CPSC 340: Assignment2

Yiwei Hou(84435156)
Qiong Zhang(85896158)

1 K-Nearest Neighbours

1.1 KNN Prediction

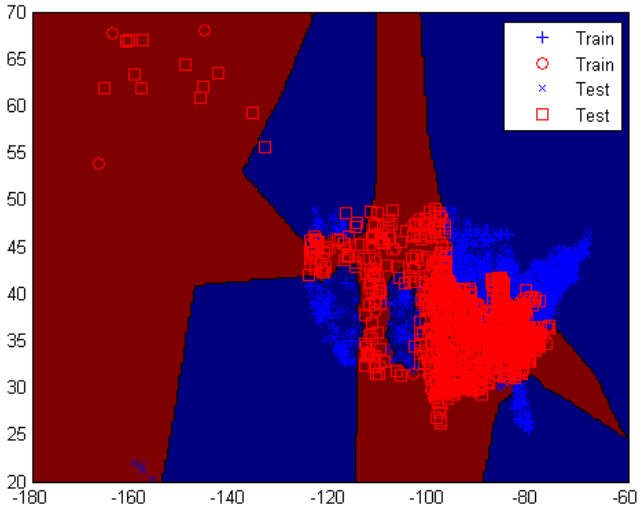
1. The code for prediction is given as follows:

```
function [yhat] = predict(model,Xtest)
[n,d]=size(model.X);
[t,d]=size(Xtest);
D=(model.X).^2*ones(d,t)+ones(n,d)*(Xtest').^2-2*(model.X)*Xtest';
[~,pos]=sort(D,1);
if model.k==1
    yhat=model.y(pos(1:model.k,:));
else
    yhat=mode(model.y(pos(1:model.k,:)),1);
end
end
```

2. The training and test error obtained on *citiesSmall.mat* dataset with different k values is given by the table below:

k	1	3	10
Training error	0	0.03	0.07
Test error	0.06	0.07	0.1

3. The plot generated by classifier2Dplot when $k = 1$ is given by:



4. When $k = 1$, the classification is done by finding out the label of the object that is closest to the object which we want to classify. The closest point to a point in training set is itself, thus by KNN, we classify all the objects in training set correctly.

5. We can use cross validation for training dataset to find out a k. The k should be choosen as the one that minimizes error.

1.2 Condensed Nearest Neighbours

1. The code for *cnn.m*:

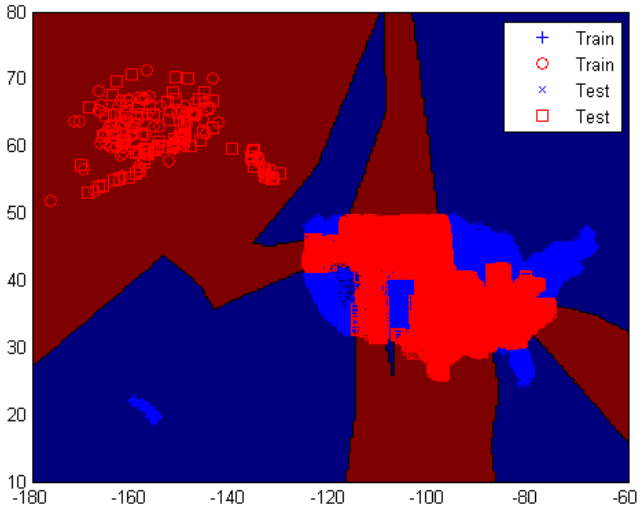
```
function [model] = cnn(X,y,k)
% [model] = cnn(X,y,k)
%
% condensed nearest neighbour
[n d]=size(X);
Xsubset=X(1,:);
ysubset=y(1);
for i=2:n;
    model=knn(Xsubset,ysubset,k);
    yhat=model.predict(model,X(i,:));
    if yhat~=y(i);
        Xsubset=[Xsubset;X(i,:)];
        ysubset=[ysubset;y(i)];
    end
end

model.X = Xsubset;
model.y = ysubset;
model.k = k;
model.c = max(y);
model.predict = @predict;
end

function [yhat] = predict(model,Xtest)
% Write me!
[n,d]=size(model.X);
[t,d]=size(Xtest);
D=(model.X).^2*ones(d,t)+ones(n,d)*(Xtest').^2-2*(model.X)*Xtest';
[~,pos]=sort(D,1);
if model.k==1
    yhat=model.y(pos(1:model.k,:));
else
    yhat=mode(model.y(pos(1:model.k,:)),1)';
end
end
```

2. For dataset *citiesBig1.mat* with $k = 1$, the training and testing errors are 0.01 and 0.02 respectively. The number of objects in subset is 457.

3. The plot generated by classifier2Dplot for *citiesBig1.mat* when $k = 1$ is given by:



4. In getting the subset of the training set, we go through the data by order. When a new example comes in, those examples who are classified correctly by a former classifier might be classified wrongly by the classifier built on the current subset. We are not using the whole data set when making predictions. Thus, the training error is greater than 0.

5. It costs $O(sd)$ to predict one test object, since we have t test objects, the total cost for prediction will be $O(std)$.

6. For dataset *citiesBig2.mat* with $k = 1$, the training and testing errors are 0.14 and 0.21 respectively. The number of subjects in subset is 30. The high error is caused by the small size of the subset of the training set. The final classifier is built on the training set of size 30 compared to a training set of size 457 in question 2, both training and test errors increase. It might be because the order of the data set influences the size of the subset that we get from going through each data once.

2 Random Forests

2.1 Random Trees

1. The training error and test error for the *decisionTree* model based on *vowel.mat* is given as follows(Figure 1):

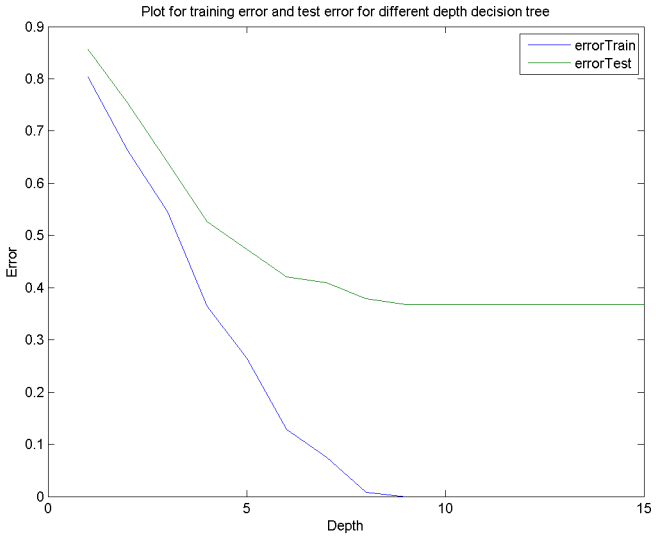


Figure 1: Training and test error for decisionTree model

2. The training set have 264 objects, a decision tree with depth 9 has $2^9 - 1 = 512$ nodes which is larger than the size of the training set. The decision tree stop growing after the depth larger than 9 since there's no more node to split.

3. New training function is:

```
function [model] = randomStump(X,y)
% [model] = randomStump(X,y)
%
% Fits a decision stump that splits on a single variable.

% Choose random features.
[n,d] = size(X);
m=floor(sqrt(d));
Xrandomfeature=zeros(n,m);
for i=1:m;
    j=ceil(rand*d);
    Xrandomfeature(:,i)=X(:,j);
end


% Computer number of class lables
k = max(y);

% Address the trivial case where we do not split
count = accumarray(y,ones(size(y)),[k 1]); % Counts the number of occurrences
    of each class
[maxCount,maxLabel] = max(count);

% Compute total entropy (needed for information gain)
p = count/sum(count); % Convert counts to probabilities
entropyTotal = -sum(p.*log0(p));

```

```
maxGain = 0;
splitVariable = [];
splitThreshold = [];
splitLabel0 = maxLabel;
splitLabel1 = [];

% Loop over features looking for the best split
if any(y ~= y(1))
    for j = 1:m
        thresholds = sort(unique(Xrandomfeature(:,j)));

        for t = thresholds'

            % Count number of class labels where the feature is greater than
                threshold
            yVals = y(X(:,j) > t);
            count1 = accumarray(yVals,ones(size(yVals)),[k 1]);
            count0 = count-count1;

            % Compute infogain
            p1 = count1/sum(count1);
            p0 = count0/sum(count0);
            H1 = -sum(p1.*log0(p1));
            H0 = -sum(p0.*log0(p0));
            prob1 = sum(X(:,j) > t)/n;
            prob0 = 1-prob1;
            infoGain = entropyTotal - prob1*H1 - prob0*H0;

            % Compare to minimum error so far
            if infoGain > maxGain
                % This is the lowest error, store this value
                maxGain = infoGain;
                splitVariable = j;
                splitThreshold = t;

                % Compute majority class
                [maxCount,splitLabel1] = max(count1);
                [maxCount,splitLabel0] = max(count0);
            end
        end
    end
end
model.splitVariable = splitVariable;
model.splitThreshold = splitThreshold;
model.label1 = splitLabel1;
model.label0 = splitLabel0;
model.predict = @predict;
end

function [y] = predict(model,X)
[t,d] = size(X);

if isempty(model.splitVariable)
    y = model.label0*ones(t,1);
else
```

```
81      y = zeros(t,1);
82      for n = 1:t
83          if X(n,model.splitVariable) > model.splitThreshold
84              y(n,1) = model.label1;
85          else
86              y(n,1) = model.label0;
87          end
88      end
89  end
90  end
```

When *randomStump* only considers $\lfloor\sqrt{d}\rfloor$ randomly-chosen features. One run of the method gives the training error and test error as follows(Figure 2):

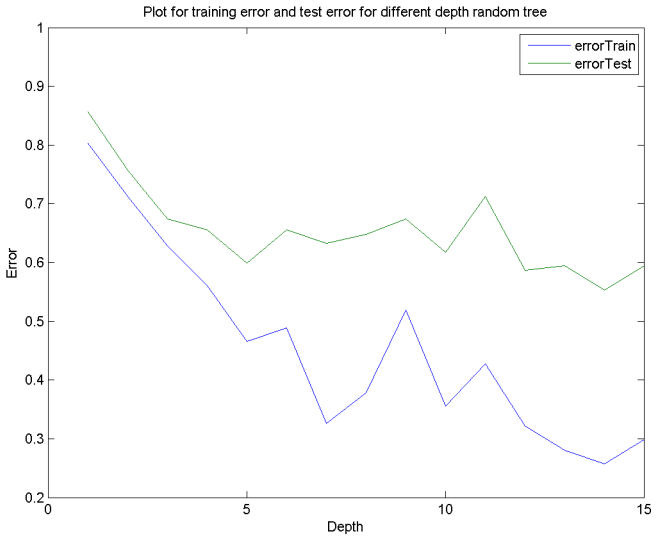


Figure 2: Training and test error for randomTree model

4. For each depth, train *decisionTree* on a different bootstrap sample, then the training error and test error become(Figure 3):

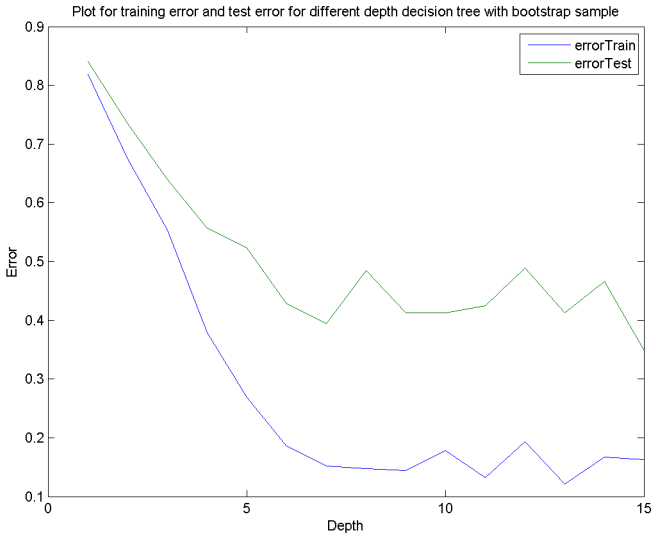


Figure 3: Training and test error of a decisionTree trained on a bootstrap sample

2.2 Random Decision Forests

When set depth equals 1000, one run gives the output as follows:

1. The test error of the *decisionForest* classifier with a depth of ∞ and 50 trees is **0.3674**.

2. When each tree is trained on a different bootstrap sample, the test error of the *decisionForest* classifier with a depth of ∞ and 50 trees becomes **0.2652**.

3. When repeatedly calls *randomTree* instead of *decisionTree* on the original dataset, the test error of the *decisionForest* classifier with a depth of ∞ and 50 trees becomes **0.4091**.

4. When use *randomTree* and train each tree on a different bootstrap sample, the test error becomes **0.4053**. The code for *decisionForest.m*:

```
function [model] = decisionForestmodified(X,y,depth,nBootstraps)
[n,d]=size(X);

% Fit model to each boostrap sample of data
for m = 1:nBootstraps
    for i=1:n;
        j=ceil(rand*n);
        Xbootstrap(i,:)=X(j,:);
        ybootstrap(i)=y(j);
    end
    model.subModel{m} = randomTree(Xbootstrap,ybootstrap',depth);
end

model.predict = @predict;

end

function [y] = predict(model,X)
```

```
19
20 % Predict using each model
21 for m = 1:length(model.subModel)
22     y(:,m) = model.subModel{m}.predict(model.subModel{m},X);
23 end
24
25 % Take the most common label
26 y = mode(y,2);
27 end
```

5. Bootstrapping can decrease the test error as it allows visits to different samples. Random split increases the test error.

3 K-Means Clustering

3.1 Selecting among Initializations

1. The code for *error* is given as:

```
1 function [error] = error(model,X)
2 [n,d] = size(X);
3 W = model.W;
4 y = model.y;
5 k = size(W,1);
6
7 % Compute the sum of squared distances between samples and their means
8 X2 = X.^2*ones(d,k);
9 distance2 = X2 + ones(n,d)*(W').^2 - 2*X*W';
10 error=0;
11 for i=1:n;
12     error=error+distance2(i,y(i));
13 end
14 end
```

2. The plot of clustering that minimizes the objective function by running k-means 50 times is give by(Figure 4):

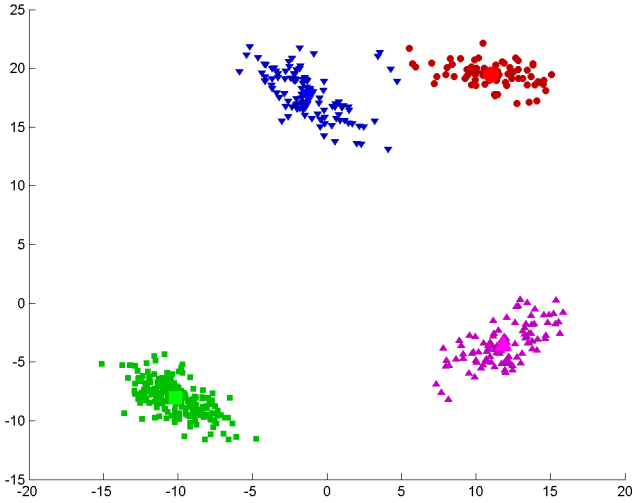


Figure 4: K-Means Clustering with the minimum error over 50 runs when $k = 4$

3.2 Selecting k

1. When k equals the number of objects in the training set, the objective function is minimized and equals to 0. This choice of k does not make any sense for clustering, only one object forms a cluster.

2. Clustering is an unsupervised learning, and we usually don't split the dataset in to training data and test data.

3. Figure 5 shows the minimum error when k varies from 1 to 10.

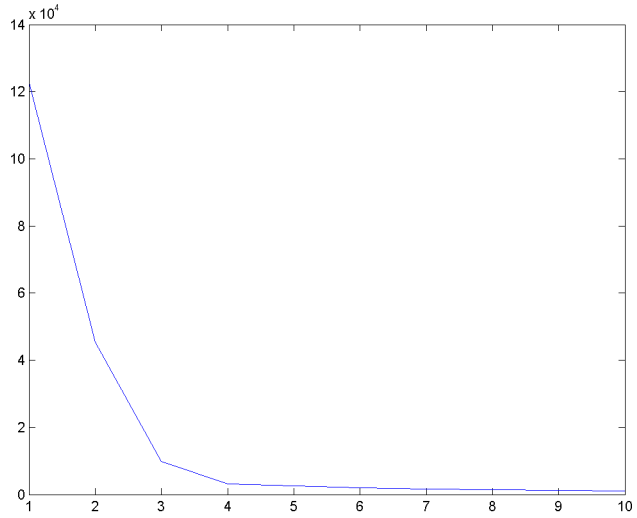


Figure 5: Error curve of Elbow Method for K-Means clustering without outlier

4. The plot for elbow method is given in Figure 5. According to the plot, $k = 3$ gives the sharpest elbow.

3.3 k-Medians

1. The clustering with the lowest error obtained by running k-means 50 times(Figure 6):

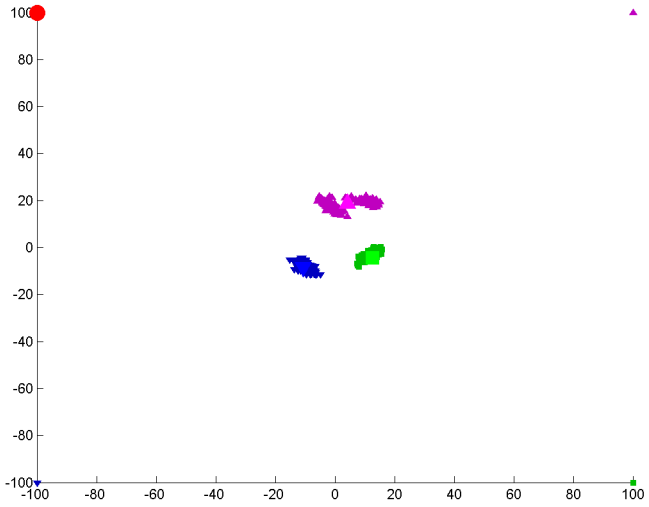


Figure 6: K-Means Clustering for dataset with an outlier

2. By elbow method, the error curve is given by(Figure 7):

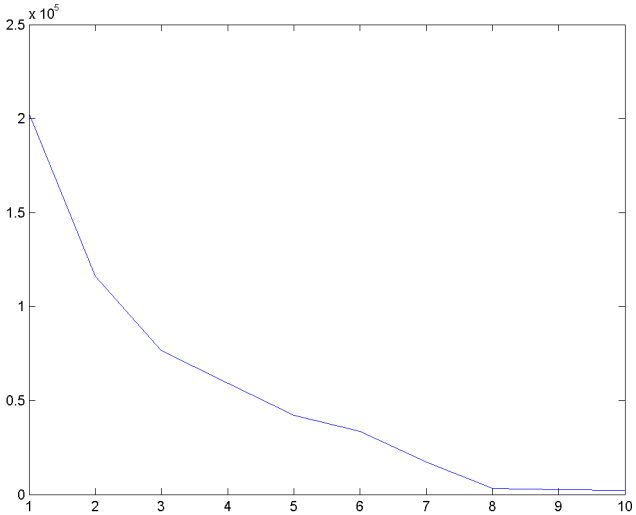


Figure 7: Error curve of Elbow Method for K-Medians clustering without outlier

The k should be chosen as 3.

3. The clustering by K-Medians is given by(Figure 8):

```
function [e] = error(medmodel,X)
[t,d] = size(X);
W = medmodel.W;
k = size(W,1);

% Compute Euclidean distance between each data point and each mean
distance=zeros(t,k);
 % Compute L1 norm between each data point and each mean
    for j=1:k
        for i=1:t
            distance(i,j)=norm((X(i,:)-W(j,:)),1);
        end
    end
sortD = sort(distance',1);
e =sum(sortD(1,:));
end
```

4. The error curve of Elbow method for K-Medians is given by(Figure 9):

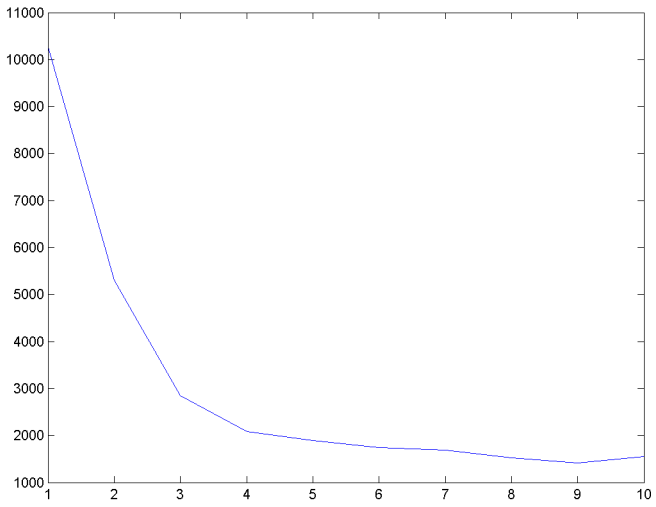


Figure 8: Error cure of Elbow method for K-Medians clustering

According to the error curve, the k should be chosen as 3.

4 Vector Quantization and Density-Based Clustering

4.1 Image Colour-Sapce Compression

1. The code for vector quantization:

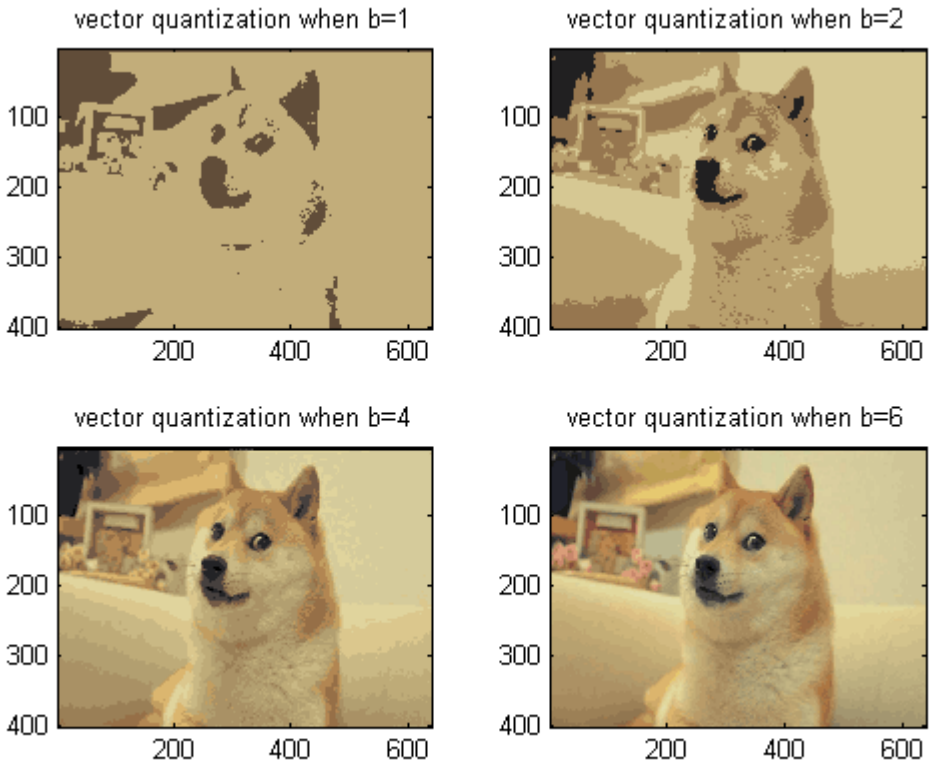
```
function [ Iquant ] = quantizeImage( I,b )
% function for vector quantization
```

```
[n d t]=size(I);
I=reshape(I,n*d,t);
% number of clusters
k=2^b;

% find the prototye
doPlot=0;
model=clusterKmeans(I,k,doPlot);

% replace the original pixel's colour with the nearest prototype
W=model.W;
y=model.y;
Iquant=W(y,:);
Iquant=reshape(Iquant,[n,d,t]);
end
```

2. When $b = 1, 2, 4, 6$, the corresponding output images are



4.2 Effect of Parameters on DBSCAN

Table 1: Values for parameters for density-based clustering method

ncluters	1	2	3	4
radius	300	200	50	10
minPts	10	10	10	10

4.3 K-Means vs. DBSCAN Clustering

1. k-Means clustering heavily depends on the initialization of k centers. If there's one initial center w_1 such that the closest center to each objects is not w_1, then there exists an empty cluster. The center of this empty cluster won't change in the following steps, this cluster returns NA when the algorithm converges.

2. Set the radius to 16 and minPoints to 3, and the clusters are given by:

   Cluster 1: grizzly+bear killer+whale beaver bat otter giant+panda polar+bear raccoon

   Cluster 2: antelope horse hippopotamus moose elephant ox sheep rhinoceros giraffe buffalo zebra deer pig cow

   Cluster 3: dalmatian persian+cat german+shepherd siamese+cat skunk mole tiger leopard fox hamster squirrel rabbit wolf chihuahua rat weasel bobcat lion mouse collie

   Cluster 4: blue+whale humpback+whale seal walrus dolphin

   Cluster 5: spider+monkey gorilla chimpanzee