challenge-1

March 15, 2024

1 C1.1 Evaluation on Object Detection

1.1 Team: *Paradox*

1.2 Teammates: 'kjamunap' , 'sriramna' , 'leonli'

1.3 Introduction

- Task: Evaluate the average precision of the data collected from SafeBench.
- Submission: the plotted P-R curve, as well as the average precision at different IoU threshold level.

[51]:
```
import joblib
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
```

[52]:
```
!git clone https://github.com/HenryLHH/24784-c1-data.git
!cp /content/24784-c1-data/results.pkl ./
!rm -r /content/24784-c1-data/
```

```
Cloning into '24784-c1-data'…
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 4 (delta 0), reused 4 (delta 0), pack-reused 0
Receiving objects: 100% (4/4), 31.64 KiB | 31.64 MiB/s, done.
```

1.3.1 Load the Detection Results from SafeBench_v2

[53]:
```
inputs = joblib.load('results.pkl')  # move the files to your current folder
inputs.keys() # dict_keys(['image_id', 'predicted_class', 'ground_truth_bbox',␣
 ↪'predicted_bbox', 'conf_scores', 'num_labels'])
```

[53]:
```
dict_keys(['image_id', 'predicted_class', 'ground_truth_bbox', 'predicted_bbox',
'conf_scores', 'num_labels'])
```

1.3.2 Compute the IoU

[54]:
```python
def box_area(box):
    """ Compute the box area, given all the vertices
    # Arguments
        box:  (4, N) ndarray
    # Returns
        areas: (N, ) ndarray
    """
    assert box.shape[0] == 4

    width = np.abs(box[0] - box[2])
    height = np.abs(box[1] - box[3])

    areas = width * height

    # TODO, compute the rectangle areas based on the vertices input
    return areas

def box_iou(box1, box2):
    '''
      Compute the iou between box1 and box2
      ONLY consider the SINGLE ground truth, which is a simplified case
      box1: (N, 4) ndarray
      box2: (N, 4) ndarray
      return: (N, ) iou scores
    '''
    eps = 1e-7

    assert box1.ndim == 2 and box2.ndim == 2, "Inputs must be 2-dimensional␣
 ↪arrays"
    assert box1.shape[1] == 4 and box2.shape[1] == 4, "Each box should have 4␣
 ↪coordinates"
    a1, a2 = np.split(box1, 2, axis=1)
    b1, b2 = np.split(box2, 2, axis=1)

    # Calculate the coordinates of intersection rectangle
    x1 = np.maximum(a1[:, 0], b1[:, 0])
    y1 = np.maximum(a1[:, 1], b1[:, 1])
    x2 = np.minimum(a2[:, 0], b2[:, 0])
    y2 = np.minimum(a2[:, 1], b2[:, 1])


    # Compute the width and height of intersection rectangle
    inter_width = np.maximum(x2 - x1, 0)
    inter_height = np.maximum(y2 - y1, 0)
```

```
    # Compute the area of intersection
    inter = inter_width * inter_height

    # Compute the area of union
    box1_area = (box1[:, 2] - box1[:, 0]) * (box1[:, 3] - box1[:, 1])
    box2_area = (box2[:, 2] - box2[:, 0]) * (box2[:, 3] - box2[:, 1])
    union = box1_area + box2_area - inter

    return inter / (union + eps)
```

1. In the `box_area` function, the area of each bounding box is calculated based on its vertices. We a numpy array box of shape (4, N), where N is the number of bounding boxes.
2. Then we calculate the width and height of each box by taking the differences between the x and y coordinates of opposite vertices, and then multiply these values to get the area.
3. The `box_iou` function, computes IoU scores between pairs of bounding boxes.
4. We take two numpy arrays **box1** and **box2**, each of shape (N, 4), representing N bounding boxes with four coordinates (x1, y1, x2, y2).
5. Calculating the coordinates of the intersection rectangle by finding maximum x and y coordinates of top-left corner and minimum x and y coordinates of bottom-right corner.
6. Then we find width and height of the intersection rectangle and calculate the area of intersection.
7. The area of the union of the two bounding boxes and divides the area of intersection by the area of the union to obtain the IoU score.
8. The IoU loss is defined as the complement of the IoU scores.

[55]:
```
# TODO: get the box iou scores from the function you implemented above
iou_scores = box_iou(inputs['ground_truth_bbox'], inputs['predicted_bbox'])
# Compute IoU loss
iou_loss = 1.0 - iou_scores

print("IoU Loss values:")
print(iou_loss)
```

```
IoU Loss values:
[1. 1. 1. … 1. 1. 1.]
```

[56]:
```
average_loss = np.mean(iou_loss)

# Print the average loss
print("Average IoU Loss:", average_loss)
```

```
Average IoU Loss: 0.9238258051395144
```

[57]:
```
# Build your dataframe from the dictionary
input_dict = {
        'image_id': inputs['image_id'],
        'predicted_class': inputs['predicted_class'],
        'conf_scores': inputs['conf_scores'],
```

```
        'iou_scores': iou_scores,
      }
df = pd.DataFrame.from_dict(input_dict)
df
```

[57]:

	image_id	predicted_class	conf_scores	iou_scores
0	0	None	-1.000000	0.00000
1	1	car	0.671808	0.00000
2	1	car	0.563249	0.00000
3	1	car	0.452749	0.80052
4	1	car	0.428004	0.00000
…	…	…	…	…
1631	91	train	0.219339	0.00000
1632	91	bus	0.173378	0.00000
1633	91	person	0.101061	0.00000
1634	91	sink	0.088729	0.00000
1635	91	dining table	0.073306	0.00000

[1636 rows x 4 columns]

[58]:
```
df = df[df.conf_scores >= 0] # drop all the frames without detection (conf␣
 ↪scores = -1)

# get all the detection results of stop signs
df_stopsign = df.loc[(df.predicted_class=='stopsign')]
df_stopsign
```

[58]:

	image_id	predicted_class	conf_scores	iou_scores
160	7	stopsign	0.071092	0.812147
345	15	stopsign	0.052572	0.823549
712	33	stopsign	0.110136	0.814429
731	34	stopsign	0.159796	0.844884
772	36	stopsign	0.128940	0.850620
…	…	…	…	…
1606	88	stopsign	0.119673	0.025573
1610	89	stopsign	0.997623	0.932374
1622	90	stopsign	0.995437	0.923847
1628	90	stopsign	0.052746	0.032057
1629	91	stopsign	0.995099	0.614191

[66 rows x 4 columns]

1.4 Compute the Avereage Precision

[59]:
```
def interp_ap(recall, precision, method = 'interp'):
    """ Compute the average precision, given the recall and precision curves
    # Arguments
        recall:    The recall curve (list)
        precision: The precision curve (list),
        methods: 'continuous', 'interp'
    # Returns
        Average precision, precision curve, recall curve
    """

    # TODO: Append sentinel values to beginning and end
    # Recall should start with 0.0 and end with 1.0
    # Precision should start with 1.0 and end with 0.0

    appended_recall = np.concatenate(([0.0], recall, [1.0]))
    appended_prec_input = np.concatenate(([1.0], precision, [0.0]))

    # Compute the precision envelope
    appended_prec = np.flip(np.maximum.accumulate(np.
↪flip(appended_prec_input)))   # get the p(r) = \max_{r<=r'} p(r')

    # Integrate area under curve
    if method == 'interp':
        x = np.linspace(0, 1, 101)  # 101-point interp (COCO)
        ap = np.trapz(np.interp(x, appended_recall, appended_prec), x)      #␣
↪get the average precision based on the areas under interpolated curves
    else:  # 'continuous'
        i = np.where(appended_recall[1:] != appended_recall[:-1])[0]  # points␣
↪where x axis (recall) changes
        ap = np.sum((appended_recall[i + 1] - appended_recall[i]) *␣
↪appended_prec[i + 1])  # area under curve

    return ap, appended_prec_input, appended_prec, appended_recall
```

[60]:
```
def compute_ap(df, num_gt, iou_thres):
    """ Compute the average precision, given the recall and precision curves
    # Arguments
        df:         DataFrame inputs containing predicted classes, iou_scores,␣
↪and confidence
        num_gt:     The precision curve (list),
        iou_thres:  IoU threshold of True Positive (TP) detection
    # Returns
        Average precision, precision curve, recall curve
    """
```

```
    df = df.sort_values(by='conf_scores', ascending=False)    # TODO sort all␣
↪the data with confidence scores 'conf_scores'
    tp_fp = np.arange(1, len(df)+1, 1)
    tp = ((df['iou_scores'] >= iou_thres) & (df['predicted_class'] ==␣
↪'stopsign')).cumsum().values    # get the true positive sets
    precision = tp / tp_fp      # array (N, )
    recall = tp / num_gt        # array (N, )

    ap, prec_raw, prec, recall = interp_ap(recall, precision)           # get␣
↪the AP and returned curve for plot

    return ap, prec_raw, prec, recall
```

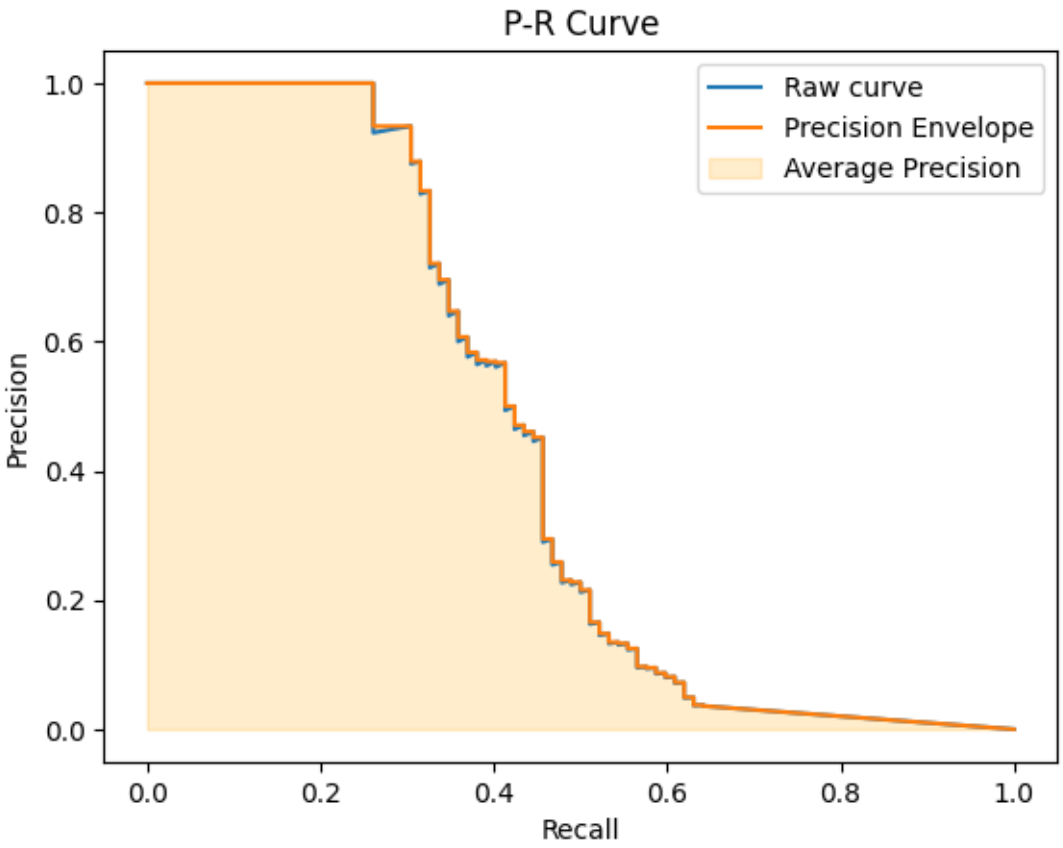
1.5 Visualization

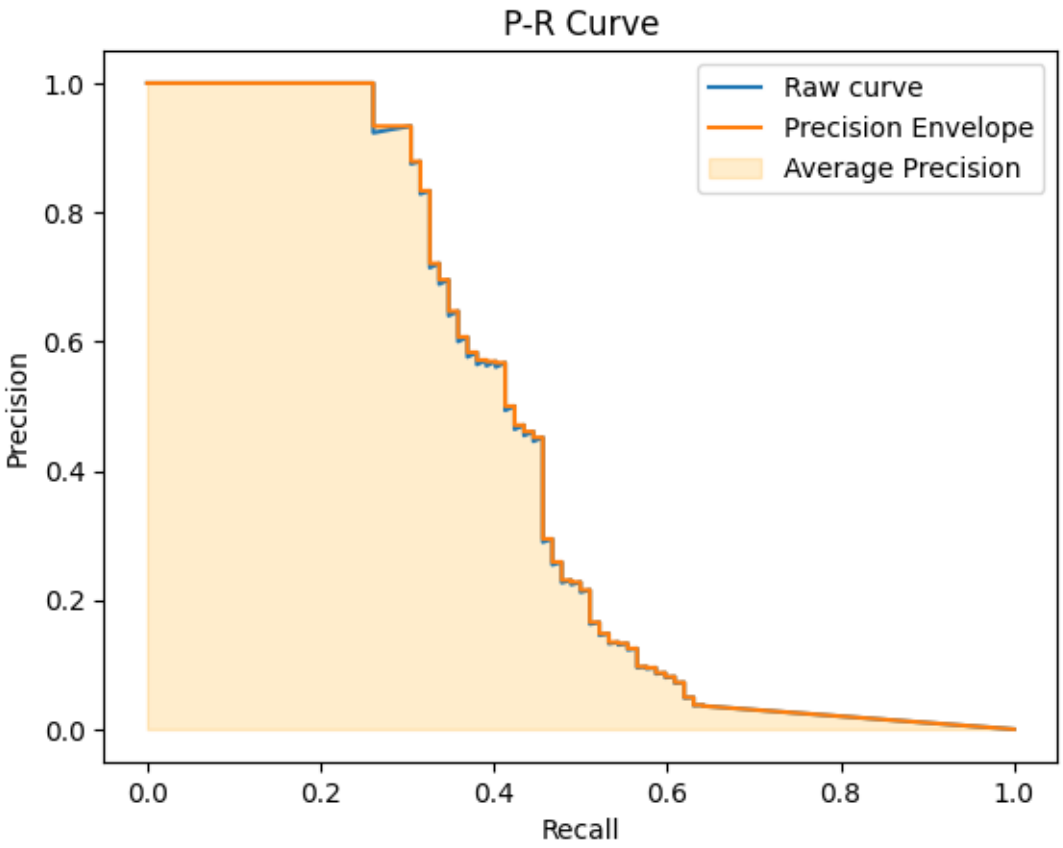
[61]:
```
def plot_pr_curves(recall, prec_raw, prec):
    """ Visualize the P-R curves
        Visualize the areas under P-R curves
    """
    plt.figure()
    plt.plot(recall, prec_raw)
    plt.plot(recall, prec)
    plt.fill_between(recall, np.zeros_like(recall), prec, color='orange',␣
↪alpha=0.2)
    plt.legend(['Raw curve', 'Precision Envelope', 'Average Precision'])
    plt.title('P-R Curve')
    plt.xlabel('Recall')
    plt.ylabel('Precision')
```

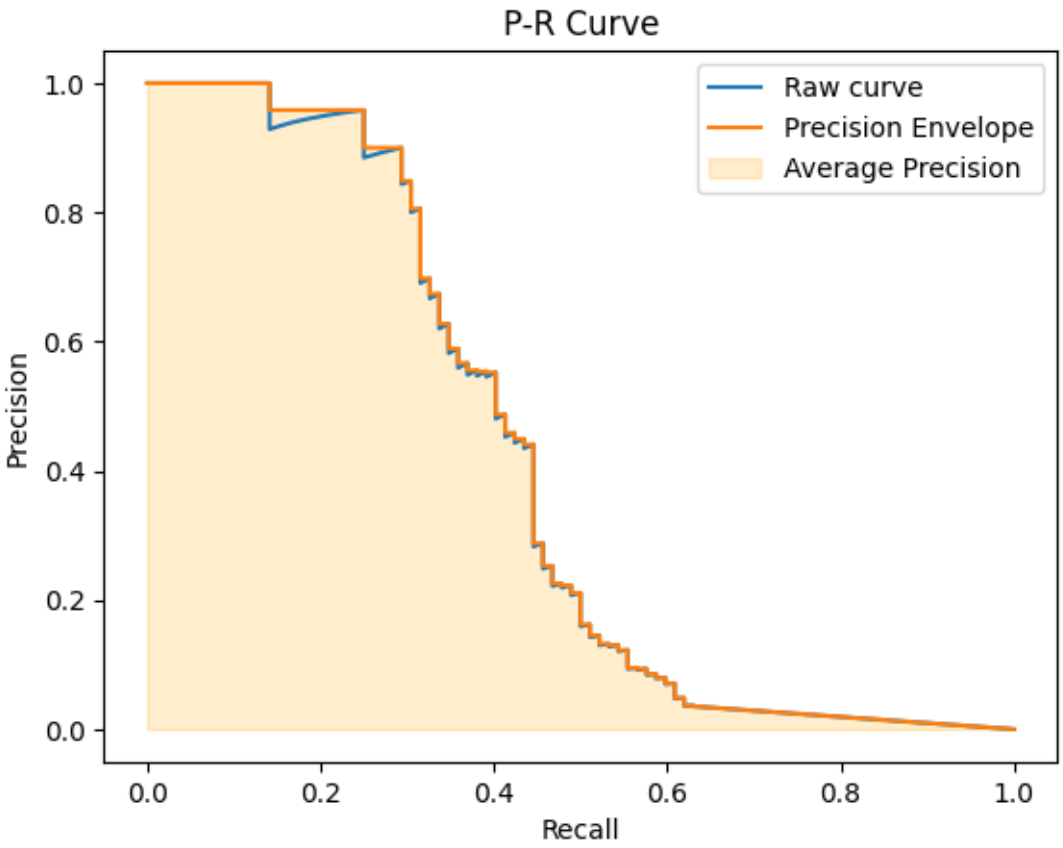
1.6 Execute the AP calculation

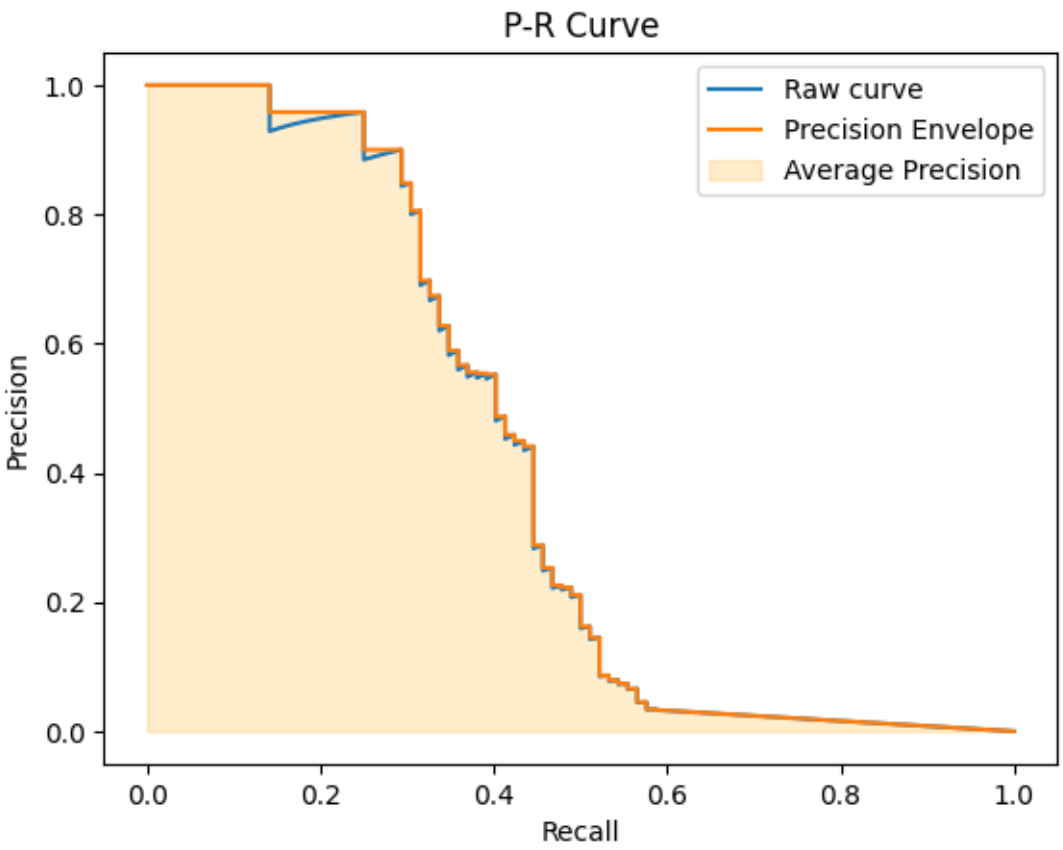
[62]:
```
# Traverse over IoU threshold, get AP@[0.5:0.1:0.9]

for iou_thres in np.arange(0.5, 1.0, 0.1):
    ap, prec_raw, prec, recall = compute_ap(df, inputs['num_labels'], iou_thres)
    print('IoU Threshold: {:.2f}'.format(iou_thres), '  |    AP@{:.2f}'.
↪format(iou_thres), ap)
    plot_pr_curves(recall, prec_raw, prec)
```

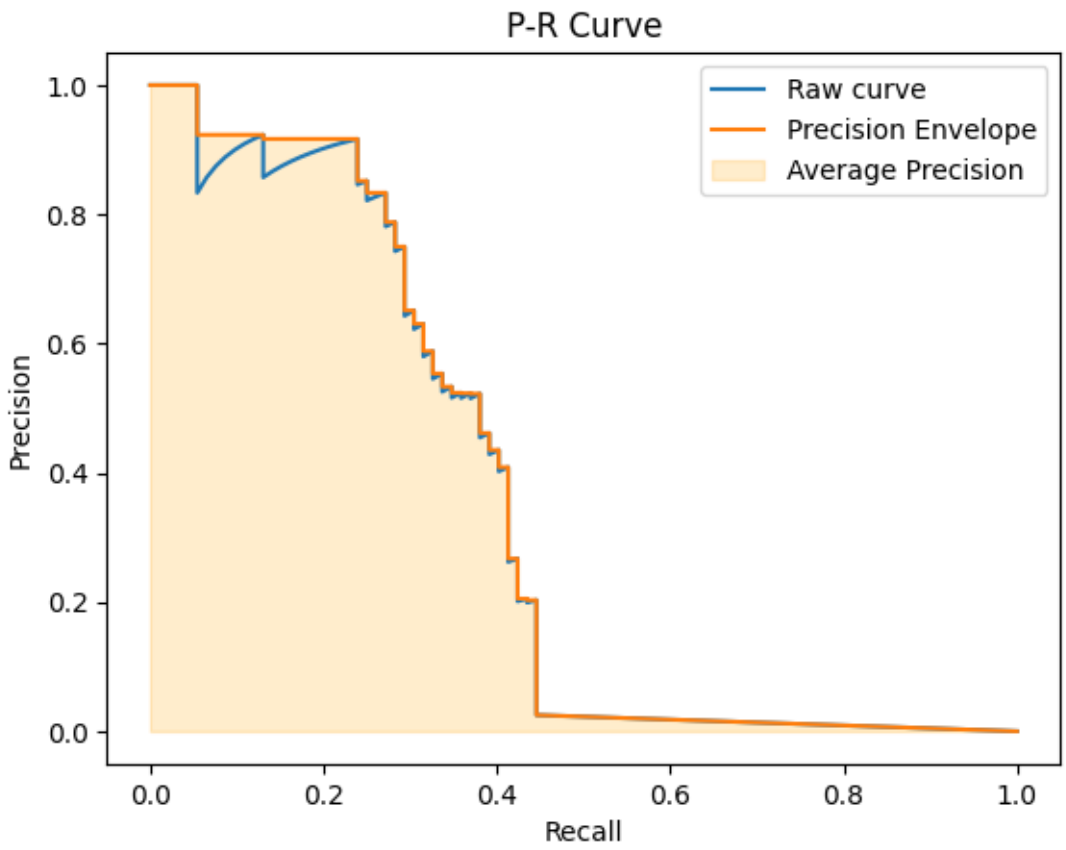
```
IoU Threshold: 0.50   |    AP@0.50 0.4272559074361477
IoU Threshold: 0.60   |    AP@0.60 0.4272559074361477
IoU Threshold: 0.70   |    AP@0.70 0.40686292416368364
IoU Threshold: 0.80   |    AP@0.80 0.4018855902817313
IoU Threshold: 0.90   |    AP@0.90 0.3466670309329718
```


P-R Curve
Raw curve
Precision Envelope
Average Precision
Precision
Recall
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0


P-R Curve
Raw curve
Precision Envelope
Average Precision
Precision
Recall
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

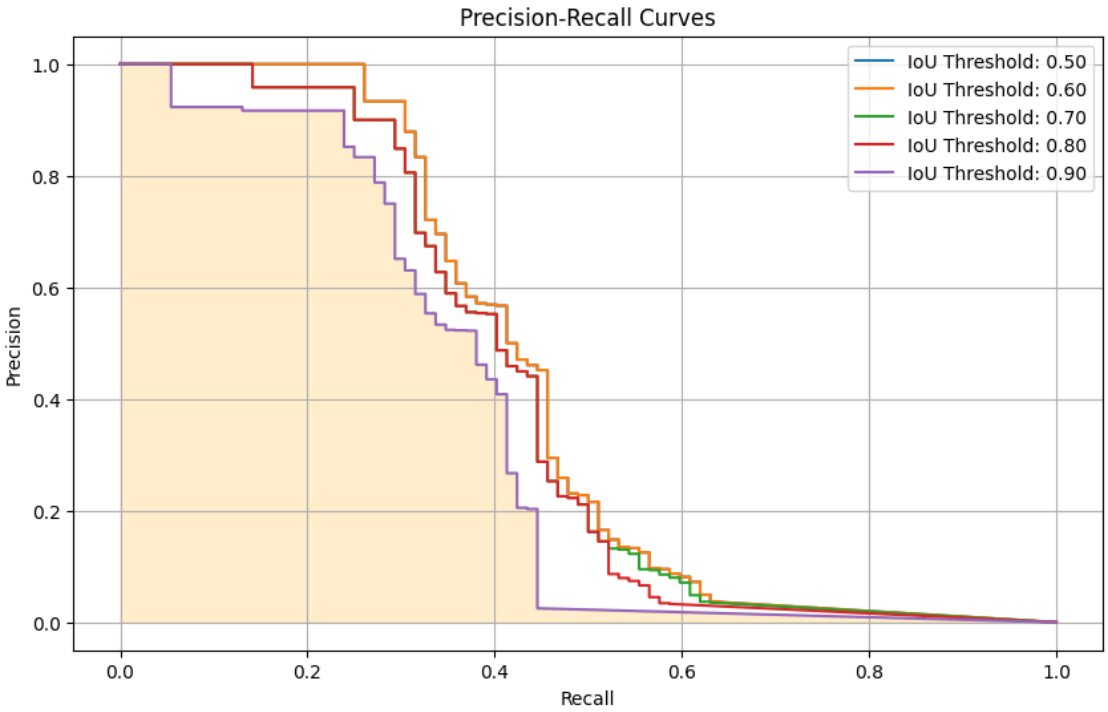

P-R Curve
Raw curve
Precision Envelope
Average Precision
Precision
Recall
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0


P-R Curve
Raw curve
Precision Envelope
Average Precision
Precision
Recall
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0



Precision-Recall curves with envelope curves

[63]:
```
for iou_thres in iou_thresholds:
    ap, _, _, _ = compute_ap(df, inputs['num_labels'], iou_thres)
    ap_values.append(ap)

plt.figure(figsize=(10, 6))

for iou_thres in iou_thresholds:
    _, prec_raw, prec, recall = compute_ap(df, inputs['num_labels'], iou_thres)
    plt.plot(recall, prec, label=f'IoU Threshold: {iou_thres:.2f}')

plt.fill_between(recall, np.zeros_like(recall), prec, color='orange', alpha=0.2)
plt.legend()
plt.title('Precision-Recall Curves')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.grid(True)
plt.show()
```



[64]:
```
iou_thresholds = np.arange(0.5, 1.0, 0.1)
ap_values = []

for iou_thres in iou_thresholds:
    ap, _, _, _ = compute_ap(df, inputs['num_labels'], iou_thres)
    ap_values.append(ap)

results_df = pd.DataFrame({
    '|IoU-Threshold | AP@IoU|': [f'| {iou:.2f} | {ap:.4f} |'
    for iou, ap in zip(iou_thresholds, ap_values)]
})

print(results_df.to_string(index=False))
```

```
|IoU-Threshold | AP@IoU|
       | 0.50 | 0.4273 |
       | 0.60 | 0.4273 |
       | 0.70 | 0.4069 |
       | 0.80 | 0.4019 |
       | 0.90 | 0.3467 |
```

IoU Threshold	AP@IoU
0.50	0.4273
0.60	0.4273

IoU Threshold	AP@IoU
0.70	0.4069
0.80	0.4019
0.90	0.3467

- The trend observed as the IoU threshold increases, a gradual decrease in the Average Precision (AP). From the results, as the IoU threshold increases from 0.50 to 0.90, the AP decreases from 0.4273 to 0.3467.
- As the IoU threshold increases, the number of true positives decreases.
- This is because only predicted bounding boxes with a higher overlap with the ground truth bounding boxes are counted as true positives.
- Bounding boxes with lower overlaps are considered false positives.
- With fewer true positives and large number of false positives, both precision and recall decrease as the IoU threshold increases.
- We know that precision usually decreases at a faster rate than recall, this affect we can see in a decrease in AP.
- Higher IoU thresholds require more precise localization of objects by the model.
- This can be challenging, especially in cases where objects have complex shapes.