Eigen representations;
Detecting faces in images

Class 5. 7 Sep 2010

Instructor: Bhiksha Raj
Administrivia

- Homeworks were up last week
  - Questions to be directed to Sourish/Sohail/Myself
  - Delays are worth negative points 😊

- Project ideas next class
  - Begin thinking about what project you will do
  - You are welcome to think up your own ideas/projects
  - Think workshop paper – novel problems, novel ideas can get published

- Projects will be done by teams
  - 2-4 people
  - Begin forming teams by yourselves
  - Students without teams will be assigned to teams

- Class of 28th: Intel’s open house
Last Class: Representing Audio

- Basic DFT
- Computing a Spectrogram
- Computing additional features from a spectrogram

7 Sep 2010
What about images?

- DCT of small segments
  - 8x8
  - Each image becomes a matrix of DCT vectors
- DCT of the image
- Haar transform (checkerboard)
- Or data-driven representations..
Returning to Eigen Computation

- A collection of faces
  - All normalized to 100x100 pixels
- What is common among all of them?
  - Do we have a common descriptor?
A least squares typical face

- Can we do better than a blank screen to find the most common portion of faces?
  - The first checkerboard; the zeroth frequency component..

- Assumption: There is a “typical” face that captures most of what is common to all faces
  - Every face can be represented by a scaled version of a typical face

- Approximate every face \( f \) as \( f = w_f V \)

- Estimate \( V \) to minimize the squared error over all faces
  - How?
  - What is \( V \)?
A collection of least squares typical faces

- Assumption: There are a set of $K$ “typical” faces that captures most of all faces

- Approximate every face $f$ as $f = w_{f,1} V_1 + w_{f,2} V_2 + w_{f,3} V_3 + \ldots + w_{f,k} V_k$
  - $V_2$ is used to “correct” errors resulting from using only $V_1$
    - So the total energy in $w_{f,2}$ must be lesser than the total energy in $w_{f,1}$
    - $\Sigma w_{f,2}^2 > \Sigma w_{f,1}^2$
  - $V_3$ corrects errors remaining after correction with $V_2$
    - The total energy in $w_{f,3}$ must be lesser than that even in $w_{f,2}$
  - And so on..
  - $V = [V_1 \ V_2 \ V_3]$

- Estimate $V$ to minimize the squared error
  - How?
  - What is $V$?
A recollection

\[ M = \]

\[ W = \]

\[ V = \text{PINV}(W) \times M \]

\[ U = \ ? \]
How about the other way?

\[ W = M \times P_{\text{inv}}(V) \]

\[ M = \]

\[ V = \]

\[ W = \]

\[ U = \]

\[ W = M \times P_{\text{inv}}(V) \]
How about the other way?

\[ M = \]

\[ V = \]

\[ W = \]

\[ U = \]

\[ WV \approx M \]
The Correlation and Covariance Matrices

Consider a set of column vectors represented as a $\text{DxN}$ matrix $\mathbf{M}$

- The \textit{correlation} matrix is
  
  $\mathbf{C} = \frac{1}{N} \mathbf{M} \mathbf{M}^\top = \frac{1}{N} \sum_i \mathbf{m}_i \mathbf{m}_i^\top$

  - Diagonal elements represent average of the squared value of each dimension
  - Off diagonal elements represent how two components are related
    - How much knowing one lets us guess the value of the other
  
  \[ \frac{1}{N} \sum_i m_{1,i}^2 \]
  \[ \frac{1}{N} \sum_i m_{k,i} m_{k,j} \]
The Correlation and Covariance Matrices

If we “center” the data first, we get the **covariance** matrix

Mean = $1/N \sum_i M_i$

- $M_i$ is the $i$th column of $M$

The centered data are obtained by subtracting the mean from every column of $M$

- $M_{\text{centered}} = M - \text{Mean} * 1$
- $1$ is a $1 \times N$ row matrix

The **Covariance** matrix is

- $\text{Cov} = (1/N) M_{\text{centered}} M_{\text{centered}}^T$

$$
\begin{align*}
M & \quad M^T \\
\begin{array}{cccc}
\vdots & \vdots & \vdots & \vdots \\
\vdots & \vdots & \vdots & \vdots \\
\vdots & \vdots & \vdots & \vdots \\
\vdots & \vdots & \vdots & \vdots \\
\vdots & \vdots & \vdots & \vdots \\
\end{array}
\end{align*}
= \begin{align*}
N & \quad C \\
\begin{array}{cccc}
\vdots & \vdots & \vdots & \vdots \\
\vdots & \vdots & \vdots & \vdots \\
\vdots & \vdots & \vdots & \vdots \\
\vdots & \vdots & \vdots & \vdots \\
\vdots & \vdots & \vdots & \vdots \\
\end{array}
\end{align*}
$$

$$(1/N) \sum_i m_{1,i}^2$$

$$(1/N) \sum_i m_{k,i} m_{k,j}$$
Correlation / Covariance matrix

- Covariance and correlation matrices are symmetric
  - $C_{ij} = C_{ji}$

Properties of symmetric matrices:

- Eigenvalues and Eigenvectors are real
- Can be expressed as
  - $C = VLV^T$
  - $V$ is the matrix of Eigenvectors
  - $L$ is a diagonal matrix of Eigenvalues
  - $V^T = V^{-1}$
Correlation / Covariance Matrix

The square root of a correlation or covariance matrix is easily derived from the eigen vectors and eigen values

- The eigen values of the square root of the correlation matrix are the square roots of the eigen values of the correlation matrix
- These are also the “singular values” of the data set
Square root of the Covariance Matrix

- The square root of the covariance matrix represents the elliptical scatter of the data.
- The eigenvectors of the matrix represent the major and minor axes.
PCA: The Covariance Matrix

Any vector $V = a_{v,1} \cdot \text{eigenvec1} + a_{v,2} \cdot \text{eigenvec2} + ..$

$\Sigma_V a_{V,i} = \text{eigenvalue}(i)$

- Projections along the N eigenvectors with the largest eigenvalues represent the N most *informative* components of the matrix
  - N directions along which variance is maximum
  - These represent the N *principal components*
An audio example

- The spectrogram has 974 vectors of dimension 1025
- The covariance matrix is size 1025 x 1025
- There are 1025 eigenvectors
Eigen Reduction

\[ M = \text{spectrogram} \quad \text{1025x1000} \]

\[ C = M_{\text{centered}} M_{\text{centered}}^T \quad \text{1025x1025} \]

\[ V = 1025x1025 \]

\[ [V, L] = \text{eig} (C) \]

\[ V_{\text{reduced}} = [V_1 \ldots V_{25}] \quad \text{1025x25} \]

\[ M_{\text{lowdim}} = \text{Pinv}(V_{\text{reduced}}) M \quad \text{25x1000} \]

\[ M_{\text{reconstructed}} = V_{\text{reduced}} M_{\text{lowdim}} \quad \text{1025x1000} \]

- Compute the **Covariance**
- Compute Eigen vectors and values
- Create matrix from the 25 Eigen vectors corresponding to 25 highest Eigen values
- Compute the weights of the 25 eigenvectors
- To reconstruct the spectrogram – compute the projection on the 25 eigen vectors
Eigenvalues and Eigenvectors

- **Left panel:** Matrix with 1025 eigen vectors
- **Right panel:** Corresponding eigen values
  - Most eigen values are close to zero
  - The corresponding eigenvectors are “unimportant”
Eigenvalues and Eigenvectors

The vectors in the spectrogram are linear combinations of all 1025 eigen vectors

The eigen vectors with low eigen values contribute very little
  - The average value of $a_i$ is proportional to the square root of the eigenvalue
  - Ignoring these will not affect the composition of the spectrogram

Vec = $a_1 \cdot \text{eigenvec1} + a_2 \cdot \text{eigenvec2} + a_3 \cdot \text{eigenvec3} \ldots$
An audio example

- The same spectrogram projected down to the 25 principal eigenvectors with the highest eigenvalues
- Only the 25-dimensional weights are shown
- The weights with which the 25 eigen vectors must be added to compose a least squares approximation to the spectrogram

\[ V_{reduced} = [V_1 \ldots V_{25}] \]

\[ M_{lowdim} = P\text{inv}(V_{reduced})M \]
An audio example

\[ M_{\text{reconstructed}} = V_{\text{reduced}} M_{\text{lowdim}} \]

- The same spectrogram constructed from only the 25 eigenvectors with the highest eigen values
  - Looks similar
    - With 100 eigenvectors, it would be indistinguishable from the original
  - Sounds pretty close
  - But now sufficient to store 25 numbers per vector (instead of 1024)
With only 5 eigenvectors

- The same spectrogram constructed from only the 5 eigen vectors with the highest eigen values
  - Highly recognizable
Covariance vs. Correlation

- If Eigenvectors are computed from the correlation matrix, they represent the most energy carrying bases
  - As opposed to the most informative bases obtained from the covariance
    - If the data are centered, the two are the same, but not otherwise

- Eigen decomposition of Correlations:
  - Direct computation using Singular Value Decomposition
Covariance vs. correlation

- Data are Gaussian, mean at [3,3]

- Left: Eigen vectors from covariance
  - Aligned to the direction of scatter
  - But not aligned to data

- Right: Eigen vectors from correlation (SVD)
  - Aligned to average direction of data
  - But not the scatter
Singular Value Decomposition

- A matrix decomposition method
  \[ M = U \cdot \Sigma \cdot V^T \]
  \[ U \cdot U^T = I, \quad V \cdot V^T = I, \quad \Sigma \text{ is diagonal} \]

- Breaks up the input into a product of three matrices, two orthogonal and one diagonal
  \[
  \begin{pmatrix}
  \mathbf{M} \\
  \mathbf{U} \\
  \mathbf{\Sigma} \\
  \mathbf{V}^T
  \end{pmatrix}
  =
  \begin{pmatrix}
  \mathbf{D} \times \mathbf{N} \\
  \mathbf{D} \times \mathbf{D} \\
  \mathbf{D} \times \mathbf{N} \\
  \mathbf{N} \times \mathbf{N}
  \end{pmatrix}
  \]

- The right matrix are Eigenvectors in row space
- The diagonal will represent how much spread is in each direction and contains the *singular values*
  - Also the square root of the eigen value matrix of the correlations
- The left matrix are the Eigen vectors of column space
  - Also Eigenvectors of correlation
SVD vs. Eigen decomposition

- Singular value decomposition is analogous to the eigen decomposition of the correlation matrix of the data.
- The “left” singular vectors are the eigenvectors of the correlation matrix.
  - Show the directions of greatest importance
- The corresponding singular values are the square roots of the eigenvalues of the correlation matrix.
  - Show the importance of the eigenvector
Thin SVD, compact SVD, reduced SVD

- **Thin SVD**: Only compute the first $N$ columns of $U$
  - All that is required if $N < M$

- **Compact SVD**: Only the left and right eigen vectors corresponding to non-zero singular values are computed

- **Reduced SVD**: Only compute the columns of $U$ corresponding to the $K$ highest singular values
Eigen Faces!

- Here $W$, $V$ and $U$ are ALL unknown and must be determined
  - Such that the squared error between $U$ and $M$ is minimum

- Eigen analysis allows you to find $W$ and $V$ such that $U = WV$ has the least squared error with respect to the original data $M$

- If the original data are a collection of faces, the columns of $W$ are eigen faces
  - Should the data be centered?
Eigen faces

- Lay all faces side by side in vector form to form a matrix
  - In my example: 300 faces. So the matrix is 10000 x 300
- Multiply the matrix by its transpose
  - The correlation matrix is 10000x10000
Eigen faces

\[ [U, S] = \text{eig}(\text{correlation}) \]

\[
S = \begin{bmatrix}
\lambda_1 & 0 & 0 \\
0 & \lambda_2 & 0 \\
\vdots & \vdots & \ddots \\
0 & 0 & \lambda_{10000}
\end{bmatrix}
\]

\[ U = \begin{bmatrix}
eigenface_1 \\
eigenface_2 \\
\vdots \\
eigenface_{10000}
\end{bmatrix} \]

- **Compute the eigen vectors**
  - Only 300 of the 10000 eigen values are non-zero
  - Why?
- **Retain eigen vectors with high eigen values (>0)**
  - Could use a higher threshold
Eigen Faces

- The eigen vector with the highest eigen value is the first typical face.
- The vector with the second highest eigen value is the second typical face.
- Etc.

$U = \begin{bmatrix}
\text{eigenface}_1 \\
\text{eigenface}_2 \\
\vdots
\end{bmatrix}$
Representing a face

\[ \begin{align*}
\text{Representation} & = [w_1 \ w_2 \ w_3 \ \ldots ]^T \\
\end{align*} \]

- The weights with which the eigen faces must be combined to compose the face are used to represent the face!
Do we need to compute a 10000 x 10000 correlation matrix and then perform Eigen analysis?
- Will take a very long time on your laptop

SVD
- Only need to perform “Thin” SVD. Very fast
  - U = 10000 x 300
    - The columns of U are the eigen faces!
    - The Us corresponding to the “zero” eigen values are not computed
  - S = 300 x 300
  - V = 300 x 300
NORMALIZING OUT VARIATIONS
What are the obvious differences in the above images?

How can we capture these differences?

- Hint – image histograms.
Images -- Variations

- Pixel histograms: what are the differences
Normalizing Image Characteristics

- Normalize the pictures
  - Eliminate lighting/contrast variations
  - All pictures must have “similar” lighting
    - How?

- Lighting and contrast are represented in the pixel value histograms:
Histogram Equalization

- Normalize histograms of images
  - Maximize the contrast
    - Contrast is defined as the “flatness” of the histogram
    - For maximal contrast, every greyscale must happen as frequently as every other greyscale

- Maximizing the contrast: Flattening the histogram
  - Doing it for every image ensures that every image has the same contrast
    - I.e. exactly the same histogram of pixel values
      - Which should be flat
Histogram Equalization

- Modify pixel values such that histogram becomes “flat”.
- For each pixel
  - New pixel value = f(old pixel value)
  - What is f()?
- Easy way to compute this function: map cumulative counts
The histogram (count) of a pixel value $X$ is the number of pixels in the image that have value $X$
- E.g. in the above image, the count of pixel value 180 is about 110

The cumulative count at pixel value $X$ is the total number of pixels that have values in the range $0 \leq x \leq X$
- $CCF(X) = H(1) + H(2) + \ldots + H(X)$
Cumulative Count Function

- The cumulative count function of a uniform histogram is a line

- We must modify the pixel values of the image so that its cumulative count is a line
Mapping CCFs

- \[ \text{CCF}(f(x)) \rightarrow a \cdot f(x) \quad [\text{of } a \cdot (f(x)+1) \text{ if pixels can take value 0}] \]
  - \( x = \) pixel value
  - \( f() \) is the function that converts the old pixel value to a new (normalized) pixel value
  - \( a = \frac{\text{total no. of pixels in image}}{\text{total no. of pixel levels}} \)
    - The no. of pixel levels is 256 in our examples
    - Total no. of pixels is 10000 in a 100x100 image

Move x axis levels around until the plot to the left looks like the plot to the right
For each pixel value $x$:
- Find the location on the red line that has the closest Y value to the observed CCF at $x$
Mapping CCFs

For each pixel value x:
- Find the location on the red line that has the closest Y value to the observed CCF at x

f(x1) = x2
f(x3) = x4
Etc.
Mapping CCFs

For each pixel in the image to the left
- The pixel has a value $x$
- Find the CCF at that pixel value $CCF(x)$
- Find $x'$ such that $CCF(x')$ in the plot to the right equals $CCF(x)$
  - $x'$ such that $CCF_{\text{flat}}(x') = CCF(x)$
- Modify the pixel value to $x'$

Move x axis levels around until the plot to the left looks like the plot to the right
Doing it Formulaically

\[ f(x) = \text{round} \left( \frac{CCF(x) - CCF_{\text{min}}}{N_{\text{pixels}} - CCF_{\text{min}}} \right) \text{Max. pixel. value} \]

- \( CCF_{\text{min}} \) is the smallest non-zero value of \( CCF(x) \)
  - The value of the CCF at the smallest observed pixel value
- \( N_{\text{pixels}} \) is the total no. of pixels in the image
  - 10000 for a 100x100 image
- \( \text{Max. pixel. value} \) is the highest pixel value
  - 255 for 8-bit pixel representations
Or even simpler

- Matlab:
  
  ```matlab
  Newimage = histeq(oldimage)
  ```
Histogram Equalization

- Left column: Original image
- Right column: Equalized image
- All images now have similar contrast levels
Eigenfaces after Equalization

- Left panel: Without HEQ
- Right panel: With HEQ
  - Eigen faces are more face like..
    - Need not always be the case
Detecting Faces in Images
Detecting Faces in Images

- Finding face like patterns
  - How do we find if a picture has faces in it?
  - Where are the faces?

- A simple solution:
  - Define a “typical face”
  - Find the “typical face” in the image
Finding faces in an image

- Picture is larger than the "typical face"
  - E.g. typical face is 100x100, picture is 600x800
- First convert to greyscale
  - $R + G + B$
  - Not very useful to work in color
Finding faces in an image

- Goal: To find out if and where images that look like the “typical” face occur in the picture.
Finding faces in an image

- Try to “match” the typical face to each location in the picture
Finding faces in an image

- Try to “match” the typical face to each location in the picture
Finding faces in an image

- Try to “match” the typical face to each location in the picture
Finding faces in an image

- Try to “match” the typical face to each location in the picture
Finding faces in an image

- Try to “match” the typical face to each location in the picture
Finding faces in an image

- Try to “match” the typical face to each location in the picture
Finding faces in an image

- Try to “match” the typical face to each location in the picture
Finding faces in an image

- Try to “match” the typical face to each location in the picture
Finding faces in an image

- Try to “match” the typical face to each location in the picture
Finding faces in an image

- Try to “match” the typical face to each location in the picture
- The “typical face” will explain some spots on the image much better than others
  - These are the spots at which we probably have a face!
How to “match”

- What exactly is the “match”
  - What is the match “score”

- The DOT Product
  - Express the typical face as a vector
  - Express the region of the image being evaluated as a vector
    - But first histogram equalize the region
      - Just the section being evaluated, without considering the rest of the image
  - Compute the dot product of the typical face vector and the “region” vector
What do we get

- The right panel shows the dot product at various locations
  - Redder is higher
    - The locations of peaks indicate locations of faces!

- This is a Matched Filter
What do we get

- The right panel shows the dot product at various locations
  - Redder is higher
    - The locations of peaks indicate locations of faces!
- Correctly detects all three faces
  - Likes George’s face most
    - He looks most like the typical face
- Also finds a face where there is none!
  - A false alarm
Scaling and Rotation Problems

- **Scaling**
  - Not all faces are the same size
  - Some people have bigger faces
  - The size of the face on the image changes with perspective
  - Our “typical face” only represents one of these sizes

- **Rotation**
  - The head need not always be upright!
    - Our typical face image was upright
Solution

- Create many “typical faces”
  - One for each scaling factor
  - One for each rotation
    - How will we do this?
- Match them all

- Does this work
  - Kind of .. Not well enough at all
  - We need more sophisticated models
Many more complex methods
- Use edge detectors and search for face like patterns
- Find “feature” detectors (noses, ears..) and employ them in complex neural networks..

The Viola Jones method
- Boosted cascaded classifiers

But first, what is boosting
And even before that – what is classification?

- Given “features” describing an entity, determine the category it belongs to
  - Walks on two legs, has no hair. Is this
    - A Chimpanzee
    - A Human
  - Has long hair, is 5’4” tall, is this
    - A man
    - A woman
  - Matches “eye” pattern with score 0.5, “mouth pattern” with score 0.25, “nose” pattern with score 0.1. Are we looking at
    - A face
    - Not a face?
Classification

- **Multi-class classification**
  - Many possible categories
    - E.g. Sounds “AH, IY, UW, EY..”
    - E.g. Images “Tree, dog, house, person..”

- **Binary classification**
  - Only two categories
    - Man vs. Woman
    - Face vs. not a face..

- **Face detection: Recast as binary face classification**
  - For each little square of the image, determine if the square represents a face or not
Face Detection as Classification

- Faces can be many sizes
- They can happen anywhere in the image
- For each face size
  - For each location
    - Classify a rectangular region of the face size, at that location, as a face or not a face
- This is a series of binary classification problems

For each square, run a classifier to find out if it is a face or not
Introduction to Boosting

- An *ensemble* method that sequentially combines many simple *BINARY* classifiers to construct a final complex classifier
  - Simple classifiers are often called “weak” learners
  - The complex classifiers are called “strong” learners

- Each weak learner focuses on instances where the previous classifier failed
  - Give greater weight to instances that have been incorrectly classified by previous learners

- Restrictions for weak learners
  - Better than 50% correct

- Final classifier is *weighted* sum of weak classifiers
Boosting: A very simple idea

- One can come up with many rules to classify
  - E.g. Chimpanzee vs. Human classifier:
    - If arms == long, entity is chimpanzee
    - If height > 5’6” entity is human
    - If lives in house == entity is human
    - If lives in zoo == entity is chimpanzee

- Each of them is a reasonable rule, but makes many mistakes
  - Each rule has an intrinsic error rate

- *Combine* the predictions of these rules
  - But not equally
  - Rules that are less accurate should be given lesser weight
The total confidence in all classifiers that classify the entity as a chimpanzee is

$$Score_{chimp} = \sum_{\text{classifier favors chimpanzee}} \alpha_{\text{classifier}}$$

The total confidence in all classifiers that classify it as a human is

$$Score_{human} = \sum_{\text{classifier favors human}} \alpha_{\text{classifier}}$$

If $$Score_{chimp} > Score_{human}$$ then the our belief that we have a chimpanzee is greater than the belief that we have a human.
Boosting as defined by Freund

- A gambler wants to write a program to predict winning horses. His program must encode the expertise of his brilliant winner friend.

- The friend has no single, encodable algorithm. Instead he has many rules of thumb:
  - He uses a different rule of thumb for each set of races:
    - E.g. “in this set, go with races that have black horses with stars on their foreheads”
  - But cannot really enumerate what rules of thumbs go with what sets of races: he simply “knows” when he encounters a set:
    - A common problem that faces us in many situations

- Problem:
  - How best to combine all of the friend’s rules of thumb
  - What is the best set of races to present to the friend, to extract the various rules of thumb
Boosting

- The basic idea: Can a “weak” learning algorithm that performs just slightly better than random guessing be boosted into an arbitrarily accurate “strong” learner
  - Each of the gambler’s rules may be just better than random guessing

- This is a “meta” algorithm, that poses no constraints on the form of the weak learners themselves
  - The gambler’s rules of thumb can be anything
Boosting: A Voting Perspective

- Boosting can be considered a form of voting
  - Let a number of different classifiers classify the data
  - Go with the majority
  - Intuition says that as the number of classifiers increases, the dependability of the majority vote increases

- The corresponding algorithms were called Boosting by majority
  - A (weighted) majority vote taken over all the classifiers
  - How do we compute weights for the classifiers?
  - How do we actually train the classifiers
ADA Boost: Adaptive algorithm for learning the weights

- ADA Boost: Not named of ADA Lovelace
- An *adaptive* algorithm that learns the weights of each classifier sequentially
  - Learning adapts to the current accuracy

Iteratively:
- Train a simple classifier from training data
  - It will make errors even on training data
  - Train a new classifier that focuses on the training data points that have been misclassified
- Red dots represent training data from Red class
- Blue dots represent training data from Blue class
Very simple weak learner

A line that is parallel to one of the two axes
First weak learner makes many mistakes

- Errors coloured black
- Second weak learner focuses on errors made by first learner
**Boosting: An Example**

- **Second strong learner**: weighted combination of first and second weak learners
  - Decision boundary shown by black lines
The second strong learner also makes mistakes.

Errors colored black.
Third weak learner concentrates on errors made by second strong learner
- Third weak learner concentrates on errors made by combination of previous weak learners

- Continue adding weak learners until….
Voila! Final strong learner: very few errors on the training data
Boosting: An Example

- The final strong learner has learnt a complicated decision boundary
Boosting: An Example

- The final strong learner has learnt a complicated decision boundary

- Decision boundaries in areas with low density of training points assumed inconsequential
Overall Learning Pattern

- Strong learner increasingly accurate with increasing number of weak learners

- Residual errors increasingly difficult to correct
  - Additional weak learners less and less effective

![Graph showing error rates over the number of weak learners]
ADABOOST

- Cannot just add new classifiers that work well only the previously misclassified data

- Problem: The new classifier will make errors on the points that the earlier classifiers got right
  - Not good
  - On test data we have no way of knowing which points were correctly classified by the first classifier

- Solution: Weight the data when training the second classifier
  - Use all the data but assign them weights
    - Data that are already correctly classified have less weight
    - Data that are currently incorrectly classified have more weight
ADA Boost

- The red and blue points (correctly classified) will have a weight $\alpha < 1$
- Black points (incorrectly classified) will have a weight $\beta (= 1/\alpha) > 1$
- To compute the optimal second classifier, we minimize the total weighted error
  - Each data point contributes $\alpha$ or $\beta$ to the total count of correctly and incorrectly classified points
  - E.g. if one of the red points is misclassified by the new classifier, the total error of the new classifier goes up by $\alpha$
ADA Boost

- Each new classifier modifies the weights of the data points based on the accuracy of the current classifier.
- The final classifier too is a weighted combination of all component classifiers.
Will continue next week

- Next class: Project ideas.

- Today’s lecture and the next lecture are the basis for HW 2.