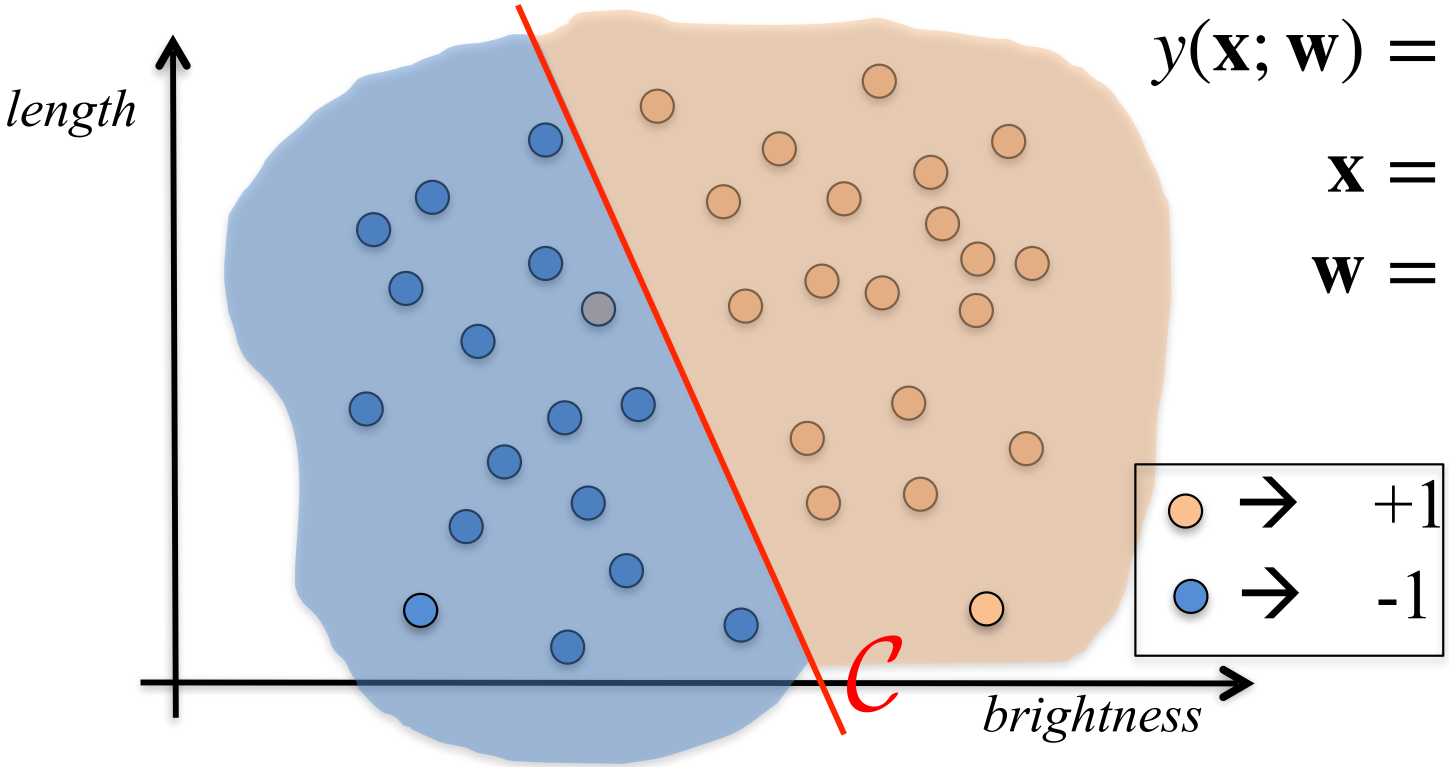
Linear Classification

Pascal Fua
IC-CVLab

Reminder: Linear 2D Model

Some algorithm $\longrightarrow$

$$\begin{pmatrix} brightness \\ length \end{pmatrix}$$



$$y(\mathbf{x};\mathbf{w}) = \mathrm{sign}(\mathrm{w_x b + w_y l + w_0})$$

$$\mathbf{x} = [b, l]$$

$$\mathbf{w} = [w_x, w_y, w_0]$$

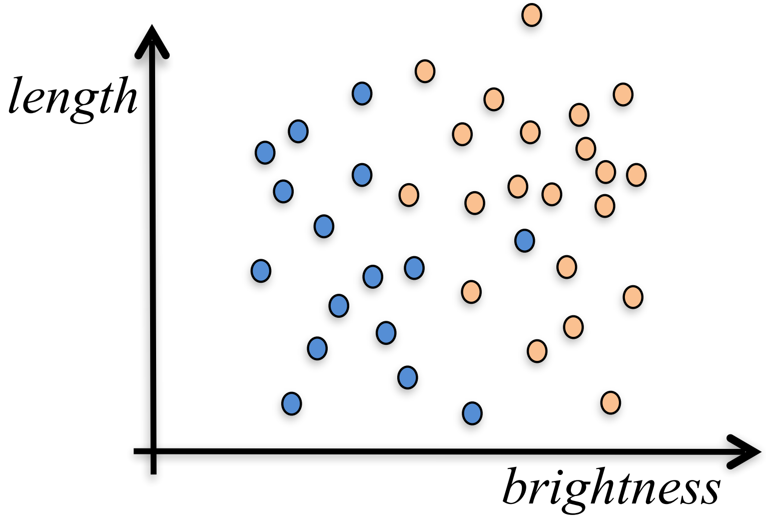
How do we find **w**?

Reminder: Training vs Testing

Supervised training:

Given a **training** set $\{(\mathbf{x}_n, t_n)_{1 \leq n \leq N}\}$ minimize:

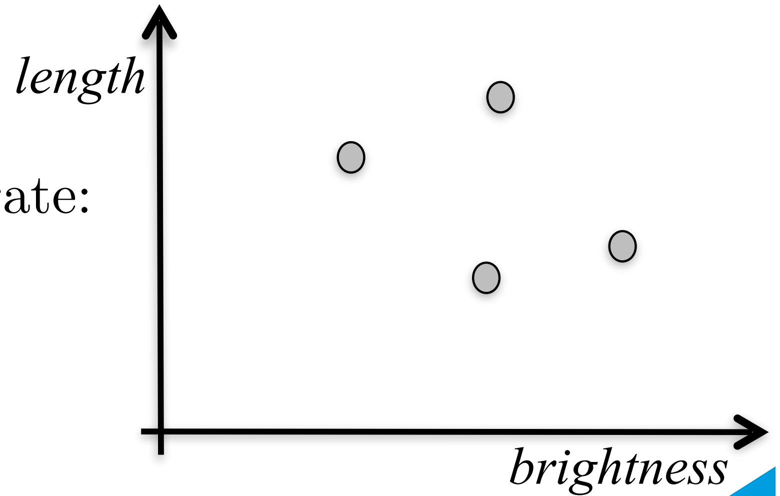
$$E(\mathbf{w}) = \sum_{n=1}^{N} L(y(\mathbf{x}_n; \mathbf{w}), t_n)$$
$$= \sum_{n=1}^{N} [y(\mathbf{x}_n; \mathbf{w}) \neq t_n]$$



Testing:

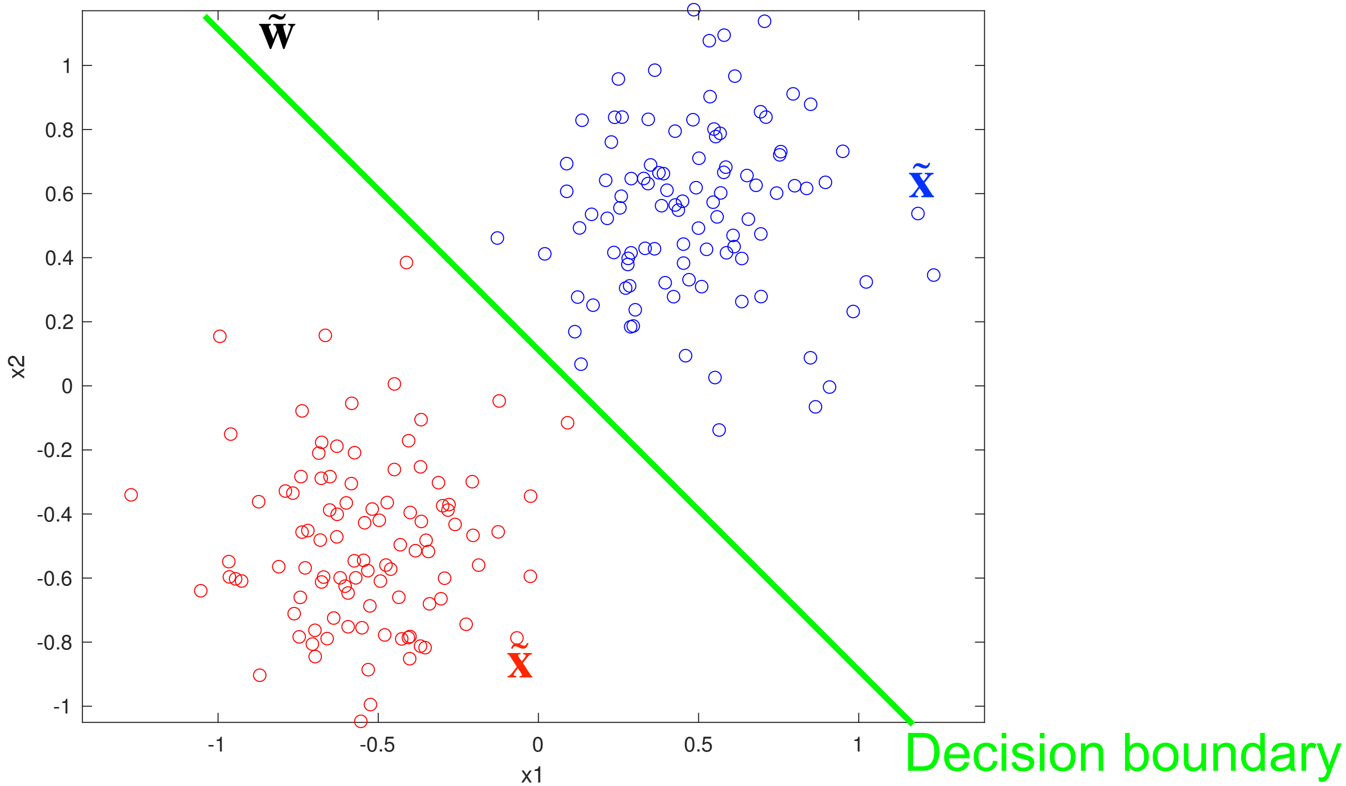
Given a **test** set $\{(\mathbf{x}_n, t_n)_{1 \leq n \leq N}\}$ compute the error rate:

$$1/N \sum_{n=1}^{N} [y(\mathbf{x}_n; \mathbf{w}) \neq \mathbf{t}_n]$$



EPFL

Desired Problem Formulation



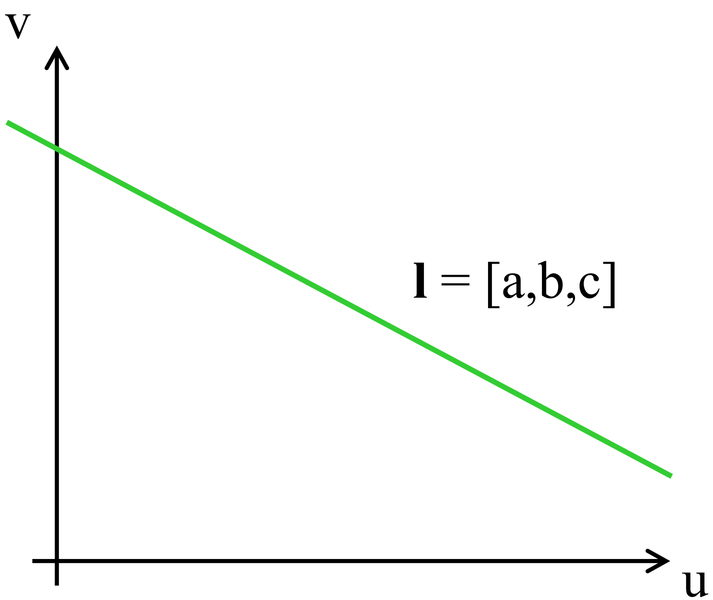
Find $\tilde{\mathbf{w}}$ such that:

- For all or most positive samples $y(\tilde{\mathbf{x}}; \tilde{\mathbf{w}}) = \tilde{\mathbf{w}} \cdot \tilde{\mathbf{x}} > 0$.
- For all or most negative samples $y(\tilde{\mathbf{x}}; \tilde{\mathbf{w}}) = \tilde{\mathbf{w}} \cdot \tilde{\mathbf{x}} < 0$

—— > Let's talk about hyperplanes.

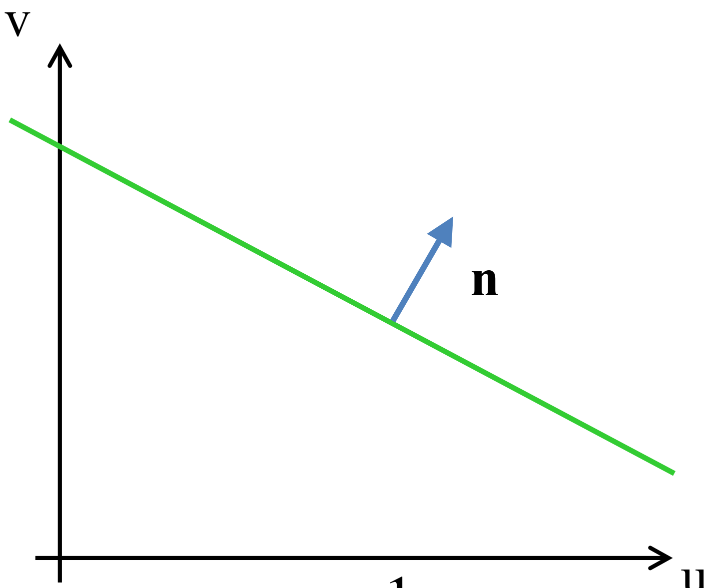
Parameterizing Lines

Equation of a line



$$(u, v) \in R^2, au + bv + c = 0$$

Normal vector

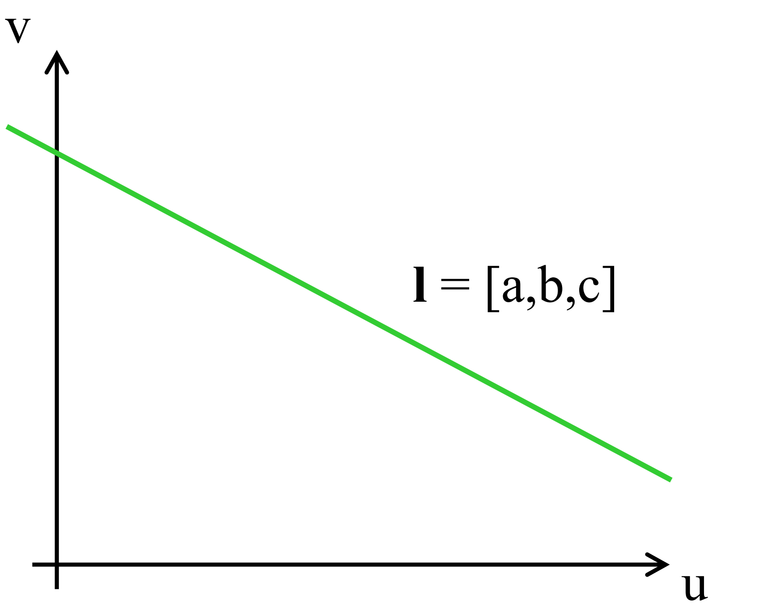


$$\mathbf{n} = \frac{1}{\sqrt{a^2 + b^2}}[a, b]$$

$[a, b, c]$ and $\dfrac{1}{\sqrt{a^2 + b^2}}[a, b, c]$ define the same line.

Normalized Parameterization

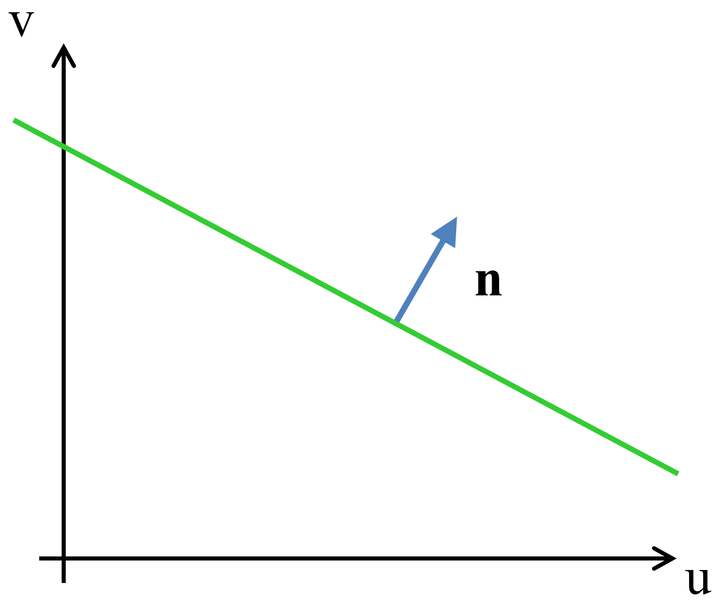
Equation of a line



$$(u,v) \in R^2, au + bv + c = 0$$

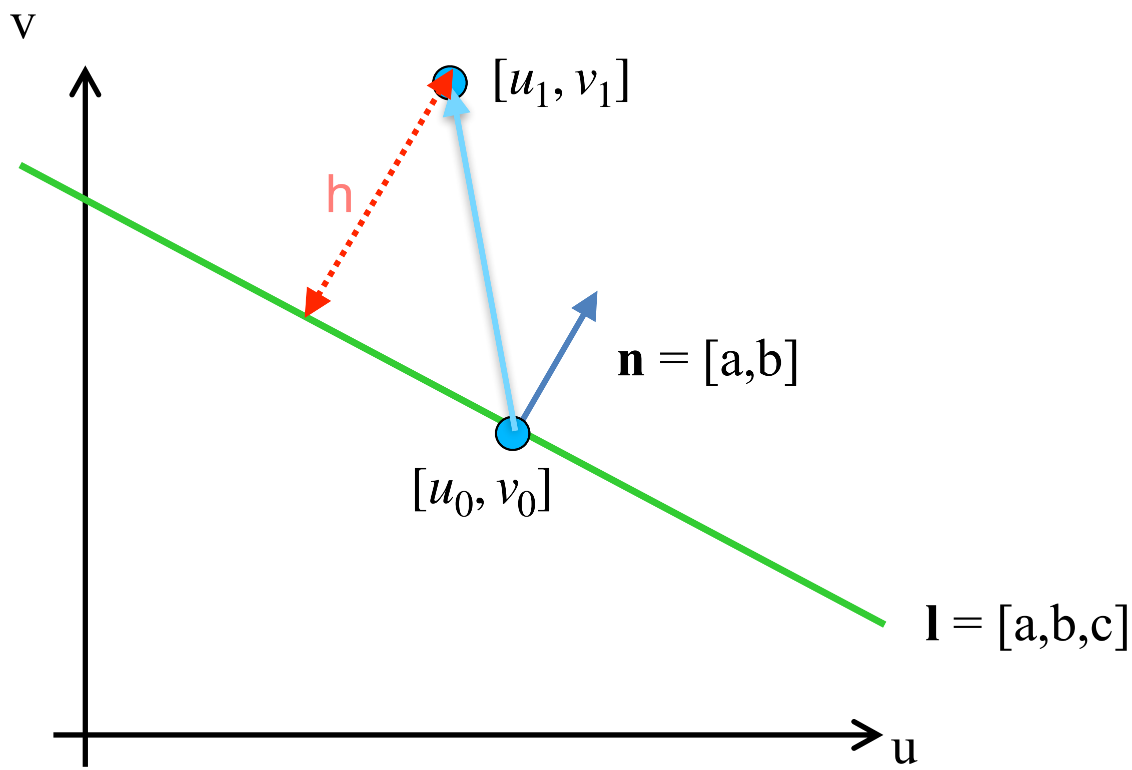
with $a^2 + b^2 = 1$

Normal vector



$$\mathbf{n} = [a, b]$$

Signed Distance to Line



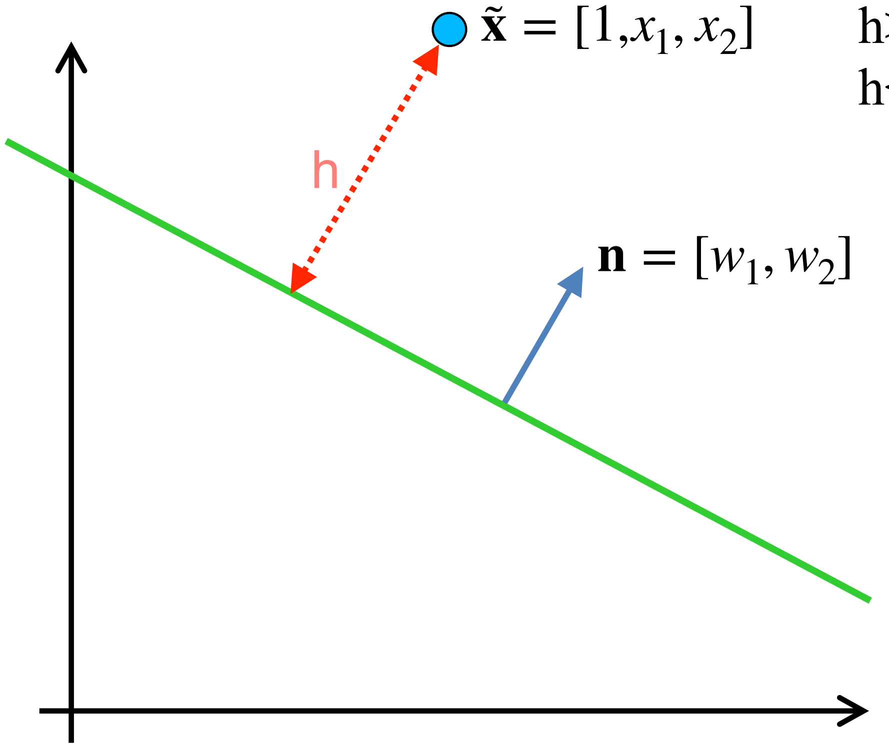
Signed distance: $h = \mathbf{n} \cdot [u_1 - u_0, v_1 - v_0]$

$$= a(u_1 - u_0) + b(v_1 - v_0)$$
$$= au_1 + bv_1 - (au_0 - bv_0)$$
$$= au_1 + bv_1 + c - (au_0 - bv_0 - c)$$
$$= au_1 + bv_1 + c$$

h=0: Point is on the line.
h>0: Point on one side.
h<0: Point on the other side.

Signed Distance Reformulated

h=0: Point is on the line.
h>0: Point in the normal's direction.
h<0: Point in the other direction.



Notation:

$$\mathbf{x} = [x_1, x_2]$$

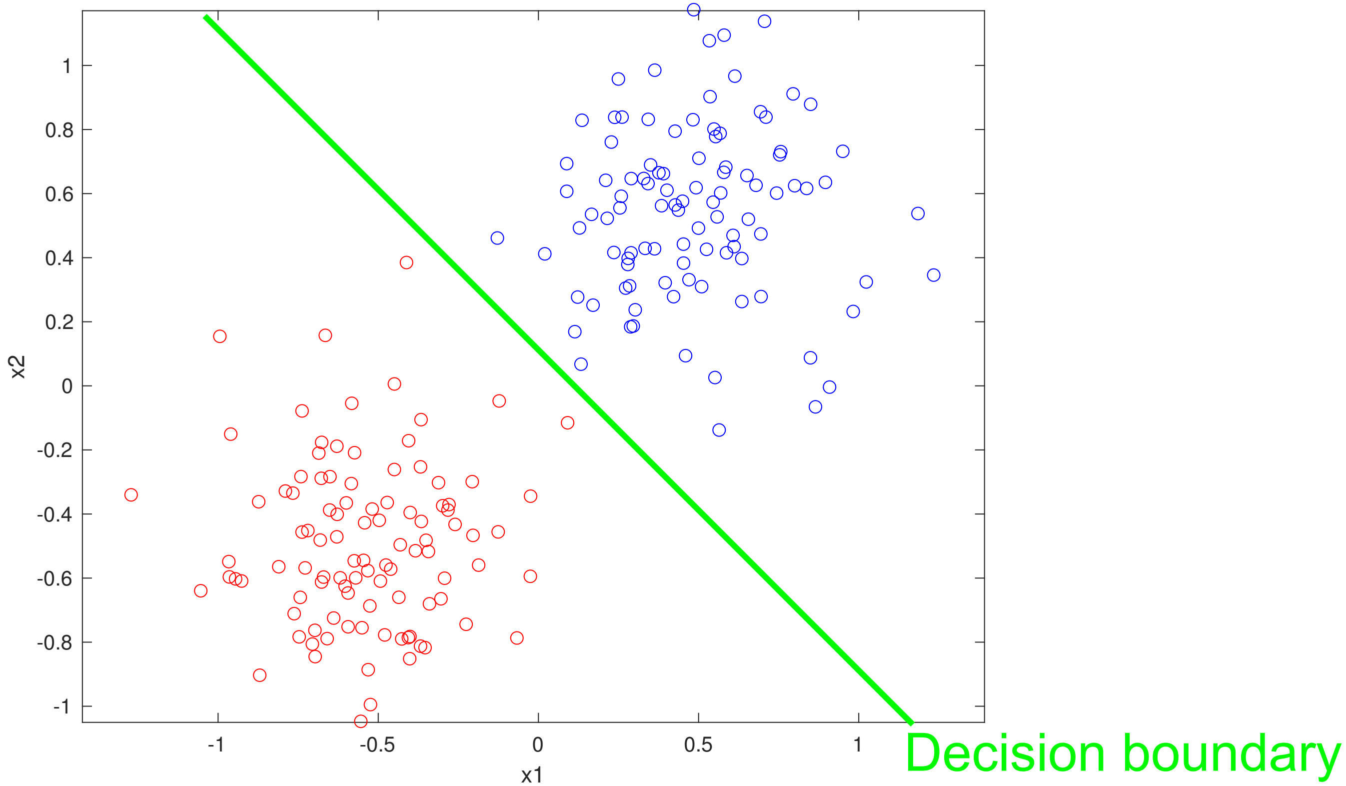
$$\tilde{\mathbf{x}} = [1, x_1, x_2]$$

Signed distance:

$$h = w_0 + w_1 x_1 + w_2 x_2$$

$$= \tilde{\mathbf{w}} \cdot \tilde{\mathbf{x}}$$

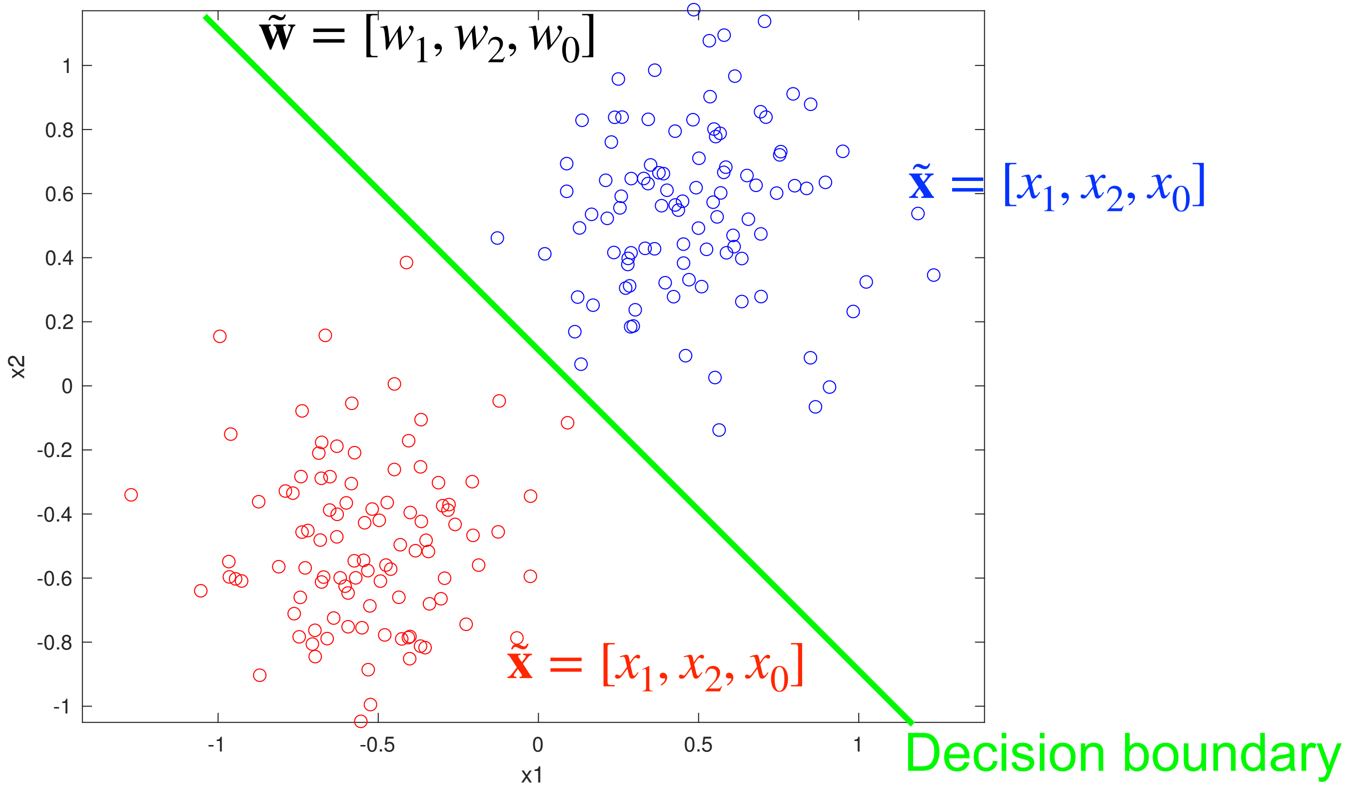
Reminder: Binary Classification



Two classes shown as different colors:

- The label $y \in \{-1,1\}$ or $y \in \{0,1\}$.
- The samples with label 1 are called positive samples.
- The samples with label -1 or 0 are called negative samples.

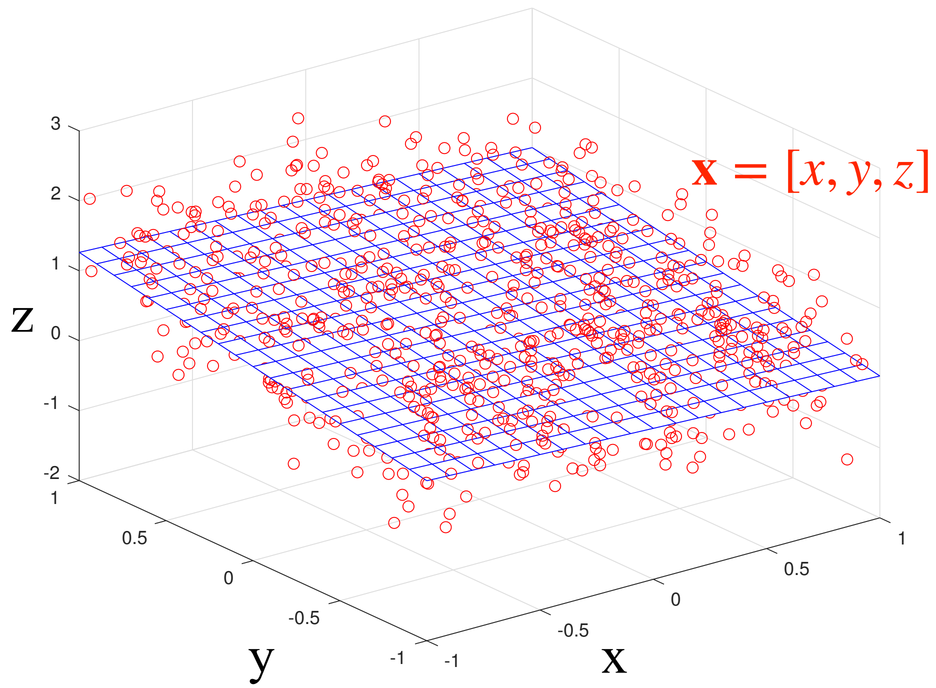
Problem Statement in 2D

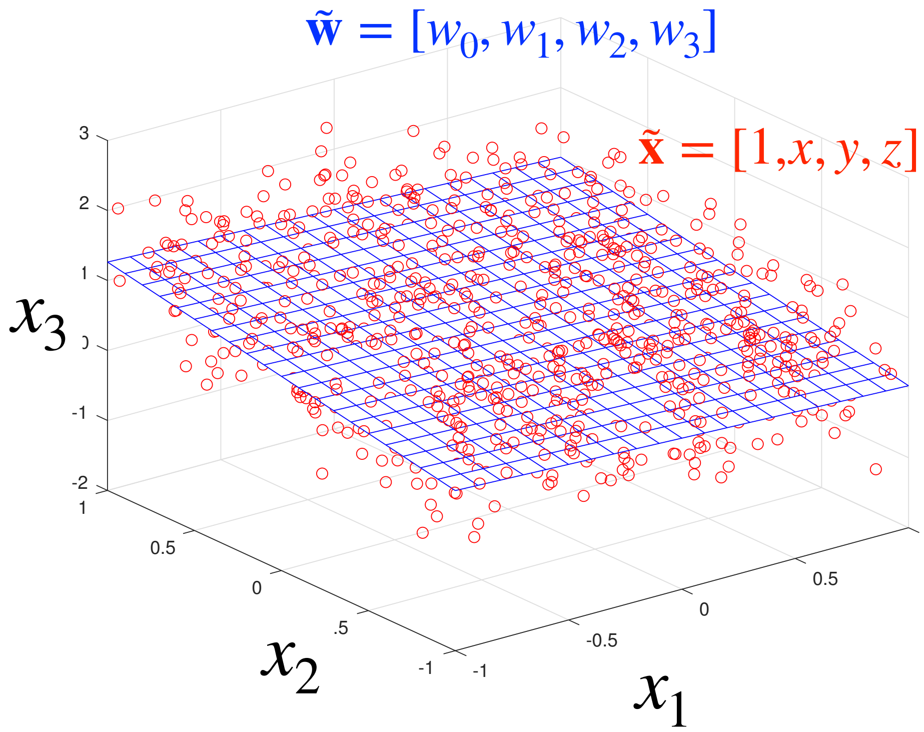


Find $\tilde{\mathbf{w}}$ such that:

- For all or most positive samples $\tilde{\mathbf{w}} \cdot \tilde{\mathbf{x}} > 0$.
- For all or most negative samples $\tilde{\mathbf{w}} \cdot \tilde{\mathbf{x}} < 0$.

Signed Distance in 3D



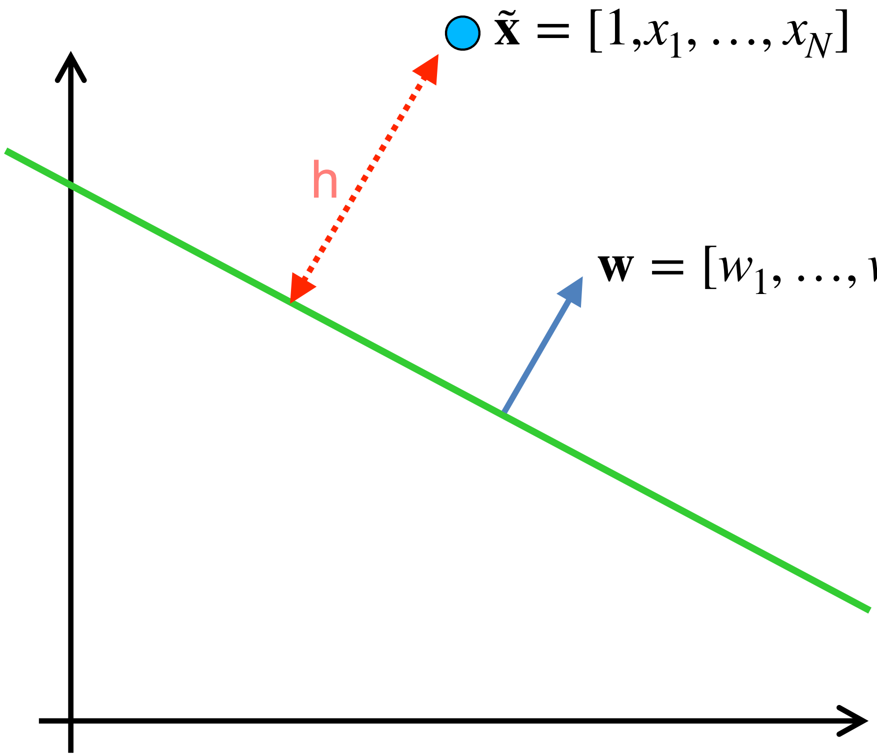


$\mathbf{x} \in R^3,\ 0 = ax + by + cz + d$

$\tilde{\mathbf{x}} \in R^4,\ \tilde{\mathbf{w}} \cdot \tilde{\mathbf{x}} = 0$

Signed distance $h = \tilde{\mathbf{w}} \cdot \tilde{\mathbf{x}}$ if $w_1^2 + w_2^2 + w_3^2 = 1$.

Signed Distance in N Dimensions



h=0: Point is on the decision boundary.
h>0: Point on one side.
h<0: Point on the other side.

$\tilde{\mathbf{w}} = [w_0, w_1, \ldots, w_N]$ with $\sum_{i=1}^{N} w_i^2 = 1$

Notation: $\mathbf{x} = [x_1, \ldots, x_N]$

$\tilde{\mathbf{x}} = [1, x_1, \ldots, x_N]$

Hyperplane: $\mathbf{x} \in R^n, \quad 0 = \tilde{\mathbf{w}} \cdot \tilde{\mathbf{x}}$

$$= w_0 + w_1 x_1 + \ldots w_N x_N$$

Signed distance: $h = \tilde{\mathbf{w}} \cdot \tilde{\mathbf{x}}$

Problem Statement in N Dimensions

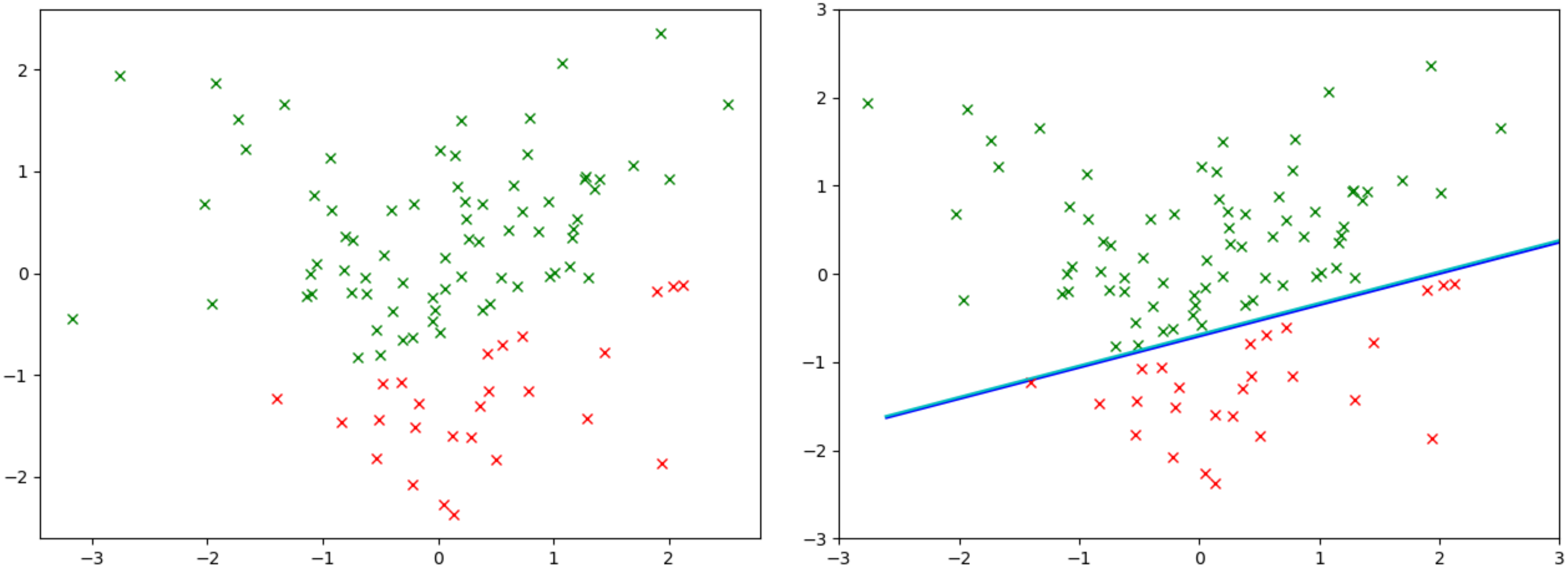
Hyperplane: $\mathbf{x} \in R^N, \tilde{\mathbf{w}} \cdot \tilde{\mathbf{x}} = 0$, with $\tilde{\mathbf{x}} = [1 \,|\, \mathbf{x}]$.

Signed distance: $\tilde{\mathbf{w}} \cdot \tilde{\mathbf{x}}$, with $\tilde{\mathbf{w}} = [w_0 \,|\, \mathbf{w}]$ and $||\mathbf{w}|| = 1$.

Find $\tilde{\mathbf{w}}$ such that

- for all or most positive samples $\tilde{\mathbf{w}} \cdot \tilde{\mathbf{x}} > 0$,
- for all or most negative samples $\tilde{\mathbf{w}} \cdot \tilde{\mathbf{x}} < 0$.

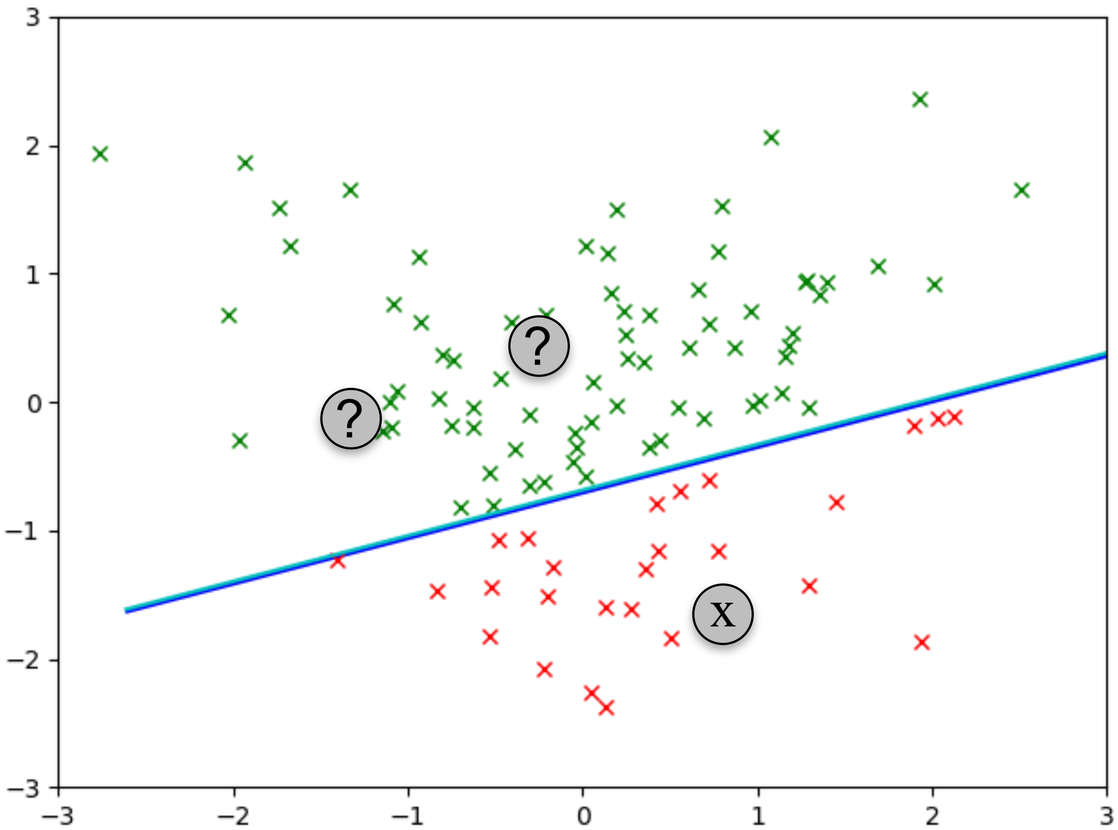
Perceptron



$$\text{Minimize: } E(\tilde{\mathbf{w}}) = -\sum_{n=1}^{N} \text{sign}(\tilde{\mathbf{w}} \cdot \tilde{\mathbf{x}}_n) t_n$$

- Set $\tilde{\mathbf{w}}_1$ to $\mathbf{0}$.
- Iteratively, pick a random index n.
  - If $\tilde{\mathbf{x}}_n$ is correctly classified, do nothing.
  - Otherwise, $\tilde{\mathbf{w}}_{t+1} = \tilde{\mathbf{w}}_t + t_n \tilde{\mathbf{x}}_n$.

Test Time

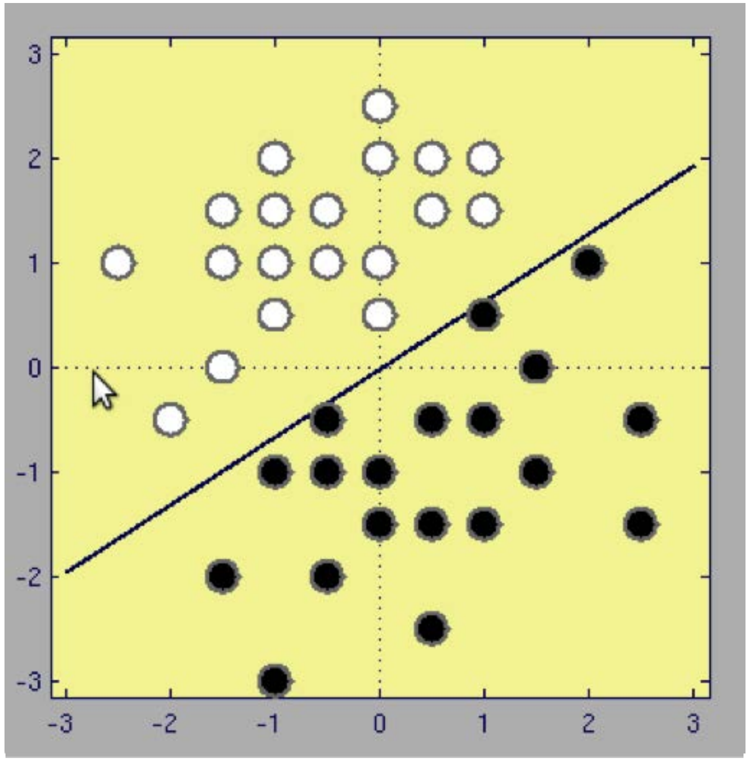


$$y(\mathbf{x}; \tilde{\mathbf{w}}) = \begin{cases} 1 & \text{if} \quad \tilde{\mathbf{w}} \cdot \tilde{\mathbf{x}} \geq 0 \, , \\ -1 & \text{otherwise.} \end{cases}$$

$$\tilde{\mathbf{x}} = [1, x_1, ..., x_N]$$

EPFL

Centered Perceptron



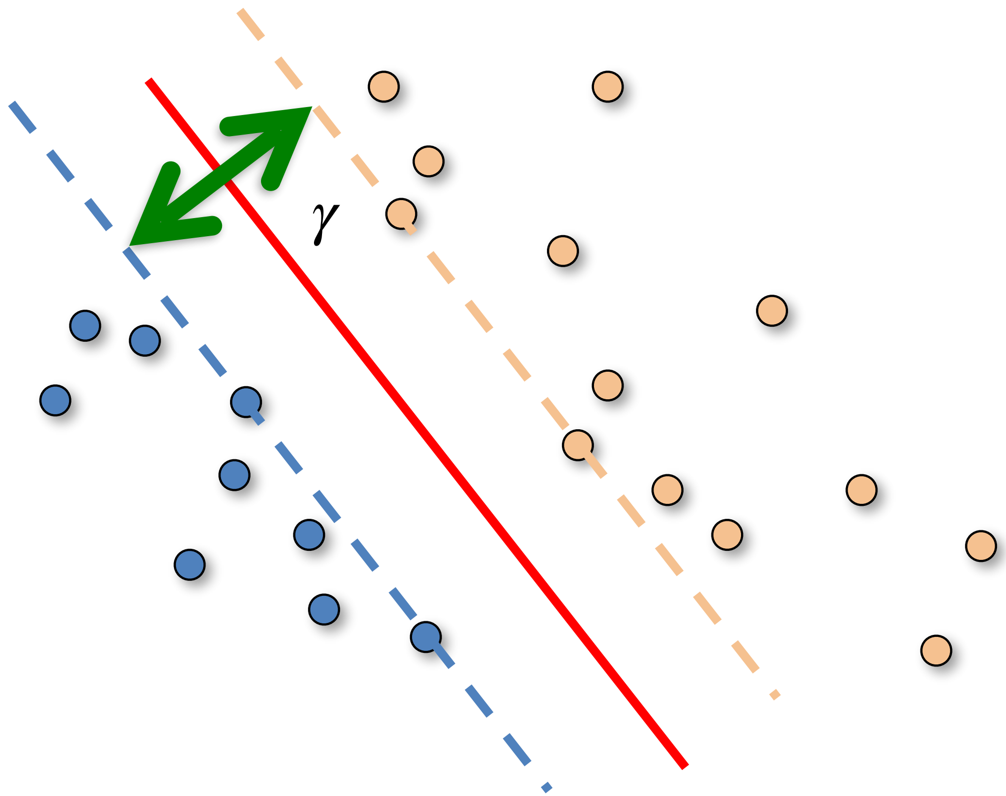
The two populations can be translated so that the decision boundary goes through the origin.

Given a **training** set $\{(\mathbf{x}_n, t_n)_{1 \leq n \leq N}\}$ minimize:

$$E(\mathbf{w}) = -\sum_{n=1}^{N} \text{sign}(\mathbf{w} \cdot \mathbf{x}_n) t_n$$

- Center the $\mathbf{x}_n$s so that $w_0 = 0$.
- Set $\mathbf{w}_1$ to $\mathbf{0}$.
- Iteratively, pick a random index n.
  - If $\mathbf{x}_n$ is correctly classified, do nothing.
  - Otherwise, $\mathbf{w}_{t+1} = \mathbf{w}_t + t_n \mathbf{x}_n$.

Convergence Theorem



γ is the margin

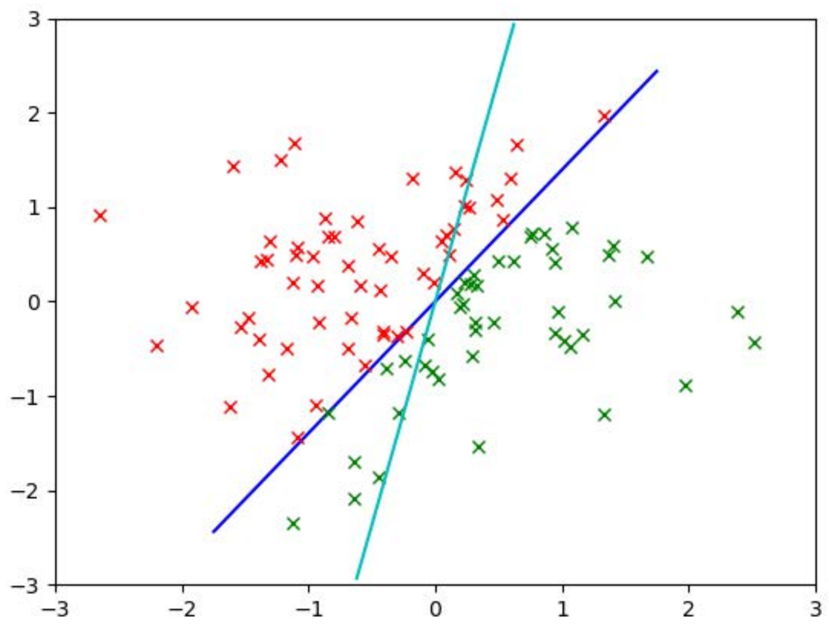
If there is a number $\gamma > 0$ and a parameter vector $\mathbf{w}^*$, with $||\mathbf{w}^*|| = 1$, such that

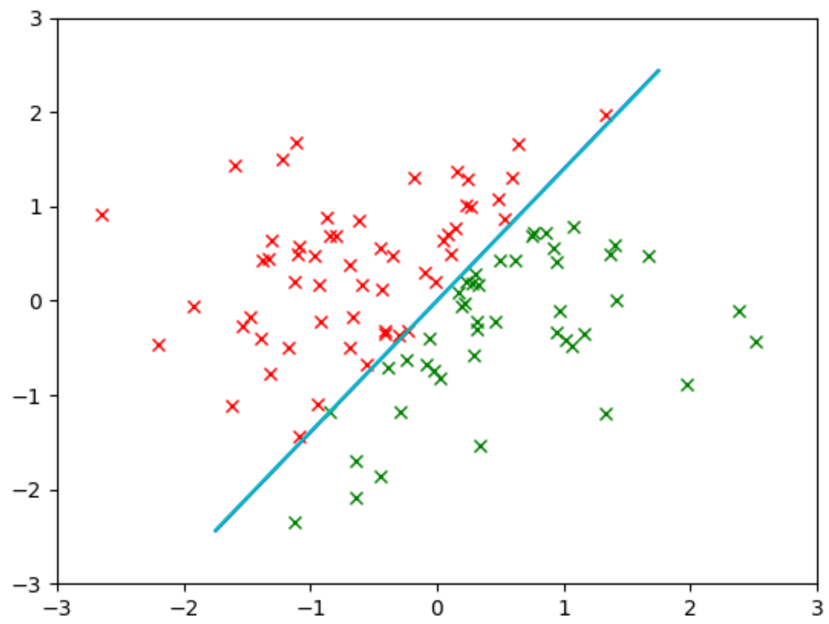
$$\forall n,\ t_n(\mathbf{w}^* \cdot \mathbf{x}_n) > \gamma\ ,$$

the perceptron algorithm makes $\boxed{\text{at most } \frac{R^2}{\gamma^2} \text{ errors,}}$ where $R = max_n ||\mathbf{x}_n||$.

EPFL

What if γ is Small?






```
for n in range(nIt):
  for i in range(ns):
```

- If $\mathbf{x}_n$ is correctly classified, do nothing.
- Otherwise, $\mathbf{w}_{t+1} = \mathbf{w}_t + t_n \mathbf{x}_n$.

Randomizing helps!

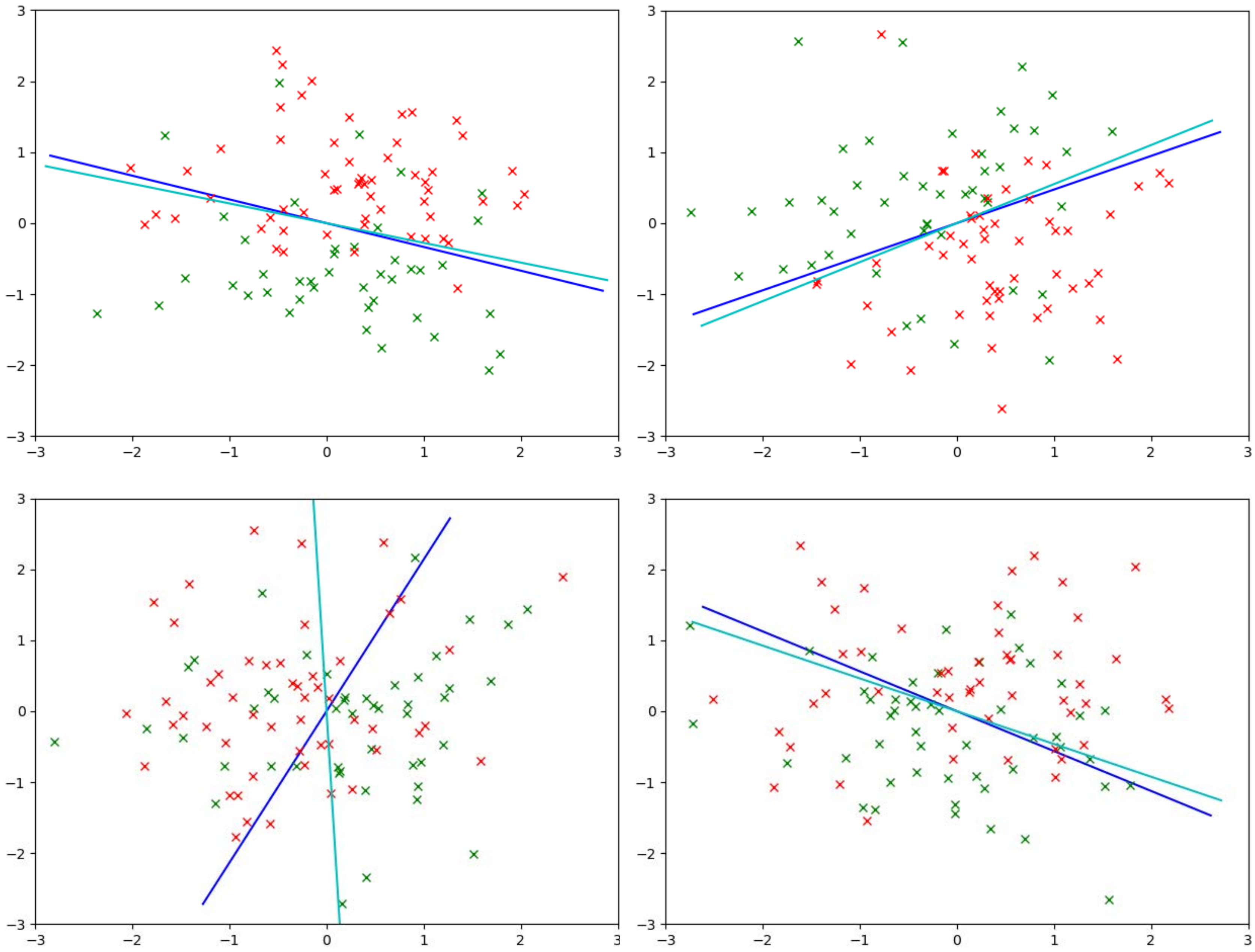
```
for n in range(nIt):
  inds=list(range(ns))
  random.shuffle(inds)
  for i in range(inds):
```

- If $\mathbf{x}_n$ is correctly classified, do nothing.
- Otherwise, $\mathbf{w}_{t+1} = \mathbf{w}_t + t_n \mathbf{x}_n$.

What if g Does Not Exist?

20% of outliers

30% of outliers



Still works up to a point but no guarantee!

EPFL

Optional: Python Implementation (1)

```
def perceptronRand(xs,ys,nIt=200,randP=True):
    N, D = xs.shape                     # Get data shape.
    w = np.zeros(D)                     # Init weights.
    for it in range(nIt):               # Train.
        allCorrect = True               # Generate indices.
        inds = np.random.permutation(N) if randP else np.arange(N)
        for i in inds:
            x = xs[i]                   # Pick one sample.
            y = 2*(np.inner(x,w) > 0)-1 # Predict the label.
            if y != ys[i]:              # Misclassified.
                w += ys[i] * x          # Update weights.
                w /= np.linalg.norm(w)  # Normalize length.
                allCorrect = False      # Something has changed.
        print('It {}: {}'.format(it + 1,linearAccuracy(xs, ys, w)))
        if allCorrect:
            break                       # Finish training.
    return w

def linearAccuracy(xs,ys,ws):
    return(sum(ys  == (2 * (xs @ ws >0)) - 1) * 100/len(ys))
```

Call to numpy. Mostly coded in C or Fortran.

Optional: Python Implementation (2)

```
def perceptronRand(xs,ys,nIt=200,randP=True):
    N, D = xs.shape                        # Get data shape.
    w = np.zeros(D)                        # Init weights.
    bestW = None
    bestA = 0.0
    for it in range(nIt):                  # Train.
        allCorrect = True                  # Generate indices.
        inds = np.random.permutation(N) if randP else np.arange(N)
        for i in inds:
            x = xs[i]                      # Pick one sample.
            y = 2*(np.inner(x,w) > 0)-1   # Predict the label.
            if y != ys[i]:                 # Misclassified.
                w += ys[i] * x             # Update weights.
                w /= np.linalg.norm(w)     # Normalize length.
                allCorrect = False         # Something has changed.
                acc  = linearAccuracy(xs, ys, w)
                if(acc>bestA):
                    bestW = w
                    bestA = acc
        print('It {}: {}'.format(it + 1,bestA))
        if allCorrect:
            break                          # Finish training.
    return bestW
```

Record best solution.

EPFL

Optional: JAVA Implementation

```
import org.nd4j.linalg.api.ndarray.INDArray;
import org.nd4j.linalg.factory.Nd4j;
import java.lang.Float;

class Perceptron {
    public Perceptron() {}

    public static INDArray perceptronRand(INDArray xs, INDArray ys, int nIt, boolean randP){
        long[] shape = xs.shape();                          // Get data shape
        long N      = shape[0];
        long D      = shape[1];

        INDArray w   = Nd4j.zeros(D,1); // Init weights

        for (int it = 0; it < nIt; it++){
            boolean allCorrect = true;
            INDArray inds = Nd4j.arange(0,D);                // Generate samples indices.

            if (randP)
                Nd4j.shuffle(inds);

            for (int i = 0; i < N; i++){
                INDArray x = xs.getRow(i);                   // Pick one sample.
                INDArray y = (x.mmul(w).gt(0)).mul(2).sub(1) ;   // Predict the label.
                if (y.data().asFloat()[0] != ys.getRow(i).data().asFloat()[0]){ // Misclassified.

                    w = x.mul(ys.getRow(i)).add(w.transpose());  // Update weights.
                    w = w.div(w.norm2().add(1e-3)).transpose();  // Unit normal length.

                    allCorrect = false;
                }
            }
            System.out.println("It " + it + ": " + linearAccuracy(xs, ys, w));
            if (allCorrect){
                break;
            }
        }
        return w;
    }

    public static String linearAccuracy(INDArray xs,INDArray ys,INDArray w){
            INDArray y = (xs.mmul(w).gt(0)).mul(2).sub(1);
            return Nd4j.sum((y.eq(ys))).div(4).toString();
    }

public class Main{
    public static void main (String[] args){

        INDArray xs      = Nd4j.create(new float[][]{{1,0},{0,1},{1,1},{0,0}});
        INDArray ys      = Nd4j.create(new float[][]{{1},{1},{1},{-1}});
        int nIt          = 200;
        boolean randP    = true;
        INDArray weights = Perceptron.perceptronRand(xs, ys, nIt, randP);
    }
}
```

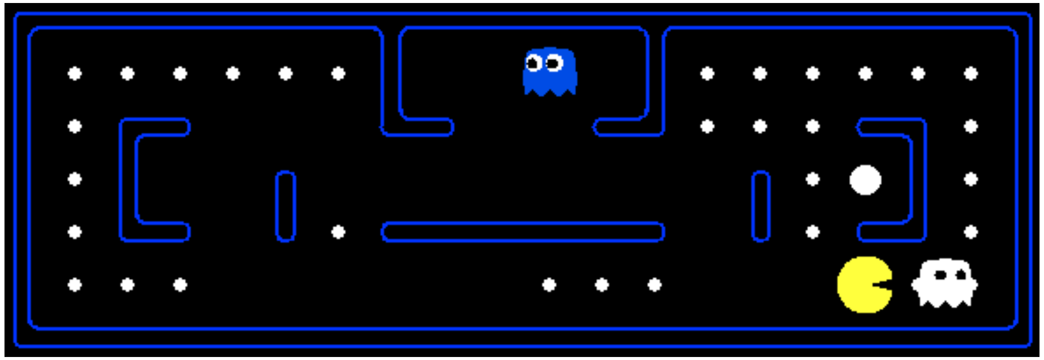
More verbose!

NumPy/SciPy

The time-critical loops are usually **implemented in C, C++ or Fortran**. Parts of SciPy are thin layers of code on top of the scientific routines that are freely available at http://www.netlib.org/. Netlib is a huge repository of incredibly valuable and robust scientific algorithms written in C and Fortran.

One of the design goals of NumPy was to make it buildable without a Fortran compiler, and if you don't have LAPACK available NumPy will use its own implementation. SciPy requires a Fortran compiler to be built, and **heavily depends on wrapped Fortran code**.

Optional: Pacman Apprenticeship



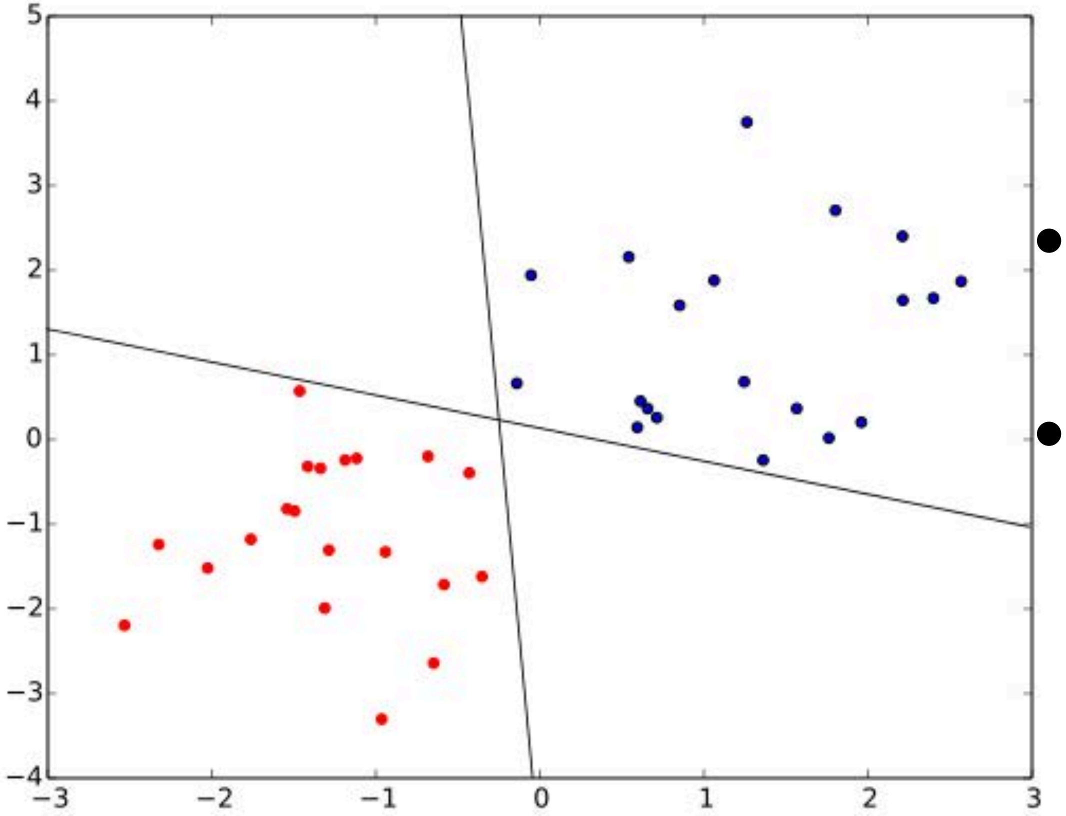
- Examples are state s.
- Correct actions are those taken by experts.
- Feature vectors defined over pairs f(a,s).
- Score of a pair taken to be **w** · f(a,s).
- Adjust **w** so that

$$\forall a, \mathbf{w} \cdot \phi(a^*, s) \geq \mathbf{w} \cdot \phi(a, s)$$

when a* is the correct action for state s.

http://ai.berkeley.edu/project_overview.html

The Problem with the Perceptron

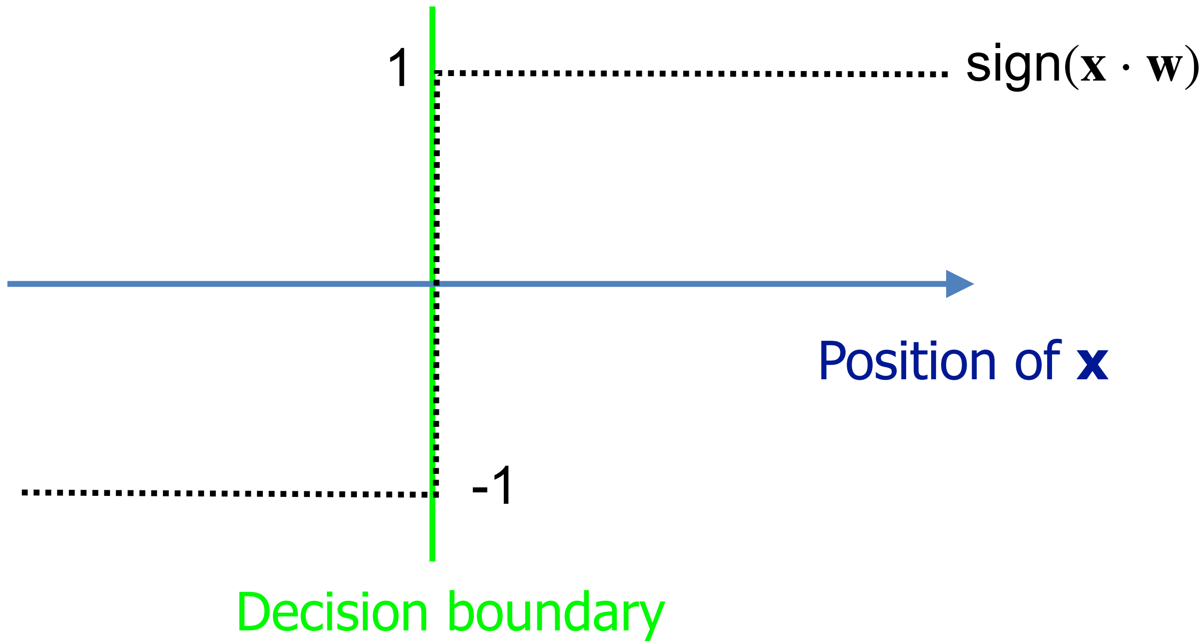


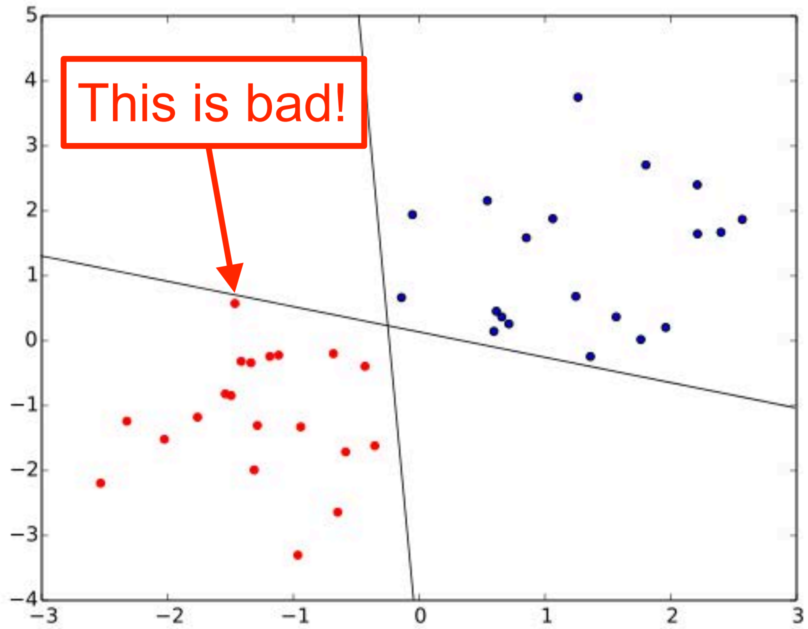
- Two different solutions among infinitely many.
- The perceptron has no way to favor one over the other.

The culprit

$$E(\tilde{\mathbf{w}}) = -\sum_{n=1}^{N} \operatorname{sign}(\tilde{\mathbf{w}} \cdot \tilde{\mathbf{x}}_n) t_n$$

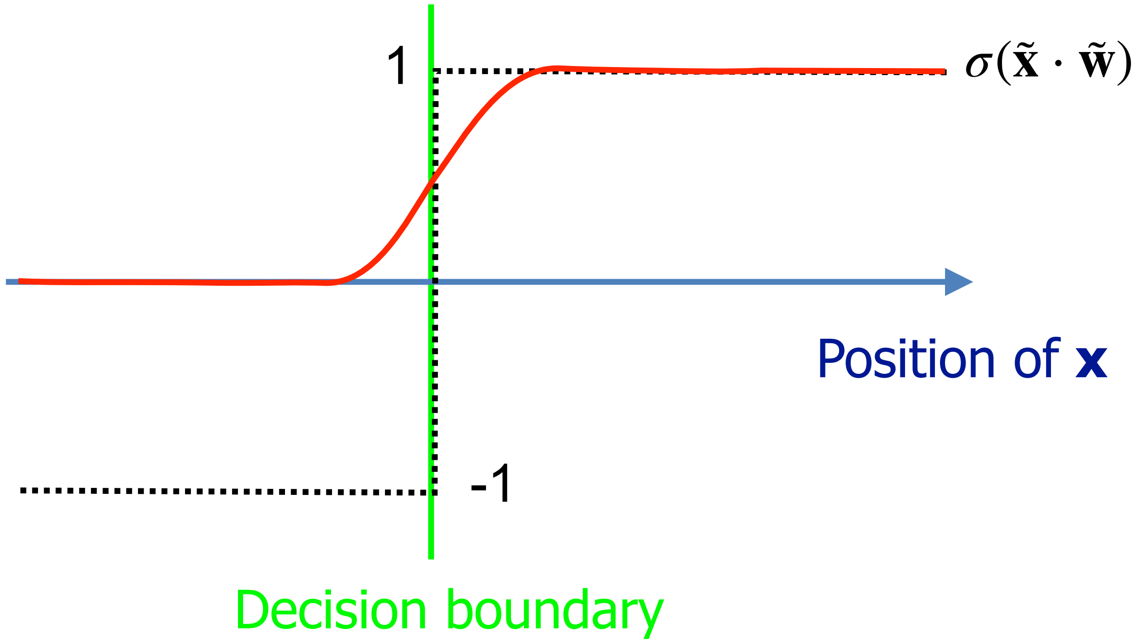
The Problem with the Perceptron





- There is no difference between close and far from the decision boundary.
- We want the positive and negative examples to be as far as possible from it.

From Perceptron to Logistic Regression



Replace the step function (black) by a smoother one (red).

- Replace the step function by a smooth function σ.
- The prediction becomes $y(\mathbf{x}; \tilde{\mathbf{w}}) = \sigma(\tilde{\mathbf{w}} \cdot \tilde{\mathbf{x}})$.
- Given the training set $\{(\mathbf{x}_n, t_n)_{1 \le n \le N}\}$ where $t_n \in \{0, 1\}$, minimize the cross-entropy

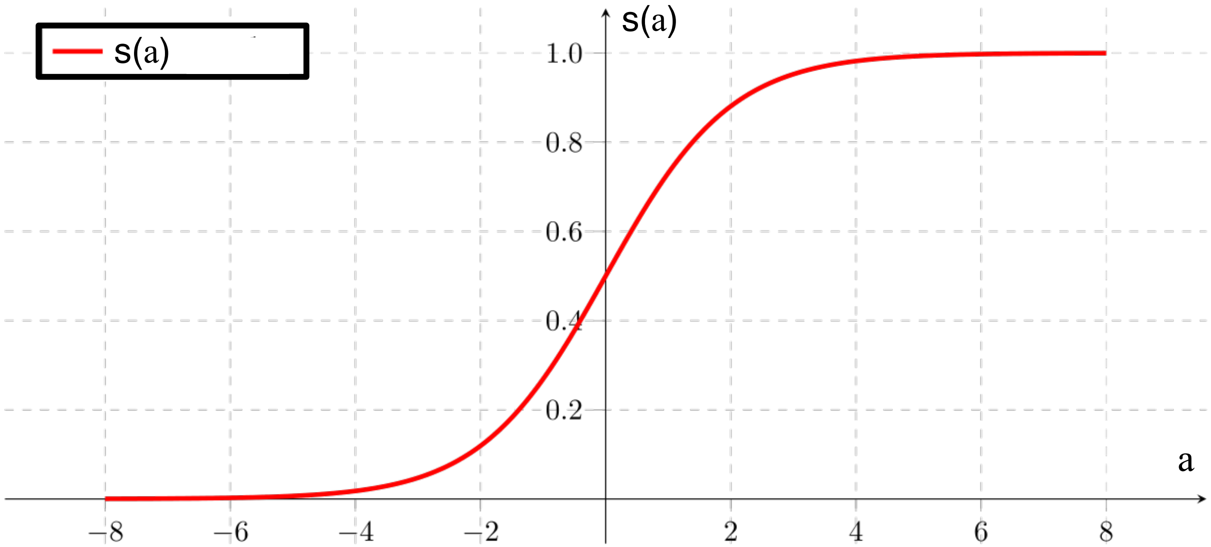
$$E(\widetilde{\mathbf{w}}) = -\sum_n \{t_n \ln y_n + (1 - t_n) \ln(1 - y_n)\}$$

$$y_n = y(\mathbf{x}_n; \widetilde{\mathbf{w}})$$

This is a convex function of w!

with respect to $\widetilde{\mathbf{w}}$.

Sigmoid Function



$$\sigma(a) = \frac{1}{1+\exp(-a)}$$

$$\frac{\partial \sigma}{\partial a} = \sigma(1-\sigma)$$

- It is infinitely differentiable.
- Its derivatives are easy to compute.
- It is asymptotically equal to zero or one.

—> Can be understood as a smoothed step function.

Cross Entropy

$$E(\tilde{\mathbf{w}}) = -\sum_n \{t_n \ln y_n + (1 - t_n) \ln(1 - y_n)\}$$

$$\nabla E(\tilde{\mathbf{w}}) = \sum_n (y_n - t_n)\tilde{\mathbf{x}}_n$$

$$y_n = \sigma(\tilde{\mathbf{w}} \cdot \tilde{\mathbf{x}}_n)$$

- $-(t_n \ln y_n + (1 - t_n)\ln(1 - y_n))$ is close to 0 if $t_n = 1$ and y_n is close to 1 or if $t_n = 0$ and y_n is close to zero. Minimizing E(**w**) encourages that.
- $-(t_n \ln y_n + (1 - t_n)\ln(1 - y_n))$ is larger if $t_n = 1$ and $y_n < 0.5$ or $t_n = 0$ and $y_n > 0.5$. Minimizing E(**w**) discourages that.
- E(**w**) is a convex function whose gradient is easy to compute.

—> The global optimum can be found very effectively.

Probabilistic Interpretation

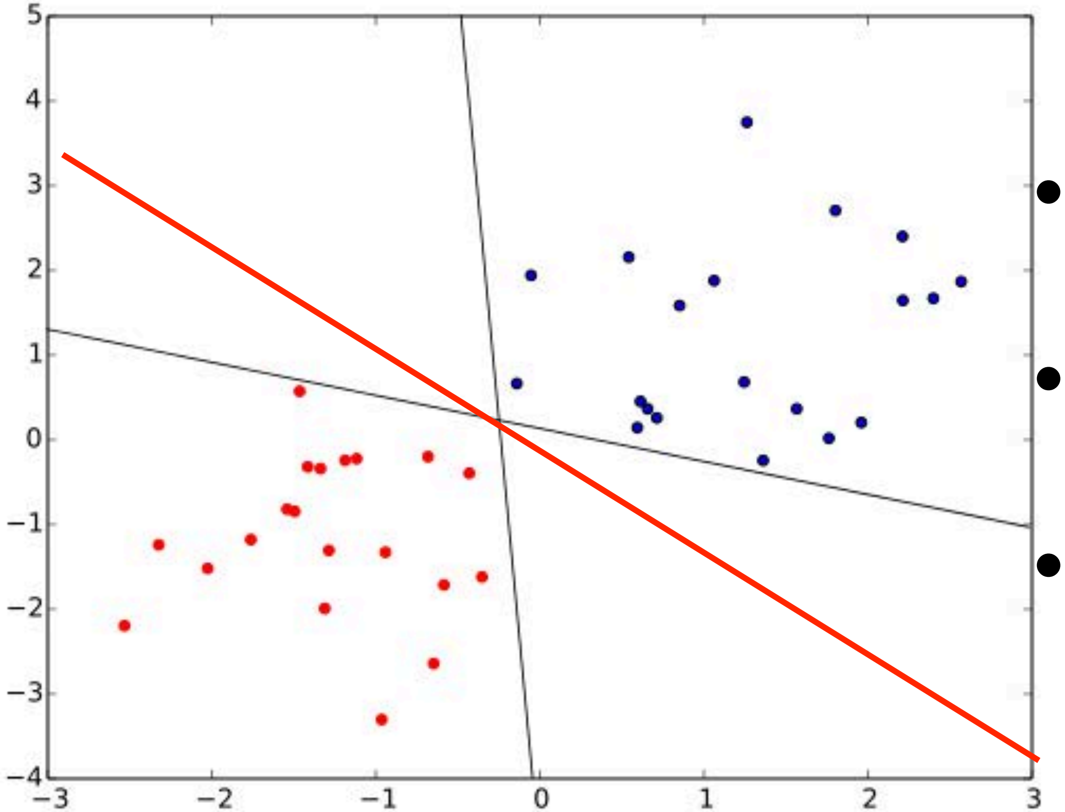
$$y(\mathbf{x}; \tilde{\mathbf{w}}) = \sigma(\tilde{\mathbf{w}} \cdot \tilde{\mathbf{x}})$$
$$= \frac{1}{1 + \exp(-\tilde{\mathbf{w}} \cdot \tilde{\mathbf{x}})}$$

- $0 \leq y(\mathbf{x}; \mathbf{w}) \leq 1$
- $y(\mathbf{x}; \mathbf{w}) = 0.5$ if $\tilde{\mathbf{w}} \cdot \tilde{\mathbf{x}} = 0$, i.e. $\mathbf{x}$ is on the decision boundary.
- $y(\mathbf{x}; \mathbf{w}) = 0.0$ or 1.0 if $\mathbf{x}$ far from the decision boundary.

$\Rightarrow$ $y(\mathbf{x}; \tilde{\mathbf{w}})$ can be interpreted as the probability that x belongs to one class or the other.

Logistic regression finds what is called the **maximum likelihood solution** under the assumption that the noise is Gaussian.

Perceptron vs Logistic Regression



- Two different solutions among infinitely many.
- The perceptron has no way to favor one over the other.
- Logistic regression does.

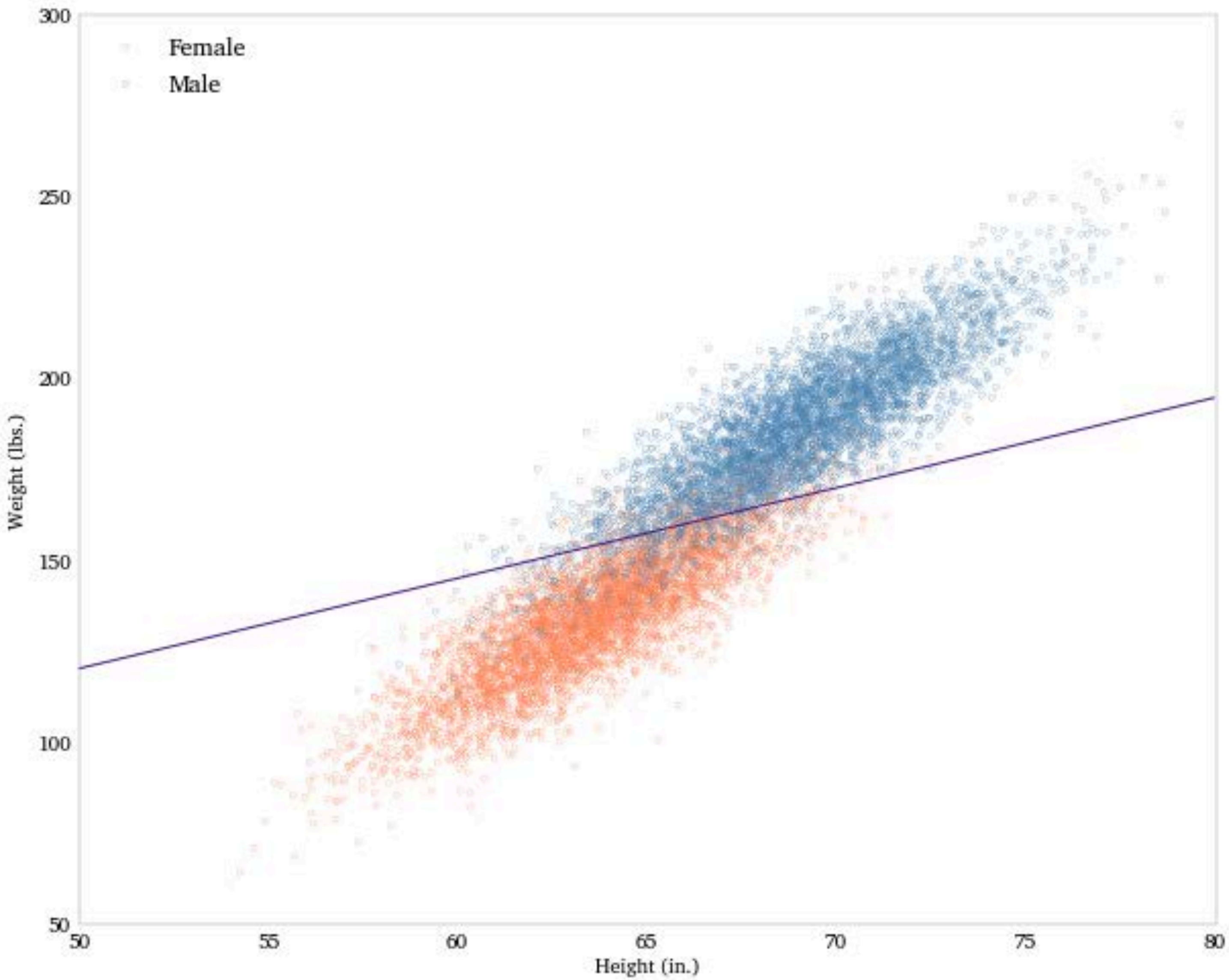
$$E(\tilde{\mathbf{w}}) = -\sum_{n=1}^{N} \operatorname{sign}(\tilde{\mathbf{w}} \cdot \tilde{\mathbf{x}}_n) t_n$$

vs

$$E(\tilde{\mathbf{w}}) = -\sum \{t_n \ln y_n + (1 - t_n) \ln(1 - y_n)\}$$

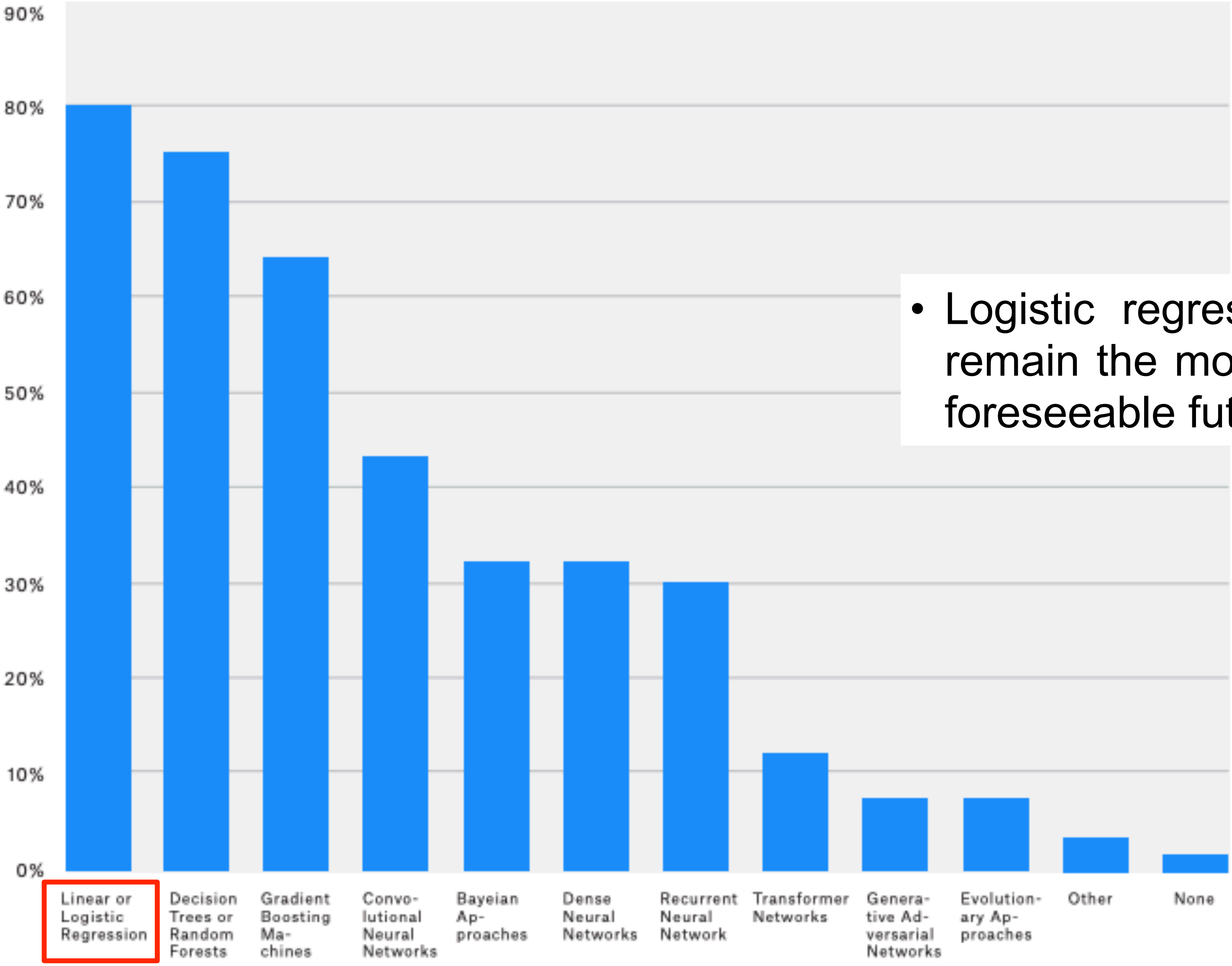
EPFL

Example



- The algorithm does the best it can.
- Some samples can be misclassified.

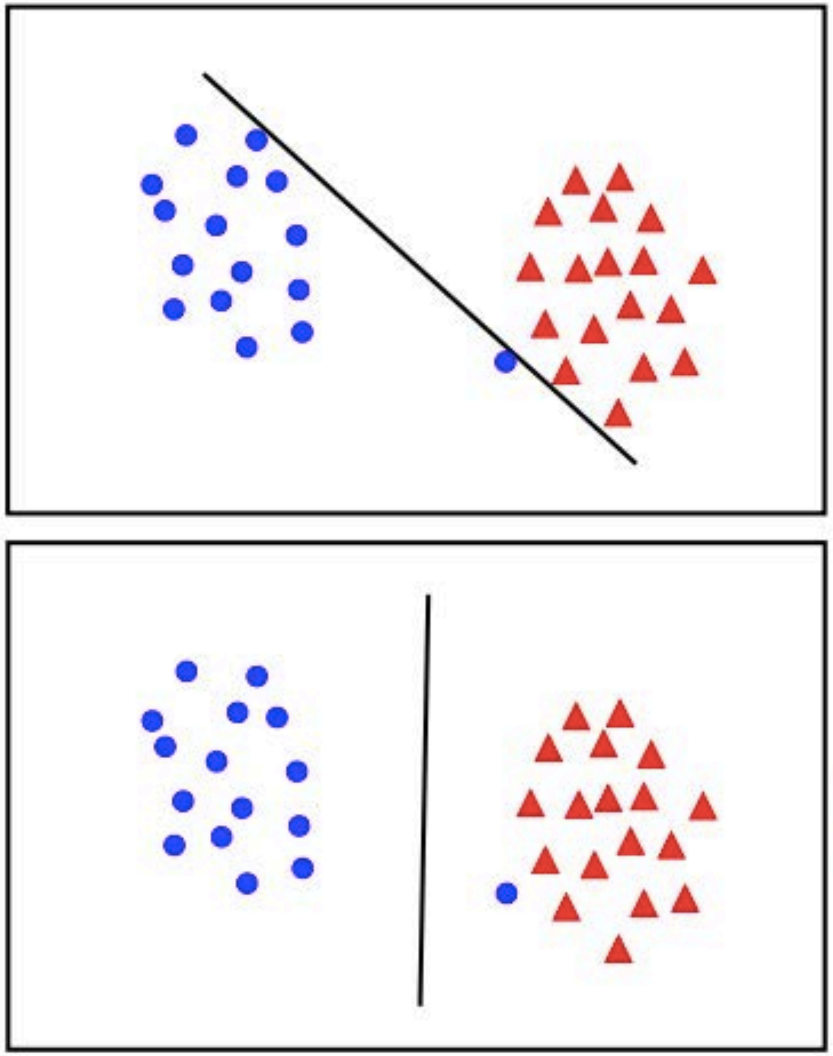
Kaggle Survey (2019)



- Logistic regression is and is likely to remain the most used technique for the foreseeable future.

What data science methods do you use at work?

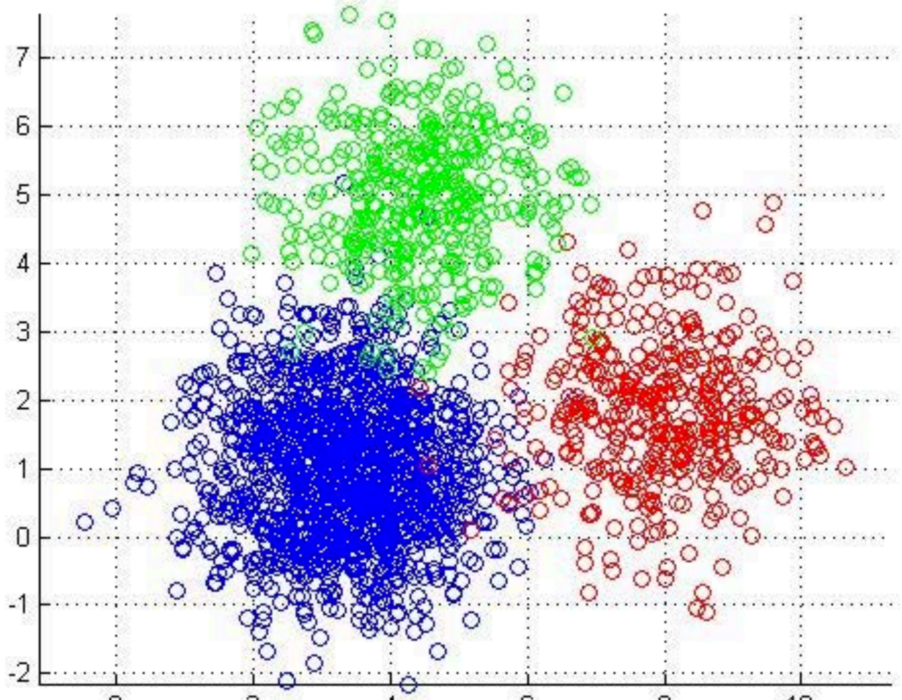
Outliers Can Cause Problems

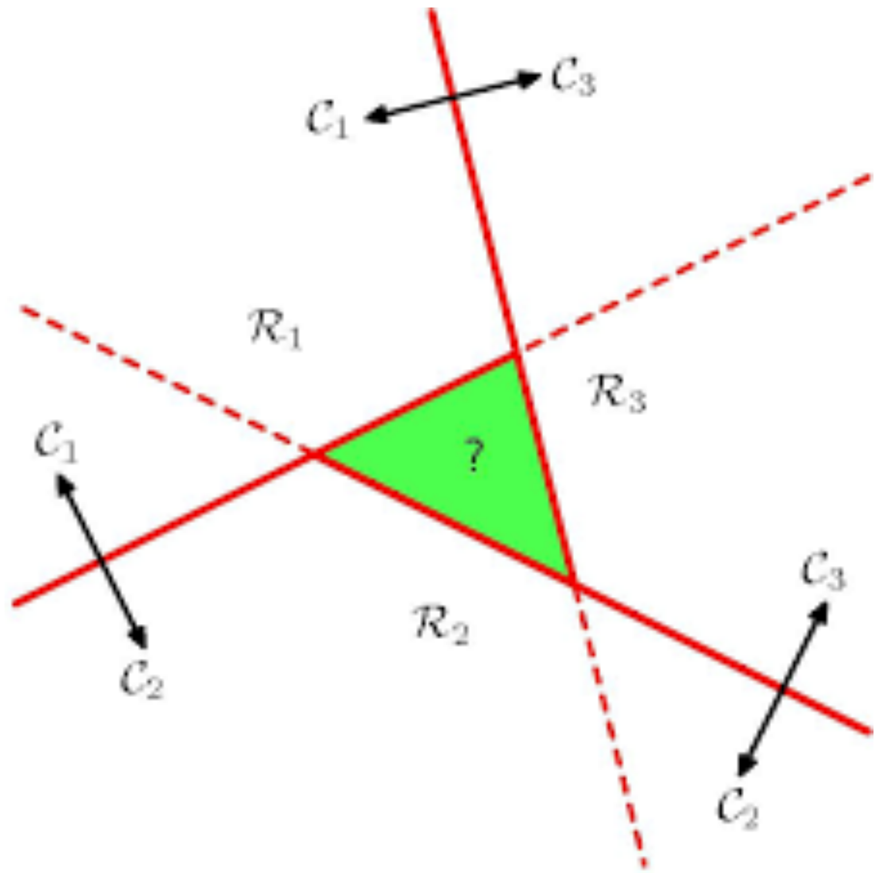


- Logistic regression tries to minimize the error-rate at training time.
- Can result in poor classification rates at test time.

—> We will talk about ways to prevent this in the next lecture.

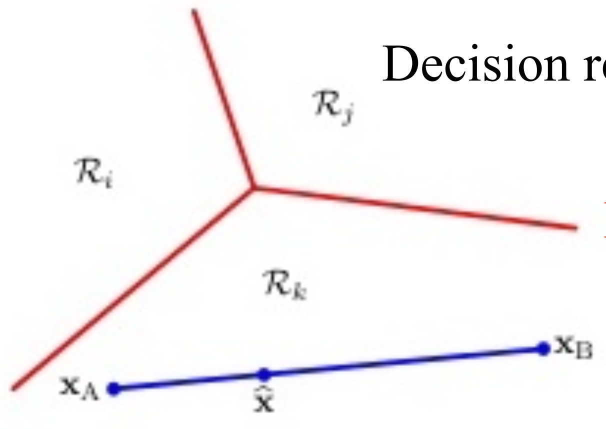
From Binary to Multi-Class





- k classes.
- Simply using k (k-1)/2 binary classifiers results in ambiguities.

Linear Discriminant



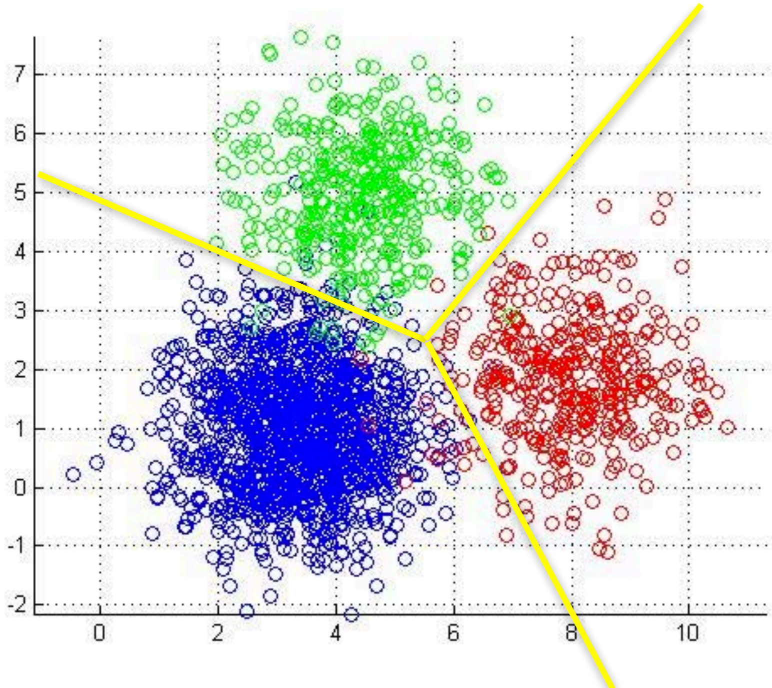
Given K linear classifiers of the form $y_k(\mathbf{x}) = \tilde{\mathbf{w}}_k \cdot \tilde{\mathbf{x}}$:

- Decision boundaries $y_k(\mathbf{x}) = y_l(\mathbf{x}) \Leftrightarrow (\tilde{\mathbf{w}}_\mathbf{k} - \tilde{\mathbf{w}}_\mathbf{l}) \cdot \tilde{\mathbf{x}} = 0.$
- These boundaries define decision regions.
- Decision regions are convex:

$$(\tilde{\mathbf{w}}_k - \tilde{\mathbf{w}}_l) \cdot \tilde{\mathbf{x}}_A > 0$$
$$(\tilde{\mathbf{w}}_k - \tilde{\mathbf{w}}_l) \cdot \tilde{\mathbf{x}}_B > 0$$
$$\Rightarrow \forall \lambda \in [0,1], \text{ if } \mathbf{x} = \lambda \mathbf{x}_A + (1-\lambda)\mathbf{x}_B, \text{ then}$$
$$(\tilde{\mathbf{w}}_k - \tilde{\mathbf{w}}_l) \cdot \tilde{\mathbf{x}} > 0$$

In other words, if two points are on the same side of a decision boundary so are all point between them.

Multi-Class Logistic Regression



- K linear classifiers of the form $y^k(\mathbf{x}) = \sigma(\mathbf{w}_k^T \mathbf{x})$.
- Assign x to class k if $y^k(\mathbf{x}) > y^l(\mathbf{x}) \forall l \neq k$.

$$k = \arg\max_j y_k(\mathbf{x})$$

- Still a linear problem.
- Because the sigmoid function is monotonic, the formulation is almost unchanged.
- Only the objective function being minimized need to be reformulated.

$$\begin{bmatrix} y_1 \\ \vdots \\ y_K \end{bmatrix} = \begin{bmatrix} \tilde{\mathbf{w}}_1^T \\ \vdots \\ \tilde{\mathbf{w}}_K^T \end{bmatrix} \tilde{\mathbf{x}}$$

$$k = \arg\max_j y_j$$

Matrix with K lines and the dimension of $\tilde{\mathbf{w}}$ columns.

Multi-Class Cross Entropy

Let the training set be $\{(\mathbf{x}_n, [t_n^1, \ldots, t_n^K])_{1 \leq n \leq N}\}$ where $t_n^k \in \{0,1\}$ is the probability that sample $\mathbf{x}_n$ belongs to class k.

Activation: $$a^k(\mathbf{x}) = \tilde{\mathbf{w}}_k^T \tilde{\mathbf{x}}$$

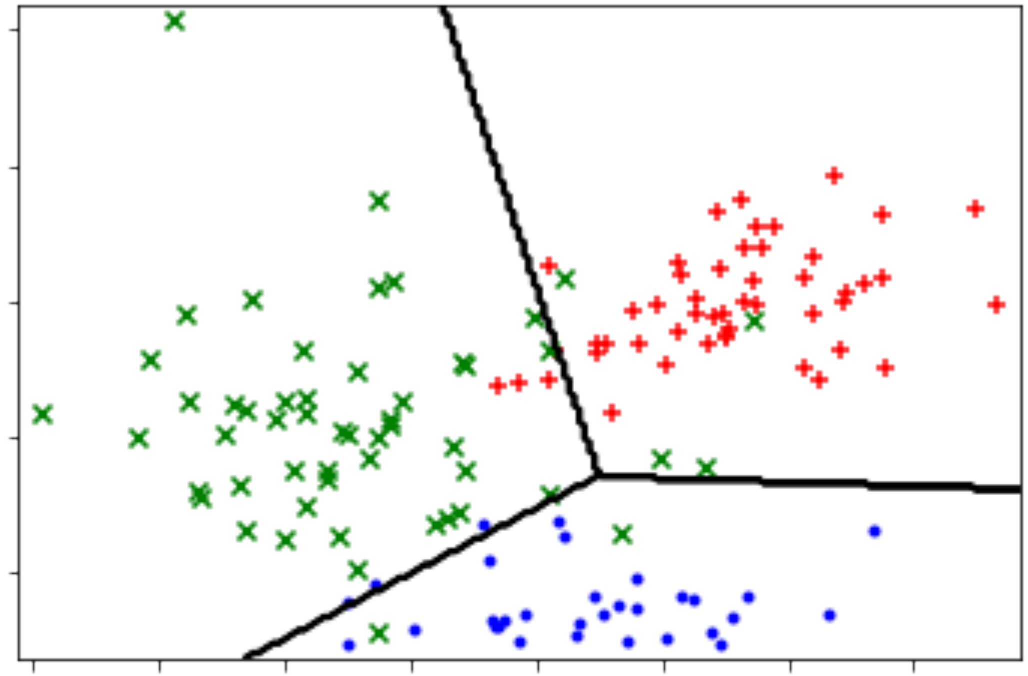
Probability that $\mathbf{x}$ belongs to class k: $$y^k(\mathbf{x}) = \frac{\exp(a^k(\mathbf{x}))}{\sum_j \exp(a^j(\mathbf{x}))}$$

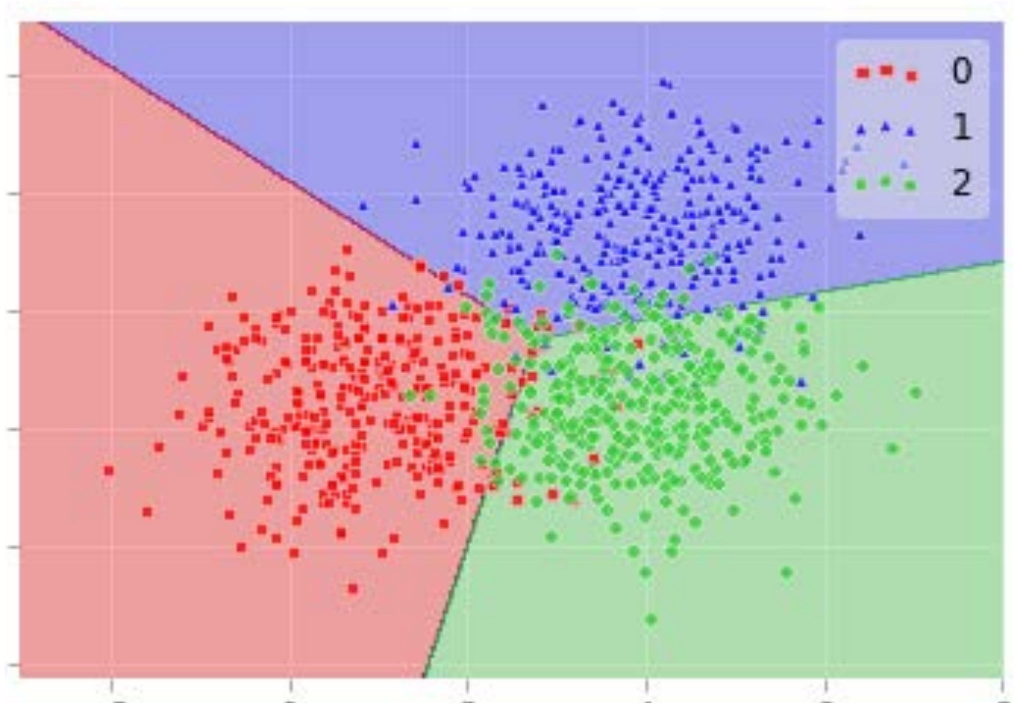
Multi-class entropy: $$E(\tilde{\mathbf{w}}_1, \ldots, \tilde{\mathbf{w}}_K) = -\sum_n \sum_k t_n^k \ln(y^k(\mathbf{x}_n))$$

Gradient of the entropy: $$\nabla E_{\mathbf{w_j}} = \sum_n (y^k(\mathbf{x}_n) - t_n^k)\mathbf{x}_n$$

- This is a natural extension of the binary case.
- The multi-class entropy is still convex and its gradient is easy to compute.

Multi-Class Results





Multiclass logistic regression is a very natural extension of binary logistic regression and has many of the same properties.