Clustering Application
Codewords: BOW for Vision

Here classifying scenes
Also works for detecting (but not localizing) objects

Beach  City  Kitchen
What is a Good Feature?

- Not “palm tree” or “toaster” (why?)
- Not individual pixels (why?)
- Not blocks or patches of pixels (why?)
- Successful systems can (often)
  - say “car” (vs. face or airplane) without necessarily finding the car
  - say “leopard” (vs. dog or elephant) even though the leopard has been excised from the image
BOW Features for Images

• Combine all training images together
• Some normalization (remove color?, even intensity,...)
• For each image collect a sample of patches
  – squares of same or (often) different sizes
  – random positions (or at interest points, or...)
  – normalize the size, intensity...
• Create a clustering for all the patches regardless of class
• These clusters are known as “codewords”
Bag of Codeword Features

Collect Patches From Training but no Labels

Normalize

Form Clusters
BOW Image Classification

- New feature space
- Any image can now be represented as a bag of codewords:
  \[ \text{Image12} = \{C_{16} \ 3, \ C_{81} \ 36, \ C_{97} \ 12, \ C_{161} \ 2\ldots\} \]
- Extreme dimensionality reduction (12MP image to distribution over k features)
- Train a classifier on the labeled re-represented training images
Class Conditional Distributions

“Beach” distribution of clusters / codewords
Codewords
Cluster Averages

New Image
BOW Image Classification

• Argmax $Pr(Class \mid Clusters=\text{image clusters})$
  – might assume conditional independence among clusters given class like naïve Bayes, linearity like perceptrons or logistic regression
  – k-variate Gaussian; respect $2^{nd}$ moment (the 2-way covariate structure)
  – Some other classifier

• Note the parallel to lossy data compression
  – Find patterns and re-represent
  – Preserve as little information as possible
Image Classification

The trained classifier assigns this:

And therefore this:

To the BEACH class

Because it is more likely to be a draw from the BEACH distribution
Final Word on Features

- Crucial for successful AI applications that rely on machine learning
- Often somewhat *ad hoc* (like tf-idf, local image patches)
- Difficult to generate (little deep understanding or guidance on inventing)
- Try to capture relevant aspects; experiment
- Try not to limit the learner
- Rely on lots of data
Rise of Big Data

Task: word substitution / usage
Ex: then/than; principle/principal

Different learning methods
analytic part (modest impact)

Different amounts of training
empirical part (large impact)

The phenomenon seems to be quite general

Best: weak, flexible prior analytic models
with *lots* of data (not human numbers)

With poor or limited features all would flatten out very early
Back to Perceptrons
(linear threshold units, linear discriminators)

• Perceptrons are not very expressive
• Due to assumption of linearity
• ANNs (including deep networks) are one solution
• Or use functions of native features (allowing interactions)
  – Feature engineering (but unprincipled)
  – Kernel methods
• Be linear but in different space
  – Higher dimensional
  – Possibly infinite dimensional
Recall Perceptrons

- If there is one perceptron,
- there are many
- Are some better?
- Is one best?
- Can we tell?
- Can we find it?
What’s the Best Separating Hyperplane?

The larger the margin, the lower the capacity, less overfitting.

But we can have any margin we want by expanding the space...

Need to normalize.
What’s the Best Separating Hyperplane?

Can use the radius $r$ of the smallest enclosing sphere.

Capacity is related to $2r/m$.

Number of mistakes in training is bounded by $(2r/m)^2$ [Novikoff 1963].
What’s the Best Separating Hyperplane?

Potentially only a few training examples determine the margin. These are called Support Vectors.

Classifier boundary as a linear combination of the support vectors:

\[ a_1 D(x,V_1) + a_2 D(x,V_2) + a_3 D(x,V_3) = 0 \]  

(distances to \( V_i \)'s)

\[ \sum a_i D(x,V_i) = 0 \]
Why are Large Margins Better?

• Classification is more robust / stable w.r.t.
  – Small changes to training examples
  – New examples
  – Re-sampling training data

• Lower expressiveness / capacity
  – Larger margin → fewer very different hypotheses
  – “Fat Shattering” dimension rather than “Shattering” & “VC dimension”

• We do not choose the margin, it emerges
  – Possibly: Train, Measure margin, Calculate significance(?)
  – Analysis bounds for calculation are very loose & cannot be quantitatively trusted
  – Margin works well as a regularization penalty; a bias to stabilize learning
  – Large margin bias is parameterized by choice of distance and (interestingly) by training examples
Support Vector Machine

• Only use the nearest / most constraining points (support vectors)
• A learner that finds them is called a Support Vector Machine (SVM)
• Finding them is a quadratic programming optimization problem
  – There are efficient iterative solutions
  – Given certain conditions to insure convexity
• Note class density estimation is not necessary (meaning?)
• Maximizing margin reduces overfitting
• Very popular although being supplanted in some applications by deep learning
• Many extensions
  – Noise, outliers, non-separable classes, imbalanced training...
  – Soft margins, Margin distributions, Asymmetric margins...
Kernel Spaces: Distance Metrics

• Instead of adding perceptron layers, choose a better distance metric
• What???
• Think about MNIST digits
  – Want 7’s to be close together
  – 8’s to be close but far from 7’s and 2’s, etc.
• Independent pixel distances does not work well
• What are we missing?
  – Pixels do not contribute monotonically nor independently
  – Must appreciate interactions among pixels
Kernel Methods

• Map to a new higher dimensional space
  – Can be very high
  – Can be infinite

• Kernel functions
  – Introduce high dimensionality efficiently
  – Computation is independent of dimensionality
  – Defined w/ dot product of inputs
    (information on the Cosine between feature vectors)

• A kernel function defines a similarity metric among examples
Mercer’s Condition / Representer Theorem

• \( K(s, x) = f(s \cdot x) \)

• Kernel uses only
  – the dot product between two examples
  – the unknown \( x \)
  – and a support vector \( s \)

• The desired (high dimensional) hyperplane can be written:

\[
\sum_{i=1}^{m} \alpha_i K(s_i, x)
\]

Linear weighted sum of similarities to support vectors

• Thus, the hypothesis space is represented efficiently by using some of the training examples – the support vectors
Consider MNIST Handwritten Digits

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>/</td>
<td>/</td>
<td>/</td>
<td>/</td>
<td>/</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td></td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Pixel input: 32 x 32 x 8
- \( x = 1024 \) features / dimensions, each 256 values
- Generic ANNs work poorly
- Carefully designed ANNs work very well
- SVMs work well with little design work
SVMs for Digit Images

- $K(x,y) = (x \cdot y)^3$ or $(x \cdot y + 1)^3$
- Dot product $\rightarrow$ scalar; cube it
  Consider how this works...
- Before $32^2$ features (or about $10^3$)
- Now $\sim (32^2)^3$ features (or about $10^9$)
- New Feature = monomial = correlation among three pixels
- $VC(lin\, sep) \sim$ # dimensions
- Overfitting problem?
  - Not if the margin is large
  - Monitor number of support vectors