Range and Similarity Queries σε ΒΔ κειμένων

Σεπτέμβριος 2022

Ζερβού Πουλχερία 1067511

Πανδής Αναστάσιος 1056287

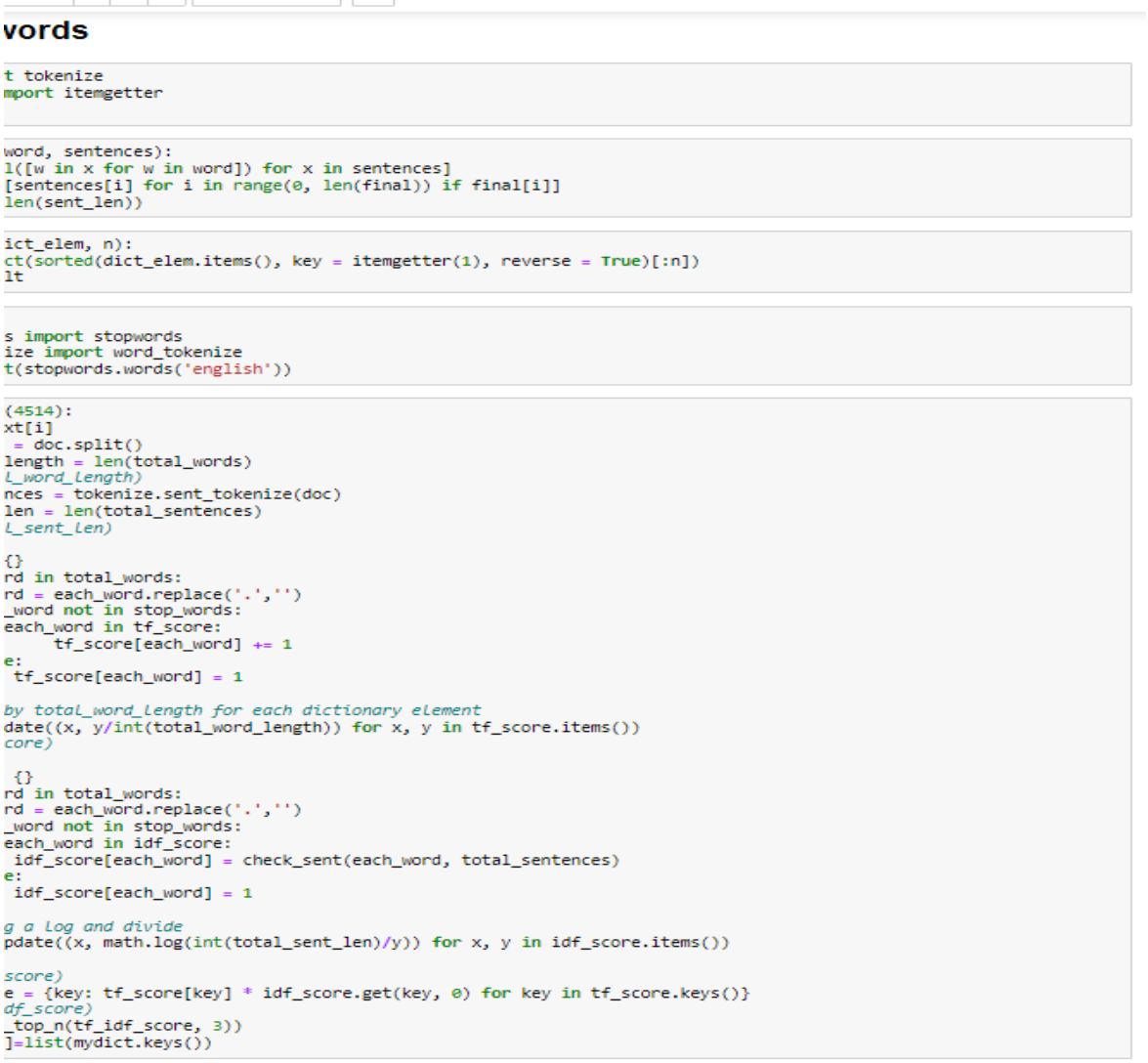
Πορτοκάλογλου Ανδρονίκη 1067539

Περιεχόμενα

1. Επιλογή και επεξεργασία δεδομένων

Για την υλοποίηση της εργασίας επιλέξαμε δεδομένα από το Kaggle.Συγκεκριμένα, διαλέξαμε νέα (news).Τα news αυτά έπρεπε κάπως να κάνουμε σημεία ώστε να μπορέσουμε να τα εισάγουμε στα δένδρα μας.

Αρχικά, με την βοήθεια του score tf-idf για καθένα από τα 4514 news που είχαμε βρήκαμε k keywords στο κώδικα μας επιλέξαμε 3 keywords.



```
vords

t tokenize
mport itemgetter

word, sentences):
l([w in x for w in word]) for x in sentences]
[sentences[i] for i in range(0, len(final)) if final[i]]
len(sent_len))

ict_elem, n):
ct(sorted(dict_elem.items(), key = itemgetter(1), reverse = True)[:n])
lt


s import stopwords
ize import word_tokenize
t(stopwords.words('english'))

(4514):
xt[i]
 = doc.split()
length = len(total_words)
l_word_length)
nces = tokenize.sent_tokenize(doc)
len = len(total_sentences)
l_sent_len)

{}
rd in total_words:
rd = each_word.replace('.','')
_word not in stop_words:
each_word in tf_score:
     tf_score[each_word] += 1
e:
 tf_score[each_word] = 1

by total_word_length for each dictionary element
date((x, y/int(total_word_length)) for x, y in tf_score.items())
core)

 {}
rd in total_words:
rd = each_word.replace('.','')
_word not in stop_words:
each_word in idf_score:
 idf_score[each_word] = check_sent(each_word, total_sentences)
e:
 idf_score[each_word] = 1

g a log and divide
pdate((x, math.log(int(total_sent_len)/y)) for x, y in idf_score.items())

score)
e = {key: tf_score[key] * idf_score.get(key, 0) for key in tf_score.keys()}
df_score)
_top_n(tf_idf_score, 3))
]=list(mydict.keys())
```

Αφού, βρήκαμε τα keywords έπρεπε να τα μετατρέψουμε σε αριθμητικά δεδομένα. Αυτό,υλοποιήθηκε με την χρήση της βιβλιοθήκες genism οπού κάθε λέξη μετατράπηκε σε αριθμό με την μέθοδο των word embeddings

```
king the points with wordembeddings

gensim.models import Word2Vec
sklearn.decomposition import PCA
matplotlib import pyplot

=[]
 in news.text:
or k in i:
   words.append(k)

ords)


orw2v=news['text'].to_numpy()
orw2v

([list(['Administration', 'Union', 'Territory']),
  list(['Arora', 'slammed', 'Instagram']),
  list(['Indira', 'Gandhi', 'Institute']), ...,
  list(['Aamir', 'Khan,', 'talking']),
  list(['Maharashtra', 'government', 'initiated']),
  list(['At', 'least', '400'])], dtype=object)

=np.unique(words)

 Word2Vec(dataforw2v, min_count=1, size = 1)
(w2v)

Vec(vocab=4652, size=1, alpha=0.025)

(w2v['Khan'])

8664798]

hon-input-295-9d40c1df3367>:1: DeprecationWarning: Call to deprecated `__getitem__` (Method will be removed in 4.0.0, use
wv.__getitem__() instead).
nt(w2v['Khan'])

s=[]
i in dataforw2v:
oints.append(w2v[i].T)

hon-input-296-74b3e21c7a63>:3: DeprecationWarning: Call to deprecated `__getitem__` (Method will be removed in 4.0.0, use
wv.__getitem__() instead).
nts.append(w2v[i].T)

s=np.array(points)
sdot=points.tolist()

tsdot

s = []
ublist in pointsdot:
or item in sublist:
   points.append(item)

s
382670223712921, 0.4727856814861297 6, -0.428498089313507 1],
6515797972679138, -0.03899844436331795, -0.4355745315551758],
```

2. Kd-Tree

Τα δεδομένα μας γίνονται sort και το μεσαίο αποτελεί την ριζά έπειτα τα στοιχεία από δεξιά και αριστερά διαλέγετε το μεσαίο τους με βάση την επόμενη διάσταση αυτή η διαδικασία γίνεται ώσπου να τελειώσουν τα σημεία.

EE

```
ee(object):


init__(self, points, dim, dist_sq_func=None):

 dist_sq_func is None:
  dist_sq_func = lambda a, b: sum((x - b[i]) ** 2
      for i, x in enumerate(a))

f make(points, i=0):
  if len(points) > 1:
      points.sort(key=lambda x: x[i])
      i = (i + 1) % dim
      m = len(points) >> 1
      return [make(points[:m], i), make(points[m + 1:], i),
          points[m]]
  if len(points) == 1:
      return [None, None, points[0]]

f add_point(node, point, i=0):
  if node is not None:
      dx = node[2][i] - point[i]
      for j, c in ((0, dx >= 0), (1, dx < 0)):
          if c and node[j] is None:
              node[j] = [None, None, point]
          elif c:
              add_point(node[j], point, (i + 1) % dim)
```

Έπειτα με την βοήθεια του heap υλοποιείται το knn ελέγχεται κάθε φορά η ριζά του υποδέντρου ώστε να προορίσουμε και να βρεθούν του k nearest neighbors.

```
t_knn(node, point, k, return_dist_sq, heap, i=0, tiebreaker=1):
 node is not None:
  dist_sq = dist_sq_func(point, node[2])
  dx = node[2][i] - point[i]
  if len(heap) < k:
      heapq.heappush(heap, (-dist_sq, tiebreaker, node[2]))
  elif dist_sq < -heap[0][0]:
      heapq.heappushpop(heap, (-dist_sq, tiebreaker, node[2]))
  i = (i + 1) % dim
  # Goes into the left branch, then the right branch if needed
  for b in (dx < 0, dx >= 0)[:1 + (dx * dx < -heap[0][0])]:
      get_knn(node[b], point, k, return_dist_sq,
          heap, i, (tiebreaker << 1) | b)
 tiebreaker == 1:
```

3. Quad tree

Για την δεικτοδότηση των κειμένων μέσω ενός Quad tree δημιουργούμε αρχικά μία κλάση Point η οποία θα περιέχει τις τιμές των σημείων, δημιουργούμε μία συνάρτηση αρχικοποίησης (θα μπορούσε να έχει άλλη μία μεταβλητή με την λέξη που αντιστοιχεί στο σημείο ή οποιαδήποτε άλλη πληροφορία χρειάζεται), ακολουθεί μία συνάρτηση η οποία επιστρέφει το αντικείμενο σε μορφή string ώστε να είναι αναγνώσιμο και μία συνάρτηση η οποία χρειάζεται για να εκτυπωθεί το επιθυμητό αποτέλεσμα και όχι οι δείκτες του αντικειμένου.

```python
import pandas as pd

class Point:

    def __init__(self, x, y):
        self.x=x
        self.y =y


    def __str__(self):
        return '('+str(self.x)+','+str(self.y)+ ')'


    def __repr__(self):
        return '('+str(self.x)+','+str(self.y)+ ')'
```

Δημιουργούμε μια κλάση Node που αναπαριστά τους κόμβους του δέντρου. Στην περίπτωση του Quad tree είναι ένα ορθογώνιο παραλληλόγραμμο. Η συνάρτηση αρχικοποίησης δέχεται ως είσοδο το κέντρο του ορθογώνιου παραλληλόγραμμου και τις διαστάσεις του. Η συνάρτηση contains ελέγχει αν ένα σημείο βρίσκεται εντός των ορίων του παραλληλογράμμου και η συνάρτηση intersects ελέγχει αν το παραλληλόγραμμο εισόδου τέμνει το παραλληλόγραμμο (θα χρειαστούν για την αναζήτηση).

```python
class Node:

    def __init__(self, kx, ky, w, h):
        self.cx = kx
        self.cy = ky
        self.w = w
        self.h = h
        self.w_edge = kx - w/2
        self.e_edge = kx + w/2
        self.n_edge = ky - h/2
        self.s_edge = ky + h/2

    def contains(self, point):

        point_x = point.x
        point_y = point.y

        return (point_x >= self.w_edge and point_x <  self.e_edge and point_y >= self.n_edge and point_y < self.s_edge)

    def intersects(self, other):

        return not (other.w_edge > self.e_edge or other.e_edge < self.w_edge or other.n_edge > self.s_edge or other.s_edge < self.n_edge)
```

Στη κλάση Quadtree υλοποιείται η βασική δομή του δέντρου. Στην πρώτη συνάρτηση αρχικοποιούμε το ορίζοντας τα όρια της ρίζας και το πόσα σημεία θα εμπεριέχονται σε κάθε κόμβο. Συνοδεύονται απο τη δημιουργία μιας λίστας που θα εμπεριέχει τα παραπάνω σημεία, το βάθος που βρίσκεται ο κόμβος και ένα flag το οποίο δηλώνει εάν έχει γίνει διαίρεση στον συγκεκριμένο κόμβο. Η συνάρτηση διαίρεσης κόμβου, υποδιαίρει τον κόμβο σε 4 συμμετρικούς υπό-κόμβους οι οποίοι καταλαμβάνουν την βορειοανατολική,

βορειοδυτική, νοτιοανατολική, νοτιοδυτική επιφάνεια του πατέρα κόμβου/παραλληλογράμμου, επίσης το που flag ορίζει αν ο κόμβος έχει διαιρεθεί γίνεται αληθές. Η συνάρτηση insert, εισάγει νέα σημεία στο δέντρο. Σε περίπτωση που το δοθέντο σημείο δεν βρίσκεται εντός των ορίων του μητρικού κόμβου επιστρέφεται Ψευδές από τη συνάρτηση, αν υπάρχει χώρος στον κόμβο το σημείο προστίθεται στη λίστα, αν δεν υπάρχει χώρος τότε καλείται η συνάρτηση διαίρεσης. Δεν καταφέραμε να υλοποιήσουμε k-nn search. Υλοποιήσαμε Range search στη συνάρτηση search, παίρνει ως ορίσματα τα όρια (σε μορφή αντικειμένου Node (παραλληλογράμμου)) στα οποία θέλουμε να αναζητήσουμε και μία κενή λίστα η οποία θα επιστραφεί με τα σημεία που βρέθηκαν. Αρχικά η συνάρτηση ελέγχει αν το παραλληλόγραμμο αναζήτησης τέμνει το παραλληλόγραμμο του κόμβου. Σε περίπτωση που τέμνει, αναζητά τα σημεία του κόμβου ώστε να δει αν είναι εντός των ορίων αναζήτησης, αν ο κόμβος έχει παιδία καλείται η ίδια συνάρτηση στα παιδιά του κόμβου αφού αποτελούν γεωμετρικό υποσύνολο του πατέρα. Τέλος επιστρέφεται η λίστα με τα σημεία που βρέθηκαν.

```python
class QuadTree:

    def __init__(self, bounds, capacity, depth=0):

        self.bounds = bounds
        self.capacity = capacity
        self.points = []
        self.depth = depth
        self.div = False

    def divide(self):

        kx = self.bounds.kx
        ky = self.bounds.ky
        w = self.bounds.w / 2
        h = self.bounds.h / 2

        self.nw = QuadTree(Node(kx - w/2, ky - h/2, w, h), self.capacity, self.depth + 1)
        self.ne = QuadTree(Node(kx + w/2, ky - h/2, w, h), self.capacity, self.depth + 1)
        self.se = QuadTree(Node(kx + w/2, ky + h/2, w, h), self.capacity, self.depth + 1)
        self.sw = QuadTree(Node(kx - w/2, ky + h/2, w, h), self.capacity, self.depth + 1)

        self.div = True

    def insert(self, point):

        if not self.bounds.contains(point):

            return False

        if len(self.points) < self.capacity:

            self.points.append(point)
            return True

        if not self.div:
            self.divide()

        return (self.ne.insert(point) or self.nw.insert(point) or self.se.insert(point) or self.sw.insert(point))

    def search(self, bounds, searchresult):

        if not self.bounds.intersects(bounds):

            return False

        for point in self.points:
            if bounds.contains(point):
                searchresult.append(point)

        if self.div:
            self.nw.search(bounds, searchresult)
            self.ne.search(bounds, searchresult)
            self.se.search(bounds, searchresult)
            self.sw.search(bounds, searchresult)
        return searchresult
```

Στη βασική ροή του προγράμματος αρχικοποιούμε τις μεταβλητές που θα ορίσουν τον χώρο που θα περιέχει όλα μας τα σημεία και κατ’ επέκταση το παραλληλόγραμμο που θα οριστεί για τον πρώτο κόμβο. Αντλούμε τα σημεία από το προ επεξεργασμένο αρχείο csv, γίνεται επιπλέον επεξεργασία με κανονικοποίηση των στοιχείων για ευκολία διαχείρισης των δεδομένων ώστε να μην υπάρχουν αρνητικές τιμές. Δημιουργούμε λίστα με αντικείμενα Point και τη γεμίζουμε με τις επεξεργασμένες συντεταγμένες. Ορίζουμε τα όρια του χώρου δημιουργώντας το παραπάνω παραλληλόγραμμο. Καλούμε το τη κλάση Quad Tree δίνοντας ως ορίσματα τον προαναφερθέντα τομέα και τον μέγιστο αριθμό σημείων που θα μπορεί να αποθηκεύσει ο κάθε κόμβος. Προσπελάσουμε τη λίστα με τα σημεία και καλούμε τη συνάρτηση insert για το καθένα από αυτά. Τέλος κάνουμε μια αναζήτηση με τη συνάρτηση search (range search) και εκτυπώνουμε τα αποτελέσματα.

```
width, height = 1.01, 1.01

df=pd.read_csv(r'c:/Users/Zksnarker420/Downloads/out2d.csv',engine='python')

df=df.drop(columns=['Unnamed: 0'])

mylist=df.values.tolist()

df_min_max_scaled = df.copy()

for column in df_min_max_scaled.columns:
    df_min_max_scaled[column] = (df_min_max_scaled[column] - df_min_max_scaled[column].min()) / (df_min_max_scaled[column].max() - df_min_max_scaled[column].min())

normlist=df_min_max_scaled.values.tolist()

points = [Point(list[0], list[1]) for list in normlist]

domain = Node(width/2, height/2, width, height)
qtree = QuadTree(domain, 1)
for point in points:
    print(point)
    qtree.insert(point)

searchresult=[]
searchspace=Node(0.5,0.5,0.1,0.1)
q=qtree.search(searchspace, searchresult)
print(q)
```

4. Range tree

Για την δεικτοδότηση των κειμένων μέσω ενός Range tree δημιουργούμε αρχικά μία κλάση Node η οποία περιέχει την τιμή του κόμβου, τη διάσταση στην οποία ανήκει ο κόμβος, το δεξί και το αριστερό υπό-δέντρο.

```
class Node(object):

    """A node in a Range Tree."""

    def __init__(self, value) -> None:
        self.value = value
        self.left = None
        self.right = None
        self.dim= 0
        self.assoc=None
```

Στη συνέχει ορίζουμε μία συνάρτηση BuildRangeTree2d(), η οποία δημιουργεί το Range tree, παίρνοντας ως είσοδο τις τιμές που θα αποθηκεύσουμε στους κόμβους ταξινομημένες κάθε φορά με βάση τη διάσταση για την οποία αποθηκεύουμε του δέντρο και μία λογική μεταβλητή enable η οποία εναλλάσσετε ανάλογα με την διάσταση στην οποία προσθέτουμε τον κόμβο.

```
def BuildRangeTree2d(data, enable=True):
    '''
    Construct a 2 dimensional range tree
    Arguments:
        data          : The data to be stored in range tree
        enable        : to toggle whether to build the xtree or ytree
    Returns:
        tree
    '''

    if not data:         #No data
        return None
    if len(data) == 1:       #size of data=1--> leaf
        node = Node(data[0])

    else:
        mid_val = len(data)//2 #get mid val to be root

        node = Node(data[mid_val])

        node.left = BuildRangeTree2d(data[:mid_val], enable)
        node.right = BuildRangeTree2d(data[mid_val+1:], enable)

    if enable==True:
        data1 =data.sort( key=lambda x: x[1]) # sort by y dimension
        node.assoc = BuildRangeTree2d(data1,enable)

    return node
```

Η βοηθητική συνάρτηση getValue() επιστρέφει τη ζητούμενη τιμή ενός κόμβου ανάλογα με την διάσταση που δίνετε.

```
def getValue (point, dim):

    '''
    Reads the desired value from node
    Returns : value of node
    '''
    if dim==0:
            value = point.value[0]
    else:
            value = point.value[1]

    return value
```

Έπειτα ορίζουμε την συνάρτηση insert() η οποία προσθέτει ένα νέο κόμβο στο ήδη υπάρχουν δέντρο και επιστρέφει το αναβαθμισμένο δέντρο.

```
def insert(tree,point,enable=True):

    '''
    inserts an node to a tree
    tree:   existing tree
    point:   values of the node to be inserted
    enable  : to toggle whether to insert on a xtree or ytree

    '''

    x = point[0]
    y = point[1]
    dim=0

    if tree is None:
        tree=Node(point)    #insert value
        return tree
    if dim==0:          #for dim=0

        if x>getValue (tree,dim):
                tree.right=insert(tree.right,point,enable)  #recursive call for right branch

        else:
            tree.left=insert(tree.left,point,enable)    #recursive call for left branch
        dim=1
    else:           #for dim=1
        if y>getValue (tree,dim):
                tree.right=insert(tree.right,point,enable)  #recursive call for right branch

        else:
            tree.left=insert(tree.left,point,enable)    #recursive call for left branch

    return tree
```

Μέσω της βοηθητική συνάρτησης print_tree() εκτυπώνουμε το δέντρο κατά διάσταση x.

```
def print_tree(root):


    '''
    prints x dimension of tree
    useful for confirmation of insertion
    '''

    quene = []
    quene.append(root)
    while len(quene) != 0 :
        node = quene[0]
        if node.left == None:
            ll = '-'
        else:
            ll = node.left.value
        if node.right == None:
            rr = '-'
        else:
            rr = node.right.value
        print('  {n}  \n _|_ \n|   |\n{l}   {r}\n==========='.format(n = node.value, l = ll, r = rr))
        quene.pop(0)
        if node.left != None:
            quene.append(node.left)
        if node.right != None:
            quene.append(node.right)
    print('\n')
    print('---------------------------------')
```

Παρακάτω μπορούμε να δούμε ένα κομμάτι του δέντρο πριν και μετά στην εισαγωγή του κόμβου με τιμή [1,5] .

```
==========
  [0.93856942653656, 0.9028908014297484]
 _|_
|   |
-   -
==========
  [0.9474087953567504, -0.4485372304916382]
 _|_
|   |
-   -
==========
  [0.95053631067276, 0.4552461504936218]
 _|_
|   |
-   -
==========
  [0.9575278162956238, -0.2192590534687042]
 _|_
|   |
-   -
==========
  [0.9701283574104308, -0.1978592872619629]
 _|_
|   |
-   -
==========
  [0.9838544130325316, -0.0698785558342933]
 _|_
|   |
-   -
==========
  [0.9868017435073853, 0.7143830060958862]
 _|_
|   |
-   -
==========
```

```
  [0.93856942653656, 0.9028908014297484]
 _|_
|   |
-   -
==========
  [0.9474087953567504, -0.4485372304916382]
 _|_
|   |
-   -
==========
  [0.95053631067276, 0.4552461504936218]
 _|_
|   |
-   -
==========
  [0.9575278162956238, -0.2192590534687042]
 _|_
|   |
-   -
==========
  [0.9701283574104308, -0.1978592872619629]
 _|_
|   |
-   -
==========
  [0.9838544130325316, -0.0698785558342933]
 _|_
|   |
-   -
==========
  [0.9868017435073853, 0.7143830060958862]
 _|_
|   |
-   -
==========
  [1, 5]
 _|_
|   |
-   -
```

Για την υλοποίηση ερωτημάτων εύρεσης k-NN δημιουργούμε την συνάρτηση NN() η οποία επιστρέφει τον nearest neighbour. Έπειτα δημιουργούμε την συνάρτηση K_NN() η οποία χρησιμοποιεί την συνάρτηση NN() επαναληπτικά k φορές για επιστρέψει τους k- nearest neighbours.

Η συνάρτηση NN() (nearest neighbour):

```
def NN(tree,query,best_node,best_distance,dim):

    '''
    Finds nearest neighbor for a specific query on a tree.
    Returns the node that is the closest to the query and the distance from it

    tree:   the existing tree we apply knn on
    query:  values of said query we are using
    best_node:  node closest to our query at the moment
    best_distance:  distance from best_node
    dim: dimension of tree dim=0-->x,dim=1-->y

    '''


    if not tree:
        return best_node


    d=distance(query[dim],getValue(tree,dim))

    if d<best_distance:
        best_node=tree
        best_distance=d

        #deciding side

        if query[dim]<getValue(tree,dim):
            good_side=tree.left
            bad_side=tree.right
        else:
            good_side=tree.right
            bad_side=tree.left

        #good side
        best_node=NN(good_side,query,best_node,best_distance,dim=1)
        #checking for the bad side
        if abs(getValue(tree,dim)-query[dim])<best_distance:
                best_node=NN(bad_side,query,best_node,best_distance,dim=1)

    return best_node
```

Η συνάρτηση K_NN() (k- nearest neighbours):

```
def K_NN(tree,query,best_node,best_distance,dim,k):

    '''
    returns k nearest neighbors using NN function
    '''

    k_node_value=[]

    for i in range(k):
        best_node=NN(tree,query,best_node,best_distance,dim)
        k_node_value.append(best_node.value)

        if query[dim]<best_node.value[dim]:
            tree=best_node.left
        else:
            tree=best_node.right


    return k_node_value
```

Για ερώτημα εύρεσης k-NN με k=8 για το σημείο [0.08,0.04] επιστρέφονται τα σημεία:

```
[[0.5263878703117371, 0.034265398979187], [0.141037568449974, 0.4177823364734649], [0.0544250458478927, 0.5320
694446563721], [0.0851075947284698, -0.0556972064077854], [0.0797259286046028, 0.8144069910049438], [0.0830982
17844963, -0.0730652809143066], [0.0815982222557067, -0.7616937756538391], [0.0808117687702179, -0.35163825750
35095]]
```

5. Lsh-Locality Sensitive Hashing

Για την υλοποίηση της μεθόδου LSH αρχικά χρησιμοποιούμε τη συνάρτηση TfidfVectorizer της βιβλιοθήκης sklearn για να μετατρέψουμε τα δεδομένα μας σε μορφή tf-idf.

```
tfidf = TfidfVectorizer(
    analyzer='word',
    ngram_range=(1, 3),
    min_df=0,
    stop_words='english')
X_tfidf = tfidf.fit_transform(df['text'])
print("TF-IDF :")
print(X_tfidf)
```

Ένα μέρος της μετατροπής αυτής φαίνεται παρακάτω:

```
TF-IDF :
  (0, 227498)   0.1123403123168247
  (0, 84850)    0.11781735350122088
  (0, 95808)    0.11781735350122088
  (0, 197939)   0.11781735350122088
  (0, 52030)    0.11781735350122088
  (0, 125497)   0.11781735350122088
  (0, 114191)   0.11781735350122088
  (0, 5332)     0.11781735350122088
  (0, 70704)    0.11781735350122088
  (0, 266185)   0.11781735350122088
  (0, 97669)    0.11781735350122088
  (0, 14890)    0.11781735350122088
  (0, 28566)    0.11781735350122088
```

Στη συνέχεια δημιουργούμε τη συνάρτηση random_vectors() η οποία παράγει τυχαία διανύσματα στο χώρο τα οποία καθορίζουν τα buckets.

```
def random_vectors(dim, n_vectors):
    """
    generate random projection vectors
    """
    return np.random.randn(dim, n_vectors)
```

Έπειτα δημιουργούμε τη συνάρτηση LSH() η οποία χρησιμοποιώντας τη συνάρτηση random_vectors() διαχωρίζει τα δεδομένα μας σε buckets. Έπειτα κωδικοποιεί τους δείκτες των buckets από δυαδικό σε δεκαδικό. Επιστρέφει ένα dictionary που περιέχει τα buckets , τα διανύσματα , τους ακέραιους δείκτες και του δυαδικούς δείκτες.

```python
def LSH(X_tfidf, n_vectors):

    dim = X_tfidf.shape[1]
    rand_vectors = random_vectors(dim, n_vectors)

    # partition data points into bins,
    # and encode bin index bits into integers
    bit_bin_pointers = X_tfidf.dot(rand_vectors) >= 0
    powers_of_two = 1 << np.arange(n_vectors - 1, -1, step=-1)  # x << y is the same as multiplying x by 2 ** y --> shift to the left by y places
    bin_pointers = bit_bin_pointers.dot(powers_of_two)

    # update `table` so that `table[i]` is the list of document ids with bin index equal to i
    table = defaultdict(list)
    for idx, bin_index in enumerate(bin_pointers):
        table[bin_index].append(idx)

    model = {'table': table,
             'random_vectors': rand_vectors,
             'bin_pointers': bin_pointers,
             'bit_bin_pointers': bit_bin_pointers}
    return model
```

Εκπαιδεύουμε το μοντέλο LSH για τα δεδομένα μας με 11 τυχαία διανύσματα.

Δημιουργούμε τη συνάρτηση search_near_bins() η οποία για ένα συγκεκριμένο ερώτημα σε δυαδική αναπαράσταση επιστρέφει όλους τους υποψήφιους γείτονες μέσα σε μία συγκεκριμένη ακτίνα χρησιμοποιώντας το μοντέλο LSH. Για την εύρεση των γειτόνων εναλλάσσονται τα bits του query για όλους τους δυνατούς συνδυασμούς ώστε να επιστρέφουμε διαφορετικά buckets που διαφέρουν κατά κάποια bits από τα bits του query. Η συνάρτηση επιστρέφει τους υποψήφιους γείτονες.

```python
def search_near_bins(query_bin_bits, table, search_radius=3, candidate_set=None):
    """
    For a given query vector and trained LSH model's table
    return all candidate neighbors with the specified search radius.

    """
    if candidate_set is None:
        candidate_set = set()

    n_vectors = query_bin_bits.shape[0]
    powers_of_two = 1 << np.arange(n_vectors - 1, -1, step=-1)

    for different_bits in combinations(range(n_vectors), search_radius):
        # flip the bits of the query bin to produce a new bit vector
        index = list(different_bits)
        alternate_bits = query_bin_bits.copy()
        alternate_bits[index] = np.logical_not(alternate_bits[index])

        # convert the new bit vector to an integer index
        nearby_bin = alternate_bits.dot(powers_of_two)

        # fetch the list of documents belonging to
        # the bin indexed by the new bit vector,
        # then add those documents to candidate_set;
        if nearby_bin in table:
            candidate_set.update(table[nearby_bin])

    return candidate_set
```

Τέλος δημιουργούμε τη συνάρτηση nearest_neighbour() χρησιμοποιεί τη συνάρτηση search_near_bins() και επιστρέφει τους κοντινότερους γείτονες υπολογίζοντας επιπλέον την ομοιότητα μεταξύ του ερωτήματος και των υποψήφιων γειτόνων χρησιμοποιώντας την μετρική του συνημίτονου.

```
def nearest_neighbour(X_tfidf, query_vector, lsh_model, max_search_radius):
    table = lsh_model['table']
    random_vectors = lsh_model['random_vectors']

    # compute bin index for the query vector, in bit representation.
    bin_index_bits = np.ravel(query_vector.dot(random_vectors) >= 0)

    # search nearby bins and collect candidates
    candidate_set = set()
    for search_radius in range(max_search_radius + 1):
        candidate_set = search_near_bins(bin_index_bits, table, search_radius, candidate_set)
    #print(candidate_set)
    # sort candidates by their true distances from the query
    candidate_list = list(candidate_set)
    candidates = X_tfidf[candidate_list]
    distance = pairwise_distances(candidates, query_vector, metric='cosine').flatten()

    distance_col = 'distance'
    nearest_neighbors = pd.DataFrame({
        'id': candidate_list, distance_col: distance
    }).sort_values(distance_col).reset_index(drop=True)

    return nearest_neighbors
```

Καλούμε τη συνάρτηση αυτή στα δεδομένα μας με ακτίνα αναζήτησης ίση με 4 και το id του query μας ίσο με 19 και παίρνουμε τα εξής αποτελέσματα:

	id	distance
0	19	1.110223e-16
1	1668	9.576242e-01
2	4360	9.619417e-01
3	4233	9.659109e-01
4	535	9.686168e-01
...	...	...
1173	2778	1.000000e+00
1174	732	1.000000e+00
1175	2783	1.000000e+00
1176	2729	1.000000e+00
1177	945	1.000000e+00

6. Αποτελέσματα

Οι παραπάνω μέθοδοι συγκρίθηκαν χρονικά. Τα αποτελέσματα ήταν τα εξής:

Μέθδος	Χρόνος ενός query (sec)
Kd-tree	0.010153770446777344 (K-NN)
Quad-tree	0.000997304916381836 (range search)
Range-tree	0.17136645317077637 (K-NN)
LSH	0.0156557559967041 (K-NN)

Παρατηρούμε ότι για ερωτήματα αναζήτησης K-NN το Kd-tree επιτυγχάνει τον καλύτερο χρόνο. Ακολουθεί η LSH και τελευταίο με τον χειρότερο χρόνο το Range tree.

Το Quad tree επιτυγχάνει τον καλύτερο χρόνο, το ερώτημα αναζήτησης που υλοποιείται όμως είναι range search επομένως δε μπορούμε να εξάγουμε κάποια παρατήρηση συγκριτικά με τους υπόλοιπους χρόνους