Advanced Topics in Computer Vision and Robotics

Classification Methods

Some slides thanks to S. Lazebnik, T. Berg, Fei-Fei Li, K. Grauman and others

Simple Example of Recognition Statistical Viewpoint

- Suppose a we obtain measurement *z*
- What is *P(zebra|z)?*

Causal vs. Diagnostic Reasoning

- *P(zebra|z)* is diagnostic.
- *P(z|zebra)* is causal.
- Often causal knowledge is easier to obtain.
- Bayes rule allows us to use causal knowledge:

count frequencies!

$$P(zebra \mid z) = \frac{P(z \mid zebra)P(zebra)}{P(z)}$$

3

Combining Evidence

- Suppose we obtain another observation z_2.
- How can we integrate this new information?
- More generally, how can we estimate $P(x \mid z_1 \ldots z_n)$?

4

Example: Second Measurement

- $P(z_2|zebra) = 0.5$ $P(z_2|\neg zebra) = 0.6$
- $P(zebra|z_1)=2/3$

$$P(zebra \mid z_2, z_1) = \frac{P(z_2 \mid zebra)\, P(zebra \mid z_1)}{P(z_2 \mid zebra)\, P(zebra \mid z_1) + P(z_2 \mid \neg zebra)\, P(\neg zebra \mid z_1)}$$

$$= \frac{\frac{1}{2}\cdot\frac{2}{3}}{\frac{1}{2}\cdot\frac{2}{3}+\frac{3}{5}\cdot\frac{1}{3}} = \frac{5}{8} = 0.625$$

- z_2 lowers the probability that the picture is zebra.

5

Object categorization: the statistical viewpoint

- MAP decision: $p(zebra \mid image)$

vs.

$p(no\ zebra \mid image)$

$$P(x \mid y) = \frac{P(y \mid x)\, P(x)}{P(y)} = \frac{\text{likelihood} \cdot \text{prior}}{\text{evidence}}$$

Object categorization: the statistical viewpoint

- MAP decision: $p(zebra \mid image)$ vs. $p(no\ zebra \mid image)$

- Bayes rule:

$$\underbrace{p(zebra \mid image)}_{\text{posterior}} \propto \underbrace{p(image \mid zebra)}_{\text{likelihood}} \underbrace{p(zebra)}_{\text{prior}}$$

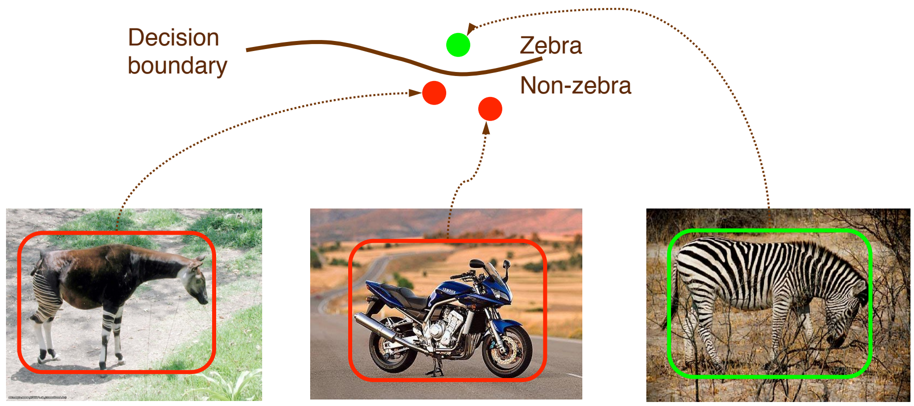
Object categorization: the statistical viewpoint

$$\underbrace{p(zebra \mid image)}_{\text{posterior}} \propto \underbrace{p(image \mid zebra)}_{\text{likelihood}} \underbrace{p(zebra)}_{\text{prior}}$$

- **Discriminative methods: model posterior**
- **Generative methods: model likelihood and prior**

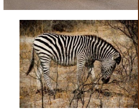
Discriminative methods

- Direct modeling of $p(zebra \mid image)$



Generative methods

- Model $p(image \mid zebra)$ and $p(image \mid no\ zebra)$

	$p(image \mid zebra)$	$p(image \mid no\ zebra)$
	Low	Middle
	High	Middle→Low

Generative vs. discriminative methods

- Generative methods
  - + Interpretable
  - + Can be learned using images from just a single category
  - – Sometimes we don't need to model the likelihood when all we want is to make a decision
- Discriminative methods
  - + Efficient
  - + Often produce better classification rates
  - – Can be hard to interpret
  - – Require positive and negative training data

Discriminative Methods

- Object/scene category recognition
- Multi-class classification problem
- Supervised setting
- Given examples of images with category labels
- Learn classifier to predict labels of new images

Discriminative methods

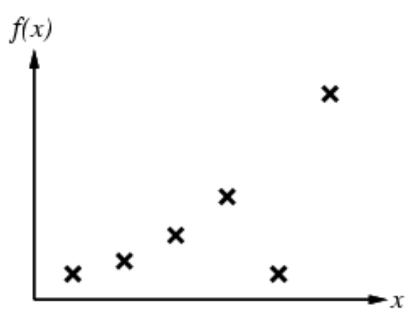
- Brief overview of machine learning
- Linear regression
- Logistic regression
- Decision trees
- Boosting idea
- SVM

Slides from Russell and Norvig, AI book

Inductive learning method

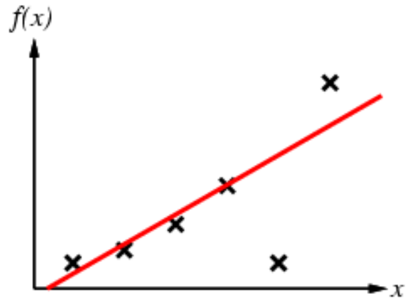
- Construct/adjust *h* to agree with *f* on training set
- (*h* is consistent if it agrees with *f* on all examples)

- E.g., curve fitting:



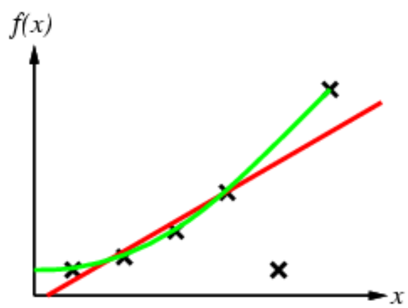
Inductive learning method

- Construct/adjust *h* to agree with *f* on training set
- (*h* is consistent if it agrees with *f* on all examples)
- E.g., curve fitting:



Inductive learning method

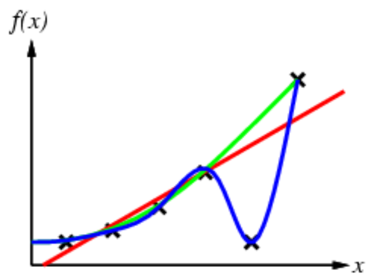
- Construct/adjust *h* to agree with *f* on training set
- (*h* is consistent if it agrees with *f* on all examples)
- E.g., curve fitting:



Inductive learning method

- Construct/adjust *h* to agree with *f* on training set
- (*h* is consistent if it agrees with *f* on all examples)

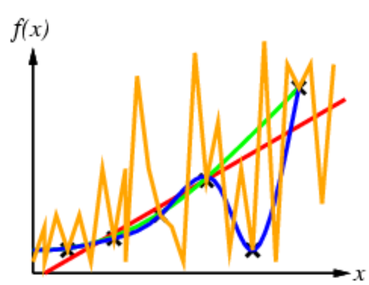
- E.g., curve fitting:



Inductive learning method

- Construct/adjust *h* to agree with *f* on training set
- (*h* is consistent if it agrees with *f* on all examples)

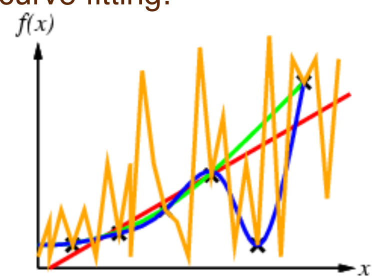
- E.g., curve fitting:



Inductive learning method

- Construct/adjust *h* to agree with *f* on training set
- (*h* is consistent if it agrees with *f* on all examples)

- E.g., curve fitting:



- Ockham's razor: prefer the simplest hypothesis consistent with data

Learning decision trees

- Problem: decide whether to wait for a table at a restaurant, based on the following attributes:
  1. Alternate: is there an alternative restaurant nearby?
  2. Bar: is there a comfortable bar area to wait in?
  3. Fri/Sat: is today Friday or Saturday?
  4. Hungry: are we hungry?
  5. Patrons: number of people in the restaurant (None, Some, Full)
  6. Price: price range ($, $$, $$$)
  7. Raining: is it raining outside?
  8. Reservation: have we made a reservation?
  9. Type: kind of restaurant (French, Italian, Thai, Burger)
  10. WaitEstimate: estimated waiting time (0-10, 10-30, 30-60, >60)

Attribute-based representations

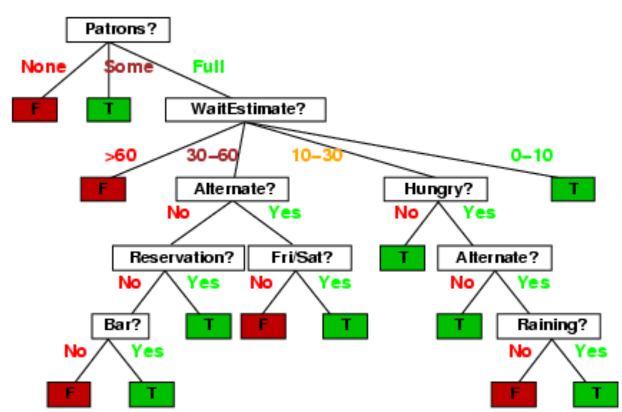
- Examples described by attribute values (Boolean, discrete, continuous)
- E.g., situations where I will/won't wait for a table:

Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*Wait*
X_1	T	F	F	T	Some	\$\$\$	F	T	French	0–10	T
X_2	T	F	F	T	Full	\$	F	F	Thai	30–60	F
X_3	F	T	F	F	Some	\$	F	F	Burger	0–10	T
X_4	T	F	T	T	Full	\$	F	F	Thai	10–30	T
X_5	T	F	T	F	Full	\$\$\$	F	T	French	>60	F
X_6	F	T	F	T	Some	\$\$	T	T	Italian	0–10	T
X_7	F	T	F	F	None	\$	T	F	Burger	0–10	F
X_8	F	F	F	T	Some	\$\$	T	T	Thai	0–10	T
X_9	F	T	T	F	Full	\$	T	F	Burger	>60	F
X_{10}	T	T	T	T	Full	\$\$\$	F	T	Italian	10–30	F
X_{11}	F	F	F	F	None	\$	F	F	Thai	0–10	F
X_{12}	T	T	T	T	Full	\$	F	F	Burger	30–60	T

- Classification of examples is positive (T) or negative (F)

Decision trees

- One possible representation for hypotheses
- E.g., here is the “true” tree for deciding whether to wait:



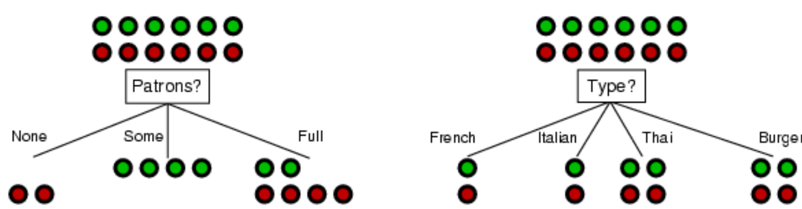
Decision tree learning

- Aim: find a small tree consistent with the training examples
- Idea: (recursively) choose "most significant" attribute as root of (sub)tree

```
function DTL(examples, attributes, default) returns a decision tree
  if examples is empty then return default
  else if all examples have the same classification then return the classification
  else if attributes is empty then return MODE(examples)
  else
     best ← CHOOSE-ATTRIBUTE(attributes, examples)
     tree ← a new decision tree with root test best
     for each value v_i of best do
        examples_i ← {elements of examples with best = v_i}
        subtree ← DTL(examples_i, attributes − best, MODE(examples))
        add a branch to tree with label v_i and subtree subtree
     return tree
```

Choosing an attribute

- Idea: a good attribute splits the examples into subsets that are (ideally) "all positive" or "all negative"



- *Patrons?* is a better choice

Using information theory

- To implement `Choose-Attribute` in the DTL algorithm
- Information Content (Entropy):
  - $I(P(v_1), \ldots, P(v_n)) = \Sigma_{i=1} -P(v_i) \log_2 P(v_i)$
- For a training set containing *p* positive examples and *n* negative examples:

$$I(\frac{p}{p+n},\frac{n}{p+n}) = -\frac{p}{p+n}\log_2\frac{p}{p+n} - \frac{n}{p+n}\log_2\frac{n}{p+n}$$

Information gain

- A chosen attribute *A* divides the training set *E* into subsets $E_1, \ldots, E_v$ according to their values for *A*, where *A* has v distinct values.

$$remainder(A) = \sum_{i=1}^{v}\frac{p_i+n_i}{p+n} I(\frac{p_i}{p_i+n_i},\frac{n_i}{p_i+n_i})$$

- Information Gain (IG) or reduction in entropy from the attribute test:

$$IG(A) = I(\frac{p}{p+n},\frac{n}{p+n}) - remainder(A)$$

- Choose the attribute with the largest IG

Information gain

- For the training set, $p = n = 6$, $I(6/12, 6/12) = 1$ bit
- Consider the attributes *Patrons* and *Type* (and others too):

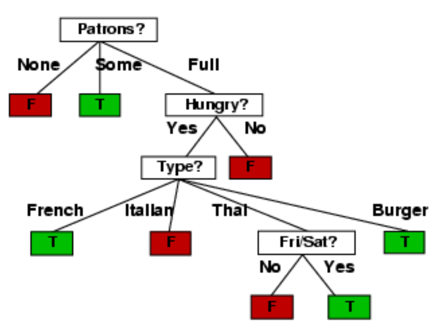
$$IG(Patrons) = 1 - [\frac{2}{12}I(0,1) + \frac{4}{12}I(1,0) + \frac{6}{12}I(\frac{2}{6},\frac{4}{6})] = .0541 \text{ bits}$$

$$IG(Type) = 1 - [\frac{2}{12}I(\frac{1}{2},\frac{1}{2}) + \frac{2}{12}I(\frac{1}{2},\frac{1}{2}) + \frac{4}{12}I(\frac{2}{4},\frac{2}{4}) + \frac{4}{12}I(\frac{2}{4},\frac{2}{4})] = 0 \text{ bits}$$

- *Patrons* has the highest IG of all attributes and so is chosen by the DTL algorithm as the root

Example contd.

- Decision tree learned from the 12 examples:



- Substantially simpler than “true” tree---a more complex hypothesis isn’t justified by small amount of data

Linear regression

- Fit function to the data so you can predict future values
- Given data *(x1,y1), (x2, y2), … (xn, yn)*
- Find such hypothesis f, such that y = f(x) has small error of future data
- Choose the form of function and estimate the data using linear least squares techniques (in closed form, or gradient descent)

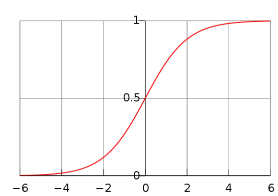
Logistic Regression

- Similar as linear regression, but now *y's* are only +/- or 0/1
- denoting positive or negative examples of a class
- We know that our function should have values 0 or 1
- Transform to continuous setting – construct such as *h* where function would have values only between 0-1

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}}$$

- g(.) is a sigmoid, logistic function

$$g(z) = \frac{1}{1 + e^{-z}}$$



- Property of the derivative

$$g'(z) = g(z)(1 - g(z))$$

Logistic Regression

- How do we find parameter θ
- Find such parameters so as to maximize likelihood of the data assume that

$$
\begin{aligned}
P(y=1|x;\theta) &= h_\theta(x) \\
P(y=0|x;\theta) &= 1-h_\theta(x)
\end{aligned}
$$

- More compactly

$$p(y|x;\theta) = (h_\theta)^y(1-h_\theta)^{(1-y)}$$

- Resulting gradient ascent rule

$$\theta_j = \theta_j + \alpha(y^{(i)} - h_\theta(x^{(i)}))x^{(i)}$$

- Closely related to perceptron learning algorithm where values are forced to be 0-1 - no clear probabilistic interpretation

Getting Ready

- Install vlfeat.org
- Install openCV
- svmlib

Instance Based Learning

- Nearest neighbor methods
- Non-parametric learning
- Define similarity measure
- Hamming distance with Boolean attributes

- K-d trees
- Locality sensitive hashing
- Min-hash

Distance Functions

- Minkowski Distance Lp norm

$$L_p(x_i, x_q) = (\sum_j (x_{j,i} - x_{q,i})^p)^{1/p}$$

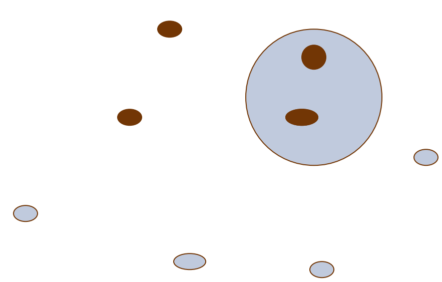
- For p = 1 Manhattan distance (often used for dissimilar attributes)
- For p = 2 Euclidean Distance
- Normalize each dimension (compute mean and standard deviation) and rescale all values to zero mean and unit variance
- Mahalanobis Distance – takes into account covariance between dimensions – where S is a covariance matrix

$$d(x_i, x_q) = \sqrt{(\vec{x} - \vec{y})^T S^{-1} (\vec{x} - \vec{y})}$$

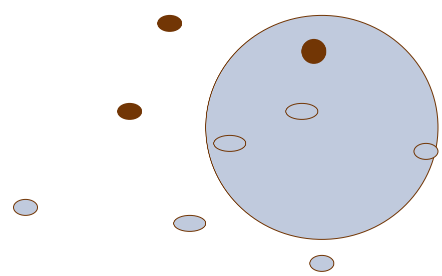
K-Nearest Neighbor

- Features
  - All instances correspond to points in an n-dimensional Euclidean space
  - Classification is delayed till a new instance arrives
  - Classification done by comparing feature vectors of the different points
  - Target function may be discrete or real-valued

1-Nearest Neighbor



3-Nearest Neighbor

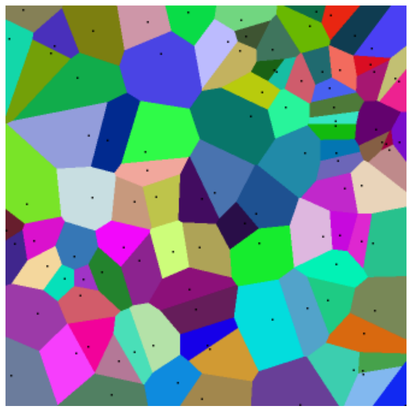


K-Nearest Neighbor

- An arbitrary instance is represented by
  $(a_1(x), a_2(x), a_3(x),.., a_n(x))$
  - $a_i(x)$ denotes features
- Euclidean distance between two instances
  $d(x_i, x_j)$=sqrt (sum for r=1 to n $(a_r(x_i) - a_r(x_j))^2$)
- Continuous valued target function
  - mean value of the k nearest training examples

Voronoi Diagram

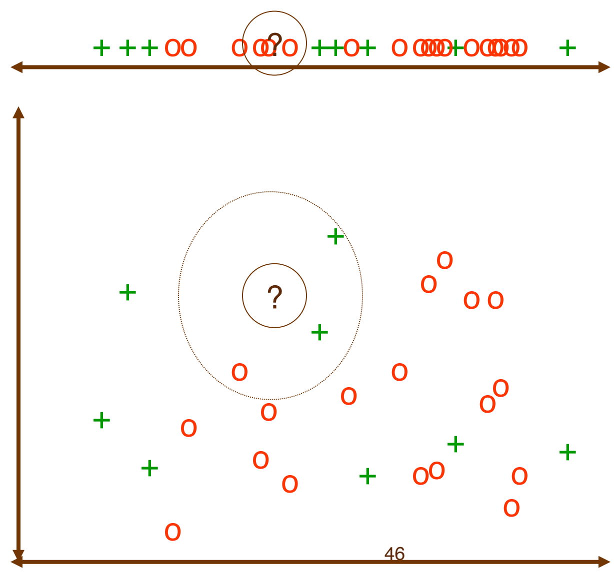
- Decision surface formed by the training examples



Distance-Weighted Nearest Neighbor Algorithm

- Assign weights to the neighbors based on their 'distance' from the query point
  - Weight 'may' be inverse square of the distances
- All training points may influence a particular instance
- Points in the local neighbourhood will influence an instance

K-NN and irrelevant features



46

Remarks

- Highly effective inductive inference method for noisy training data and complex target functions
- Target function for a whole space may be described as a combination of less complex local approximations
- Learning is very simple
- Classification is time consuming

- Curse of dimensionality

Curse of dimensionality

- Curse of dimensionality – nearest neighbors in high-dimensional spaces are very far – we often do not have enough data
- E.g. for N uniformly distributed points in n dimensional space length of average neighbourhood is
- Average volume containing k points
- Average of the edge length of the cube

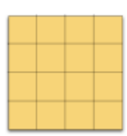
$$l^n = k / N$$

$$l = (k / N)^{1/n}$$

- *For n=2 , l = 0.003*
- For *n=17* half edge length
- For *n=200* 94% of the length on the unit cube

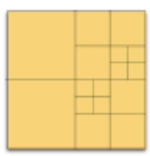
Nearest neighbours

- Instance based learning is simple and effective
- Naively comparing a test example against entire training set is expensive
- Important issue – structure data intelligently in order to avoid comparing against every point
- Structuring multi-dimensional data:
- Option: k-dimensional array of buckets; uniformly partition each dimension (ideal if data is uniformly distributed – which is never the case in multiple dimensions
- 2D example



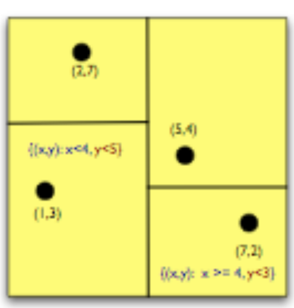
Nearest Neighbours

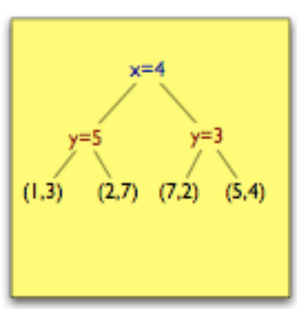
- Another alternative – quad tree – each dimension partitioned in two – the cells with lot of data are partitioned further
- Binary tree splits each dimension in half
- Memory intensive, in 2D quadtree



Nearest Neighbours

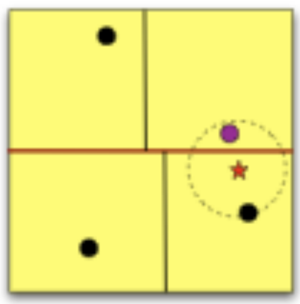
- k-d Tree - binary tree, which has both dimension number and splitting values at each node
- Splits the space into (hyper) rectangles
- Searching for a point in the tree is accomplished by traversing the tree *O(lg(n))* if there are n datapoints





Nearest Neighbour Seach

- Depth first search to nearest neighbour
- End up in hyper rectangle, find the nearest point there – not necessarily the nearest neighbour
- There can be another point then leaf which is closer



Constructing a kD-tree

- Select dimension to spliting values
- Select the dimension to split on
- Do this recursively for each child of the split
- Want to split along dimension along which examples are well spread
  1. choose the dimension with greatest variances
  2. choose the dimension with greatest range
- Want to split in the middle – median, mean

Constructing KD tree

```
procedure KDCONSTRUCT(trainingSet)
                                                  ▷ Returns a kdTree
    if trainingSet.isEmpty() then
        return emptyKdtree
    end if
    (s, val) ← CHOOSESPLIT(trainingSet)           ▷ s is splitting dimension
    trainLeft ← {x ∈ trainingSet : x_s < val}
    trainRight ← {x ∈ trainingSet : x_s ≥ val}
    kdLeft ← KDCONSTRUCT(trainLeft)
    kdRight ← KDCONSTRUCT(trainRight)
    return kdtree(s, val, kdLeft, kdRight)
end procedure
```

References

- Wikipedia on kD-tree
- Introductory tutorial in kD-tree Andrew Moore

Quantization

- KD trees effective approximate NN search

- Alternatives for local feature representations
- Quantization of the descriptor space
- Takes advantage of the repeatability of descriptors
- Many descriptors occur in more then one image

- Generates descriptor codebook – visual vocabulary
- Compact representation of local features

K-means

- One of the most popular iterative descent clustering methods.

- Features: quantitative type.

- Dissimilarity measure: Euclidean distance.

- First flat k-means

K-means

The "***within cluster point scatter***" becomes:

$$W(C) = \frac{1}{2} \sum_{k=1}^{K} \sum_{i \in C_k} \sum_{j \in C_k} \left\| \boldsymbol{x}_i - \boldsymbol{x}_j \right\|^2$$

$W(C)$ can be rewritten as:

$$W(C) = \sum_{k=1}^{K} |C_k| \sum_{i \in C_k} \left\| \boldsymbol{x}_i - \bar{\boldsymbol{x}}_k \right\|^2$$

(obtained by rewriting $(\boldsymbol{x}_i - \boldsymbol{x}_j) = (\boldsymbol{x}_i - \bar{\boldsymbol{x}}_k) - (\boldsymbol{x}_j - \bar{\boldsymbol{x}}_k)$)

where

$\bar{\boldsymbol{x}}_k = \frac{1}{|C_k|} \sum_{i \in C_k} \boldsymbol{x}_i$ is the mean vector of cluster C_k

$|C_k|$ is the number of points in cluster C_k

K-means

The objective is:

$$\min_{C} \sum_{k=1}^{K} |C_k| \sum_{i \in C_k} \left\| \boldsymbol{x}_i - \bar{\boldsymbol{x}}_k \right\|^2$$

We can solve this problem by noticing:

for any set of data S

$$\bar{\boldsymbol{x}}_S = \arg\min_{\boldsymbol{m}} \sum_{i \in S} \left\| \boldsymbol{x}_i - \boldsymbol{m} \right\|^2$$

(this is obtained by setting $\frac{\partial \sum_{i \in S} \left\| \boldsymbol{x}_i - \boldsymbol{m} \right\|^2}{\partial \boldsymbol{m}} = 0$)

So we can solve the enlarged optimization problem:

$$\min_{C, \boldsymbol{m}_k} \sum_{k=1}^{K} |C_k| \sum_{i \in C_k} \left\| \boldsymbol{x}_i - \boldsymbol{m}_k \right\|^2$$

K-means: The Algorithm

1. Given a cluster assignment C, the total within cluster scatter

$$\sum_{k=1}^{K} |C_k| \sum_{i \in C_k} \|\boldsymbol{x}_i - \boldsymbol{m}_k\|^2 \text{ is minimized with respect to the } \{\boldsymbol{m}_1, \cdots, \boldsymbol{m}_K\}$$

giving the means of the currently assigned clusters;

2. Given a current set of means $\{\boldsymbol{m}_1, \cdots, \boldsymbol{m}_K\}$,

$$\sum_{k=1}^{K} |C_k| \sum_{i \in C_k} \|\boldsymbol{x}_i - \boldsymbol{m}_k\|^2 \text{ is minimized with respect to } C$$

by assigning each point to the closest current cluster mean;

3. Steps 1 and 2 are iterated until the assignments do not change.

K-means: Properties and Limitations

- The algorithm converges to a local minimum
- The solution depends on the initial partition
- One should start the algorithm with many different random choices for the initial means, and choose the solution having smallest value of the objective function

K-means: Properties and Limitations

- The algorithm is sensitive to outliers
- A variation of K-means improves upon robustness (K-medoids):
  - Centers for each cluster are restricted to be one of the points assigned to the cluster;
  - The center (*medoid*) is set to be the point that minimizes the total distance to other points in the cluster;
  - K-medoids is more computationally intensive than K-means.

K-means: Properties and Limitations

- The algorithm requires the number of clusters *K*;
- Often *K* is unknown, and must be estimated from the data:

We can test $K \in \{1, 2, \cdots, K_{\max}\}$

Compute $\{W_1, W_2, \cdots, W_{\max}\}$

In general: $W_1 > W_2 > \cdots > W_{\max}$

$K^* =$ actual number of clusters in the data,

when $K < K^*$, we can expect $W_K >> W_{K+1}$

when $K > K^*$, further splits provide smaller decrease of W

Set $\hat{K}^*$ by identifying an "elbow shape" in the plot of W_k

An Application of K-means: (Lossy) Data compression

- Original image has N pixels
- Each pixel → (R,G,B) values
- Each value is stored with 8 bits of precision
- Transmitting the whole image costs 24N bits

Compression achieved by K-means:

- Identify each pixel with the corresponding centroid
- We have K such centroids → we need $\log_2 K$ bits per pixel
- For each centroid we need 24 bits
- Transmitting the whole image costs $24K + N \log 2K$ bits
- Original image = 240x180=43,200 pixels → 43,200x24=1,036,800 bits
- Compressed images:

K=2: 43,248 bits   K=3: 86,472   K=10: 173,040 bits

Applications Computer Vision Object Recognition, Image Based Retrieval

Applications of NN and clustering techniques in computer vision

Example Problems

- **Problem1:** Given an image find me similar image/ duplicate in the database
- **Problem 2:** Given an image, is this an image of a car ? Given database of images labeled as cars, faces, aeroplanes
- **Problem 3:** Given a document, what is the topic of the document ?
- **Problem 4:** Given an image, find me the closest image (looking most alike) in a large database

- Issues: definition of the similarity between two images, two documents (the key), deployment of clustering, large scale instance based learning

Recognition using local features

- Define a set of local feature templates (we will discuss in more detail how to do this in Computer Vision section)

- Define similarity measure between two images

- Bags of visual features models

Bag-of-features models

Many slides adapted from S. Lazebnik, Fei-Fei Li, Rob Fergus, and Antonio Torralba and many others

Origin 2: Bag-of-words models

- Orderless document representation: frequencies of words from a dictionary Salton & McGill (1983)

Origin 2: Bag-of-words models

- Orderless document representation: frequencies of words from a dictionary Salton & McGill (1983)

Origin 2: Bag-of-words models

- Orderless document representation: frequencies of words from a dictionary Salton & McGill (1983)

Origin 2: Bag-of-words models

- Orderless document representation: frequencies of words from a dictionary Salton & McGill (1983)

Bags of features for object recognition

face, flowers, building

- Works pretty well for image-level classification

Bag of features: outline

1. Extract features

Bag of features: outline

1. Extract features
2. Learn “visual vocabulary”

Bag of features: outline

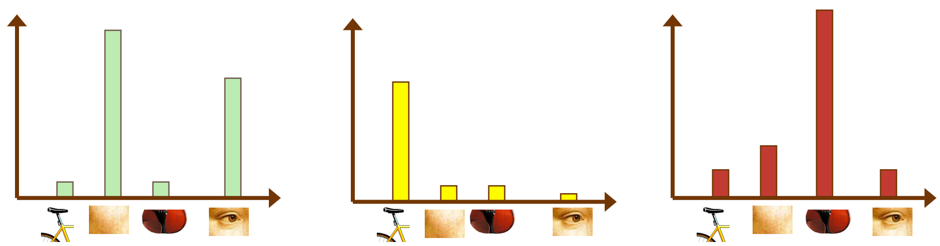
1. Extract features
2. Learn “visual vocabulary”
3. Quantize features using visual vocabulary

Bag of features: outline

1. Extract features
2. Learn “visual vocabulary”
3. Quantize features using visual vocabulary

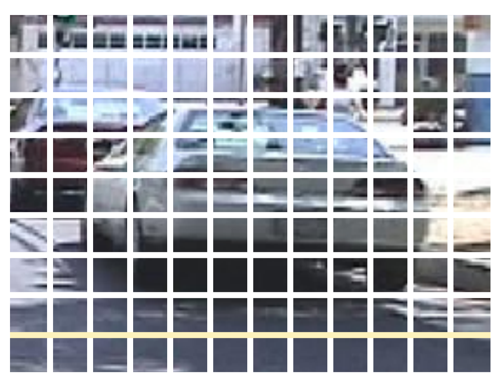
Bag of features: outline

1. Extract features
2. Learn "visual vocabulary"
3. Quantize features using visual vocabulary
4. Represent images by frequencies of "visual words"



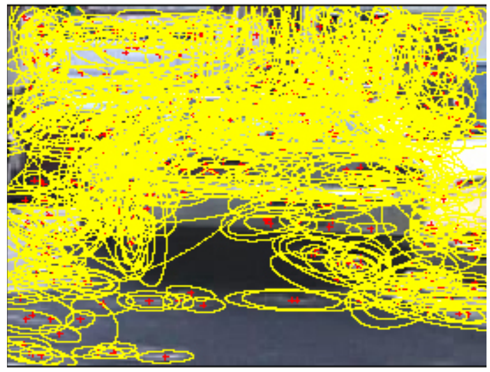
1. Feature extraction

- Regular grid
  - Vogel & Schiele, 2003
  - Fei-Fei & Perona, 2005

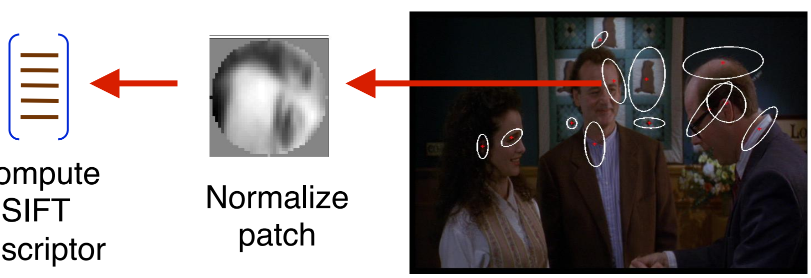


1. Feature extraction

- Regular grid
  - Vogel & Schiele, 2003
  - Fei-Fei & Perona, 2005
- Interest point detector
  - Csurka et al. 2004
  - Fei-Fei & Perona, 2005
  - Sivic et al. 2005



1. Feature extraction



Compute SIFT descriptor

Normalize patch

Detect patches

[Lowe'99]

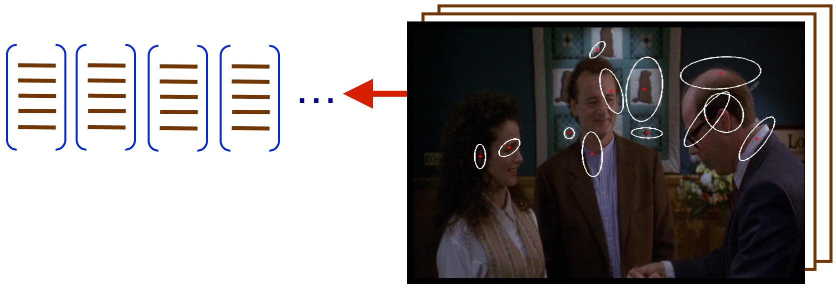
[Mikojaczyk and Schmid '02]

[Mata, Chum, Urban & Pajdla, '02]

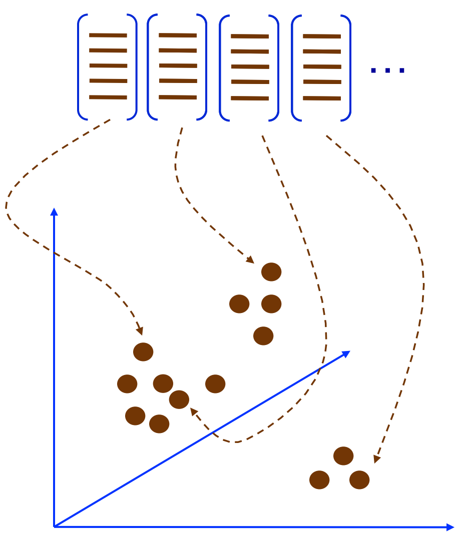
[Sivic & Zisserman, '03]

Slide credit: Josef Sivic

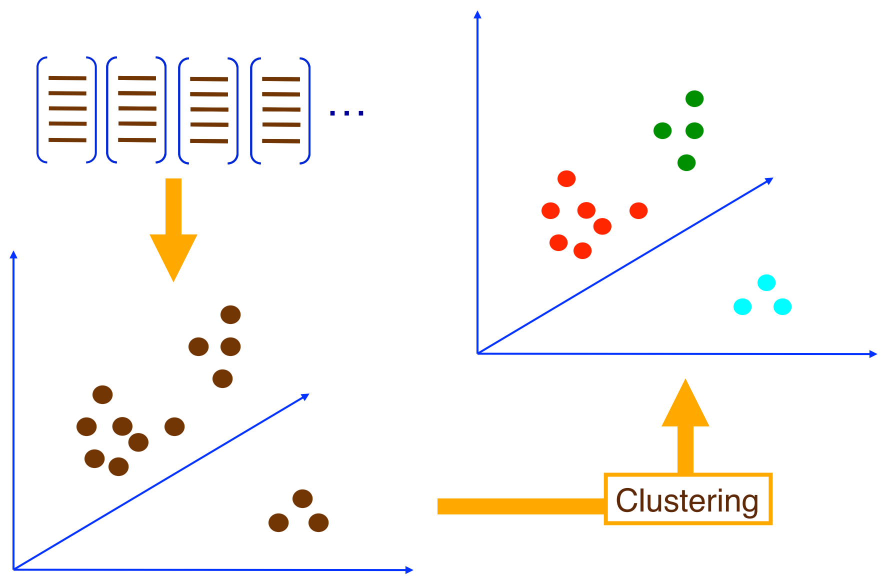
1. Feature extraction



2. Learning the visual vocabulary

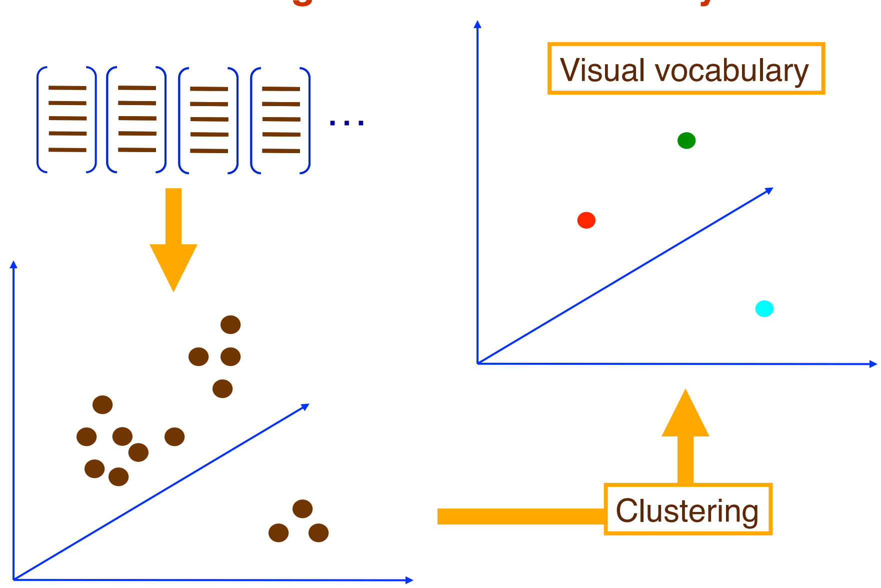


2. Learning the visual vocabulary



Slide credit: Josef Sivic

2. Learning the visual vocabulary



Slide credit: Josef Sivic

K-means clustering

- Want to minimize sum of squared Euclidean distances between points x_i and their nearest cluster centers m_k

$$D(X,M) = \sum_{\text{cluster } k} \sum_{\substack{\text{point } i \text{ in} \\ \text{cluster } k}} (x_i - m_k)^2$$

- Algorithm:
- Randomly initialize K cluster centers
- Iterate until convergence:
  - Assign each data point to the nearest center
  - Recompute each cluster center as the mean of all points assigned to it

Expectation Maximization

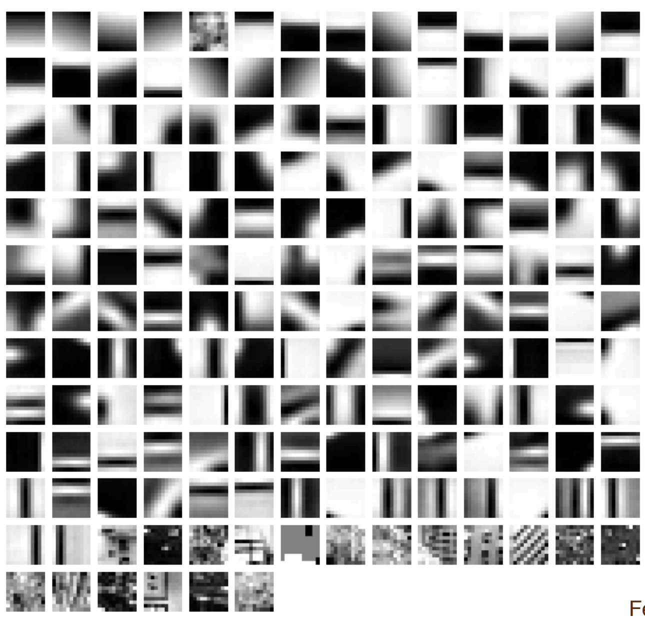
- Probabilistic version of k-means clustering
- Soft assignments of points to clusters
- Weighted recomputation of cluster centers
- Probabilistic formulation (20.3)

- Blackboard

From clustering to vector quantization

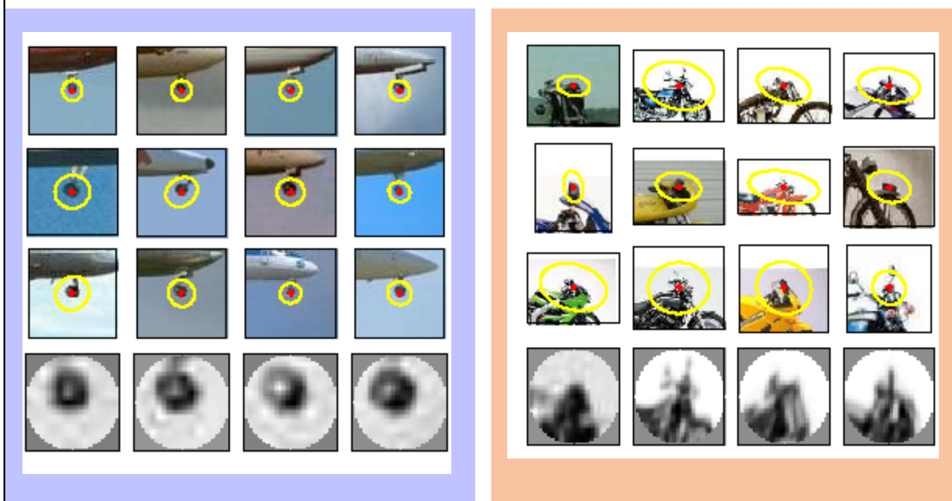
- Clustering is a common method for learning a visual vocabulary or codebook
  - Unsupervised learning process
  - Each cluster center produced by k-means becomes a codevector
  - Codebook can be learned on separate training set
  - Provided the training set is sufficiently representative, the codebook will be “universal”
- The codebook is used for quantizing features
  - A *vector quantizer* takes a feature vector and maps it to the index of the nearest codevector in a codebook
  - Codebook = visual vocabulary
  - Codevector = visual word

Example visual vocabulary



Fei-Fei et al. 2005

Image patch examples of visual words



Sivic et al. 2005

Visual vocabularies: Issues

- How to choose vocabulary size?
  - Too small: visual words not representative of all patches
  - Too large: quantization artifacts, overfitting
- Generative or discriminative learning?
- Computational efficiency
  - Vocabulary trees
    (Nister & Stewenius, 2006)

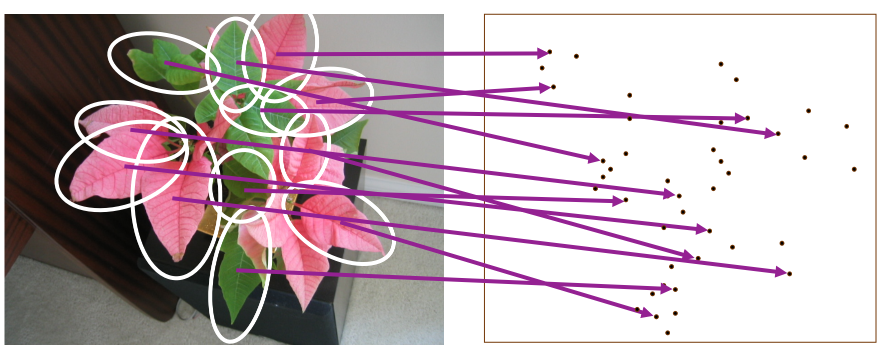
Hierarchical k-means

- We have many, many of these features
- 100000 images ~1000 features per image
- If we can get repeatable, discriminative features,
- then recognition can scale to very large databases
- using the vocabulary tree and indexing approach

- Quantize the feature descriptor space + efficient search
- Hierarchical k-means - Nister&Stewenius [CVPR 2006]
- Visual vocabulary trees

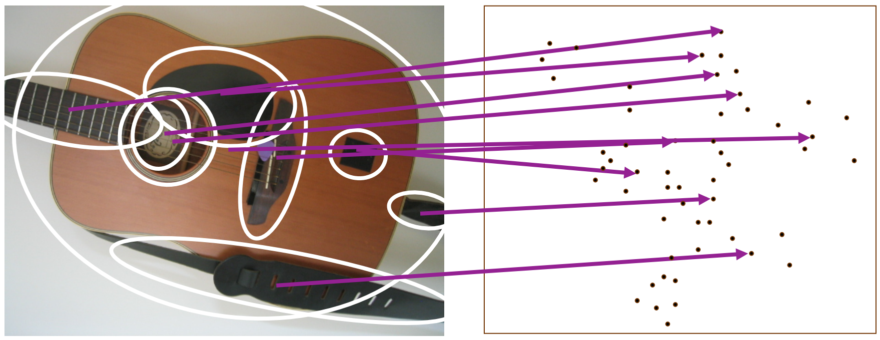
Slides from Nister & Stewenius 06

Building Visual Vocabulary Tree



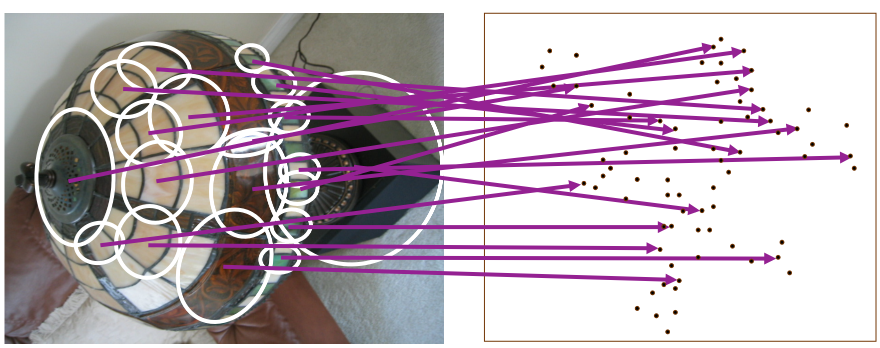
Slides from Nister & Stewenius 06

Building Visual Vocabulary Tree



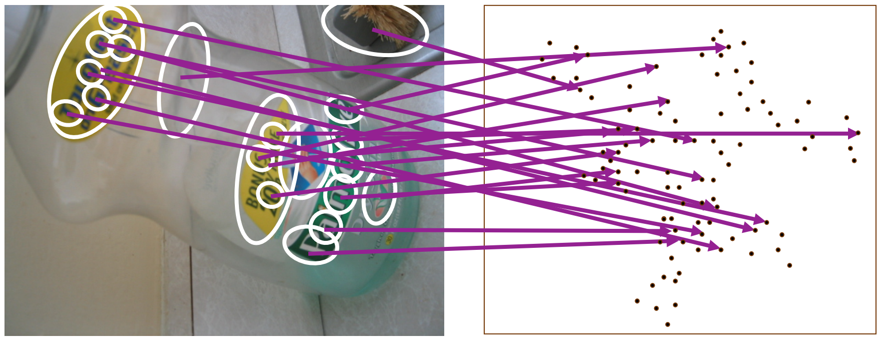
Slides from Nister & Stewenius 06

Building Visual Vocabulary Tree



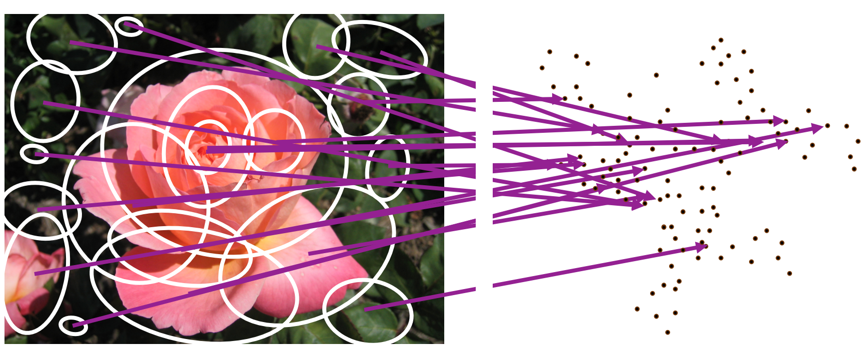
Slides from Nister & Stewenius 06

Building Visual Vocabulary Tree

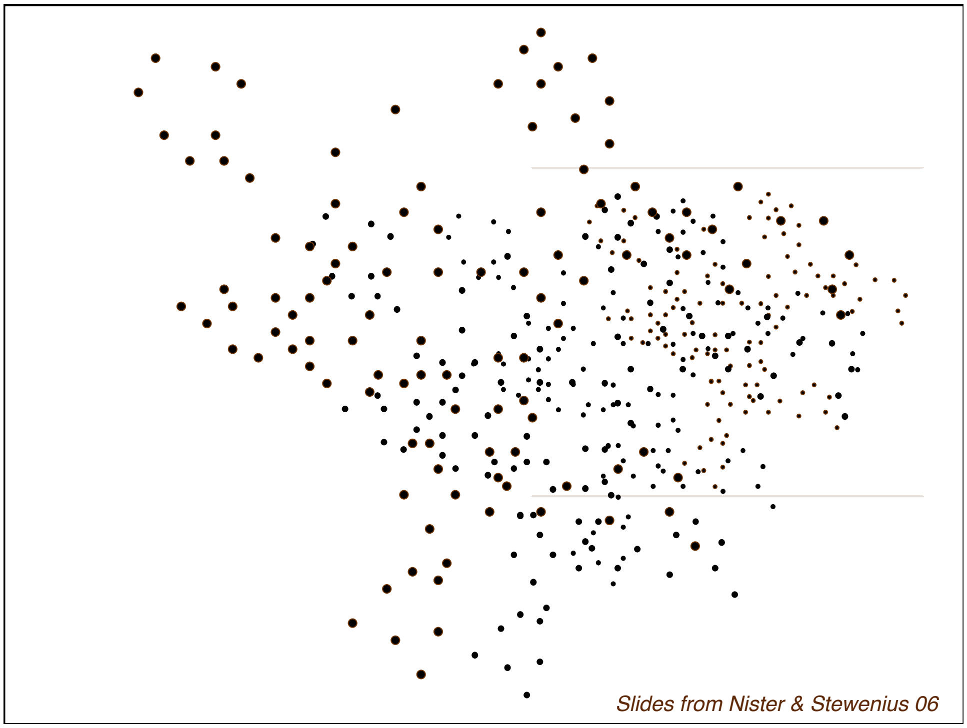


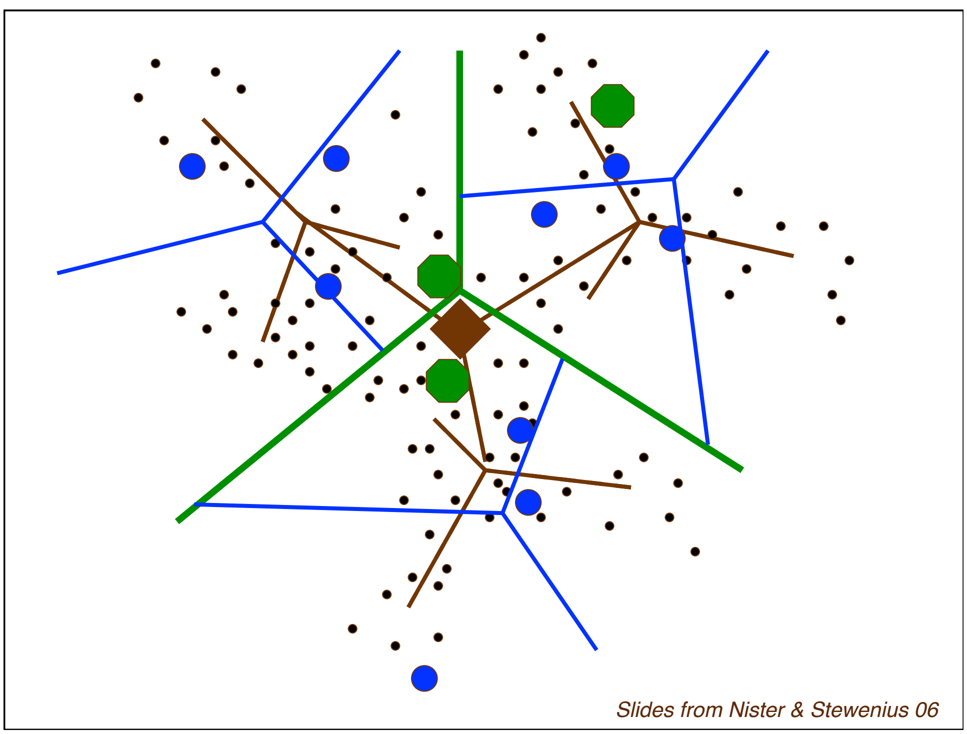
Slides from Nister & Stewenius 06

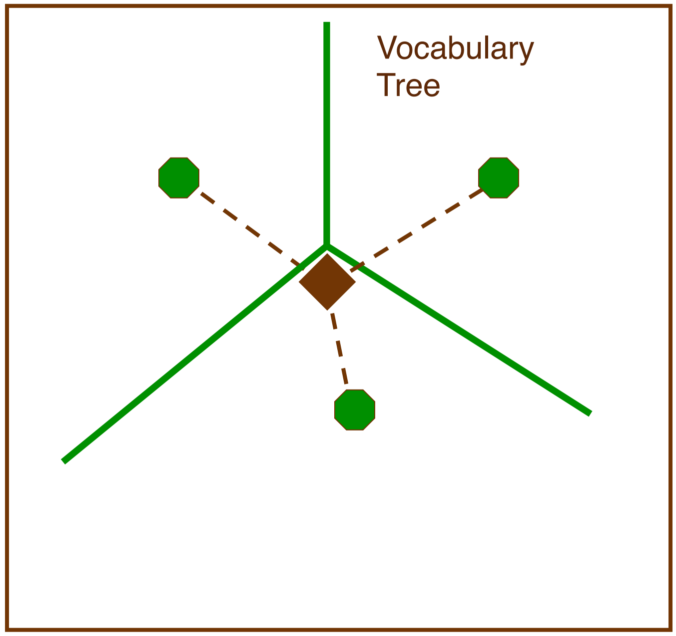
Building Visual Vocabulary Tree



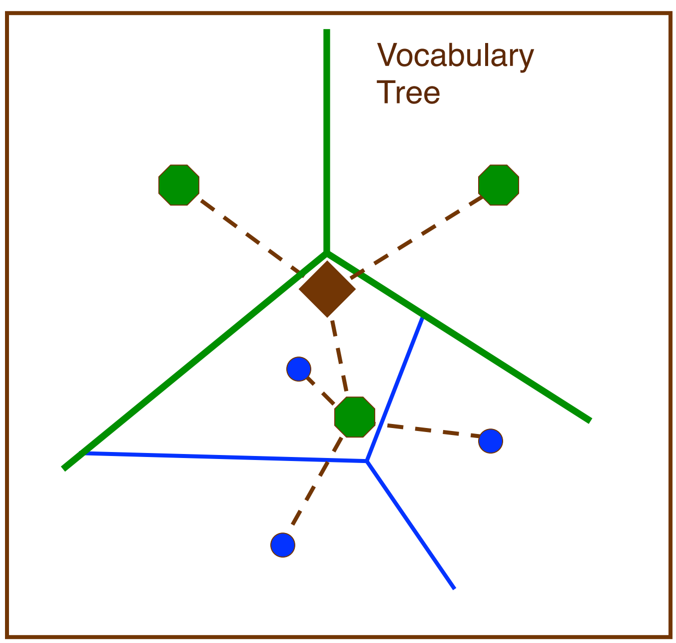
Slides from Nister & Stewenius 06


Slides from Nister & Stewenius 06

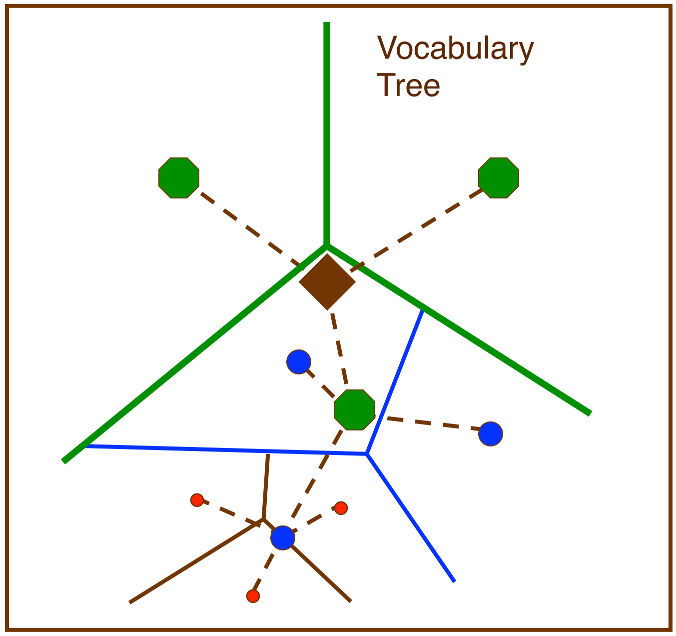

Slides from Nister & Stewenius 06



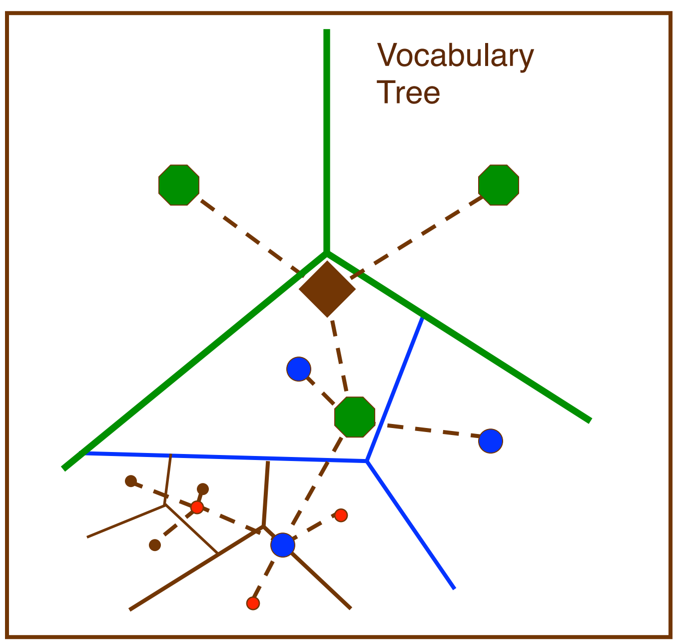
Slides from Nister & Stewenius 06



Slides from Nister & Stewenius 06



Slides from Nister & Stewenius 06



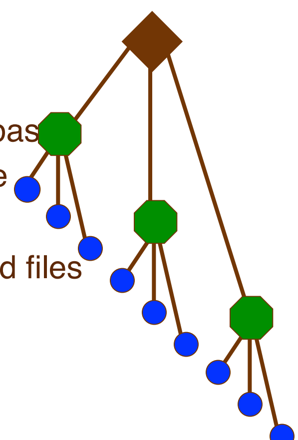
Slides from Nister & Stewenius 06

Vocabulary Trees

- Easy to add/remove images from the database
- Suitable for incremental approach
- Suitable for creating single generic vocabulary
-
- **Approach**
- Extract descriptors from many/many images
- Acquire enough statistics about the descriptor distribution
- Run k-means hierarchically k- is the branching factor of the tree
- E.g. Branching factor of 10 and 6 levels – million leaves

Slides from Nister & Stewenius 06

Vocabulary Trees

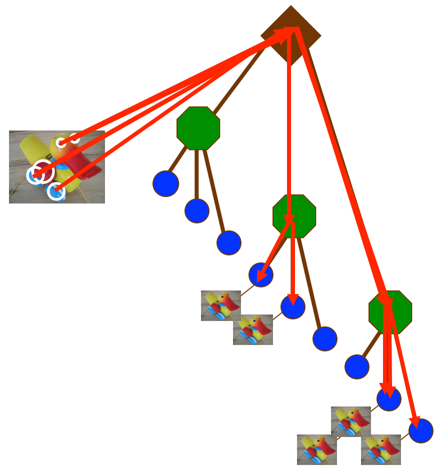


- Training phase – add images to the databas
- Extract descriptors – drop it down the tree
- Each node has an inverted file index
- Index to that image is added to all inverted files

- When we want to query image
- Pass each descriptor down the tree
- Accumulate scores for each image in the database

- At each level do k dot products total of kL dot products
- For k^L leafs and integer descriptors we need Dk^L bytes for 1M leaf
- nodes use 143 MB of memory

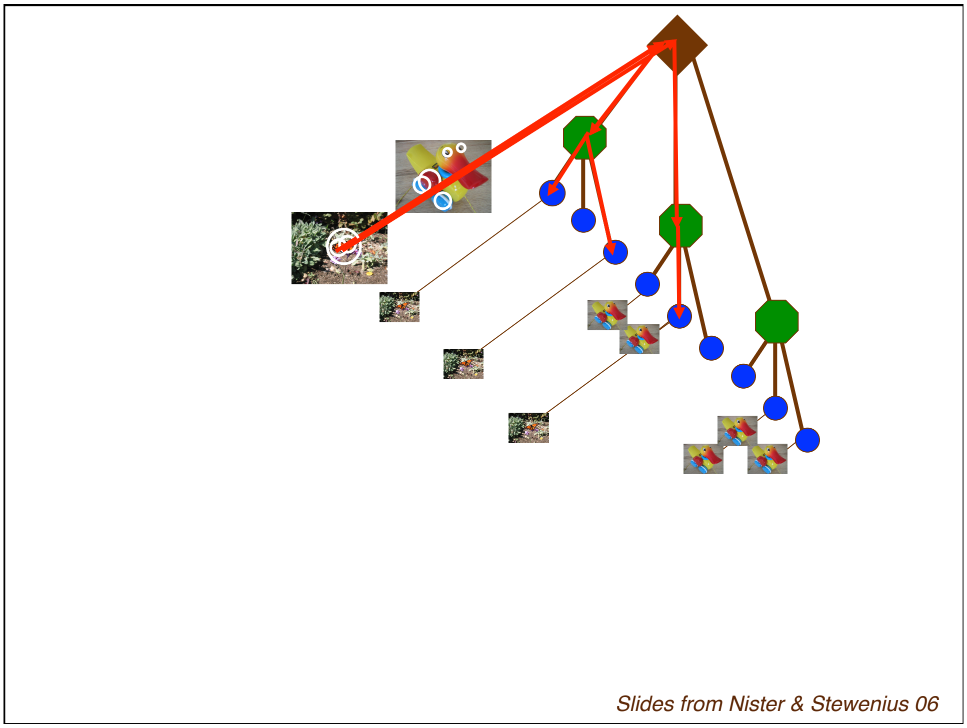
Slides from Nister & Stewenius 06

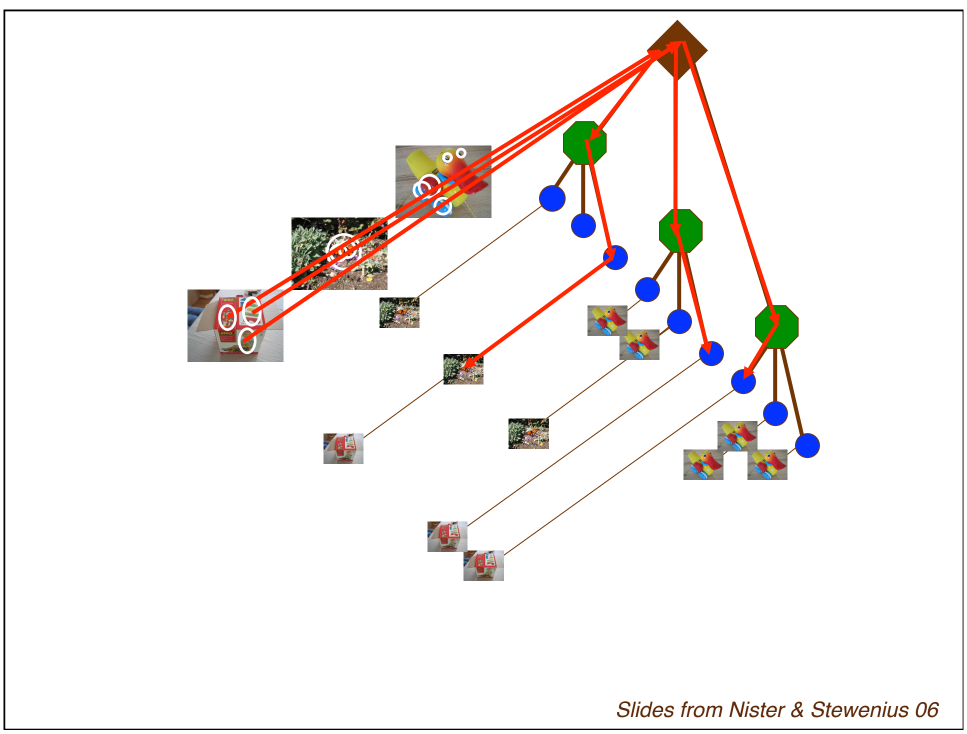


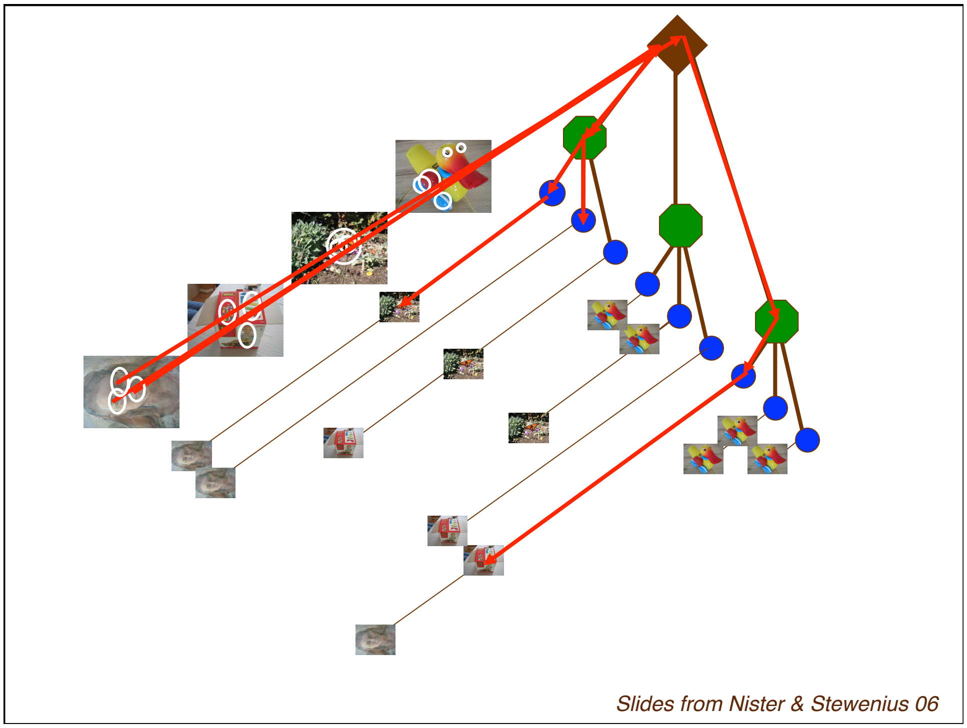
Slides from Nister & Stewenius 06

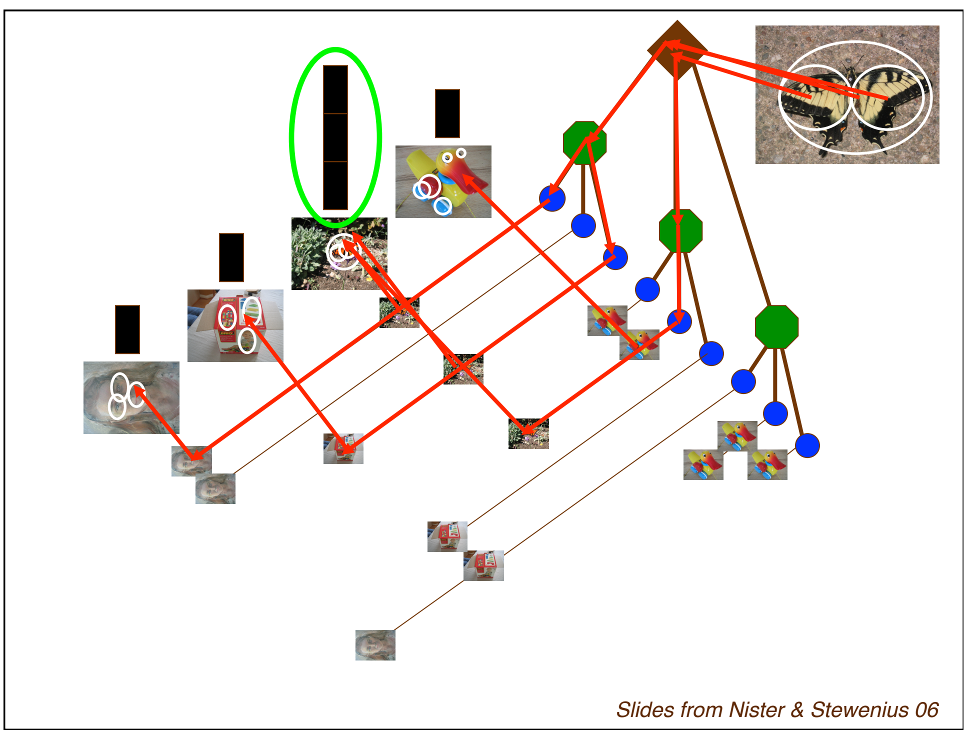
Application

- Vocabulary trees for finding closest image in the database


Slides from Nister & Stewenius 06


Slides from Nister & Stewenius 06


Slides from Nister & Stewenius 06


Slides from Nister & Stewenius 06

TF-IDF scoring

- TF-IDF term frequency – inverse document frequency
- Used in the information retrieval and text mining
- To evaluate how important is a word to document

- Importance depends on how many times the word appears in document – offset by number of occurrence of that word in the whole document corpus
- In image based retrieval
- image ~ document analogy
- visual word ~ word analogy

TF-IDF scoring

- TF-IDF term frequency – inverse document frequency
- Number of occurrences of a word in a document / number of occurrences of all words in the document

$$\mathrm{tf}_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}} \qquad |d : t_i \in d|$$

- Number of documents / number of documents where term appears

$$\mathrm{idf}_{i,j} = \log \frac{|D|}{|\{d : t_i \in d\}|} \qquad |D|$$

- High weight of a word/term is when it has high frequency and low term document frequency

$$\mathrm{tfidf}_{i,j} = \mathrm{tf}_{i,j} \times \mathrm{idf}_i$$

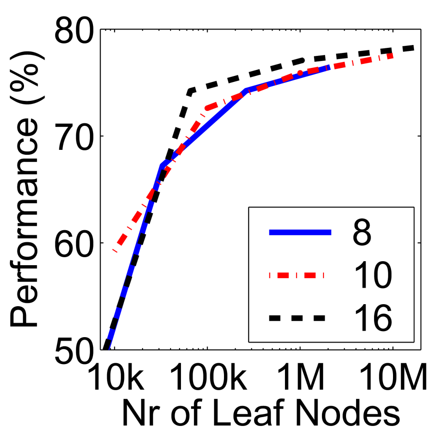
Inverted file index

- Idea – large number of images (documents) – how to find out quickly which image does the word occur ?
- Forward index – list of words per document
- Inverted index – list of documents per word
- Once the word is encountered – we can quickly figure out which image it belongs to

Word 1 {1, 2 , 10, 12}
Word 2 {0, 2, 100, 7, 18}
Word 3 {5, 10, 12}
…

Size Matters

Performance improves with the
Size of the database



Improves Retrieval

Improves Speed

Here the results of particular object instance retrieval, database
Of ~ 40,000 objects, real-time performance

Slides from Nister & Stewenius 06