Name:

Karan Pandya

Netid:

karandp2

CS 441 - HW3: PDFs and Outliers

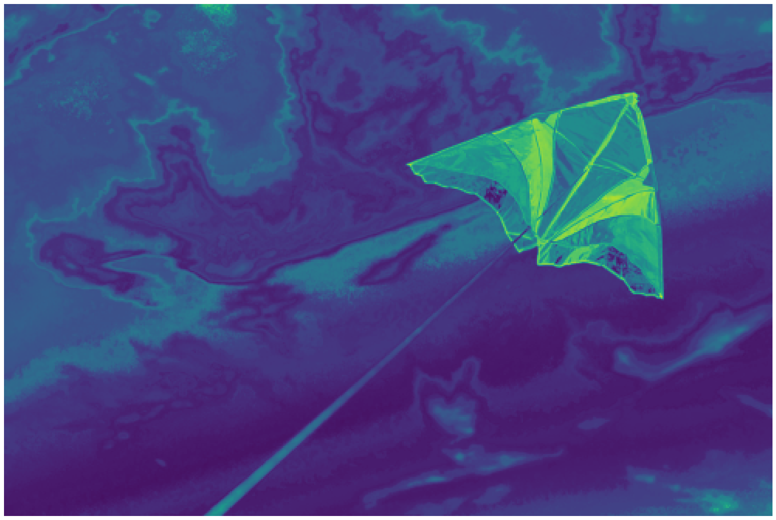
Complete the sections below. You do not need to fill out the checklist.

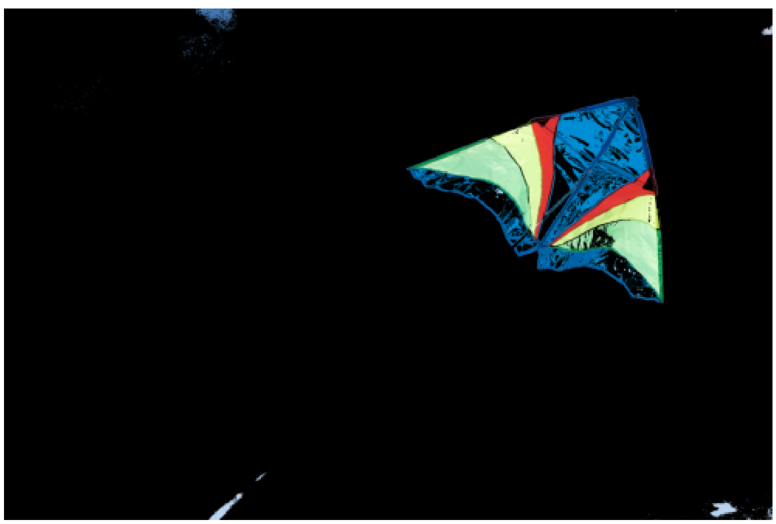
Total Points Available	**[ ] / 160**
1. Estimating PDFs	
a. Segmentation with per-channel PDFs	[ ] / 15
b. Segmentation with clustered value PDFs	[ ] / 15
c. Segmentation with GMMs	[ ] / 20
2. Robust Estimation	
a. Assume no noise	[ ] / 10
b. Robust estimation with percentiles	[ ] / 15
c. Robust estimation with EM	[ ] / 25
3. Stretch Goals	
a. Impact of school on salary	[ ] / 20
b. Impact of experience on salary	[ ] / 20
c. Mutual information: discrete pdf	[ ] / 10
d. Mutual information: GMM	[ ] / 10

1. Estimating PDFs

Include the generated images (score map and thresholded RGB) from the display code. List any parameters.

a. Method 1 (Per-channel discrete):

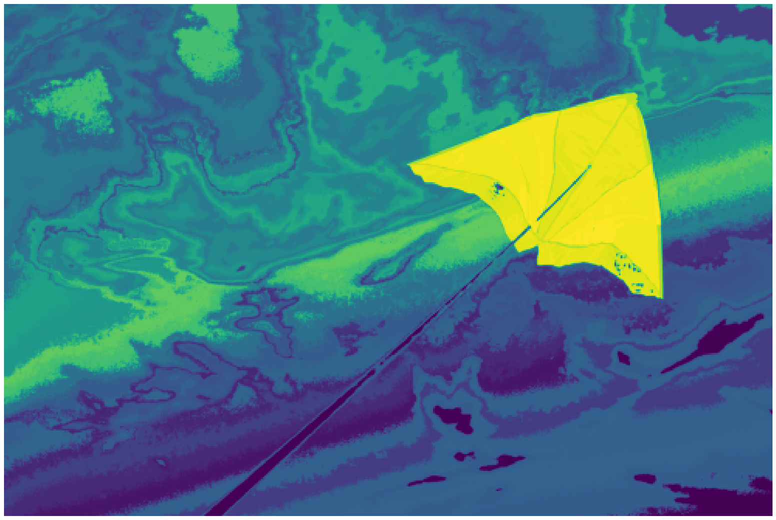


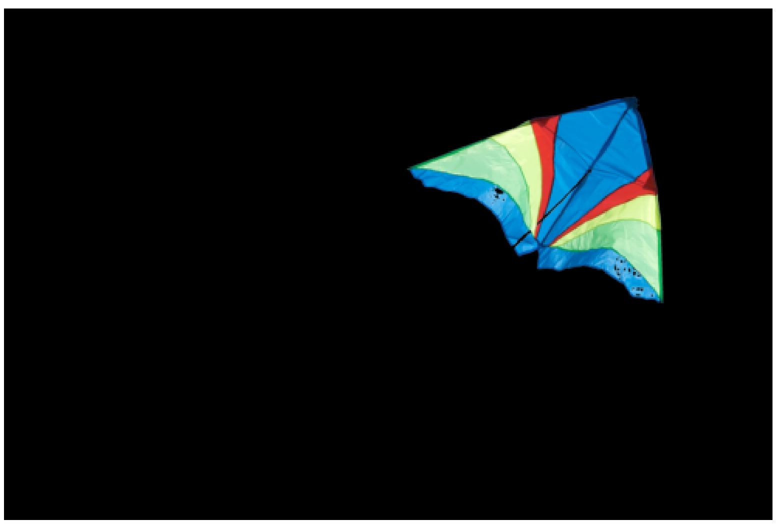


Number of bins / discrete values per channel, threshold

Nbins = 110
Threshold = 1.1

b. Method 2 (Clustering, discrete):

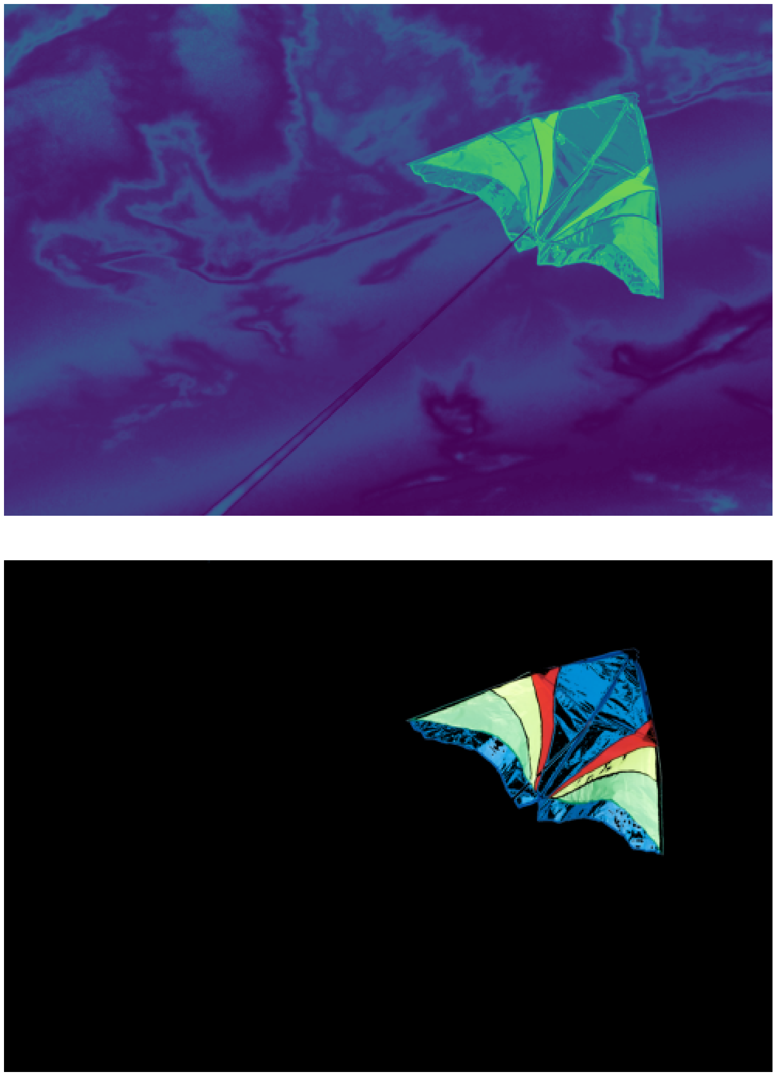




Number of clusters, threshold

Clusters= 56
Threshold = 1.4

c. Method 3 (Gaussian Mixture Model):



Number of components, variance model, threshold

Components: 4
Covariance: diag
Threshold: 1.2

2. Robust Estimation

Round to nearest whole number.

	a. No noise	b. Percentiles	c. EM
Min	64694	75493	64694
Mean	123750	113085	112066
Std	61954	16582	18315
Max	611,494	159,901	169,008

First five indices of invalid data (based on EM solution, you add last 3)

18	28	49	127	128

3. Stretch Goals

a. Impact of school on salary

Report mean salary overall and for each school

	Average Salary
Overall	111,984
School 0 (UIUC)	118,932
School 1 (MIT)	105,498
School 2 (Cornell)	112,115

Describe your approach to estimate this.

First find the good and bad salaries in the data then get the weights and then from the good weights for all salaries, separate the weights for UIUC, MIT and Cornell. Now we have the list of weights of UIUC, MIT and Cornell separately. Then, we take the mean of these weights of all three universities to get the probabilities of the salary being from UIUC,MIT and Cornell. Then using this information we calculate the likelihood that the missing salary is from each school and assign it the school with maximum probability. After that we perform the M step and calculate the mean for all schools.

b. Impact of years of experience on salary

How much are salaries expected to increase with one year of experience?

23,122

Describe your approach to estimate this.

For calculating the increase in salary with one year experience we first calculate the increase in salary per year of experience by dividing the salary by the number of years for each individual. After that we calculate the average increase in salary per year of experience for all individuals. Then we update this estimate in each iteration of the EM algorithm.

c. Mutual information of sex and age, discrete approach

Mutual information (base natural log)

-

d. Mutual information of sex and age, GMM approach

Mutual information (base natural log)

-

CS441_SP24_HW3_karan

March 7, 2024

0.1 CS441: Applied ML - HW 3

0.1.1 Part 1: Estimating PDFs

[ ]:
```
# initalization code

import numpy as np
from matplotlib import pyplot as plt
import cv2

# read images
im = cv2.imread('kite.jpg')  # this is the full image
im = cv2.cvtColor(im, cv2.COLOR_BGR2RGB)/255
im = cv2.blur(im, (3, 3))

crop = cv2.imread('kite_crop.jpg')  # this is the cropped image
crop = cv2.cvtColor(crop, cv2.COLOR_BGR2RGB)/255
crop = cv2.blur(crop, (3, 3))

# displays a single image
def display_image(im):
  plt.imshow(im)
  plt.axis('off')
  plt.show()

# displays the image, score map, thresholded score map, and masked image
def display_score(im, score_map, thresh):
  display_image(im)
  display_image(np.reshape(score_map, (im.shape[:2])))
  plt.imshow(np.reshape(score_map>thresh, (im.shape[0], im.shape[1])),␣
↪cmap='gray')
  plt.axis('off')
  plt.show()
  display_image(np.tile(np.reshape(score_map>thresh, (im.shape[0], im.shape[1],␣
↪1)), (1,1,3))*im)

print('Whole image')
display_image(im)
```

```
print('Foreground')
display_image(crop)
```

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

Whole image

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

Foreground

Method 1 (per channel hist)

```
[ ]: import numpy as np

     # Reference for the utility functions (slide colab notebook by prof: https://
      ↪colab.research.google.com/drive/1H4_jS1oxiOxZkfvh5w5KEWF2zFs4axty?
      ↪usp=sharing)
     im_reshaped = np.reshape(im, (im.shape[0]*im.shape[1], 3))
     crop_reshaped = np.reshape(crop, (crop.shape[0]*crop.shape[1], 3))

     def create_bins(data, num_bins=28):
         '''
         based on data range, creates bin edges and returns the edges as a list
         '''
         max_value = max(data)
         min_value = min(data)
         data_range = (max_value - min_value) / num_bins
         bins = range(num_bins + 1) * data_range + min_value

         return bins

     def discretize(x, bins):
         '''
         Assigns bin index to each element in x
         '''
         xd = np.zeros(x.shape, dtype='uint32')
```

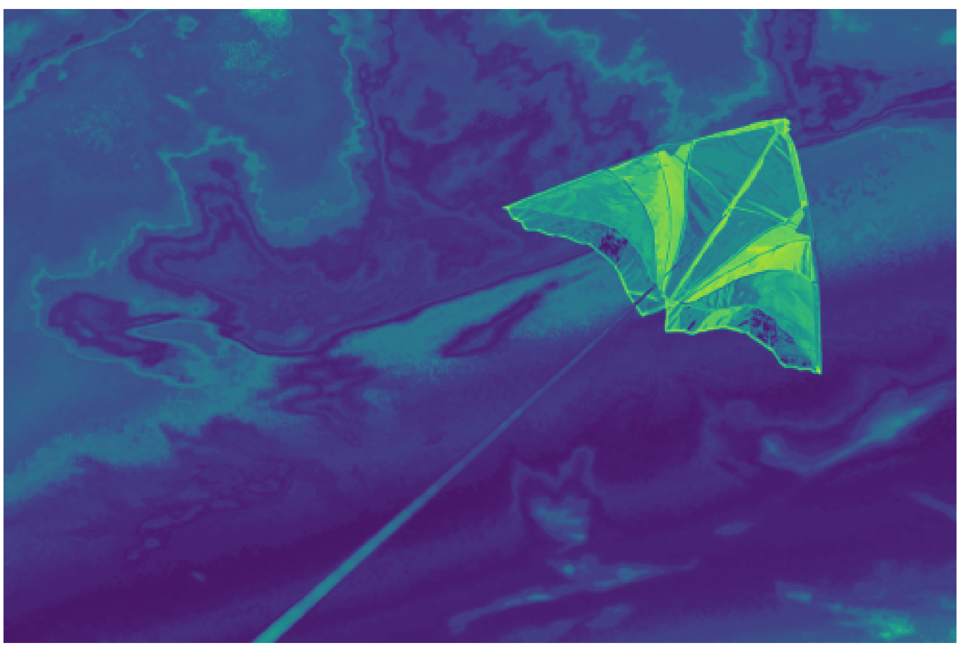
```
    for i in range(1, len(bins)):
        xd += x > bins[i]
    xd[xd < 0] = 0
    xd[xd > len(bins) - 2] = len(bins) - 2
    # print(xd)
    return xd

# estimate discrete pdf
def estimate_discrete_pdf(values, nvalues, prior=1):
    '''
    Estimate P(values=v) for each possible v in (0, nvalues)
    Input:
      values: the values of the data
      nvalues: range of values, such that 0 <= values < nvalues
      prior: initial count used to prevent any value from having zero␣
↪probability
    Output:
      p[nvalues,]: P(values=v) for each v
    '''
    p = np.ones(len(nvalues) - 1, ) * prior
    print('P.shape: ', p.shape)
    for v in values:
        p[v] += 1
    p_total = p.sum()
    for v in range(len(p)):
        p[v] = p[v] / p_total / (nvalues[v + 1] - nvalues[v])  # (bin_count /␣
↪total_count)/(bin_size)
    return p

def calc_pixel_score(all_px, px_inside_box, num_bins=64, prior=1):
    ''' Engine function '''
    scores = np.zeros(all_px.shape[0])

    for channel in range(3):
        # Selecting a channel
        channel_px_inside_box = px_inside_box[:, channel]
        channel_all_px = all_px[:, channel]

        # create bin edges
        bins = create_bins(channel_all_px, num_bins=num_bins)

        # split pixels into bins (gives the index of assigned bin)
        discrete_values_inside_box = discretize(channel_px_inside_box, bins)
        discrete_values_all_px = discretize(channel_all_px, bins)

        print('Discrete: ', discrete_values_inside_box.shape,␣
↪discrete_values_inside_box[:4])
```

```
        pdf_inside_box = estimate_discrete_pdf(discrete_values_inside_box,␣
↪bins, prior=prior)
        pdf_all_px = estimate_discrete_pdf(discrete_values_all_px, bins,␣
↪prior=prior)

        scores += np.log(pdf_inside_box[discrete_values_all_px]) - np.
↪log(pdf_all_px[discrete_values_all_px])

    return scores

# MAIN
all_pixels = np.reshape(im, (im.shape[0]*im.shape[1], 3))
pixels_inside_box = np.reshape(crop, (crop.shape[0]*crop.shape[1], 3))
num_bins = 110

score = calc_pixel_score(all_pixels, pixels_inside_box, num_bins=num_bins,␣
↪prior=1)
print(min(score), max(score), np.mean(score))
print(score.shape)
threshold = 1.25
display_score(im, score_map=score, thresh=threshold)
```

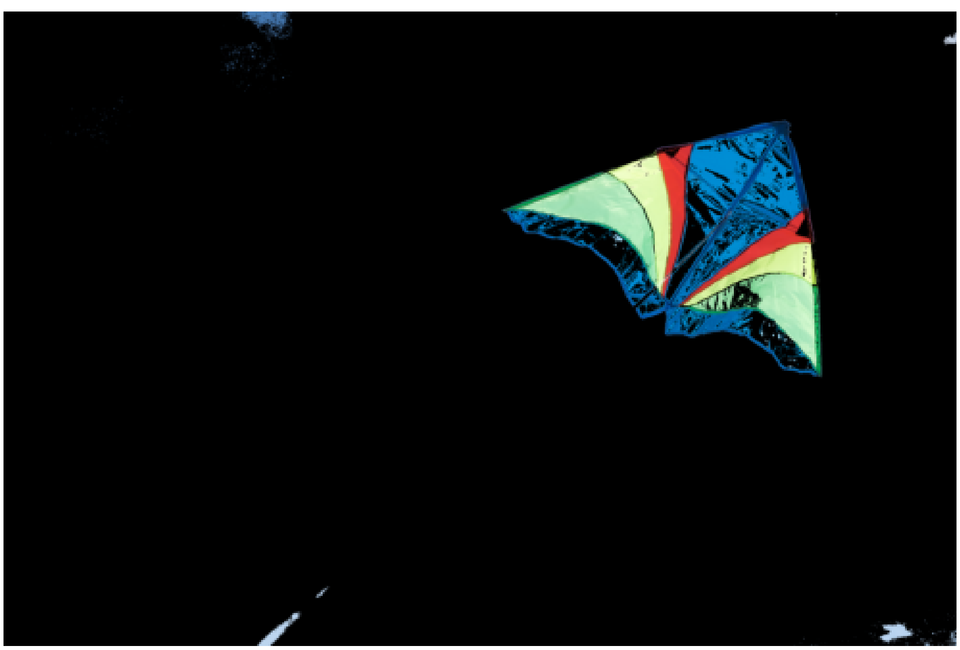
```
Discrete:  (26999,) [62 62 62 61]
P.shape:  (110,)
P.shape:  (110,)
Discrete:  (26999,) [70 70 70 70]
P.shape:  (110,)
P.shape:  (110,)
Discrete:  (26999,) [86 86 87 87]
P.shape:  (110,)
P.shape:  (110,)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
```

```
-6.161450145101288 10.66478909083693 -2.4660814705345664
(6685530,)
```



Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).



Method 2 (Kmeans)

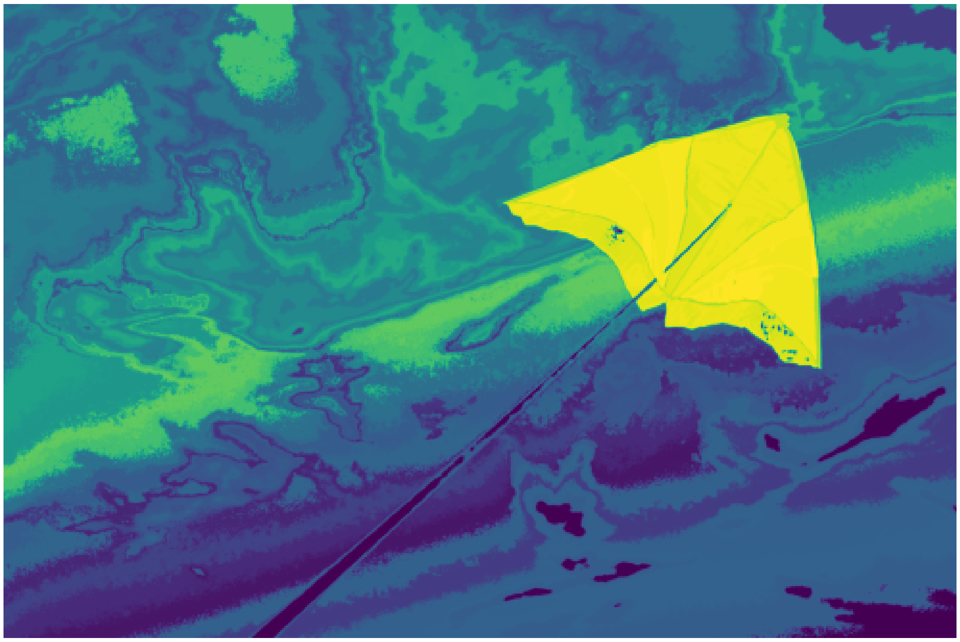
[ ]:
```
# init
!apt install libomp-dev > /dev/null 2>&1
!pip install faiss-cpu > /dev/null 2>&1
import faiss
```

[ ]:
```
def estimate_discrete_pdf_kmeans(data_values, data_range, prior=1):
    '''
    Estimate P(data_values = v) for each possible v in (0, data_range)
    Input:
        data_values: the values of the data
        data_range: range of values, such that 0 <= data_values < data_range
        prior: initial count used to prevent any value from having zero␣
↪probability
    Output:
        p[data_range,]: P(data_values = v) for each v
    '''
    p = np.ones(len(data_range),) * prior
    print('PDF shape', p.shape)
    for v in data_values:
        p[v] += 1
    ptotal = p.sum()
    for v in range(len(p)):
        p[v] = (p[v] / ptotal) / (data_range[v] + 1)
    return p

def compute_pixelwise_score(all_data, data_inside_box, num_bins=32, prior=1,␣
↪num_clusters=56):

    # Perform k-means clustering on all_data
    kmeans = faiss.Kmeans(d=all_data.shape[1], k=num_clusters, niter=20,␣
↪verbose=True)
    kmeans.train(all_data.astype(np.float32))

    _, cluster_indices            = kmeans.index.search(all_data.astype(np.
↪float32), 1)
    _, cluster_indices_inside_box = kmeans.index.search(data_inside_box.
↪astype(np.float32), 1)

    cluster_probs_image = estimate_discrete_pdf_kmeans(cluster_indices.
↪flatten(), np.arange(num_clusters), prior=prior)

    cluster_probs_box = estimate_discrete_pdf_kmeans(cluster_indices_inside_box.
↪flatten(), np.arange(num_clusters), prior=prior)

    score = np.zeros(all_data.shape[0])
    for i in range(all_data.shape[0]):
```

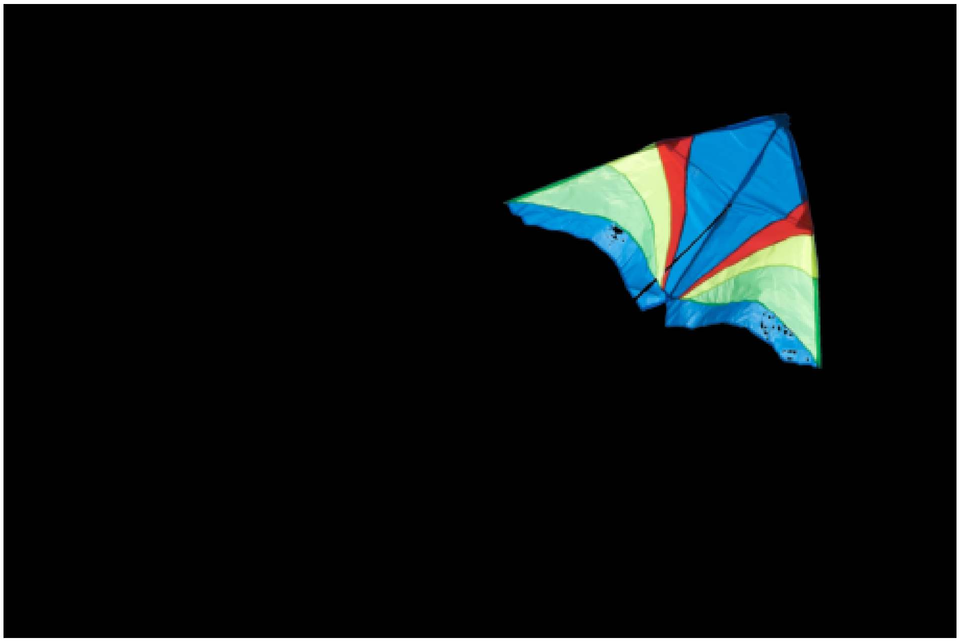
```python
        score[i] = np.log(cluster_probs_box[cluster_indices[i][0]]) - np.
 ↪log(cluster_probs_image[cluster_indices[i][0]])

    return score

# reshape
im_data   = np.reshape(im, (im.shape[0]*im.shape[1], 3))
crop_data = np.reshape(crop, (crop.shape[0]*crop.shape[1], 3))

# estimate PDFs
score_kmeans = compute_pixelwise_score(im_data, crop_data)

threshold_kmeans = 1.4
display_score(im=im, score_map=score_kmeans, thresh=threshold_kmeans)
```

```
Sampling a subset of 16384 / 6685530 for training
Clustering 16384 points in 3D to 64 clusters, redo 1 times, 20 iterations
  Preprocessing in 0.08 s
  Iteration 19 (0.01 s, search 0.00 s): objective=16.2752 imbalance=1.282
nsplit=0
PDF shape (64,)
PDF shape (64,)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
```



```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
```



Method 3 (GMM)

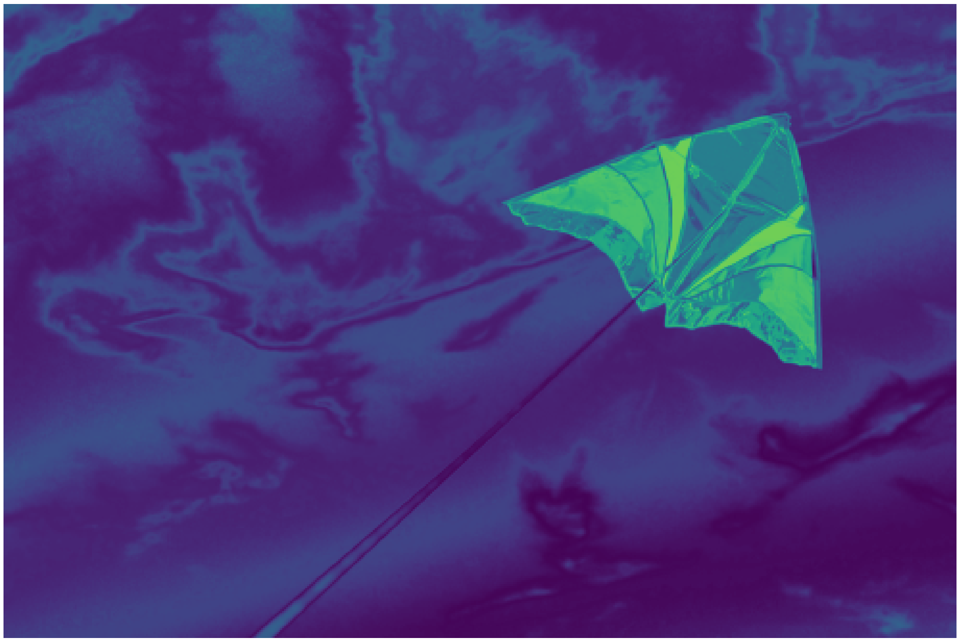
```
[ ]: from sklearn.mixture import GaussianMixture as GMM

def compute_pixelwise_score_gmm(all_data, data_inside_box, num_components=4,␣
 ↪covar_type='diag'):
    gmm_all = GMM(n_components=num_components, covariance_type=covar_type)
    gmm_all.fit(all_data)

    log_likelihood_all = gmm_all.score_samples(all_data)

    gmm_box = GMM(n_components=num_components, covariance_type=covar_type)
    gmm_box.fit(data_inside_box)

    log_likelihood_box = gmm_box.score_samples(all_data)

    scores = log_likelihood_box - log_likelihood_all

    print(f'Min, Max: {min(scores)}, {max(scores)}, Mean Score: {np.
 ↪mean(scores)}')

    return scores

# GMM
print('GMM\n')
```

```
full_data = np.reshape(im, (im.shape[0]*im.shape[1], 3))
box_crop_data = np.reshape(crop, (crop.shape[0]*crop.shape[1], 3))

score_gmm = compute_pixelwise_score_gmm(full_data, box_crop_data,␣
 ↪num_components=5, covar_type='diag')
THRESHOLD = 1.2

display_score(im=im, score_map=score_gmm, thresh=THRESHOLD)
```

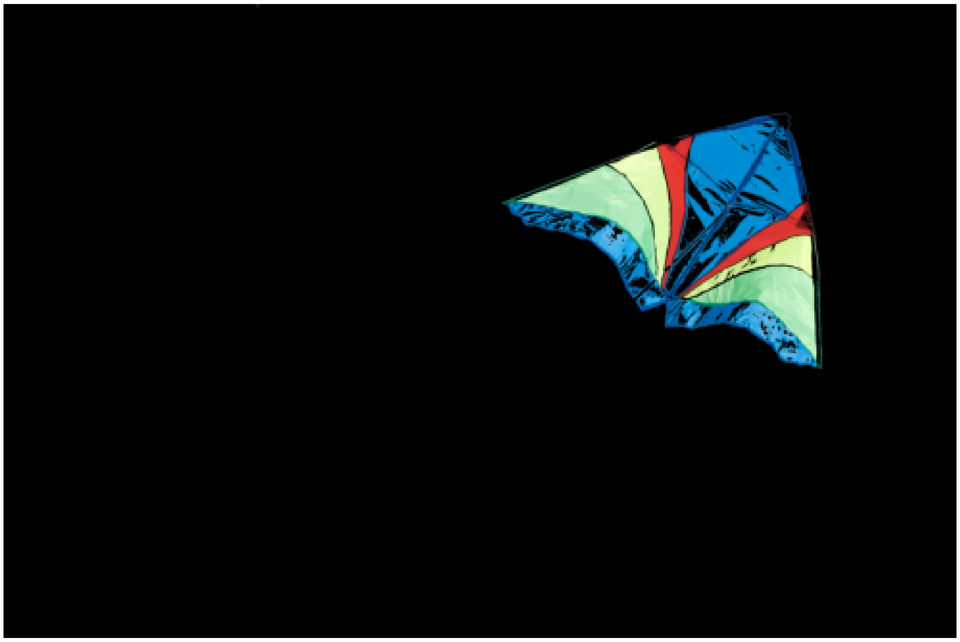
```
GMM METHOD 3
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
```

```
Min, Max: -5.814334799458206, 10.934270930775032, Mean Score: -3.085887751306602
```



```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
```



0.2 Part 2: Robust Estimation

[ ]:
```
import numpy as np
from matplotlib import pyplot as plt

# load data
T = np.load('salary.npz')
(salary, years, school) = (T['salary'], T['years'], T['school'])
```

1. No noise Compute the statistics for the data as a whole

[ ]:
```
# TO DO
salary_mu = np.mean(salary)
salary_std = np.std(salary)
salary_min = np.min(salary)
salary_max = np.max(salary)
print('Mean: {}  Std: {}  Min: {}   Max: {}'.format(salary_mu, salary_std,␣
 ↪salary_min, salary_max))
```

```
Mean: 123749.835  Std: 61953.77348723623  Min: 64694.0   Max: 611494.0
```

2. Percentiles Assume valid data will fall between the 5th and 95th percentile.

[ ]:
```
```

```
# TO DO
salary_max = np.percentile(salary, 95) + (np.percentile(salary,95) - np.
 ↪percentile(salary,5))*0.05/0.9
salary_min = np.percentile(salary, 5) - (np.percentile(salary,95) - np.
 ↪percentile(salary,5))*0.05/0.9
salary_clean = salary[(salary_min < salary) & (salary< salary_max)]
salary_clean_mu = np.mean(salary_clean)
salary_clean_std = np.std(salary_clean)
print('Mean: {}  Std: {}  Min: {}   Max: {}'.format(salary_clean_mu,␣
 ↪salary_clean_std, salary_min, salary_max))
```

```
Mean: 113084.95652173914  Std: 16582.440683441288  Min: 75493.8   Max:
159900.79999999973
```

3. EM Assume valid data follows a Gaussian distribution, while the fake data has a uniform distribution between the minimum and maximum value of salary.

[ ]:
```
# TO DO
niter = 20
pz = 0.5
N = len(salary)
salary_mean = salary_mu
salary_std = np.sqrt(np.sum((salary-salary_mean)**2)/ N)
score_std = salary_std
score_mean = salary_mu

for t in range(niter):
  last_mean = score_mean
  # E-step
  p_s_good = pz/(np.sqrt(2*np.pi)*score_std) * np.exp(-1/2 *␣
 ↪((salary-score_mean)/score_std)**2)
  p_s_bad = (1-pz)/(np.max(salary) - np.min(salary)) # uniform in range [0, 10]
  weights = p_s_good / (p_s_good + p_s_bad)

  # M-step
  # assign parameters that maximize likelihood under latent variable likelihood
  # estimate mean for salaries
  weighted_sum = np.sum(weights*salary)
  score_mean = weighted_sum / np.sum(weights)

  # estimate std
  score_std = np.sqrt(np.sum(weights*(salary-score_mean)**2) / np.sum(weights))

  # estimate pz
  pz = np.mean(weights)
  print("salary mean", score_mean, "salary std", score_std, "iter", t)
  if np.all(np.abs(last_mean-score_mean)<0.00001): # check for convergence
```

```
    break
# print the first five indices of salaries that are not likely to be valid
not_valid_indices = np.where(weights <0.5)[0][:5]

valid = salary[np.where(weights>=0.5)].copy()
print(len(salary[weights>=0.5]), N)
score_max = np.max(valid)
score_min = np.min(valid)
print("nvi", not_valid_indices ,"\n")
print("max salary with prob >0.5" ,max(salary[weights>0.5]), "\nmin salary with␣
 ↪prob >0.5",  min(salary[weights>0.5]))
print('Mean: {}  Std: {}  Min: {}   Max: {}'.format(np.mean(valid), np.
 ↪std(valid),score_min, score_max))
```

```
salary mean 113061.51544611696 salary std 20334.864145666437 iter 0
salary mean 112102.03646435999 salary std 17542.424453113388 iter 1
salary mean 111972.77900816128 salary std 17731.76572043111 iter 2
salary mean 111973.30398264094 salary std 17895.18053687427 iter 3
salary mean 111980.70995130215 salary std 17947.089773113807 iter 4
salary mean 111983.36491461674 salary std 17961.27546276651 iter 5
salary mean 111984.11362062384 salary std 17965.027751972026 iter 6
salary mean 111984.31337933977 salary std 17966.01292683679 iter 7
salary mean 111984.36594826641 salary std 17966.271137994085 iter 8
salary mean 111984.3797348272 salary std 17966.338786323766 iter 9
salary mean 111984.38334732175 salary std 17966.35650763947 iter 10
salary mean 111984.38429369849 salary std 17966.361149845103 iter 11
salary mean 111984.38454161024 salary std 17966.362365891735 iter 12
salary mean 111984.38460655203 salary std 17966.362684440155 iter 13
salary mean 111984.38462356382 salary std 17966.362767885195 iter 14
salary mean 111984.38462802011 salary std 17966.36278974396 iter 15
191 200
nvi [ 18  28  49 127 128]

max salary with prob >0.5 169008.0
min salary with prob >0.5 64694.0
Mean: 112065.86387434555  Std: 18314.982159703723  Min: 64694.0   Max: 169008.0
```

0.3 Part 4: Stretch Goals

Include all your code used for any stretch goals in this section. Add headings where appropriate.

[ ]:
```
import numpy as np



num_iterations = 20
prior_probability = 0.33
num_salaries = len(salary)
```

```
salary_mean = np.mean(salary)
salary_std = np.sqrt(np.sum((salary - salary_mean) ** 2) / num_salaries)


mean_score_UIUC    = np.mean(salary[school == 0])  # UIUC
mean_score_MIT     = np.mean(salary[school == 1])  # MIT
mean_score_Cornell = np.mean(salary[school == 2])  # CORNELL

std_score = np.std(salary)  # common


prior_prob_UIUC    = prior_probability
prior_prob_MIT     = prior_probability
prior_prob_Cornell = prior_probability


missing_school_assignments = np.zeros(num_salaries)

experience_increase_mean = np.mean(salary / years)
experience_increase_std = np.std(salary / years)

for iteration in range(num_iterations):
    prev_mean = salary_mean

    # I) E-step
    # update probability that each salary is good
    p_salary_good = (
        prior_probability
        / (np.sqrt(2 * np.pi) * salary_std)
        * np.exp(-1 / 2 * ((salary - salary_mean) / salary_std) ** 2)
    )
    p_salary_bad = (1 - prior_probability) / (np.max(salary) - np.min(salary))
    weights = p_salary_good / (p_salary_good + p_salary_bad)

    # E-step for schools
    prior_prob_UIUC = np.mean(weights[school == 0])
    prior_prob_MIT = np.mean(weights[school == 1])
    prior_prob_Cornell = np.mean(weights[school == 2])

    # Assign missing values to the most likely school or as invalid
    for i in range(num_salaries):
        if school[i] == -1:
            likelihood_UIUC = (
                prior_prob_UIUC
                / (np.sqrt(2 * np.pi) * std_score)
                * np.exp(-1 / 2 * ((salary[i] - mean_score_UIUC) / std_score)
 ↪** 2)
```

```
        )
        likelihood_MIT = (
            prior_prob_MIT
            / (np.sqrt(2 * np.pi) * std_score)
            * np.exp(-1 / 2 * ((salary[i] - mean_score_MIT) / std_score) **␣
↪2)
        )
        likelihood_Cornell = (
            prior_prob_Cornell
            / (np.sqrt(2 * np.pi) * std_score)
            * np.exp(-1 / 2 * ((salary[i] - mean_score_Cornell) /␣
↪std_score) ** 2)
        )

        # Assign the missing value to the school with the highest likelihood
        max_likelihood_school = np.argmax([likelihood_UIUC, likelihood_MIT,␣
↪likelihood_Cornell])
        missing_school_assignments[i] = max_likelihood_school

    # II) M-step
    weighted_sum = np.sum(weights * salary)
    salary_mean = weighted_sum / np.sum(weights)

    salary_std = np.sqrt(np.sum(weights * (salary - salary_mean) ** 2) / np.
↪sum(weights))

    prior_probability = np.mean(weights)

    mean_score_UIUC = np.sum(weights[school == 0] * salary[school == 0]) / np.
↪sum(weights[school == 0])
    mean_score_MIT = np.sum(weights[school == 1] * salary[school == 1]) / np.
↪sum(weights[school == 1])
    mean_score_Cornell = np.sum(weights[school == 2] * salary[school == 2]) /␣
↪np.sum(weights[school == 2])

    # M-step for salary increase per year
    valid_experience_indices = np.where(years != 0)[0]
    if len(valid_experience_indices) > 0:
        experience_increase_mean = np.sum(weights[valid_experience_indices] *␣
↪(salary[valid_experience_indices] / years[valid_experience_indices])) / np.
↪sum(weights[valid_experience_indices])
        experience_increase_std = np.sqrt(np.
↪sum(weights[valid_experience_indices] * ((salary[valid_experience_indices] /␣
↪years[valid_experience_indices]) - experience_increase_mean) ** 2) / np.
↪sum(weights[valid_experience_indices]))
    else:
```

```
        experience_increase_mean = 0
        experience_increase_std = 0

    print(
        f"Iteration {iteration}:\t mean_salary = {salary_mean:0.2f}\tsalary_std␣
↪= {salary_std:0.2f}\t"
        f"UIUC_mean = {mean_score_UIUC:0.2f}\tMIT_mean = {mean_score_MIT:0.
↪2f}\tCORNELL_mean = {mean_score_Cornell:0.2f}\t"
        f"Experience_increase_mean = {experience_increase_mean:0.
↪2f}\tExperience_increase_std = {experience_increase_std:0.2f}"
    )

    # III) check for convergence
    if np.all(np.abs(prev_mean - salary_mean) < 0.00001):
        print(f'Converged on iter: {iteration}')
        break
```

```
Iteration 0:     mean_salary = 113061.52        salary_std = 20334.86
UIUC_mean = 119113.74   MIT_mean = 107414.10    CORNELL_mean = 113379.20
Experience_increase_mean = 24330.55     Experience_increase_std = 32956.91
Iteration 1:     mean_salary = 112102.04        salary_std = 17542.42
UIUC_mean = 118768.98   MIT_mean = 105680.72    CORNELL_mean = 112331.24
Experience_increase_mean = 23086.71     Experience_increase_std = 29095.72
Iteration 2:     mean_salary = 111972.78        salary_std = 17731.77
UIUC_mean = 118851.58   MIT_mean = 105527.63    CORNELL_mean = 112196.23
Experience_increase_mean = 23127.08     Experience_increase_std = 29099.18
Iteration 3:     mean_salary = 111973.30        salary_std = 17895.18
UIUC_mean = 118909.90   MIT_mean = 105503.61    CORNELL_mean = 112135.02
Experience_increase_mean = 23125.40     Experience_increase_std = 29088.83
Iteration 4:     mean_salary = 111980.71        salary_std = 17947.09
UIUC_mean = 118926.19   MIT_mean = 105499.85    CORNELL_mean = 112119.54
Experience_increase_mean = 23123.15     Experience_increase_std = 29085.30
Iteration 5:     mean_salary = 111983.36        salary_std = 17961.28
UIUC_mean = 118930.51   MIT_mean = 105499.14    CORNELL_mean = 112115.66
Experience_increase_mean = 23122.43     Experience_increase_std = 29084.40
Iteration 6:     mean_salary = 111984.11        salary_std = 17965.03
UIUC_mean = 118931.65   MIT_mean = 105498.98    CORNELL_mean = 112114.66
Experience_increase_mean = 23122.24     Experience_increase_std = 29084.17
Iteration 7:     mean_salary = 111984.31        salary_std = 17966.01
UIUC_mean = 118931.95   MIT_mean = 105498.94    CORNELL_mean = 112114.40
Experience_increase_mean = 23122.19     Experience_increase_std = 29084.12
Iteration 8:     mean_salary = 111984.37        salary_std = 17966.27
UIUC_mean = 118932.03   MIT_mean = 105498.93    CORNELL_mean = 112114.34
Experience_increase_mean = 23122.18     Experience_increase_std = 29084.10
Iteration 9:     mean_salary = 111984.38        salary_std = 17966.34
UIUC_mean = 118932.05   MIT_mean = 105498.93    CORNELL_mean = 112114.32
Experience_increase_mean = 23122.17     Experience_increase_std = 29084.10
```

```
Iteration 10:    mean_salary = 111984.38        salary_std = 17966.36
UIUC_mean = 118932.05   MIT_mean = 105498.93    CORNELL_mean = 112114.31
Experience_increase_mean = 23122.17     Experience_increase_std = 29084.09
Iteration 11:    mean_salary = 111984.38        salary_std = 17966.36
UIUC_mean = 118932.05   MIT_mean = 105498.93    CORNELL_mean = 112114.31
Experience_increase_mean = 23122.17     Experience_increase_std = 29084.09
Iteration 12:    mean_salary = 111984.38        salary_std = 17966.36
UIUC_mean = 118932.05   MIT_mean = 105498.93    CORNELL_mean = 112114.31
Experience_increase_mean = 23122.17     Experience_increase_std = 29084.09
Iteration 13:    mean_salary = 111984.38        salary_std = 17966.36
UIUC_mean = 118932.05   MIT_mean = 105498.93    CORNELL_mean = 112114.31
Experience_increase_mean = 23122.17     Experience_increase_std = 29084.09
Iteration 14:    mean_salary = 111984.38        salary_std = 17966.36
UIUC_mean = 118932.05   MIT_mean = 105498.93    CORNELL_mean = 112114.31
Experience_increase_mean = 23122.17     Experience_increase_std = 29084.09
Iteration 15:    mean_salary = 111984.38        salary_std = 17966.36
UIUC_mean = 118932.05   MIT_mean = 105498.93    CORNELL_mean = 112114.31
Experience_increase_mean = 23122.17     Experience_increase_std = 29084.09
Converged on iter: 15
```

```
/tmp/ipykernel_21505/4291314431.py:27: RuntimeWarning: divide by zero
encountered in divide
  experience_increase_mean = np.mean(salary / years)
/tmp/ipykernel_21505/4291314431.py:28: RuntimeWarning: divide by zero
encountered in divide
  experience_increase_std = np.std(salary / years)
```

[ ]:

```python
# from https://gist.github.com/jonathanagustin/b67b97ef12c53a8dec27b343dca4abba
# install can take a minute

import os
# @title Convert Notebook to PDF. Save Notebook to given directory
NOTEBOOKS_DIR = "/content/drive/My Drive/CS441/24SP/hw2" # @param {type:
↪"string"}
NOTEBOOK_NAME = "CS441_SP24_HW2_Solution.ipynb" # @param {type:"string"}
#------------------------------------------------------------------------------#
from google.colab import drive
drive.mount("/content/drive/", force_remount=True)
NOTEBOOK_PATH = f"{NOTEBOOKS_DIR}/{NOTEBOOK_NAME}"
assert os.path.exists(NOTEBOOK_PATH), f"NOTEBOOK NOT FOUND: {NOTEBOOK_PATH}"
!apt install -y texlive-xetex texlive-fonts-recommended texlive-plain-generic >␣
↪/dev/null 2>&1
!jupyter nbconvert "$NOTEBOOK_PATH" --to pdf > /dev/null 2>&1
NOTEBOOK_PDF = NOTEBOOK_PATH.rsplit('.', 1)[0] + '.pdf'
assert os.path.exists(NOTEBOOK_PDF), f"ERROR MAKING PDF: {NOTEBOOK_PDF}"
print(f"PDF CREATED: {NOTEBOOK_PDF}")
```