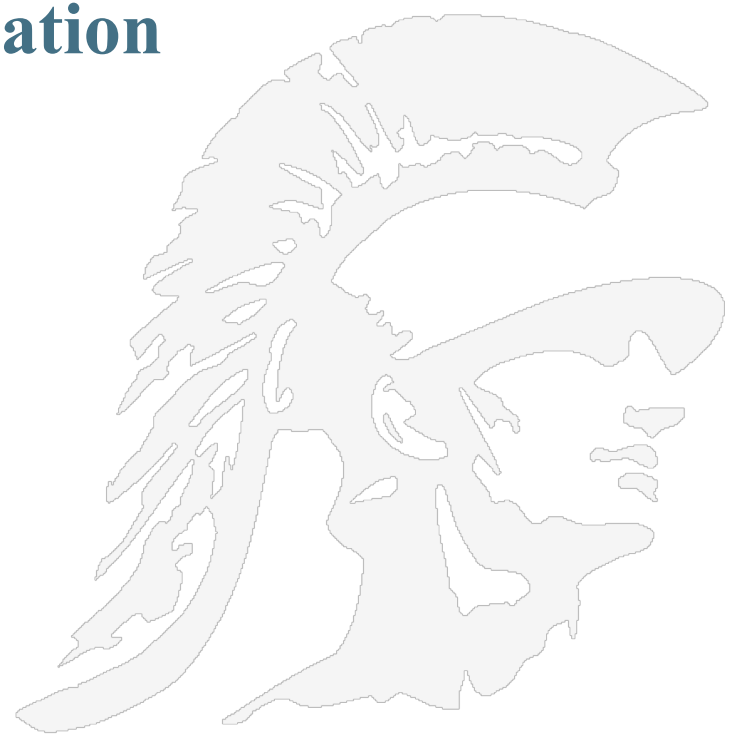
Introduction to

Classification



What Do We Mean By Classification

Assign labels to each document or web-page:

- **Labels are most often topics such as Yahoo-categories**

  e.g., "finance," "sports," "news>world>asia>business"
- **Labels may be genres**

  e.g., "editorials" "movie-reviews" "news"
- **Labels may be opinion**

  e.g., "like", "hate", "neutral"
- **Labels may be domain-specific and binary in choice**

  e.g., "interesting-to-me" : "not-interesting-to-me"

  e.g., "spam": "not-spam"

  e.g., "contains adult language": "doesn't"

Classification is Different from Clustering

- In general, in ***classification*** you have a set of predefined classes and want to know which class a new object belongs to.
- ***Clustering*** tries to group a set of objects and find whether there is *some* relationship between the objects.
  - recall we already saw two algorithms for clustering, K-means Algorithm and Agglomerative Clustering algorithm
- In the context of machine learning, classification is *supervised learning* and clustering is *unsupervised learning*
  - **Clustering** requires a. an algorithm, b. a similarity measure, and c. a number of clusters
  - **classification** has each document labeled in a class and an algorithm that assigns documents to one of the classes

Classification Methods

- **Manual classification**
  - Used by the original Yahoo! Directory
  - **Other search engines did similar things, e.g.**
    - Looksmart, http://www.looksmart.com/
    - Open Directory Project, https://www.dmoz.org/
    - PubMed, http://www.ncbi.nlm.nih.gov/pubmed
  - **Accurate when job is done by experts**
  - **Consistent when the problem size and team is small**
  - **Difficult and expensive to scale**
    - Means we need automatic classification methods for big problems

The Problem Statement for Classification

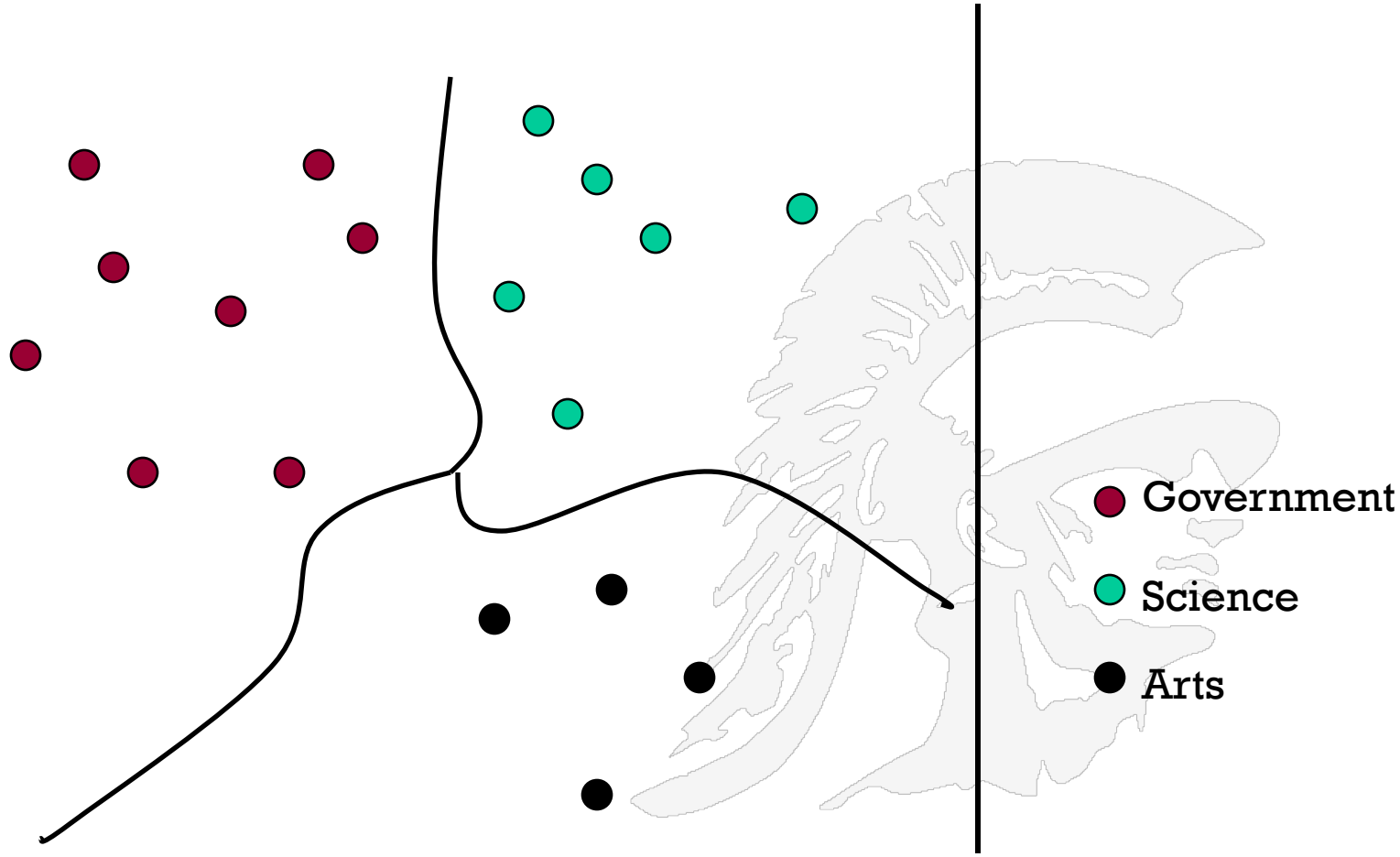
- **Given two things:**
  1. **A description of an instance, $x \in X$, where X is the *instance language* or *instance space*, and**
  2. **A fixed set of categories:**

     $C = \{c_1, c_2, \ldots, c_n\}$
- **Determine:**
  - **The category of x: $c(x) \in C$, where $c(x)$ is a *categorization function* whose domain is X and whose range is C.**
    - **Functions that categorize are called "classifiers"**

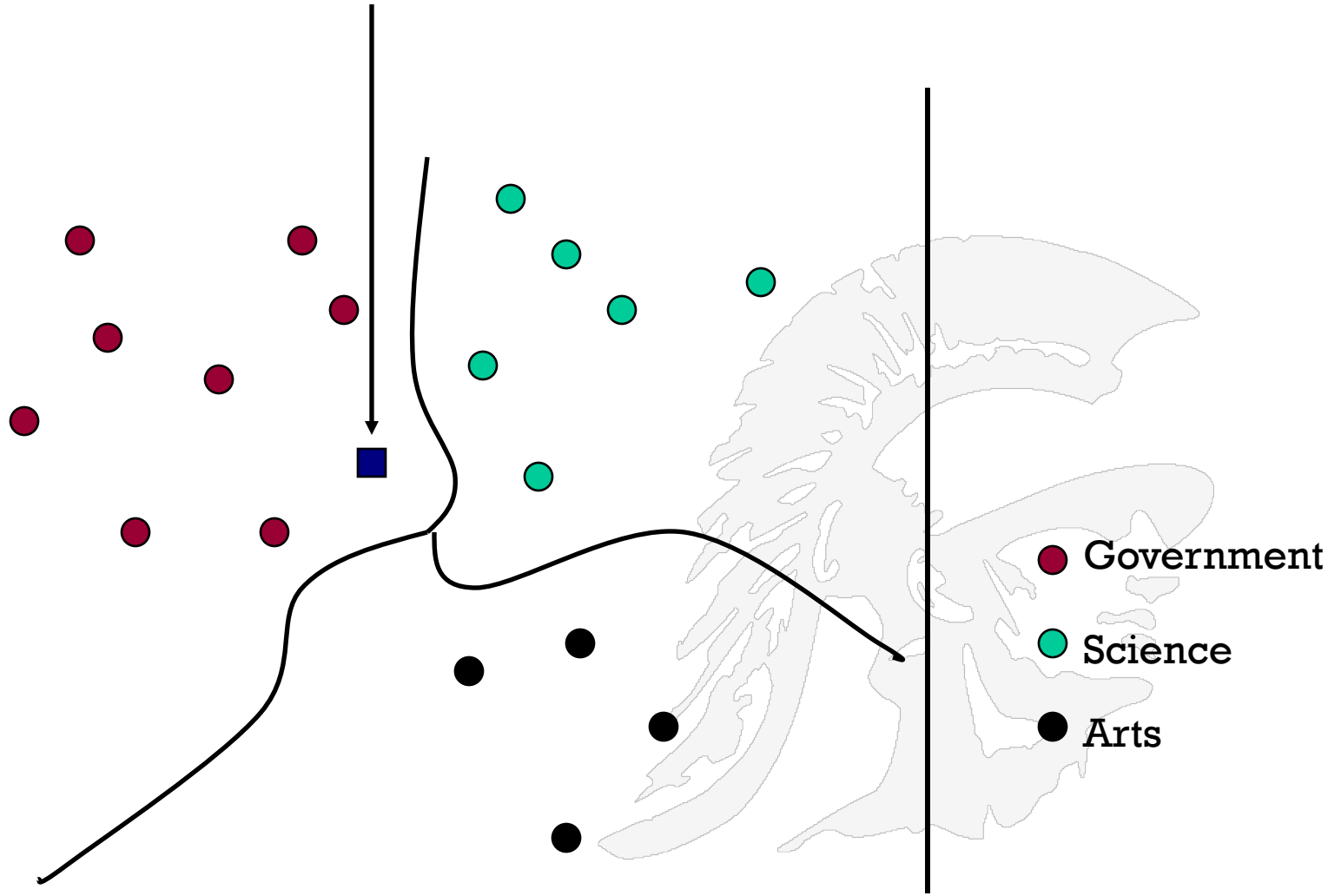
Recall: Vector Space Representation

- **Each document is represented as a vector, one component for each term (= word) in the lexicon**
- **The documents can be viewed as forming a high-dimensional vector space where:**
  - Terms are axes
  - 10,000+ dimensions, or even 100,000+
  - Docs are vectors in this space
- **Normalize the vector elements to unit length and use Euclidean distance as the distance measure**
  - For normalized vectors Euclidean distance and cosine similarity correspond

Example:Three Named Classes in a Vector Space



A New Document is Assigned to the Class Government



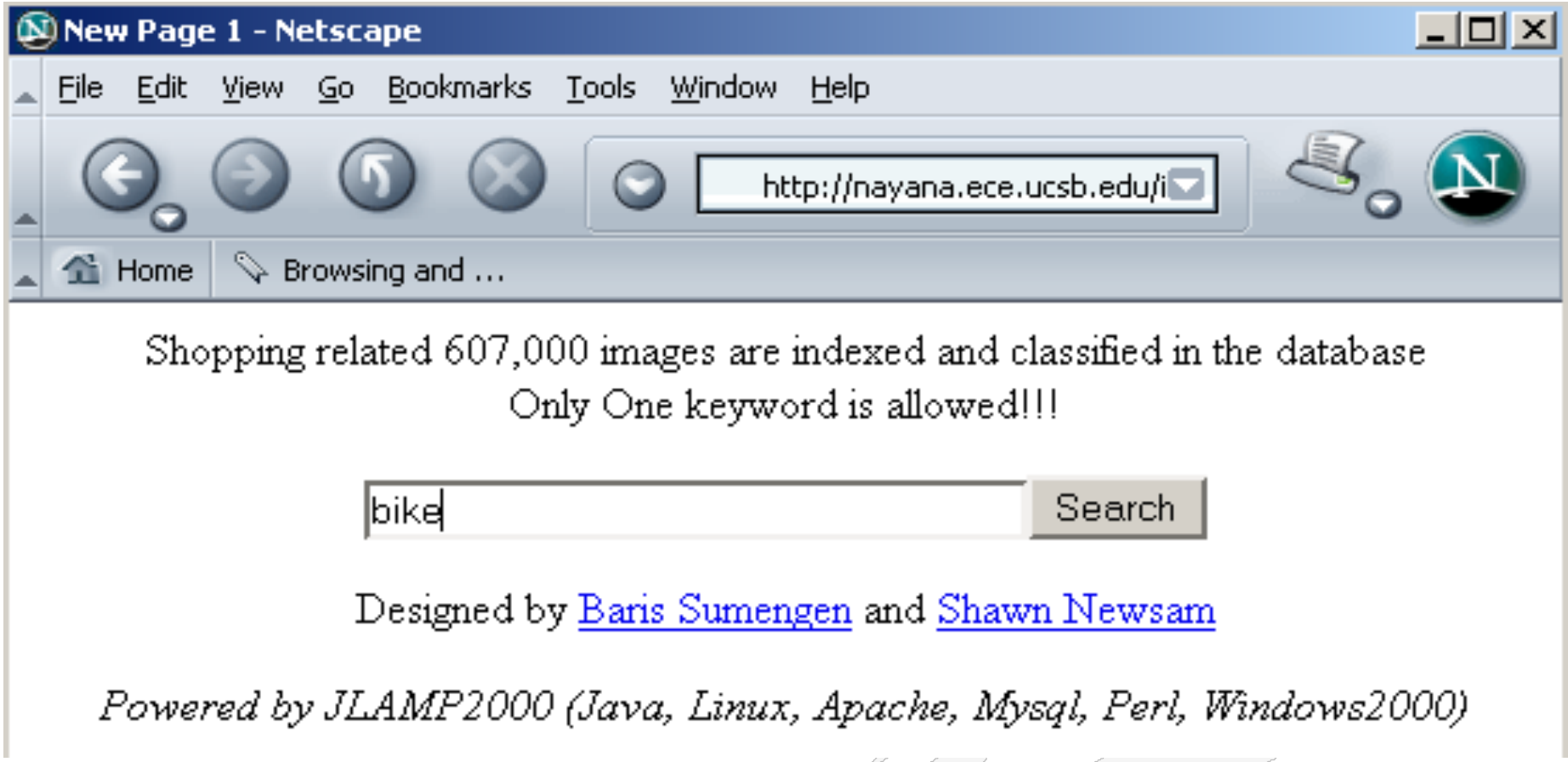
Classification Using Vector Spaces

- **In vector space classification, training set corresponds to a labeled set of points (equivalently, vectors)**
- **Premise 1: Documents in the same class form a contiguous region of space**
  - This is referred to as the *contiguity hypothesis*
- **Premise 2: Documents from different classes don't overlap (much)**
- **Learning a classifier is equivalent to building (defining) surfaces to delineate classes in the space**

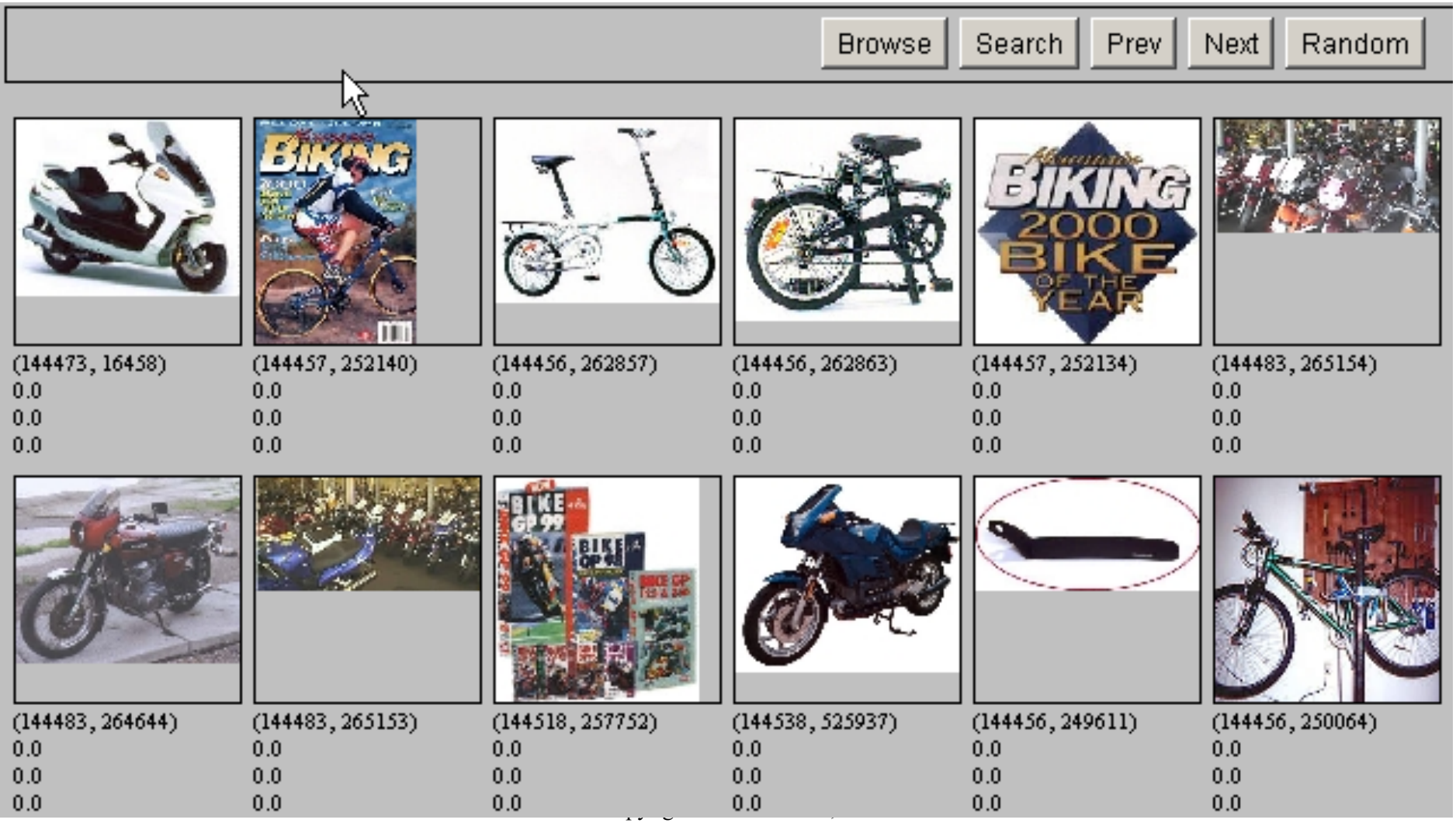
Relevance Feedback

- ***Idea*: It may be difficult to formulate a good query when you don't know the collection well, or cannot express it, but can judge relevance of a result. So iterate …**
- **User feedback on relevance of docs in initial set of results**
  - **User issues a (short, simple) query**
  - **The user marks some results as relevant/non-relevant.**
  - **The system computes a better representation of the information need based on feedback.**
  - **New results have (hopefully) better recall.**
  - **Relevance feedback can go through one or more iterations.**

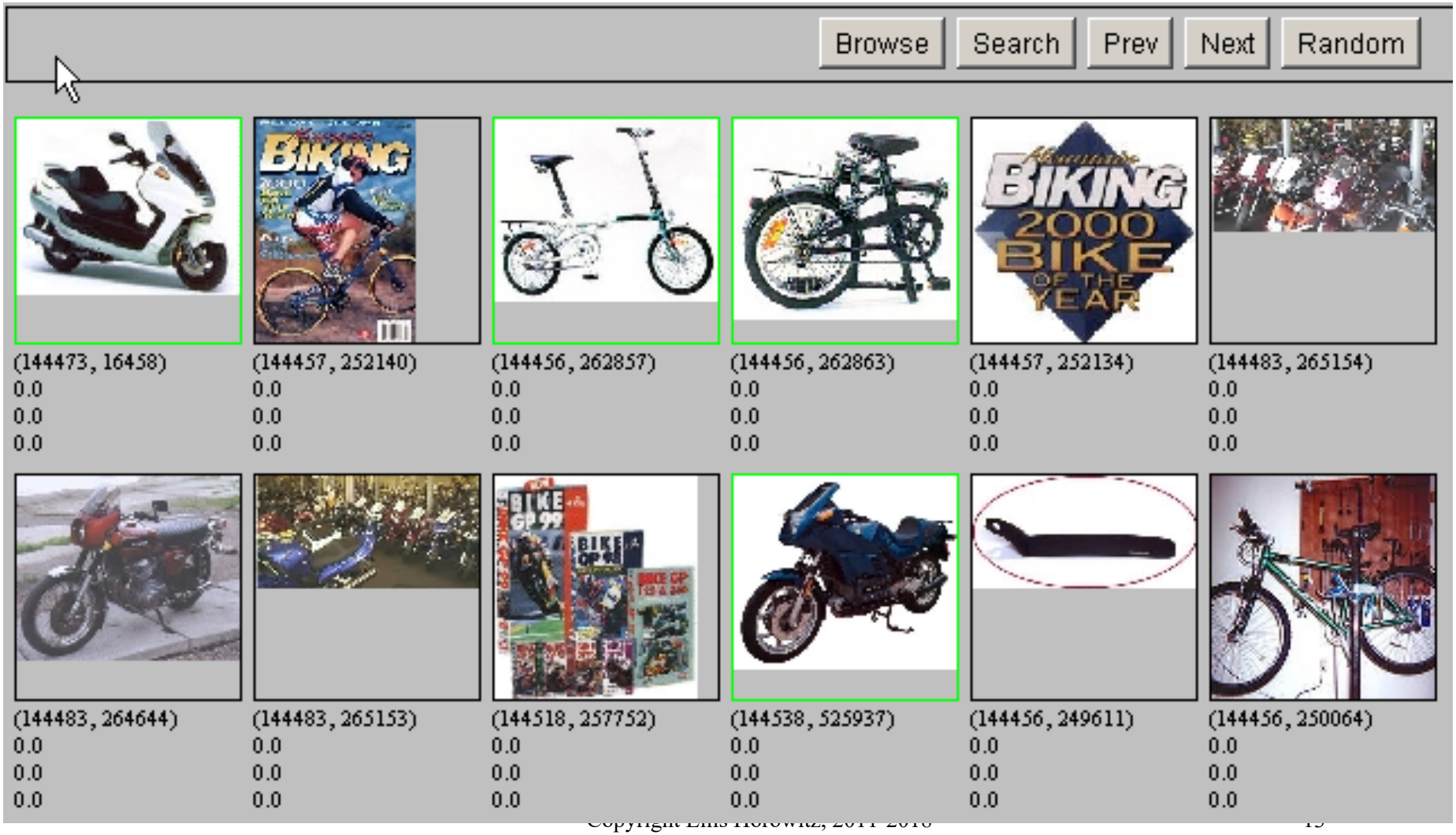
Relevance Feedback Example Initial Query



Relevance Feedback Example
Results for Initial Query



Relevance Feedback
User Identifies Relevant Results



Results after Relevance Feedback

Browse | Search | Prev | Next | Random

(144538, 523493)
0.54182
0.231944
0.309876

(144538, 523835)
0.56319296
0.267304
0.295889

(144538, 523529)
0.584279
0.280881
0.303398

(144456, 253569)
0.64501
0.351395
0.293615

(144456, 253568)
0.650275
0.411745
0.23853

(144538, 523799)
0.66709197
0.358033
0.309059

(144473, 16249)
0.6721
0.393922
0.278178

(144456, 249634)
0.675018
0.4639
0.211118

(144456, 253693)
0.676901
0.47645
0.200451

(144473, 16328)
0.700339
0.309002
0.391337

(144483, 265264)
0.70170796
0.36176
0.339948

(144478, 512410)
0.70297
0.469111
0.233859

Rocchio Method is Used for Relevance Feedback

- **For relevance feedback, Rocchio is limited to *determining two classes*: relevant and non-relevant documents**
- **Classes in Rocchio classification for relevance will have the approximate shape of spheres with similar radii**
- **Since classes are rarely distributed like spheres with similar radii, a modified decision rule is used**

Rocchio Method for Relevance Feedback says:

$$\text{Assign } d \text{ to class } c \text{ iff } |\vec{\mu}(c) - \vec{v}(d)| < |\vec{\mu}(\bar{c}) - \vec{v}(d)| - b$$

Explanation: assign *d* to class *c* iff the distance from *d* to *c* is less than the distance to all other centroids, even including a small factor *b*

General Rocchio Method Key Concept: Centroid

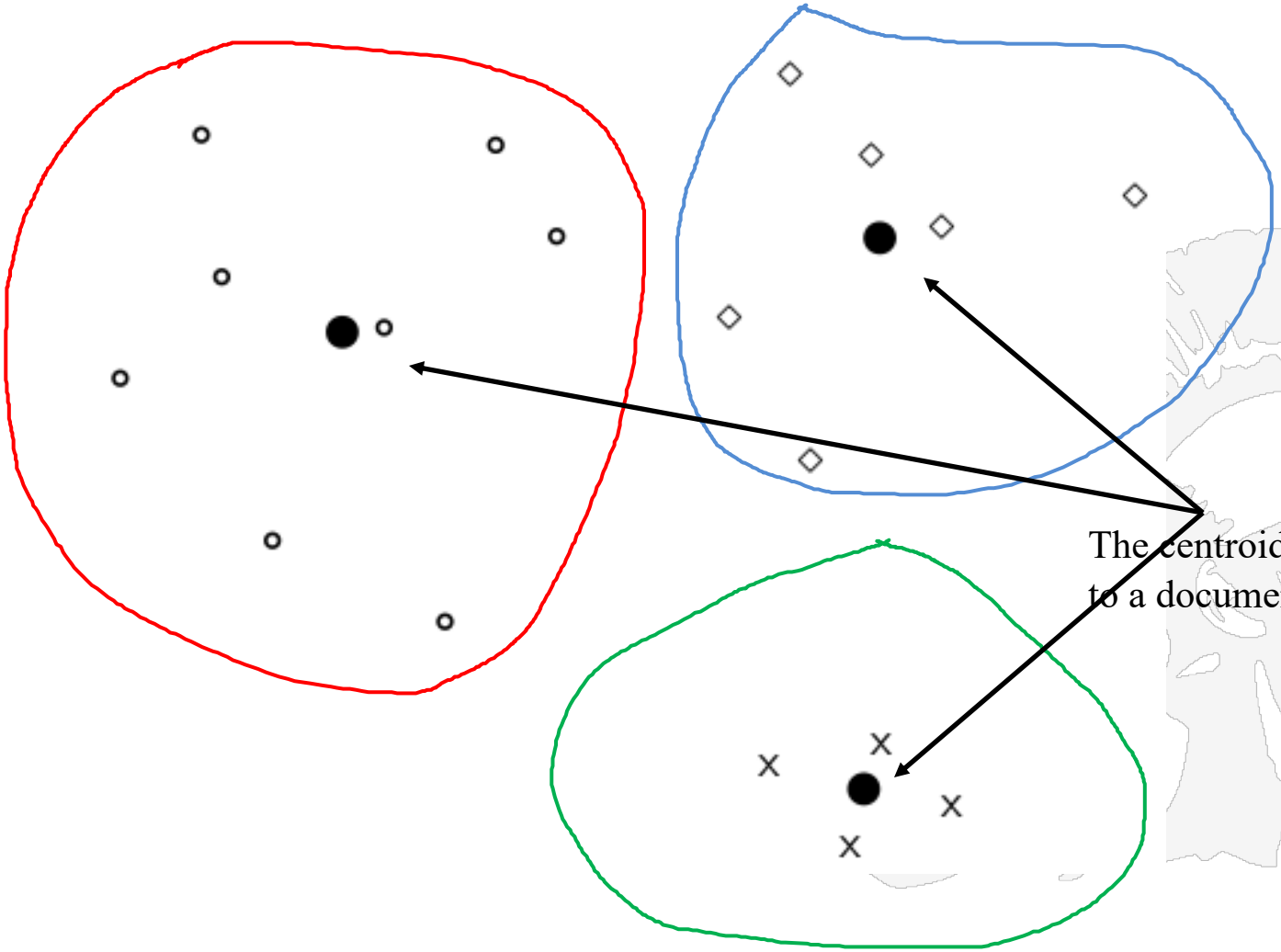
- **The centroid is the center of mass (or vector average) of a set of points.**
- ***Definition*: Centroid**

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

where C is a set of documents, |C| is the size of the set and $\vec{v}(d) = \vec{d}$ is the normalized vector representing document *d*

- We have seen centroids before in the k-means algorithm

General Rocchio Method
Centroid Example



3 regions each with its own centroid

The centroid may not correspond to a document in your set

Rocchio Classification

- **Rocchio chooses the centroid as the representative for each class**
- **Classification Rule:** Given a new document, take its vector representation and determine the nearest centroid
- **Note: This method does not guarantee that classifications are consistent with the given training data**

General Rocchio Method: Boundaries in Rocchio Classification are Hyperplanes

- **Boundaries in Rocchio classification are hyperplanes**
  - **Any line divides a plane into two parts**
  - **Any plane divides a 3D surface in two**
  - **Hyperplanes divide higher dimensional spaces in two**
- **Determine the centroid *u(c)* that the point is closest to and then assign it to *c***
- **Classification rule:** classify a point in accordance with the region it falls into

General Rocchio Algorithm

- ***TrainRocchio(C, D)***

1. **for each c_j *in C***
2. **do $D_j \leftarrow$ *{d: <d, c_j> is in D}***
3. $u_j \leftarrow (1/|D_j|)$ *Sum for d in Dj (v(d))*
4. **return** $\{u_1, \ldots, u_j\}$

APPLYROCCHIO($\{\vec{\mu}_1, \ldots, \vec{\mu}_J\}, d$)
1 **return** $\arg\min_j |\vec{\mu}_j - \vec{v}(d)|$

TRAINROCCHIO takes a set of initial class ids in *C* and a set of documents in *D* and for each class member c_j in *C computes* the set of documents belonging to D_j

$u_1, \ldots u_j$ are the centroids determined by *TRAINROCCHIO*; for each D_j a centroid is computed as the vector average of its members;

APPLYROCCHIO takes the centroids and a new document *d* represented as a vector *v(d)* and the Euclidean distance is computed between each centroid u_j and the vector form of *d*, *v(d);* the centroid with minimum distance is returned;

Note: line 1 in APPLYROCCHIO could be replaced by cosine similarity,
assign d to class c = arg max cos(u(c),v(d))

Computing Time for Rocchio

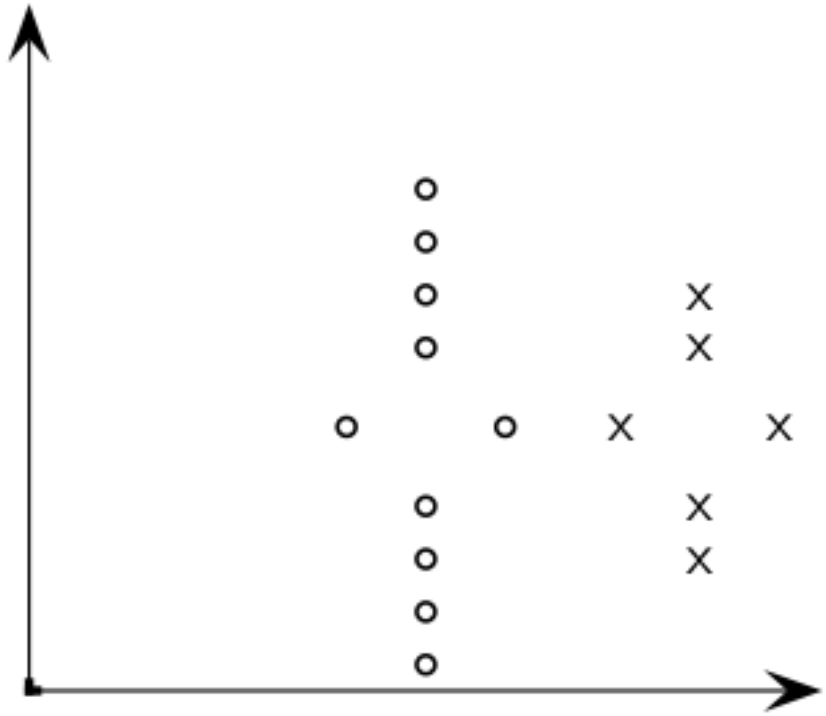
Mode	Complexity
Training	$O(\|D\|L_{ave} + \|C\|\|V\|)$
testing	$O(L_a + \|C\|M_a) = O(\|C\|M_a)$

- L_{ave} is the average number of tokens per document
- L_a and M_a are the numbers of tokens and types respectively in the test document
- The time to compute the Euclidean distance between the class centroid and a document is $O(|C|M_a)$
- Adding all documents to their respective (unnormalized) centroid is $\Theta(|D|L_{ave})$ (as opposed to $\Theta(|D||V|)$) since we need only consider non-zero entries.
- Dividing each vector sum by the size of its class to compute the centroid is $\Theta(|V|)$.

See Table 14.2 in our textbook

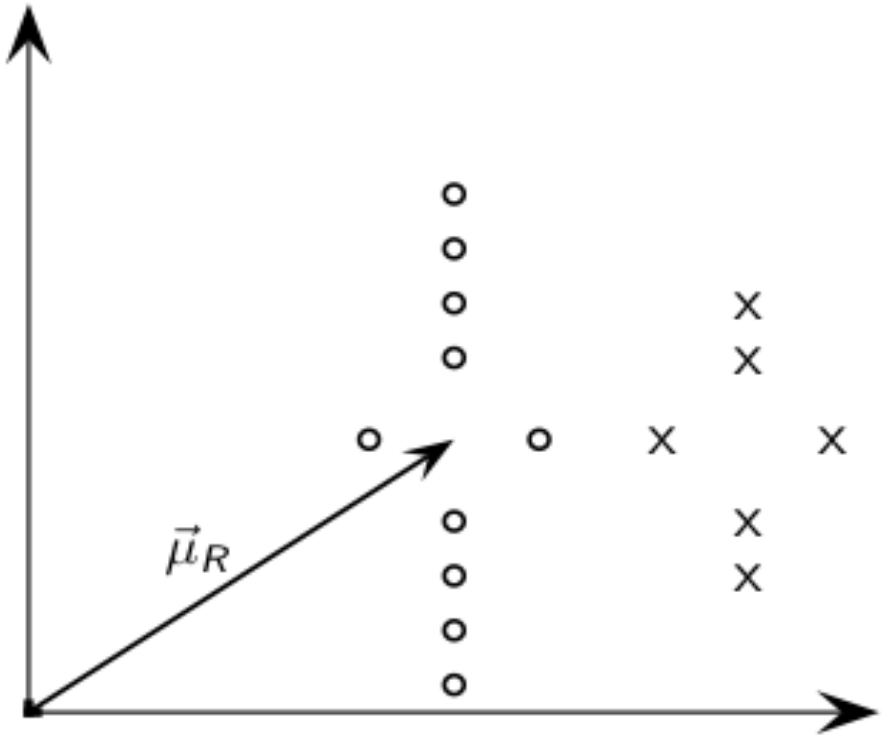
Conclusion: Overall training time is linear in the size of the collection

Example: Compute Rocchio' Vector



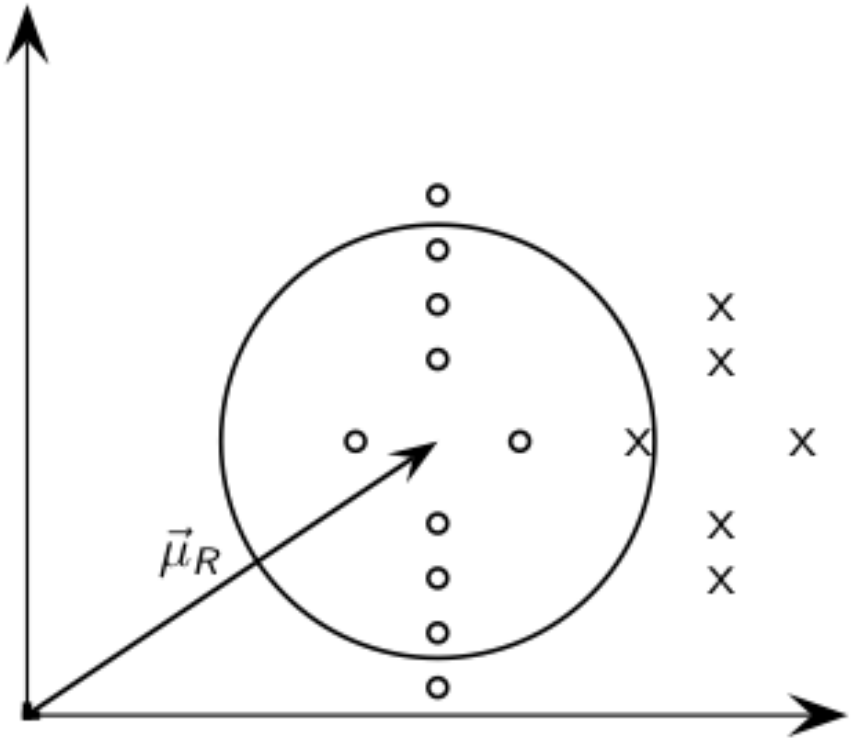
Let circles represent relevant documents,
Let Xs represent nonrelevant documents

Rocchio' illustrated (1 of 9)



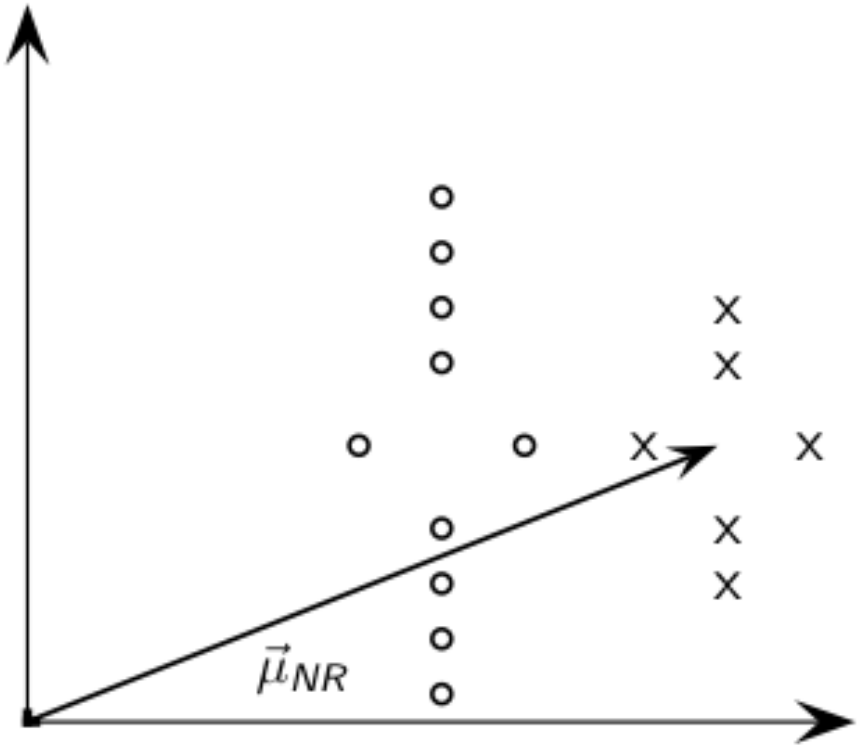
$\vec{\mu}_R$: centroid of relevant documents

Rocchio' illustrated (2 of 9)



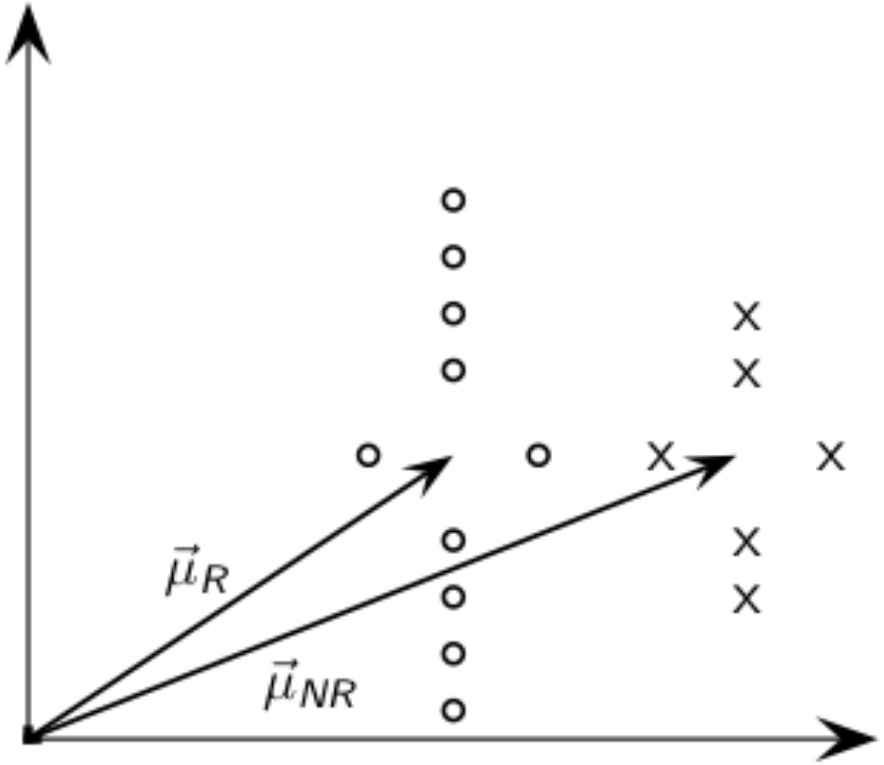
$\vec{\mu}_R$ does not separate relevant / nonrelevant.

Rocchio' illustrated (3 of 9)

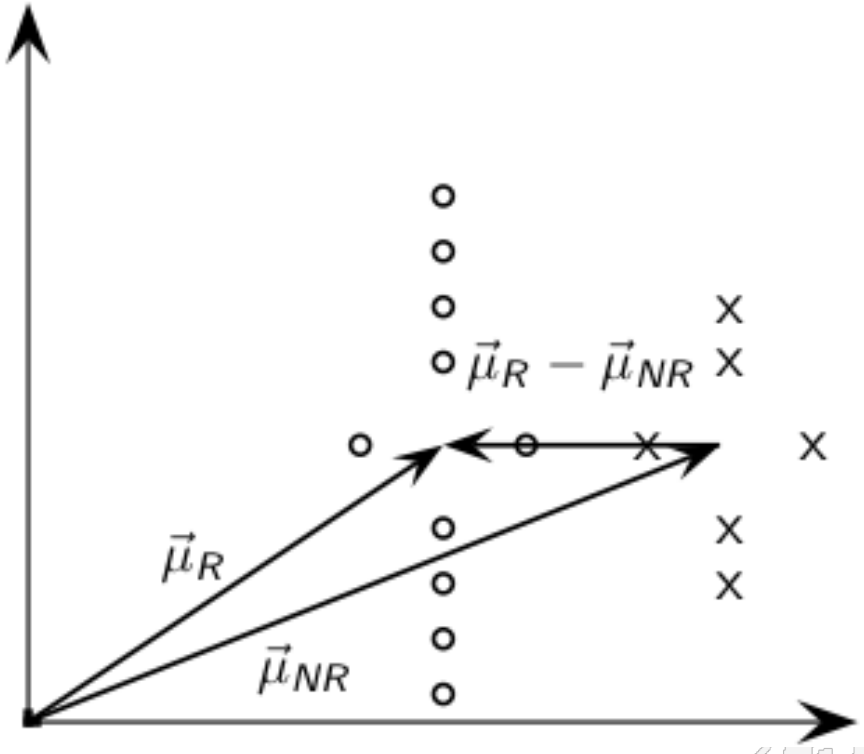


$\vec{\mu}_{NR}$: centroid of nonrelevant documents.

Rocchio' illustrated (4 of 9)

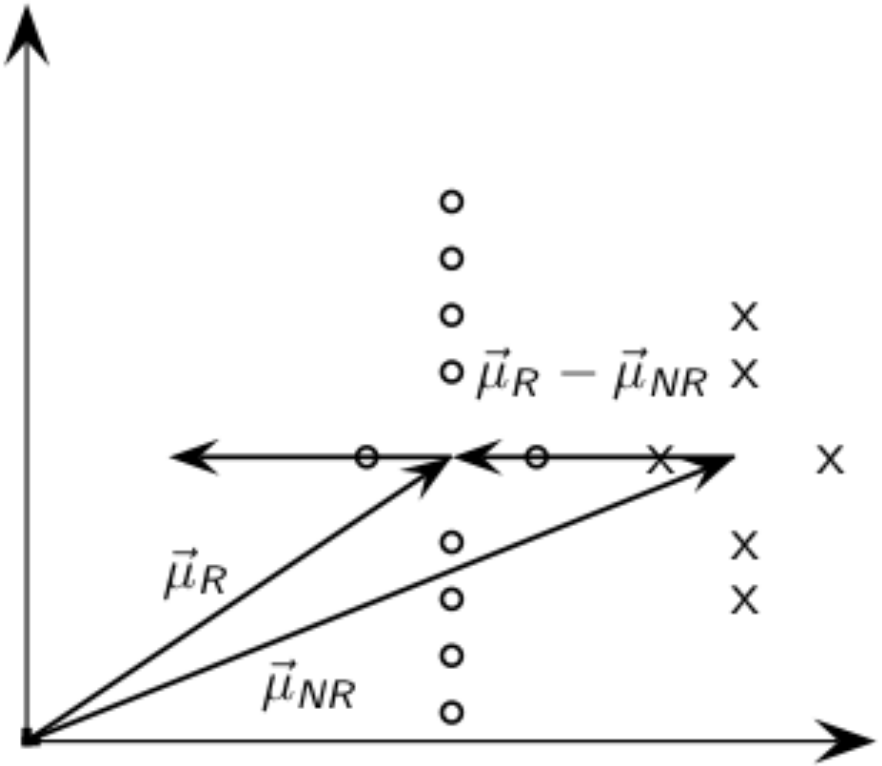


Rocchio' illustrated(5 of 9)



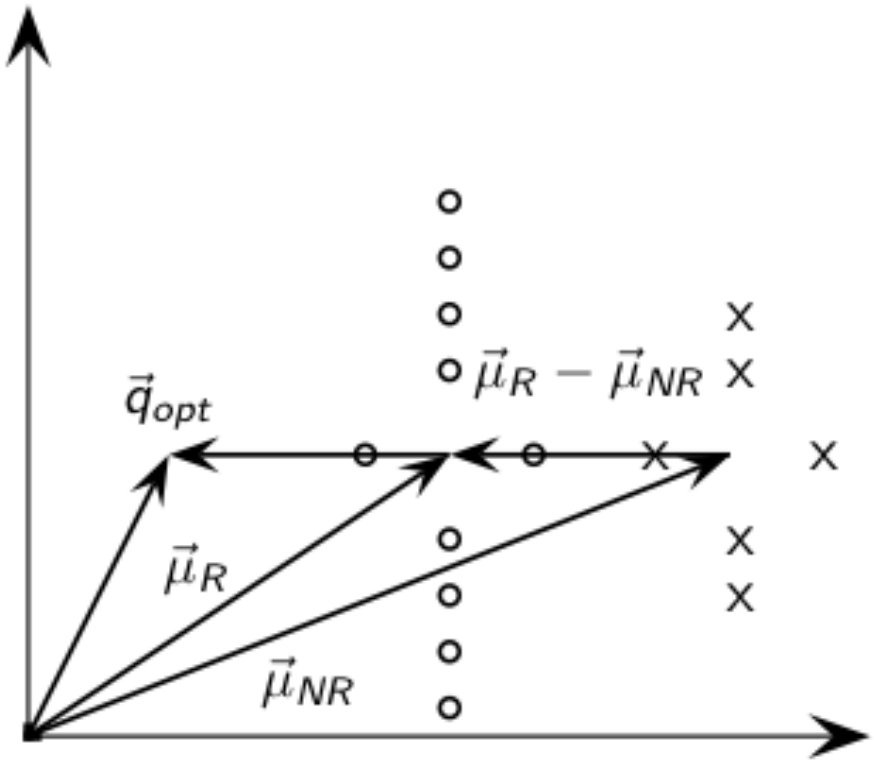
$\vec{\mu}_R$ - $\vec{\mu}_{NR}$: difference vector

Rocchio' illustrated(6 of 9)



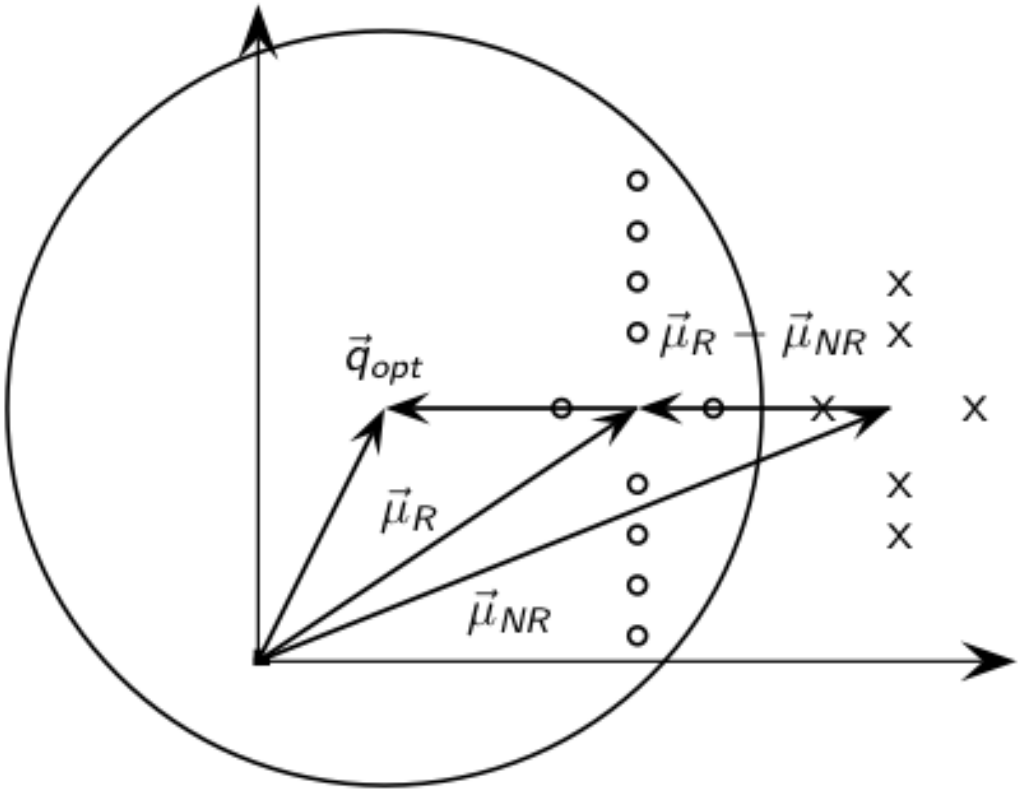
Add difference vector to $\vec{\mu}_R$ …

Rocchio' illustrated(7 of 9)



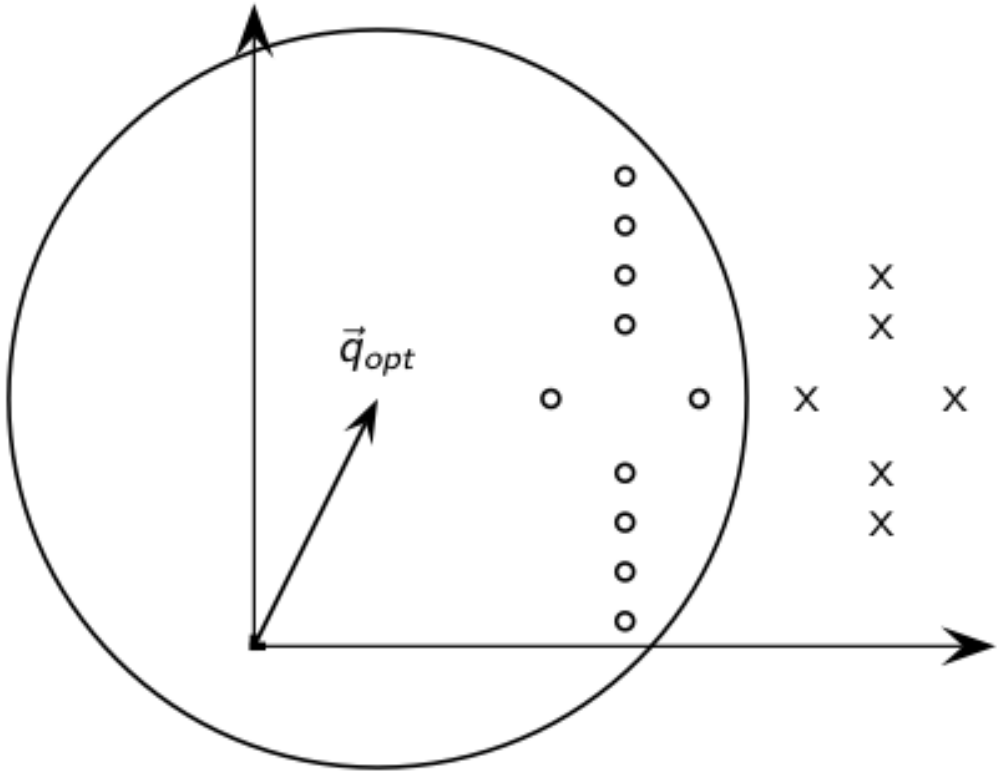
… to get $\vec{q}_{opt}$

Rocchio' illustrated(8 of 9)



$\vec{q}_{opt}$ now separates relevant / nonrelevant perfectly.

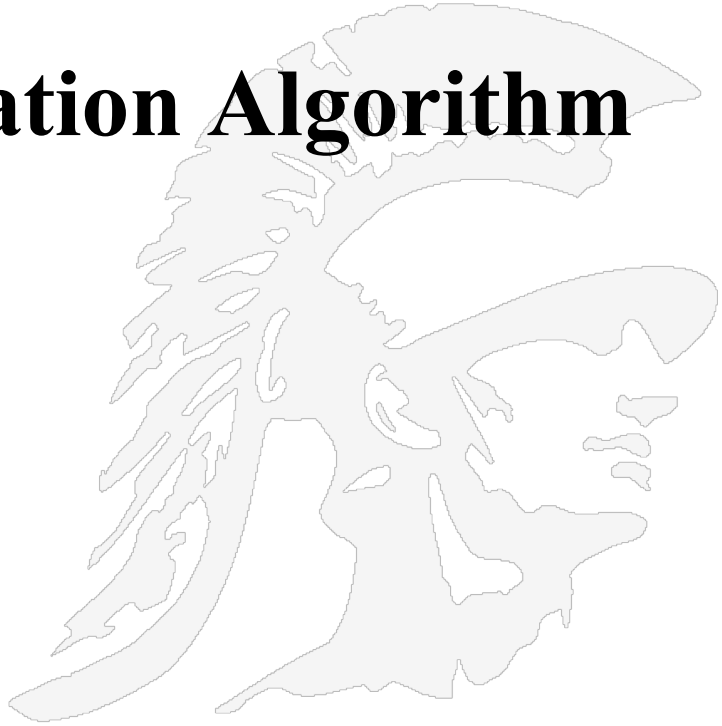
Rocchio' illustrated(9 of 9)



$\vec{q}_{opt}$ separates relevant / nonrelevant perfectly.

kNN - *k* Nearest Neighbor Method

Second Classification Algorithm

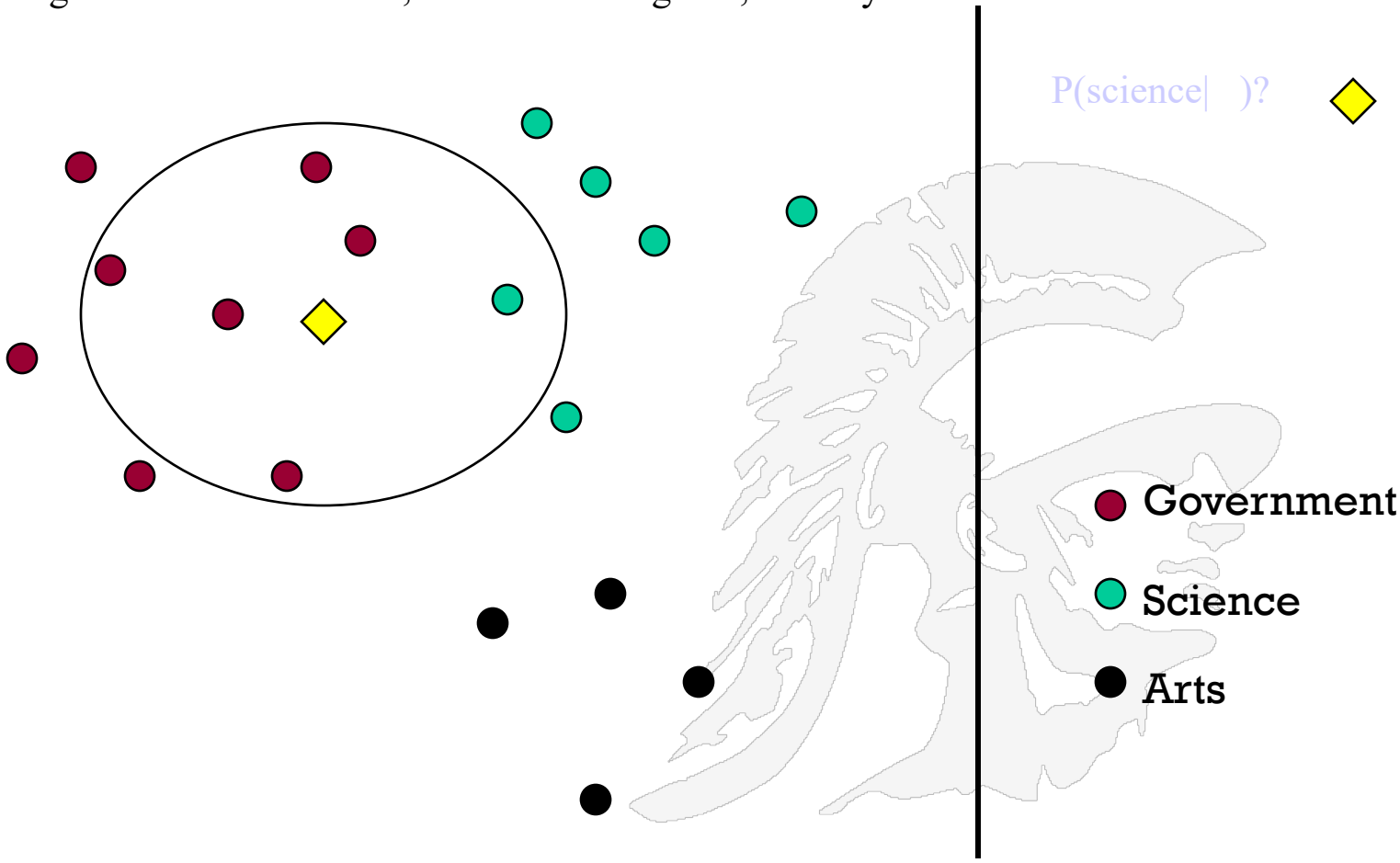


k Nearest Neighbor Classification Algorithm

- **Initially we assume we have a set of *N* documents that have already been classified**
  - the WDM videos assume the class names are colors (see the Schedule of Lectures)
- **To classify a document *d***
  - locate among the *N* documents the *k* closest ones
  - from these *k* neighbors, pick the class that occurs most often, the majority class, and use that as the label for *d*

Example: *k=6 (6NN)*

5 neighbors are colored red, one is colored green, so the yellow diamond is colored red



Distance Function Measurements

Distance functions

Euclidean $$\sqrt{\sum_{i=1}^{k}(x_i - y_i)^2}$$

Manhattan $$\sum_{i=1}^{k}|x_i - y_i|$$

Minkowski $$\left(\sum_{i=1}^{k}(|x_i - y_i|)^q\right)^{1/q}$$

Nearest-Neighbor Non-Learning

- **Learning: there is no learning step; just store the labeled training examples *D***
- **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**
  - if documents do form contiguous regions in space, then *kNN* makes sense
- **Does not compute anything beyond storing the examples**
  - we are NOT determining any hierarchy or model
- **K-NN has also been called:**
  - Case-based learning
  - Memory-based learning
  - Lazy learning

The Algorithm

- The training examples are vectors in a multidimensional feature space, each with a class label
- The ***training phase*** of the algorithm consists only of storing the feature vectors and class labels of the training samples
- In the ***classification phase***, *k* is a user-defined constant, and an unlabeled vector (a query) is classified by assigning the label which is most frequent among the *k* training samples nearest to that query point.
- A commonly used distance metric for this algorithm is Euclidean distance.

The Algorithm

- **Given a set of points each of which has a class indicator and given an integer *k*, and a new point, then**
  1. compute the distance between the new point and all other points
  2. determine the *k* closest points to the new point
  3. examine the class indicators of the *k* closest points and choose the class indicator that occurs most often

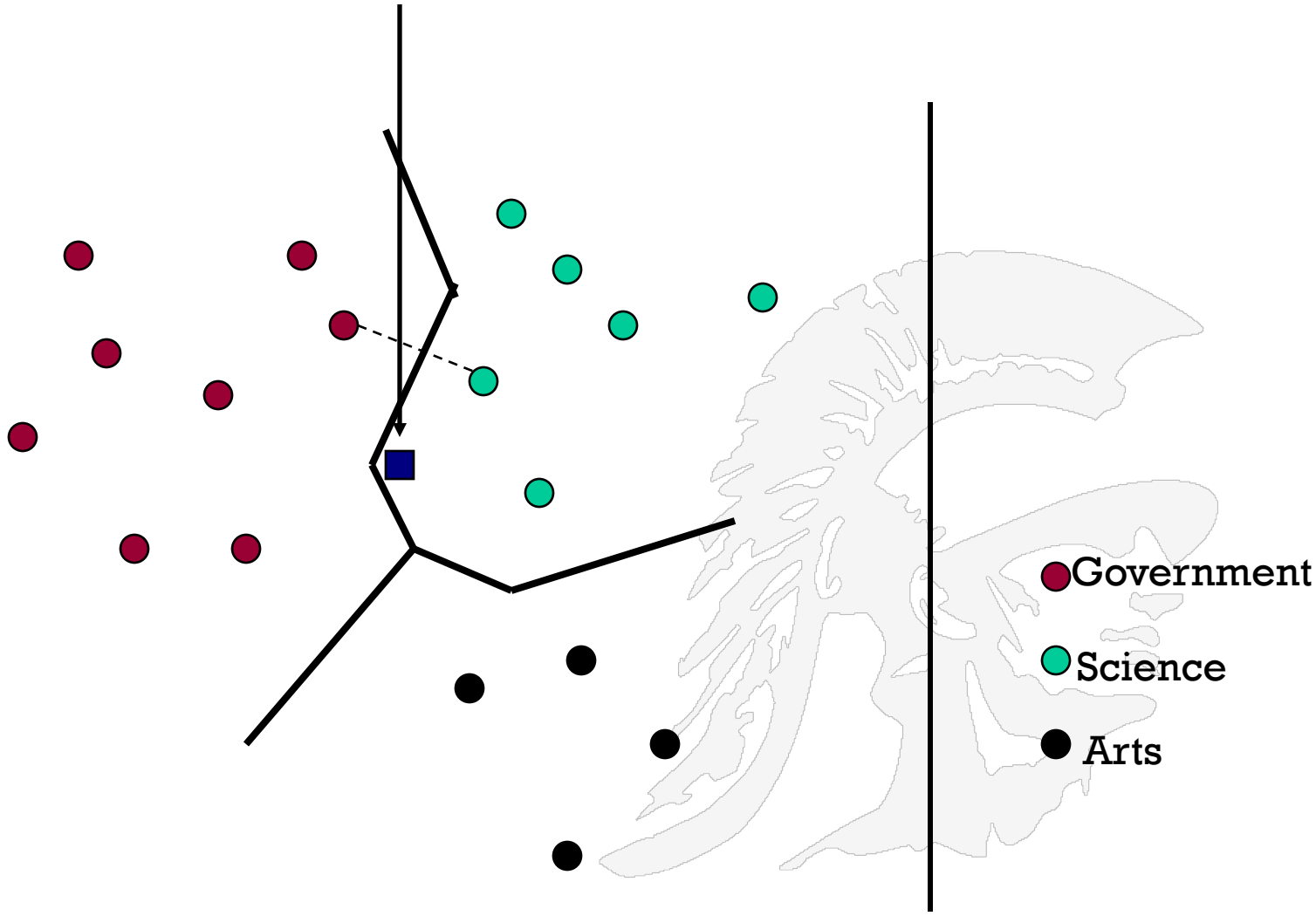
Choice of *K*

- **The best choice of *k* depends upon the data;**
  - generally, larger values of *k* reduce the effect of noise on the classification, but make boundaries between classes less distinct.
  - The special case where the class is predicted to be the class of the closest training sample (i.e. when $k = 1$) is called the nearest neighbor algorithm.
- The accuracy of the *k-NN* algorithm can be severely degraded by the presence of noisy or irrelevant features, or if the feature scales are not consistent with their importance
- In binary (two class) classification problems, it is helpful to choose *k* to be an odd number as this avoids tied votes

Choosing the most suitable K

- Choosing the optimal value for K is best done by first inspecting the data
- In general, a large K value is more precise as it reduces the overall noise but there is no guarantee
- Cross-validation is another way to retrospectively determine a good K value by using an independent dataset to validate the K value
- Historically, the optimal K for most datasets has been between 3-10. That produces much better results than 1NN

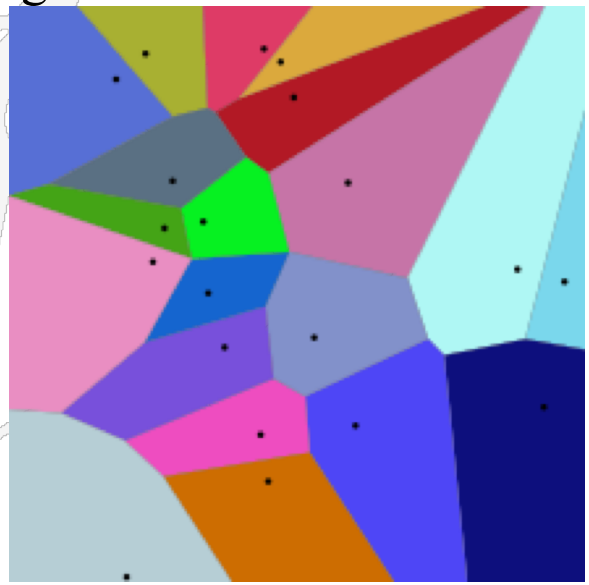
Test Document Assigned to Science



Voroni Diagram

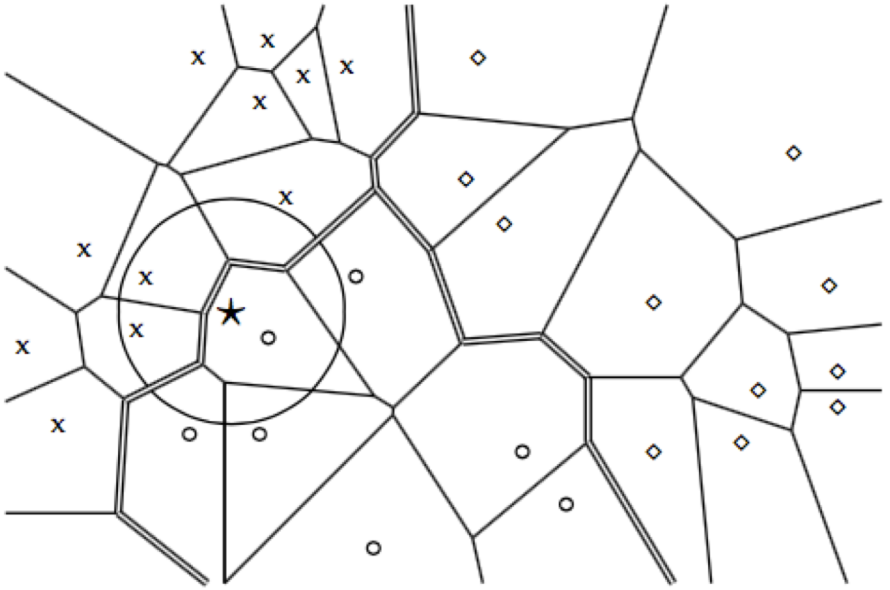
- Decision boundaries in 1NN are concatenated segments of a Voronoi tessellation
- A **Voronoi diagram** is a partitioning of a plane into regions based on distance to points in a specific subset of the plane
- The set of points (called class labels, seeds, sites, or generators) is specified beforehand
- For each class label there is a corresponding region consisting of all points closer to that class label than to any other. These regions are called Voronoi cells

20 points (class labels) and their Voroni regions; Line segments are all points equidistant to three or more regions



K Nearest Neighbor Regions are Polygons

- **For *1NN* we assign each document to the class of its closest neighbor**
- **For *kNN* we assign each document to the majority class of its *k* closest neighbors where *k* is a parameter**



The two classes are: X and circle, and the star document is falling into the circle area;
Double lines define the regions in space where documents are similar;
think of each region as defining a cellphone tower

***kNN* is an example of a non-linear classifier; (Rocchio is a linear classifier)**

Computing Times

- **Training a *kNN* classifier simply consists of determining *k* and pre-processing the documents**
- **If we preselect *k* and do not pre-process, then *kNN* requires no training at all**
- **It makes more sense to pre-process training documents once as part of the training phase rather than repeatedly every time we classify a new test document**

kNN with preprocessing of training set

training $\Theta(|\mathbb{D}|L_{ave})$

testing $\Theta(L_a + |\mathbb{D}|M_{ave}M_a) = \Theta(|\mathbb{D}|M_{ave}M_a)$

kNN without preprocessing of training set

training $\Theta(1)$

testing $\Theta(L_a + |\mathbb{D}|L_{ave}M_a) = \Theta(|\mathbb{D}|L_{ave}M_a)$

Recall from previous slide: L_{ave} is the average number of tokens per document
L_a and M_a are the numbers of tokens and types respectively in the test document

Observations on K-NN

- **The K-Nearest-Neighbor model has two major drawbacks**.

1. **performance**.
   - you have to load all of your training data and calculate distances to all training samples; Over a big training set, this is a major drawback.
2. **distance metric**. Using Euclidean distance can be a problem in high dimensions as well as with noisy features

- Different variants of k-NN are used in practice
  - Neighborhood Components Analysis
  - https://en.wikipedia.org/wiki/Neighbourhood_components_analysis
- **To summarize**, if you have a system that has to learn a sophisticated (i.e. nonlinear) pattern with a small number of samples (and usually dimensions), K-NN models are usually one of the best options out there.

kNN: Final Points

- **No feature selection necessary**
- **No training necessary**
- **Scales well with large number of classes**
  - Don't need to train *n* classifiers for *n* classes
- **Classes can influence each other**
  - Small changes to one class can have ripple effect
- **In most cases it's more accurate than Rocchio**

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 bias and variance.**
- **Factors to take into account:**
  - How much training data is available?
  - How simple/complex is the problem? (linear vs. nonlinear decision boundary)
  - 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.