Text classification II

CE-324: Modern Information Retrieval

Sharif University of Technology

M. Soleymani

Spring 2020

Some slides have been adapted from: Profs. Manning, Nayak & Raghavan (CS-276, Stanford)

Outline

- Vector space classification
  - Rocchio
  - Linear classifiers
  - kNN

Features

- Supervised learning classifiers can use any sort of feature
  - URL, email address, punctuation, capitalization, dictionaries, network features

- In the simplest bag of words view of documents
  - We use **only** word features we use **all** of the words in the text (not a subset)

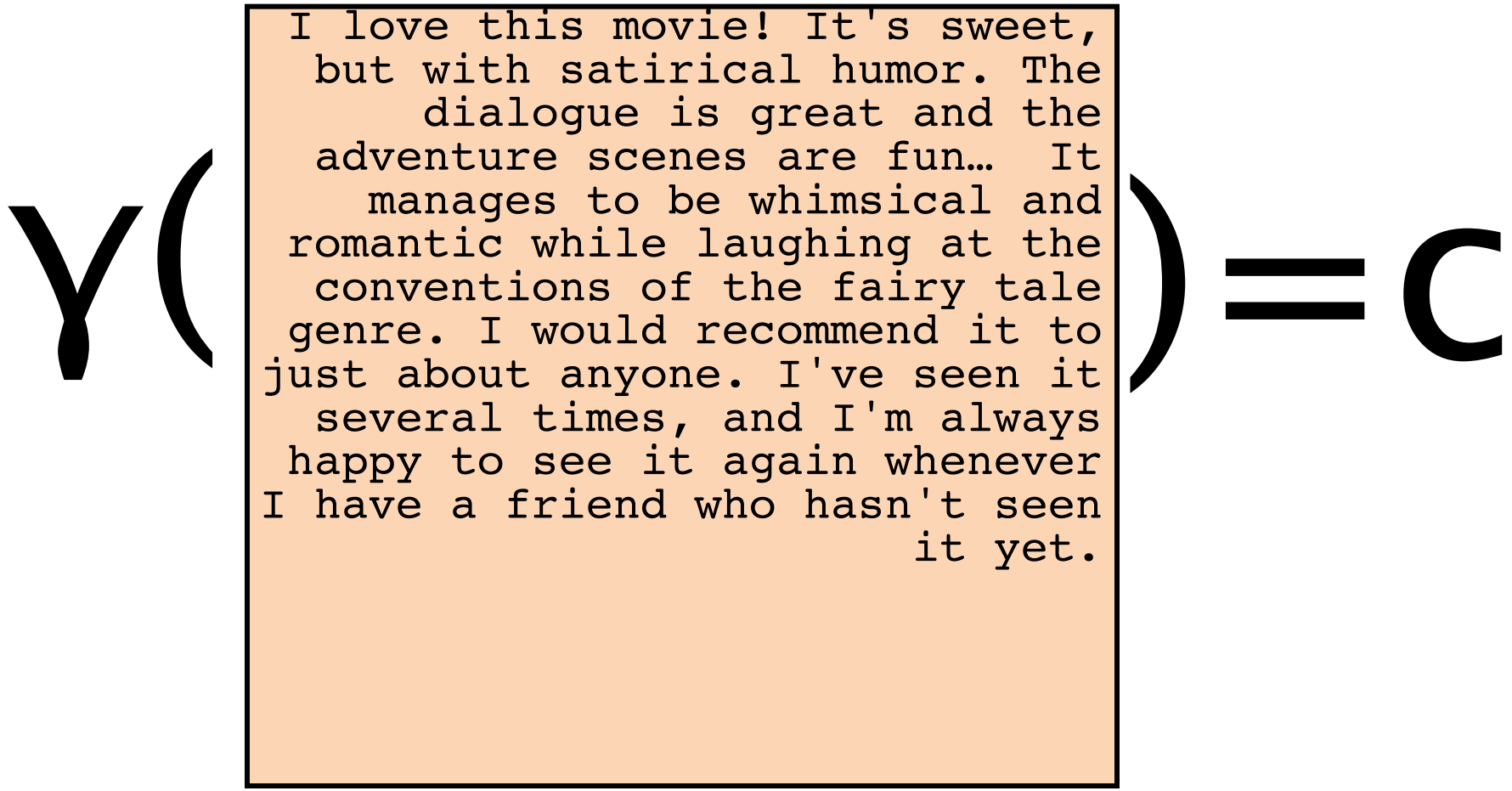
Recall: vector space representation

- Each doc is a vector
  - One component for each term (= word).
    - Terms are axes
  - Usually normalize vectors to unit length.

- High-dimensional vector space:
  - 10,000+ dimensions, or even 100,000+
  - Docs are vectors in this space

- How can we do classification in this space?

The bag of words representation



The bag of words representation

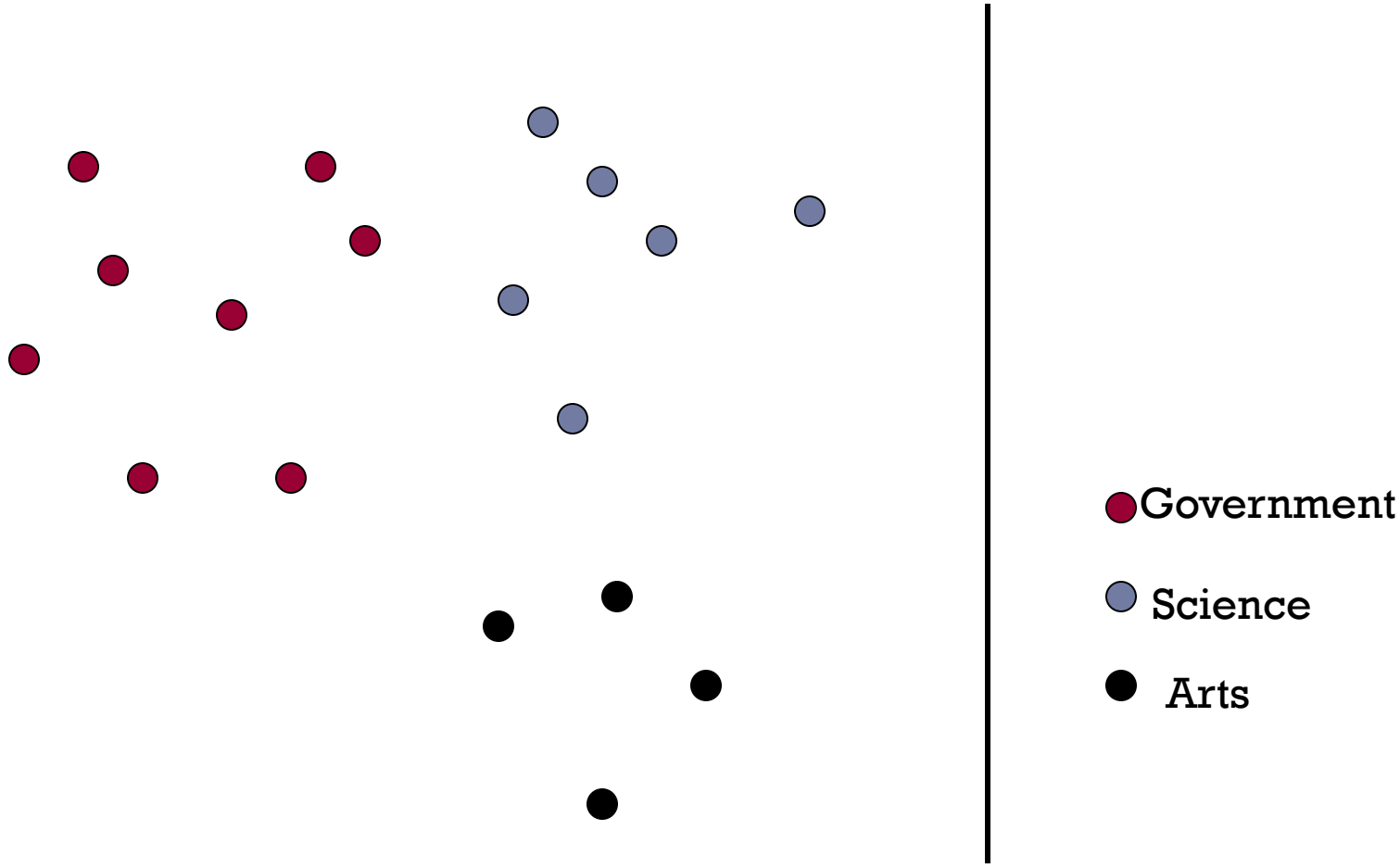
Classification using vector spaces

- Training set: a set of docs, each labeled with its class (e.g., topic)
  - This set corresponds to a labeled set of points (or, equivalently, vectors) in the vector space

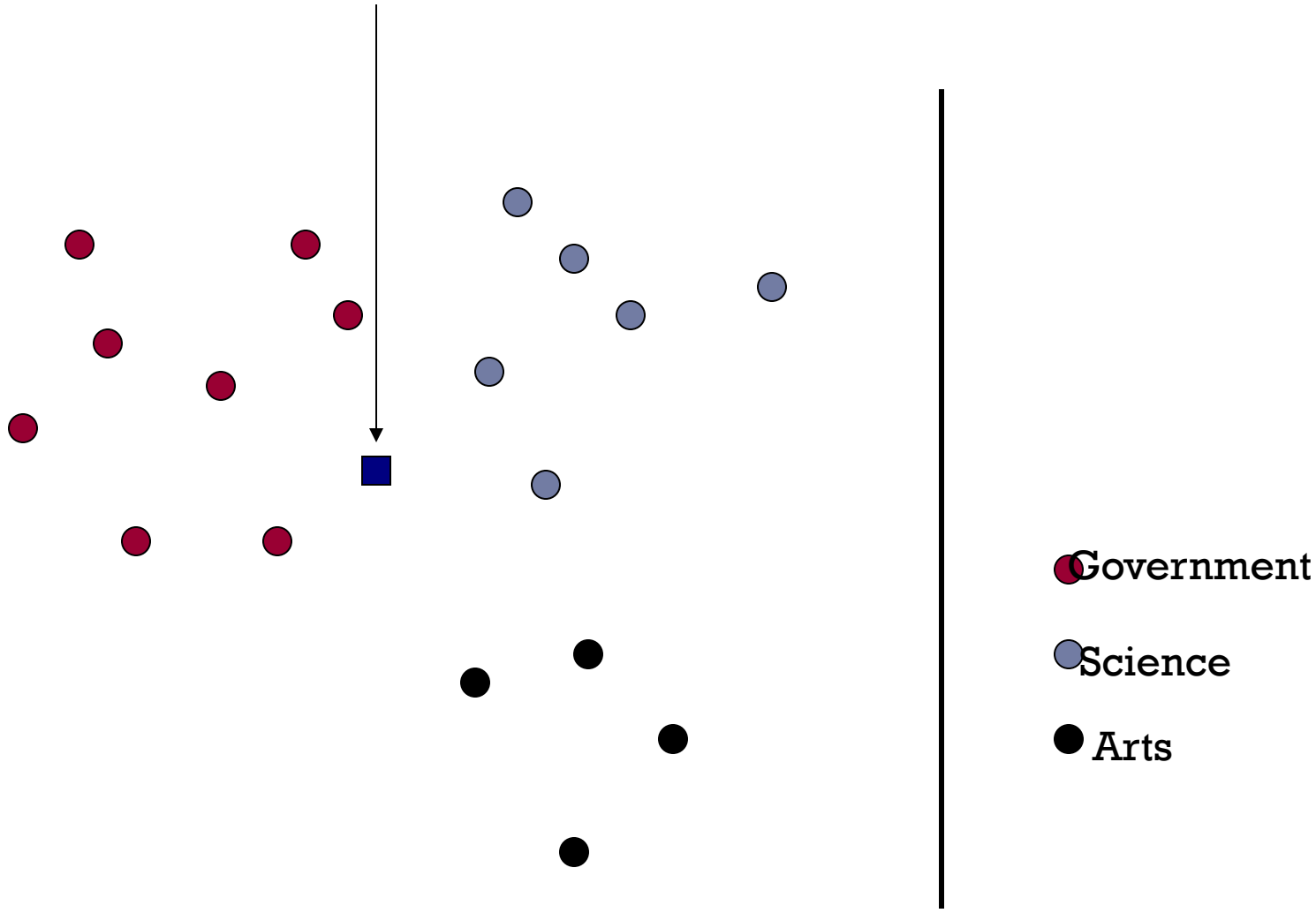
- **Premise 1:** Docs in the same class form a contiguous regions of space
- **Premise 2:** Docs from different classes don't overlap (much)

- We define surfaces to delineate classes in the space

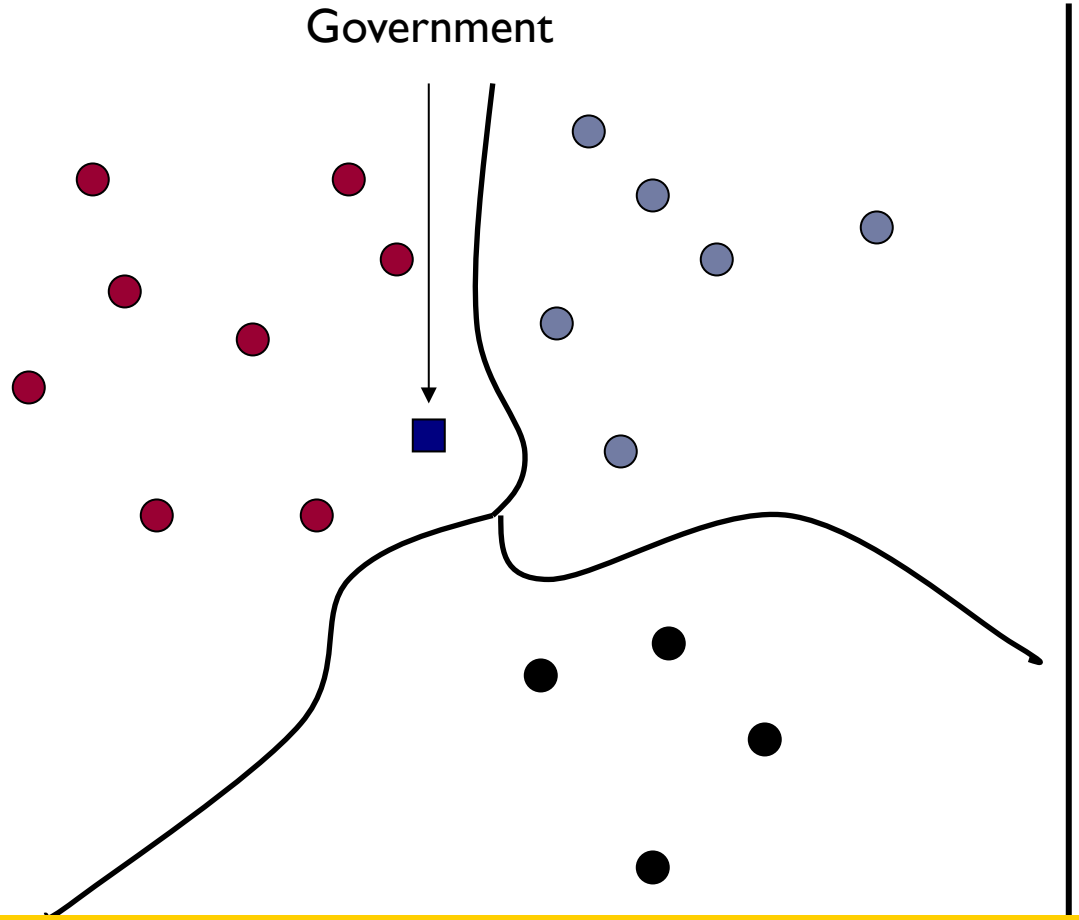
Documents in a vector space



Test document of what class?



Test document of what class?



Is this similarity hypothesis true in general?

Government

Science

Arts

Our main topic today is how to find good separators

Rocchio for text classification

- For training docs in each category, compute a prototype as centroid of the vectors of the training docs in the category.
  - Prototype = centroid of members of class
- Assign test docs to the category with the closest prototype vector based on cosine similarity.
- In relevance feedback, the user marks docs as relevant/non-relevant.
  - Relevant/non-relevant can be viewed as classes or categories.

Definition of centroid

$$\vec{\mu}(c) = \frac{1}{|D_c|} \sum_{d \in D_c} \vec{d}$$

- D_c: docs that belong to class c
- $\vec{d}$: vector space representation of d.

- Centroid will in general not be a unit vector even when the inputs are unit vectors.

Rocchino algorithm

$\textsc{TrainRocchio}(\mathbb{C}, \mathbb{D})$

1 **for each** $c_j \in \mathbb{C}$
2 **do** $D_j \leftarrow \{d : \langle d, c_j \rangle \in \mathbb{D}\}$
3 $\quad \vec{\mu}_j \leftarrow \frac{1}{|D_j|} \sum_{d \in D_j} \vec{v}(d)$
4 **return** $\{\vec{\mu}_1, \ldots, \vec{\mu}_J\}$

$\textsc{ApplyRocchio}(\{\vec{\mu}_1, \ldots, \vec{\mu}_J\}, d)$

1 **return** $\arg\min_j |\vec{\mu}_j - \vec{v}(d)|$

mode	time complexity
training	$\Theta(\lvert\mathbb{D}\rvert L_{\text{ave}} + \lvert\mathbb{C}\rvert\lvert V\rvert) \approx \Theta(\lvert\mathbb{D}\rvert L_{\text{ave}})$
testing	$\Theta(L_{\text{a}} + \lvert\mathbb{C}\rvert M_{\text{a}}) \approx \Theta(\lvert\mathbb{C}\rvert M_{\text{a}})$

Rocchio: example

- We will see that Rocchino finds linear boundaries between classes

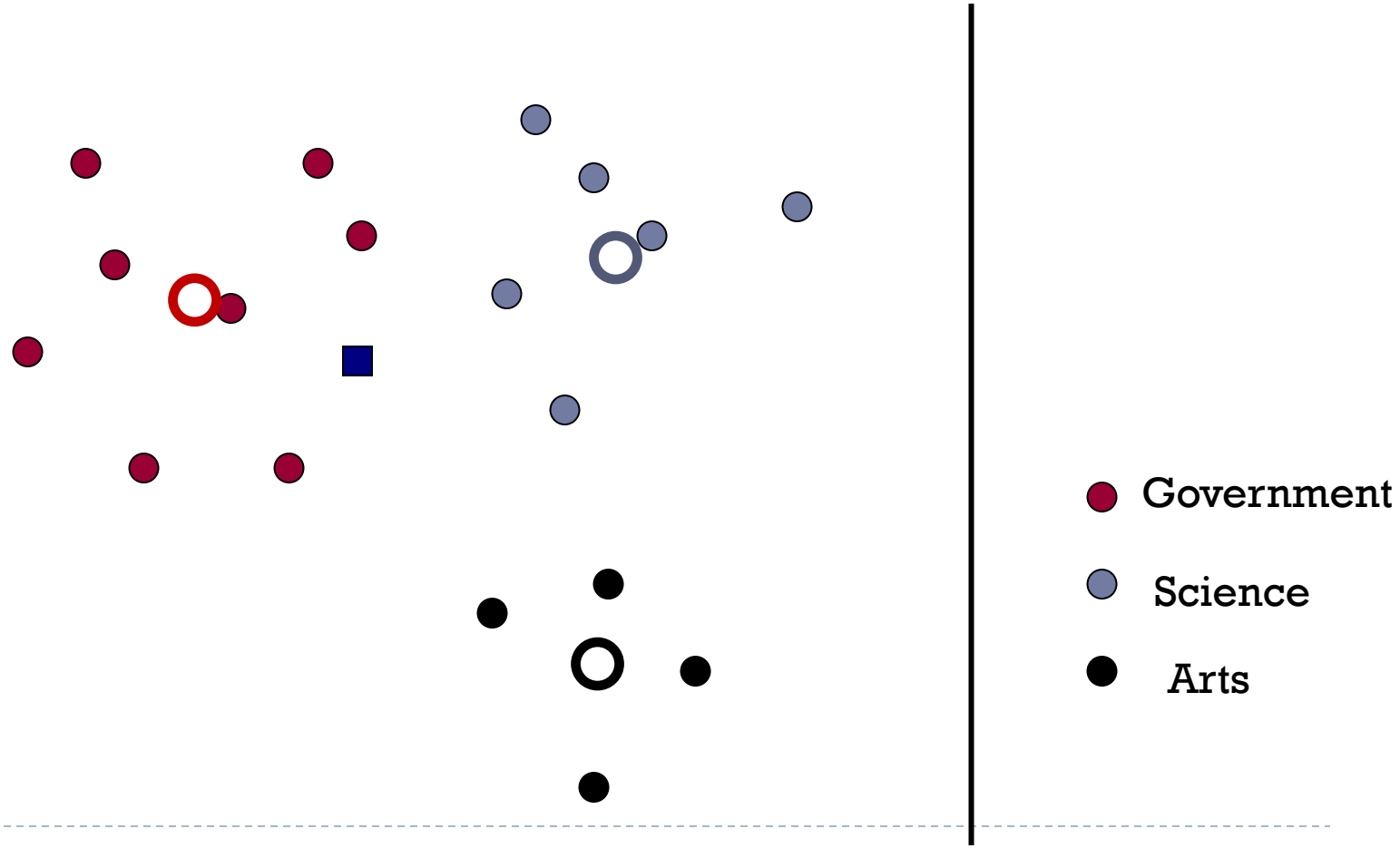
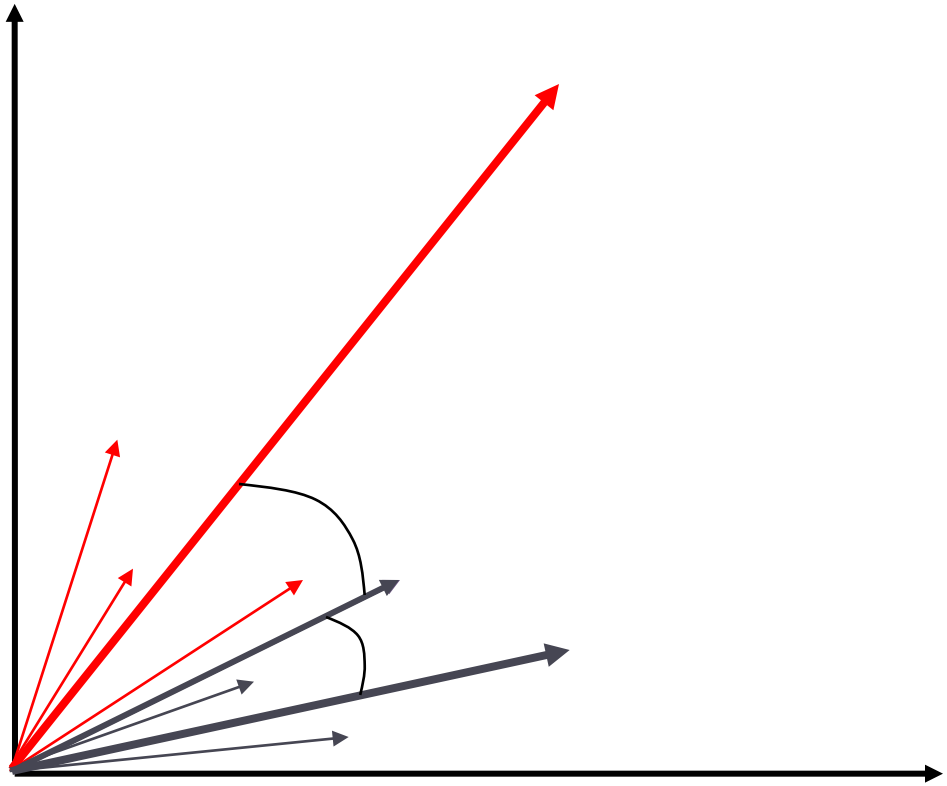


Illustration of Rocchio: text classification

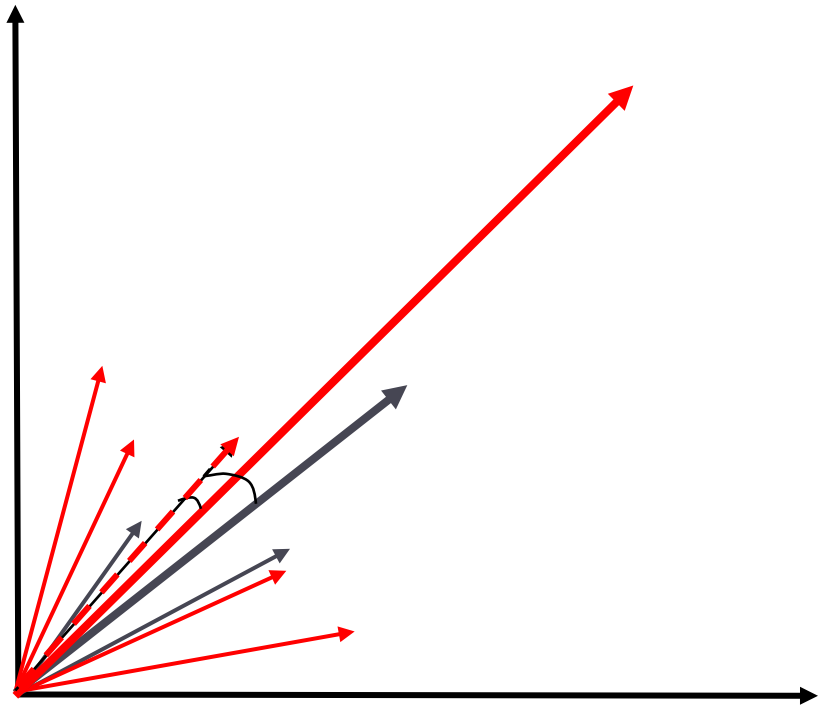


Rocchio properties

- Forms a simple generalization of the examples in each class (a *prototype*).
  - Prototype vector does not need to be normalized.
- Classification is based on similarity to class prototypes.
- Does not guarantee classifications are consistent with the given training data.

Rocchio anomaly

- Prototype models have problems with polymorphic (disjunctive) categories.



Rocchio classification: summary

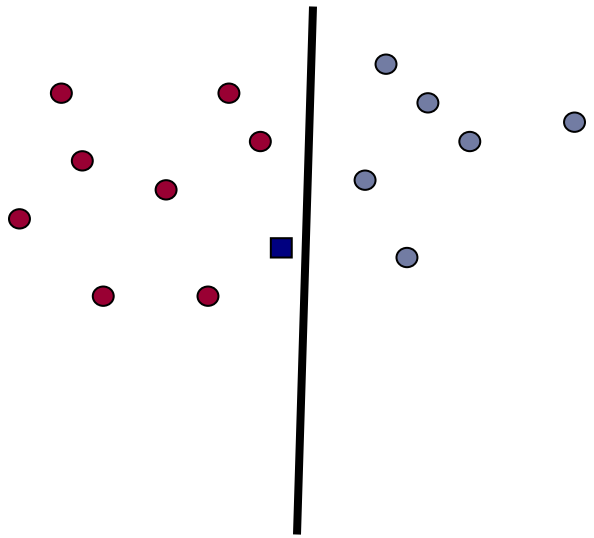
- Rocchio forms a simple representation for each class:
  - Centroid/prototype
  - Classification is based on similarity to the prototype
- It does not guarantee that classifications are consistent with the given training data
- It is little used outside text classification
  - It has been used quite effectively for text classification
  - But in general worse than many other classifiers
- Rocchio does not handle nonconvex, multimodal classes correctly.

Linear classifiers

- Assumption: The classes are linearly separable.
- Classification decision: $\sum_{i=1}^{m} w_i x_i + w_0 > 0$?
- First, we only consider binary classifiers.
  - Geometrically, this corresponds to a line (2D), a plane (3D) or a hyperplane (higher dimensionalities) decision boundary.
- Find the parameters $w_0, w_1, \ldots, w_m$ based on training set.
  - Methods for finding these parameters: Perceptron, Rocchio, …

Separation by hyperplanes

- A simplifying assumption is *linear separability*:
  - in 2 dimensions, can separate classes by a line
  - in higher dimensions, need hyperplanes



Two-class Rocchio as a linear classifier

- Line or hyperplane defined by:

$$w_0 + \sum_{i=1}^{M} w_i d_i = w_0 + \vec{w}^T \vec{d} \geq 0$$

- For Rocchio, set:

$$\vec{w} = \vec{\mu}(c_1) - \vec{\mu}(c_2)$$

$$w_0 = \frac{1}{2}\left(|\vec{\mu}(c_1)|^2 - |\vec{\mu}(c_2)|^2\right)$$

Linear classifier: example

- Class: "interest" (as in interest rate)
- Example features of a linear classifier

w_i	t_i	w_i	t_i
0.70	prime	−0.71	dlrs
0.67	rate	−0.35	world
0.63	interest	−0.33	sees
0.60	rates	−0.25	year
0.46	discount	−0.24	group
0.43	bundesbank	−0.24	dlr

- To classify, find dot product of feature vector and weights

Linear classifier: example

t_i	w_i	d_{1i}	d_{2i}	t_i	w_i	d_{1i}	d_{2i}
prime	0.70	0	1	dlrs	-0.71	1	1
rate	0.67	1	0	world	-0.35	1	0
interest	0.63	0	0	sees	-0.33	0	0
rates	0.60	0	0	year	-0.25	0	0
discount	0.46	1	0	group	-0.24	0	0
bundesbank	0.43	0	0	dlr	-0.24	0	0

$w_0 = 0$

- Class "interest" in Reuters-21578
- d_1:"rate discount dlrs world"
- d_2:"prime dlrs"

- $\vec{w}^T \vec{d}_1 = 0.07 \Rightarrow d_1$ is assigned to the "interest" class
- $\vec{w}^T \vec{d}_2 = -0.01 \Rightarrow d_2$ is not assigned to this class

Naïve Bayes as a linear classifier

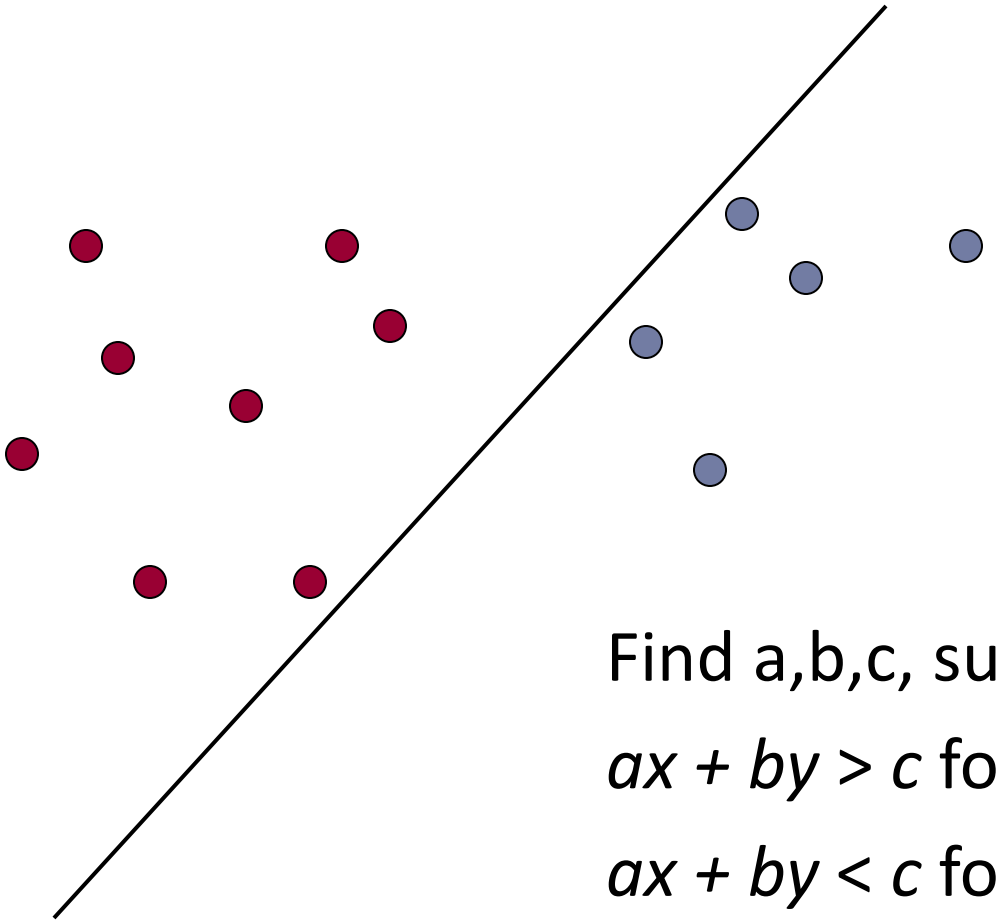
$$P(C_1)\prod_{i=1}^{M} P(t_i|C_1)^{tf_{i,d}} > P(C_2)\prod_{i=1}^{M} P(t_i|C_2)^{tf_{i,d}}$$

$$\log P(C_1) + \sum_{i=1}^{M} tf_{i,d} \times \log P(t_i|C_1) > \log P(C_2) + \sum_{i=1}^{M} tf_{i,d} \times \log P(t_i|C_2)$$

$$\log \frac{P(C_1)}{P(C_2)} + \sum_{i=1}^{M} tf_{i,d} \times \log \frac{P(t_i|C_1)}{P(t_i|C_2)} > 0$$

$$w_i = \log \frac{P(t_i|C_1)}{P(t_i|C_2)} \qquad x_i = tf_{i,d} \qquad w_0 = \log \frac{P(C_1)}{P(C_2)}$$

Linear programming / Perceptron



Find a,b,c, such that

$ax + by > c$ for red points

$ax + by < c$ for blue points

Stochastic gradient descent for Perceptron

- Single-sample perceptron:
  - If $\boldsymbol{x}^{(i)}$ is misclassified:

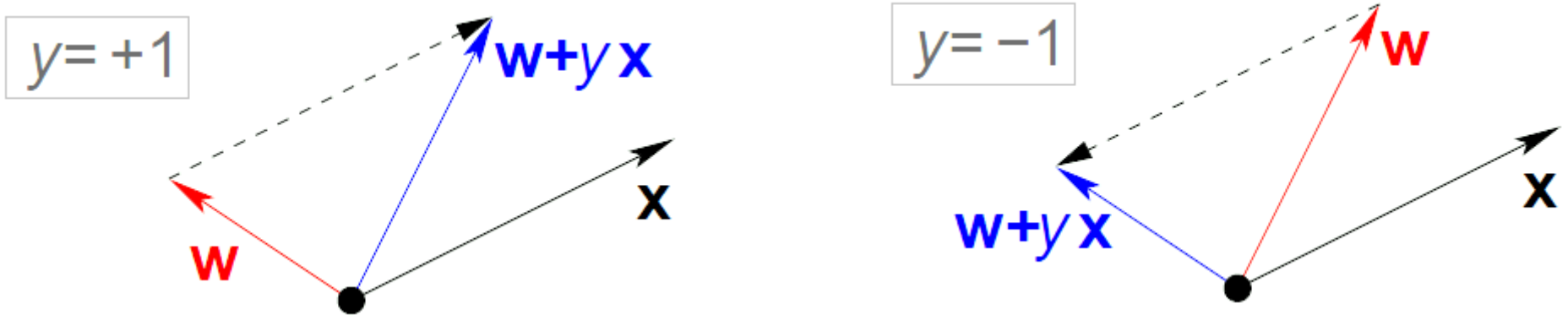
$$\boldsymbol{w}^{t+1} = \boldsymbol{w}^{t} + \boldsymbol{x}^{(i)} y^{(i)}$$

- Perceptron convergence theorem: for linearly separable data
  - If training data are linearly separable, the single-sample perceptron is also guaranteed to find a solution in a finite number of steps

Fixed-Increment single sample Perceptron

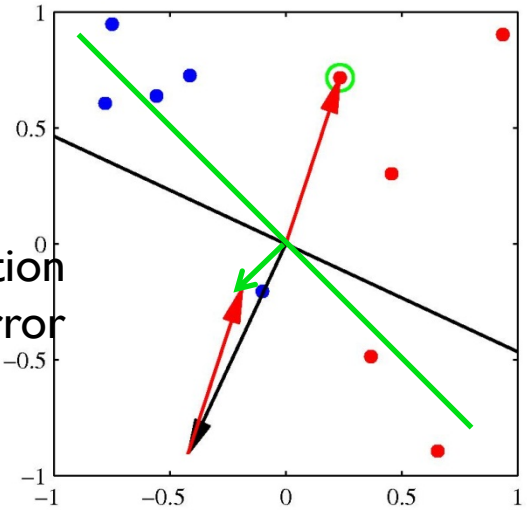
$\boldsymbol{w} \leftarrow \mathbf{0}$
$t \leftarrow 0$
repeat
    $t \leftarrow t + 1$
    $i \leftarrow t \bmod N$
    if $\boldsymbol{x}^{(i)}$ is misclassified then
        $\boldsymbol{w} = \boldsymbol{w} + \boldsymbol{x}^{(i)} y^{(i)}$
Until all patterns properly classified

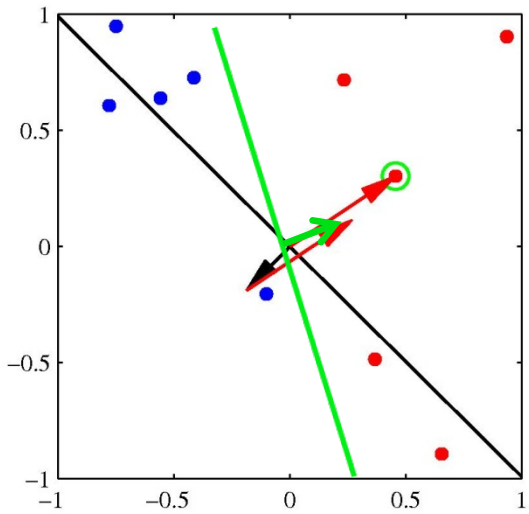
Example



Perceptron: Example

Change $\boldsymbol{w}$ in a direction that corrects the error



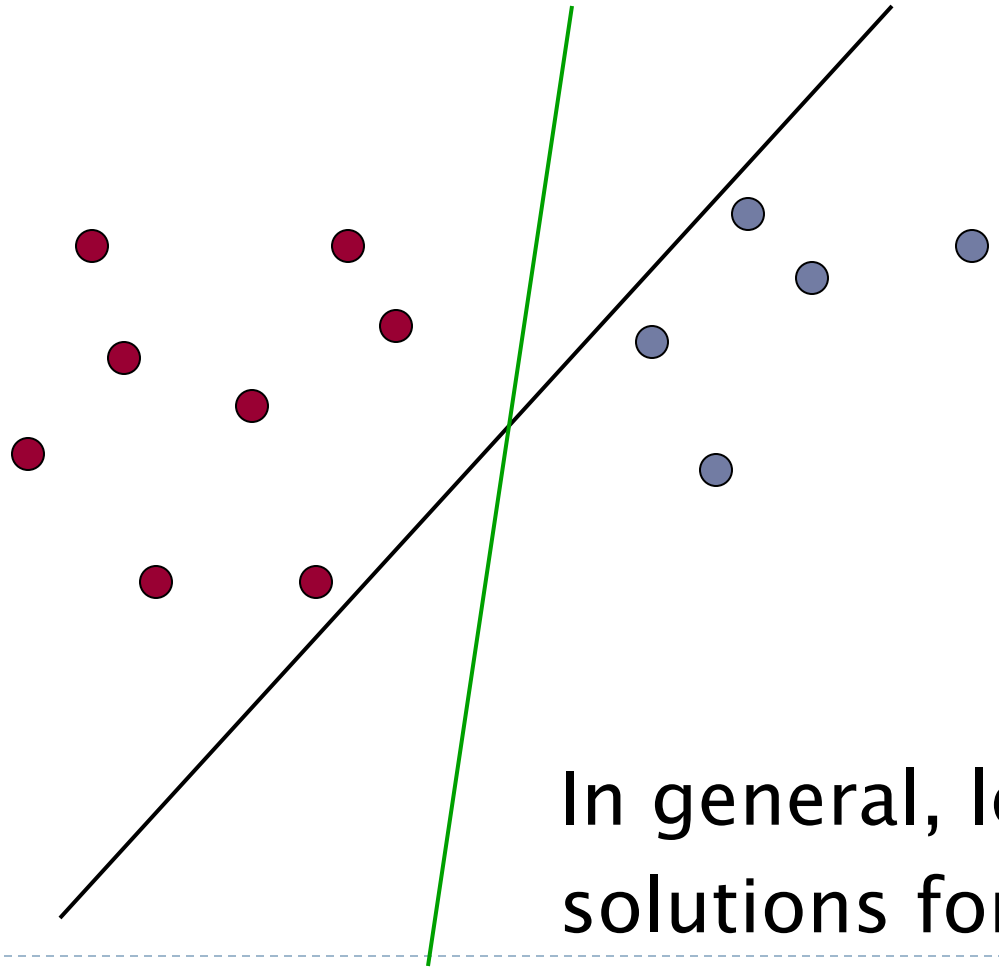


[Bishop]

Linear classifiers

- Many common text classifiers are linear classifiers
- Classifiers more powerful than linear often don't perform (much) better on text problems. Why?
- Despite the similarity of linear classifiers, noticeable performance differences between them
  - For separable problems, there is an infinite number of separating hyperplanes.
    - Different training methods pick different hyperplanes.
  - Also different strategies for non-separable problems

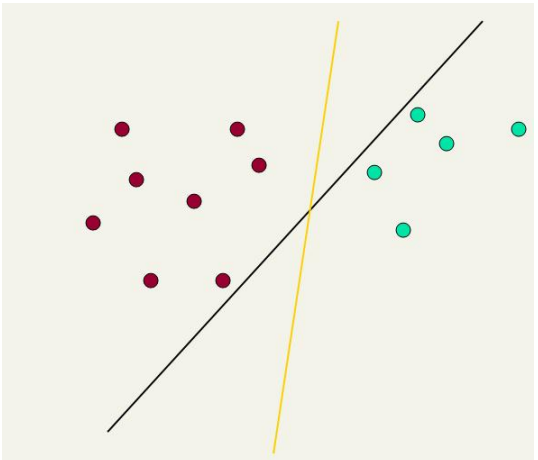
Which hyperplane?



In general, lots of possible solutions for *a*,*b*,*c*.

Which hyperplane?

- Lots of possible solutions for *a,b,c*.
- Some methods find a separating hyperplane, but not the optimal one [according to some criterion of expected goodness]
- Which points should influence optimality?
  - All points
    - E.g., Rocchino
  - Only "difficult points" close to decision boundary
    - E.g., Support Vector Machine (SVM)



Linear classifiers: Which Hyperplane?

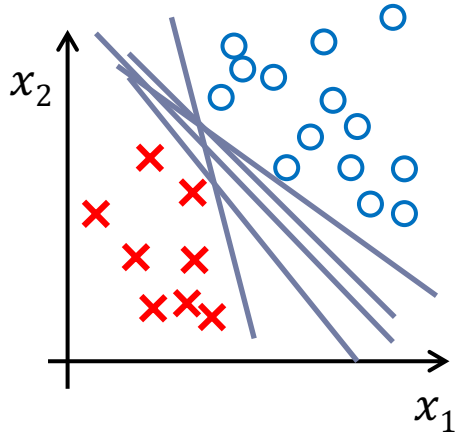
- Some methods find a separating hyperplane, but not the optimal one [according to some criterion of expected goodness]
  - E.g., perceptron

- A Support Vector Machine (SVM) finds an optimal* solution.
  - Maximizes the distance between the hyperplane and the "difficult points" close to decision boundary
  - One intuition: if there are no points near the decision surface, then there are no very uncertain classification decisions

Margin

- Which line is better to select as the boundary to provide more generalization capability?

Larger margin provides better generalization to unseen data

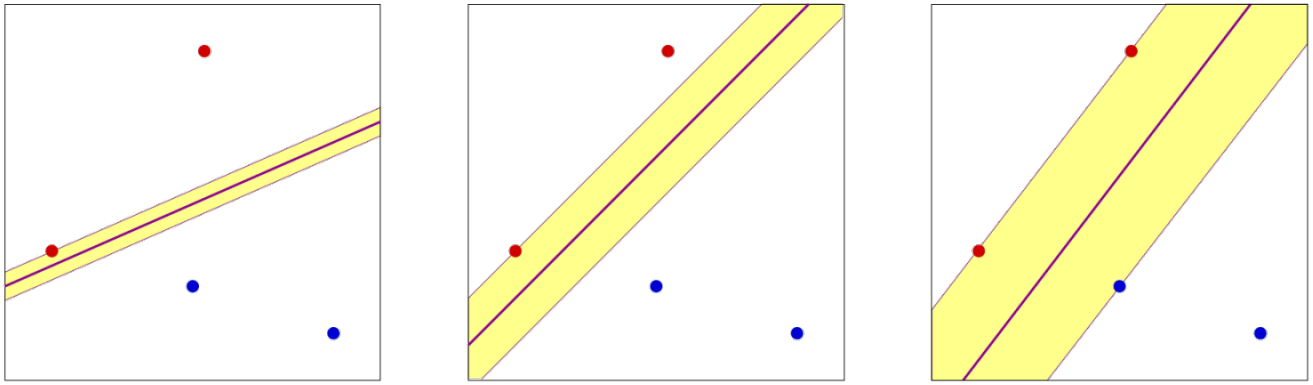


- **Margin** for a hyperplane that separates samples of two linearly separable classes is:
  - The smallest distance between the decision boundary and any of the training samples

What is better linear separation

- Linearly separable data

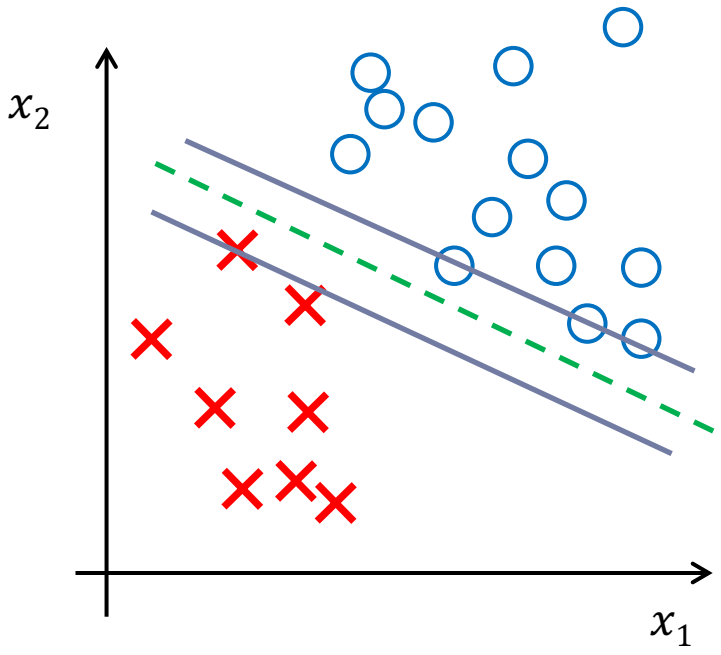
- Which line is better?

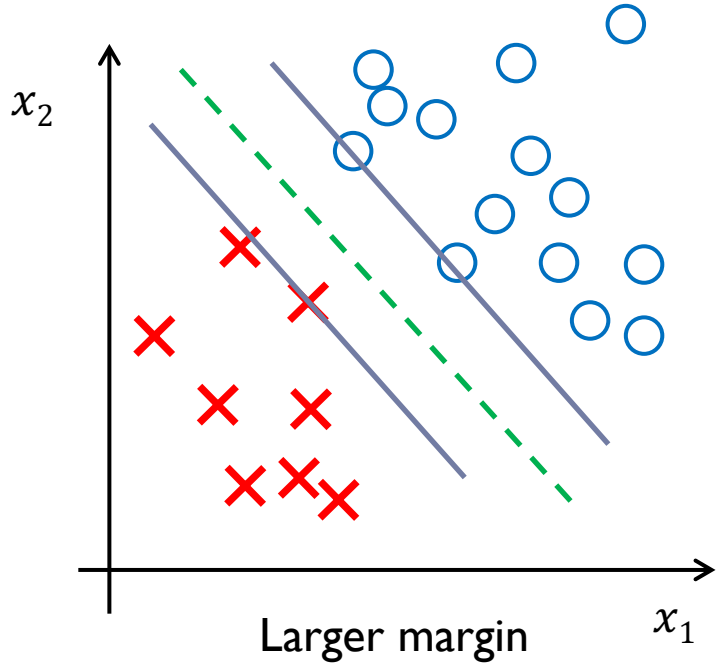


- Why the bigger margin?

Maximum margin

- SVM finds the solution with maximum margin
  - Solution: a hyperplane that is farthest from all training samples

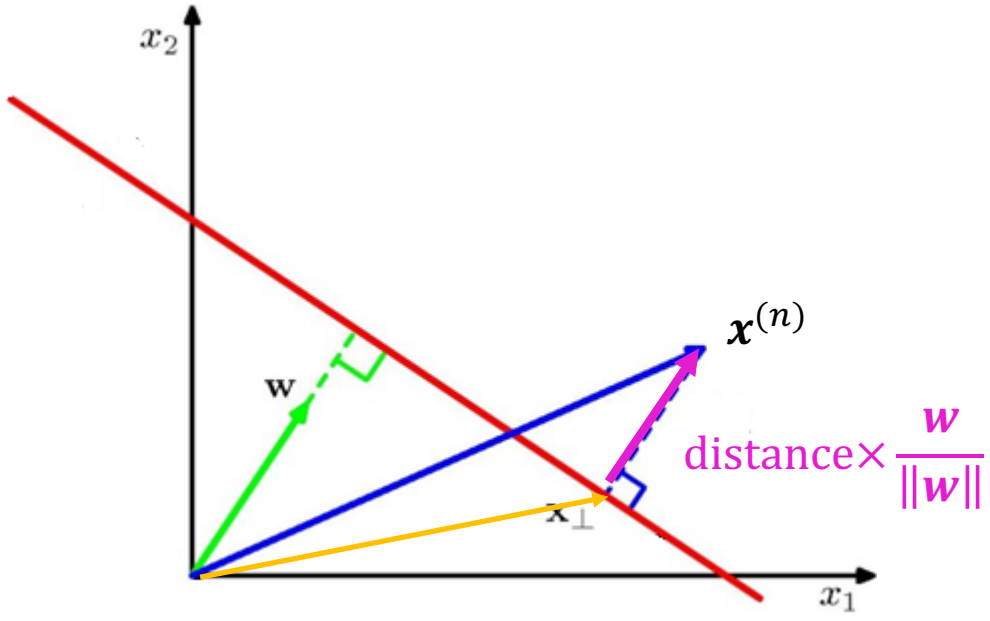




- The hyperplane with the largest margin has equal distances to the nearest sample of both classes

Distance between an $\boldsymbol{x}^{(n)}$ and the plane

$$\text{distance} = \frac{\left|\boldsymbol{w}^T \boldsymbol{x}^{(n)} + w_0\right|}{\|\boldsymbol{w}\|}$$



Geometric Margin

- Distance from example to the separator is $r = y\frac{w^T x+b}{\|w\|}$
- Examples closest to the hyperplane are ***support vectors***.
- ***Margin*** ρ of the separator is the width of separation between support vectors of classes.

Maximum Margin: Formalization

- **w**: decision hyperplane normal vector
- $\boldsymbol{x}^{(i)}$: data point *i*
- $y^{(i)}$: class of data point *i* (+1 or -1)
- Classifier is: $f(\boldsymbol{x}^{(i)}) = sign(\boldsymbol{w}^T \boldsymbol{x}^{(i)} + b)$
- Functional margin of $\boldsymbol{x}^{(i)}$ is: $y^{(i)}(\boldsymbol{w}^T \boldsymbol{x}^{(i)} + b)$

- The functional margin of a dataset is twice the minimum functional margin for any point

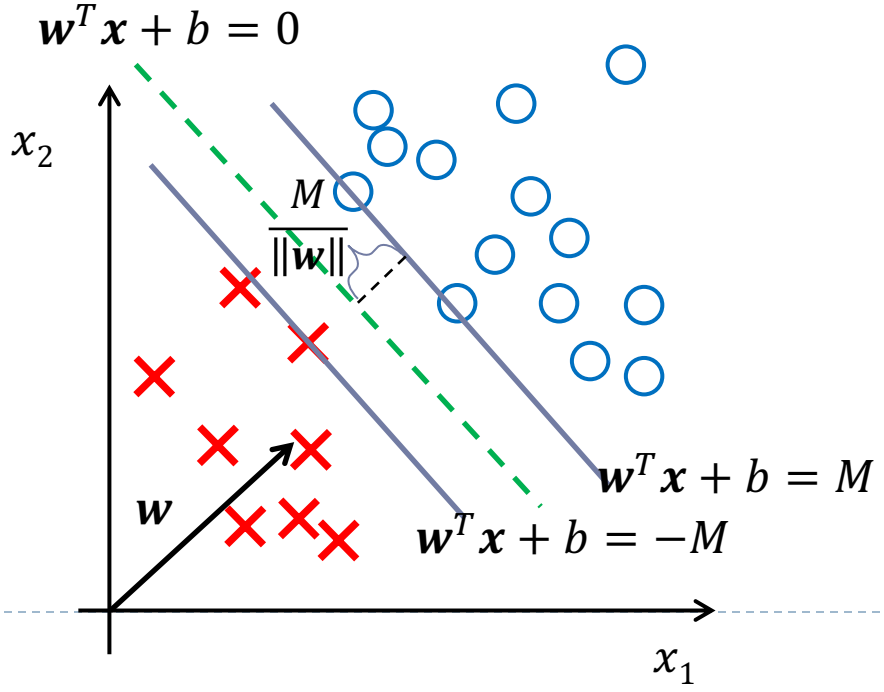
Hard-margin SVM: Optimization problem

$$\max_{M,\boldsymbol{w},b} \frac{2M}{\|\boldsymbol{w}\|}$$

$$\text{s.t.} \left(\boldsymbol{w}^T \boldsymbol{x}^{(i)} + b\right) \geq M \quad \forall \boldsymbol{x}^{(i)} \in C_1 \longrightarrow y^{(i)} = 1$$

$$\left(\boldsymbol{w}^T \boldsymbol{x}^{(i)} + b\right) \leq -M \quad \forall \boldsymbol{x}^{(i)} \in C_2 \longrightarrow y^{(i)} = -1$$

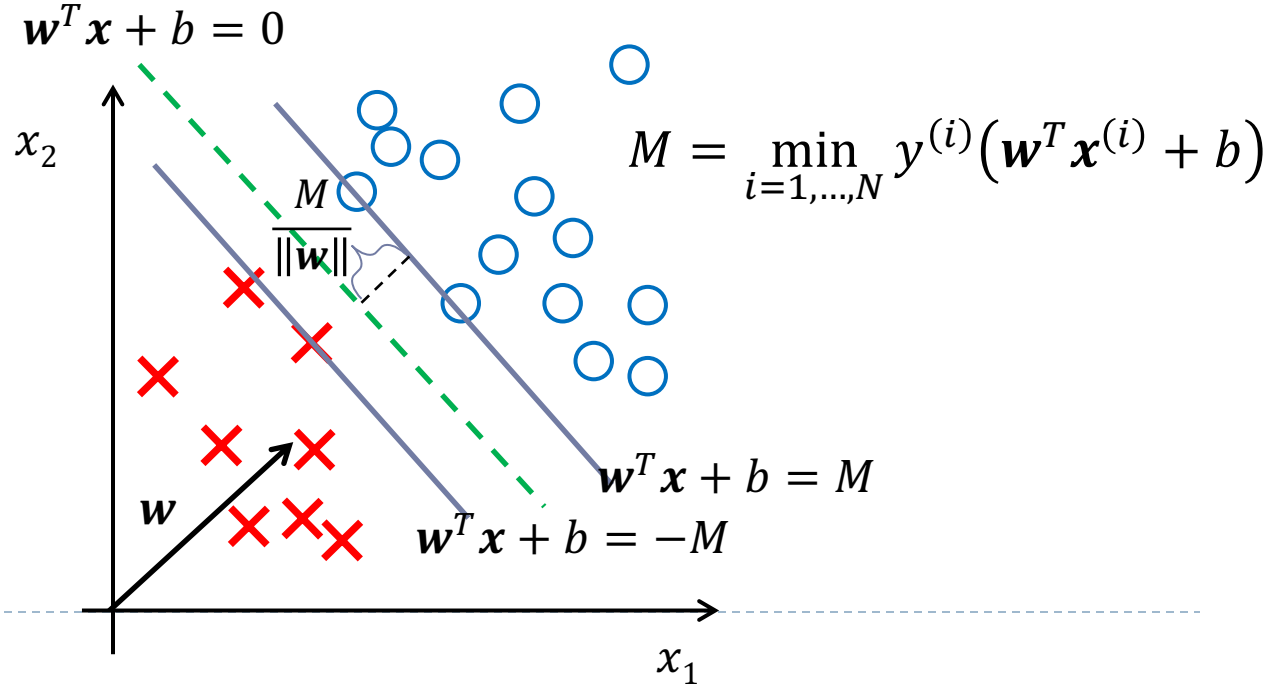
Margin: $2\frac{M}{\|\boldsymbol{w}\|}$



Hard-margin SVM: Optimization problem

$$\max_{M,\boldsymbol{w},b} \frac{2M}{\|\boldsymbol{w}\|}$$

$$\text{s.t. } y^{(i)}\left(\boldsymbol{w}^T\boldsymbol{x}^{(i)} + b\right) \geq M \quad i = 1, \ldots, N$$

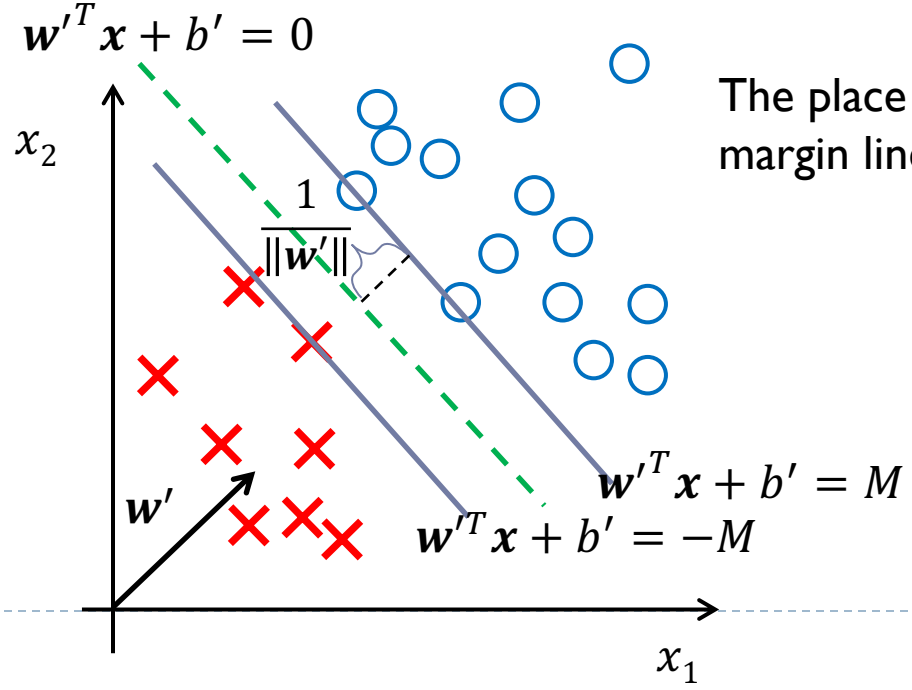


Hard-margin SVM: Optimization problem

We can set $\boldsymbol{w}' = \frac{\boldsymbol{w}}{M}, b' = \frac{b}{M}$:

$$\max_{\boldsymbol{w}', b'} \frac{2}{\|\boldsymbol{w}'\|}$$

$$\text{s.t. } y^{(i)}\left(\boldsymbol{w}'^{T}\boldsymbol{x}^{(i)} + b'\right) \geq 1 \quad i = 1, \ldots, N$$



The place of boundary and margin lines do not change

Linear SVM Mathematically
The linearly separable case

- Assume that the functional margin of each data item is at least 1, then the following two constraints follow for a training set $\{(\boldsymbol{x}^{(i)}, y^{(i)})\}$

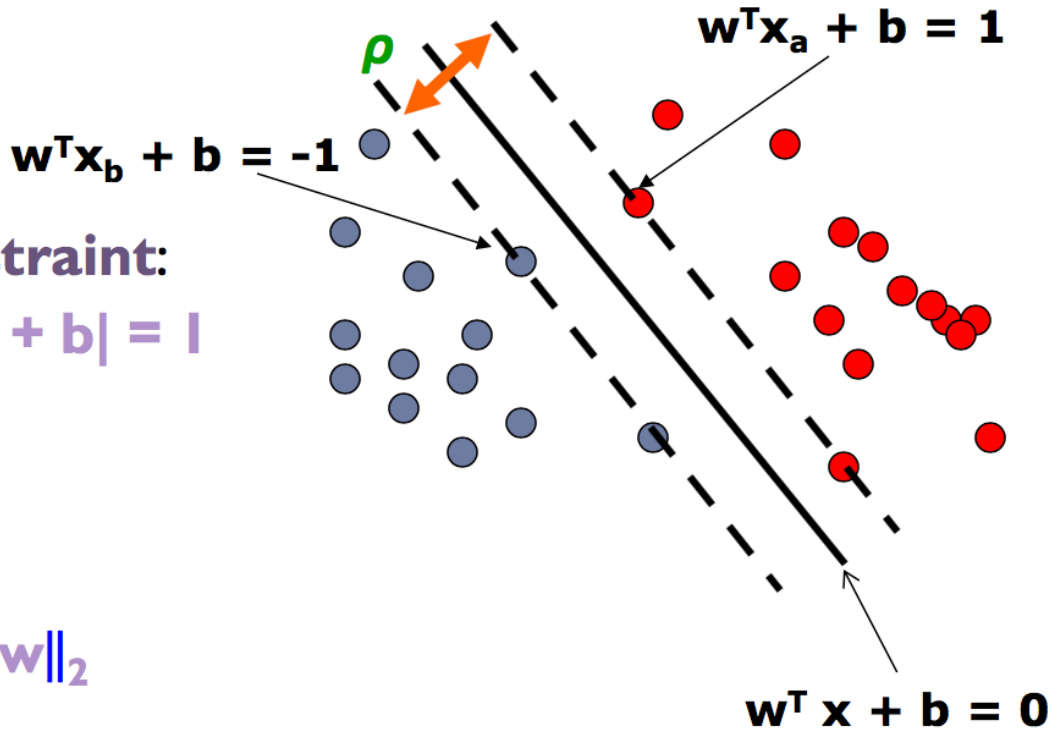
$$(\boldsymbol{w}^T \boldsymbol{x}^{(i)} + b) \geq 1 \text{ if } y^{(i)} = 1$$

$$(\boldsymbol{w}^T \boldsymbol{x}^{(i)} + b) \leq 1 \text{ if } y^{(i)} = -1$$

- For support vectors, the inequality becomes an equality
- Then, since each example's distance from the hyperplane is $r = y\frac{\boldsymbol{w}^T \boldsymbol{x}+b}{\|\boldsymbol{w}\|}$
- The margin is: $\rho = \frac{2}{\|\boldsymbol{w}\|}$

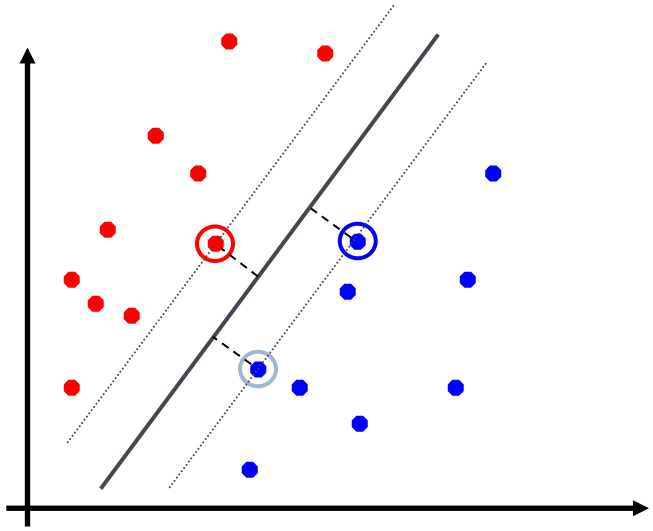
Linear Support Vector Machine (SVM)

- **Hyperplane**
  $\mathbf{w}^T \mathbf{x} + b = 0$
- **Extra scale constraint:**
  $\min_{i=1,\ldots,n} |\mathbf{w}^T \mathbf{x}_i + b| = 1$
- This implies:
  $\mathbf{w}^T(\mathbf{x}_a - \mathbf{x}_b) = 2$
  $\rho = \|\mathbf{x}_a - \mathbf{x}_b\|_2 = 2/\|\mathbf{w}\|_2$



Support Vector Machines

- Maximizing the margin: good according to intuition, theory, practice
- Support vector machines (SVMs) find the separator with max margin



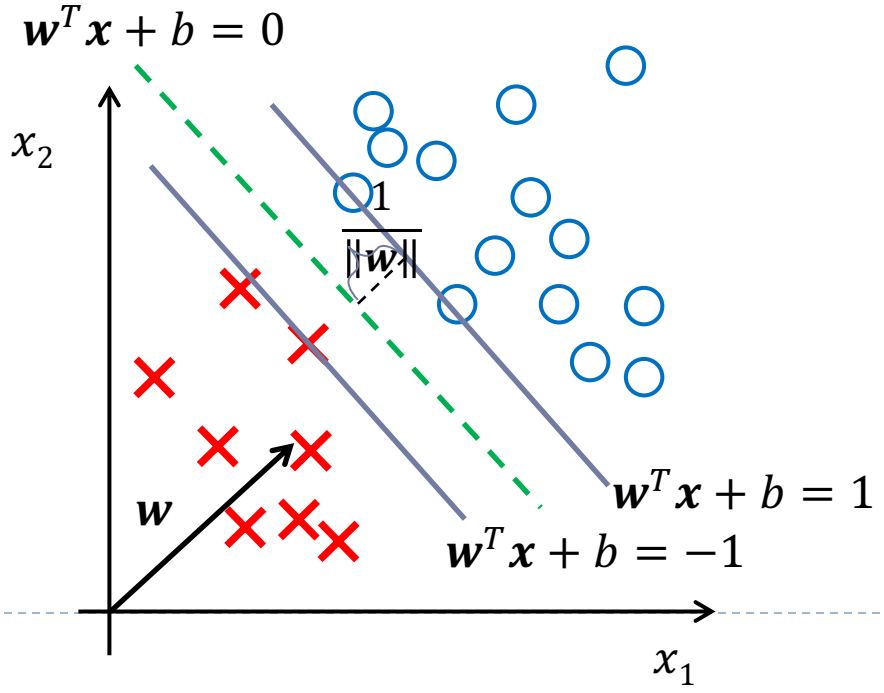
Hard-margin SVM: Optimization problem

$$\max_{\boldsymbol{w},b} \frac{2}{\|\boldsymbol{w}\|}$$

$$\text{s.t.} \left(\boldsymbol{w}^T \boldsymbol{x}^{(n)} + b\right) \geq 1 \quad \forall y^{(n)} = 1$$

$$\left(\boldsymbol{w}^T \boldsymbol{x}^{(n)} + b\right) \leq -1 \quad \forall y^{(n)} = -1$$

Margin: $\frac{2}{\|\boldsymbol{w}\|}$



Hard-margin SVM: Optimization problem

We can equivalently optimize:

min ||**w**||= max 1/||**w**||

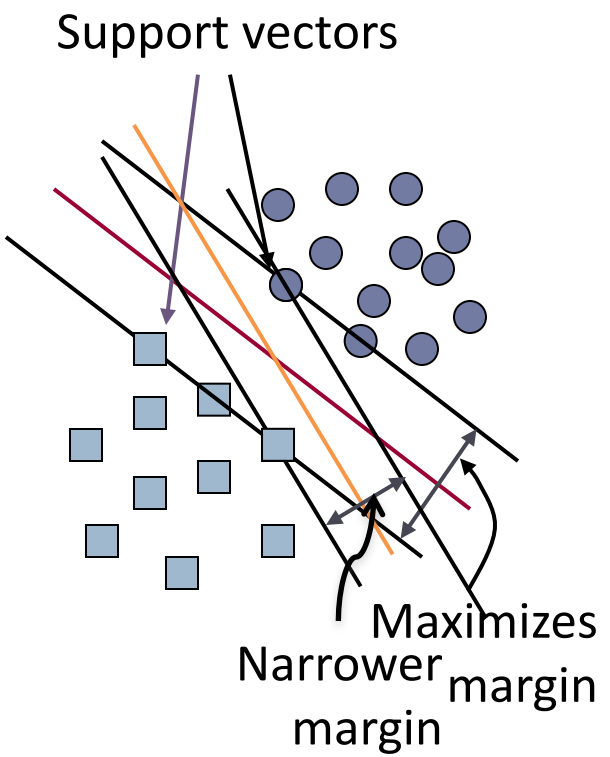
$$\min_{\boldsymbol{w},b} \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w}$$

$$\text{s.t.} \quad y^{(n)}\left(\boldsymbol{w}^T\boldsymbol{x}^{(n)} + b\right) \geq 1 \quad n = 1, \dots, N$$

- It is a convex Quadratic Programming (QP) problem
  - There are computationally efficient packages to solve it.
  - It has a global minimum (if any).
- This is now optimizing a *quadratic* function subject to *linear* constraints
  - QPs are a well-known class of mathematical programming problem, and many (intricate) algorithms exist for solving them

Support Vector Machine (SVM)

- SVMs maximize the *margin* around the separating hyperplane.
  - A.k.a. large margin classifiers
- The decision function is fully specified by a subset of training samples, *the support vectors*.
- Solving SVMs is a *quadratic programming* problem
- Seen by many as the most successful current text classification method*



*but other discriminative methods often perform very similarly

Soft Margin Classification

- If the training data is not linearly separable, *slack variables* ξi can be added to allow misclassification of difficult or noisy examples.

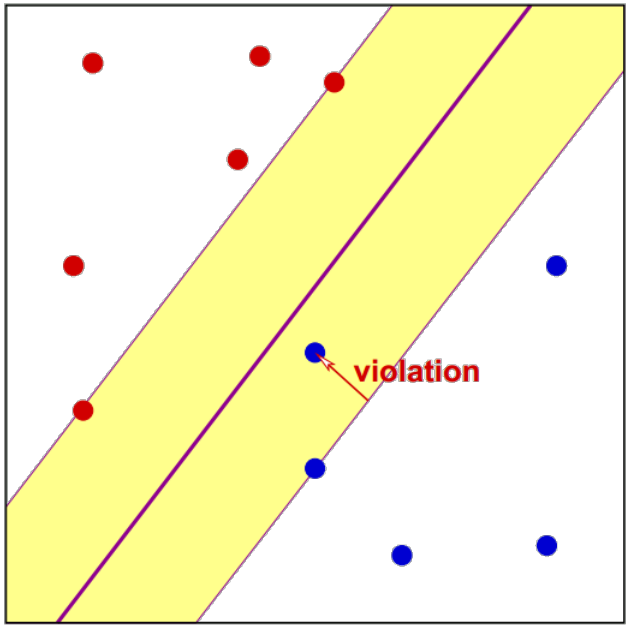
- Allow some errors
  - Let some points be moved to where they belong at a cost

- Still, try to minimize training set errors, and to place hyperplane "far" from each class (large margin)

Error measure

- Margin violation amount ξ_n ($\xi_n \geq 0$):
  - $y^{(n)}\left(\boldsymbol{w}^T \boldsymbol{x}^{(n)} + b\right) \geq 1 - \xi_n$

- Total violation: $\sum_{n=1}^{N} \xi_n$



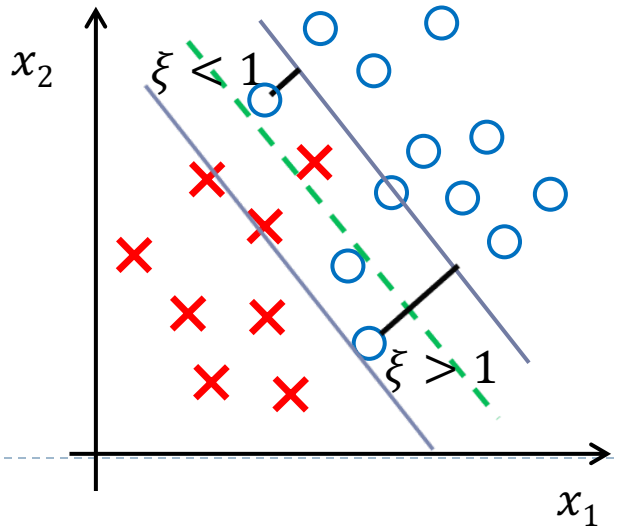
Soft-margin SVM: Optimization problem

- SVM with slack variables: allows samples to fall within the margin, but penalizes them

$$\min_{\boldsymbol{w},b,\{\xi_n\}_{n=1}^N} \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{n=1}^{N}\xi_n$$

$$\text{s.t.} \quad y^{(n)}\left(\boldsymbol{w}^T\boldsymbol{x}^{(n)} + b\right) \geq 1 - \xi_n \quad n = 1, \ldots, N$$

$$\xi_n \geq 0$$



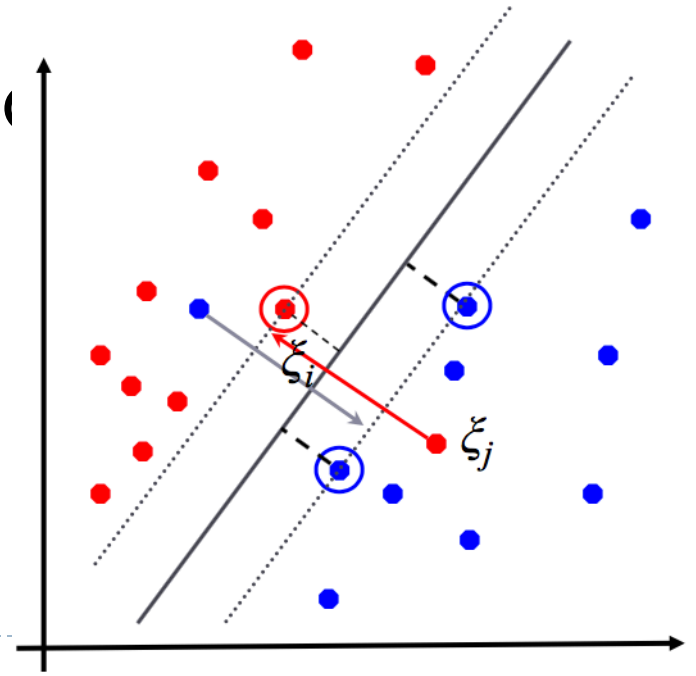
ξ_n: **slack** variables

$0 < \xi_n < 1$: if $\boldsymbol{x}^{(n)}$ is correctly classified but inside margin

$\xi_n > 1$: if $\boldsymbol{x}^{(n)}$ is misclassifed

Soft-margin SVM

- linear penalty (hinge loss) for a sample if it is misclassified or lied in the margin
  - tries to maintain ξ_n small while maximizing the margin.
  - always finds a solution (as opposed to hard-margin SVM)
  - more robust to the outliers
- Soft margin problem is still a convex (



SVM Summary

- Choose hyperplane based on support vectors
  - Support vector = "critical" point close to decision boundary
- Perhaps best performing text classifier
  - But there are other methods that perform about as well as SVM, such as regularized logistic regression (Zhang & Oles 2001)
- Partly popular due to availability of good software
  - SVMlight is accurate and fast – and free (for research)
  - Now lots of good software: libsvm, TinySVM, ...

Linear classifiers: binary and multiclass classification

- Consider 2 class problems
  - Deciding between two classes, perhaps, government and non-government

- Multi-class
  - How do we define (and find) the separating surface?
  - How do we decide which region a test doc is in?

More than two classes

- Any-of classification
  - Classes are not mutually exclusive.
  - A doc can belong to 0, 1, or >1 classes.
  - For simplicity, decompose into *K* binary problems
  - Quite common for docs

- One-of classification
  - Classes are mutually exclusive.
  - Each doc belongs to exactly one class
  - E.g., digit recognition 4
    - Digits are mutually exclusive

Set of binary classifiers: any of

- Build a separator between each class and its complementary set (docs from all other classes).
- Given test doc, evaluate it for membership in each class.
- Apply decision criterion of classifiers independently
  - It works although considering dependencies between categories may be more accurate

k Nearest Neighbor Classification

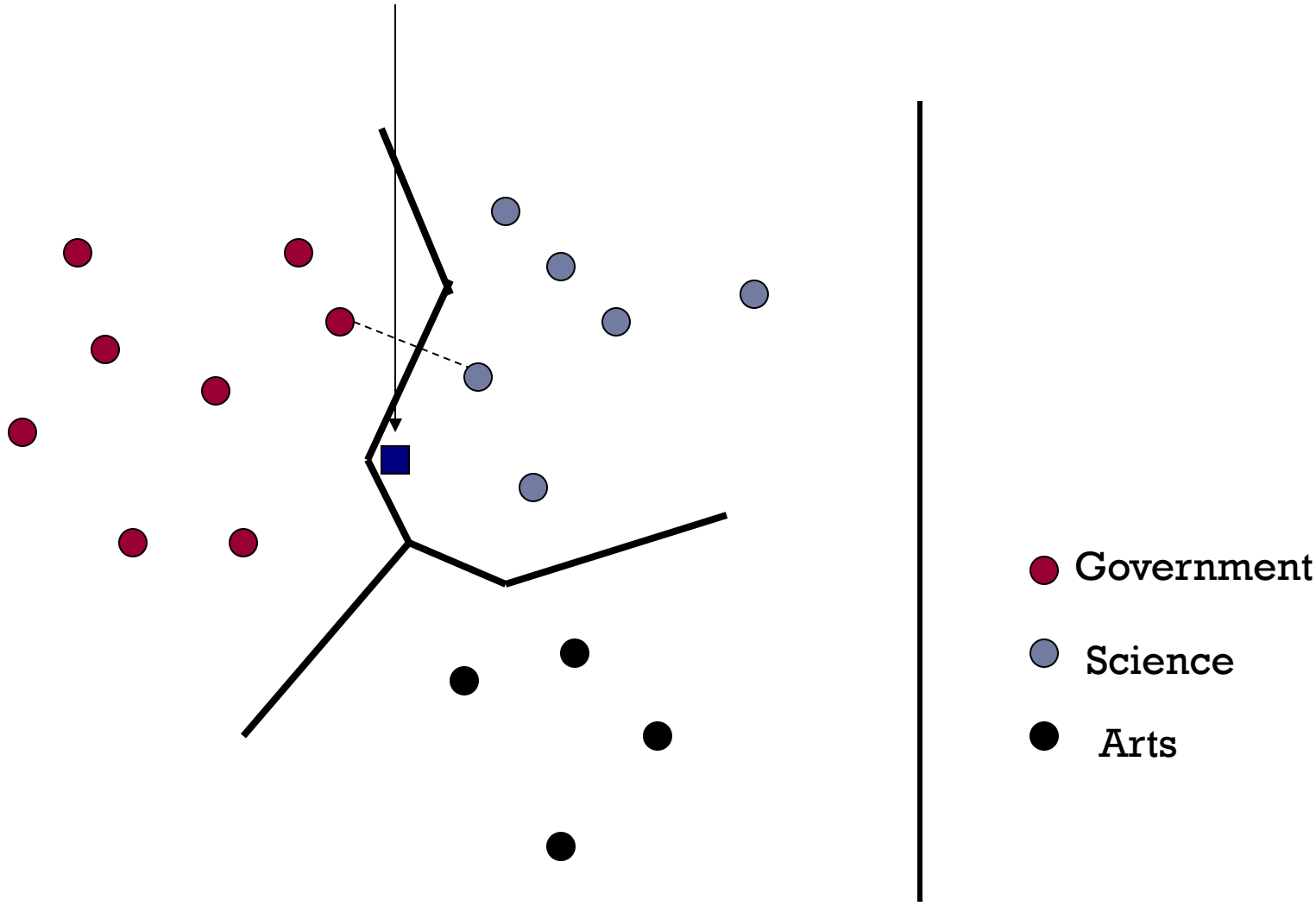
- kNN = *k* Nearest Neighbor

- To classify a document *d*:
  - Define *k*-neighborhood as the *k* nearest neighbors of *d*
  - Pick the majority class label in the *k*-neighborhood

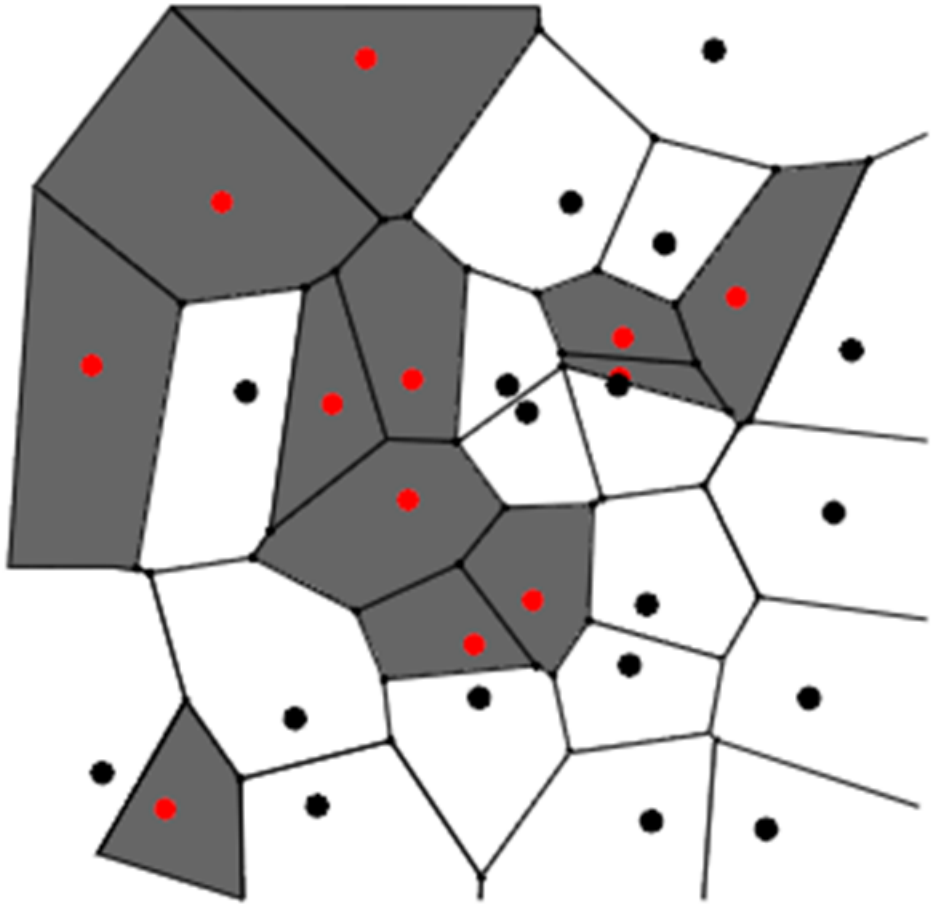
Nearest-Neighbor (1NN) classifier

- Learning phase:
  - Just storing the representations of the training examples in *D*.
  - Does not explicitly compute category prototypes.
- Testing instance *x (under 1NN)*:
  - Compute similarity between *x* and all examples in *D*.
  - Assign *x* the category of the most similar example in *D*.
- Rationale of kNN: contiguity hypothesis
  - We expect a test doc d to have the same label as the training docs located in the local region surrounding d.

Test Document = Science



1NN: Voronoi tessellation



The decision boundaries between classes are piecewise linear.

k Nearest Neighbor (kNN) classifier

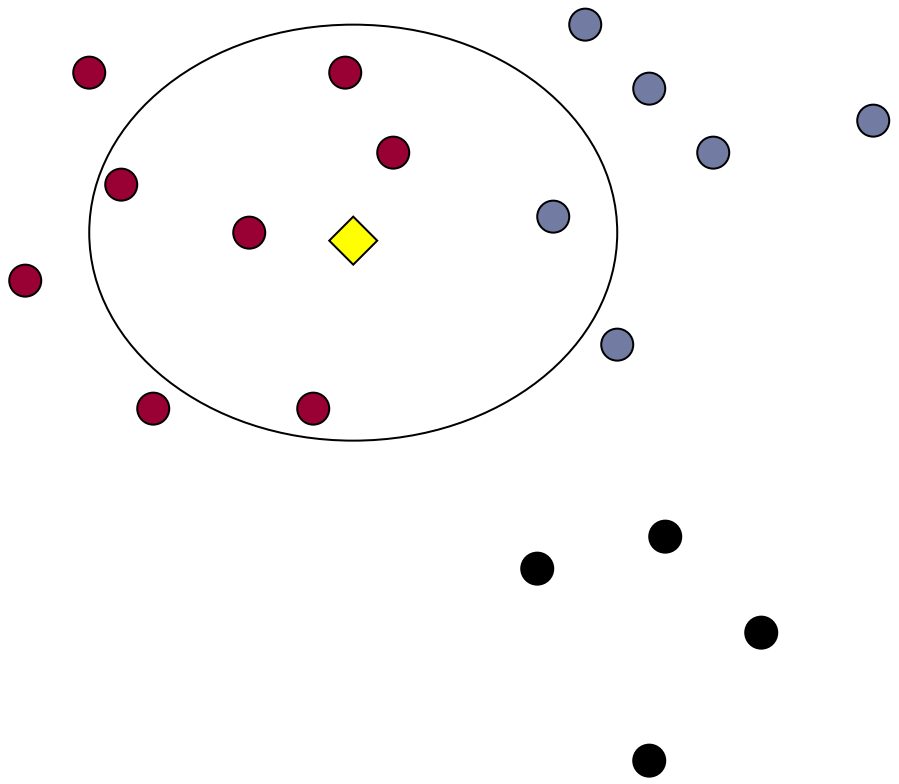
- 1NN: subject to errors due to
  - A single atypical example.
  - Noise (i.e., an error) in the category label of a single training example.

- More robust alternative:
  - find the *k* most-similar examples
  - return the majority category of these *k* examples.

kNN classifier: probabilistic perspective

- To classify *d*:
  - Find its k nearest neighbors in training data
  - For each class c, find the neighbors that are in the class *c*
  - Choose the class to which most of the neighbors belong

kNN example: k=6



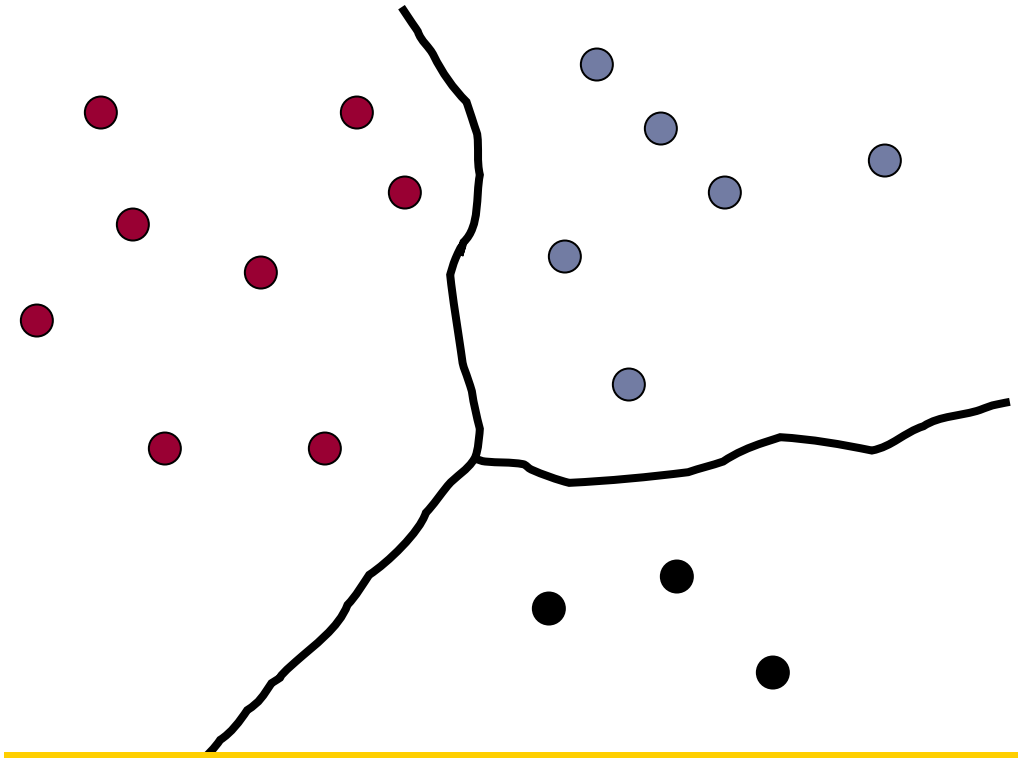
P(science|◆)?

Government

Science

Arts

kNN decision boundaries



Boundaries are in principle arbitrary surfaces (polyhedral)

Government

Science

Arts

kNN gives locally defined decision boundaries between classes – far away points do not influence each classification decision (unlike Rocchio, etc.)

kNN algorithm

TRAIN-KNN($\mathbb{C}, \mathbb{D}$)
1 $\mathbb{D}' \leftarrow$ PREPROCESS($\mathbb{D}$)
2 $k \leftarrow$ SELECT-K($\mathbb{C}, \mathbb{D}'$)
3 **return** $\mathbb{D}', k$

APPLY-KNN($\mathbb{D}', k, d$)
1 $S_k \leftarrow$ COMPUTENEARESTNEIGHBORS($\mathbb{D}', k, d$)
2 **for each** $c_j \in \mathbb{C}(\mathbb{D}')$
3 **do** $p_j \leftarrow |S_k \cap c_j|/k$
4 **return** $\arg\max_j p_j$

Time complexity of kNN

kNN with preprocessing of training set

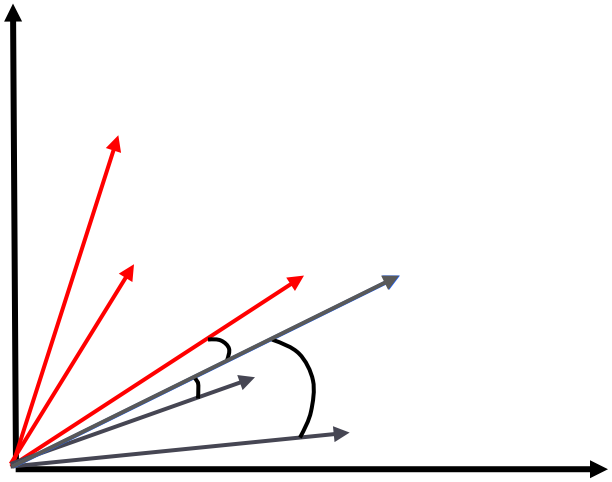
training	$\Theta(\lvert\mathbb{D}\rvert L_{\text{ave}})$
testing	$\Theta(L_{\text{a}} + \lvert\mathbb{D}\rvert M_{\text{ave}} M_{\text{a}}) = \Theta(\lvert\mathbb{D}\rvert M_{\text{ave}} M_{\text{a}})$

- kNN test time proportional to the size of the training set!
  - kNN is inefficient for very large training sets.

Similarity metrics

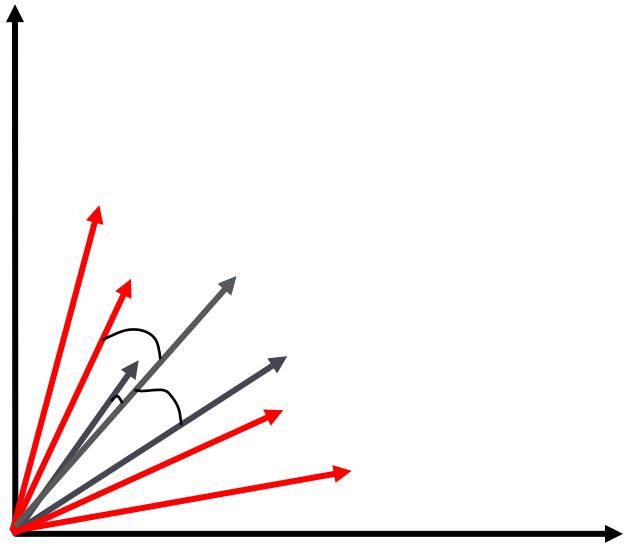
- Nearest neighbor method depends on a **similarity** (or distance) metric.
- *Euclidean distance*: Simplest for continuous vector space.
- *Hamming distance*: Simplest for binary instance space.
  - number of feature values that differ
- For text, cosine similarity of tf.idf weighted vectors is typically most effective.

Illustration of kNN (k=3) for text vector space



3-NN vs. Rocchio

- Nearest Neighbor tends to handle polymorphic categories better than Rocchio/NB.

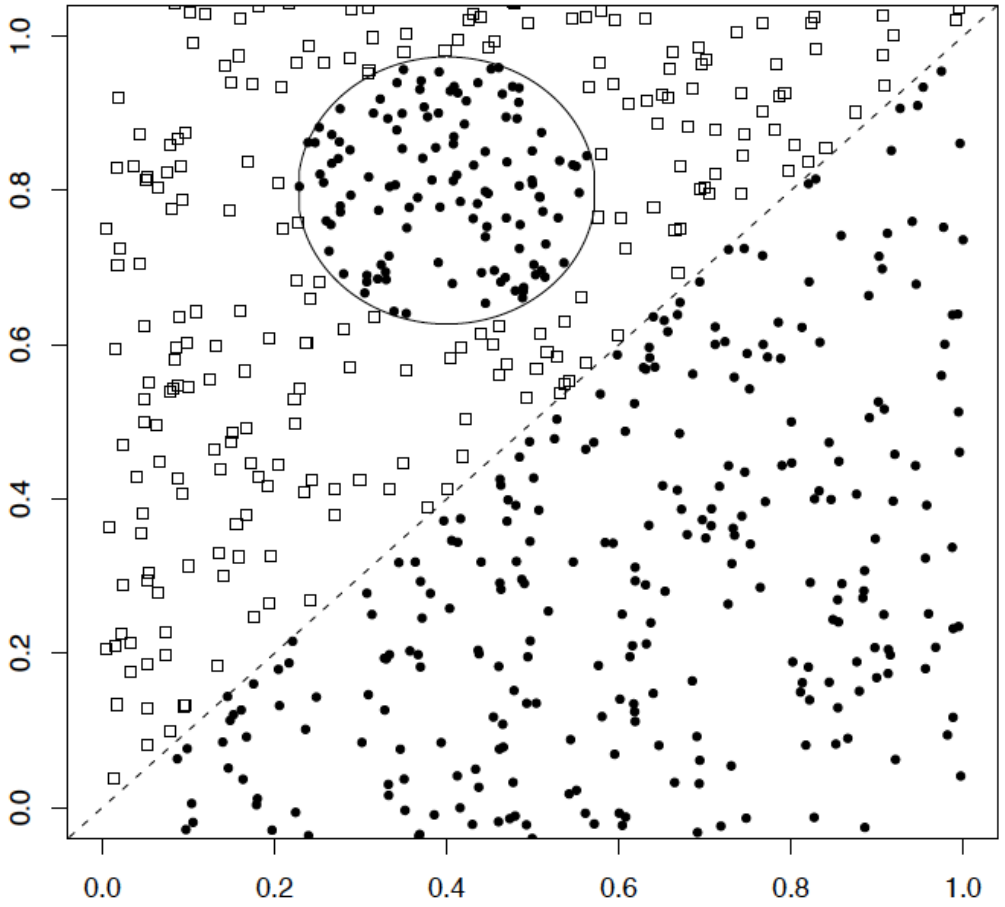


Nearest neighbor with inverted index

- Naively, finding nearest neighbors requires a linear search through $|D|$ docs in collection
  - Similar to determining the k best retrievals using the test doc as a query to a database of training docs.
- Use standard vector space inverted index methods to find the *k* nearest neighbors.
- Testing Time: $O(B|V_t|)$
  - Typically $B << |D|$

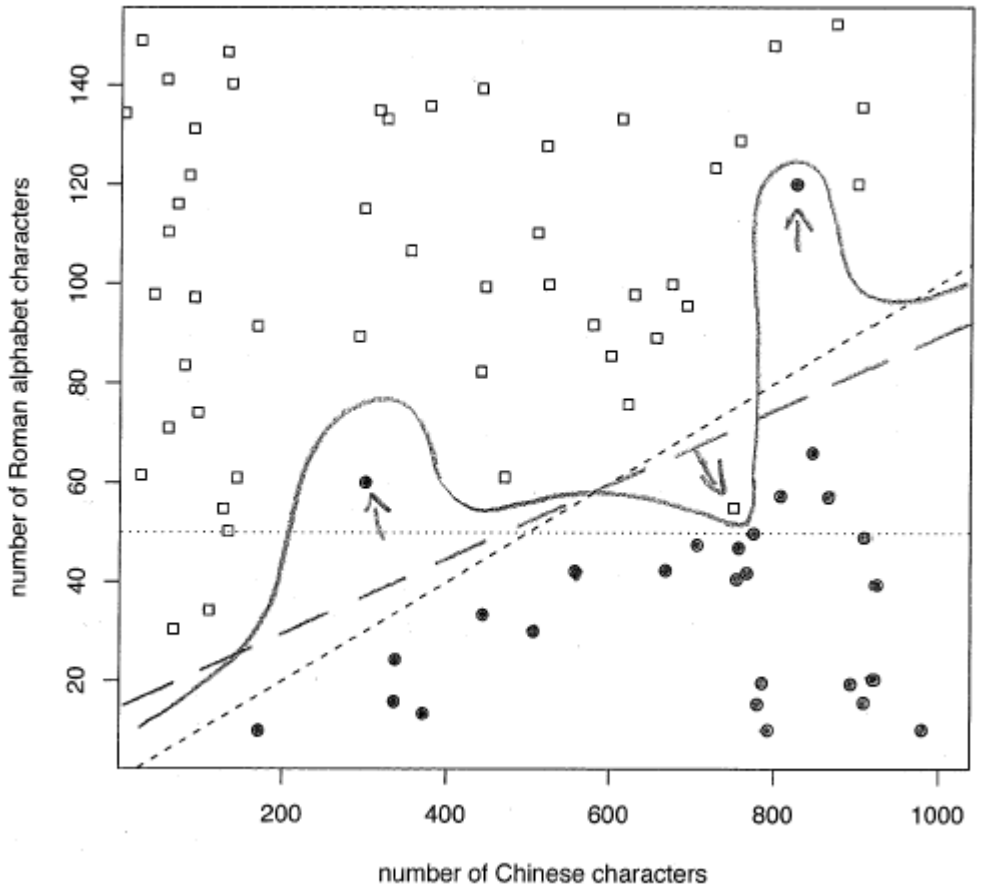
B is the average number of training docs in which a test-document word appears

A nonlinear problem



- Linear classifiers do badly on this task
- kNN will do very well (assuming enough training data)

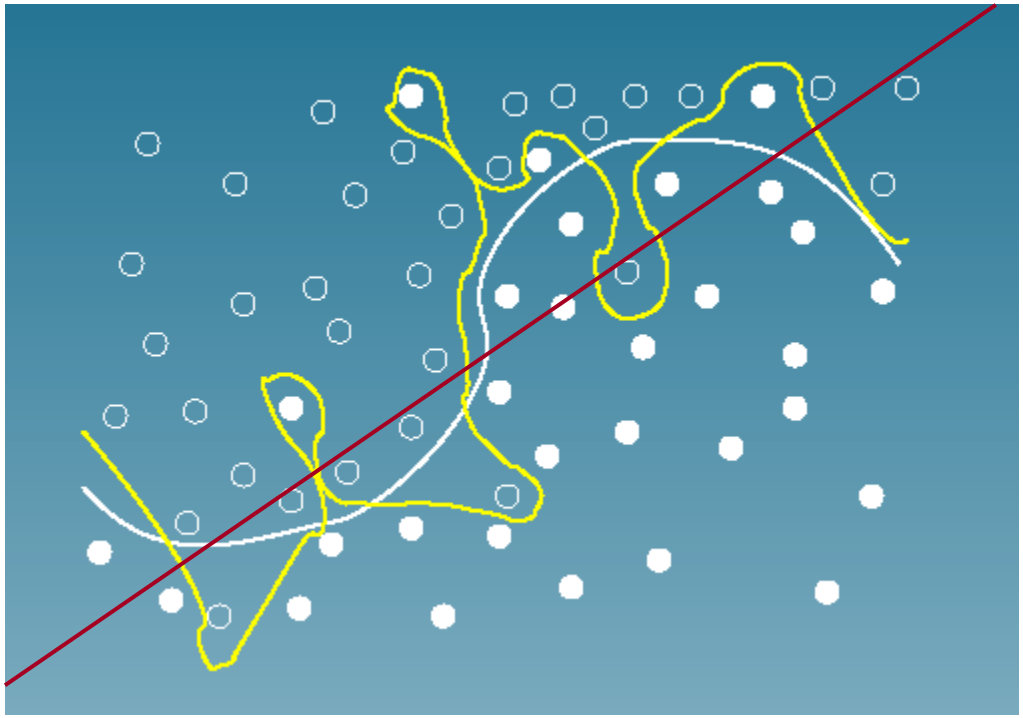
Overfitting example



kNN: summary

- No training phase necessary
  - Actually: We always preprocess the training set, so in reality training time of kNN is linear.
- May be expensive at test time
- kNN is very accurate if training set is large.
  - In most cases it's more accurate than linear classifiers
  - Optimality result: asymptotically zero error if Bayes rate is zero.
- But kNN can be very inaccurate if training set is small.
- Scales well with large number of classes
  - Don't need to train *C* classifiers for *C* classes
- Classes can influence each other
  - Small changes to one class can have ripple effect

Choosing the correct model capacity



Linear classifiers for doc classification

- We typically encounter high-dimensional spaces in text applications.
- With increased dimensionality, the likelihood of linear separability increases rapidly
- Many of the best-known text classification algorithms are linear.
  - More powerful nonlinear learning methods are more sensitive to noise in the training data.
- Nonlinear learning methods sometimes perform better if the training set is large, but by no means in all cases.

Which classifier do I use for a given text classification problem?

- Is there a learning method that is optimal for all text classification problems?
  - No, because there is a tradeoff between complexity of the classifier and its performance on new data points.
- Factors to take into account:
  - How much training data is available?
  - How simple/complex is the problem?
  - How noisy is the data?
  - How stable is the problem over time?
    - For an unstable problem, it's better to use a simple and robust classifier.