2015124129 오혜빈

project-3

1. (Mean shift) Implement the followings, and analyze and compare the results.
 1) Implement the mean shift algorithm.
 2) Cluster the image based on the (R,G,B)vector of the image.
 3) Cluster the image based on the (H,S,I)vector of the image.
 4) Cluster the image based on the (R,G,B,x,y)vector of the image.
 (where (x,y) indicates a position of each pixel)

1) mean shift algorithm

영상의 픽셀값, 거리값등에 따라 d 차원 공간에 그 점을 mapping하고, 커널 속에 들어온 샘플들의 정규화된 가중치의 합으로 바꿔준다. 이 때 가중치는 커널의 크기 h 이내에 있는 점만 0이 아닌 값을 갖는다.
모드 탐색에서 차원 d가 커지면 계산량이 급속도로 증가하기 때문에 우회적으로 각 점의 소속을 결정하는 mean shift 방식을 사용한다.

$$\mathbf{y}_{t+1} = \mathbf{y}_t + \mathbf{m}(\mathbf{y}_t)$$

이때 $\mathbf{m}(\mathbf{y}_t) = \mathbf{y}_{t+1} - \mathbf{y}_t$

$$= \frac{\sum_{i=1}^{n} \mathbf{x}_i k\left(\frac{\mathbf{x}_i - \mathbf{y}_t}{h}\right)}{\sum_{i=1}^{n} k\left(\frac{\mathbf{x}_i - \mathbf{y}_t}{h}\right)} - \mathbf{y}_t$$

$$= \frac{\sum_{i=1}^{n} (\mathbf{x}_i - \mathbf{y}_t) k\left(\frac{\mathbf{x}_i - \mathbf{y}_t}{h}\right)}{\sum_{i=1}^{n} k\left(\frac{\mathbf{x}_i - \mathbf{y}_t}{h}\right)}$$

-code

1. m_y(t+1)을 구한다.

```
52 -        for b= t:t+2*h+1
53
54 -            temp= abs((x(b,:)-y(t,:))/h);
55 -            nn=temp.*temp;
56
57 -             if    sum(nn)/3<=1
58 -                    k(ii)=1;
59 -             else
60 -                    k(ii)=0;
61 -             end
62
63 -            m_y(t,:)=m_y(t,:)+abs((x(b,:)-y(t,:)))*k(ii);
64
65 -            ii=ii+1;
66
67 -         end
```

1-1. 커널의 값을 구한다. 거리가 1보다 가까우면 값은 1이고 나머지는 0이다.

1-2. m_y(t)에 (x_i-y_i)*k(__)값을 구하고 누적시킨다.

1-3. 이를 커널의 크기 2xh+1번 반복하여 커널 속에 있는 모든 x에 대해 계산하여 m_y(t)값을 구한다.

2. 앞에서 구한 m_y(t)를 이용하여 y(t+1)을 구한다.

```
49 -    for t=1:size(im,2)*size(im,1)-2*h-1
50
51 -        ii=1;
52 -       for b= t:t+2*h+1
53
54 -             temp= abs((x(b,:)-y(t,:))/h);
55 -             nn=temp.*temp;
56
57 -              if    sum(nn)/3<=1
58 -                    k(ii)=1;
59 -              else
60 -                    k(ii)=0;
61 -              end
62
63 -             m_y(t,:)=m_y(t,:)+abs((x(b,:)-y(t,:)))*k(ii);
64
65 -            ii=ii+1;
66
67 -         end
68
69 -      m_y(t,:)=m_y(t,:)/sum(k(:));
70 -       y(t+1,:)=y(t,:)+m_y(t,:);
71 -      tempp=sum(m_y(t).*m_y(t))/3;
72
73 -      if tempp<threshold
74 -          break;
75 -      end
76 -    end
```

2-1. 앞에서 구한 m_y(t)를 커널의 값으로 정규화시키고, 이전의 y(t)와 더해 y(t+1)을 구한다.

2-2. ||y(t+1)-y(t)||의 값이 threshold보다 작을 때의 y(t+1)값이 x_i의 수렴점이다. 그러므로 반복문을 벗어난다.

2)Cluster the image (r,g,b) vector

r,g,b 3차원에 픽셀값을 mapping하여 cluster한다. r,g,b 단지 컬러값만 표현하여 cluster하면 멀리 떨어진 화소가 같은 컬러값을 갖는 경우, 이는 같은 모드로 수렴하여 동일한 군집에 속하게 되는 한계가 발생한다.

- code

1. r은 1차원, g는 2차원, b는 3차원이므로 각각의 r,g,b값을 행렬 x에 저장했다.

```
16
17-    x(t,:)=[im(j,i,1),im(j,i,2),im(j,i,3)];
```

2. clustering

```
45-  for s=h+1:size(im,2)*size(im,1)
46
47-      y(1,:)=x(s,:);
48-      t=1;
```

```
78-      fj=ceil(s/128);
79-      fi=s-128*(fj-1);
80-      out(fj,fi+1)=x(s);
81        %v(s,:)=y(t+1,:);
82-  end
```

점들을 군집 중심 y(t+1)의 값에 매핑시키기 위해 모든 x에 대해 meanshift algorithm을 수행한다.

3)Cluster the image (H,S,I) vector

h,s,l 3차원에 픽셀값을 mapping하여 cluster한다. r, g, b 형태의 원본영상을 H,S,l로 바꾼 후 clustering한다. H는 hue, S는 saturation, I는 intensity를 뜻한다. 사람은 r,g,b 형식의 데이터보다 HSI형태의 데이터를 보고 더 직관적으로 어떤 색인지 바로 알 수 있다. 다음의 식을 사용하여 r,g,b값을 HSI 값으로 변환할 수 있다.

$$I = \frac{1}{3}(R+G+B)$$

$$S = 1 - \frac{3}{(R+G+B)}\min(R,G,B)$$

$$H = \begin{cases} \theta & if\ B \le G \\ 360-\theta & if\ B \ge G \end{cases}$$

$$\theta = \cos^{-1}\left\{\frac{2R-G-B}{2\sqrt{(R-G)^2+(R-B)(G-B)}}\right\}$$

- code

```
for j=1:size(im,1)
    for i=1:size(im,2)

    x(t,:)=[im(j,i,1),im(j,i,2),im(j,i,3)];

    R=im(j,i,1);
    G=im(j,i,2);
    B=im(j,i,3);

    H(j,i)= acos((2*R-G_B))/2*root(sqr(R-G)+(R-B)(G-B)));

    if B>=G
        H(j,i)=360-H(j,i);
    end

    S(j,i)=(1-3/(R+G+B)) * min(R,G,B);

    I=(R+G+B)/3;

    x(t,:)=[H(j,i),S(j,i),L(j,i)];

% x(t,:)=[im(j,i,1),im(j,i,2),im(j,i,3),i,j];
    t=t+1;

    end
end
```

위의 식을 이용하여 r,g,b값을 HSI값으로 변환하여 행렬에 저장했다.

위의 이미지는 rgb값을 hsl로 변환후 출력해본 이미지이다. 색의 차이를 더 뚜렷하게 알 수 있다.

4) Cluster the image using (r,g,b,x,y) vector

화소의 컬러에 해당하는 3차원 (r,g,b) 와 화소의 위치를 나타내는 2차원(x,y)를 결합하여 5차원 공간으로 매핑한다. 공간좌표와 컬러좌표는 다른 스케일을 갖기 때문에 이런 차이를 보정하기 위해 폭이 다른 두 개의 커널 함수의 곱을 사용한다.

- code

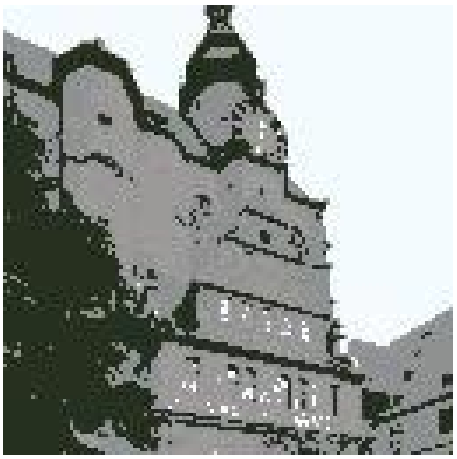
```
14 -    for j=1:size(im,1)
15 -    for i=1:size(im,2)
16
17      %x(t,:)=[im(j,i,1),im(j,i,2),im(j,i,3)];
18
19 -     x(t,:)=[im(j,i,1),im(j,i,2),im(j,i,3),i,j];
20 -    t=t+1;
21
22 -    end
23 -    end
```

-픽셀의 좌표값에 해당하는 i,j를 추가했다.

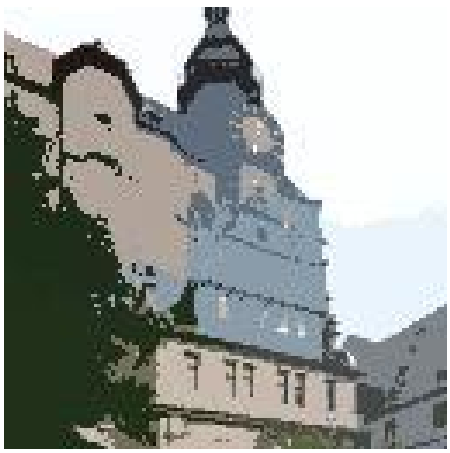
3) 결과 분석

-original image

2번 결과-h=4



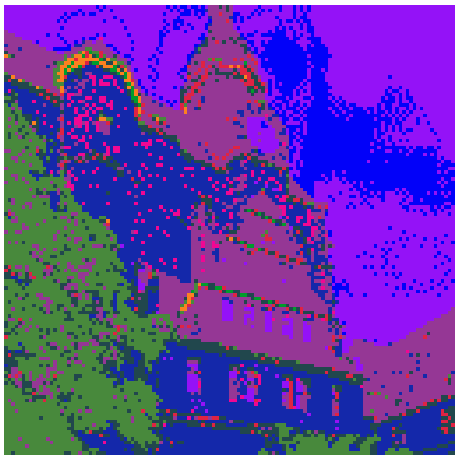
4번 결과 -spatial 고려



spatial을 고려하지 않고 clustering했을 때는 건물의 외벽이 거의 하나의 군집으로 분류되었다. spatial 영상의 좌표값을 고려한 clustering에서는 건물의 외벽이 여러개의 군집으로 분류되었다. 또, 하늘 부분에서도 spatial을 고려하지 않았을 때는 모두 하늘색으로 군집되었는데, spatial을 고려한 결과 구름이 많은 부분과 없는 하늘을 구분할 수 있다.

orginal image

3번 결과-h=4



-spatial 고려

HSI로 나타낸 결과 rgb형태였을 때 비슷해보이는 색의 차이가 더 뚜렷하게 느껴진다.

이 이미지 역시 위와같이 spatial을 고려하여 clustering했을 때 더 정확하다.

orginal image

2번 결과-h=4

4번 결과 -spatial 고려

이미지의 수풀 부분을 보면 spatial을 고려한 결과 더 정확하게 군집화되고 분활되었다.

2. (Multi-Layer Perceptron) Implement the followings, and analyze and compare the results.

1) Build a feed-forward MLP for the classification of MNIST dataset.

2) Implement the back-propagation using test set.

3) Measure the performance of the MLP.

1) feed-forward MLP

multi layer perceptron은 입력층과 은닉층 출력층 노드를 갖고 각 노드를 연결한 weight vector u, v를 갖는다. 이 u, v를 구하여 출력층에서 0~9까지의 숫자를 분류한다. 28x28 size의 이미지를 이용하므로 입력층 노드는 28x28개가 된다. 은닉층의 노드개수를 p라고 설정한다. u 는 (d+1)p개 생성되고(여기서 d는 입력층 노드의 개수이다.), v는 (p+1)m개 생성된다. 여기서 m은 출력층 노드의 개수이다. 출력층 노드의 값은 vector o로 1X10 size이다. 3으로 분리되었다면 o=[0 0 0 1 0 0 0 0 0 0]가 된다.

$\mathbf{o} = f(\mathbf{x})$ 또는 두 단계로 나누어 쓰면 $\mathbf{o} = q(\mathbf{z}),\ \mathbf{z} = p(\mathbf{x})$ (8.3)

은닉층의 j번째 노드, $1 \le j \le p$:

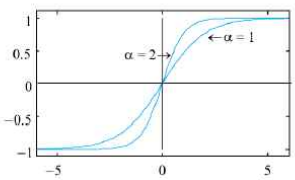
$$z\_sum_j = \sum_{i=1}^{d} x_i u_{ij} + u_{0j} \qquad (8.4)$$
$$z_j = \tau(z\_sum_j)$$

출력층의 k번째 노드, $1 \le k \le m$:

$$o\_sum_k = \sum_{j=1}^{p} z_j v_{jk} + v_{0k} \qquad (8.5)$$
$$o_k = \tau(o\_sum_k)$$

양극 시그모이드 함수와 도함수 :

$$\tau(x) = \frac{2}{1+e^{-\alpha x}} - 1$$
$$\tau'(x) = \frac{\alpha}{2}(1+\tau(x))(1-\tau(x))$$



-code

1. 변수들을 설정해준다.

```
d= 10      %length(filepaths): % input node 개수
t=zeros(10,1)  %예상 결과값 벡터
p=40: % number of hidden layer
m=10: %number of ouput node
l_r=0.5:
pp=40:
u=rand(d+1,p): %input-hidden layer weight
v=rand(p+1,m): %hidden-output layer weight

z_sum=rand(pp+1,1):
o=zeros(10,1):
o_sum=zeros(10,1);

```

2. 입력층 벡터와 타겟벡터를 저장한다.

```
    im_raw = imread(fullfile(train_img_dir,filepaths(i).name)): % one image

    im_lab = str2num(fgetl(fp)): % one label
%     imshow(im_raw): % display image
%     fprintf('%d \n', im_lab): % display label (commnad line)
    %pause



    im_raw=im2bw(im_raw):
    x=im_raw(:):  %input vector
    t(im_lab+1)=1: %expected output vector

```

1. 예제 코드를 이용해서 im_raw에 이미지의 픽셀값을 저장하고, im_lab에는 라벨값을 저장한다.

2. x는 입력층 벡터로 28X28의 1차원 벡터로 이미지의 픽셀값을 저장한다.

3. t는 타겟벡터로 예상되는 출력값을 저장하는 벡터이다. label값이 3이면 4번째 원소만 1이므로, 행렬 인덱스가 im_lab+1일 때 1값을 저장한다.

3. 전방 계산

```
51    %은닉층의 j번째 노드
52 -  for j=1:pp
53 -      for i=1:d
54 -          temp= x(i)*u(i+1,j);
55 -          z_sum(j)=temp+z_sum(j);
56 -      end
57 -      z_sum(j)=z_sum(j)+u(1,j);
58 -      [z(j),tau_d_z(j)]=tau(z_sum(j),0.5);
59
60 -  end
61
62
63    %출력층의 k번째 노드
64 -  for k=1:m
65 -      for j=1:pp
66 -          temp= z(j)*v(j+1,k);
67 -          z_sum(k)=temp+z_sum(k);
68 -      end
69
70 -      o_sum(k)=o_sum(k)+v(1,k);
71 -      [o(k),tau_d_o(k)]=tau(o_sum(k),0.5);
72
73 -  end
74 -   [~,t]=max(o(:));
75 -      o=zeros(10);
76 -      o(t+1)=1;
77
78
```

1. 입력노드에 weight값 u를 각각 곱하고 더한다.
2. 은닉층 노드에 weight v를 각각 곱하고 양극 시그모이드 함수를 취한다.
3. 함수를 취한 결과값중 가장 큰 값을 1이라하고 나머지는 0으로 하여 이값을 결과값 벡터인 o에 저장한다.

2) back propagation

3. train cell

```
79    %train cell
80
81 -  delta=(t-o).*tau_d_o;
82
83 -  for k=1:m
84 -      for j=1:pp+1
85 -          delta_v(j,k)=l_r*delta(k)*z(k);
86 -      end
87 -  end
88
89 -  for j=1:pp
90 -  ww(j)=tau_d_z(j)*sum(sum(delta.*v(j,:)));
91
92 -  end
93
94 -  for i=1:d+1
95 -      for j=1:pp
96 -          delta_u(i,j)=l_r*ww(j)*x(i);
97 -      end
98 -  end
99
100 -  v=delta_v+v;
101 -  u=delta_u+u;
102
103
104 -  end
105
```

weight vecotr u, v를 학습시키기 위해 위의 train cell을 반복한다.
back-propagation방법으로 학습한다. 식은 다음과 같다.

$$\delta_k = (t_k - o_k)\tau'(o\_sum_k),\ 1 \le k \le m \tag{8.9}$$

$$\Delta v_{jk} = -\rho\frac{\partial E}{\partial v_{jk}} = \rho\delta_k z_j,\ 0 \le j \le p,\ 1 \le k \le m \tag{8.10}$$

$$\eta_j = \tau'(z\_sum_j)\sum_{k=1}^{m}\delta_k v_{jk},\ 1 \le j \le p \tag{8.11}$$

$$\Delta u_{ij} = -\rho\frac{\partial E}{\partial u_{ij}} = \rho\eta_j x_i,\ 0 \le i \le d,\ 1 \le j \le p \tag{8.12}$$

4. 먼저 train set으로 전방계산을 한다.
5. 오류 역전파 알고리즘을 여러번 수행시켜 정확한 u, v를 얻어 u,v값을 업데이트 한다.
6. test set을 전방 계산한다.