Group HuBroPri

Nen Huynh, Steve Brown, Jody Pritchett

Department of Mathematics and Statistics
California State University, Long Beach

May 10, 2012

Outline

Principal Component Analysis

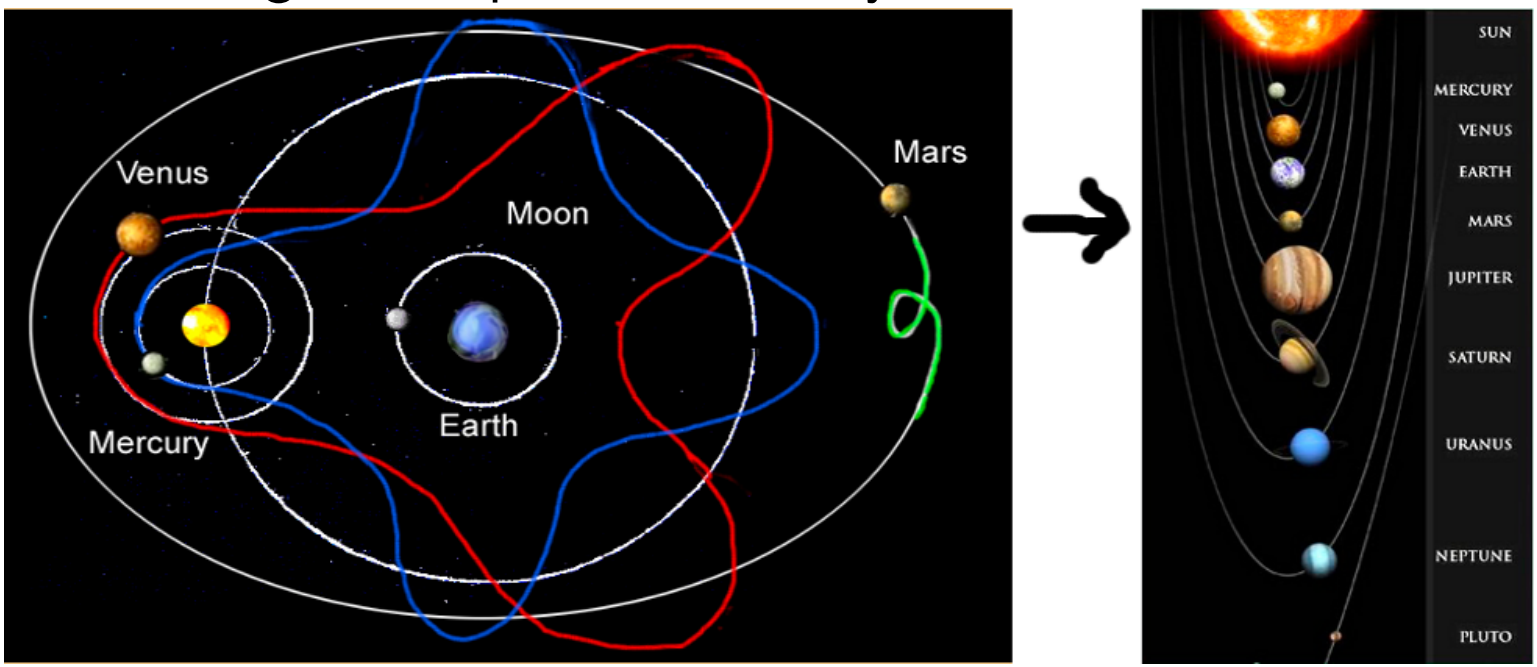
What is PCA?

- Principal component analysis (PCA) is a mathematical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components.
- The number of principal components is less than or equal to the number of original variables.
- This transformation is defined in such a way that the first principal component has the largest possible variance (that is, accounts for as much of the variability in the data as possible), and each succeeding component in turn has the highest variance possible under the constraint that it be orthogonal to (i.e., uncorrelated with) the preceding components.

(http://en.wikipedia.org/wiki/Principal_component_analysis)

Visual example of PCA

Early astronomers used a technique to simplify the motion of the solar system This led to stunning insight into the laws of motion of objects in space under gravitational forces. Of course, the technique wasn't called PCA then.
Beginning with a map of motion of planets how it looks from Earth, we get a map of the solar system as it looks from the sun.



(http://adultera.awardspace.com/RECON/PCA1.html)

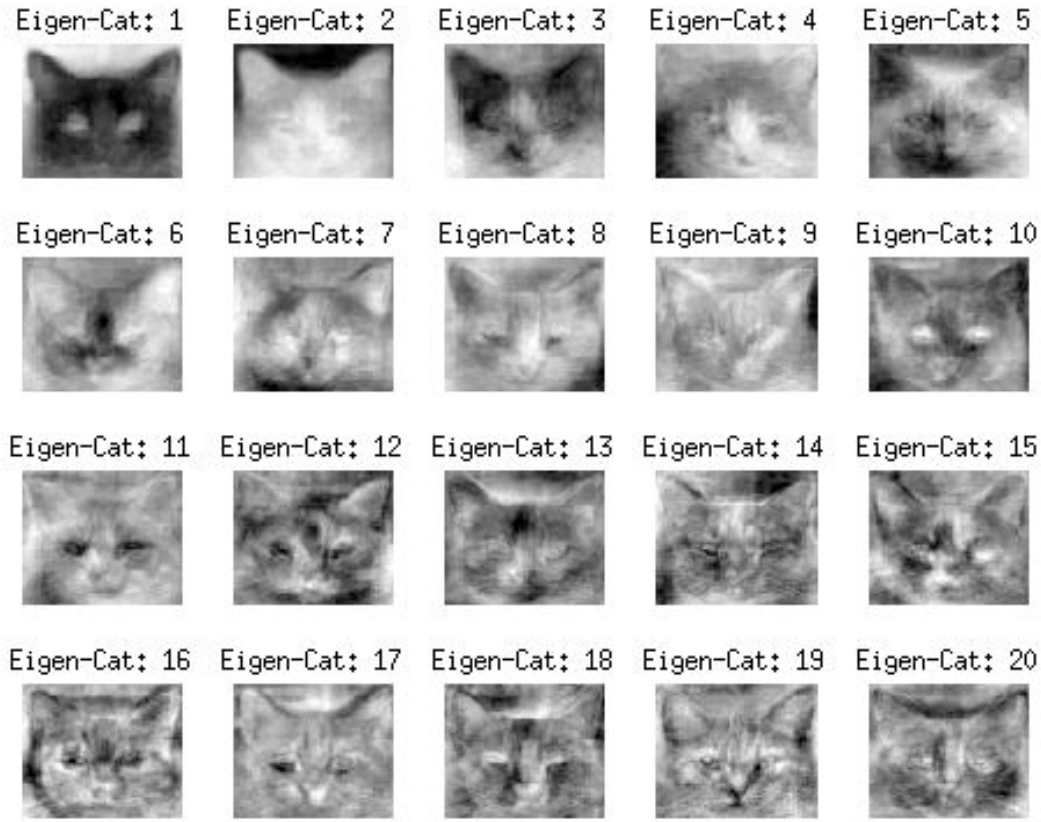
Classification of Cats and Dogs Using PCA

The procedure:

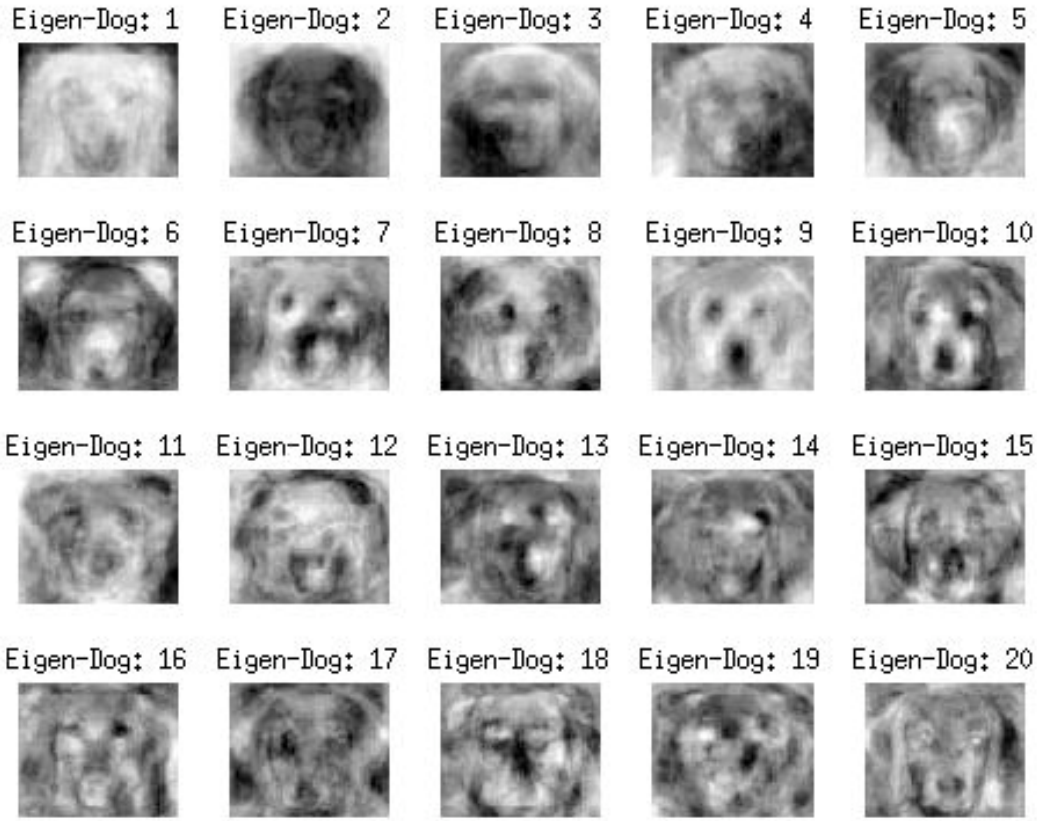
- Load ensemble of training images of cats and also of dogs.
- Mean average both ensembles.
- Call `svd` to get a training basis for cats and also for dogs.
- Retain 95% of the cumulative energy in each ensemble.
- Load ensemble of testing images of cats and also of dogs.
- Mean average both testing ensembles.
- For each test image:
  - Project test image onto training basis for cats and for dogs.
  - Take the two norm of both projections.
  - If the cat-basis norm is larger than the dog-basis norm then classify the test image as a cat, else as a dog.
- Compute success rates.

Training Set Eigen-Pets

To keep the percentage of cumulative energy in each training set, a little more than forty dimensions out of eighty were required for each basis. Here are some of the eigen-cats and eigen-dogs.

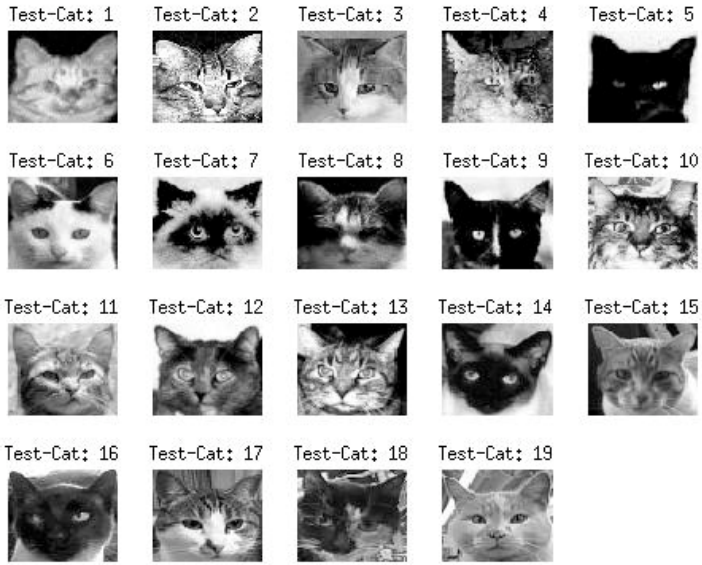


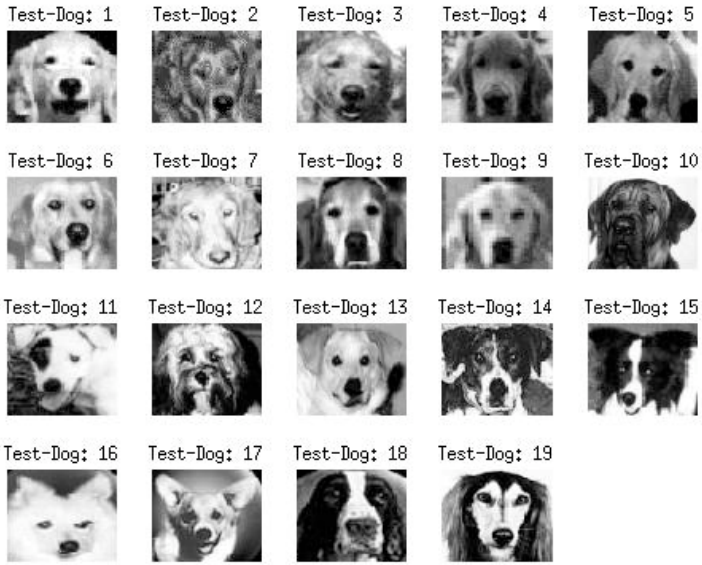
Training Set Eigen-Pets



The Test-Pets

Here are the cats and dogs to be classified.





The Result

Here is a table showing the results of the classifications.

Confusion matrix	Cats	Dogs
Classified as Cat	16	2
Classified as Dog	3	17
Total	19	19
Success Ratio	84.42%	89.47%

Overall results were good but there is room for improvement.

The Misfits

Here are the cats and dogs that were classified incorrectly.

Perhaps the two dogs were mistaken for cats because they are two of only four dogs that have perky ears while all the cats have perky ears. As for why the three cats were taken to be dogs, it's a mystery.

Learning Vector Quantization (LVQ)

Preprocessed Steps

- Applied filters (Laplacian filter or Median filter)
- Principle Component Analysis (PCA)
- Obtained numerical rank D of the matrix
  - D = 65 (median filter)
  - D = 131 (Laplacian filter)
  - D = 78 (raw data)

Self Organizing Feature Map (SOFM)

Figures

The following are SOM using 25 neurons.

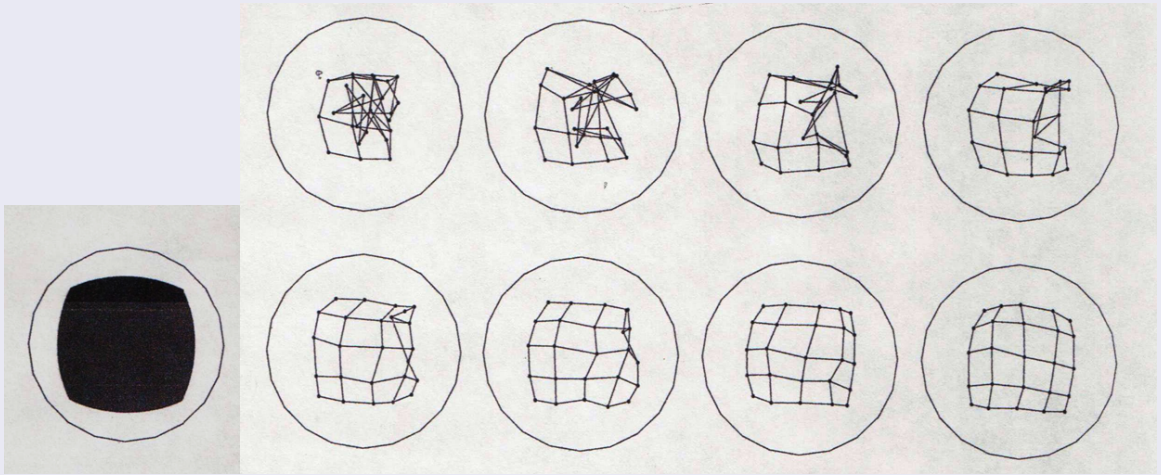


Figure: Self-organizing Map

SOFM Plot Animation

Learning Vector Quantization (LVQ)

1. Unsupervising (first layer) and supervising (second layer) learning
2. Competitive first layer (Self-Organizing Feature Map (SOFM))
   - $a^1 = compet(n^1)$
   - $n^1 = -\begin{pmatrix} \|_1 w^T - p\| \\ \|_2 w^T - p\| \\ \vdots \\ \|_s w^T - p\| \end{pmatrix}$
3. Supervising learning for second layer
   - $a^2 = w^2 a^1$
   - $w^2_{ki} = 1 \longmapsto$subclass i is a part of class k
   - $\{p_1, t_1\}, \{p_2, t_2\}, \ldots, \{p_Q, t_Q\}$

Learning Vector Quantization (LVQ)

1. Unsupervising (first layer) and supervising (second layer) learning
2. Competitive first layer (Self-Organizing Feature Map (SOFM))
   - $a^1 = compet(n^1)$
   - $n^1 = -\begin{pmatrix} \|_1 w^T - p\| \\ \|_2 w^T - p\| \\ \vdots \\ \|_s w^T - p\| \end{pmatrix}$
3. Supervising learning for second layer
   - $a^2 = w^2 a^1$
   - $w^2_{ki} = 1 \longmapsto$subclass i is a part of class k
   - $\{p_1, t_1\}, \{p_2, t_2\}, \ldots, \{p_Q, t_Q\}$

Learning Vector Quantization (LVQ)

1. Unsupervising (first layer) and supervising (second layer) learning
2. Competitive first layer (Self-Organizing Feature Map (SOFM))
   - $a^1 = compet(n^1)$
   - $n^1 = -\begin{pmatrix} \|_1 w^T - p\| \\ \|_2 w^T - p\| \\ \vdots \\ \|_s w^T - p\| \end{pmatrix}$
3. Supervising learning for second layer
   - $a^2 = w^2 a^1$
   - $w^2_{ki} = 1 \longmapsto$subclass i is a part of class k
   - $\{p_1, t_1\}, \{p_2, t_2\}, \ldots, \{p_Q, t_Q\}$

Learning Vector Quantization (LVQ)

- LVQ Network Learning with the Kohonen Rule
- ${}_iw(q) = {}_i\, w(q-1) + \alpha(p(q) - {}_i\, w(q-1))$, if $a_k^2 = t_k = 1$
- ${}_iw(q) = {}_i\, w(q-1) - \alpha(p(q) - {}_i\, w(q-1))$, if $a_k^2 \neq t_k = 0$
- where i is the index of the weight, ${}_iw(q)$ is the new weight, ${}_iw(q-1)$ is the old weight, and α is a learning rate

Figures

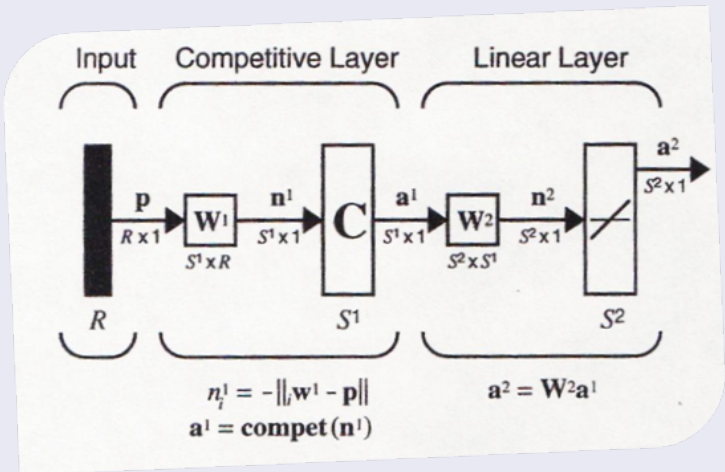


Figure: Learning Vector Quantization

Experiment/Simulation

- Apply PCA to get the KL coefficient
- Train and Test on raw data

Figures

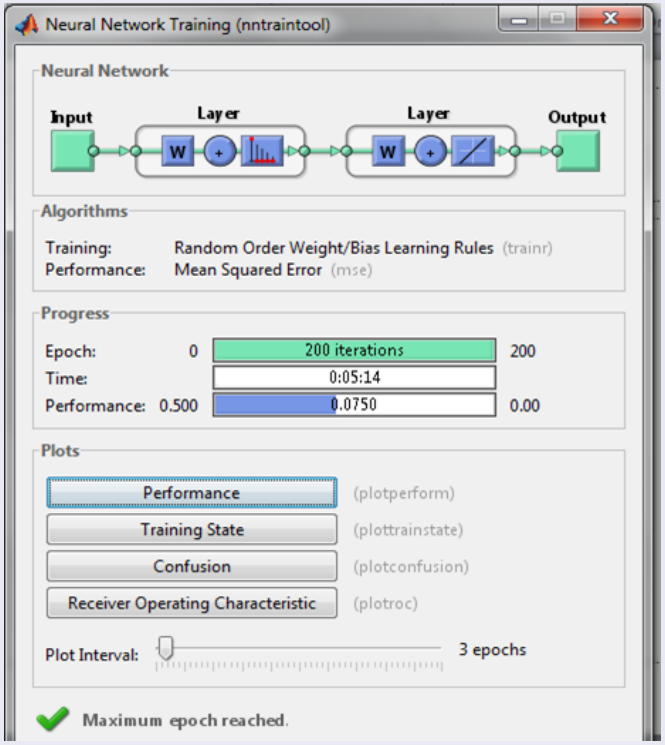


Figure: Simulation Neural Network Training Block

Experiment/Simulation

Result:

Confusion matrix	Cats	Dogs
Classified as Cat	16	5
Classified as Dog	3	14
Total	19	19

Probability form	Cats	Dogs
Classified as Cat	84.21%	26.32%
Classified as Dog	15.79%	73.68%

Experiment/Simulation

Figures

The following are misclassified images of cats and dogs.

Figure: Misclassified Images w/Raw Data

Experiment/Simulation

- Apply PCA to get the KL coefficient
- Train and Test on "median filtered" data

Confusion matrix	Cats	Dogs
Classified as Cat	17	3
Classified as Dog	2	16
Total	19	19

Probability form	Cats	Dogs
Classified as Cat	89.47%	15.19%
Classified as Dog	10.53%	84.21%

Experiment/Simulation

Figures

The following are misclassified images of cats and dogs.

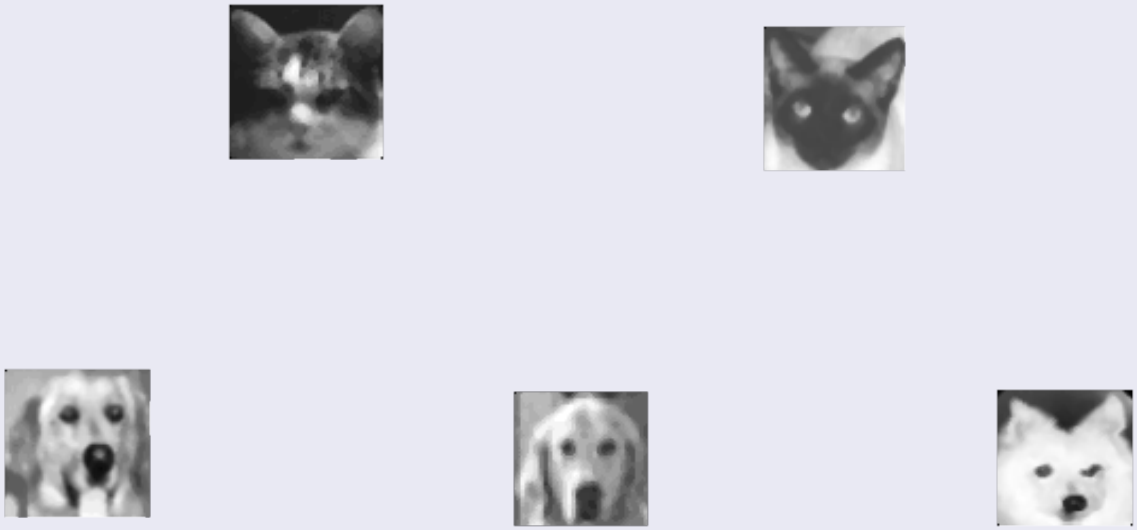
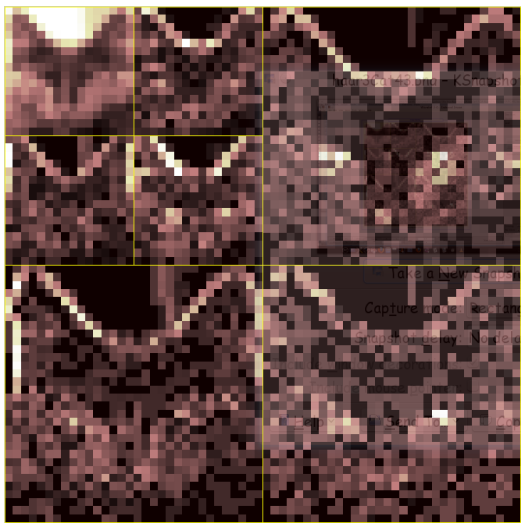


Figure: Misclassified Images w/Median Filtered Data

Wavelet PCA Novelty (WPCAN) Filter

Step 1: Wavelet transform

For each image, wavelet transform using 2 levels of discrete Haar wavelets. And then vectorize using the wavelet coefficients (no scaling coefficients).



$$\Rightarrow \begin{bmatrix} V^1 & H^1 & D^1 & V^2 & H^2 & D^2 \end{bmatrix}^T$$

Note: We do not store C^2, the top left corner because C^2 mostly contains only fur and background color.

Step 2: Use PCA to get the reduced KL basis for each of the classes.

1. Demean: $\tilde{X} = X - mean(X)$.
2. Get the KL basis using svd: $[U, \ , \ ] = svd(\tilde{X})$.
3. Reduce the number of basis elements using the stretching dimension D_δ and energy dimension D_γ with $\gamma = 0.95$ and $\delta = 0.01$: $P_C = U_C(:, 1 : D)$, $P_D = U_D(:, 1 : D)$.

Step 2: Use PCA to get the reduced KL basis for each of the classes.

1. Demean: $\tilde{X} = X - mean(X)$.
2. Get the KL basis using svd: $[U, \ , \ ] = svd(\tilde{X})$.
3. Reduce the number of basis elements using the stretching dimension D_δ and energy dimension D_γ with $\gamma = 0.95$ and $\delta = 0.01$: $P_C = U_C(:, 1:D)$, $P_D = U_D(:, 1:D)$.

Step 2: Use PCA to get the reduced KL basis for each of the classes.

1. Demean: $\tilde{X} = X - mean(X)$.
2. Get the KL basis using svd: $[U, \ , \ ] = svd(\tilde{X})$.
3. Reduce the number of basis elements using the stretching dimension D_δ and energy dimension D_γ with $\gamma = 0.95$ and $\delta = 0.01$: $P_C = U_C(:, 1:D)$, $P_D = U_D(:, 1:D)$.

Step 3: Classify the samples using the novelty filter.

1. Demean the samples S with the means of the two classes:

$$S_C = S - mean(C), S_D = S - mean(D)$$

2. Project the demeaned sample, using the reduced KL basis of each of the classes and get the residue:

$$R_C = S_C - P_C P_C^T S_C, R_D = S_D - P_D P_D^T S_D$$

3. If $\|R_C\|_2 < \|R_D\|_2$ then it is a cat and if $\|R_C\|_2 > \|R_D\|_2$ then it is a dog.

Step 3: Classify the samples using the novelty filter.

1. Demean the samples S with the means of the two classes:

$$S_C = S - mean(C), S_D = S - mean(D)$$

2. Project the demeaned sample, using the reduced KL basis of each of the classes and get the residue:

$$R_C = S_C - P_C P_C^T S_C, R_D = S_D - P_D P_D^T S_D$$

3. If $\|R_C\|_2 < \|R_D\|_2$ then it is a cat and if $\|R_C\|_2 > \|R_D\|_2$ then it is a dog.

Step 3: Classify the samples using the novelty filter.

1. Demean the samples S with the means of the two classes:

$$S_C = S - mean(C), S_D = S - mean(D)$$

2. Project the demeaned sample, using the reduced KL basis of each of the classes and get the residue:

$$R_C = S_C - P_C P_C^T S_C, R_D = S_D - P_D P_D^T S_D$$

3. If $\|R_C\|_2 < \|R_D\|_2$ then it is a cat and if $\|R_C\|_2 > \|R_D\|_2$ then it is a dog.

So what's the result?

Withholding 10% of the data and using the other 90% to classify them, we get on average:

Confusion matrix	Cats	Dogs
Classified as Cat	97.3%	6.7%
Classified as Dog	2.7%	93.3%

This is done $N = 10,000$ times to calculate the average. From the secret 38 images,

Confusion matrix	Cats	Dogs
Classified as Cat	19	3
Classified as Dog	0	16
Total	19	19
Success Ratio	100%	84.21%

So what's the result?

(a) (b) (c)

Figure: The dog images that were misclassified as cats.

Notice the ears. In fact, the pointing of ears upward directly matches exactly with misclassified.

Analysis

Why does it work?

1. The novelty filter is best when the images have all the images have the same foreground and background colors (i.e. binary images).
2. PCA "groups" pixels that correlate in intensity so given edges of an object, the edge pixels are related.
3. Haar wavelets (with it's weak compression of edges) gives you a way to get the edges with multiple scales.

Analysis

Why does it work?

1. The novelty filter is best when the images have all the images have the same foreground and background colors (i.e. binary images).
2. PCA "groups" pixels that correlate in intensity so given edges of an object, the edge pixels are related.
3. Haar wavelets (with it's weak compression of edges) gives you a way to get the edges with multiple scales.

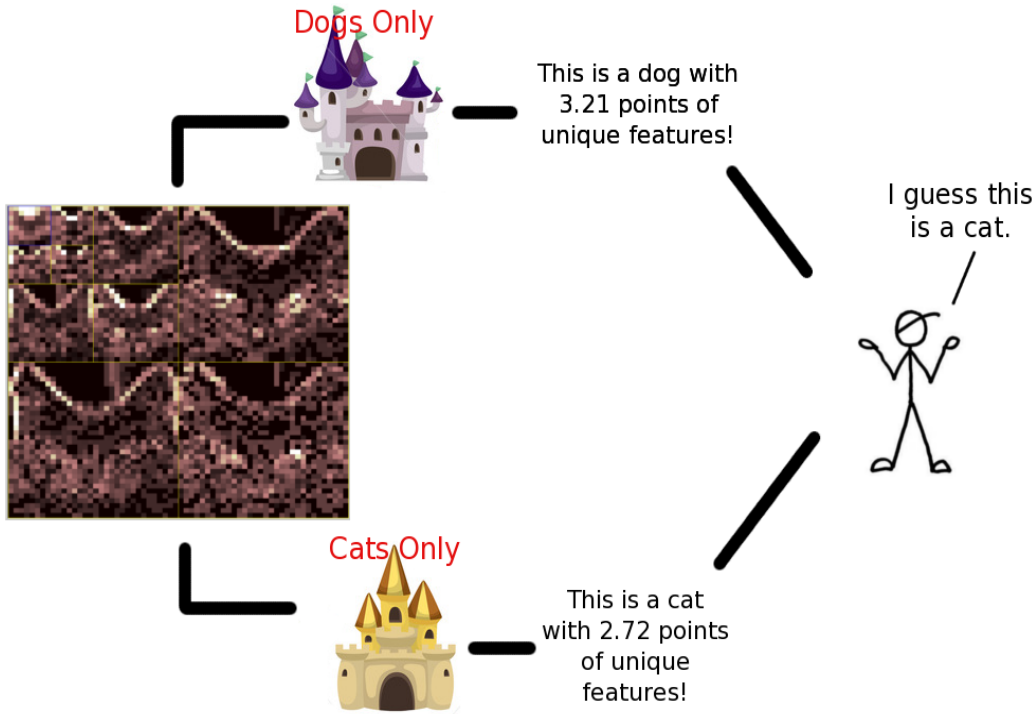
Analysis

Why does it work?

1. The novelty filter is best when the images have all the images have the same foreground and background colors (i.e. binary images).
2. PCA "groups" pixels that correlate in intensity so given edges of an object, the edge pixels are related.
3. Haar wavelets (with it's weak compression of edges) gives you a way to get the edges with multiple scales.

Analysis

But here's something strange. This process is like this:



Notice that no one in either of the castles has ever seen both a cat AND a dog and we task a person who's never seen either creature to judge whether it's a cat or dog based on information from the two castles.

Improvement

But it works so well! Still, we can give the person at the end an extra duty to improve the quality. Instead of finding

$$\min\{\|R_C\|, \|R_D\|\},$$

we should scale them by how well or how poor each of the castles did with the training sets:

$$\min\{t\|R_C\|, (1-t)\|R_D\|\} \text{ for some } t \in [0,1]$$

That is, the person at the end chooses the best scaling t to minimize the misclassification of the training data. This t is then used during classification.

Improvement

Solving for t,

$$\text{We want } t < \frac{\|C_k - P_D P_D^T C_k\|}{\|C_k - P_C P_C^T C_k\| + \|C_k - P_D P_D^T C_k\|} =: t_k^C$$

meaning t small enough that as much cats from the training sets $\{C_k\}$ are identified as cats.

$$\text{We want } t > \frac{\|D_k - P_D P_D^T D_k\|}{\|D_k - P_C P_C^T D_k\| + \|D_k - P_D P_D^T D_k\|} =: t_k^D$$

meaning t large enough that as much dogs from the training sets $\{D_k\}$ are identified as dogs.

Improvement

So we want to minimize the number of misclassifications,

$$M(t) = \sum_{k=1}^{m} \chi_{(-\infty, t_k^C]}(t) + \sum_{k=1}^{n} \chi_{[t_k^D, \infty)}(t).$$

This reduces to plugging values from the finite set $\{t_k^C\} \cup \{t_k^D\} \cup \{\frac{t_k^C + t_k^D}{2} | 1 \leq i \leq m, 1 \leq j \leq n\}$ to find the minimization $M(t)$.
We note that without the improvement to find the best t, the original version before is equivalent to when $t = 1/2$.

Improvement...

So what is the result? ...It's the same result as without the improvement. That is, we have $t = 1/2$ for our simulated classifications. And we currently do not have an explanation for this.

Confusion matrix	Cats	Dogs
Classified as Cat	97.3%	6.7%
Classified as Dog	2.7%	93.3%

This is anticlimactic but it's interesting that somehow two castles with no individual knowledge of the whole training sets can produce good results and our attempt to mitigate that did nothing.

References

Hagan et al, "Neural Network Design", Coloraldo University Bookstore, 2003

Jen-Mei Chang, "Matrix Methods for Geometric Data Analysis and Pattern Recognition", 2003

Questions?