• This week’s office hours are today after class
• Canceling Wednesday’s office hours because …

• Homework 4 due on Wednesday
Lecture outline

• Origin and motivation of the “bag of words” model

• Algorithm pipeline
  • Extracting local features
  • Learning a dictionary — clustering using k-means
  • Encoding methods — hard vs. soft assignment
  • Spatial pooling — pyramid representations
  • Similarity functions and classifiers

Figure from Chatfield et al.,2011
Bag of features
Texture is characterized by the repetition of basic elements or *textons*.

For stochastic textures, it is the identity of the textons, not their spatial arrangement, that matters.

---

Origin 1: Texture recognition

Origin 2: Bag-of-words models

Origin 2: Bag-of-words models

Origin 2: Bag-of-words models

- Orderless document representation: frequencies of words from a dictionary  
  Salton & McGill (1983)
Origin 2: Bag-of-words models

Lecture outline

• Origin and motivation of the “bag of words” model

• Algorithm pipeline
  • Extracting local features
  • Learning a dictionary — clustering using k-means
  • Encoding methods — hard vs. soft assignment
  • Spatial pooling — pyramid representations
  • Similarity functions and classifiers

Figure from Chatfield et al., 2011
Local feature extraction

- Regular grid or interest regions

blob detector
Local feature extraction

**Compute descriptor**

**Normalize patch**

**Detect patches**

**Choices of descriptor:**
- SIFT
- Filterbank histograms
- The patch itself
Local feature extraction

Extract features from many images

Slide credit: Josef Sivic
Lecture outline

- Origin and motivation of the “bag of words” model
- Algorithm pipeline
  - Extracting local features
  - Learning a dictionary — clustering using k-means
  - Encoding methods — hard vs. soft assignment
  - Spatial pooling — pyramid representations
  - Similarity functions and classifiers

Figure from Chatfield et al., 2011
Learning a dictionary

Slide credit: Josef Sivic
Learning a dictionary
Learning a dictionary

Visual vocabulary

Clustering

Slide credit: Josef Sivic
Review: K-means clustering

- Want to minimize sum of squared Euclidean distances between features \( x_i \) and their nearest cluster centers \( m_k \)

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

**Algorithm:**

- Randomly initialize K cluster centers
- Iterate until convergence:
  - Assign each feature to the nearest center
  - Recompute each cluster center as the mean of all features assigned to it
Example codebook

Source: B. Leibe
Another codebook

Source: B. Leibe
Lecture outline

- Origin and motivation of the “bag of words” model
- Algorithm pipeline
  - Extracting local features
  - Learning a dictionary — clustering using k-means
  - Encoding methods — hard vs. soft assignment
  - Spatial pooling — pyramid representations
  - Similarity functions and classifiers

Figure from Chatfield et al., 2011
Encoding methods

- Assigning words to features

Visual vocabulary

Also called hard assignment

Partition of space
Encoding methods

- Assigning words to features

Visual vocabulary

Different words
Similar features

Partition of space

Hard assignment

Large quantization error
Encoding methods

- Assigning words to features

**Visual vocabulary**

**Soft assignment**

\[ \alpha_i \propto e^{-f(d(x,c_i))} \]

assign high weights to centers that are close in practice non-zero to only k-nearest neighbors.

partition of space
Encoding methods

- Assigning words to features

Visual vocabulary

Soft assignment

\[ \alpha_i \propto e^{-f(d(x, c_i))} \]

Similar features

Partition of space

Soft assignment

0.6 0 0.4 0.4 0 0.6

Hard assignment

1 0 0 0 0 1
• What should be the size of the dictionary?
  • Too small: don’t capture the variability of the dataset
  • Too large: have too few points per cluster
  • The right size depends on the task and amount of data
    - e.g. instance retrieval (e.g. Nister) uses a vocabulary of 1 million, whereas recognition (e.g., texture) uses a vocabulary of about a hundred.

• Speed of embedding
  • Tree structured vocabulary (e.g. Nister)
  • Hashing, product quantization

• More accurate embeddings
  • Generalizations of soft embedding: LLC coding, sparse coding
  • Higher order statistics: Fisher vectors, VLAD, etc.
Lecture outline

- Origin and motivation of the “bag of words” model
- Algorithm pipeline
  - Extracting local features
  - Learning a dictionary — clustering using k-means
  - Encoding methods — hard vs. soft assignment
  - Spatial pooling — pyramid representations
- Similarity functions and classifiers

Figure from Chatfield et al., 2011
Spatial pyramids

**pooling**: sum embeddings of local features within a region

Lazebnik, Schmid & Ponce (CVPR 2006)
Spatial pyramids

**pooling:** sum embeddings of local features within a region

Same motivation as **SIFT** — keep coarse layout information

Lazebnik, Schmid & Ponce (CVPR 2006)
Spatial pyramids

**pooling:** sum embeddings of local features within a region

Same motivation as **SIFT** — keep coarse layout information

Lazebnik, Schmid & Ponce (CVPR 2006)
• Origin and motivation of the “bag of words” model

• Algorithm pipeline
  • Extracting local features
  • Learning a dictionary — clustering using k-means
  • Encoding methods — hard vs. soft assignment
  • Spatial pooling — pyramid representations
  • Similarity functions and classifiers

Figure from Chatfield et al., 2011
Bags of features representation

\[ I \]

\[ h = \Phi(I) \]

image similarity = feature similarity
Comparing features

- **Euclidean distance:**
  \[ D(h_1, h_2) = \sqrt{\sum_{i=1}^{N} (h_1(i) - h_2(i))^2} \]

- **L1 distance:**
  \[ D(h_1, h_2) = \sum_{i=1}^{N} |h_1(i) - h_2(i)| \]

- **\(\chi^2\) distance:**
  \[ D(h_1, h_2) = \sum_{i=1}^{N} \frac{(h_1(i) - h_2(i))^2}{h_1(i) + h_2(i)} \]

- **Histogram intersection (similarity):**
  \[ I(h_1, h_2) = \sum_{i=1}^{N} \min(h_1(i), h_2(i)) \]

- **Hellinger kernel (similarity):**
  \[ K(h_1, h_2) = \sum_{i=1}^{N} \sqrt{h_1(i) \cdot h_2(i)} \]
Given a feature representation for images, how do we learn a model for distinguishing features from different classes?
Classifiers

- Given a feature representation for images, how do we learn a model for distinguishing features from different classes?
- Examples of commonly used classifiers
  - Nearest neighbor classifiers
  - Linear classifiers: support vector machines
Nearest neighbor classifier

- Assign label of nearest training data point to each test data point

(from Duda et al.)
For a new point, find the $k$ closest points from training data.

Labels of the $k$ points "vote" to classify.
Linear classifiers
Linear classifiers

- Find linear function (hyperplane) to separate positive and negative examples

\[ x_i \text{ positive: } x_i \cdot w + b \geq 0 \]
\[ x_i \text{ negative: } x_i \cdot w + b < 0 \]

Which hyperplane is best?
Support vector machines

- Find hyperplane that maximizes the *margin* between the positive and negative examples

• Find hyperplane that maximizes the margin between the positive and negative examples

\[ x_i \text{ positive } (y_i = 1): \quad x_i \cdot w + b \geq 1 \]
\[ x_i \text{ negative } (y_i = -1): \quad x_i \cdot w + b \leq -1 \]

For support vectors,
\[ x_i \cdot w + b = \pm 1 \]

Distance between point and hyperplane:
\[ \frac{|x_i \cdot w + b|}{\|w\|} \]

Therefore, the margin is \( \frac{2}{\|w\|} \)

1. Maximize margin \( 2 / \|w\| \)

2. Correctly classify all training data:

\[
x_i \text{ positive } (y_i = 1) : \quad x_i \cdot w + b \geq 1 \\
x_i \text{ negative } (y_i = -1) : \quad x_i \cdot w + b \leq -1
\]

**Quadratic optimization problem:**

\[
\min_{w,b} \frac{1}{2} \|w\|^2 \quad \text{subject to} \quad y_i (w \cdot x_i + b) \geq 1
\]

Finding the maximum margin hyperplane

• Solution:

\[ w = \sum_i \alpha_i y_i x_i \]

Learned weight
(nonzero only for support vectors)

Finding the maximum margin hyperplane

- Solution:
  \[ w = \sum_{i} \alpha_i y_i x_i \]
  \[ w \cdot x_i + b = y_i, \text{ for any support vector} \]

- Classification function (decision boundary):
  \[ w \cdot x + b = \sum_{i} \alpha_i y_i x_i \cdot x + b \]

- Notice that it relies on an inner product between the test point \( x \) and the support vectors \( x_i \).

- Solving the optimization problem also involves computing the inner products \( x_i \cdot x_j \) between all pairs of training points.
What if the data is not linearly separable?

- **Separable:**
  \[
  \min_{w,b} \frac{1}{2} \|w\|^2 \quad \text{subject to} \quad y_i(w \cdot x_i + b) \geq 1
  \]

- **Non-separable:**
  \[
  \min_{w,b} \frac{1}{2} \|w\|^2 + C \sum_{i=1}^{n} \xi_i \quad \text{subject to} \quad y_i(w \cdot x_i + b) - 1 + \xi_i \geq 0
  \]

- **C:** tradeoff constant, \( \xi_i: \text{slack variable} \) (positive)
- Whenever margin is \( \geq 1 \), \( \xi_i = 0 \)
  \[
  \xi_i = 1 - y_i(w \cdot x_i + b)
  \]
- Whenever margin is \( < 1 \),
What if the data is not linearly separable?

\[
\min_{w,b} \frac{1}{2} \|w\|^2 + C \sum_{i=1}^{n} \max(0, 1 - y_i (w \cdot x_i + b))
\]

- Maximize margin
- Minimize classification mistakes
What if the data is not linearly separable?

$$\min_{w,b} \frac{1}{2} \|w\|^2 + C \sum_{i=1}^{n} \max(0, 1 - y_i (w \cdot x_i + b))$$

Demo: [http://cs.stanford.edu/people/karpathy/svmjs/demo](http://cs.stanford.edu/people/karpathy/svmjs/demo)
Datasets that are linearly separable work out great:

But what if the dataset is just too hard?

We can map it to a higher-dimensional space:
Nonlinear SVMs

- General idea: the original input space can always be mapped to some higher-dimensional feature space where the training set is separable:

$$\Phi: \mathbf{x} \rightarrow \varphi(\mathbf{x})$$
Nonlinear SVMs

• **The kernel trick:** instead of explicitly computing the lifting transformation $\varphi(x)$, define a kernel function $K$ such that

$$K(x, y) = \varphi(x) \cdot \varphi(y)$$

(the kernel function must satisfy *Mercer’s condition*)

• This gives a nonlinear decision boundary in the original feature space:

$$\sum_i \alpha_i y_i \varphi(x_i) \cdot \varphi(x) + b = \sum_i \alpha_i y_i K(x_i, x) + b$$

Non-linear kernels for histograms

- Histogram intersection kernel:
  \[ I(h_1, h_2) = \sum_{i=1}^{N} \min(h_1(i), h_2(i)) \]

- Hellinger kernel:
  \[ K(h_1, h_2) = \sum_{i=1}^{N} \sqrt{h_1(i) h_2(i)} \]

- Generalized Gaussian kernel:
  \[ K(h_1, h_2) = \exp\left( -\frac{1}{A} D(h_1, h_2)^2 \right) \]

- \( D \) can be L1, Euclidean, \( \chi^2 \) distance, etc.

Summary: SVMs for image classification

1. Pick an image representation (in our case, bag of features)
2. Pick a kernel function for that representation
3. Feed the kernel and features into your favorite SVM solver to obtain support vectors and weights
4. At test time: compute kernel values for your test example and each support vector, and combine them with the learned weights to get the value of the decision function

\[ \sum_i \alpha_i y_i \varphi(x_i) \cdot \varphi(x) + b = \sum_i \alpha_i y_i K(x_i, x) + b \]

Lots of software available! LIBSVM, LIBLINEAR, SVMLight
What about multi-class SVMs?

• Many options!

• For example, we have to obtain a multi-class SVM by combining multiple two-class SVMs

• One vs. rest
  • Training: learn an SVM for each class vs. the rest
  • Testing: apply each SVM to test example and assign to it the class of the SVM that returns the highest decision value

• One vs. one
  • Training: learn an SVM for each pair of classes
  • Testing: each learned SVM “votes” for a class to assign to the test example

• [http://www.kernel-machines.org/software](http://www.kernel-machines.org/software)
Lecture outline

• Origin and motivation of the “bag of words” model
• Algorithm pipeline
  • Extracting local features
  • Learning a dictionary — clustering using k-means
  • Encoding methods — hard vs. soft assignment
  • Spatial pooling — pyramid representations
• Similarity functions and classifiers

Putting it all together

Figure from Chatfield et al., 2011
Multi-class classification results  
(100 training images per class)

<table>
<thead>
<tr>
<th>Level</th>
<th>Weak features (vocabulary size: 16)</th>
<th>Strong features (vocabulary size: 200)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Single-level</td>
<td>Pyramid</td>
</tr>
<tr>
<td>0 (1 × 1)</td>
<td>45.3 ±0.5</td>
<td>56.2 ±0.6</td>
</tr>
<tr>
<td>1 (2 × 2)</td>
<td>53.6 ±0.3</td>
<td>64.7 ±0.7</td>
</tr>
<tr>
<td>2 (4 × 4)</td>
<td>61.7 ±0.6</td>
<td>66.8 ±0.6</td>
</tr>
<tr>
<td>3 (8 × 8)</td>
<td>63.3 ±0.8</td>
<td>66.8 ±0.6</td>
</tr>
</tbody>
</table>
## Results: Caltech-101 dataset

### Multi-class classification results (30 training images per class)

<table>
<thead>
<tr>
<th>Level</th>
<th>Weak features (16)</th>
<th>Strong features (200)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Single-level</td>
<td>Pyramid</td>
</tr>
<tr>
<td>0</td>
<td>15.5 ±0.9</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>31.4 ±1.2</td>
<td>32.8 ±1.3</td>
</tr>
<tr>
<td>2</td>
<td>47.2 ±1.1</td>
<td>49.3 ±1.4</td>
</tr>
<tr>
<td>3</td>
<td>52.2 ±0.8</td>
<td><strong>54.0</strong> ±1.1</td>
</tr>
</tbody>
</table>
Further thoughts and readings ...

- All about embeddings (detailed experiments and code)
  - K. Chatfield et al., The devil is in the details: an evaluation of recent feature encoding methods, BMVC 2011
  - [http://www.robots.ox.ac.uk/~vgg/research/encoding_eval/](http://www.robots.ox.ac.uk/~vgg/research/encoding_eval/)
  - Includes discussion of advanced embeddings such as Fisher vector representations and locally linear coding (LLC)


- Fast non-linear SVM evaluation (scales linearly with #SVs)
  - Classification using Intersection kernel SVMs is efficient, Maji et al., CVPR 2008 — O(1) evaluation ~ 1000x faster on on large datasets! (Also see the PAMI 2013 paper on my webpage)
  - Approximate embeddings for kernels (Maji and Berg, Vedaldi and Zisserman) — O(n) training ~ 100x faster on large datasets!