier, only some of the *n* constraints will be active — for the others (inactive), $\alpha^{(i)} = 0$.

$$
\begin{aligned}
L(\mathbf{w}, b, \alpha) &= \frac{1}{2}\mathbf{w}^\top \mathbf{w} - \sum_{i=1}^{n} \alpha^{(i)} \left(y^{(i)} \left(\mathbf{x}^{(i)\top} \mathbf{w} + b - 1 \right) \right) \\
\frac{\partial L}{\partial \mathbf{w}} &= \mathbf{w} - \sum_{i=1}^{n} \alpha^{(i)} y^{(i)} \mathbf{x}^{(i)} \\
\implies \mathbf{w} &= \sum_{i=1}^{n} \alpha^{(i)} y^{(i)} \mathbf{x}^{(i)}
\end{aligned}
$$

SVM optimization problem

- This means that **w** will actually only be a linear combination of a *subset* of the input vectors $\mathbf{x}^{(i)}$.
    - The data $\mathbf{x}^{(i)}$ for which $\alpha^{(i)} > 0$ are called **support vectors**.

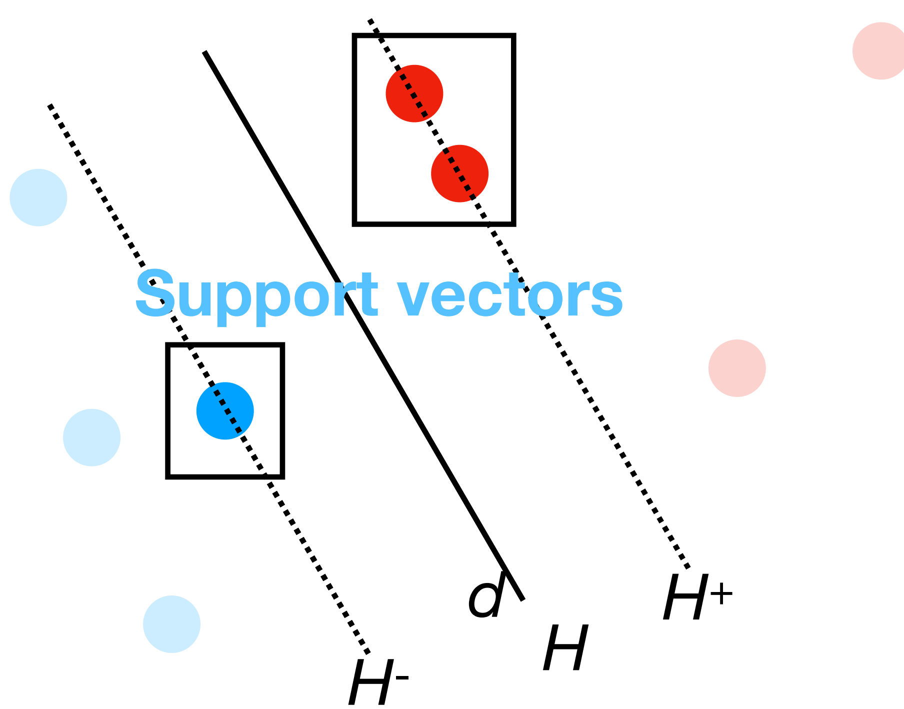
- The other data (for which $\alpha^{(i)} = 0$) are essentially *irrelevant — they do not influence the location or orientation of the hyperplane.*

$$
\begin{aligned}
L(\mathbf{w}, b, \alpha) &= \frac{1}{2}\mathbf{w}^\top \mathbf{w} - \sum_{i=1}^{n} \alpha^{(i)} \left(y^{(i)} \left(\mathbf{x}^{(i)\top} \mathbf{w} + b - 1 \right) \right) \\
\frac{\partial L}{\partial \mathbf{w}} &= \mathbf{w} - \sum_{i=1}^{n} \alpha^{(i)} y^{(i)} \mathbf{x}^{(i)} \\
\Longrightarrow \mathbf{w} &= \sum_{i=1}^{n} \alpha^{(i)} y^{(i)} \mathbf{x}^{(i)}
\end{aligned}
$$

SVM optimization problem

- This means that **w** will actually only be a linear combination of a *subset* of the input vectors $\mathbf{x}^{(i)}$.
    - The data $\mathbf{x}^{(i)}$ for which $\alpha^{(i)} > 0$ are called **support vectors**.

- The other data (for which $\alpha^{(i)} = 0$) are essentially *irrelevant — they do not influence the location or orientation of the hyperplane.*



Quadratic programming

Quadratic programming

- Quadratic programming is *not* a kind of computer programming.
- **Quadratic programming (QP)** problems are a kind of mathematical optimization problem:
  - Quadratic objective function (which we want to minimize or maximize).
  - Linear equality and/or inequality constraints.
- Same vein as linear programming, dynamic programming.

Quadratic programming

- Nonetheless, quadratic programs are typically *solved* using computer programs.
- As part of homework 4, you will use an off-the-shelf Python-based quadratic programming solver (**`cvxopt`**).