Lecture 19
The SVD and Linear Discriminant Analysis

Jason J. Bramburger

In the previous lecture we used the discrete wavelet transform (DWT) to gain information about the edges in our pictures of dogs and cats. Our goal in this lecture is to come up with a statistical method to tell the difference between dog edges and cat edges. We will use principal component analysis (PCA) because it will tell us something about the ways in which we have the most variation in our data. It will help us to pick out some of the most important features of our data.

Let's start by loading in our data. We can use the function dc_wavelet we created in the last lecture to perform the wavelet transforms.

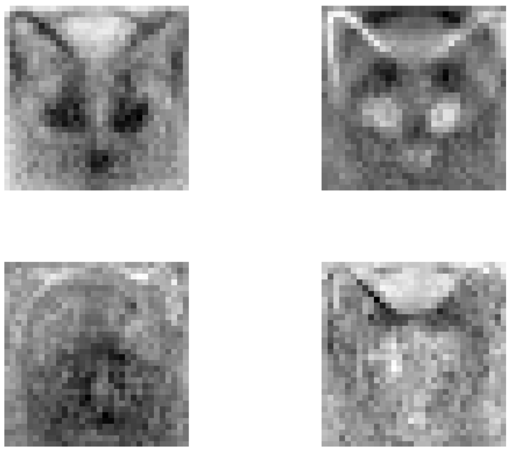
```
load('catData.mat')
load('dogData.mat')

dog_wave = dc_wavelet(dog);
cat_wave = dc_wavelet(cat);
```

We can start by applying the SVD to get our principal components. We will combine our dog and cat data into one big matrix.

```
[U,S,V] = svd([dog_wave cat_wave],'econ');
```

To get an idea of what we are working with, let's plot the first four principal components.

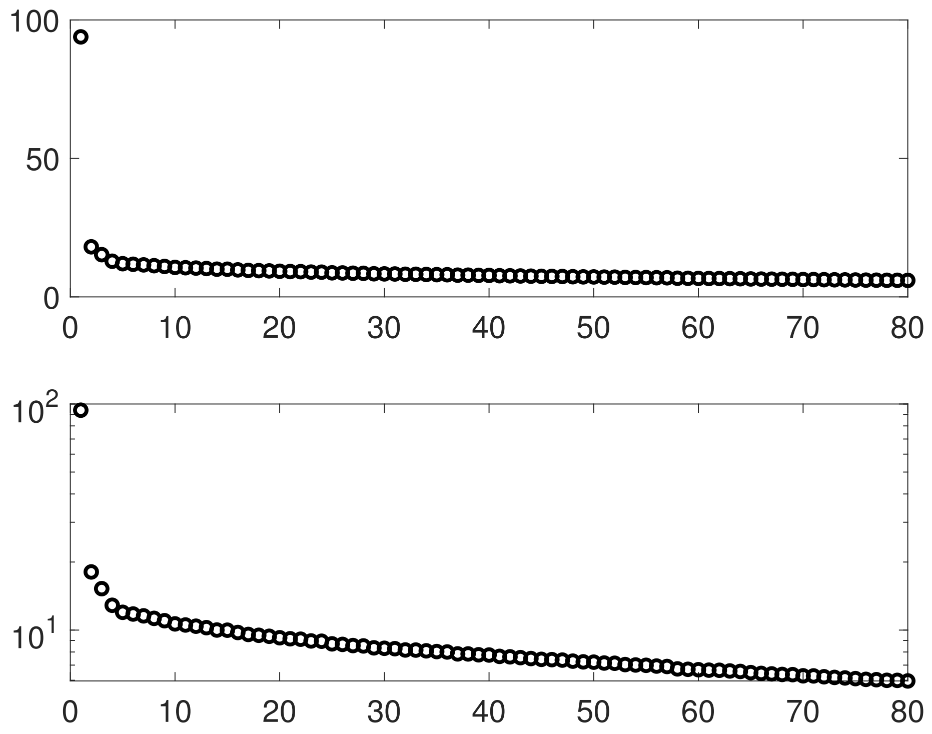
```
for k = 1:4
   subplot(2,2,k)
   ut1 = reshape(U(:,k),32,32);
   ut2 = rescale(ut1);
   imshow(ut2)
end
```



Take a minute to look over these modes. What do you notice about them? Particularly the first two.

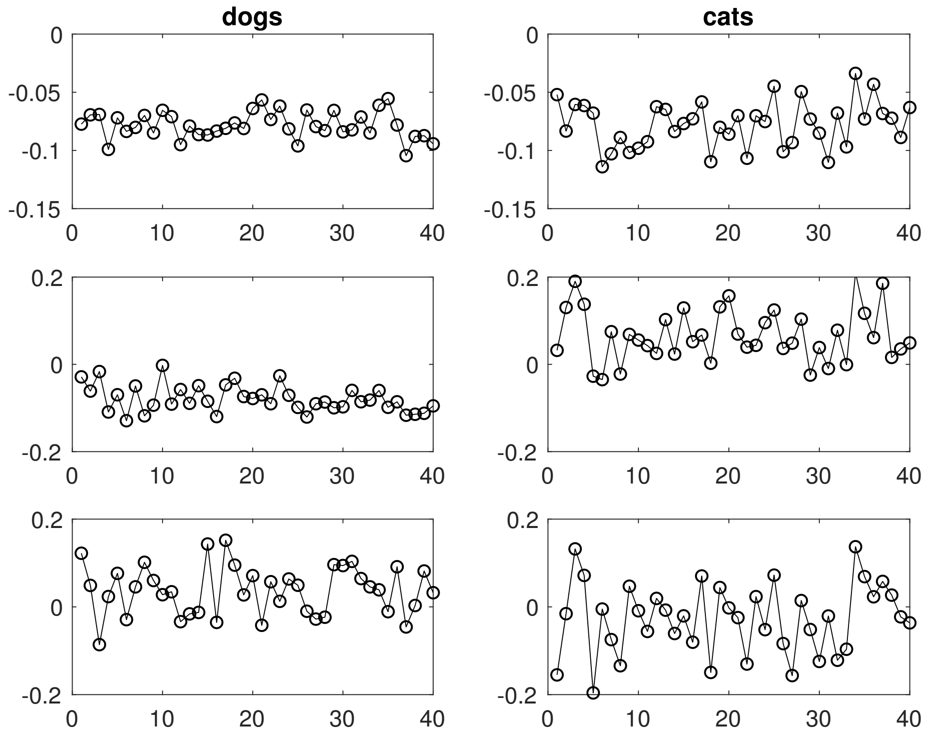
The eyes are prevalent in the first two modes. But, most importantly we can see an emphasis on the pointy triangular ears. This is one of the main features we are going to use to differentiate between dogs and cats. Let's look at the singular values to see how much weight each principal component carries.

```
figure(2)
subplot(2,1,1)
plot(diag(S),'ko','Linewidth',2)
set(gca,'Fontsize',16,'Xlim',[0 80])
subplot(2,1,2)
semilogy(diag(S),'ko','Linewidth',2)
set(gca,'Fontsize',16,'Xlim',[0 80])
```



It is clear that the first mode is dominant. But, we can also see that there isn't a sharp drop-off in the singular values. That is, we have a heavy-tail distribution, meaning there is still information in the later modes. Now that we have looked at the principal components (the left singular vectors) and the singular values, let's look at the right singular vectors in V. These vectors will show how each of the dog/cat images is represented in the PCA basis. We will just plot the first three columns (projections onto the first 3 PCA modes). We will also split the vectors into the first 80 rows for the dogs and the last 80 rows for the cats.

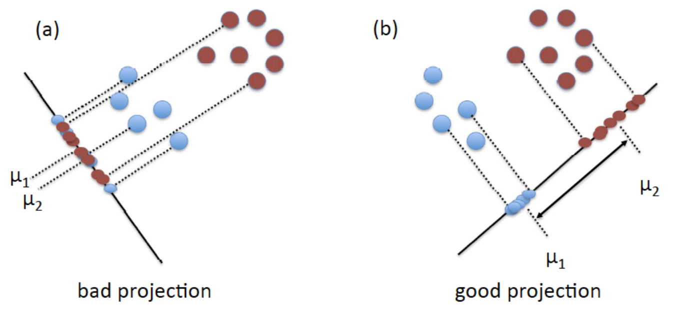
```
figure(3)
for k = 1:3
   subplot(3,2,2*k-1)
   plot(1:40,V(1:40,k),'ko-')
   subplot(3,2,2*k)
   plot(81:120,V(81:120,k),'ko-')
end
subplot(3,2,1), set(gca,'Ylim',[-.15 0],'Fontsize',12), title('dogs')
subplot(3,2,2), set(gca,'Ylim',[-.15 0],'Fontsize',12), title('cats')
subplot(3,2,3), set(gca,'Ylim',[-.2 .2],'Fontsize',12)
subplot(3,2,4), set(gca,'Ylim',[-.2 .2],'Fontsize',12)
subplot(3,2,5), set(gca,'Ylim',[-.2 .2],'Fontsize',12)
subplot(3,2,6), set(gca,'Ylim',[-.2 .2],'Fontsize',12)
```



On inspection, the first and third modes are not that useful for differentiating between dogs and cats. The second one is though. The dogs are almost all negative, while the cats have opposite sign. Recall that modes 1 and 2 both had very pointy ears, but the ears in mode 1 are black while the ears in mode 2 are white. Loosely, we can understand what we are seeing in the following way. The first and second dog modes above are both negative, thus 'flipping' the black and white cells in the first two principal components. Hence, the first two principal components for dogs have black ears added to white ears, thus cancelling each other out. In the case of cats, we have opposite signs in the first two right singular vectors above, meaning that when we multiply them by the respective principal components and add them together, the ears will both be white. Therefore, the cats have accentuated ears (in white), while the dogs don't have pointy ears since their 'ear' modes cancel each other out.

Linear Discriminant Analysis (LDA)

Now we have a rough understanding of how the computer is going to tell the difference between dogs and cats. Let's implement this now. In order to do so, we need to introduce something new: Linear Discriminant Analysis (LDA). The figure below illustrates the idea of LDA. In our example, two data sets (cats and dogs) are considered



and projected onto new bases. On the left (a), the projection shows the data to be completely mixed, not allowing for any reasonable way to separate the data from each other. On the right (b), we can see the ideal caricature for LDA since the data is completely separated. In particular, the means μ_1 and μ_2 of the two data sets are well apart when projected onto the chosen subspace.

<u>**Goal of LDA**</u>: *Find a suitable projection that maximizes the distance between the inter-class data while minimizing the intra-class data.*

Implementing LDA for 2 Datasets

The obvious question is now: how do we find the right subspace to project onto? First, we calculate the means for each of our groups for each feature. As above, we will call these μ_1 and μ_2. Note that these μ are column vectors since they are means across each row (see the code below if you're confused). We can then define the **between-class scatter matrix**

$$\mathbf{S}_B = (\mu_2 - \mu_1)(\mu_2 - \mu_1)^T.$$

This is a measure of the variance between the groups (between the means). Then we can define the **within-class scatter matrix**

$$\mathbf{S}_w = \sum_{j=1}^{2} \sum_{\mathbf{x}} (\mathbf{x} - \mu_j)(\mathbf{x} - \mu_j)^T.$$

This is a measure of the variance within each group. The goal is then to find a vector $\mathbf{w}$ such that

$$\mathbf{w} = \text{argmax} \frac{\mathbf{w}^T \mathbf{S}_B \mathbf{w}}{\mathbf{w}^T \mathbf{S}_w \mathbf{w}}.$$

This is a tough problem to solve, but luckily for us it turns out (we won't prove this) that the vector $\mathbf{w}$ that maximizes the above quotient is the eigenvector corresponding to the largest eigenvalue of the generalized eigenvalue problem

$$\mathbf{S}_B \mathbf{w} = \lambda \mathbf{S}_w \mathbf{w}.$$

This is something we can actually solve easily with MATLAB! In particular, we can just use the eig command. Once we have $\mathbf{w}$ using the above operations, you can choose a cutoff between dogs and cats to be used as a threshold for decision making.

LDA for More Groups

In the future it might be possible that you will want to classify between more than two groups. All is good - we can handle that too! You can make the following changes to the scatter matrices:

$$\mathbf{S}_B = \sum_{j=1}^{N} (\mu_j - \mu)(\mu_j - \mu)^T$$

where μ is the overall mean and μ_j is the mean of each of the $N \geq 3$ groups/classes. The within-class scatter matrix is

$$\mathbf{S}_w = \sum_{j=1}^{N} \sum_{\mathbf{x}} (\mathbf{x} - \mu_j)(\mathbf{x} - \mu_j)^T.$$

Then, $\mathbf{w}$ is found as it was above.

LDA in MATLAB

Now that we have some theory, let's put it into practice. We will start by taking advantage of the SVD that we already computed by projecting onto some number of PCA modes. We often refer to these modes as 'features'. We will have a variable set of features, which we take to be 20 for demonstration.

```matlab
feature = 20;

nd = size(dog_wave,2);
nc = size(cat_wave,2);
animals = S*V'; % projection onto principal components: X = USV' --> U'X = SV'
dogs = animals(1:feature,1:nd);
cats = animals(1:feature,nd+1:nd+nc);
```

Now let's calculate the scatter matrices.

```
md = mean(dogs,2);
mc = mean(cats,2);

Sw = 0; % within class variances
for k = 1:nd
    Sw = Sw + (dogs(:,k) - md)*(dogs(:,k) - md)';
end
for k = 1:nc
   Sw =  Sw + (cats(:,k) - mc)*(cats(:,k) - mc)';
end

Sb = (md-mc)*(md-mc)'; % between class
```

We can find **w** using the MATLAB eig() command with two arguments.

```
[V2, D] = eig(Sb,Sw); % linear disciminant analysis
[lambda, ind] = max(abs(diag(D)));
w = V2(:,ind);
w = w/norm(w,2);
```

To project onto **w** just multiply by **w**′.

```
vdog = w'*dogs;
vcat = w'*cats;
```

Now we need to set a threshold for our classifier. Later on, it will make it easier for us to make decision if we have consistency for whether dogs are above or below the threshold (there is some ambiguity with the SVD). So we make it so dogs are always below.

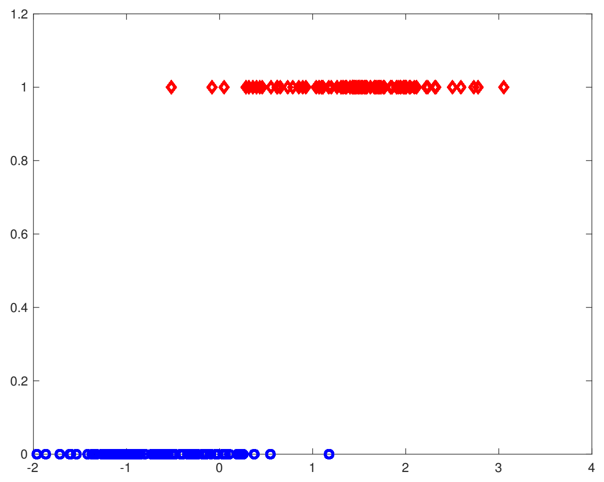
```
if mean(vdog) > mean(vcat)
    w = -w;
    vdog = -vdog;
    vcat = -vcat;
end
```

Let's see what we have!

```
figure(4)
plot(vdog,zeros(80),'ob','Linewidth',2)
hold on
plot(vcat,ones(80),'dr','Linewidth',2)
ylim([0 1.2])
```

We can see that there is a bit of an overlap between the dog and cat values. Therefore, no matter what we choose for a cutoff/threshold we aren't going to be able to perfectly distinguish between dogs and cats. That's okay! We are going to try to pick a threshold so that we get (approximately) the same number wrong for dogs and cats. So, we can start with the largest dog value and smallest cat value. If the cat value is less than the dog value, we move in one on each. We continue until the dog value is less than the cat value and we set our threshold between those values (the midpoint).

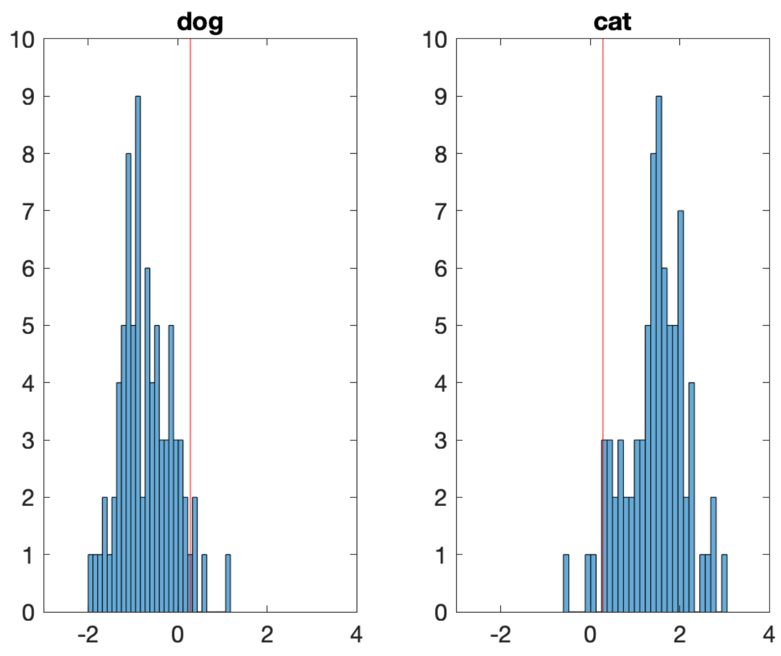
```
sortdog = sort(vdog);
sortcat = sort(vcat);

t1 = length(sortdog);
t2 = 1;
while sortdog(t1) > sortcat(t2)
```



```
    t1 = t1 - 1;
    t2 = t2 + 1;
end
threshold = (sortdog(t1) + sortcat(t2))/2;
```

Let's see how we did. We can plot histograms of the dog and cat values and draw a vertical line for the decision-making threshold.

```
figure(5)
subplot(1,2,1)
histogram(sortdog,30); hold on, plot([threshold threshold], [0 10],'r')
set(gca,'Xlim',[-3 4],'Ylim',[0 10],'Fontsize',14)
title('dog')
subplot(1,2,2)
histogram(sortcat,30); hold on, plot([threshold threshold], [0 10],'r')
set(gca,'Xlim',[-3 4],'Ylim',[0 10],'Fontsize',14)
title('cat')
```



We did pretty well! We got about 3 or 4 wrong out of 80. To finish, we are going to take what is above and put it into a function. Just in case we might need them, we will output the SVD, the threshold, the projection vector **w**, and the sorted projections. The inputs will be the dog and cat wavelets and the number of features we want to use.

```
function [U,S,V,threshold,w,sortdog,sortcat] = dc_trainer(dog_wave,cat_wave,feature)

    nd = size(dog_wave,2);
    nc = size(cat_wave,2);
    [U,S,V] = svd([dog_wave cat_wave],'econ');
    animals = S*V';
    U = U(:,1:feature); % Add this in
    dogs = animals(1:feature,1:nd);
    cats = animals(1:feature,nd+1:nd+nc);
    md = mean(dogs,2);
    mc = mean(cats,2);

    Sw = 0;
    for k=1:nd
        Sw = Sw + (dogs(:,k)-md)*(dogs(:,k)-md)';
    end
    for k=1:nc
        Sw = Sw + (cats(:,k)-mc)*(cats(:,k)-mc)';
    end
    Sb = (md-mc)*(md-mc)';

    [V2,D] = eig(Sb,Sw);
    [lambda,ind] = max(abs(diag(D)));
    w = V2(:,ind);
    w = w/norm(w,2);
    vdog = w'*dogs;
    vcat = w'*cats;

    if mean(vdog)>mean(vcat)
        w = -w;
        vdog = -vdog;
        vcat = -vcat;
    end

    % Don't need plotting here
    sortdog = sort(vdog);
    sortcat = sort(vcat);
    t1 = length(sortdog);
    t2 = 1;
    while sortdog(t1)>sortcat(t2)
    t1 = t1-1;
    t2 = t2+1;
    end
    threshold = (sortdog(t1)+sortcat(t2))/2;

    % We don't need to plot results
end
```

For running tests, we can now use the following command. Remember that you need to input the number of features to use and we can use the our dc_wavelet function to produce dog_wave and cat_wave.

```
[U,S,V,threshold,w,sortdog,sortcat] = dc_trainer(dog_wave,cat_wave,feature)
```