CS 349:Categorization: Similarity, VSM, K-NN etc.

Text Categorization Applications

- Web pages
  - Recommending
  - Yahoo-like classification
- Newsgroup/Blog Messages
  - Recommending
  - spam filtering
  - Sentiment analysis for marketing
- News articles
  - Personalized newspaper
- Email messages
  - Routing
  - Prioritizing
  - Folderizing
  - spam filtering
  - Advertising on Gmail

Textual Similarity Metrics

- Measuring similarity of two texts: a well-studied problem
- Standard metrics:
  - *bag of words* model of a document that ignores *word order* and *syntactic structure*
- May involve removing common “*stop words*” and *stemming* to reduce words to their root form
- *Vector-space model* from Information Retrieval (IR) is the standard approach
- Other metrics (e.g. *edit-distance*) are also used

The Vector-Space Model

- Assume t distinct terms remain after *preprocessing*; call them *index terms or the vocabulary*
- These "orthogonal" terms form a vector space

  Dimension = t = |vocabulary|
- Each term, i, in a document or query, j, is given a real-valued weight, w_{ij}.
- Both documents and queries are expressed as t-dimensional vectors:

  $d_j = (w_{1j}, w_{2j}, \ldots, w_{tj})$

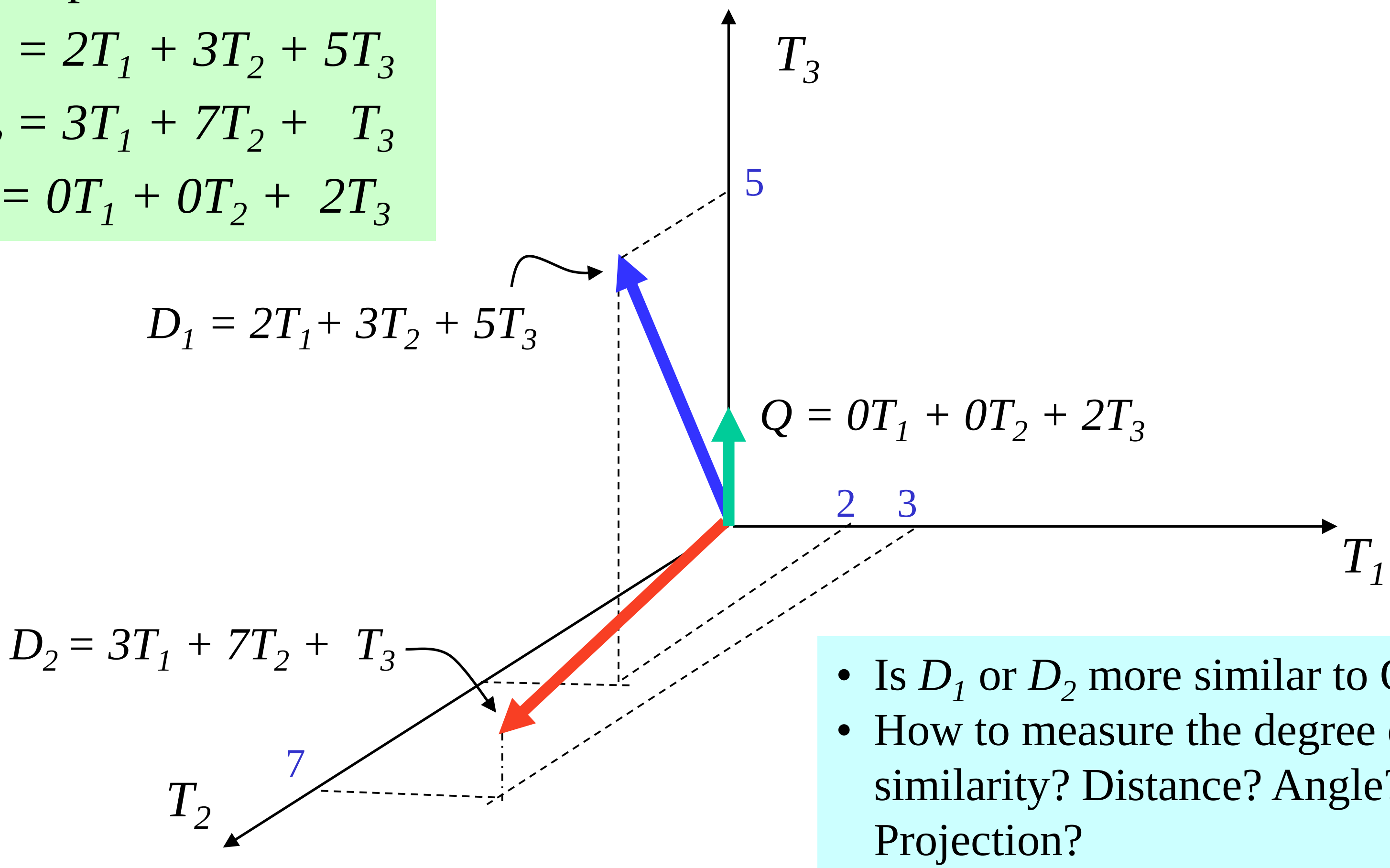
Graphic Representation

Example:

$D_1 = 2T_1 + 3T_2 + 5T_3$

$D_2 = 3T_1 + 7T_2 + T_3$

$Q = 0T_1 + 0T_2 + 2T_3$



- Is D_1 or D_2 more similar to Q?
- How to measure the degree of similarity? Distance? Angle? Projection?

Document Collection

- A collection of n documents can be represented in the vector space model by a term-document matrix
- An entry in the matrix corresponds to the "weight" of a term in the document; zero means the term has no significance in the document or it simply doesn't exist in the document

$$
\begin{pmatrix}
 & T_1 & T_2 & \dots & T_t \\
D_1 & w_{11} & w_{21} & \dots & w_{t1} \\
D_2 & w_{12} & w_{22} & \dots & w_{t2} \\
\vdots & \vdots & \vdots & & \vdots \\
\vdots & \vdots & \vdots & & \vdots \\
D_n & w_{1n} & w_{2n} & \dots & w_{tn}
\end{pmatrix}
$$

Term Weights: Term Frequency

- More frequent terms in a document are more important, i.e. more indicative of the topic

  f_{ij} = frequency of term i in document j

- May want to normalize *term frequency* (*tf*) dividing by the frequency of the most common term in the document:

  $tf_{ij} = f_{ij} / max_i\{f_{ij}\}$

Term Weights: Inverse Document Frequency

- Terms that appear in many *different* documents are *less* indicative of overall topic

  df_i = document frequency of term i

  = number of documents containing term i

  idf_i = inverse document frequency of term i,

  $= \log_2 (N/ df_i)$

  (N: total number of documents)

- An indication of a term's *discrimination* power
- Log used to dampen the effect relative to *tf*

TF-IDF Weighting

- A typical combined term importance indicator is *tf-idf weighting*:

$$w_{ij} = tf_{ij}\, idf_i = tf_{ij} \log_2 (N/\, df_i)$$

- A term occurring frequently in the document but rarely in the rest of the collection is given high weight
- Many ways exist for determining term weights
- Experimentally, *tf-idf* has been found to work well

Computing TF-IDF -- An Example

Given a document containing terms with given frequencies:

A(3), B(2), C(1)

Assume collection contains 10,000 documents and document frequencies of these terms are:

A(50), B(1300), C(250)

Then:

A: tf = 3/3; idf = log(10000/50) = 5.3; tf-idf = 5.3

B: tf = 2/3; idf = log(10000/1300) = 2.0; tf-idf = 1.3

C: tf = 1/3; idf = log(10000/250) = 3.7; tf-idf = 1.2

Similarity Measure

- A similarity measure is a function that computes the *degree of similarity* between two vectors

- Using a similarity measure between the query and each document:
  - It is possible to rank the retrieved documents in the order of presumed relevance
  - It is possible to enforce a certain threshold so that the size of the retrieved set can be controlled

Similarity Measure - Inner Product

- Similarity between vectors for the document $\boldsymbol{d}_j$ and query $\boldsymbol{q}$ can be computed as the vector inner product:

$$\text{sim}(\boldsymbol{d}_j,\boldsymbol{q}) = \boldsymbol{d}_{\text{j}}\bullet\boldsymbol{q} = \sum_{i=1}^{t} w_{ij} \cdot w_{iq}$$

where w_{ij} is the weight of term i in document j and w_{iq} is the weight of term i in the query

- *For binary vectors*: inner product is the number of matched query terms in the document (size of intersection)

- *For weighted term vectors*: sum of the products of the weights of the matched terms

Properties of Inner Product

- The inner product is unbounded
- Favors long documents with a large number of unique terms
- Measures how many terms *matched* but not how many terms are *not matched*

Inner Product -- Examples

Binary:

	retrieval	database	architecture	computer	text	management	information
– D =	1,	1,	1,	0,	1,	1,	0
– Q =	1,	0 ,	1,	0,	0,	1,	1

Size of vector = size of vocabulary = 7

0 means corresponding term not found in document or query

sim(D, Q) = 3

Weighted:

$D_1 = 2T_1 + 3T_2 + 5T_3$ $D_2 = 3T_1 + 7T_2 + 1T_3$

$Q = 0T_1 + 0T_2 + 2T_3$

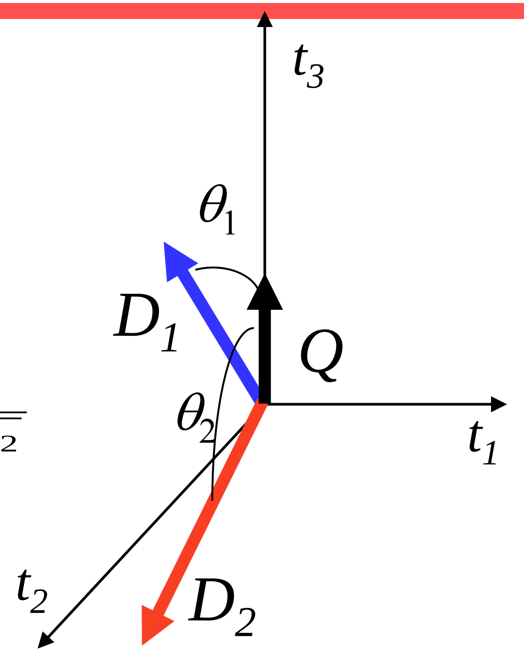
sim(D_1 , Q) = 2*0 + 3*0 + 5*2 = 10

sim(D_2 , Q) = 3*0 + 7*0 + 1*2 = 2

Cosine Similarity Measure

- Cosine similarity measures the cosine of the angle between two vectors
- Inner product normalized by the vector lengths

$$\text{CosSim}(\boldsymbol{d}_j, \boldsymbol{q}) = \frac{\vec{d}_j \cdot \vec{q}}{\left|\vec{d}_j\right| \cdot \left|\vec{q}\right|} = \frac{\sum_{i=1}^{t} (w_{ij} \cdot w_{iq})}{\sqrt{\sum_{i=1}^{t} w_{ij}^{2} \cdot \sum_{i=1}^{t} w_{iq}^{2}}}$$



$D_1 = 2T_1 + 3T_2 + 5T_3$ CosSim(D_1 , Q) = 10 / $\sqrt{(4+9+25)(0+0+4)}$ = 0.81
$D_2 = 3T_1 + 7T_2 + 1T_3$ CosSim(D_2 , Q) = 2 / $\sqrt{(9+49+1)(0+0+4)}$ = 0.13
$Q = 0T_1 + 0T_2 + 2T_3$

D_1 is 6 times better than D_2 using cosine similarity but only 5 times better using inner product.

K Nearest Neighbor for Text

Training:

For each training example $<x, c(x)> \in D$

Compute the corresponding TF-IDF vector, $\mathbf{d}_x$, for document x

Test instance *y*:

Compute TF-IDF vector $\mathbf{d}$ for document y

For each $<x, c(x)> \in D$

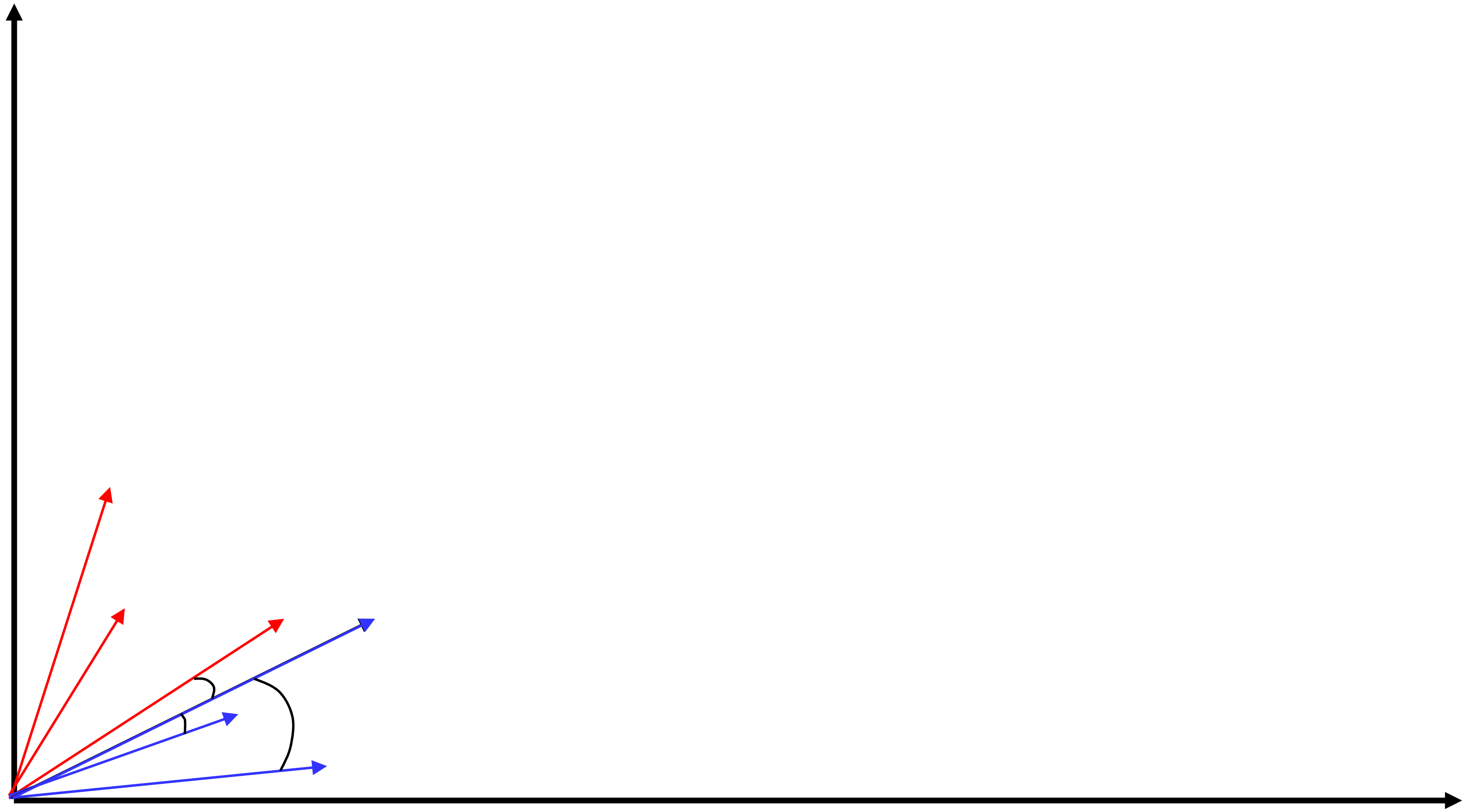
Let $s_x = \text{cosSim}(\mathbf{d}, \mathbf{d}_x)$

Sort examples, x, in D by decreasing value of s_x

Let N be the first k examples in D (*get most similar neighbors*)

Return the majority class of examples in N

Illustration of 3 Nearest Neighbor for Text



Nearest Neighbor Time Complexity

- Training Time: O($|D| \; L_d$) to compose TF-IDF vectors
- Testing Time: O($L_t + |D||V_t|$) to compare to all training vectors
  - Assume lengths of $\mathbf{d}_x$ vectors are computed and stored during training, allowing cosSim($\mathbf{d}$, $\mathbf{d}_x$) to be computed in time proportional to the number of non-zero entries in $\mathbf{d}$ (i.e. $|V_t|$)
- Testing time can be high for large training sets

Nearest Neighbor with Inverted Index

- Determining *k* nearest neighbors is same as determining the *k* best retrievals using the test document as a query to a database of training documents

- An index that points from words to documents that contain them allows more rapid retrieval of similar documents

- After stop-words removal
  - remaining words are rare
  - an inverted index narrows attention to a relatively small number of documents that share meaningful vocabulary with the test document

Nearest Neighbor with Inverted Index

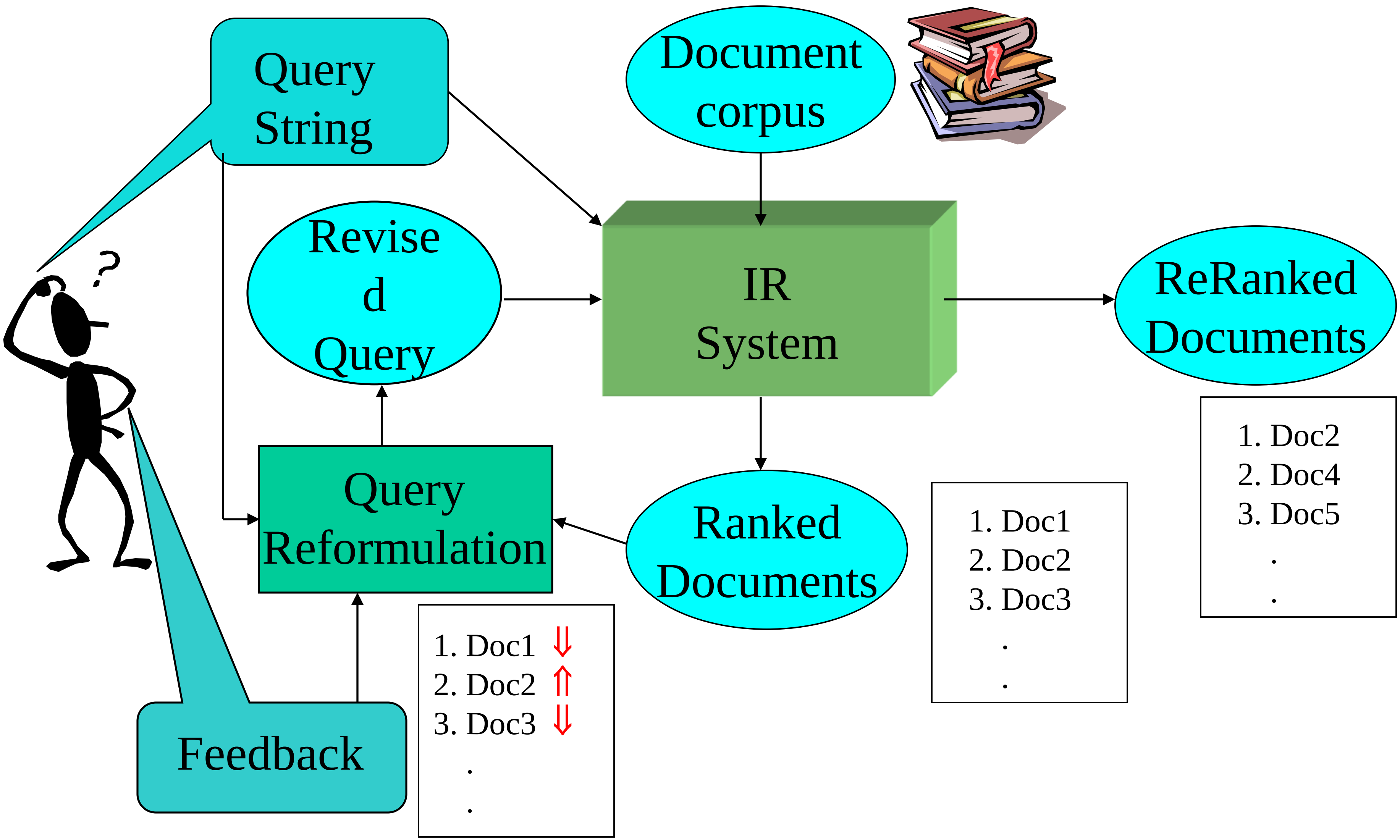
- Testing Time: $O(B|V_t|)$, where B is the average number of training documents in which a test-document word appears

- Overall classification: $O(L_t + B|V_t|)$
  - Typically $B << |D|$

Relevance Feedback in IR

- After initial retrieval results are presented, allow the user to provide feedback on the relevance of one or more of the retrieved documents
- Use this feedback information to reformulate the query
- Produce new results based on reformulated query
- Allows more interactive, multi-pass process

Relevance Feedback Architecture



Using Relevance Feedback (Rocchio)

- Relevance feedback methods can be adapted for text categorization
  - relevance feedback can be viewed as 2-class classification
    - Relevant vs. nonrelevant documents
- Use standard TF/IDF weighted vectors to represent text documents
- For training documents in each category, compute a *prototype* vector by summing the vectors of the training documents in the category
  - Prototype = centroid of members of class
- Assign test documents to the category with the closest prototype vector based on cosine similarity

Definition of centroid

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

- Where D_c is the set of all documents that belong to class c and $v(d)$ is the vector space representation of d
- *Note that centroid will in general* not *be a unit vector even when the inputs are unit vectors*

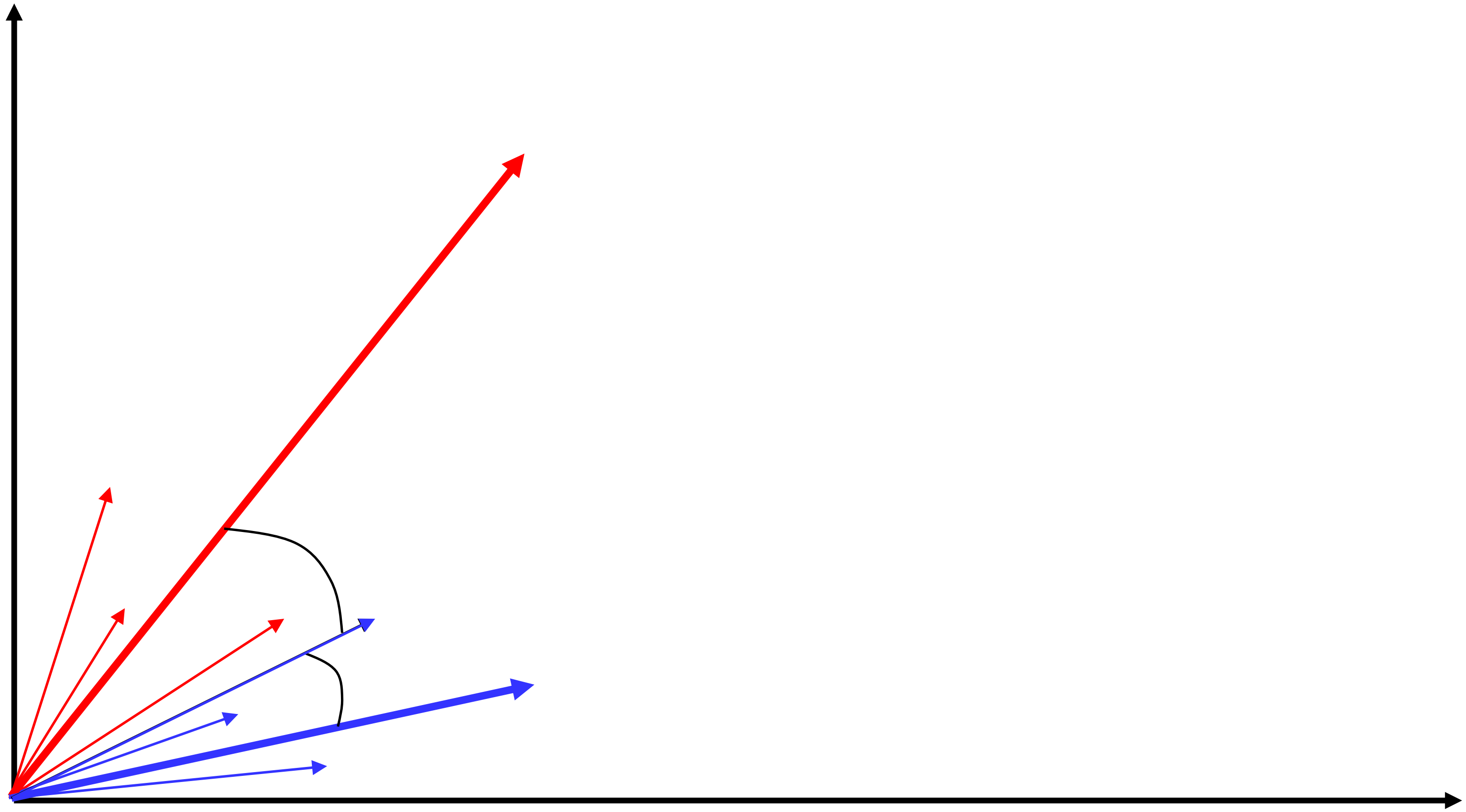
Rocchio Text Categorization Algorithm (Training)

Assume the set of categories is $\{c_1, c_2, \ldots c_n\}$
For i from 1 to n let $\mathbf{p}_i = \langle 0, 0, \ldots, 0 \rangle$ (*init. prototype vectors*)
For each training example $\langle x, c(x) \rangle \in D$
Let $\mathbf{d}$ be the frequency normalized TF/IDF term vector for doc x
Let $i = j: (c_j = c(x))$
(*sum all the document vectors in* c_i *to get* $\boldsymbol{p}_i$)
Let $\mathbf{p}_i = \mathbf{p}_i + \mathbf{d}$

Rocchio Text Categorization Algorithm (Test)

Given test document x
Let $\mathbf{d}$ be the TF/IDF weighted term vector for x
Let $m = -2$ (*init. maximum cosSim*)
For i from 1 to n:
    (*compute similarity to prototype vector*)
    Let $s = \text{cosSim}(\mathbf{d}, \mathbf{p}_i)$
    if $s > m$
        let $m = s$
        let $r = c_i$ (*update most similar class prototype*)
Return class r

Illustration of Rocchio Text Categorization



Rocchio Time Complexity

- Note: The time to add two sparse vectors is proportional to minimum number of non-zero entries in the two vectors
- Training Time: $O(|D|(L_d + |V_d|)) = O(|D|\ L_d)$ where L_d is the average length of a document in D and V_d is the average vocabulary size for a document in D
- Test Time: $O(L_t + |C||V_t|)$ where L_t is the average length of a test document and $|V_t|$ is the average vocabulary size for a test document
  - Assumes lengths of $\mathbf{p}_i$ vectors are computed and stored during training, allowing cosSim($\mathbf{d}$, $\mathbf{p}_i$) to be computed in time proportional to the number of non-zero entries in $\mathbf{d}$ (i.e. $|V_t|$)

Conclusions

- Many important applications of classification to text
- Requires an approach that works well with large, sparse features vectors, since typically each word is a feature and most words are rare
  - Naïve Bayes
  - kNN with cosine similarity
  - SVMs