Visual Recognition: Combining Features

Raquel Urtasun

TTI Chicago

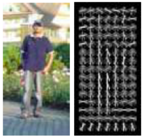
Feb 14, 2012

Which detectors?

Window-based

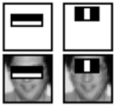
NN + scene Gist classification

e.g., Hays & Efros



SVM + person detection

e.g., Dalal & Triggs



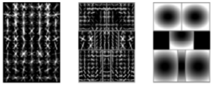
Boosting + face detection

Viola & Jones

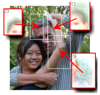
Part-based

BOW, pyramids
e.g., [Grauman et al.]

ISM: voting
e.g., [Leibe & Shiele]



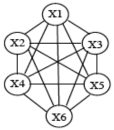
deformable parts
e.g., [Felzenszwalb et al.]



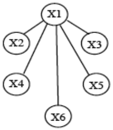
poselets
[Bourdev et al.]

Models of local features

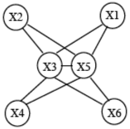
- How is spatial information encoded for models with bad of features?
- See [Carneiro et al. 06] for a comprehensive study of all possibilities.



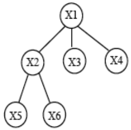
a) Constellation [13]



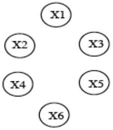
b) Star shape [9, 14]



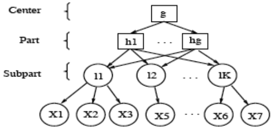
c) k-fan ($k = 2$) [9]



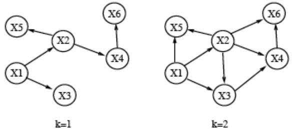
d) Tree [12]



e) Bag of features [10, 21]

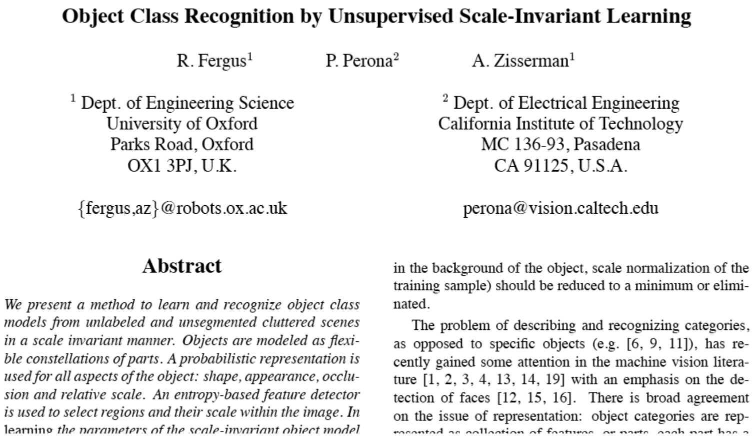


f) Hierarchy [4]



g) Sparse flexible model

Constellation Model



Object Class Recognition by Unsupervised Scale-Invariant Learning

R. Fergus[1] P. Perona[2] A. Zisserman[1]

[1] Dept. of Engineering Science
University of Oxford
Parks Road, Oxford
OX1 3PJ, U.K.

{fergus,az}@robots.ox.ac.uk

[2] Dept. of Electrical Engineering
California Institute of Technology
MC 136-93, Pasadena
CA 91125, U.S.A.

perona@vision.caltech.edu

Abstract

We present a method to learn and recognize object class models from unlabeled and unsegmented cluttered scenes in a scale invariant manner. Objects are modeled as flexible constellations of parts. A probabilistic representation is used for all aspects of the object: shape, appearance, occlusion and relative scale. An entropy-based feature detector is used to select regions and their scale within the image. In learning the parameters of the scale-invariant object model

in the background of the object, scale normalization of the training sample) should be reduced to a minimum or eliminated.

The problem of describing and recognizing categories, as opposed to specific objects (e.g. [6, 9, 11]), has recently gained some attention in the machine vision literature [1, 2, 3, 4, 13, 14, 19] with an emphasis on the detection of faces [12, 15, 16]. There is broad agreement on the issue of representation: object categories are rep-

Main idea

- An object model consists of a number of parts.
- Each part has an appearance, relative scale and can be occluded or not.

Main idea

- An object model consists of a number of parts.
- Each part has an appearance, relative scale and can be occluded or not.
- Shape is represented by the mutual position of the parts.

Main idea

- An object model consists of a number of parts.
- Each part has an appearance, relative scale and can be occluded or not.
- Shape is represented by the mutual position of the parts.
- The entire model is generative and probabilistic, so appearance, scale, shape and occlusion are all modeled by pdf, i.e., Gaussians.

Main idea

- An object model consists of a number of parts.
- Each part has an appearance, relative scale and can be occluded or not.
- Shape is represented by the mutual position of the parts.
- The entire model is generative and probabilistic, so appearance, scale, shape and occlusion are all modeled by pdf, i.e., Gaussians.
- Learning: first detecting regions and their scales, and then estimating the parameters of the above densities from these regions using max. likelihood.

Main idea

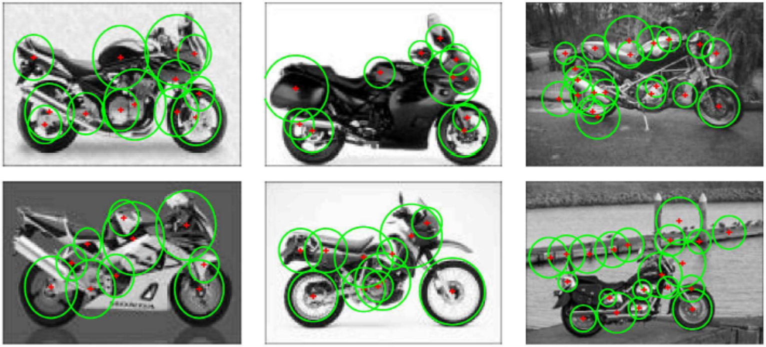
- An object model consists of a number of parts.
- Each part has an appearance, relative scale and can be occluded or not.
- Shape is represented by the mutual position of the parts.
- The entire model is generative and probabilistic, so appearance, scale, shape and occlusion are all modeled by pdf, i.e., Gaussians.
- Learning: first detecting regions and their scales, and then estimating the parameters of the above densities from these regions using max. likelihood.
- Recognition by first detecting regions and their scales, and then evaluating the regions in a Bayesian manner, using the model parameters estimated in the learning.
- In this setting we do not know where the object of interest is in the image.

Main idea

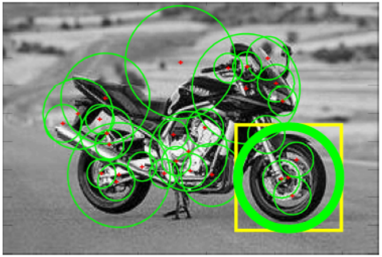
- An object model consists of a number of parts.
- Each part has an appearance, relative scale and can be occluded or not.
- Shape is represented by the mutual position of the parts.
- The entire model is generative and probabilistic, so appearance, scale, shape and occlusion are all modeled by pdf, i.e., Gaussians.
- Learning: first detecting regions and their scales, and then estimating the parameters of the above densities from these regions using max. likelihood.
- Recognition by first detecting regions and their scales, and then evaluating the regions in a Bayesian manner, using the model parameters estimated in the learning.
- In this setting we do not know where the object of interest is in the image.

Detecting Feature Points

- Kadir & Brady saliency region detector



Constellation Model



- Find regions within image
- Use salient region operator (Kadir & Brady 01)

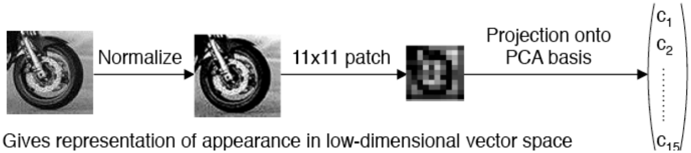
Location

(x,y) coords. of region centre

Scale

Radius of region (pixels)

Appearance



Gives representation of appearance in low-dimensional vector space

Generative probabilistic model

- We have identified N image features, with locations $\mathbf{X}$, scales $\mathbf{S}$ and appearances $\mathbf{A}$.
- We define a generative model with P parts and parameters θ as

$$\begin{aligned} p(\mathbf{X}, \mathbf{S}, \mathbf{A}|\theta) &= \sum_{\mathbf{h}\in\mathcal{H}} p(\mathbf{X}, \mathbf{S}, \mathbf{A}, \mathbf{h}|\theta) \\ &= \sum_{h\in\mathcal{H}} \underbrace{p(\mathbf{A}|\mathbf{X}, \mathbf{S}, \mathbf{h}, \theta)}_{appearance} \underbrace{p(\mathbf{X}|\mathbf{S}, \mathbf{h}, \theta)}_{shape} \underbrace{p(\mathbf{S}|\mathbf{h}, \theta)}_{Rel.scale} p(\mathbf{h}|\theta) \end{aligned}$$

with $\mathbf{h}$ an indexing variable, called hypothesis.

Generative probabilistic model

- We have identified N image features, with locations $\mathbf{X}$, scales $\mathbf{S}$ and appearances $\mathbf{A}$.
- We define a generative model with P parts and parameters θ as

$$\begin{aligned} p(\mathbf{X},\mathbf{S},\mathbf{A}|\theta) &= \sum_{\mathbf{h}\in\mathcal{H}} p(\mathbf{X},\mathbf{S},\mathbf{A},\mathbf{h}|\theta) \\ &= \sum_{h\in\mathcal{H}} \underbrace{p(\mathbf{A}|\mathbf{X},\mathbf{S},\mathbf{h},\theta)}_{appearance} \underbrace{p(\mathbf{X}|\mathbf{S},\mathbf{h},\theta)}_{shape} \underbrace{p(\mathbf{S}|\mathbf{h},\theta)}_{Rel.scale} p(\mathbf{h}|\theta) \end{aligned}$$

  with $\mathbf{h}$ an indexing variable, called hypothesis.
- $\mathbf{h}$ is a vector of length P where each entry is between 0 and N (0 is occlusion).

Generative probabilistic model

- We have identified N image features, with locations $\mathbf{X}$, scales $\mathbf{S}$ and appearances $\mathbf{A}$.
- We define a generative model with P parts and parameters θ as

$$\begin{aligned} p(\mathbf{X}, \mathbf{S}, \mathbf{A}|\theta) &= \sum_{\mathbf{h} \in \mathcal{H}} p(\mathbf{X}, \mathbf{S}, \mathbf{A}, \mathbf{h}|\theta) \\ &= \sum_{h \in \mathcal{H}} \underbrace{p(\mathbf{A}|\mathbf{X}, \mathbf{S}, \mathbf{h}, \theta)}_{appearance} \underbrace{p(\mathbf{X}|\mathbf{S}, \mathbf{h}, \theta)}_{shape} \underbrace{p(\mathbf{S}|\mathbf{h}, \theta)}_{Rel.scale} p(\mathbf{h}|\theta) \end{aligned}$$

  with $\mathbf{h}$ an indexing variable, called hypothesis.
- $\mathbf{h}$ is a vector of length P where each entry is between 0 and N (0 is occlusion).
- The set $\mathcal{H}$ has complexity $O(N^P)$.

Generative probabilistic model

- We have identified N image features, with locations $\mathbf{X}$, scales $\mathbf{S}$ and appearances $\mathbf{A}$.
- We define a generative model with P parts and parameters θ as

$$\begin{aligned} p(\mathbf{X},\mathbf{S},\mathbf{A}|\theta) &= \sum_{\mathbf{h}\in\mathcal{H}} p(\mathbf{X},\mathbf{S},\mathbf{A},\mathbf{h}|\theta) \\ &= \sum_{h\in\mathcal{H}} \underbrace{p(\mathbf{A}|\mathbf{X},\mathbf{S},\mathbf{h},\theta)}_{appearance}\underbrace{p(\mathbf{X}|\mathbf{S},\mathbf{h},\theta)}_{shape}\underbrace{p(\mathbf{S}|\mathbf{h},\theta)}_{Rel.scale}p(\mathbf{h}|\theta) \end{aligned}$$

  with $\mathbf{h}$ an indexing variable, called hypothesis.
- $\mathbf{h}$ is a vector of length P where each entry is between 0 and N (0 is occlusion).
- The set $\mathcal{H}$ has complexity $O(N^P)$.
- Decision made base on the ratio

$$\frac{p(\mathbf{X},\mathbf{S},\mathbf{A}|\theta)p(\text{object})}{p(\mathbf{X},\mathbf{S},\mathbf{A}|\theta_{bg})p(\text{No-object})}$$

- Learning using EM

Generative probabilistic model

- We have identified N image features, with locations $\mathbf{X}$, scales $\mathbf{S}$ and appearances $\mathbf{A}$.
- We define a generative model with P parts and parameters θ as

$$\begin{aligned} p(\mathbf{X}, \mathbf{S}, \mathbf{A}|\theta) &= \sum_{\mathbf{h} \in \mathcal{H}} p(\mathbf{X}, \mathbf{S}, \mathbf{A}, \mathbf{h}|\theta) \\ &= \sum_{h \in \mathcal{H}} \underbrace{p(\mathbf{A}|\mathbf{X}, \mathbf{S}, \mathbf{h}, \theta)}_{appearance} \underbrace{p(\mathbf{X}|\mathbf{S}, \mathbf{h}, \theta)}_{shape} \underbrace{p(\mathbf{S}|\mathbf{h}, \theta)}_{Rel.scale} p(\mathbf{h}|\theta) \end{aligned}$$

  with $\mathbf{h}$ an indexing variable, called hypothesis.
- $\mathbf{h}$ is a vector of length P where each entry is between 0 and N (0 is occlusion).
- The set $\mathcal{H}$ has complexity $O(N^P)$.
- Decision made base on the ratio

$$\frac{p(\mathbf{X}, \mathbf{S}, \mathbf{A}|\theta) p(\text{object})}{p(\mathbf{X}, \mathbf{S}, \mathbf{A}|\theta_{bg}) p(\text{No-object})}$$

- Learning using EM

More explicitly ...

- The generative model is defined as

$$p(\mathbf{X}, \mathbf{S}, \mathbf{A}|\theta) = \sum_{h \in \mathcal{H}} \underbrace{p(\mathbf{A}|\mathbf{X}, \mathbf{S}, \mathbf{h}, \theta)}_{appearance} \underbrace{p(\mathbf{X}|\mathbf{S}, \mathbf{h}, \theta)}_{shape} \underbrace{p(\mathbf{S}|\mathbf{h}, \theta)}_{Rel.scale} p(\mathbf{h}|\theta)$$

- Let $\mathbf{d} = sign(\mathbf{h})$ tells which parts are background, n the number of background features, and f the number of foreground features.

More explicitly ...

- The generative model is defined as

$$p(\mathbf{X},\mathbf{S},\mathbf{A}|\theta) = \sum_{h\in\mathcal{H}} \underbrace{p(\mathbf{A}|\mathbf{X},\mathbf{S},\mathbf{h},\theta)}_{appearance} \underbrace{p(\mathbf{X}|\mathbf{S},\mathbf{h},\theta)}_{shape} \underbrace{p(\mathbf{S}|\mathbf{h},\theta)}_{Rel.scale} p(\mathbf{h}|\theta)$$

- Let $\mathbf{d} = sign(\mathbf{h})$ tells which parts are background, n the number of background features, and f the number of foreground features.
- **Appearance** is represented with a Gaussian with diagonal covariance

$$\frac{p(\mathbf{A}|\mathbf{X},\mathbf{S},\mathbf{h},\theta)}{p(\mathbf{A}|\mathbf{X},\mathbf{S},\mathbf{h},\theta_{bg})} = \prod_{p=1}^{P}\left(\frac{\mathcal{N}(\mathbf{A}(h_p)|\mathbf{c}_p,\mathbf{V}_p)}{\mathcal{N}(\mathbf{A}(h_p)|\mathbf{c}_{bg},\mathbf{V}_{bg})}\right)^{d_p}$$

More explicitly ...

- The generative model is defined as

$$p(\mathbf{X}, \mathbf{S}, \mathbf{A}|\theta) = \sum_{h \in \mathcal{H}} \underbrace{p(\mathbf{A}|\mathbf{X}, \mathbf{S}, \mathbf{h}, \theta)}_{appearance} \underbrace{p(\mathbf{X}|\mathbf{S}, \mathbf{h}, \theta)}_{shape} \underbrace{p(\mathbf{S}|\mathbf{h}, \theta)}_{Rel.scale} p(\mathbf{h}|\theta)$$

- Let $\mathbf{d} = sign(\mathbf{h})$ tells which parts are background, n the number of background features, and f the number of foreground features.
- **Appearance** is represented with a Gaussian with diagonal covariance

$$\frac{p(\mathbf{A}|\mathbf{X}, \mathbf{S}, \mathbf{h}, \theta)}{p(\mathbf{A}|\mathbf{X}, \mathbf{S}, \mathbf{h}, \theta_{bg})} = \prod_{p=1}^{P} \left(\frac{\mathcal{N}(\mathbf{A}(h_p)|\mathbf{c}_p, \mathbf{V}_p)}{\mathcal{N}(\mathbf{A}(h_p)|\mathbf{c}_{bg}, \mathbf{V}_{bg})} \right)^{d_p}$$

- **Shape** is represented by a joint Gaussian density (full covariance) of the locations of features within a hypothesis in scale-invariant space

$$\frac{p(\mathbf{X}|\mathbf{S}, \mathbf{h}, \theta)}{p(\mathbf{X}|\mathbf{S}, \mathbf{h}, \theta_{bg})} = \mathcal{N}(\mathbf{X}(\mathbf{h})|\mu, \Sigma)\alpha^f$$

More explicitly ...

- The generative model is defined as

$$p(\mathbf{X}, \mathbf{S}, \mathbf{A}|\theta) = \sum_{h \in \mathcal{H}} \underbrace{p(\mathbf{A}|\mathbf{X}, \mathbf{S}, \mathbf{h}, \theta)}_{appearance} \underbrace{p(\mathbf{X}|\mathbf{S}, \mathbf{h}, \theta)}_{shape} \underbrace{p(\mathbf{S}|\mathbf{h}, \theta)}_{Rel.scale} p(\mathbf{h}|\theta)$$

- Let $\mathbf{d} = sign(\mathbf{h})$ tells which parts are background, n the number of background features, and f the number of foreground features.
- **Appearance** is represented with a Gaussian with diagonal covariance

$$\frac{p(\mathbf{A}|\mathbf{X}, \mathbf{S}, \mathbf{h}, \theta)}{p(\mathbf{A}|\mathbf{X}, \mathbf{S}, \mathbf{h}, \theta_{bg})} = \prod_{p=1}^{P} \left(\frac{\mathcal{N}(\mathbf{A}(h_p)|\mathbf{c}_p, \mathbf{V}_p)}{\mathcal{N}(\mathbf{A}(h_p)|\mathbf{c}_{bg}, \mathbf{V}_{bg})} \right)^{d_p}$$

- **Shape** is represented by a joint Gaussian density (full covariance) of the locations of features within a hypothesis in scale-invariant space

$$\frac{p(\mathbf{X}|\mathbf{S}, \mathbf{h}, \theta)}{p(\mathbf{X}|\mathbf{S}, \mathbf{h}, \theta_{bg})} = \mathcal{N}(\mathbf{X}(\mathbf{h})|\mu, \Sigma)\alpha^f$$

More explicitly ...

- The generative model is defined as

$$p(\mathbf{X}, \mathbf{S}, \mathbf{A}|\theta) = \sum_{h \in \mathcal{H}} \underbrace{p(\mathbf{A}|\mathbf{X}, \mathbf{S}, \mathbf{h}, \theta)}_{appearance} \underbrace{p(\mathbf{X}|\mathbf{S}, \mathbf{h}, \theta)}_{shape} \underbrace{p(\mathbf{S}|\mathbf{h}, \theta)}_{Rel.scale} p(\mathbf{h}|\theta)$$

- Let $\mathbf{d} = sign(\mathbf{h})$ tells which parts are background, n the number of background features, and f the number of foreground features.
- **Relative Scale**: The scale of each part p relative to a reference frame is modeled by a Gaussian density, where the parts are assumed to be independent of one another. Background is uniform.

$$\frac{p(\mathbf{S}|\mathbf{h}, \theta)}{p(\mathbf{S}|\mathbf{h}, \theta_{bg})} = \prod_{p=1}^{P} \mathcal{N}(\mathbf{S}(h_p)|t_p, U_p)^{d_p} r^f$$

More explicitly ...

- The generative model is defined as

$$p(\mathbf{X},\mathbf{S},\mathbf{A}|\theta) = \sum_{h\in\mathcal{H}} \underbrace{p(\mathbf{A}|\mathbf{X},\mathbf{S},\mathbf{h},\theta)}_{appearance} \underbrace{p(\mathbf{X}|\mathbf{S},\mathbf{h},\theta)}_{shape} \underbrace{p(\mathbf{S}|\mathbf{h},\theta)}_{Rel.scale} p(\mathbf{h}|\theta)$$

- Let $\mathbf{d} = sign(\mathbf{h})$ tells which parts are background, n the number of background features, and f the number of foreground features.
- **Relative Scale**: The scale of each part p relative to a reference frame is modeled by a Gaussian density, where the parts are assumed to be independent of one another. Background is uniform.

$$\frac{p(\mathbf{S}|\mathbf{h},\theta)}{p(\mathbf{S}|\mathbf{h},\theta_{bg})} = \prod_{p=1}^{P} \mathcal{N}(\mathbf{S}(h_p)|t_p, U_p)^{d_p} r^f$$

- $p(\mathbf{h}|\theta)$ modeled using a Poisson distribution, book-keeping and a prob. table for all possible occlusion patters.

More explicitly ...

- The generative model is defined as

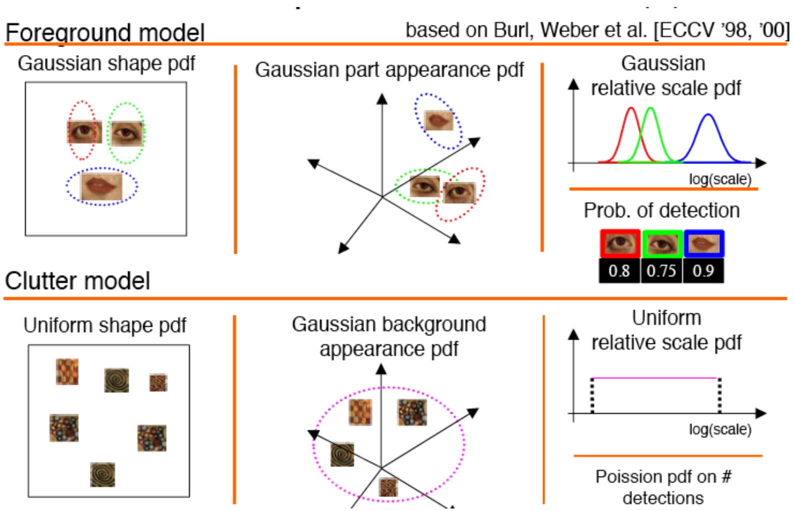
$$p(\mathbf{X}, \mathbf{S}, \mathbf{A}|\theta) = \sum_{h \in \mathcal{H}} \underbrace{p(\mathbf{A}|\mathbf{X}, \mathbf{S}, \mathbf{h}, \theta)}_{appearance} \underbrace{p(\mathbf{X}|\mathbf{S}, \mathbf{h}, \theta)}_{shape} \underbrace{p(\mathbf{S}|\mathbf{h}, \theta)}_{Rel.scale} p(\mathbf{h}|\theta)$$

- Let $\mathbf{d} = sign(\mathbf{h})$ tells which parts are background, n the number of background features, and f the number of foreground features.
- **Relative Scale**: The scale of each part p relative to a reference frame is modeled by a Gaussian density, where the parts are assumed to be independent of one another. Background is uniform.

$$\frac{p(\mathbf{S}|\mathbf{h}, \theta)}{p(\mathbf{S}|\mathbf{h}, \theta_{bg})} = \prod_{p=1}^{P} \mathcal{N}(\mathbf{S}(h_p)|t_p, U_p)^{d_p} r^f$$

- $p(\mathbf{h}|\theta)$ modeled using a Poisson distribution, book-keeping and a prob. table for all possible occlusion patters.

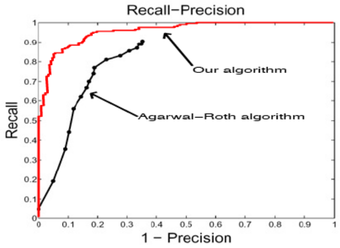
Generative probabilistic model



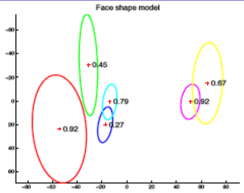
Results

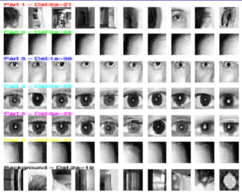
- Simple datasets in 2003

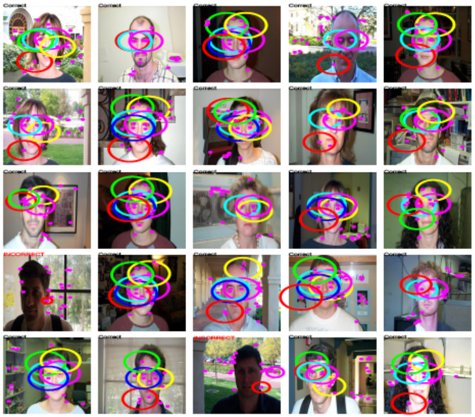
Dataset	Ours	Others	Ref.
Motorbikes	92.5	84	[17]
Faces	96.4	94	[19]
Airplanes	90.2	68	[17]
Cars(Side)	88.5	79	[1]



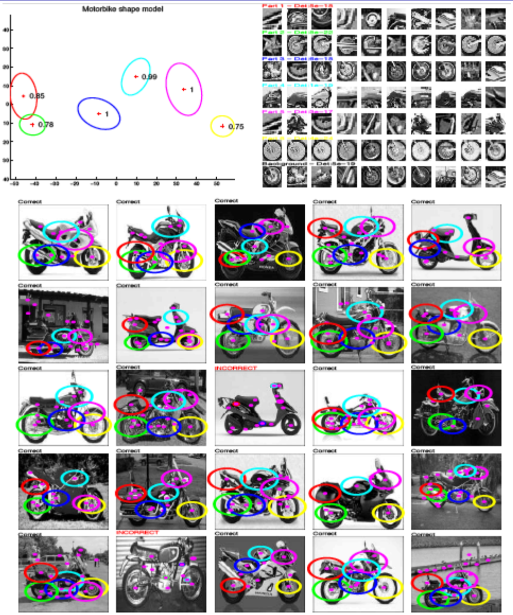
Model examples



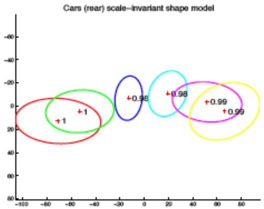




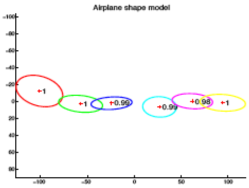
Model examples

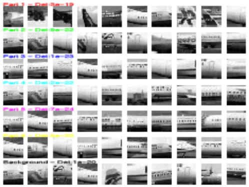


Model examples



Model examples



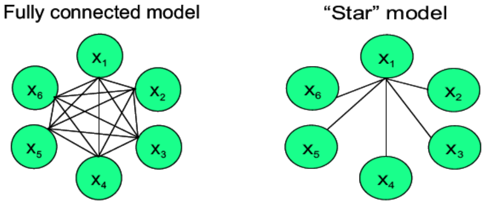


Extensions

- Complexity of the costellation mdoel is too high, i.e., $O(N^P)$
- Use a star model to reduce this to $O(N^2 P)$

$$p(\mathbf{X}|\mathbf{S},\mathbf{h},\theta) = p(x_L|h_L) \prod_{j \neq L} p(x_j|x_L, s_L, h_j, \theta_j)$$

with L the anchor point.



- This can be further improve using distance transform to $O(NP)$

What now?

- We are done with part-based models.
- Let's see something on how to compute multiple sources of information...
- ... and how to learn good representations

Combining information

- We have a lot of different descriptors focusing on, e.g., shape, gradients, texture.
- We have multiple ways to computer similarity (distance) between images (bounding boxes), e.g., histograms, intersection kernels, pyramids.

Combining information

- We have a lot of different descriptors focusing on, e.g., shape, gradients, texture.
- We have multiple ways to computer similarity (distance) between images (bounding boxes), e.g., histograms, intersection kernels, pyramids.
- Which one should we use?

Combining information

- We have a lot of different descriptors focusing on, e.g., shape, gradients, texture.
- We have multiple ways to computer similarity (distance) between images (bounding boxes), e.g., histograms, intersection kernels, pyramids.
- Which one should we use?
- In general there is not a single one that it's always best.

Combining information

- We have a lot of different descriptors focusing on, e.g., shape, gradients, texture.
- We have multiple ways to computer similarity (distance) between images (bounding boxes), e.g., histograms, intersection kernels, pyramids.
- Which one should we use?
- In general there is not a single one that it's always best.
- Even if it was, maybe we can perform better by unifying forces ;)

Combining information

- We have a lot of different descriptors focusing on, e.g., shape, gradients, texture.
- We have multiple ways to computer similarity (distance) between images (bounding boxes), e.g., histograms, intersection kernels, pyramids.
- Which one should we use?
- In general there is not a single one that it's always best.
- Even if it was, maybe we can perform better by unifying forces ;)

Combining information

Multiple ways to combine information

- Stack the feature vectors
- Information fusion

Combining information

Multiple ways to combine information

- Stack the feature vectors
- Information fusion
- Boosting inherently incorporates multiple features

Combining information

Multiple ways to combine information

- Stack the feature vectors
- Information fusion
- Boosting inherently incorporates multiple features
- Use NN with sum of distances or something more clever

Combining information

Multiple ways to combine information

- Stack the feature vectors
- Information fusion
- Boosting inherently incorporates multiple features
- Use NN with sum of distances or something more clever
- Voting via generalized hough transform, with votes coming from different feature types

Combining information

Multiple ways to combine information

- Stack the feature vectors
- Information fusion
- Boosting inherently incorporates multiple features
- Use NN with sum of distances or something more clever
- Voting via generalized hough transform, with votes coming from different feature types
- Multiple kernel learning

Combining information

Multiple ways to combine information

- Stack the feature vectors
- Information fusion
- Boosting inherently incorporates multiple features
- Use NN with sum of distances or something more clever
- Voting via generalized hough transform, with votes coming from different feature types
- Multiple kernel learning
- Random forest

Combining information

Multiple ways to combine information

- Stack the feature vectors
- Information fusion
- Boosting inherently incorporates multiple features
- Use NN with sum of distances or something more clever
- Voting via generalized hough transform, with votes coming from different feature types
- Multiple kernel learning
- Random forest
- etc

Let's look into some of this strategies.

Combining information

Multiple ways to combine information

- Stack the feature vectors
- Information fusion
- Boosting inherently incorporates multiple features
- Use NN with sum of distances or something more clever
- Voting via generalized hough transform, with votes coming from different feature types
- Multiple kernel learning
- Random forest
- etc

Let's look into some of this strategies.

Simple combinations: stacking

- Let $\mathbf{x}_t^{(f)}$ be example t of feature type f.
- We can combine this information by creating a new feature representation $\mathbf{x}_t = [\mathbf{x}_t^{(1)}, \cdots, \mathbf{x}_t^{(F)}]$ for F feature types.

Simple combinations: stacking

- Let $\mathbf{x}_t^{(f)}$ be example t of feature type f.
- We can combine this information by creating a new feature representation $\mathbf{x}_t = [\mathbf{x}_t^{(1)}, \cdots, \mathbf{x}_t^{(F)}]$ for F feature types.
- Problem: features can be of very different mean and variance.

Simple combinations: stacking

- Let $\mathbf{x}_t^{(f)}$ be example t of feature type f.
- We can combine this information by creating a new feature representation $\mathbf{x}_t = [\mathbf{x}_t^{(1)}, \cdots, \mathbf{x}_t^{(F)}]$ for F feature types.
- Problem: features can be of very different mean and variance.
- Typically normalize them to have mean 0 and variance 1 before stacking them.

Simple combinations: stacking

- Let $\mathbf{x}_t^{(f)}$ be example t of feature type f.
- We can combine this information by creating a new feature representation $\mathbf{x}_t = [\mathbf{x}_t^{(1)}, \cdots, \mathbf{x}_t^{(F)}]$ for F feature types.
- Problem: features can be of very different mean and variance.
- Typically normalize them to have mean 0 and variance 1 before stacking them.
- Problem: Dimensionality increases with the number of features.

Simple combinations: stacking

- Let $\mathbf{x}_t^{(f)}$ be example t of feature type f.
- We can combine this information by creating a new feature representation $\mathbf{x}_t = [\mathbf{x}_t^{(1)}, \cdots, \mathbf{x}_t^{(F)}]$ for F feature types.
- Problem: features can be of very different mean and variance.
- Typically normalize them to have mean 0 and variance 1 before stacking them.
- Problem: Dimensionality increases with the number of features.

Ad-hoc Information fusion

- Train a classifier for each feature type (using kernels if wanted)
- Fuse their responses typically by summing the responses

$$f(\mathbf{x}) = \frac{1}{F}\sum_{i=1}^{F} f^{(i)}(\mathbf{x}^{(i)})$$

with f the i-th classifier, which takes as input the i-th feature type.

Ad-hoc Information fusion

- Train a classifier for each feature type (using kernels if wanted)
- Fuse their responses typically by summing the responses

$$f(\mathbf{x}) = \frac{1}{F}\sum_{i=1}^{F} f^{(i)}(\mathbf{x}^{(i)})$$

  with f the i-th classifier, which takes as input the i-th feature type.

- Typically done in the probabilistic setting $f^{(i)}(\mathbf{x}) = p(y|\mathbf{x}^{(i)})$.

Ad-hoc Information fusion

- Train a classifier for each feature type (using kernels if wanted)
- Fuse their responses typically by summing the responses

$$f(\mathbf{x}) = \frac{1}{F}\sum_{i=1}^{F} f^{(i)}(\mathbf{x}^{(i)})$$

  with f the i-th classifier, which takes as input the i-th feature type.
- Typically done in the probabilistic setting $f^{(i)}(\mathbf{x}) = p(y|\mathbf{x}^{(i)})$.
- Advantage: We can use any classifier we want.

Ad-hoc Information fusion

- Train a classifier for each feature type (using kernels if wanted)
- Fuse their responses typically by summing the responses

$$f(\mathbf{x}) = \frac{1}{F}\sum_{i=1}^{F} f^{(i)}(\mathbf{x}^{(i)})$$

  with f the i-th classifier, which takes as input the i-th feature type.
- Typically done in the probabilistic setting $f^{(i)}(\mathbf{x}) = p(y|\mathbf{x}^{(i)})$.
- Advantage: We can use any classifier we want.
- Disadvantage: We do not exploit correlation between features and the outputs are typically not in the same scale.

Ad-hoc Information fusion

- Train a classifier for each feature type (using kernels if wanted)
- Fuse their responses typically by summing the responses

$$f(\mathbf{x}) = \frac{1}{F} \sum_{i=1}^{F} f^{(i)}(\mathbf{x}^{(i)})$$

  with f the i-th classifier, which takes as input the i-th feature type.
- Typically done in the probabilistic setting $f^{(i)}(\mathbf{x}) = p(y|\mathbf{x}^{(i)})$.
- Advantage: We can use any classifier we want.
- Disadvantage: We do not exploit correlation between features and the outputs are typically not in the same scale.
- Some times, people train a classifier (logistic) on the output of individual classifiers.

Ad-hoc Information fusion

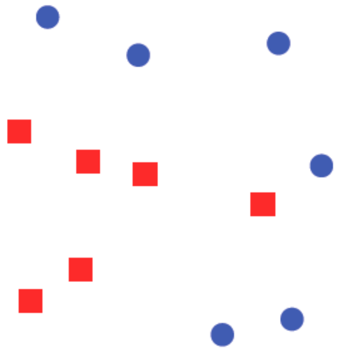
- Train a classifier for each feature type (using kernels if wanted)
- Fuse their responses typically by summing the responses

$$f(\mathbf{x}) = \frac{1}{F} \sum_{i=1}^{F} f^{(i)}(\mathbf{x}^{(i)})$$

  with f the i-th classifier, which takes as input the i-th feature type.
- Typically done in the probabilistic setting $f^{(i)}(\mathbf{x}) = p(y|\mathbf{x}^{(i)})$.
- Advantage: We can use any classifier we want.
- Disadvantage: We do not exploit correlation between features and the outputs are typically not in the same scale.
- Some times, people train a classifier (logistic) on the output of individual classifiers.

Boosting

- Inherently combines features, via combination of learners.
- Our weak-learners can be using each a subset of the features.

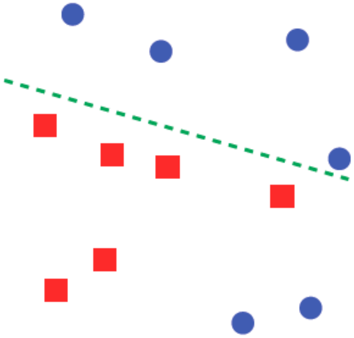


Greedy algorithm: for $m = 1, \ldots, M$

- Pick a weak classifier h_m
- Adjust weights: misclassified examples get "heavier"
- α_m set according to weighted error of h_m

Boosting

- Inherently combines features, via combination of learners.
- Our weak-learners can be using each a subset of the features.

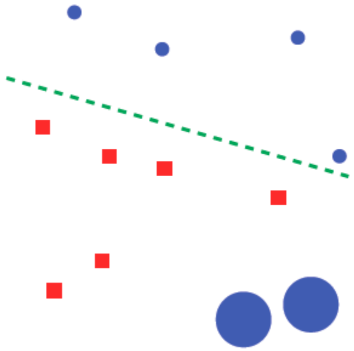


Greedy algorithm: for $m = 1, \ldots, M$

- Pick a weak classifier h_m
- Adjust weights: misclassified examples get "heavier"
- α_m set according to weighted error of h_m

Boosting

- Inherently combines features, via combination of learners.
- Our weak-learners can be using each a subset of the features.

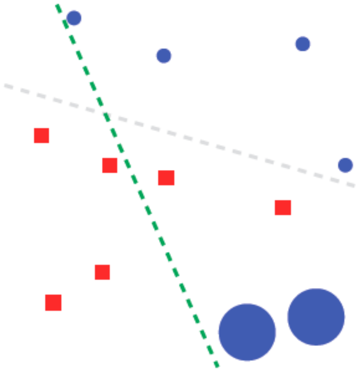


Greedy algorithm: for $m = 1, \dots, M$

- Pick a weak classifier h_m
- Adjust weights: misclassified examples get "heavier"
- α_m set according to weighted error of h_m

Boosting

- Inherently combines features, via combination of learners.
- Our weak-learners can be using each a subset of the features.

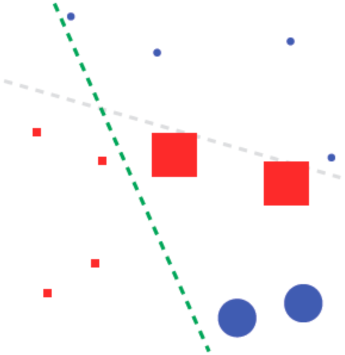


Greedy algorithm: for $m = 1, \ldots, M$

- Pick a weak classifier h_m
- Adjust weights: misclassified examples get "heavier"
- α_m set according to weighted error of h_m

Boosting

- Inherently combines features, via combination of learners.
- Our weak-learners can be using each a subset of the features.

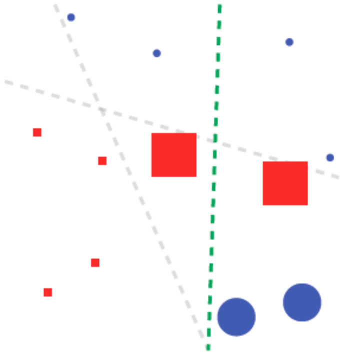


Greedy algorithm: for $m = 1, \ldots, M$

- Pick a weak classifier h_m
- Adjust weights: misclassified examples get "heavier"
- α_m set according to weighted error of h_m

Boosting

- Inherently combines features, via combination of learners.
- Our weak-learners can be using each a subset of the features.

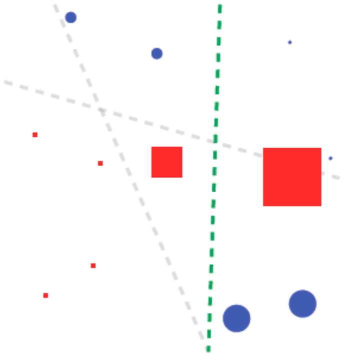


Greedy algorithm: for $m = 1, \ldots, M$

- Pick a weak classifier h_m
- Adjust weights: misclassified examples get "heavier"
- α_m set according to weighted error of h_m

Boosting

- Inherently combines features, via combination of learners.
- Our weak-learners can be using each a subset of the features.



Greedy algorithm: for $m = 1, \ldots, M$

- Pick a weak classifier h_m
- Adjust weights: misclassified examples get "heavier"
- α_m set according to weighted error of h_m

Combining Kernels

- An alternative to information fusion a posteriori is to combine information a priori.
- We can combine the kernels by summing or multiplying them to have an AND or OR effect

$$\begin{aligned} K^{OR}(\mathbf{x}_i, \mathbf{x}_j) &= \sum_{f=1}^{F} K^{(f)}(\mathbf{x}_i^{(f)}, \mathbf{x}_j^{(f)}) \\ K^{AND}(\mathbf{x}_i, \mathbf{x}_j) &= \prod_{f=1}^{F} K^{(f)}(\mathbf{x}_i^{(f)}, \mathbf{x}_j^{(f)}) \end{aligned}$$

with element-wise sum and product.

Combining Kernels

- An alternative to information fusion a posteriori is to combine information a priori.
- We can combine the kernels by summing or multiplying them to have an AND or OR effect

$$\begin{aligned} K^{OR}(\mathbf{x}_i, \mathbf{x}_j) &= \sum_{f=1}^{F} K^{(f)}(\mathbf{x}_i^{(f)}, \mathbf{x}_j^{(f)}) \\ K^{AND}(\mathbf{x}_i, \mathbf{x}_j) &= \prod_{f=1}^{F} K^{(f)}(\mathbf{x}_i^{(f)}, \mathbf{x}_j^{(f)}) \end{aligned}$$

  with element-wise sum and product.
- Sums and products of mercer kernels are still mercer.

Combining Kernels

- An alternative to information fusion a posteriori is to combine information a priori.
- We can combine the kernels by summing or multiplying them to have an AND or OR effect

$$\begin{aligned} K^{OR}(\mathbf{x}_i, \mathbf{x}_j) &= \sum_{f=1}^{F} K^{(f)}(\mathbf{x}_i^{(f)}, \mathbf{x}_j^{(f)}) \\ K^{AND}(\mathbf{x}_i, \mathbf{x}_j) &= \prod_{f=1}^{F} K^{(f)}(\mathbf{x}_i^{(f)}, \mathbf{x}_j^{(f)}) \end{aligned}$$

  with element-wise sum and product.
- Sums and products of mercer kernels are still mercer.
- It will be great if we could learn the importance of each kernel in the OR setting.

Combining Kernels

- An alternative to information fusion a posteriori is to combine information a priori.
- We can combine the kernels by summing or multiplying them to have an AND or OR effect

$$\begin{aligned} K^{OR}(\mathbf{x}_i, \mathbf{x}_j) &= \sum_{f=1}^{F} K^{(f)}(\mathbf{x}_i^{(f)}, \mathbf{x}_j^{(f)}) \\ K^{AND}(\mathbf{x}_i, \mathbf{x}_j) &= \prod_{f=1}^{F} K^{(f)}(\mathbf{x}_i^{(f)}, \mathbf{x}_j^{(f)}) \end{aligned}$$

  with element-wise sum and product.
- Sums and products of mercer kernels are still mercer.
- It will be great if we could learn the importance of each kernel in the OR setting.

Multiple Kernel Learning

- Introduce to the vision community by [Varma & Ray, 07]
- Recall the SVM formulation the primal is

$$\min_{\mathbf{w}} \ \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N}\xi_i .$$

$$\text{subject to } \ y_i(\mathbf{w}^T\phi(\mathbf{x}_i)+b) - 1 + \xi_i \geq 0, \quad i = 1,\ldots,N.$$

and the dual

$$\max \left\{ \sum_{i=1}^{N}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{N}\alpha_i\alpha_j y_i y_j K(\mathbf{x}_i,\mathbf{x}_j) \right\}$$

$$\text{subject to} \sum_{i=1}^{N}\alpha_i y_i = 0, \ 0 \leq \alpha_i \leq C \text{ for all } i = 1,\ldots,N.$$

Multiple Kernel Learning

- Varma & Ray introduced the following primal formulation

$$\min_{\mathbf{w},\mathbf{d},\xi} \quad \frac{1}{2}\|\mathbf{w}\|^2 + \; C\sum_{i=1}^{N}\xi_i + \sigma^t\mathbf{d}$$

$$\text{subject to} \quad y_i(\mathbf{w}^T\phi(\mathbf{x}_i) + b) - 1 + \xi_i \;\geq\; 0,$$

$$\xi \geq 0, \mathbf{d} \geq 0, \mathbf{A}\mathbf{d} \geq \mathbf{p}$$

$$\text{where} \quad \phi^t(\mathbf{x}_i)\phi(\mathbf{x}_j) = \sum_k d_k\phi_k^t(\mathbf{x}_i)\phi_k(\mathbf{x}_j)$$

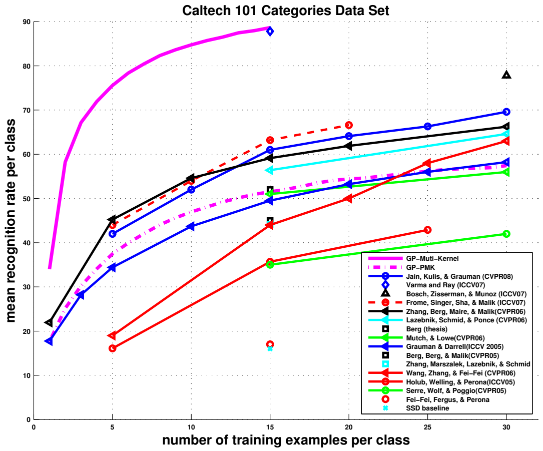
- New: ℓ_1 regularization on the weights **d** to discover a minimal set
- Most of the weights will be 0 depending on σ which encode prior preferences for descriptors
- Two additional constraints have been incorporated
  - $\mathbf{d} \geq 0$ ensures interpretable weights
  - $\mathbf{A}\mathbf{d} \geq \mathbf{p}$ encodes prior knowledge about the problem
  - Last equation encodes $\mathbf{K}_{opt} = \sum_k d_k\mathbf{K}_k$
- Minimization is carried out in the dual

Regularization for multiple kernels

- Summing kernels is equivalent to concatenating feature spaces
  - m feature maps
  - Minimization with respect to weights
  - Results in a predictor $f(x) = d_1\phi_1(\mathbf{x}) + \cdots + d_m\phi(\mathbf{x})$
- Regularization by $\sum_j ||d_j||_2$ is equivalent to $K = \sum_j K_j$
- Regularization $\sum_j ||d_j||$ imposes sparsity
- We can regularize by blocks: structured sparsity

Is computer vision solved?

- We thought so for a few days as it performs great on Caltech 101



Unfortunately, there was a bug in the kernels ...

Other SVM-MKL formulations

- More standard formulation [Bach 04]

$$\min_{\mathbf{w},b,\xi} \; \frac{1}{2}\left(\sum_k ||w_k||_2\right) + \; C\sum_{i=1}^{N}\xi_i$$

$$\text{subject to} \;\; \xi \geq 0 \text{ and } y_i\left(\sum_k \mathbf{w}_k^T\phi_k(\mathbf{x}_i) + b\right) - 1 + \xi_i \;\geq\; 0$$

- The solution can be written as $\mathbf{w}_k = \beta_k \mathbf{w}'_k$ with $\beta_k \geq 0$ and $\sum_i \beta_k = 1$
- The dual

$$\min_{\gamma,\alpha} \gamma - \sum_{i=1}^{N}\alpha_i$$

$$\text{subject to} \sum_{i=1}^{N}\alpha_i y_i = 0,\; 0 \leq \alpha_i \leq \mathbf{1}C \text{ for all } i = 1,\ldots,N.$$

$$\frac{1}{2}\sum_{i,j=1}^{N}\alpha_i\alpha_j y_i y_j K_k(\mathbf{x}_i,\mathbf{x}_j) \leq \gamma \;\; \forall k = 1,\cdots,K$$

Gaussian process as an alternative to SVMs

Gaussian processes (GPs)

Definition

A Gaussian process is a collection of random variables, any finite number of which have a joint Gaussian distribution.

- Probability Distribution over Functions
- Functions are infinite dimensional.
  - Prior distribution over *instantiations* of the function: finite dimensional objects.
- GPs are consistent.

Gaussian processes

- A (zero mean) Gaussian process likelihood is of the form

$$p\left(\mathbf{y}|\mathbf{X}\right) = N\left(\mathbf{y}|\mathbf{0},\mathbf{K}\right),$$

where $\mathbf{K}$ is the covariance function or *kernel*.

- Covariance samples

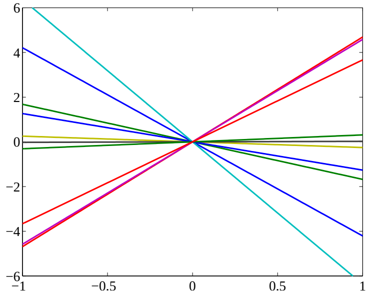


Figure: linear kernel, $\mathbf{K} = \mathbf{X}\mathbf{X}^{\mathsf{T}}$

Gaussian processes

- A (zero mean) Gaussian process likelihood is of the form

$$p(\mathbf{y}|\mathbf{X}) = N(\mathbf{y}|\mathbf{0}, \mathbf{K}),$$

  where $\mathbf{K}$ is the covariance function or *kernel*.
- Covariance samples

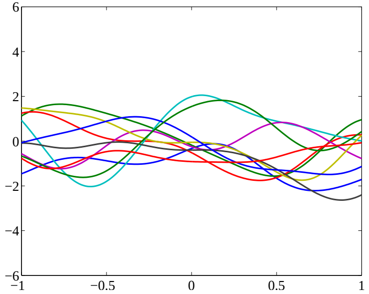


Figure: RBF kernel, $k_{i,j} = \alpha \exp\left(-\frac{1}{2l}\left\|\mathbf{x}_i - \mathbf{x}_j\right\|^2\right)$, with $l = 0.32$, $\alpha = 1$

Gaussian processes

- A (zero mean) Gaussian process likelihood is of the form

$$p\left(\mathbf{y}|\mathbf{X}\right) = N\left(\mathbf{y}|\mathbf{0}, \mathbf{K}\right),$$

  where $\mathbf{K}$ is the covariance function or *kernel*.
- Covariance samples

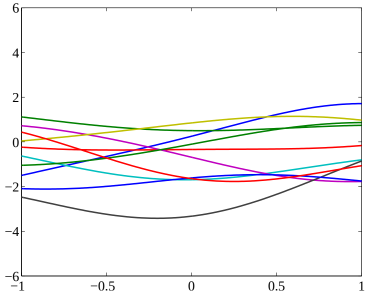


Figure: RBF kernel, $k_{i,j} = \alpha \exp\left(-\frac{1}{2l}\left\|\mathbf{x}_i - \mathbf{x}_j\right\|^2\right)$, with $l = 1$, $\alpha = 1$

Gaussian processes

- A (zero mean) Gaussian process likelihood is of the form

$$p(\mathbf{y}|\mathbf{X}) = N(\mathbf{y}|\mathbf{0}, \mathbf{K}),$$

  where $\mathbf{K}$ is the covariance function or *kernel*.
- Covariance samples

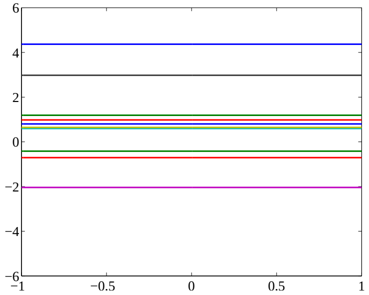


Figure: bias 'kernel', $k_{i,j} = \alpha$, with $\alpha = 1$ and

Gaussian processes

- A (zero mean) Gaussian process likelihood is of the form

$$p(\mathbf{y}|\mathbf{X}) = N(\mathbf{y}|\mathbf{0}, \mathbf{K}),$$

  where **K** is the covariance function or *kernel*.
- Covariance samples

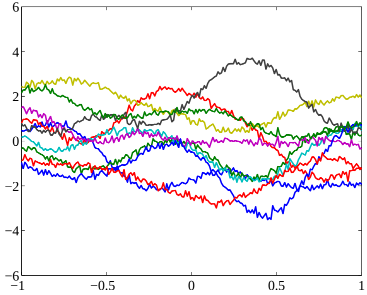
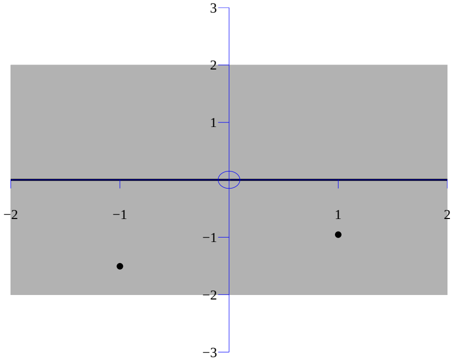


Figure: summed combination of: RBF kernel, $\alpha = 1$, $l = 0.3$; bias kernel, $\alpha = 1$; and white noise kernel, $\beta = 100$

Gaussian process regression

Posterior Distribution over Functions

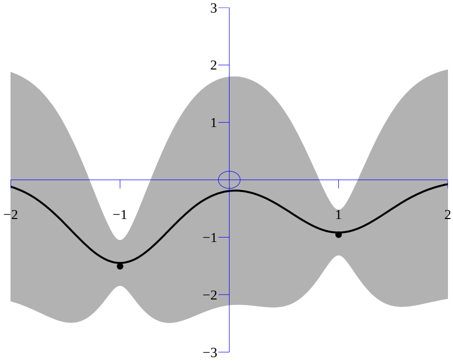
- Gaussian processes are often used for regression.
- We are given a known inputs **X** and targets **Y**.
- We assume a prior distribution over functions by selecting a kernel.
- Combine the prior with data to get a *posterior* distribution over functions.



Gaussian process regression

Posterior Distribution over Functions

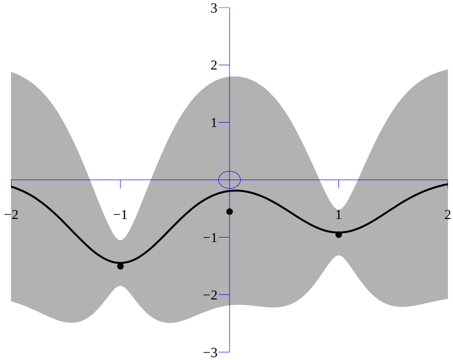
- Gaussian processes are often used for regression.
- We are given a known inputs **X** and targets **Y**.
- We assume a prior distribution over functions by selecting a kernel.
- Combine the prior with data to get a *posterior* distribution over functions.



Gaussian process regression

Posterior Distribution over Functions

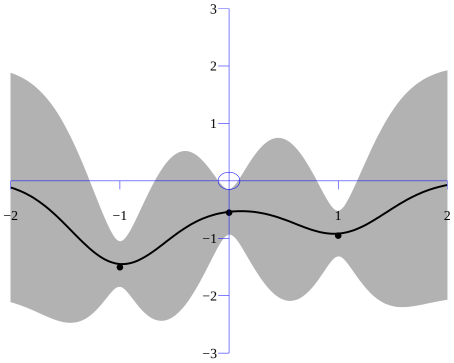
- Gaussian processes are often used for regression.
- We are given a known inputs **X** and targets **Y**.
- We assume a prior distribution over functions by selecting a kernel.
- Combine the prior with data to get a *posterior* distribution over functions.



Gaussian process regression

Posterior Distribution over Functions

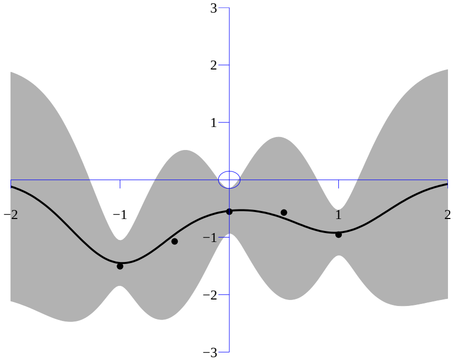
- Gaussian processes are often used for regression.
- We are given a known inputs **X** and targets **Y**.
- We assume a prior distribution over functions by selecting a kernel.
- Combine the prior with data to get a *posterior* distribution over functions.



Gaussian process regression

Posterior Distribution over Functions

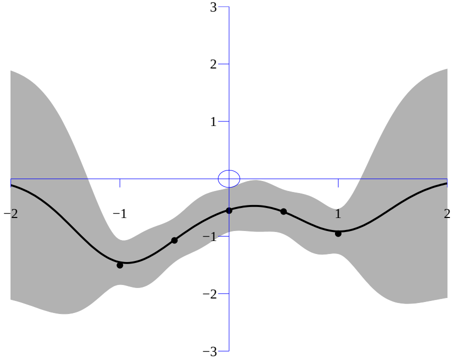
- Gaussian processes are often used for regression.
- We are given a known inputs **X** and targets **Y**.
- We assume a prior distribution over functions by selecting a kernel.
- Combine the prior with data to get a *posterior* distribution over functions.



Gaussian process regression

Posterior Distribution over Functions

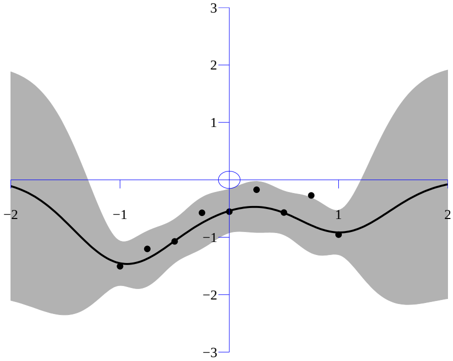
- Gaussian processes are often used for regression.
- We are given a known inputs **X** and targets **Y**.
- We assume a prior distribution over functions by selecting a kernel.
- Combine the prior with data to get a *posterior* distribution over functions.



Gaussian process regression

Posterior Distribution over Functions

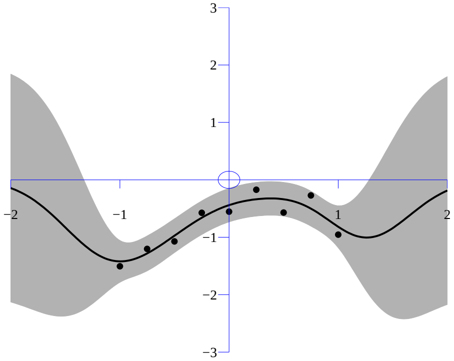
- Gaussian processes are often used for regression.
- We are given a known inputs **X** and targets **Y**.
- We assume a prior distribution over functions by selecting a kernel.
- Combine the prior with data to get a *posterior* distribution over functions.



Gaussian process regression

Posterior Distribution over Functions

- Gaussian processes are often used for regression.
- We are given a known inputs **X** and targets **Y**.
- We assume a prior distribution over functions by selecting a kernel.
- Combine the prior with data to get a *posterior* distribution over functions.



MKL is much simpler with Gaussian Processes

- Let $\mathbf{X}$ be the matrix of all training inputs and let $\mathbf{Y}$ be the associated labels.
- We assume a GP prior

$$p(\mathbf{Y}|\mathbf{X}) \sim \mathcal{N}(0, \mathbf{K}).$$

MKL is much simpler with Gaussian Processes

- Let $\mathbf{X}$ be the matrix of all training inputs and let $\mathbf{Y}$ be the associated labels.
- We assume a GP prior
$$p(\mathbf{Y}|\mathbf{X}) \sim \mathcal{N}(0, \mathbf{K}).$$
- Assuming Gaussian noise, the posterior can be computed as
$$\log p(\mathbf{t}_L|\mathbf{X}, \Theta) = -\frac{1}{2}\mathbf{t}_L^T(\sigma^2\mathbf{I} + \mathbf{K})^{-1}\mathbf{t}_L - \frac{1}{2}\log|\sigma^2\mathbf{I} + \mathbf{K}| - Const.$$
with $\mathbf{K} = \sum_{i=1}^{k} \alpha_i \mathbf{K}^{(i)}$,

MKL is much simpler with Gaussian Processes

- Let $\mathbf{X}$ be the matrix of all training inputs and let $\mathbf{Y}$ be the associated labels.
- We assume a GP prior

$$p(\mathbf{Y}|\mathbf{X}) \sim \mathcal{N}(0, \mathbf{K}).$$

- Assuming Gaussian noise, the posterior can be computed as

$$\log p(\mathbf{t}_L|\mathbf{X}, \Theta) = -\frac{1}{2}\mathbf{t}_L^T(\sigma^2\mathbf{I} + \mathbf{K})^{-1}\mathbf{t}_L - \frac{1}{2}\log|\sigma^2\mathbf{I} + \mathbf{K}| - Const.$$

with $\mathbf{K} = \sum_{i=1}^{k} \alpha_i \mathbf{K}^{(i)}$,

- Learning can then be formulated as

$$\arg\min_{\boldsymbol{\alpha}} -\log p(\mathbf{t}_L|\mathbf{X}, \boldsymbol{\alpha}) + \gamma_1||\alpha||_1 + \gamma_2||\alpha||_2$$

$$\text{subject to:} \quad \alpha_i \geq 0 \text{ for } i \in \{0, .., k\}.$$

MKL is much simpler with Gaussian Processes

- Let $\mathbf{X}$ be the matrix of all training inputs and let $\mathbf{Y}$ be the associated labels.
- We assume a GP prior
$$p(\mathbf{Y}|\mathbf{X}) \sim \mathcal{N}(0, \mathbf{K}).$$
- Assuming Gaussian noise, the posterior can be computed as
$$\log p(\mathbf{t}_L|\mathbf{X}, \Theta) = -\frac{1}{2}\mathbf{t}_L^T(\sigma^2\mathbf{I} + \mathbf{K})^{-1}\mathbf{t}_L - \frac{1}{2}\log|\sigma^2\mathbf{I} + \mathbf{K}| - Const.$$
with $\mathbf{K} = \sum_{i=1}^k \alpha_i \mathbf{K}^{(i)}$,
- Learning can then be formulated as
$$\arg\min_{\boldsymbol{\alpha}} - \log p(\mathbf{t}_L|\mathbf{X}, \boldsymbol{\alpha}) + \gamma_1||\alpha||_1 + \gamma_2||\alpha||_2$$
$$\text{subject to:} \qquad \alpha_i \geq 0 \text{ for } i \in \{0, .., k\}.$$
- Prediction using $\mathbf{y} = \mathbf{k}(\mathbf{x}_*)^T(\sigma^2\mathbf{I} + \mathbf{K})^{-1}\mathbf{t}$

MKL is much simpler with Gaussian Processes

- Let $\mathbf{X}$ be the matrix of all training inputs and let $\mathbf{Y}$ be the associated labels.
- We assume a GP prior

$$p(\mathbf{Y}|\mathbf{X}) \sim \mathcal{N}(0, \mathbf{K}).$$

- Assuming Gaussian noise, the posterior can be computed as

$$\log p(\mathbf{t}_L|\mathbf{X}, \Theta) = -\frac{1}{2}\mathbf{t}_L^T(\sigma^2\mathbf{I} + \mathbf{K})^{-1}\mathbf{t}_L - \frac{1}{2}\log|\sigma^2\mathbf{I} + \mathbf{K}| - Const.$$

  with $\mathbf{K} = \sum_{i=1}^k \alpha_i \mathbf{K}^{(i)}$,

- Learning can then be formulated as

$$\arg\min_{\boldsymbol{\alpha}} - \log p(\mathbf{t}_L|\mathbf{X}, \boldsymbol{\alpha}) + \gamma_1||\alpha||_1 + \gamma_2||\alpha||_2$$

$$\text{subject to:} \qquad \alpha_i \geq 0 \text{ for } i \in \{0, .., k\}.$$

- Prediction using $\mathbf{y} = \mathbf{k}(\mathbf{x}_*)^T(\sigma^2\mathbf{I} + \mathbf{K})^{-1}\mathbf{t}$

Results: Caltech 101

Comparison with SVM kernel combination [Kapoor et al. 09]

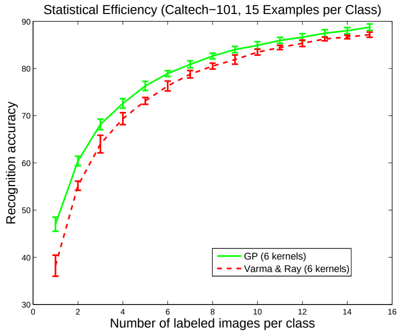


Figure: Average precision.

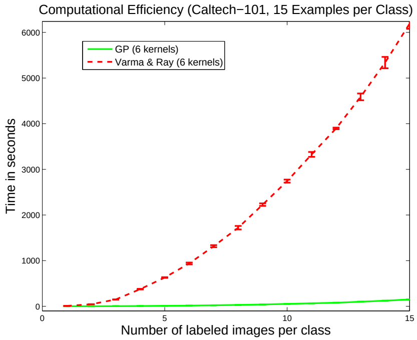


Figure: Time of computation.

Results: Caltech 101 for real ;)

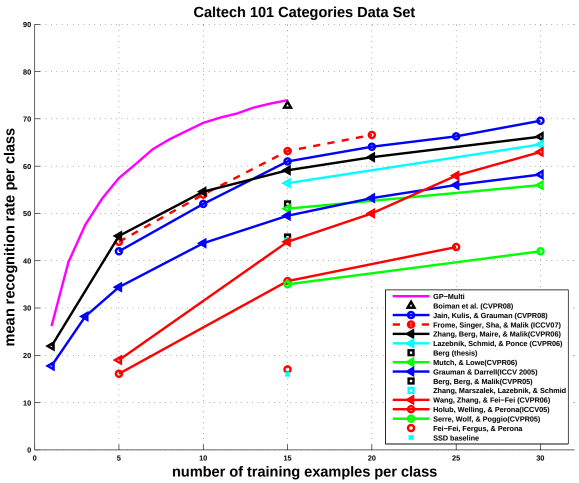


Figure: Comparison with the state of the art [Kapoor et al. 09].

Is learning the weights important?

- Unfortunately not really...
- In general very similar performance if you learn or not the weights.
- If you don't learn the weights, for GP you don't have to do training, just invert a matrix!
- Life is simple ;)

NN approaches

NN approaches perform worst than more complex classifiers but [Boiman et al. 08] argue that this is due to

- Quantization of local image descriptors (used to generate bags-of-words, codebooks).
- Computation of Image-to-Image distance, instead of Image-to-Class distance.

NN approaches

NN approaches perform worst than more complex classifiers but [Boiman et al. 08] argue that this is due to

- Quantization of local image descriptors (used to generate bags-of-words, codebooks).
- Computation of Image-to-Image distance, instead of Image-to-Class distance.
- They proposed an effective NN-based classifier NBNN, (Naive-Bayes Nearest-Neighbor), which employs NN distances in the space of the local image descriptors (not images).

NN approaches

NN approaches perform worst than more complex classifiers but [Boiman et al. 08] argue that this is due to

- Quantization of local image descriptors (used to generate bags-of-words, codebooks).
- Computation of Image-to-Image distance, instead of Image-to-Class distance.
- They proposed an effective NN-based classifier NBNN, (Naive-Bayes Nearest-Neighbor), which employs NN distances in the space of the local image descriptors (not images).
- NBNN computes direct Image to- Class distances without descriptor quantization.

NN approaches

NN approaches perform worst than more complex classifiers but [Boiman et al. 08] argue that this is due to

- Quantization of local image descriptors (used to generate bags-of-words, codebooks).
- Computation of Image-to-Image distance, instead of Image-to-Class distance.
- They proposed an effective NN-based classifier NBNN, (Naive-Bayes Nearest-Neighbor), which employs NN distances in the space of the local image descriptors (not images).
- NBNN computes direct Image to- Class distances without descriptor quantization.
- No learning/training phase.

NN approaches

NN approaches perform worst than more complex classifiers but [Boiman et al. 08] argue that this is due to

- Quantization of local image descriptors (used to generate bags-of-words, codebooks).
- Computation of Image-to-Image distance, instead of Image-to-Class distance.
- They proposed an effective NN-based classifier NBNN, (Naive-Bayes Nearest-Neighbor), which employs NN distances in the space of the local image descriptors (not images).
- NBNN computes direct Image to- Class distances without descriptor quantization.
- No learning/training phase.
- Similarities with ISM but now for classification.

NN approaches

NN approaches perform worst than more complex classifiers but [Boiman et al. 08] argue that this is due to

- Quantization of local image descriptors (used to generate bags-of-words, codebooks).
- Computation of Image-to-Image distance, instead of Image-to-Class distance.
- They proposed an effective NN-based classifier NBNN, (Naive-Bayes Nearest-Neighbor), which employs NN distances in the space of the local image descriptors (not images).
- NBNN computes direct Image to- Class distances without descriptor quantization.
- No learning/training phase.
- Similarities with ISM but now for classification.

Algortithm of NBNN

- Given a query image, compute all its local image descriptors $d_1, \cdots, d_n$.
- Search for the class C which minimizes

$$\sum_{i=1}^{n} ||d_i - NN_C(d_i)||^2$$

with $NN_C(d_i)$ the NN descriptor of d_i in class C.

- Requires fast NN search.

Why quantization is bad

- When densely sampled image descriptors are divided into fine bins, the bin-density follows a power-law.
- There are almost no clusters in the descriptor space.
- Therefore, any clustering to a small number of clusters (even thousands) will inevitably incur a very high quantization error.
- Informative descriptors have low database frequency, leading to high quantization error.

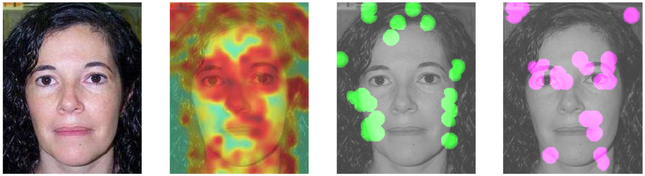
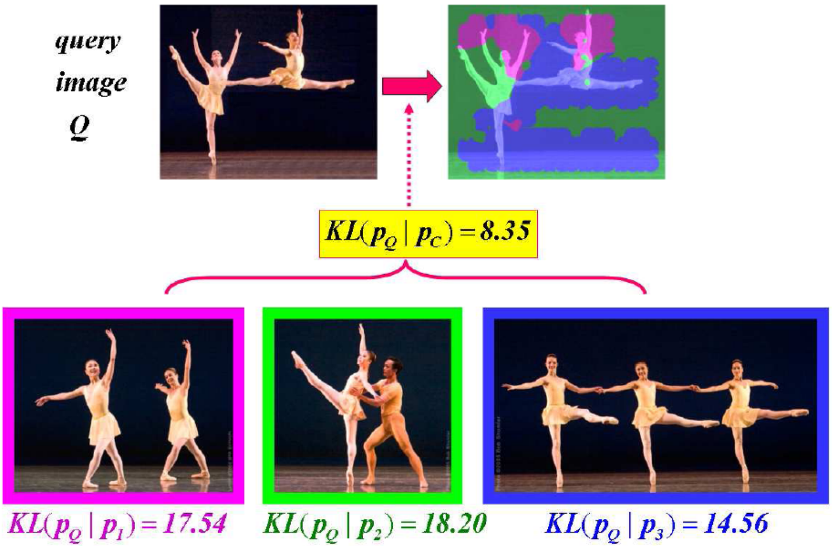
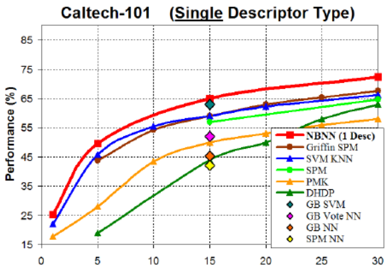


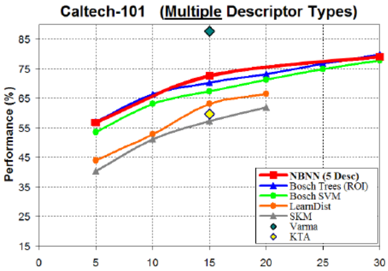
Image-to-Image vs. Image-to-Class distance



query image Q
$KL(p_Q \mid p_C) = 8.35$
$KL(p_Q \mid p_1) = 17.54$
$KL(p_Q \mid p_2) = 18.20$
$KL(p_Q \mid p_3) = 14.56$

Results Caltech 101

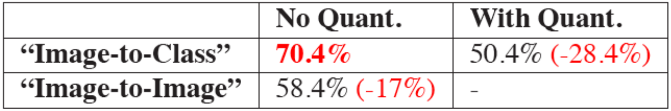




Multiple descriptors by summing weighted distances.

Effects of Quantization

Impact of introducing descriptor quantization or Imageto- Image distance into NBNN (using SIFT descriptor on Caltech- 101, nlabel = 30).

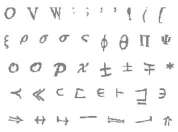


	No Quant.	**With Quant.**
“Image-to-Class”	**70.4%**	50.4% (-28.4%)
“Image-to-Image”	58.4% (-17%)	-

Randomized Decision Forests

- Very fast tools for classification, clustering and regression
- Good generalization through randomized training
- Inherently multi-class: automatic feature sharing
- Simple training / testing algorithms

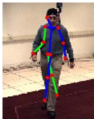
Randomized Forests in Vision

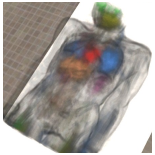

[Amit & Geman, 97]
digit recognition

[Lepetit *et al.*, 06]
keypoint recognition

[Moosmann *et al.*, 06]
visual word clustering

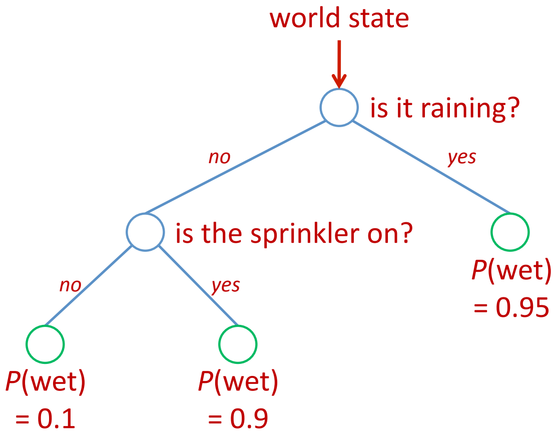
[Shotton *et al.*, 08]
object segmentation


[Rogez *et al.*, 08]
pose estimation


[Criminisi *et al.*, 09]
organ detection

[Source: Shotton et al.]

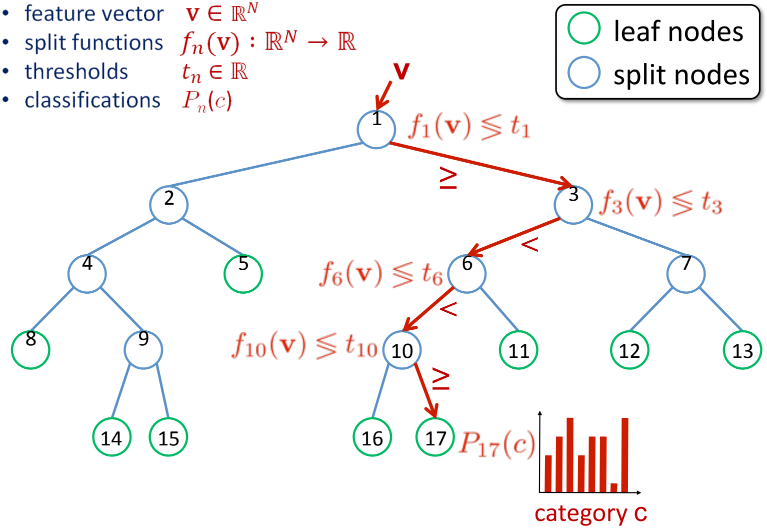
Is the grass wet?



[Source: Shotton et al.]

Binary Decision Trees

- feature vector $\mathbf{v} \in \mathbb{R}^N$
- split functions $f_n(\mathbf{v}) : \mathbb{R}^N \to \mathbb{R}$
- thresholds $t_n \in \mathbb{R}$
- classifications $P_n(c)$



◯ leaf nodes
◯ split nodes
v
1
$f_1(\mathbf{v}) \lessgtr t_1$
≥
2
3
$f_3(\mathbf{v}) \lessgtr t_3$
<
4
5
$f_6(\mathbf{v}) \lessgtr t_6$
6
7
<
8
9
$f_{10}(\mathbf{v}) \lessgtr t_{10}$
10
11
12
13
≥
14
15
16
17
$P_{17}(c)$
category c

[Source: Shotton et al.]

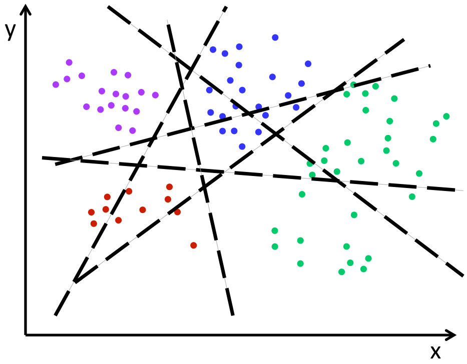
Decision Tree Pseudo-Code

```
double[] ClassifyDT(node, v)
    if node.IsSplitNode then
        if node.f(v) >= node.t then
            return ClassifyDT(node.right, v)
        else
            return ClassifyDT(node.left, v)
        end
    else
        return node.P
    end
end
```

[Source: Shotton et al.]

Toy Example

- **Try several lines, chosen at random**
- **Keep line that best separates data**
  - information gain
- **Recurse**

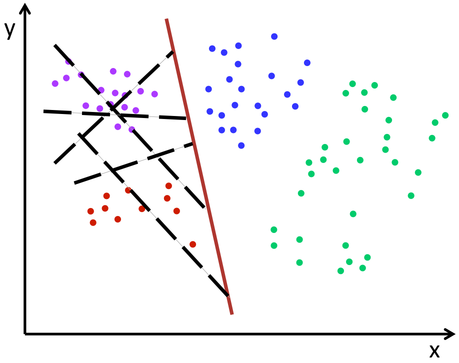


- feature vectors are x, y coordinates: $\mathbf{v} = [x, y]^T$
- split functions are lines with parameters a, b: $f_n(\mathbf{v}) = ax + by$
- threshold determines intercepts: t_n
- four classes: purple, blue, red, green

[Source: Shotton et al.]

Toy Example

- **Try several lines, chosen at random**
- **Keep line that best separates data**
  - information gain
- **Recurse**

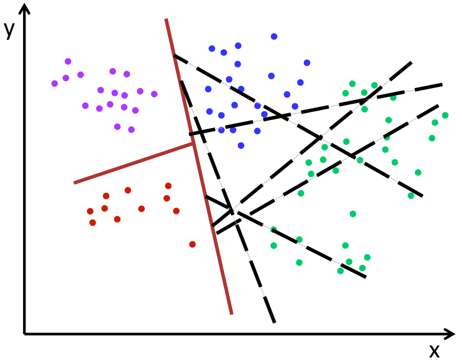


- feature vectors are x, y coordinates: $\mathbf{v} = [x, y]^T$
- split functions are lines with parameters a, b: $f_n(\mathbf{v}) = ax + by$
- threshold determines intercepts: t_n
- four classes: purple, blue, red, green

[Source: Shotton et al.]

Toy Example

- **Try several lines, chosen at random**
- **Keep line that best separates data**
  - information gain
- **Recurse**

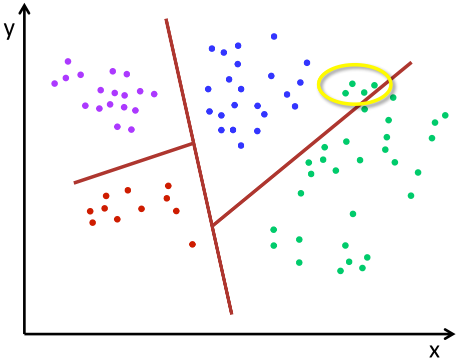


- feature vectors are x, y coordinates: $\mathbf{v} = [x, y]^T$
- split functions are lines with parameters a, b: $f_n(\mathbf{v}) = ax + by$
- threshold determines intercepts: t_n
- four classes: purple, blue, red, green

[Source: Shotton et al.]

Toy Example

- **Try several lines, chosen at random**
- **Keep line that best separates data**
  - information gain
- **Recurse**



- feature vectors are x, y coordinates: $\mathbf{v} = [x, y]^T$
- split functions are lines with parameters a, b: $f_n(\mathbf{v}) = ax + by$
- threshold determines intercepts: t_n
- four classes: purple, blue, red, green

[Source: Shotton et al.]

Randomized Learning

- Recursively split examples at node n: set I_n indexes labeled training examples $(\mathbf{v}_i, l_i)$

$$\begin{aligned} \text{left split} \rightarrow I_l &= \{i \in I_n \mid \underbrace{f(\mathbf{v}_i)}_{\text{function of example i's feature vector}} < \underbrace{t}_{\text{threshold}}\} \\ \text{right split} \rightarrow I_r &= I_n \setminus I_l \end{aligned}$$

- At node n, $P_n(c)$ is histogram of example labels l_i.

[Source: Shotton et al.]

Randomized Learning

$$\begin{array}{rlcl} \text{left split} & I_\text{l} & = & \{i \in I_n \mid f(\mathbf{v}_i) < t\} \\ \text{right split} & I_\text{r} & = & I_n \setminus I_\text{l} \end{array}$$

- **Features** f(**v**) **chosen at random from feature pool** f **2** F
- **Thresholds** t **chosen in range** $t \in (\min_i f(\mathbf{v}_i), \max_i f(\mathbf{v}_i))$
- **Choose** f **and** t **to maximize gain in information**

$$\Delta E = -\frac{|I_\text{l}|}{|I_n|}E(I_\text{l}) - \frac{|I_\text{r}|}{|I_n|}E(I_\text{r})$$

Entropy E calculated from histogram of labels in I

[Source: Shotton et al.]

Details

How many features and thresholds to try?

- just one $=$ extremely randomized
- few $\rightarrow$ fast training, may under-fit, maybe too deep
- many $\rightarrow$ slower training, may over-fit

When to stop growing the tree?

- maximum depth
- minimum entropy gain
- delta class distribution
- pruning

[Source: Shotton et al.]

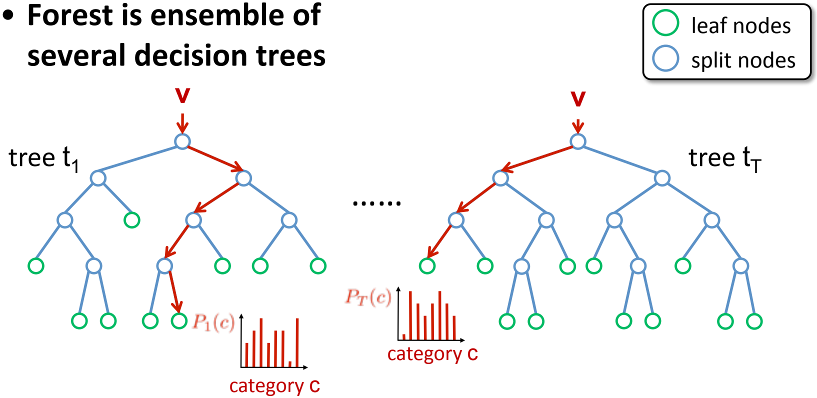
Randomized Learning Pseudo Code

```
TreeNode LearnDT(I)

   repeat featureTests times
      let f = RndFeature()
      let r = EvaluateFeatureResponses(I, f)

      repeat threshTests times
         let t = RndThreshold(r)
         let (I_l, I_r) = Split(I, r, t)
         let gain = InfoGain(I_l, I_r)
         if gain is best then remember f, t, I_l, I_r
      end
   end

   if best gain is sufficient
      return SplitNode(f, t, LearnDT(I_l), LearnDT(I_r))
   else
      return LeafNode(HistogramExamples(I))
   end
end
```

[Source: Shotton et al.]

A forests of trees

- **Forest is ensemble of several decision trees**



– classification is $P(c|\mathbf{v}) = \frac{1}{T}\sum_{t=1}^{T} P_t(c|\mathbf{v})$

[Amit & Geman 97]
[Breiman 01]
[Lepetit *et al.* 06]

[Source: Shotton et al.]

Forest Pseudo

```
double[] ClassifyDF(forest, v)
    // allocate memory
    let P = double[forest.CountClasses]

    // loop over trees in forest
    for t = 1 to forest.CountTrees
        let P’ = ClassifyDT(forest.Tree[t], v)
        P = P + P’ // sum distributions
    end

    // normalise
    P = P / forest.CountTrees
end
```

[Source: Shotton et al.]

Learning

- **Divide training examples into T subsets I_t μ I**
  - **improves generalization**
  - reduces **memory requirements** & **training time**
- **Train each decision tree t on subset I_t**
  - same decision tree learning as before
- **Multi-core friendly**

> - Subsets can be chosen at random or hand-picked
> - Subsets can have overlap (and usually do)
> - Can enforce subsets of *images* (not just examples)
> - Could also divide the feature pool into subsets

[Source: Shotton et al.]

Learning

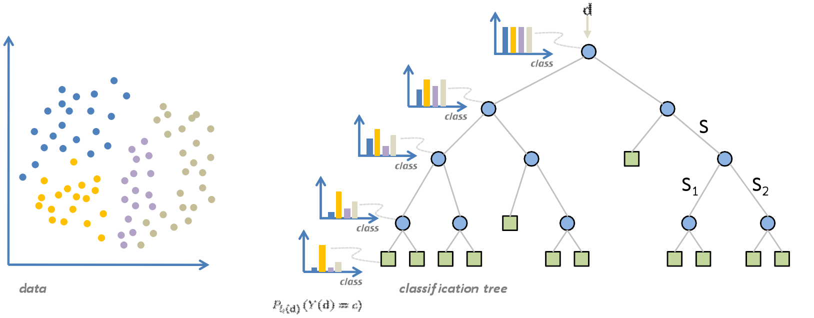
```
Forest LearnDF(countTrees, I)
    // allocate memory
    let forest = Forest(countTrees)

    // loop over trees in forest
    for t = 1 to countTrees
        let I_t = RandomSplit(I)
        forest[t] = LearnDT(I_t)
    end

    // return forest object
    return forest
end
```

[Source: Shotton et al.]

Classification

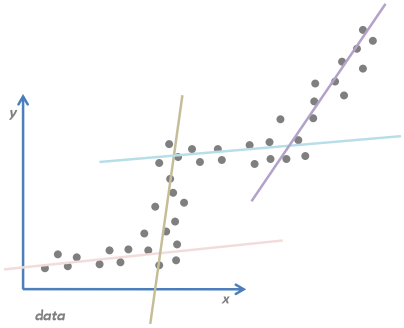
- **Trees can be trained for**
  - classification, regression, or clustering

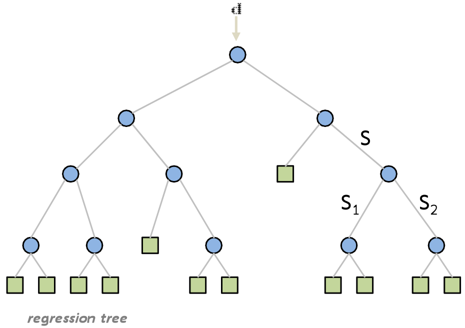
- **Change the object function**
  - information gain for classification: $I = H(S) - \sum_{i=1}^{2} \frac{|S_i|}{|S|} H(S_i)$ measure of distribution purity



[Source: Shotton et al.]

Regression





- Real-valued output *y*
- Object function: maximize $Err(S) - \sum_{i=1}^{2} \frac{|S_i|}{|S|} Err(S_i)$

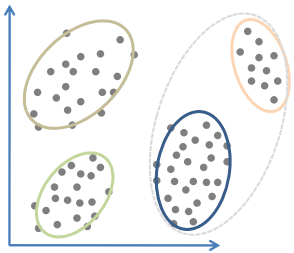
measure of fit of model

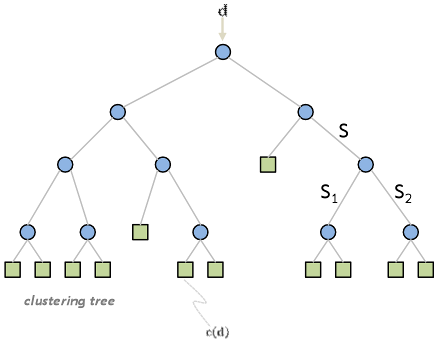
$$Err(S) = \sum_{j \in S} \left(y_j - y(x_j)\right)^2$$

e.g. linear model y = ax+b,
Or just constant model

[Source: Shotton et al.]

Clustering





- Output is cluster membership
- Option 1 – minimize imbalance: $B = |\log|S_1| - \log|S_2||$ [Moosmann *et al.* 06]
- Option 2 – maximize Gaussian likelihood:

$$T = |\Lambda_S| - \sum_{i=1}^{2} \frac{|S_i|}{|S|} |\Lambda_{S_i}|$$

measure of cluster tightness (maximizing a function of info gain for Gaussian distributions)

[Source: Shotton et al.]

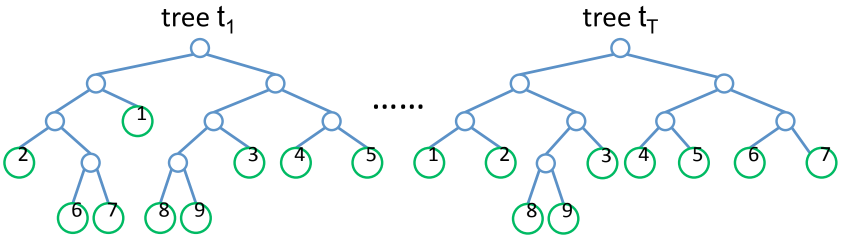
Clustering example [Moosmann et al. 06]

- **Visual words good for e.g. matching, recognition but *k*-means clustering very slow**

[Sivic *et al.* 03]
[Csurka *et al.* 04]

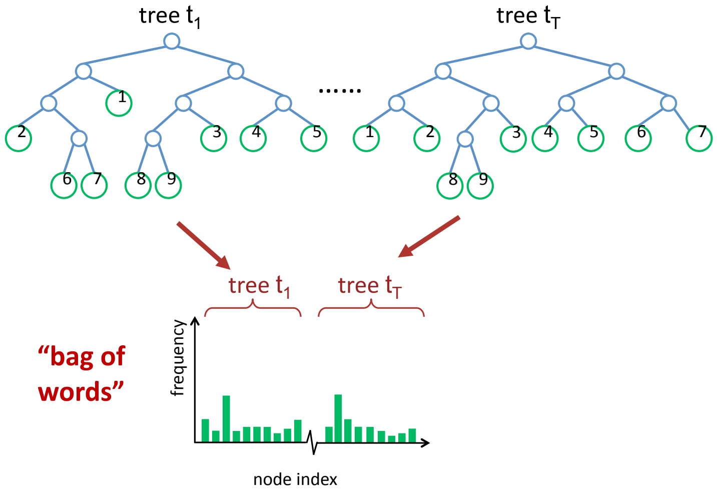
- **Randomized forests for clustering descriptors**
  - e.g. SIFT, texton filter-banks, etc.

- **Leaf nodes in forest are clusters**
  - concatenate histograms from trees in forest



[Source: Shotton et al.]

Clustering example [Moosmann et al. 06]

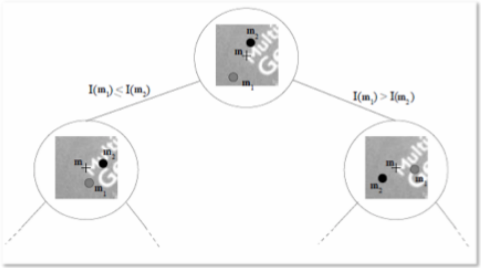


[Source: Shotton et al.]

Applications: keypoint detection [LePetit 06]

- **Wide-baseline matching as classification problem**

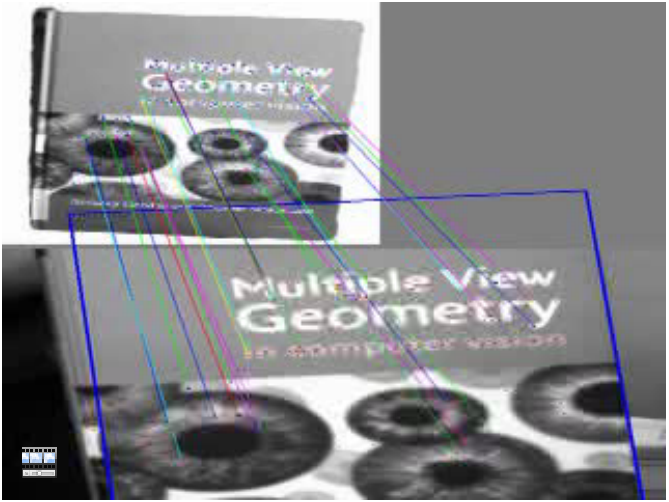
- **Extract prominent key-points in training images**



- **Forest classifies**
  - patches -> keypoints
- **Features**
  - pixel comparisons
- **Augmented training set**
  - gives robustness to patch scaling, translation, rotation

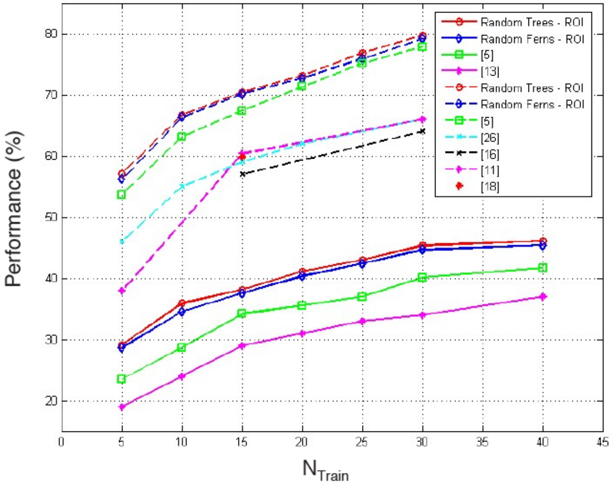
[Source: Shotton et al.]

Fast Keypoint Recognition

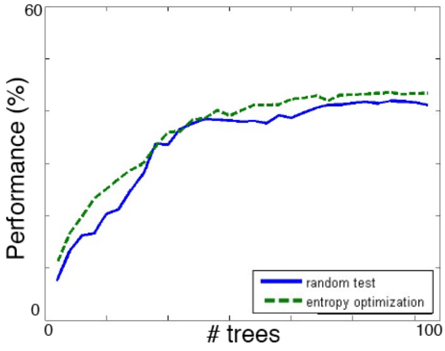


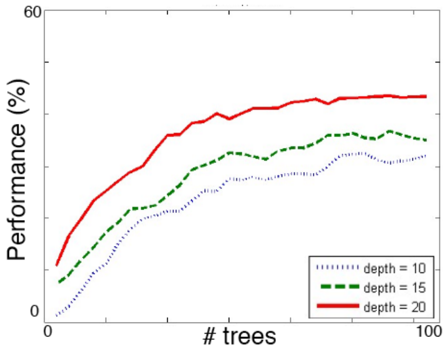
[Source: Shotton et al.]

Classification

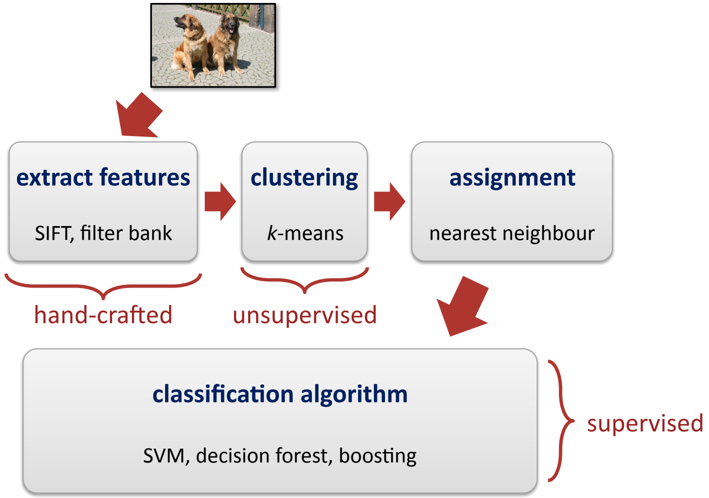


Classification



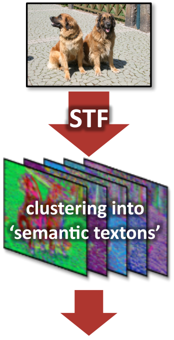


Object Recognition Pipeline



[Source: Shotton et al.]

Object Recognition Pipeline



Semantic Texton Forest (STF)

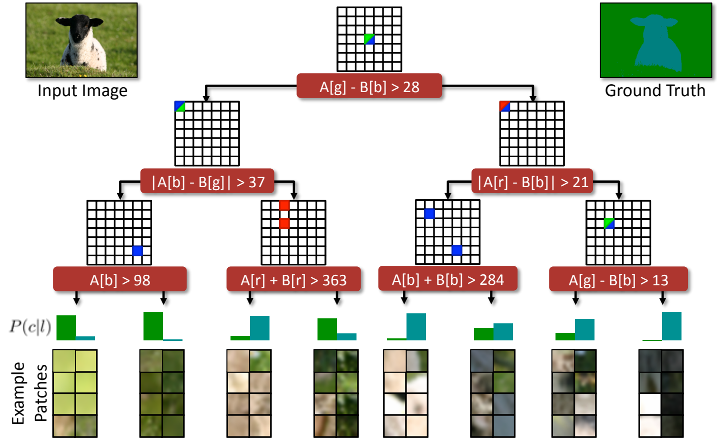
- decision forest for **clustering** & **classification**
- tree nodes have learned object category associations

classification algorithm

SVM, decision forest, boosting

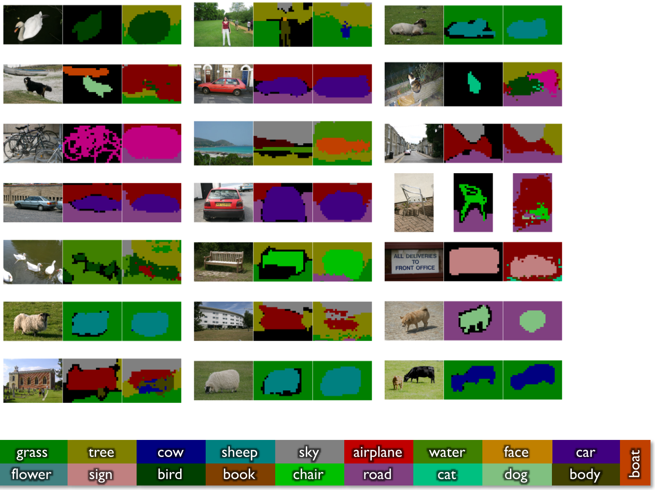
[Source: Shotton et al.]

Example Semantic Texton Forest



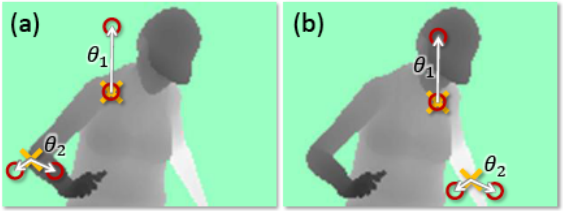
[Source: Shotton et al.]

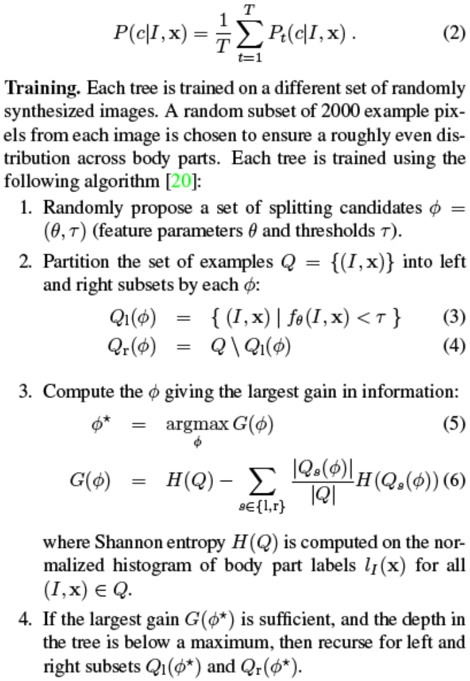
MSRC Dataset Results



[Source: Shotton et al.]

Microsoft Kinect





$$P(c|I,\mathbf{x}) = \frac{1}{T}\sum_{t=1}^{T} P_t(c|I,\mathbf{x})\,. \tag{2}$$

Training. Each tree is trained on a different set of randomly synthesized images. A random subset of 2000 example pixels from each image is chosen to ensure a roughly even distribution across body parts. Each tree is trained using the following algorithm [20]:

1. Randomly propose a set of splitting candidates $\phi = (\theta, \tau)$ (feature parameters θ and thresholds τ).
2. Partition the set of examples $Q = \{(I, \mathbf{x})\}$ into left and right subsets by each ϕ:

$$Q_{\mathrm{l}}(\phi) = \{\, (I,\mathbf{x}) \mid f_\theta(I,\mathbf{x}) < \tau \,\} \tag{3}$$

$$Q_{\mathrm{r}}(\phi) = Q \setminus Q_{\mathrm{l}}(\phi) \tag{4}$$

3. Compute the ϕ giving the largest gain in information:

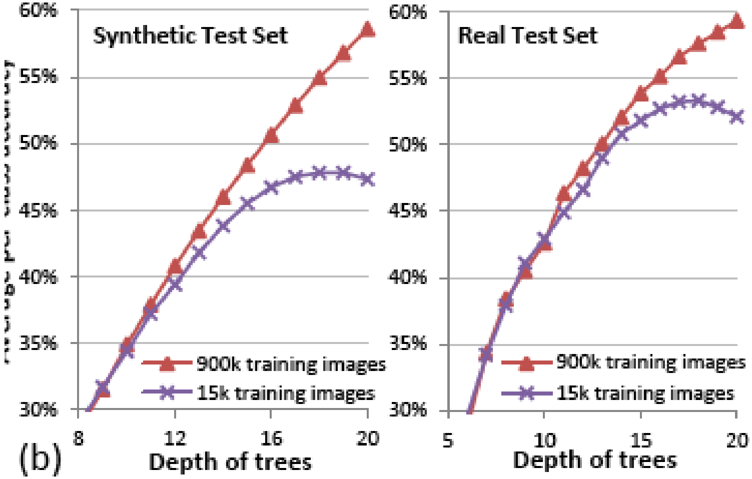
$$\phi^\star = \operatorname*{argmax}_{\phi} G(\phi) \tag{5}$$

$$G(\phi) = H(Q) - \sum_{s\in\{\mathrm{l},\mathrm{r}\}} \frac{|Q_s(\phi)|}{|Q|} H(Q_s(\phi)) \tag{6}$$

   where Shannon entropy $H(Q)$ is computed on the normalized histogram of body part labels $l_I(\mathbf{x})$ for all $(I, \mathbf{x}) \in Q$.

4. If the largest gain $G(\phi^\star)$ is sufficient, and the depth in the tree is below a maximum, then recurse for left and right subsets $Q_{\mathrm{l}}(\phi^\star)$ and $Q_{\mathrm{r}}(\phi^\star)$.

Microsoft Kinect


Synthetic Test Set
Real Test Set
900k training images
15k training images
Depth of trees
(b)