1 Implementation of Quad nearest neighbors

1.1 Quad Tree

1. **The Quad tree data structure**[1]

   - It's a tree data structure have four children
   - Partition two dimension space recursively by dividing it into four quadrant
   - **Space subdivision rule**
     - Divide in the middle - can cause the points distribution unevenly(the rule used in this Implementation)
     - Divide by the median of the points within the current Quad tree
   - **Recursion stopping criteria**
     When the number of points within the current quadrant $< c$. C decided by the user(In this implementation, I used c = 5/10/20).
     Turned out different cs can cause the algorithm speed different by 10% ,i.e. from .5 s to .4 or so. In the final demo, I used 10 neither too small or too big comparing to the k(number of nearest neighbors).

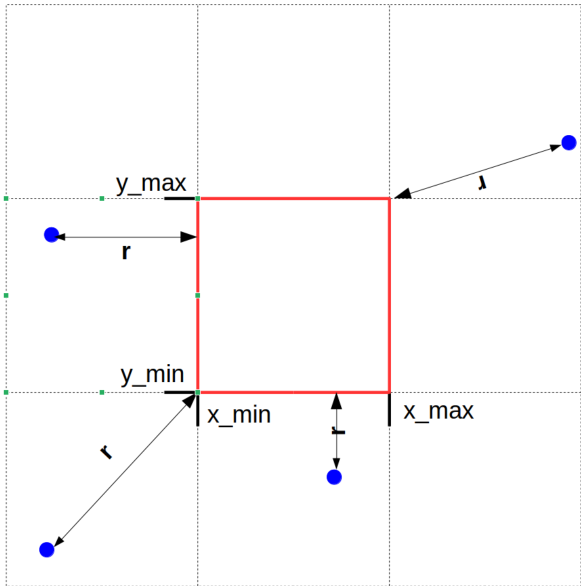
2. **Data structure implementation**

   Class built for the quad tree/point/rectangular and their attributes

   - **Points**: the points in the training set
     - x: x-axis
     - y: y-axis
     - cls_: the category of this point

```
class Point:
def __init__(self,x,y,_cls):
    self.x = x
    self.y = y
    self.cls_ = _cls
```

   - **Rect**: Rectangular store the shape and other attribute of the quadtree
     - x: rectangular center point x-axis
     - y: rectangular center point y-axis
     - width: the 1/2 width of the rectangular
     - height: the 1/2 height of the rectangular
     - **method** contain: to validate whether a point is inside of the rectangular



- **method** within: to validate whether the rectangular is with in the circle of center(point_x,point_y) and the radius r Note: this is calculated follow the figure below.[2]

```
class Rect:
    def __init__(self,x,y,width,height):
        self.x = x
        self.y = y
        self.width = width
        self.height = height
    def contain(self,point):
        return (point.x>=(self.x - self.width)) and \\
               (point.x <= (self.x +self.width)) and \\
               (point.y>=(self.y-self.height)) and \\
               (point.y<=(self.y + self.height))
    def within(self,point,d):
        def dist(x1,y1,x2,y2):
            return math.sqrt((x1-x2)**2+(y1-y2)**2)
        l_x = self.x - self.width
        b_y = self.y - self.height
        h_x = self.x + self.width
        t_y = self.y + self.height
        if point.x>=h_x and point.y>=t_y:
            return dist(h_x,t_y,point.x,point.y)<=d
        elif point.x>=h_x and point.y<=b_y:
            return dist(h_x,b_y,point.x,point.y)<d
        elif point.x>=h_x and point.y<t_y and point.y >b_y:
```

```
            return dist(h_x,0,point.x,0)<=d
        elif point.x<=l_x and point.y<=b_y:
            return dist(l_x,b_y,point.x,point.y)<d
        elif point.x<=l_x and point.y>=t_y:
            return dist(l_x,t_y,point.x,point.y)<d
        elif point.x<=l_x and point.y>=b_y:
            return dist(l_x,0,point.x,0)<d
        elif point.x>=l_x and point.x<=h_x and point.y>=t_y:
            return dist(0,t_y,0,point.y)<d
        elif point.x>=l_x and point.x<=h_x and point.y<=b_y:
            return dist(0,b_y,0,point.y)<d
        elif self.contain(point):
            return True
```

- **Quad Tree**:
  Attributes:
  - boundary: the rectangular that store the shape and center
  - capacity: the maximum number of points can be stored in this quadtree.
  - isleaf: denote whether the quadtree is subdivided or not.
  - points: the points that are stored in the quadtree, only is not empty when isleaf is True
  - northwest/southwest/northeast/southeast: the sub quadtree node stored by the relative direction
  - color the style of coloring for quadtree visualization(optional)

  Methods:
  - subdivide: subdivide the big quad tree into 4 sub quadtree and distribute the points into the corresponding sub quadtree

```
def subdivide(self):
    x = self.boudary.x
    y = self.boudary.y
    width = self.boudary.width
    height = self.boudary.height
    ne = Rect(x + width/2,y+height/2, width/2, height/2)
    nw = Rect(x - width/2,y+height/2, width/2, height/2)
    sw = Rect(x - width/2,y-height/2, width/2, height/2)
    se = Rect(x + width/2,y-height/2, width/2, height/2)
    self.northwest = quadTree(nw,\\
    [p for p in self.points if p.x<=x and p.y>=y],self.capacity)
    self.southwest = quadTree(sw,\\
    [p for p in self.points if p.x<=x and p.y<y],self.capacity)
    self.northeast = quadTree(ne,\\
    [p for p in self.points if p.x>x and p.y>=y],self.capacity)
    self.southeast = quadTree(se,\\
    [p for p in self.points if p.x>x and p.y<y],self.capacity)
```

  - construct: construct the quadtree if not reaching the stopping criteria then keep subdividing.
  - showfig: quadtree visualization

1.2 k Nearest Neighbor algorithm

1. **Steps**

Algorithm 1 Quad nearest neighbor

```
for p ∈ points do
    Initialize r = ∞, stack = [quad_tree_root],pnt = []
    while stack is not null do
        cur:= pop stack
        if cur is a leaf quad tree then
            for i ∈ points in the cur do
                if length pnt < k then append to pnt,update r
                else
                    if  if the distance between point i and the center p then
                        pop up the point in pnt that has the largest r_p
                        update r
                    end if
                end if
            end for
        end if
        if cur is not a leaf quad and the distance(p, cur.boundary) < r then
            append the children of cur to the stack
        end if
    end while
end forresult is the mode of the pnt list
```

Note: pnt is list for k nearest neighbors of each point,it's maintained as a heap, thus it can pop out the item with the max r and insert new point, with a time complexity of $O(logn)$

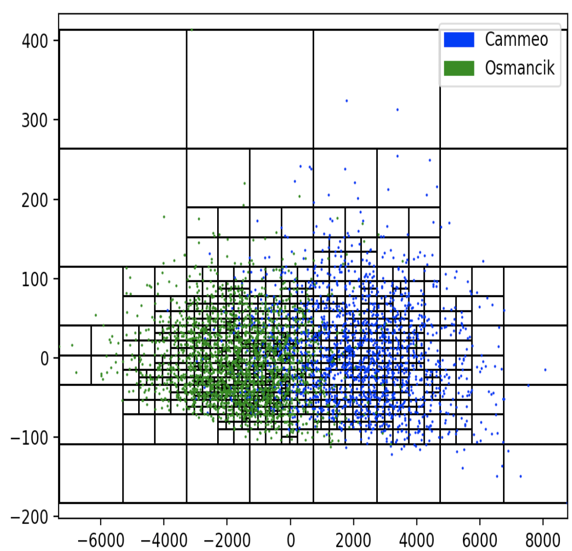
2. Implementation in python

```python
def knn(quad,pnt,k):
    res = []
    for p in tqdm(pnt):
        stack = [quad]
        r = (float('-inf'),"")
        pnt_ = []
        while len(stack):
            cur = stack.pop(-1)
            if cur.isleaf and cur.boudary.within(p,-r[0]):
```

```
                    for i in cur.points:
                        if len(pnt_)<k:
                            heapq.heappush(pnt_,\\
                            (-math.sqrt((i.x - p.x)**2+(i.y - p.y)**2),i.cls_))
                            r = heapq.nsmallest(1,pnt_)[0]
                        elif math.sqrt((i.x - p.x)**2+(i.y - p.y)**2)<-r[0]:
                            heapq.heappop(pnt_)
                            heapq.heappush(pnt_,\\
                            (-math.sqrt((i.x - p.x)**2+(i.y - p.y)**2),i.cls_))
                            r = heapq.nsmallest(1,pnt_)[0]
                elif not cur.isleaf:
                    if cur.boudary.within(p,-r[0]):
                        if cur.northwest:
                            stack.append(cur.northwest)
                        if cur.southeast:
                            stack.append(cur.southeast)
                        if cur.northeast:
                            stack.append(cur.northeast)
                        if cur.southwest:
                            stack.append(cur.southwest)
            res.append(mode([itr[1] for itr in pnt_]))
        return res
```

1.3 Results

Some outputs
1. scatter plot



2. The model time and the fitting(compared with naive knn)

```
100%|██████████| 762/762 [00:01<00:00, 683.93it/s]
quad tree knn time: 1.171703361s
naive knn time: 4.9777012570000005s
100%|██████████| 762/762 [00:01<00:00, 756.88it/s]
quad tree knn time: 1.0891073149999997s
naive knn time: 4.6406222889999995s
100%|██████████| 762/762 [00:00<00:00, 1273.61it/s]
quad tree knn time: 0.647694928s
naive knn time: 4.568587840999999s
100%|██████████| 762/762 [00:00<00:00, 2208.47it/s]
quad tree knn time: 0.3867595989999977s
naive knn time: 4.187944944000002s
100%|██████████| 762/762 [00:00<00:00, 1664.41it/s]
quad tree knn time: 0.49846629600000014s
naive knn time: 4.655969850000002s
########################################
When k = 1
----------------------------------------
The confusion matrix of quad tree nearest_neighbors
[[1224  406]
 [ 414 1766]]
----------------------------------------
The confusion matrix of naive k nearest_neighbors
[[1224  406]
 [ 414 1766]]
########################################
100%|██████████| 762/762 [00:01<00:00, 564.09it/s]
quad tree knn time: 1.3904680899999988s
naive knn time: 4.701068543000002s
100%|██████████| 762/762 [00:01<00:00, 543.60it/s]
quad tree knn time: 1.4436367650000008s
naive knn time: 4.6885418130000005s
100%|██████████| 762/762 [00:00<00:00, 1007.36it/s]
quad tree knn time: 0.8005380179999975s
naive knn time: 4.9679432859999935s
100%|██████████| 762/762 [00:00<00:00, 1352.74it/s]
quad tree knn time: 0.6335236950000009s
naive knn time: 4.583810810000003s
100%|██████████| 762/762 [00:00<00:00, 1134.19it/s]
quad tree knn time: 0.7136914870000055s
naive knn time: 4.685330620000002s
########################################
When k = 5
----------------------------------------
The confusion matrix of quad tree nearest_neighbors
[[1253  377]
 [ 313 1867]]
----------------------------------------
The confusion matrix of naive k nearest_neighbors
[[1253  377]
 [ 313 1867]]
########################################
```

3. Interpretation

The confusion matrix means for $k = 1$, of all the points, there are 414 entires that are Osmancik but were predicted as Cammeo wrongly and here are 406 entires that are Cammeo but were predicted as Osmancik wrongly, while 1224 Cammeo were correctly classfied as Cammeo,1766 Osmancik were correctly classfied as Osmancik.
The confusion matrix means for $k = 5$, of all the points, there are 313 entires that are Osmancik but were predicted as Cammeo wrongly and here are 377 entires that are Cammeo but were predicted as Osmancik wrongly, while 1253 Cammeo were correctly classfied as Cammeo,1867 Osmancik were correctly classfied as Osmancik.
Overall when $k = 5$, the accuracy is higher, this might because in case $k = 1$, the model is overfitted.

References

[1] Quad tree wikipidia. `https://www.wikiwand.com/en/Quadtree#:~:text=A%20quadtree%20is%20a%20tree,into%20four%20quadrants%20or%20regions.`

[2] Stack overflow. Compute shortest distance between point and a rectangle. `https://codereview.stackexchange.com/questions/175566/compute-shortest-distance-between-point-and-a-rectangl`.