Introduction to Machine Learning and Data Mining Lecture-14: Final Review

Prof. Eugene Chang

Today

- Homework #4 & #5 solutions
- Class project presentation
- Final review
- Final exam
  - Fri 08/21 6-8:30
  - 2 rooms with seating assignment
  - Calculators and pens only
  - No cellphone, no computer, no notes

Class Project Submission

- Send me & grader by email
  - Your presentation, including your names & ID’s
  - Link to your github page and/or blog site that has all your code and necessary data
- I will give grades for the projects before the final exam
  - You need to make sure your code and data are always valid and working between now and the final exam
  - You may improve your code further after today, but I don’t guarantee to look at it

1-Page Quick View

- Supervised learning
  - Linear regression, Decision tree, Naïve Bayes, Nearest neighbors
- Unsupervised learning
  - K-means, Hierarchy clustering (agglomerative, dendrogram)
- Evaluation
  - Cross validation: k-fold
  - Metrics: accuracy, precision, recall, F1-measure
- Advanced methods
  - Support Vector Machine
  - Bagging and Boosting
  - Random Forest
- Text processing
  - Boolean query, Naïve Bayes text classifier
  - Cosine similarity, TF-IDF weight
  - Rocchio classifier
  - Inverted index

2D Linear Regression

- For the 2-*d* problem (line) there are coefficients for the bias and the independent variable (*y*-intercept and slope)

$$Y = w_0 + w_1 X$$

- To find the values for the coefficients which minimize the objective function we take the partial derivates of the objective function (SSE) with respect to the coefficients. Set these to 0, and solve.

$$w_1 = \frac{n\sum xy - \sum x \sum y}{n\sum x^2 - \left(\sum x\right)^2} \qquad w_0 = \frac{\sum y - w_1 \sum x}{n}$$

Regression Error

- How can we evaluate the performance of a regression solution?
- Error Functions (or Loss functions)
  - Squared Error

$$E(t_i, y(\vec{x}_i, \vec{w})) = \frac{1}{2}(t_i - y(\vec{x}_i, \vec{w}))^2$$

  - Linear Error

$$E(t_i, y(\vec{x}_i, \vec{w})) = |t_i - y(\vec{x}_i, \vec{w})|$$

Least Square Solution Examples

Y	3	5	7
X	1	2	3

$$Y = 1 + 2X$$

X	1	2	3
Predict-Y	3.5	5	6.5
Target-Y	3	6	6
Error-Y	0.5	0.5	0.5

Y	3	6	6
X	1	2	3

$$Y = 2 + 1.5X$$

$$w_1 = \frac{n\sum xy - \sum x \sum y}{n\sum x^2 - \left(\sum x\right)^2}$$

$$w_0 = \frac{\sum y - w_1 \sum x}{n}$$

Definition of a Decision Tree

- A **Tree** data structure
- Each **internal node** corresponds to a feature
- **Leaves** are associated with target values.
- Nodes with **nominal features** have *N* children, where *N* is the number of nominal values
- Nodes with **continuous features** have two children for values less than and greater than or equal to a **break point**.

Decision Tree Induction Algorithm

- Basic algorithm (a greedy algorithm)
  - Tree is constructed in top-down recursive divide-and-conquer
  - At start, all the training examples are at the root
  - Attributes are categorical (if continuous-valued, they are discretized in advance)
  - Examples are partitioned recursively based on selected attributes
  - Test attributes are selected on the basis of a heuristic or statistical measure (e.g., information gain)
- Conditions for stopping partitioning
  - All samples for a given node belong to the same class
  - There are no remaining attributes for further partitioning – majority voting is employed for classifying the leaf
  - There are no samples left

Attribute Selection Measure: Information Gain

- Let p_i be the probability that an arbitrary tuple in D belongs to class C_i, estimated by $|C_{i,D}|/|D|$
- Expected information (entropy) needed to classify a tuple in D:

$$Info(D) = -\sum_{i=1}^{m} p_i \log_2(p_i)$$

- Information needed (after using A to split D into v partitions) to classify D:

$$Info_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} \times Info(D_j)$$

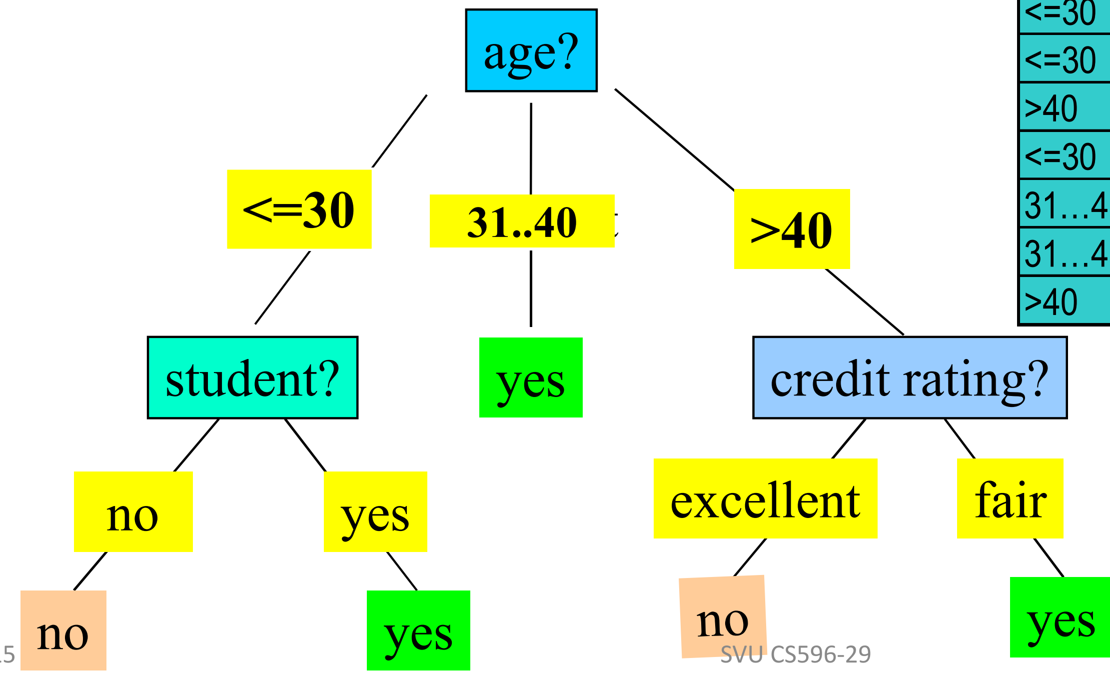
- Information gained by branching on attribute A

$$Gain(A) = Info(D) - Info_A(D)$$

- Select the attribute with the highest information gain

Decision Tree Results

- Training data set: Buys_computer
- The data set follows an example of Quinlan's ID3 (Playing Tennis)
- Resulting tree:



age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

Attribute Selection: Information Gain

- Class P: buys_computer = "yes"
- Class N: buys_computer = "no"

$$Info(D) = I(9,5) = -\frac{9}{14}\log_2\left(\frac{9}{14}\right) - \frac{5}{14}\log_2\left(\frac{5}{14}\right) = 0.940$$

$$Info_{age}(D) = \frac{5}{14}I(2,3) + \frac{4}{14}I(4,0) + \frac{5}{14}I(3,2) = 0.694$$

age	p_i	n_i	$I(p_i, n_i)$
<=30	2	3	0.971
31…40	4	0	0
>40	3	2	0.971

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

$\frac{5}{14}I(2,3)$ means "age <=30" has 5 out of 14 samples, with 2 yes'es and 3 no's. Hence

$$Gain(age) = Info(D) - Info_{age}(D) = 0.246$$

Similarly,

$$Gain(income) = 0.029$$
$$Gain(student) = 0.151$$
$$Gain(credit\_rating) = 0.048$$

Bayes Probability

- Example: drawing a fruit from 2 color boxes
- Here the Box is the class (source), and the fruit is a feature, or observation.

	Orange	Apple	
Blue box	1	3	4
Red box	6	2	8
	7	5	12

Interpretation of Bayes Rule

Likelihood

Prior

Posterior

$$p(Y|X) = \frac{p(X|Y)p(Y)}{p(X)}$$

- **Prior**: Information we have before observation.
- **Posterior**: The distribution of Y after observing X
- **Likelihood:** The likelihood of observing X given Y
- $P(X) = \sum_i P(X|Y_i)P(Y_i)$
- Example
  - P(Y_0) red box Prior probability = 1/3, P(Y_1) blue box Prior probability = 2/3
  - $P(X_0|Y_0)$ orange given red box = ¼, $P(X_1|Y_0)$ apple given red box = 3/4
  - $P(X_0|Y_1)$ orange given blue box = 3/4, $P(X_1|Y_1)$ apple given blue box = 1/4

Naïve Bayes Classifier: Training Dataset

Class:

C1:buys_computer = 'yes'

C2:buys_computer = 'no'

Data to be classified:

X = (age <=30,

Income = medium,

Student = yes

Credit_rating = Fair)

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

Naïve Bayes Classifier Example

- $P(C_i)$: P(buys_computer = "yes") = 9/14 = 0.643

  P(buys_computer = "no") = 5/14= 0.357
- Compute $P(X|C_i)$ for each class

  P(age = "<=30" | buys_computer = "yes") = 2/9 = 0.222

  P(age = "<= 30" | buys_computer = "no") = 3/5 = 0.6

  P(income = "medium" | buys_computer = "yes") = 4/9 = 0.444

  P(income = "medium" | buys_computer = "no") = 2/5 = 0.4

  P(student = "yes" | buys_computer = "yes) = 6/9 = 0.667

  P(student = "yes" | buys_computer = "no") = 1/5 = 0.2

  P(credit_rating = "fair" | buys_computer = "yes") = 6/9 = 0.667

  P(credit_rating = "fair" | buys_computer = "no") = 2/5 = 0.4
- **X = (age <= 30 , income = medium, student = yes, credit_rating = fair)**

age	income	student	credit_rating	com
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

$P(X|C_i)$: P(X|buys_computer = "yes") = 0.222 x 0.444 x 0.667 x 0.667 = 0.044

P(X|buys_computer = "no") = 0.6 x 0.4 x 0.2 x 0.4 = 0.019

$P(X|C_i)*P(C_i)$: P(X|buys_computer = "yes") * P(buys_computer = "yes") = 0.028

P(X|buys_computer = "no") * P(buys_computer = "no") = 0.007

Therefore, X belongs to class ("buys_computer = yes")

Smoothing

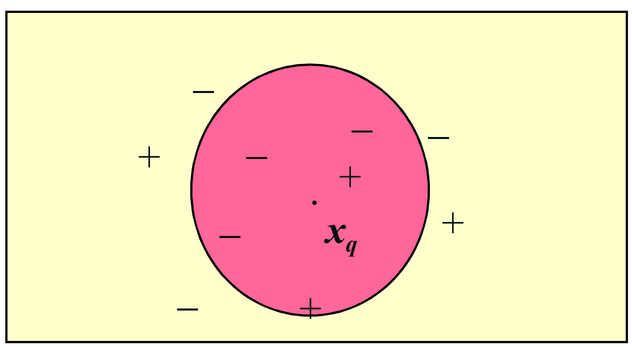
- To account for estimation from small samples, probability estimates are adjusted or *smoothed*.
- Laplace smoothing using an *m*-estimate assumes that each feature is given a prior probability, *p*, that is assumed to have been previously observed in a "virtual" sample of size *m*.

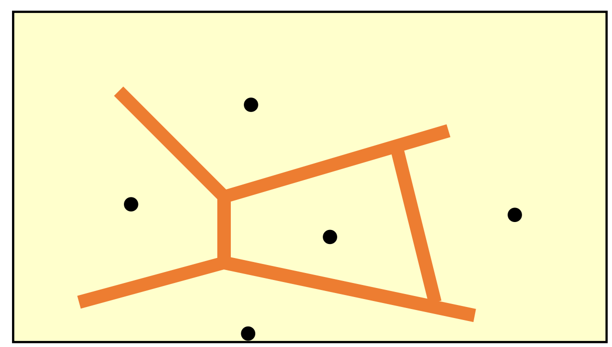
$$P(X_i = x_{ij} \mid Y = y_k) = \frac{n_{ijk} + mp}{n_k + m}$$

- For binary features, *p* is simply assumed to be 0.5.

k-Nearest Neighbor (kNN) Classification

- All instances correspond to points in the n-D space
- The nearest neighbor are defined in terms of Euclidean distance, dist($\mathbf{X_1}$, $\mathbf{X_2}$)
- Target function could be discrete- or real- valued
- For discrete-valued, *k*-NN returns the most common value among the *k* training examples nearest to x_q
- Vonoroi diagram: the decision surface induced by 1-NN for a typical set of training examples





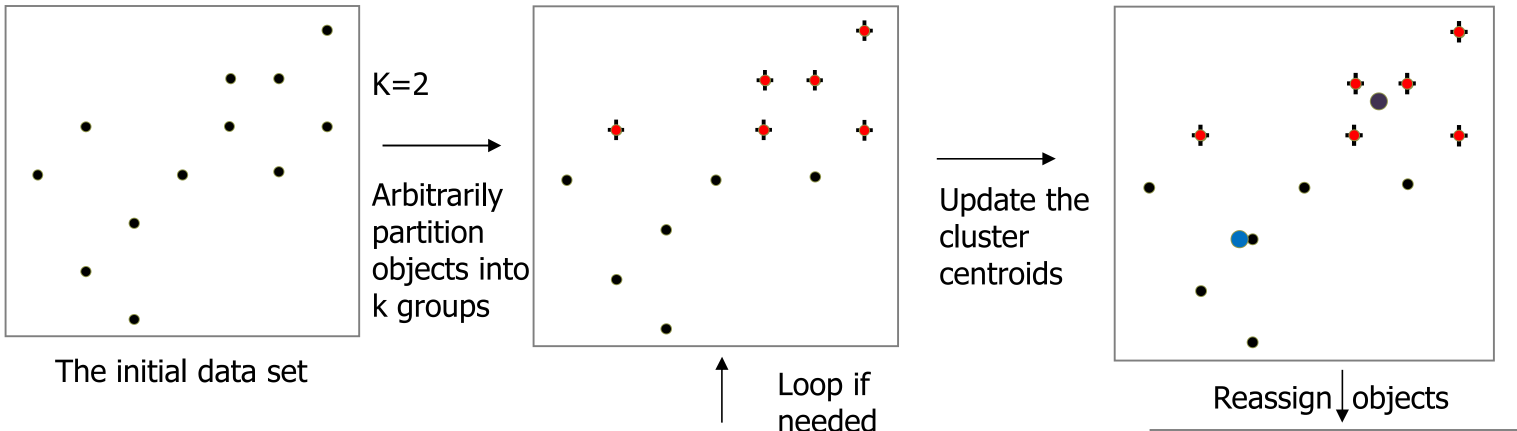
K-Means Algorithm

- Given an integer K specifying the number of clusters
- Initialize K cluster centroids
  - Select K points from the data set at random
  - Select K points from the space at random
- For each point in the data set, assign it to the cluster center it is closest to $\underset{C_i}{\operatorname{argmin}}\, d(\vec{x}, C_i)$
- Update each centroid based on the points that are assigned to it

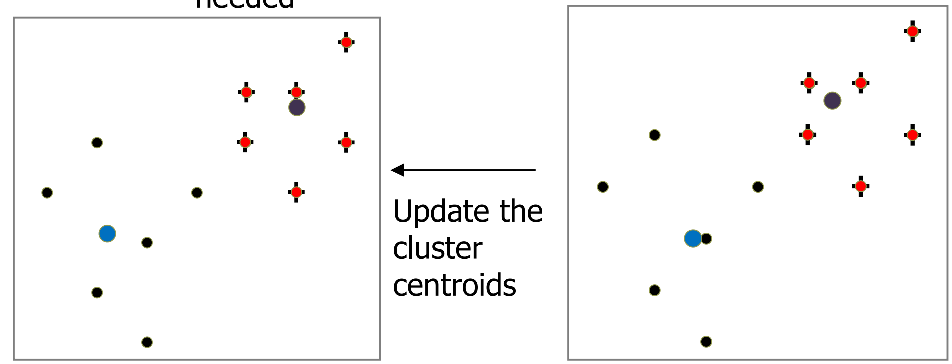
$$C_i = \frac{1}{|C_i|} \sum_{\vec{x} \in C_i} \vec{x}$$

- If any data point has changed clusters, repeat

An Example of *K-Means* Clustering

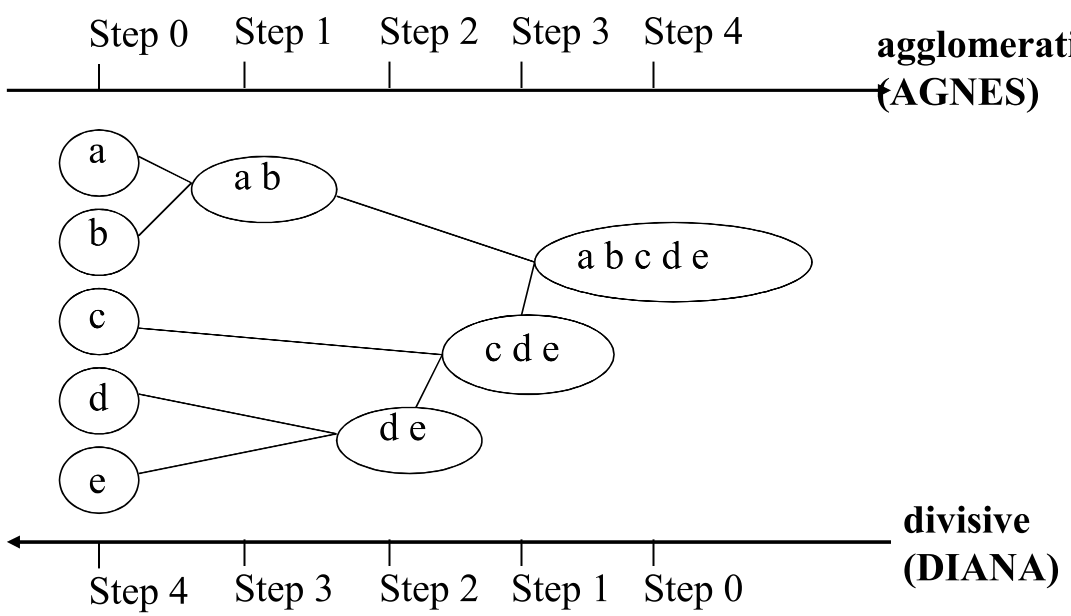


- Partition objects into *k* nonempty subsets
- Repeat
  - Compute centroid (i.e., mean point) for each partition
  - Assign each object to the cluster of its nearest centroid
- Until no change



Hierarchical Clustering

- Use distance matrix as clustering criteria. This method does not require the number of clusters ***k*** as an input, but needs a termination condition



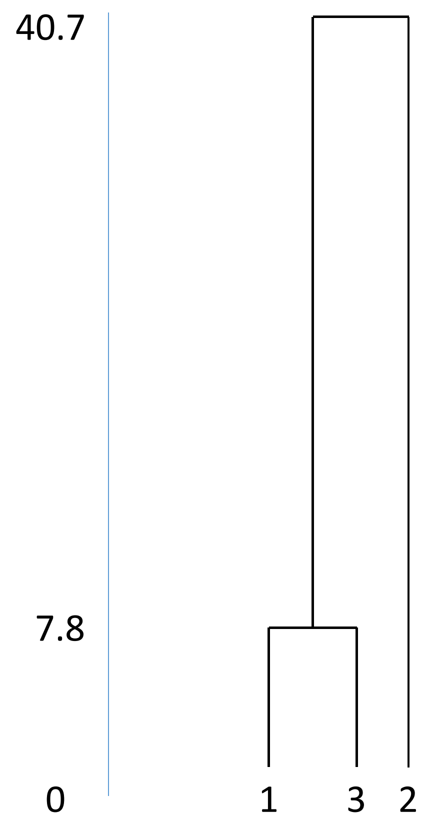
Dendogram

- Dendogram: a graphic plot to visualize hierarchical sequence of clustering assignments
- Tree with the following properties
  - Each node represent a grouping
  - Root node is the whole dataset
  - Each leaf node (at bottom) is a singleton (data point)
  - Each internal node has two links connecting to the child nodes
  - Choice of links are determined by the dissimilarity measure
  - If we put the leaf modes at level zero, then each internal node is drawn at the height proportional to the dissimilarity

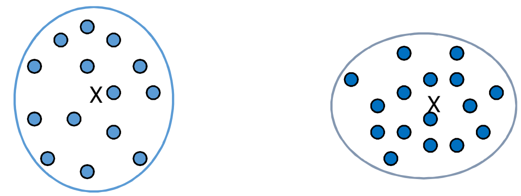
Dendogram Example

ID	1	2	3
Height	66	73	72
Weight	170	210	165

dissimilarity	1	2	3
1	0	40.7	7.8
2		0	45
3			0



Distance between Clusters



- Single link: smallest distance between an element in one cluster and an element in the other, i.e., $dist(K_i, K_j) = min(t_{ip}, t_{jq})$
- Complete link: largest distance between an element in one cluster and an element in the other, i.e., $dist(K_i, K_j) = max(t_{ip}, t_{jq})$
- Average: avg distance between an element in one cluster and an element in the other, i.e., $dist(K_i, K_j) = avg(t_{ip}, t_{jq})$
- Centroid: distance between the centroids of two clusters, i.e., $dist(K_i, K_j) = dist(C_i, C_j)$
- Medoid: distance between the medoids of two clusters, i.e., $dist(K_i, K_j) = dist(M_i, M_j)$
  - Medoid: a chosen, centrally located object in the cluster

Precision, Recall, and F-measures

- **Precision**: exactness – what % of tuples that the classifier labeled as positive are actually positive

$$precision = \frac{TP}{TP + FP}$$

- **Recall:** completeness – what % of positive tuples did the classifier label as positive?

$$recall = \frac{TP}{TP + FN}$$

- Perfect score is 1.0
- Inverse relationship between precision & recall
- ***F* measure (*F_1* or *F*-score)**: harmonic mean of precision and recall,

$$F = \frac{2 \times precision \times recall}{precision + recall}$$

- ***F_β*:** weighted measure of precision and recall
    - assigns ß times as much weight to recall as to precision

$$F_\beta = \frac{(1 + \beta^2) \times precision \times recall}{\beta^2 \times precision + recall}$$

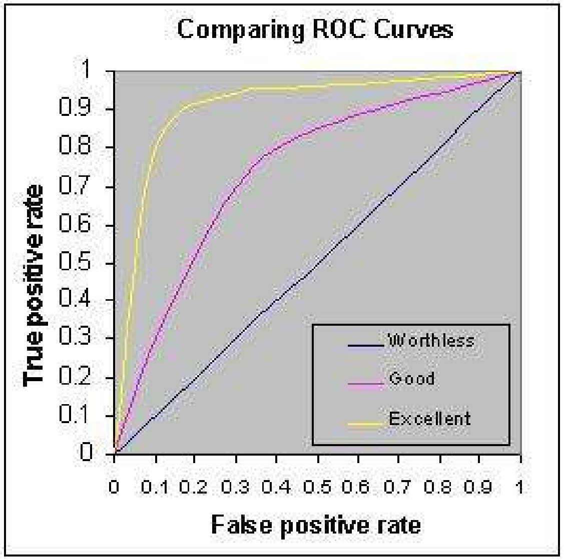
Classifier Evaluation Metrics: Example

Actual class/Predicted class	buy_computer = yes	buy_computer = no	Total
buy_computer = yes	**6954**	**46**	7000
buy_computer = no	**412**	**2588**	3000
Total	7366	2634	10000

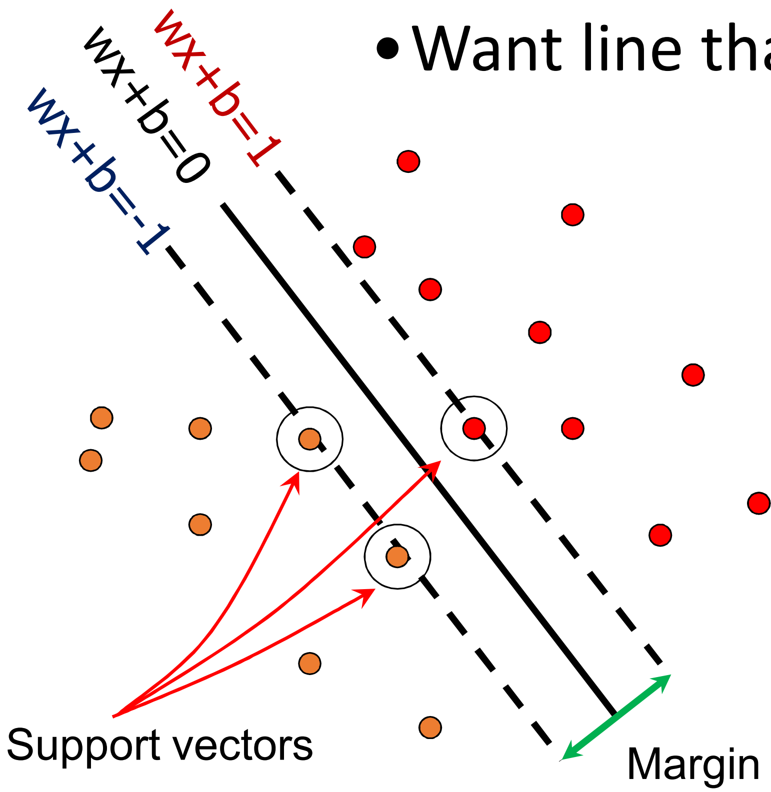
- *Accuracy = (***6954+2588)** / 10000 = 95.42%
- *Precision* = **6954**/7366 = 94.4%
- *Recall* = *Sensitivity* = **6954**/7000 = 99.34%
- Specificity = **2588** /3000 = 86.27%
- F1 = P*R / 2(P+R) = 48.4%

ROC Curves

- It is common to plot classifier performance at a variety of settings or thresholds
- Receiver Operating Characteristic (ROC) curves plot true positives against false positives.
- The overall performance is calculated by the Area Under the Curve (AUC)



Support Vector Machines



- Want line that maximizes the margin.

$\mathbf{x}_i$ positive $(y_i = 1)$: $\quad \mathbf{x}_i \cdot \mathbf{w} + b \geq 1$

$\mathbf{x}_i$ negative $(y_i = -1)$: $\quad \mathbf{x}_i \cdot \mathbf{w} + b \leq -1$

For support, vectors, $\quad \mathbf{x}_i \cdot \mathbf{w} + b = \pm 1$

Distance between point and line: $\quad \dfrac{|\mathbf{x}_i \cdot \mathbf{w} + b|}{\|\mathbf{w}\|}$

Therefore, the margin is $2 / \|\mathbf{w}\|$

Finding the maximum margin line

- Solution: $\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$

$$b = y_i - \mathbf{w} \cdot \mathbf{x}_i \quad \text{(for any support vector)}$$

$$\mathbf{w} \cdot \mathbf{x} + b = \sum_i \alpha_i y_i \mathbf{x}_i \cdot \mathbf{x} + b$$

- Classification function:

$$f(x) = \operatorname{sign}(\mathbf{w} \cdot \mathbf{x} + \mathrm{b})$$
$$= \operatorname{sign}\left(\sum_{\mathrm{i}} \alpha_{\mathrm{i}} \boxed{\mathbf{x}_i \cdot \mathbf{x}} + b\right)$$

If f(x) < 0, classify as negative, if f(x) > 0, classify as positive

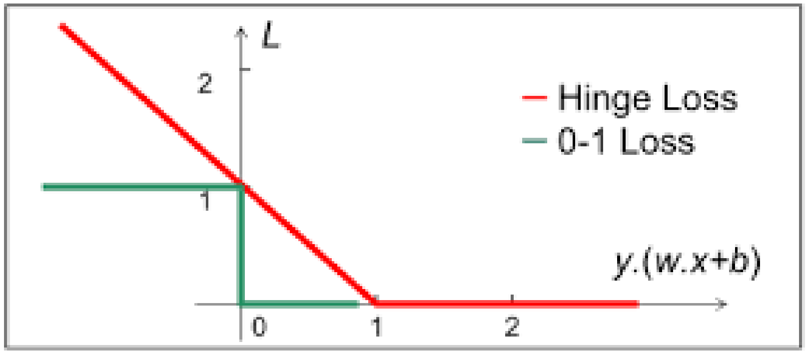
- Notice that it relies on an *inner product* between the test point $\boldsymbol{x}$ and the support vectors $\boldsymbol{x}_i$
- (Solving the optimization problem also involves computing the inner products $\boldsymbol{x}_i \cdot \boldsymbol{x}_j$ between all pairs of training points)

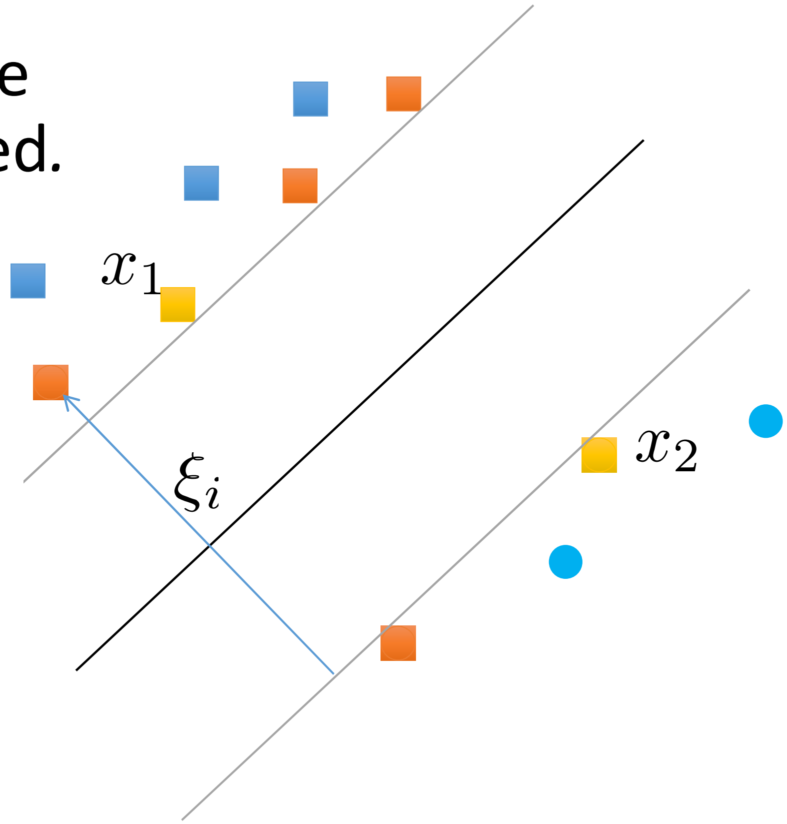
C. Burges, A Tutorial on Support Vector Machines for Pattern Recognition, Data Mining and Knowledge Discovery, 1998

Soft Margin Example

- Points are allowed within the margin, but cost is introduced.

Hinge Loss





Soft Margin Classification

- Solution: Introduce a penalty term to the constraint function

$$\min \|\vec{w}\| + C \sum_{i=0}^{N-1} \xi_i$$

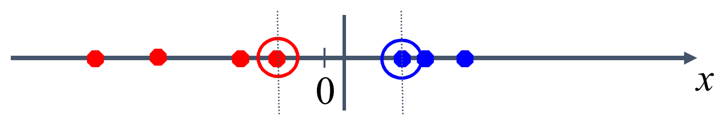
$$\text{where } t_i(\vec{w}^T x_i + b) \geq 1 - \xi_i \text{ and } \xi_i \geq 0$$

$$L(\vec{w}, b) = \frac{1}{2}\vec{w} \cdot \vec{w} + C \sum_{i=0}^{N-1} \xi_i - \sum_{i=0}^{N-1} \alpha_i [t_i((\vec{w} \cdot \vec{x_i}) + b) + \xi_i - 1]$$

- C is a regularization parameter:
  - small C allows constraints to be easily ignored → large margin
  - large C makes constraints hard to ignore → narrow margin
  - C = ∞ enforces all constraints: hard margin

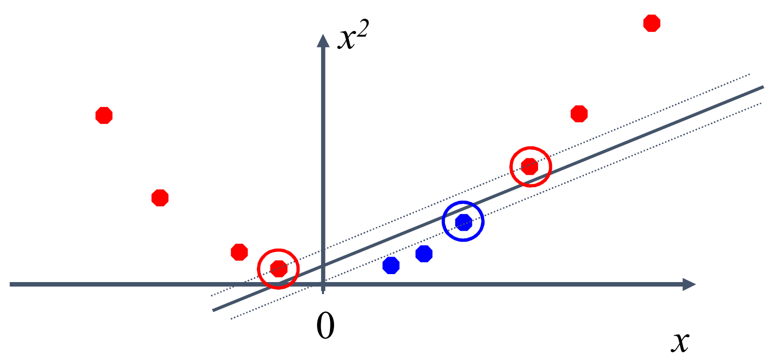
Non-linear SVMs

- Datasets that are linearly separable with some noise work out great:



- But what are we going to do if the dataset is just too hard?

- How about… mapping data to a higher-dimensional space:



Slide from Andrew Moore's tutorial: http://www.autonlab.org/tutorials/svm.html

Examples of General Purpose Kernel Functions

- Linear: $K(\mathbf{x_i},\mathbf{x_j}) = \mathbf{x_i}^T\mathbf{x_j}$

- Polynomial of power p: $K(\mathbf{x_i},\mathbf{x_j}) = (1+ \mathbf{x_i}^T\mathbf{x_j})^p$

- Gaussian (radial-basis function network):

$$K(\mathbf{x_i}, \mathbf{x_j}) = \exp\left(-\frac{\left\|\mathbf{x_i} - \mathbf{x_j}\right\|^2}{2\sigma^2}\right)$$

Slide from Andrew Moore's tutorial: http://www.autonlab.org/tutorials/svm.html

Bagging

- Decision Stump
- Single level decision binary tree
- Entropy – x<=0.35 or x<=0.75
- Accuracy at most 70%

X	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
y	1	1	1	-1	-1	-1	-1	1	1	1

Bagging Round 1:

x	0.1	0.2	0.2	0.3	0.4	0.4	0.5	0.6	0.9	0.9
y	1	1	1	1	-1	-1	-1	-1	1	1

x <= 0.35 ==> y = 1
x > 0.35 ==> y = -1

Bagging Round 2:

x	0.1	0.2	0.3	0.4	0.5	0.8	0.9	1	1	1
y	1	1	1	-1	-1	1	1	1	1	1

x <= 0.65 ==> y = 1
x > 0.65 ==> y = 1

Bagging Round 3:

x	0.1	0.2	0.3	0.4	0.4	0.5	0.7	0.7	0.8	0.9
y	1	1	1	-1	-1	-1	-1	-1	1	1

x <= 0.35 ==> y = 1
x > 0.35 ==> y = -1

Bagging Round 4:

x	0.1	0.1	0.2	0.4	0.4	0.5	0.5	0.7	0.8	0.9
y	1	1	1	-1	-1	-1	-1	-1	1	1

x <= 0.3 ==> y = 1
x > 0.3 ==> y = -1

Bagging Round 5:

x	0.1	0.1	0.2	0.5	0.6	0.6	0.6	1	1	1
y	1	1	1	-1	-1	-1	-1	1	1	1

x <= 0.35 ==> y = 1
x > 0.35 ==> y = -1

Bagging Round 6:

x	0.2	0.4	0.5	0.6	0.7	0.7	0.7	0.8	0.9	1
y	1	-1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 ==> y = -1
x > 0.75 ==> y = 1

Bagging Round 7:

x	0.1	0.4	0.4	0.6	0.7	0.8	0.9	0.9	0.9	1
y	1	-1	-1	-1	-1	1	1	1	1	1

x <= 0.75 ==> y = -1
x > 0.75 ==> y = 1

Bagging Round 8:

x	0.1	0.2	0.5	0.5	0.5	0.7	0.7	0.8	0.9	1
y	1	1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 ==> y = -1
x > 0.75 ==> y = 1

Bagging Round 9:

x	0.1	0.3	0.4	0.4	0.6	0.7	0.7	0.8	1	1
y	1	1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 ==> y = -1
x > 0.75 ==> y = 1

Bagging Round 10:

x	0.1	0.1	0.1	0.1	0.3	0.3	0.8	0.8	0.9	0.9
y	1	1	1	1	1	1	1	1	1	1

x <= 0.05 ==> y = -1
x > 0.05 ==> y = 1

Figure 5.35. Example of bagging.

Bagging

Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	1	1	1	-1	-1	-1	-1	-1	-1	-1
2	1	1	1	1	1	1	1	1	1	1
3	1	1	1	-1	-1	-1	-1	-1	-1	-1
4	1	1	1	-1	-1	-1	-1	-1	-1	-1
5	1	1	1	-1	-1	-1	-1	-1	-1	-1
6	-1	-1	-1	-1	-1	-1	-1	1	1	1
7	-1	-1	-1	-1	-1	-1	-1	1	1	1
8	-1	-1	-1	-1	-1	-1	-1	1	1	1
9	-1	-1	-1	-1	-1	-1	-1	1	1	1
10	1	1	1	1	1	1	1	1	1	1
Sum	2	2	2	-6	-6	-6	-6	2	2	2
Sign	1	1	1	-1	-1	-1	-1	1	1	1
True Class	1	1	1	-1	-1	-1	-1	1	1	1

Figure 5.36. Example of combining classifiers constructed using the bagging approach.

Accuracy of ensemble classifier: 100% ☺

Boosting

- Equal weights are assigned to each training tuple (1/d for round 1)
- After a classifier M_i is learned, the weights are adjusted to allow the subsequent classifier M_{i+1} to "pay more attention" to tuples that were misclassified by M_i.
- Final boosted classifier M* combines the votes of each individual classifier
- Weight of each classifier's vote is a function of its accuracy
- Adaboost – popular boosting algorithm

Boosting

- Records that are wrongly classified will have their weights increased
- Records that are classified correctly will have their weights decreased

Original Data	1	2	3	4	5	6	7	8	9	10
Boosting (Round 1)	7	3	2	8	7	9	4	10	6	3
Boosting (Round 2)	5	4	9	4	2	5	1	7	4	2
Boosting (Round 3)	4	4	8	10	4	5	4	6	3	4

- Example 4 is hard to classify
- Its weight is increased, therefore it is more likely to be chosen again in subsequent rounds

Adaboost

- Given a set of *d* class-labeled tuples, ($\mathbf{X_1}$, y_1), …, ($\mathbf{X_d}$, y_d)
- Initially, all the weights of tuples are set the same (1/d)
- Generate k classifiers in k rounds. At round i,
  - Tuples from D are sampled (with replacement) to form a training set D_i of the same size
  - Each tuple's chance of being selected is based on its weight
  - A classification model M_i is derived from D_i
  - Its error rate is calculated using D_i as a test set
  - If a tuple is misclassified, its weight is increased, o.w. it is decreased
- Error rate: err($\mathbf{X_j}$) is the misclassification error of tuple $\mathbf{X_j}$. Classifier M_i error rate is the sum of the weights of the misclassified tuples:

$$error(M_i) = \sum_{j}^{d} w_j \times err(\mathbf{X_j})$$

- The weight of classifier M_i's vote is $\log \frac{1 - error(M_i)}{error(M_i)}$

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.

$$
\begin{array}{c|cccc}
 & T_1 & T_2 & \ldots & T_t \\
\hline
D_1 & w_{11} & w_{21} & \ldots & w_{t1} \\
D_2 & w_{12} & w_{22} & \ldots & w_{t2} \\
\vdots & \vdots & \vdots & & \vdots \\
\vdots & \vdots & \vdots & & \vdots \\
D_n & w_{1n} & w_{2n} & \ldots & w_{tn}
\end{array}
$$

Boolean model (contd)

- Query terms are combined logically using the Boolean operators **AND**, **OR**, and **NOT.**
  - E.g., ((*data* AND *mining*) AND (NOT *text*))
- Retrieval
  - Given a Boolean query, the system retrieves every document that makes the query logically true.
  - Called **exact match**.
- The retrieval results are usually quite poor because term frequency is not considered.

Naïve Bayes for Text

- Modeled as generating a bag of words for a document in a given category by repeatedly sampling with replacement from a vocabulary $V = \{w_1, w_2, \ldots w_m\}$ based on the probabilities $P(w_j \mid c_i)$.
- Smooth probability estimates with Laplace *m*-estimates assuming a uniform distribution over all words ($p = 1/|V|$) and $m = |V|$
  - Equivalent to a virtual sample of seeing each word in each category exactly once.

Text Naïve Bayes Algorithm (Train)

Let V be the vocabulary of all words in the documents in D
For each category $c_i \in C$
  Let D_i be the subset of documents in D in category c_i
  $P(c_i) = |D_i| / |D|$
  Let T_i be the concatenation of all the documents in D_i
  Let n_i be the total number of word occurrences in T_i
  For each word $w_j \in V$
    Let n_{ij} be the number of occurrences of w_j in T_i
    Let $P(w_j \mid c_i) = (n_{ij} + 1) / (n_i + |V|)$ ⇐ *Laplacian smoothing*

Example Naïve Bayes

	docID		c = China?
Training set	1	Chinese Beijing Chinese	Yes
	2	Chinese Chinese Shangai	Yes
	3	Chinese Macao	Yes
	4	Tokyo Japan Chinese	No
Test set	5	Chinese Chinese Chinese Tokyo Japan	?

Two classes: “China”, “not China”

V = {Beijing, Chinese, Shangai, Japan, Macao, Tokyo}

N = 4 $\hat{P}(c) = 3/4$ $\hat{P}(\overline{c}) = 1/4$

Example Naïve Bayes

	docID		c = China?
Training set	1	Chinese Beijing Chinese	Yes
	2	Chinese Chinese Shangai	Yes
	3	Chinese Macao	Yes
	4	Tokyo Japan Chinese	No
Test set	5	Chinese Chinese Chinese Tokyo Japan	?

Estimation

$$\hat{P}(\text{Chinese} \mid c) = (5+1)/(8+6) = 3/7$$

$$\hat{P}(\text{Tokyo} \mid c) = \hat{P}(\text{Japan} \mid c) = (0+1)/(8+6) = 1/14$$

$$\hat{P}(\text{Chinese} \mid \overline{c}) = (1+1)/(3+6) = 2/9$$

$$\hat{P}(\text{Tokyo} \mid c) = \hat{P}(\text{Japan} \mid c) = (1+1)/(3+6) = 2/9$$

Classification

$$P(c \mid d) \propto P(c) \prod_{1 \le k \le n_d} P(t_k \mid c)$$

$$P(c \mid d_5) \propto 3/4 \cdot (3/7)^3 \cdot 1/14 \cdot 1/14 \approx 0.0003$$

$$P(\overline{c} \mid d_5) \propto 1/4 \cdot (2/9)^3 \cdot 2/9 \cdot 2/9 \approx 0.0001$$

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:

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

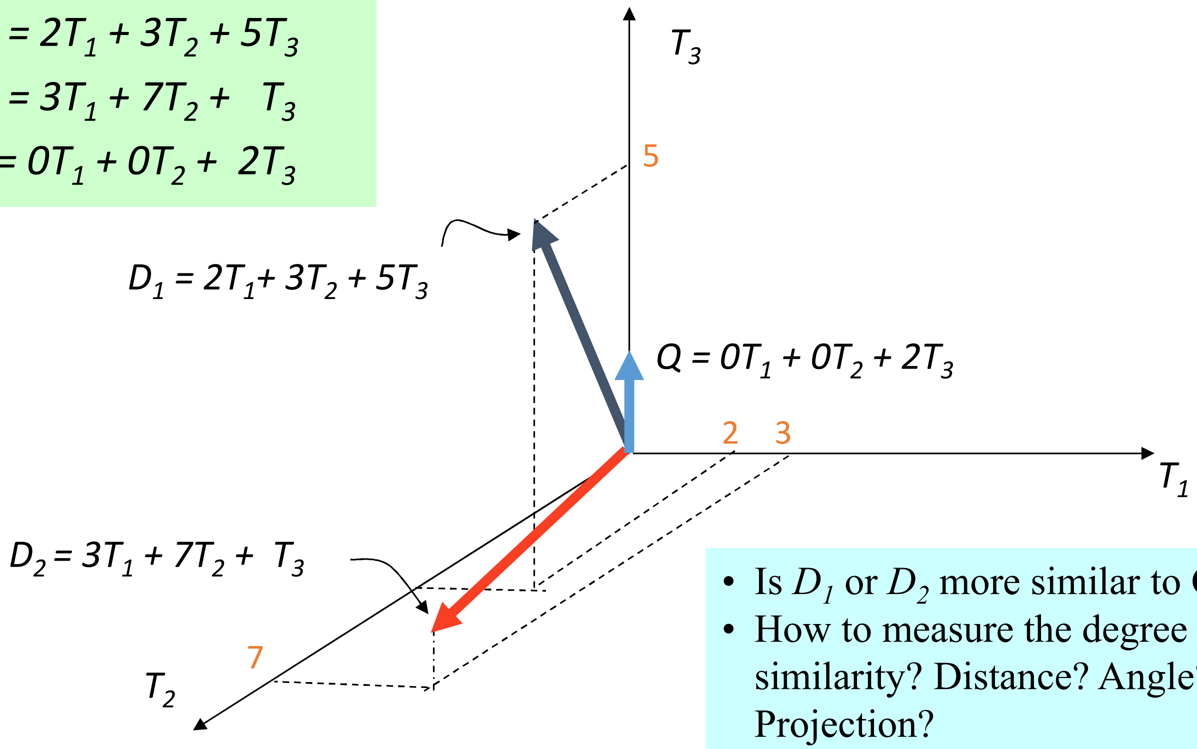
Graphic Representation

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?

Retrieval in Vector Space Model

- Query **q** is represented in the same way or slightly differently.
- Relevance of **d**$_i$ to **q**: Compare the similarity of query **q** and document **d**$_i$.
- Cosine similarity (the cosine of the angle between the two vectors)

$$cosine(\mathbf{d}_j, \mathbf{q}) = \frac{\langle \mathbf{d}_j \bullet \mathbf{q} \rangle}{\| \mathbf{d}_j \| \times \| \mathbf{q} \|} = \frac{\sum_{i=1}^{|V|} w_{ij} \times w_{iq}}{\sqrt{\sum_{i=1}^{|V|} w_{ij}^2} \times \sqrt{\sum_{i=1}^{|V|} w_{iq}^2}}$$

- Cosine is also commonly used in text clustering

Vector Space with Weights

- Documents are also treated as a “bag” of words or terms.
- Each document is represented as a vector.
- However, the term weights are no longer 0 or 1. Each term weight is computed based on some variations of TF or TF-IDF scheme.

- Term Frequency (TF) Scheme: The weight of a term t_i in document $\mathbf{d}_j$ is the number of times that t_i appears in $\mathbf{d}_j$, denoted by f_{ij}. Normalization may also be applied.

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 other ways of determining term weights have been proposed.
- Experimentally, *tf-idf* has been found to work well.

Example of TF-IDF

- Original counts=

```
[[1, 0, 0],
 [0, 1, 0],
 [0, 0, 1],
 [1, 1, 0],
 [1, 0, 1],
 [0, 1, 1],
 [1, 1, 1],
 [1, 0, 1],
 [0, 1, 1]]
```

- Tfidf =

```
[[ 1.          0.          0.        ]
 [ 0.          1.          0.        ]
 [ 0.          0.          1.        ]
 [ 0.70710678  0.70710678  0.        ]
 [ 0.74404499  0.          0.66812952]
 [ 0.          0.74404499  0.66812952]
 [ 0.59693793  0.59693793  0.53603191]
 [ 0.74404499  0.          0.66812952]
 [ 0.          0.74404499  0.66812952]]
```

Rocchio text classifier

- In fact, a variation of the Rocchio method above, called the **Rocchio classification** method, can be used to improve retrieval effectiveness
  - so are other machine learning methods. Why?
- Rocchio classifier is constructed by producing a prototype vector $\mathbf{c}_i$ for each class *i* (*relevant* or *irrelevant* in this case):

$$\mathbf{c}_i = \frac{\alpha}{|D_i|}\sum_{\mathbf{d}\in D_i}\frac{\mathbf{d}}{\|\mathbf{d}\|} - \frac{\beta}{|D - D_i|}\sum_{\mathbf{d}\in D - D_i}\frac{\mathbf{d}}{\|\mathbf{d}\|}$$

- In classification, cosine is used.
- Also knows as Nearest Centroid Classifier

Inverted index

- The inverted index of a document collection is basically a data structure that
  - attaches each distinctive term with a list of all documents that contains the term.
- Thus, in retrieval, it takes constant time to
  - find the documents that contains a query term.
  - multiple query terms are also easy handle as we will see soon.

An example

Example 3: We have three documents of id_1, id_2, and id_3:

id_1: Web mining is useful.
1 2 3 4

id_2: Usage mining applications.
1 2 3

id_3: Web structure mining studies the Web hyperlink structure.
1 2 3 4 5 6 7 8

(A)		(B)	
Applications:	id_2	Applications:	<id_2, 1, [3]>
Hyperlink:	id_3	Hyperlink:	<id_3, 1, [7]>
Mining:	id_1, id_2, id_3	Mining:	<id_1, 1, [2]>, <id_2, 1, [2]>, <id_3, 1, [3]>
Structure:	id_3	Structure:	<id_3, 2, [2, 8]>
Studies:	id_3	Studies:	<id_3, 1, [4]>
Usage:	id_2	Usage:	<id_2, 1, [1]>
Useful:	id_1	Useful:	<id_1, 1, [4]>
Web:	id_1, id_3	Web:	<id_1, 1, [1]>, <id_3, 2, [1, 6]>

Fig. 6.7. Two inverted indices: a simple version and a more complex version