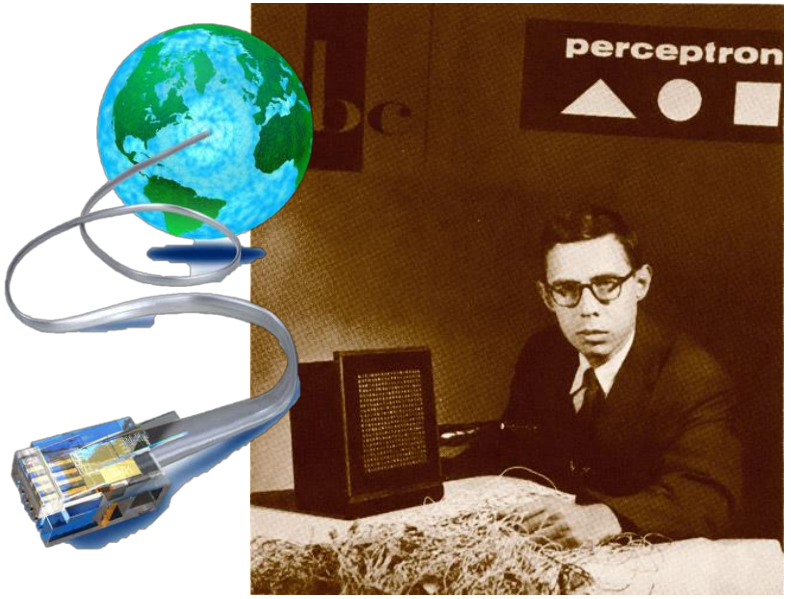
Feature spaces and kernels

Homogeneous linear classifiers

- Homogeneous linear classifier: $w \in \mathbb{R}^d$ (*weight vector*)

$$f_w(x) = f_{w,0}(x) = \begin{cases} +1, & \langle x, w \rangle > 0 \\ -1, & \langle x, w \rangle \leq 0 \end{cases}$$

Online Perceptron



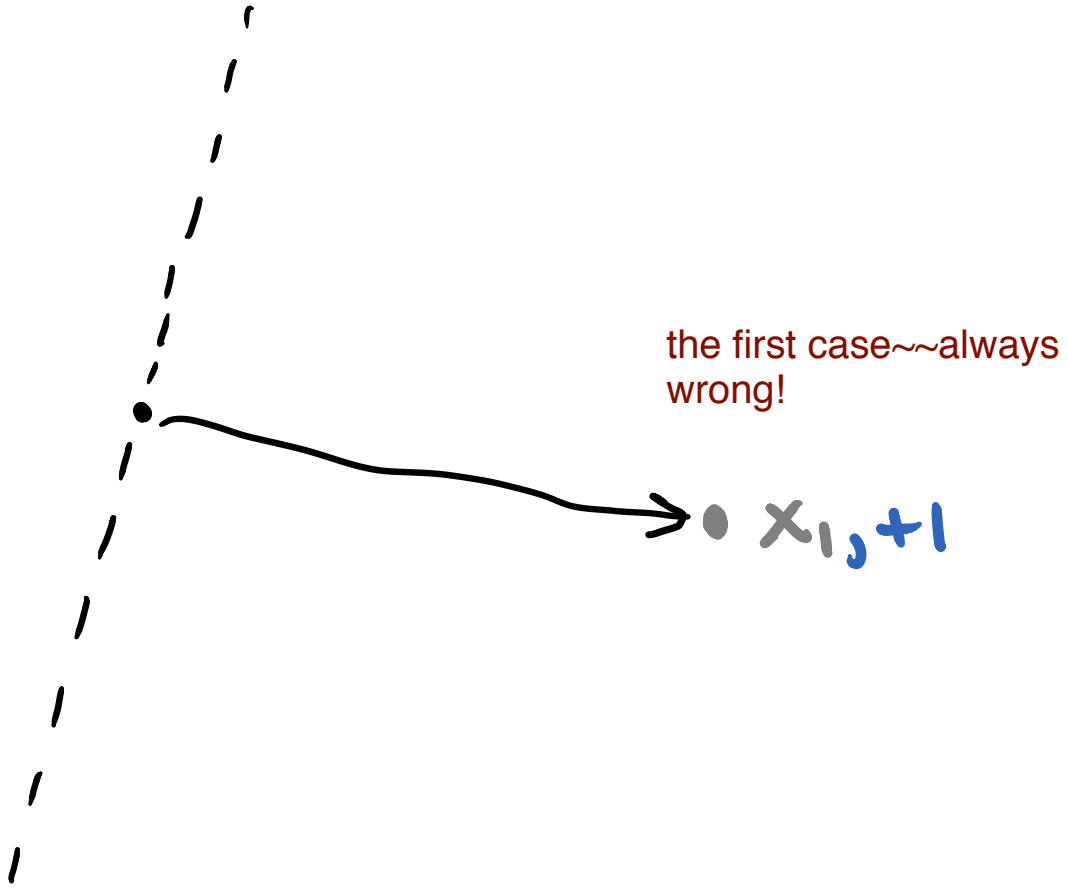
Input: training data S as an *input stream*.

- **Let** $w = \vec{0}$.
- **For** each $(x, y) \in S$:
  - **If** $f_w(x) \neq y$, **then**:
    - **Update**: $w := w + yx$
- **Return** w

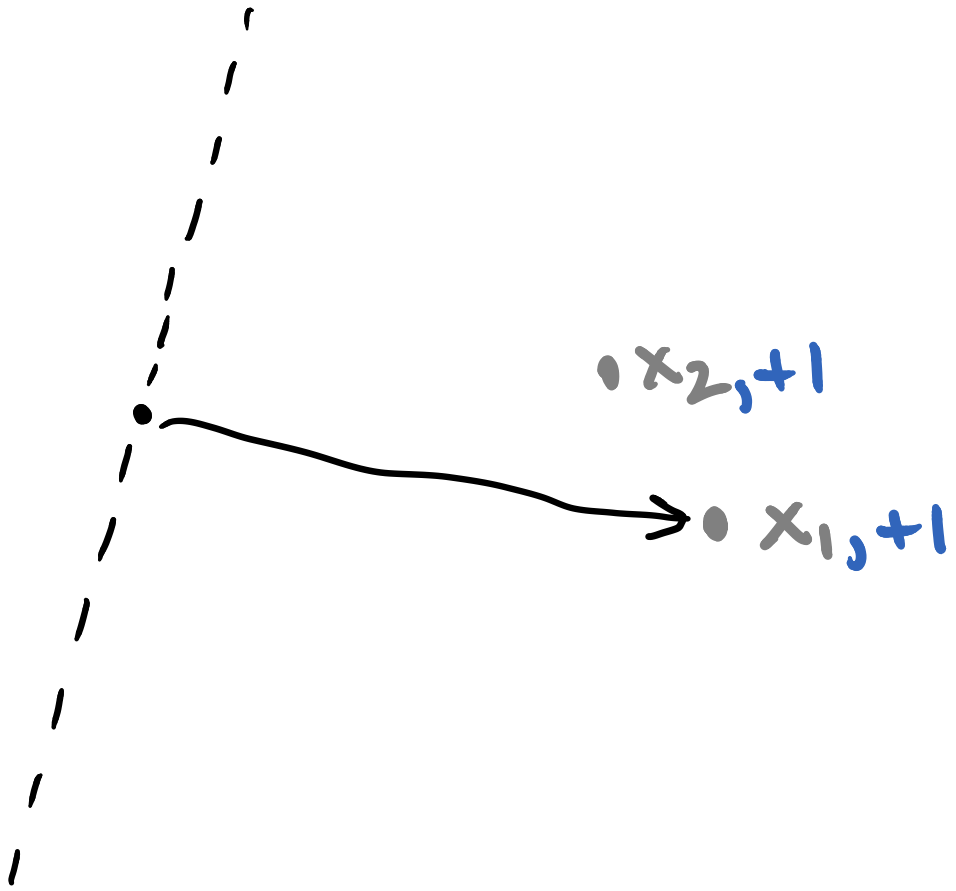
If S is separable with margin $\gamma > 0$, and $R := \max_{(x,y)\in S} \|x\|$,

then Online Perceptron makes at most $\left(\frac{R}{\gamma}\right)^2$ mistakes (and updates).

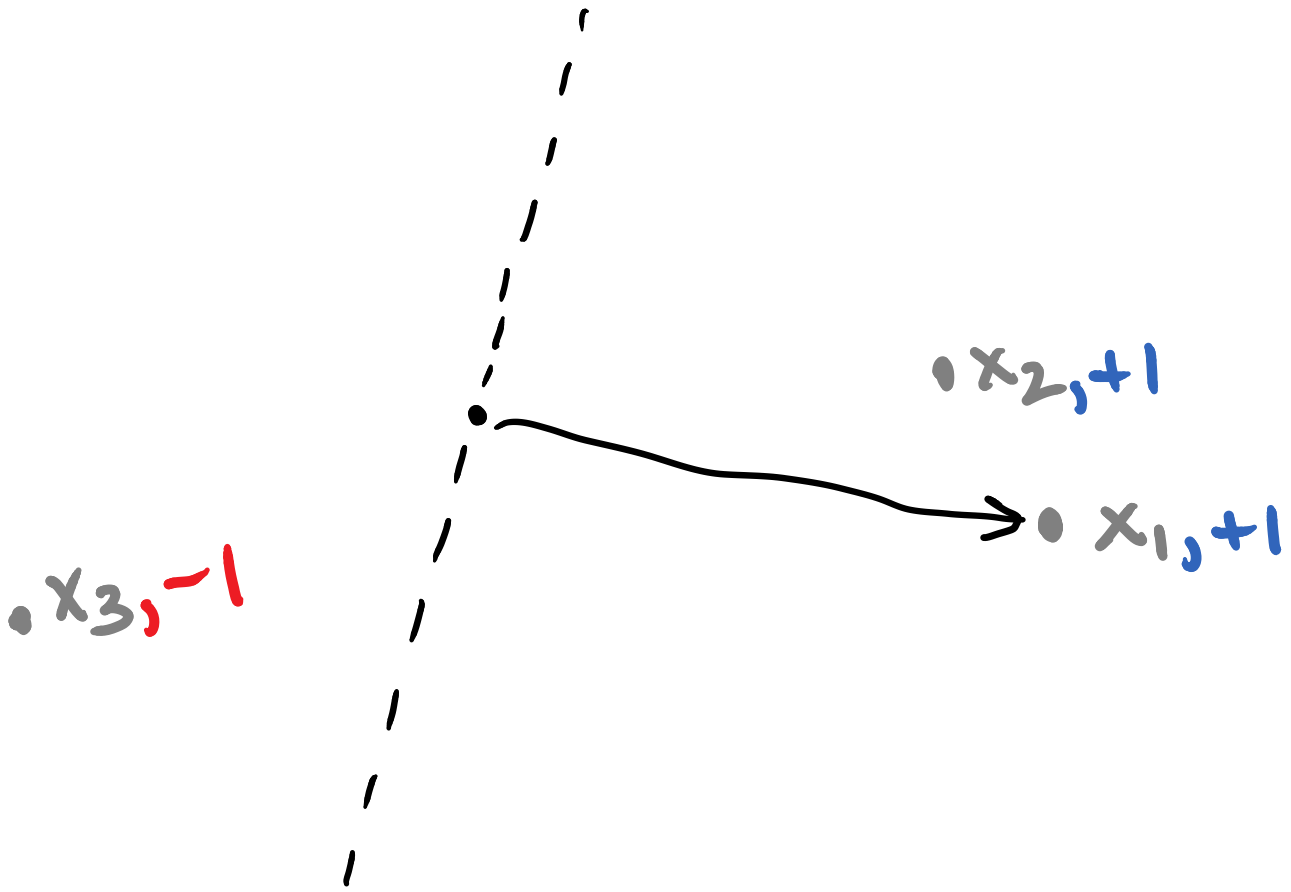
Example run of Online Perceptron



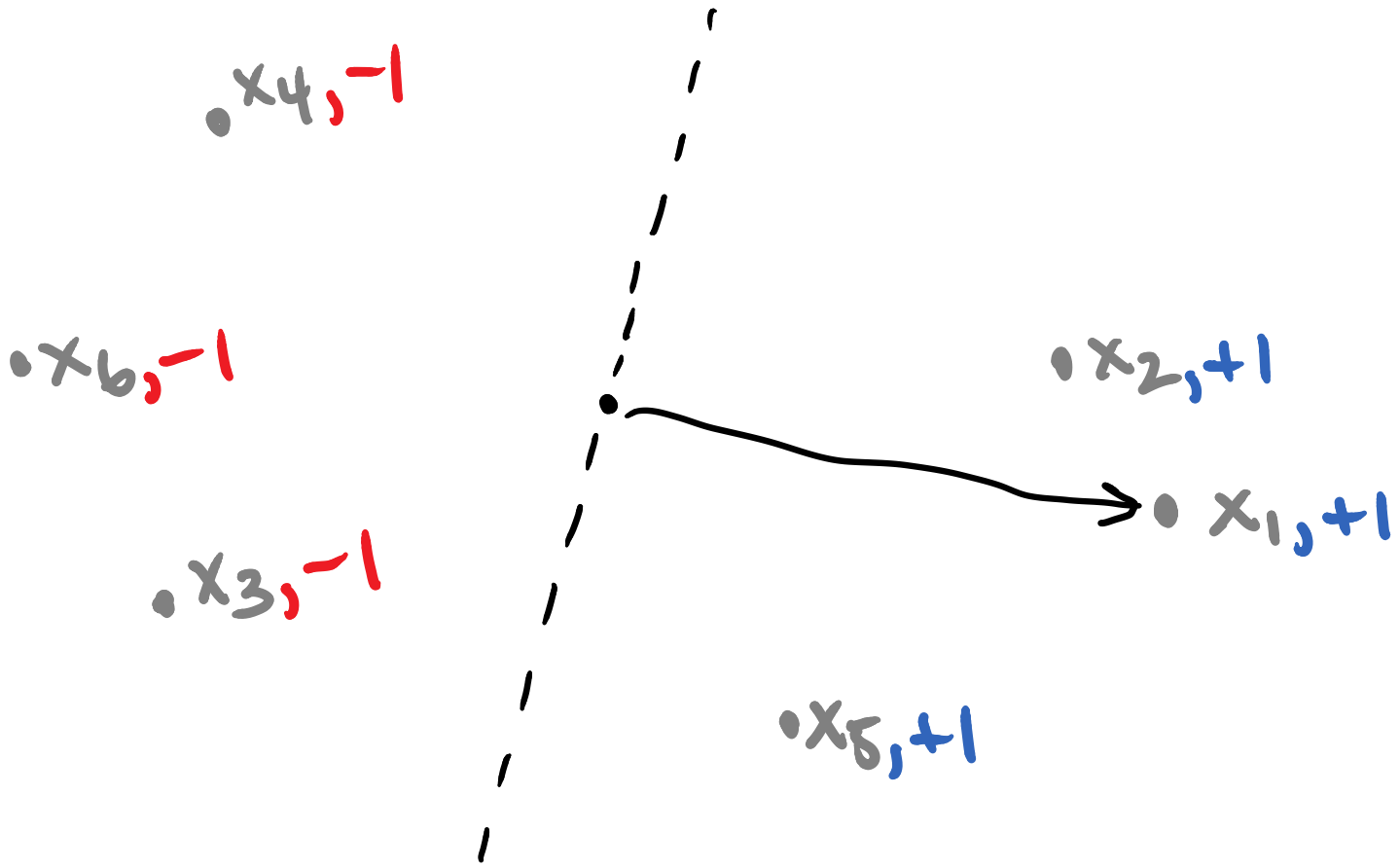
Example run of Online Perceptron



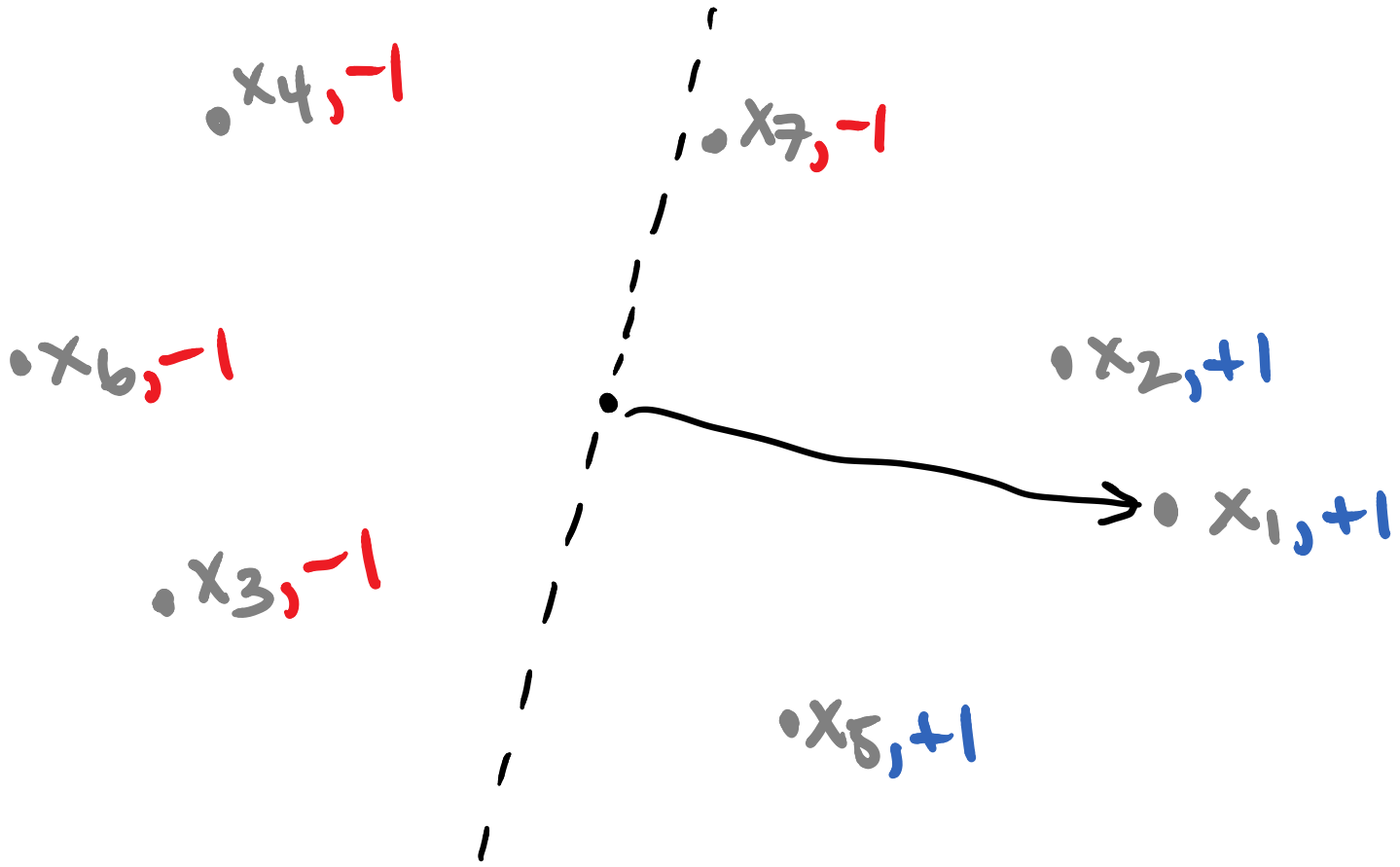
Example run of Online Perceptron



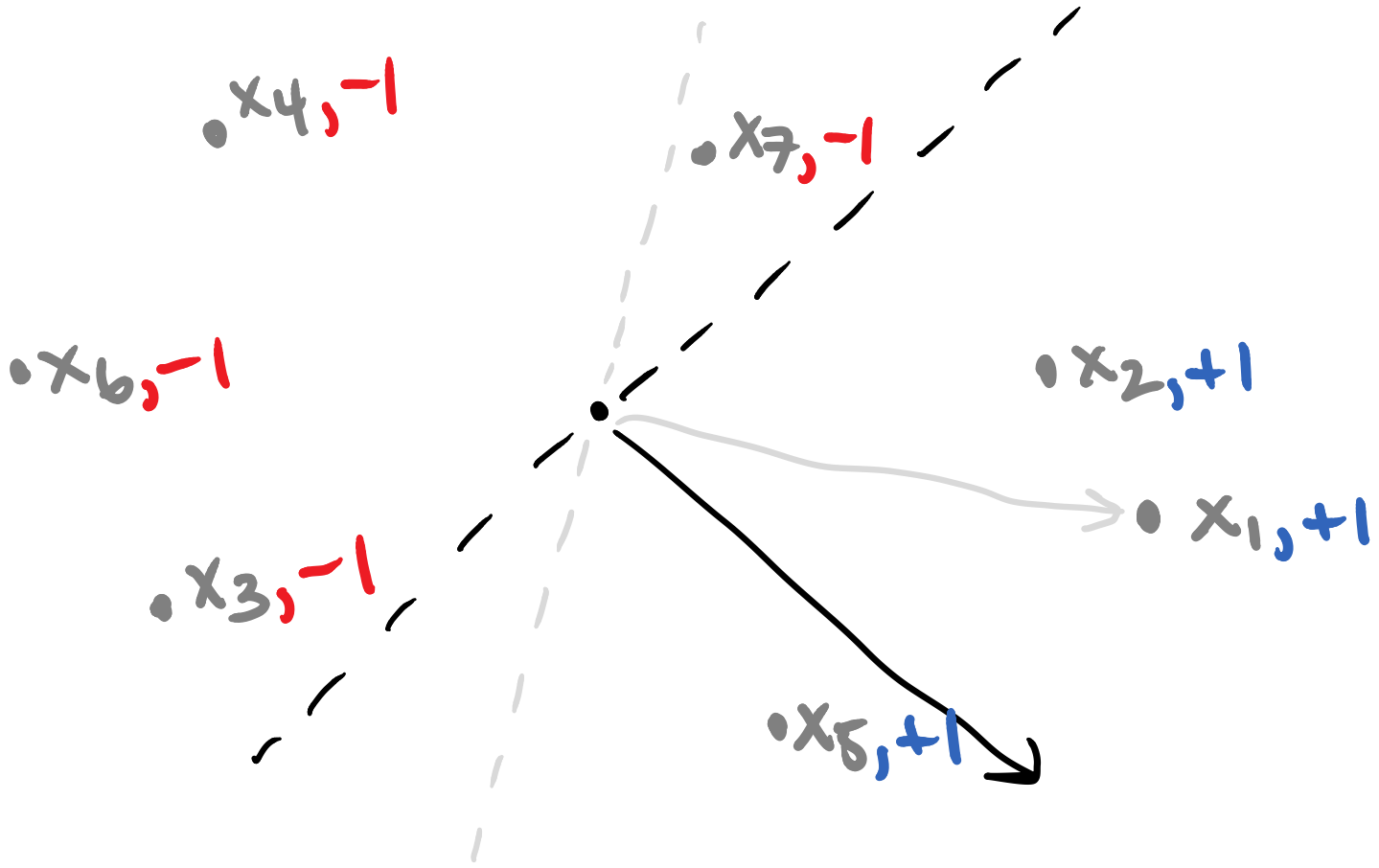
Example run of Online Perceptron



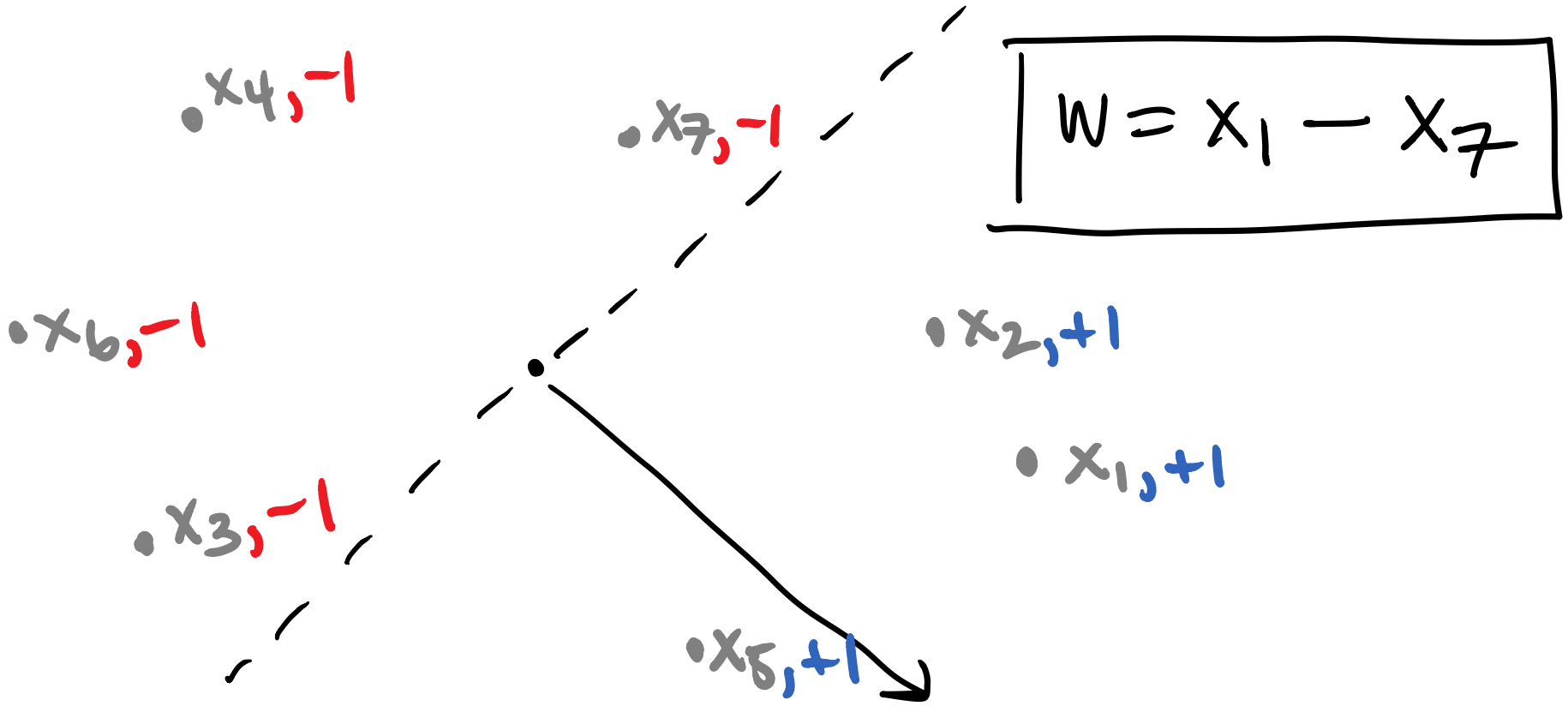
Example run of Online Perceptron



Example run of Online Perceptron



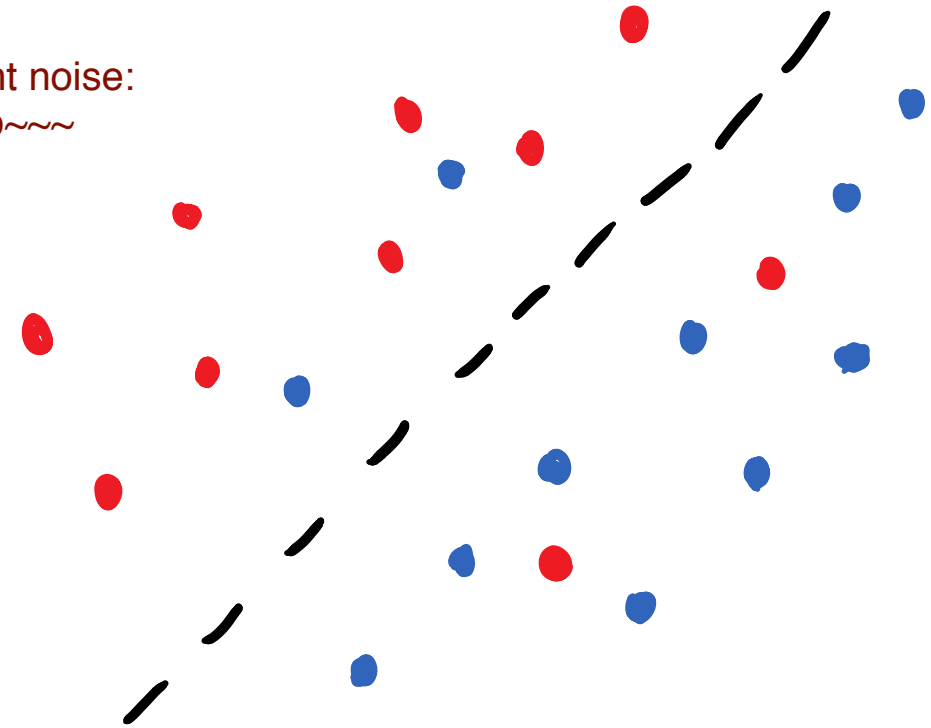
Example run of Online Perceptron



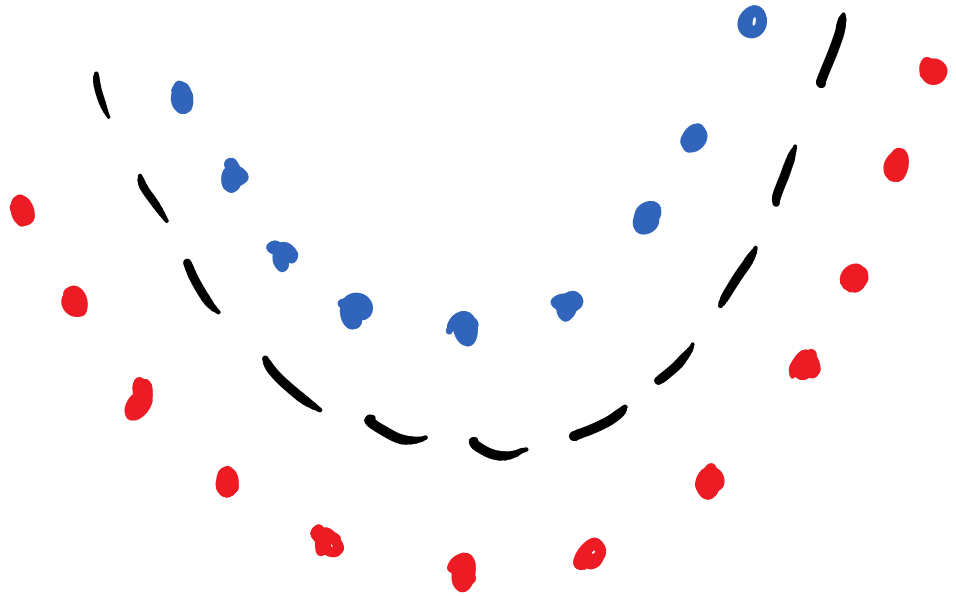
What if data is not linearly separable?

Noise: even Bayes classifier not perfect

inherent noise:
overlap~~~



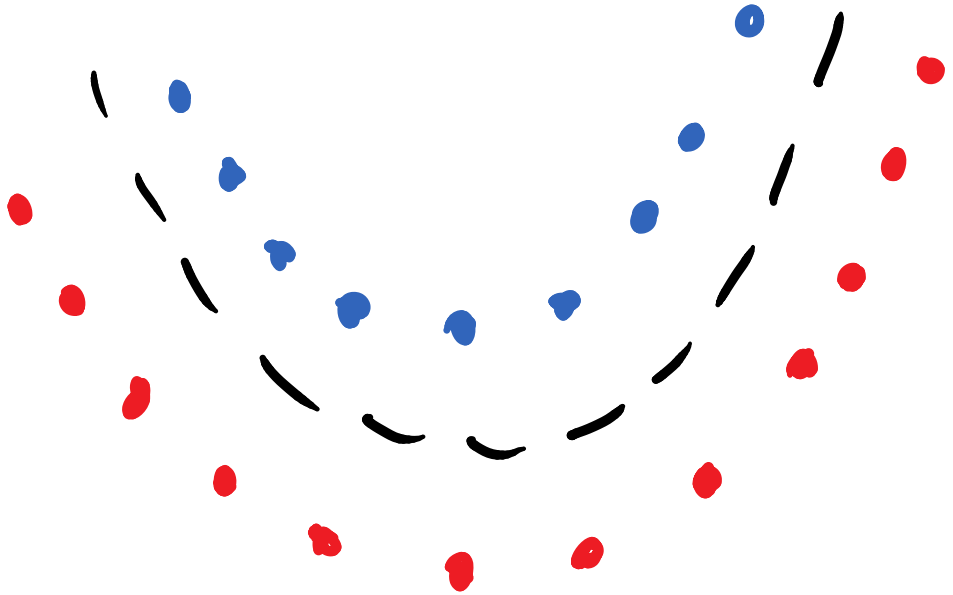
Bayes classifier not (approx.) linear



Adding new features

Original feature vector: $x = (1, x_1, x_2)$

New feature vector: $\phi(x) = \left(1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, \sqrt{2}x_1x_2, x_2^2\right)$



Decision boundary non-linear in x, but is linear in $\phi(x)$.

Getting the most out of linear classifiers

Often, with the “correct” set of features, a linear classifier can very well-approximate the Bayes classifier.

Two approaches:

1. Think very hard and carefully about which features to use.
2. Use all features you can think of.

The kitchen sink of features

- **Example**: document classification
  - **Word features**:

    $$1\{\text{aardvark} \in \text{doc}\}, 1\{\text{abacus} \in \text{doc}\}, \dots, 1\{\text{zygote} \in \text{doc}\}$$

  - **Bi-gram features**:

    $$1\{\text{bank deposit} \in \text{doc}\}, 1\{\text{river bank} \in \text{doc}\}, \dots$$

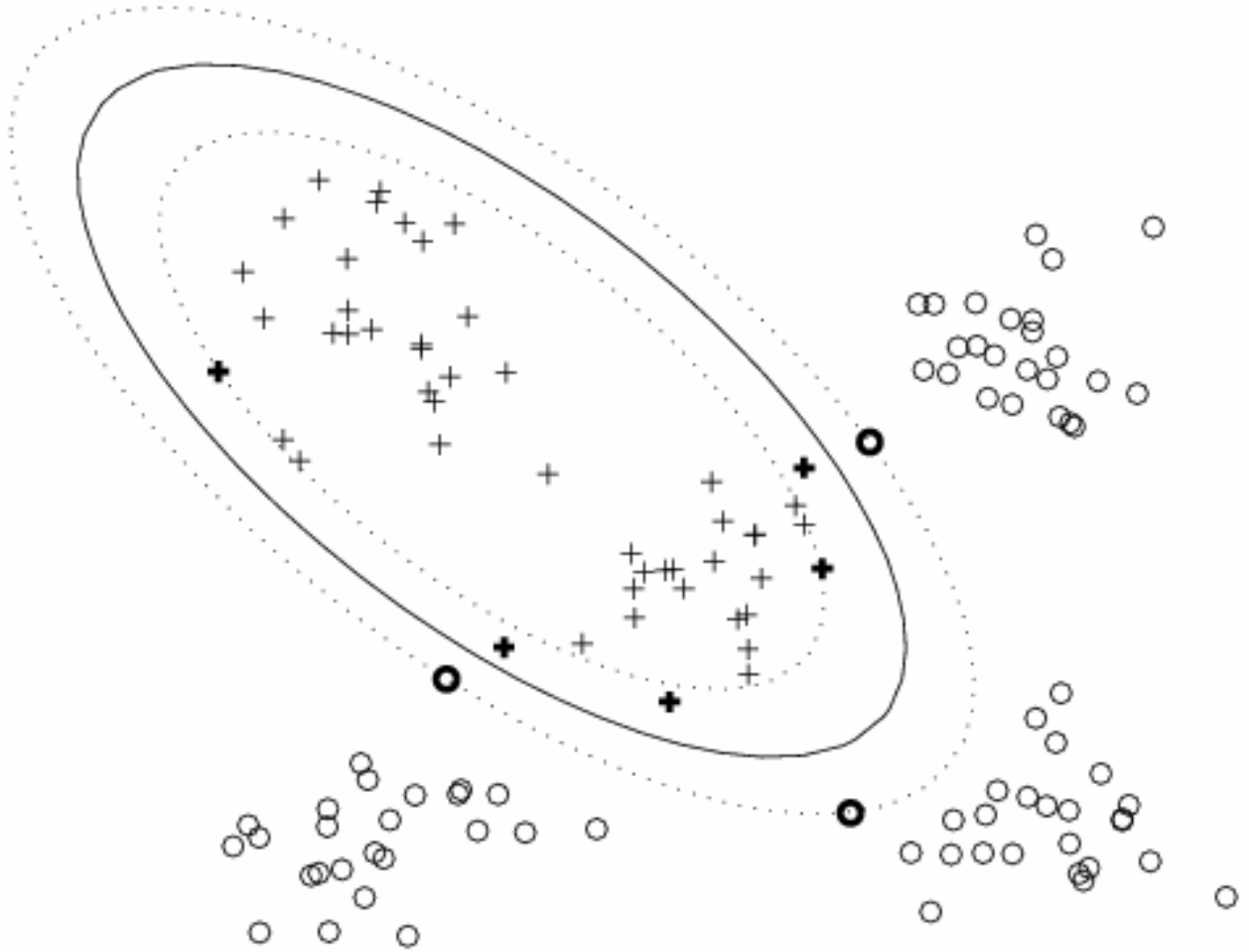
  - **Tri-gram features**:

    $$1\{\text{new york city} \in \text{doc}\}, 1\{\text{wherefore art thou} \in \text{doc}\}, \dots$$

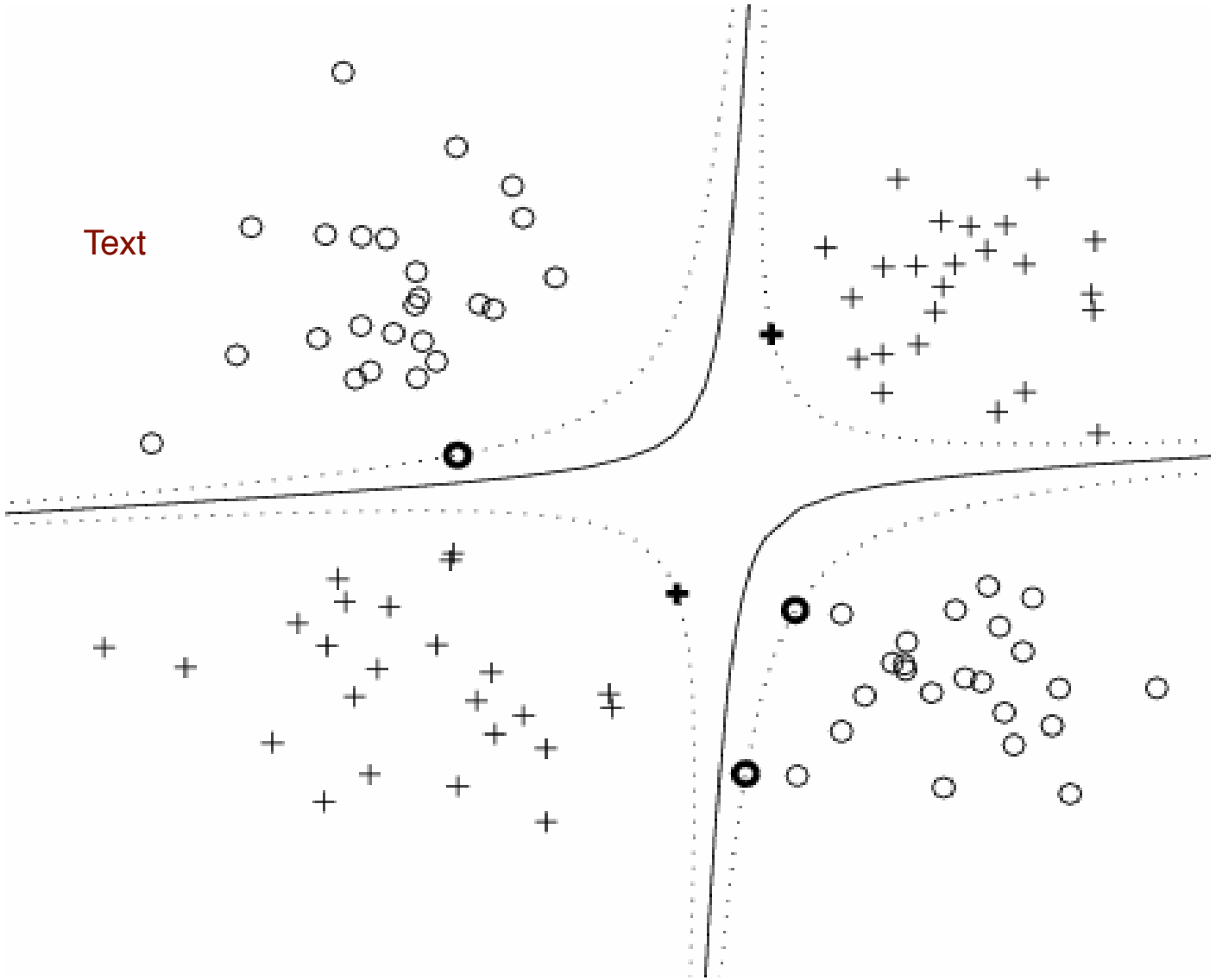
- **Example**: new features from old features $x \in \mathbb{R}^d$
  - Pairwise interactions:

    $$(x_1x_2, x_1x_3, \dots, x_1x_d, x_2x_3, \dots, x_{d-1}x_d) \in \mathbb{R}^{\binom{d}{2}}$$

All degree ≤ 2 interaction features



All degree ≤ 2 interaction features



Learning with the kitchen sink of features

- Let $\phi: \mathbb{R}^d \to \mathbb{R}^D$ be the expanded feature mapping
  (e.g., throw in all quadratic interaction features, so $D = \Omega(d^2)$)
  so $\phi(x) \in \mathbb{R}^D$ is the feature vector we actually want to use

features mapping

- Learn linear classifier $f_w: \mathbb{R}^D \to \{\pm 1\}$ (i.e., weight vector $w \in \mathbb{R}^D$)
  using data with expanded features: $\big((\phi(x_1), y_1), (\phi(x_2), y_2), \dots\big)$

- Can be computationally expensive to do this directly when D is large
  (naively: $\Omega(D)$ time to make a prediction)

The kernel trick

- **Perceptron weight vector (using expanded features)**:

$$w = \sum_{(x,y)\in\mathcal{M}} y\phi(x)$$

  where $\mathcal{M} \subseteq S$ are examples where Online Perceptron makes update.

- **Perceptron prediction**: on new point x', note the transformation

$$\langle \phi(x'), w \rangle = \sum_{(x,y)\in\mathcal{M}} y\langle \phi(x), \phi(x') \rangle$$

Text

$|M| \times$ [time to compute inner product $\langle \phi(x), \phi(x') \rangle$]

All degree ≤ 2 interaction features

$$\phi: \mathbb{R}^d \to \mathbb{R}^{1+2d+\binom{d}{2}}$$

$$\phi(x) := (1, \sqrt{2}x_1, \sqrt{2}x_2, \dots, \sqrt{2}x_d, x_1^2, x_2^2, \dots, x_d^2,$$
$$\sqrt{2}x_1x_2, \sqrt{2}x_1x_3, \dots, \sqrt{2}x_1x_d, \sqrt{2}x_2x_3, \dots, \sqrt{2}x_{d-1}x_d)$$

(Don't mind the $\sqrt{2}$'s)

Computation of $\langle \phi(x), \phi(x') \rangle$ in $O(d)$ time:

$$(1 + \langle x, x' \rangle)^2 = \langle \phi(x), \phi(x') \rangle$$

Products of all feature subsets

$$\phi: \mathbb{R}^d \to \mathbb{R}^{2^d}$$

$$\phi(x) := \left(\prod_{i \in S} x_i : S \subseteq [d] \right)$$

features' subset~~~
Degree

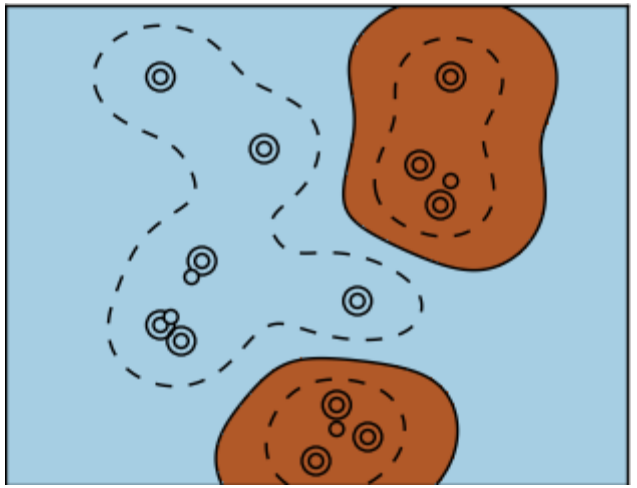
Computation of $\langle \phi(x), \phi(x') \rangle$ in $O(d)$ time:

$$\prod_{i=1}^{d} (1 + x_i x_i') = \sum_{S \subseteq [d]} \prod_{i \in S} (x_i x_i') = \langle \phi(x), \phi(x') \rangle$$

An infinite dimensional feature expansion

For any $\sigma > 0$, there is an infinite feature expansion $\phi: \mathbb{R}^d \to \mathbb{R}^\infty$ (involving Hermite polynomials of all orders) such that

$$\langle \phi(x), \phi(x') \rangle = \exp\left(-\frac{\|x - x'\|^2}{2\sigma^2}\right)$$



This can be computed in $O(d)$ time.

This inner product is called the *Gaussian kernel with bandwidth* σ.

Kernels

A (positive definite) **kernel function** $K: \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is a symmetric function with the following property:

For any $x_1, x_2, \ldots, x_n \in \mathbb{R}^d$, the $n \times n$ matrix whose (i, j)-th entry is $K(x_i, x_j)$, is **positive-semidefinite** (all of its eigenvalues are ≥ 0).

For any kernel K, there is a feature mapping $\phi: \mathbb{R}^d \to \mathbb{H}$ such that

$$\langle \phi(x), \phi(x') \rangle = K(x, x').$$

($\mathbb{H}$ is a Hilbert space [a special kind of inner product space], called the Reproducing Kernel Hilbert Space corresponding to the kernel K.)

String kernels

$$\phi: \text{Strings} \to \mathbb{N}^{\text{Strings}}$$
$$\phi(x) = (\text{number of times } s \text{ appears in } x : s \in \text{Strings})$$

$$K(x, x') = \langle \phi(x), \phi(x') \rangle = \text{measure of similarity between strings}$$

For each substring s in x:

Count how often s appears in x' and add to total.

Dynamic programming: $O\big(\text{length}(x) \cdot \text{length}(x')\big)$ time

The kernel approach

- Focus on designing good kernels (rather than feature maps), which means designing good **similarity functions**.
- Lots of ways to construct kernels (e.g., combine existing kernels).
- Lots of algorithms can be “kernelized” (whole industry around this).

Experimental results

- Using OCR digits data, binary classification problem of distinguishing "9" from other digits.
- # training examples: 60000 (about 6000 are from class "9").
- Using Kernelized Averaged Perceptron

# passes	0.1	1	2	3	4	10
Test error (linear)	0.045	0.039	0.038	0.038	0.038	0.037
Test error (degree 2)	0.024	0.012	0.010	0.010	0.009	0.009
Test error (degree 4)	0.020	0.009	0.008	0.007	0.007	0.006

Computational issues with Kernel methods

- Recall representation of Perceptron weight vector:

$$w = \sum_{(x,y)\in\mathcal{M}} y\phi(x) = \sum_{(x,y)\in\mathcal{M}} y\, K(\cdot, x)$$

- Number of mistakes $|\mathcal{M}|$ could be $\Omega(n)$!
  - Computing predictions as expensive as brute-force NN search.
  - Training can also be quite slow.

Kernel approximations

- Many ways to try to speed-up kernel methods using approximations.
- Some possibilities:
  - Limit number of examples used to represent weight vector.
    - “Nystrom approximation”
    - “Budgeted Perceptron”
  - Explicit feature maps z: $\mathbb{R}^d \rightarrow \mathbb{R}^m$ such that
    $$\langle z(x), z(x') \rangle \approx K(x, x')$$
    - “Random projections / feature hashing”
    - “Random kitchen sinks”

Experimental results

- Recall “Spam” data set (4601 e-mail messages, 39.4% are spam)
- $\mathcal{Y} = \{\text{spam}, \text{not spam}\}$
- $\mathcal{X} = \mathbb{R}^{57}$, features based on content of e-mail message
- # training examples: 3065, # test examples: 1536

- Decision tree learning (from lecture 4): **9.3%** test error
- Averaged Perceptron (128 passes): **8.27%**
- Random Kitchen Sink Averaged Perceptron (64 passes): **6.12%**

Recap and final remarks

- **Linear classifiers** only as good as the given feature representation
- **Sometimes explicit feature expansion is okay** (e.g., when x is sparse)
- **Kernel trick**: sometimes never need $\phi(x)$ directly, but only $\langle \phi(x), \phi(x') \rangle$, but can compute this quickly as $K(x, x')$
- **Kernel approach**: switch from designing good features to designing good kernels (similarity functions)
- **Computational issues**: sometimes alleviated with approximations