CS412, Spring 2021

Assignment 2

Due: April 30th, 2021, 23:59, Blackboard

Instructions:

- The main submission of Assignment 2 must be uploaded on Blackboard. Your submission will be graded via Blackboard platform.
- The answers to the questions are integrated into a pdf file. Your PDF file should be named as **"ID_name(Chinese) _Assigment2.pdf"**. For example, if your name is 张三, name your file as "3xxx_张三_Assignment2.pdf".
- You need to upload **"ID_name(Chinese) _Assigment2.pdf" and "HOG.py".**
- Type out your solutions on a separate blank file (using Word, Latex, markdown, etc.), and do not forget to convert the file to PDF extension before submitting. You should not type out on the original pdf file.
- Every student has 3 late days in total for the assignments.
- You are to complete this assignment individually. We require you to:
  - Not explicitly tell each other the answers
  - Not to copy answers
  - Not to allow your answers to be copied

Question 1. (8 points)

Consider a binary classification problem with the following set of attributes and attribute values:

- Air Conditioner = { Working, Broken }
- Engine = { Good, Bad }
- Mileage = { High, Medium, Low }
- Rust = { Yes, No }

Suppose a rule-based classifier produces the following rule set:

$\text{Mileage} = \text{High} \longrightarrow \text{Value} = \text{Low}$
$\text{Mileage} = \text{Low} \longrightarrow \text{Value} = \text{High}$
$\text{Air Conditioner} = \text{Working}, \text{Engine} = \text{Good} \longrightarrow \text{Value} = \text{High}$
$\text{Air Conditioner} = \text{Working}, \text{Engine} = \text{Bad} \longrightarrow \text{Value} = \text{Low}$
$\text{Air Conditioner} = \text{Broken} \longrightarrow \text{Value} = \text{Low}$

(a) Are the rules mutually exclusive?

(b) Is the rule set exhaustive?

(c) Is ordering needed for this set of rules?

(d) Do you need a default class for the rule set?

Question 2. (10 points)

Consider the data set shown in Table 1

Table 1

Record	A	B	C	Class
1	0	0	0	+
2	0	0	1	−
3	0	1	1	−
4	0	1	1	−
5	0	0	1	+
6	1	0	1	+
7	1	0	1	−
8	1	0	1	−
9	1	1	1	+
10	1	0	1	+

(a) Estimate the conditional probabilities for $P(\text{A}|+)$, $P(\text{B}|+)$, $P(\text{C}|+)$, $P(\text{A}|-)$, $P(\text{B}|-)$, and $P(\text{C}|-)$.

(b) Use the estimate of conditional probabilities given in the previous question to predict the class label for a test sample ($A = 0$, $B = 1$, $C = 0$) using the naive Bayes approach.

(c) Estimate the conditional probabilities using the m-estimate approach, with $p = 1/2$ and $m = 4$.

(d) Repeat part (b) using the conditional probabilities given in part (c).

(e) Compare the two methods for estimating probabilities. Which method is better and why?

Question 3. (12 points)

The figure below illustrates the Bayesian belief network for the data set shown in Table 2 (Assume that all the attributes are binary).

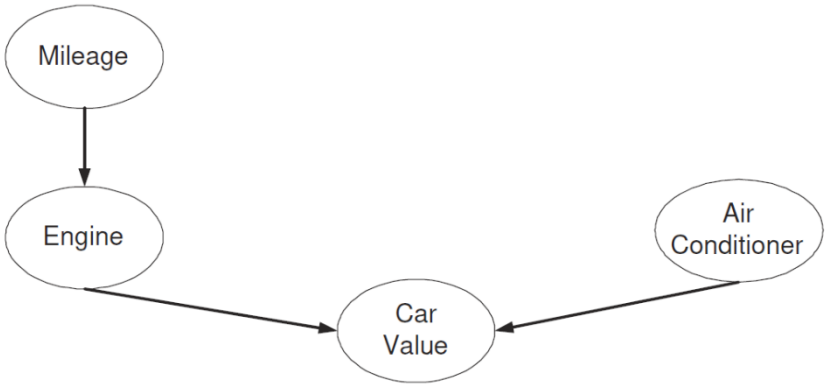


Figure. Bayesian belief network

Table 2. Data set for Exercise 2.

Mileage	Engine	Air Conditioner	Number of Records with Car Value=Hi	Number of Records with Car Value=Lo
Hi	Good	Working	3	4
Hi	Good	Broken	1	2
Hi	Bad	Working	1	5
Hi	Bad	Broken	0	4
Lo	Good	Working	9	0
Lo	Good	Broken	5	1
Lo	Bad	Working	1	2
Lo	Bad	Broken	0	2

(a) Draw the probability table for each node in the network.

(b) Use the Bayesian network to compute P (Engine = Bad, Air Conditioner = Broken).

Question 4. (8 points)

Consider the one-dimensional data set shown in Table.

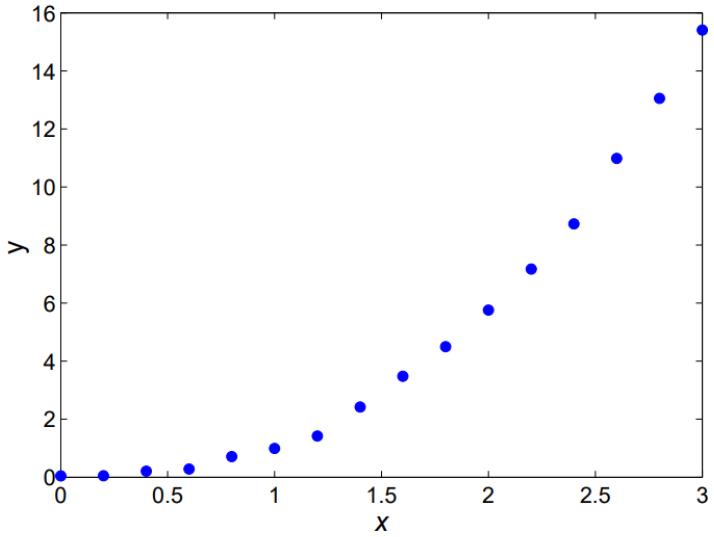
x	0.5	3.0	4.5	4.6	4.9	5.2	5.3	5.5	7.0	9.5
y	−	−	+	+	+	−	−	+	−	−

(a) Classify the data point x = 5.0 according to its 1-, 3-, 5-, and 9-nearest neighbors (using majority vote).

(b) Repeat the previous analysis using the distance-weighted voting approach ($w = 1/d^2$).

Question 5. (12 points)

We will design a gradient descent approach for fitting some data that is given to us. Suppose we have *N* input-output pairs $(x_i, y_i)_{i=1}^{N}$. Our goal is to find the parameters that predict the output *y* from the input *x* according to some function $y = f(x)$. The pairs are plotted in the graph below.



(a) Just by looking at the plot intuitively, which one of the following is the best choice for a function $y = f(x)$ in order to fit the given data? Assume *w* is some parameter.

1. $y = wx$
2. $y = x^w$
3. $y = \sqrt[w]{x}$

(b) For any setting of *w*, we can measure how well a function fits the data. According to your function *f* above, write down the sum-of-squared-error function *E* between predictions *y* and inputs *x*.

(c) The parameter *w* can be determined iteratively using gradient descent. For the error function you formulated above, derive the gradient descent update rule $w \leftarrow w - \alpha \frac{dE}{dw}$ and write it down below.

Question 6. (50 points)

- You will complete HOG.py that contains the following functions:
  - `extract_hog`
  - `get_differential_filter`
  - `filter_image`
  - `get_gradient`
  - `build_histogram`
  - `get_block_descriptor`

The code can be downloaded from the Blackboard (HOG.zip)

The function that does not comply with its specification will not be graded (no credit).

- The code must be run with Python 3 interpreter.
- Required python packages: numpy, matplotlib, and opencv.
  - numpy & matplotlib: https://scipy.org/install.html
  - opencv: https://pypi.org/project/opencv-python/
- We provide a visualization code for HOG. The resulting HOG descriptor must be able to be visualized with the provided code:
  `def visualize_hog(im, hog, cell_size, block_size)`
- Please place the code (HOG.py) as well as a one-page summary write-up with resulting visualization (in pdf format; more than 1-page assignment will be automatically returned.) into a folder, compress it, and submit it.

Reference:

- https://lilianweng.github.io/lil-log/2017/10/29/object-recognition-for-dummies-part-1.html#histogram-of-oriented-gradients-hog
- https://learnopencv.com/histogram-of-oriented-gradients/

Submit

- Upload the HOG.py to the Blackboard
- 1-page summary write-up with resulting visualization for Question6.

5.1 HOG

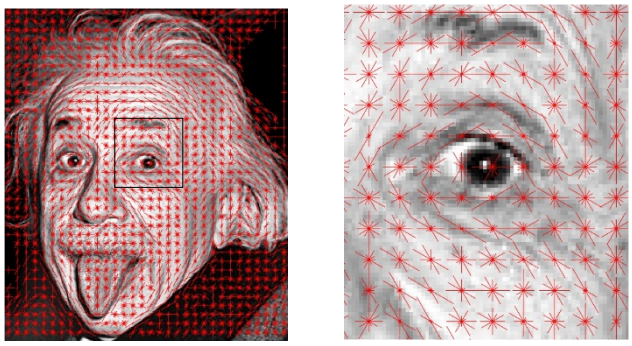


Figure 1: Histogram of oriented gradients. HOG feature is extracted and visualized for (a) the entire image and (b) zoom-in image. The orientation and magnitude of the red lines represent the gradient components in a local cell.

In this part, you will implement a variant of HOG (Histogram of Oriented Gradients) in Python proposed by Dalal and Trigg (2015 Longuet-Higgins Prize Winner). Given an input image, your algorithm will compute the HOG feature and visualize it as shown in Figure 1 (the line directions are perpendicular to the gradient to show edge alignment). The orientation and magnitude of the red lines represent the gradient components in a local cell.

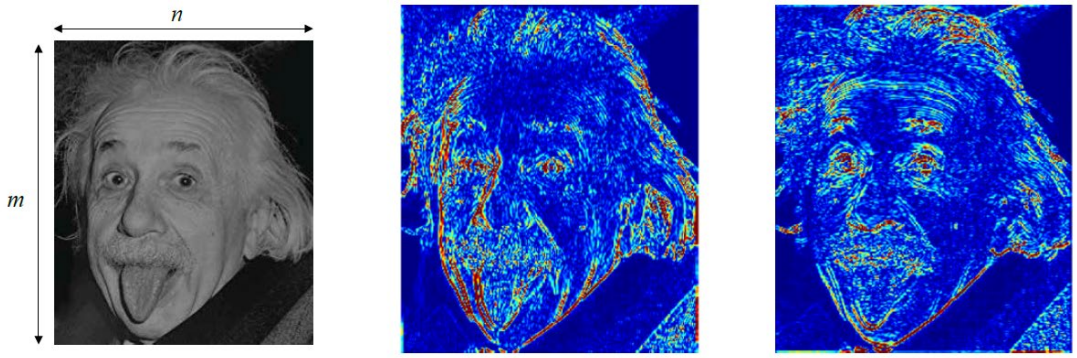
```
def extract_hog(im):
  ...
  return hog
```

Input: input gray-scale image with uint8 format.
Output: HOG descriptor.
Description: You will compute the HOG descriptor of input image `im`. The pseudo-code can be found below:

Algorithm 1 HOG

1: Convert the gray-scale image to `float` format and normalize to range [0, 1].
2: Get differential images using `get_differential_filter` and `filter_image`
3: Compute the gradients using `get_gradient`
4: Build the histogram of oriented gradients for all cells using `build_histogram`
5: Build the descriptor of all blocks with normalization using `get_block_descriptor`
6: Return a long vector (`hog`) by concatenating all block descriptors.

5.2 Image filtering



(a) Input image (b) Differential along x direction (c) Differential along y direction

Figure 2: (a) Input image dimension. (b-c) Differential image along x and y directions.

```
def get_differential_filter():
  ...
  return filter_x, filter_y
```

Input: none.

Output: `filter_x` and `filter_y` are 3×3 filters that differentiate along x and y directions, respectively.

Description: You will compute the gradient by differentiating the image along x and y directions. This code will output the differential filters.

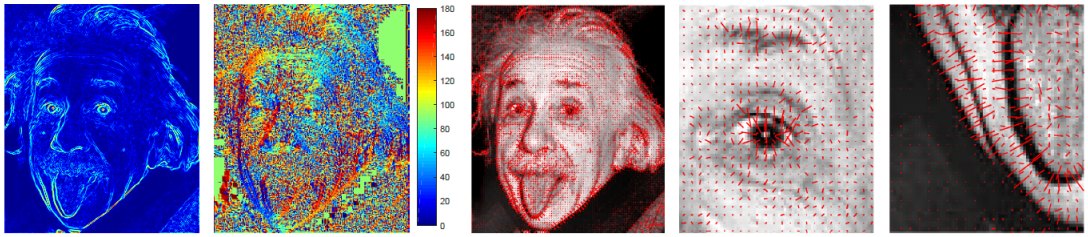
```
def filter_image(im, filter):
  ...
  return im_filtered
```

Input: `im` is the grayscale $m \times n$ image (Figure 2(a)) converted to float format and filter is a filter ($k \times k$ matrix)

Output: `im_filtered` is $m \times n$ filtered image. You may need to pad zeros on the boundary on the input image to get the same size filtered image.

Description: Given an image and filter, you will compute the filtered image. Given the two functions above, you can generate differential images by visualizing the magnitude of the filter response as shown in Figure 2(b) and 2(c).

5.3 Gradient Computation



(a) Magnitude (b) Angle (c) Gradient (d) Zoomed eye (e) Zoomed neck

Figure 3: Visualization of (a) magnitude and (b) orientation of image gradients. (c-e) Visualization of gradients at every 3rd pixel (the magnitudes are re-scaled for illustrative purpose.).

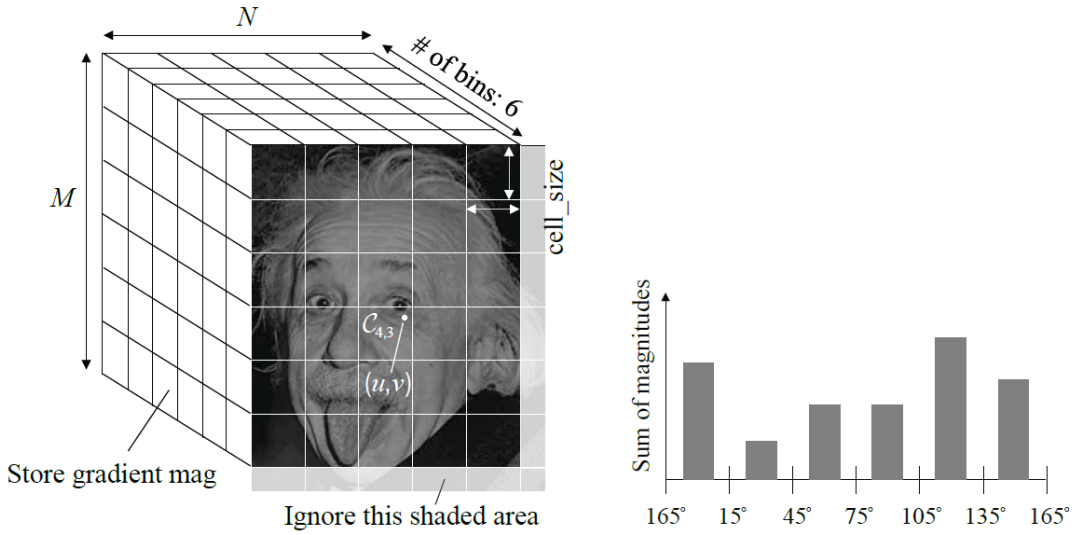
```
def get_gradient(im_dx, im_dy):
  ...
  return grad_mag, grad_angle
```

Input: `im_dx` and `im_dy` are the x and y differential images (size: $m \times n$).

Output: `grad_mag` and `grad_angle` are the magnitude and orientation of the gradient images (size: $m \times n$). Note that the range of the angle should be $[0, \pi)$, i.e., unsigned angle ($\theta == \theta + \pi$).

Description: Given the differential images, you will compute the magnitude and angle of the gradient. Using the gradients, you can visualize and have some sense with the image, i.e., the magnitude of the gradient is proportional to the contrast (edge) of the local patch and the orientation is perpendicular to the edge direction as shown in Figure 3.

5.4 Orientation Binning



(a) ori_histo (b) Histogram per cell

Figure 4: (a) Histogram of oriented gradients can be built by (b) binning the gradients to the corresponding bin.

```
def build_histogram(grad_mag, grad_angle, cell_size):
  ...
  return ori_histo
```

Input: `grad_mag` and `grad_angle` are the magnitude and orientation of the gradient images (size: $m \times n$); cell_size is the size of each cell, which is a positive integer.

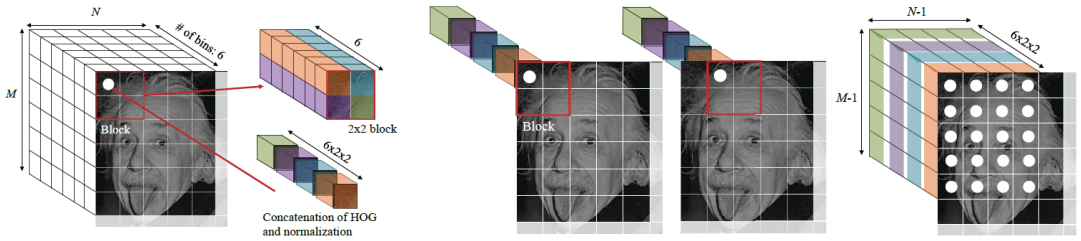
Output: `ori_histo` is a 3D tensor with size $M \times N \times 6$ where M and N are the number of cells along y and x axes, respectively, i.e., $\mathrm{M} = \lfloor m/\mathrm{cell\_size} \rceil$ and $N = \lfloor n/cell\_size \rceil$ where $\lfloor \rceil$ is the round-off operation as shown in Figure 4(a).

Description: Given the magnitude and orientation of the gradients per pixel, you can build the histogram of oriented gradients for each cell.

$$ori\_histo(i,j,k) = \sum_{(u,v)\in C_{i,j}} grad\_mag(u,v) \;\; if \;\; grad\_angle(u,v) \in \Theta_k$$

Where $C_{i,j}$ is a set of x and y coordinates within the (i, j) cell, and Θ_k is the angle range of each bin, e.g., $\Theta_1 = [165°, 180°) \cup [0°, 15°)$, $\Theta_2 = [15°, 45°)$, $\Theta_3 = [45°, 75°)$, $\Theta_4 = [75°, 105°)$, $\Theta_5 = [105°, 135°)$, and $\Theta_6 = [135°, 165°)$.Therefore, `ori_histo(i, j, :)` returns the histogram of the oriented gradients at (i, j) cell as shown in Figure 4(b). Using the `ori_histo`, you can visualize HOG per cell where the magnitude of the line proportional to the histogram as shown in Figure1. The typical `cell_size` is 8.

5.5 Block Normalization



(a) Block descriptor (b) Block overlap with stride 1

Figure 5: HOG is normalized to account illumination and contrast to form a descriptor for a block. (a) HOG within (1,1) block is concatenated and normalized to form a long vector of size 24. (b) This applies to the rest block with overlap and stride 1 to form the normalized HOG.

```
def get_block_descriptor(ori_histo, block_size):
  ...
  return ori_histo_normalized
```

Input: `ori_histo` is the histogram of oriented gradients without normalization. `block_size` is the size of each block (e.g., the number of cells in each row/column), which is a positive integer.

Output: `ori_histo_normalized` is the normalized histogram (size: (M-(block_size-1))×(N-(block_size-1))×(6×block_size^2).

Description: To account for changes in illumination and contrast, the gradient strengths must be locally normalized, which requires grouping the cells together into larger, spatially connected blocks (adjacent cells). Given the histogram of oriented gradients, you apply L_2 normalization as follow:

1. Build a descriptor of the first block by concatenating the HOG within the block. You can use `block_size`=2, i.e., 2×2 block will contain 2×2×6 entries that will be concatenated to form one long vector as shown in Figure 6(c).
2. Normalize the descriptor as follow:

$$\hat{h}_i = \frac{h_i}{\sqrt{\sum_i {h_i}^2 + e^2}}$$

   where h_i is the i^{th} element of the histogram and h_i is the normalized histogram, e is the normalization constant to prevent division by zero (e.g., $e = 0.001$)
3. Assign the normalized histogram to `ori_histo_normalized(1,1)` (white dot location in Figure 5(a)).
4. Move to the next block `ori_histo_normalized(1,2)` with the stride 1 and iterate1-3 steps above.

The resulting `ori_histo_normalized` will have the size of $(M-1)\times(N-1)\times 24$