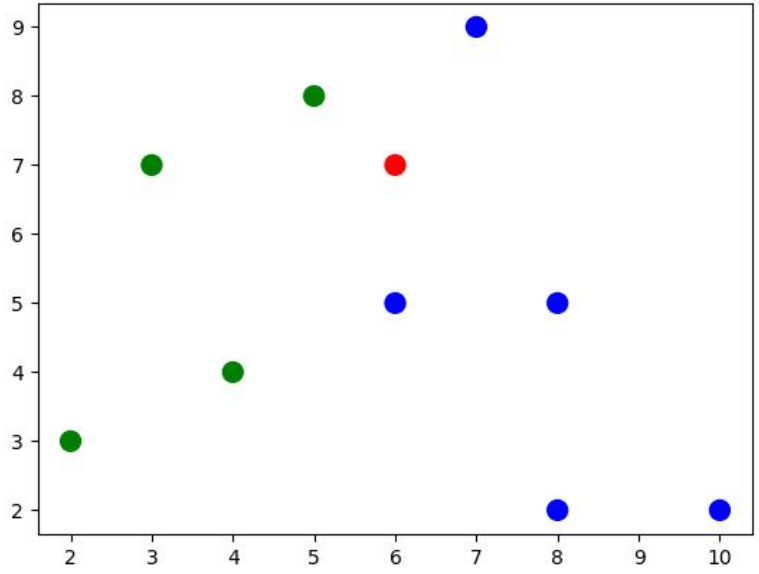
K-NN אלגוריתם לפיתוח מכונה לומדת

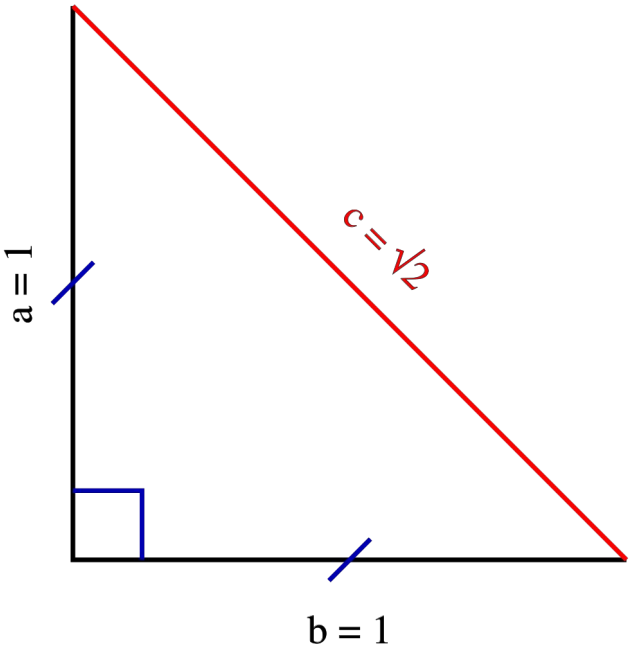
K-Nearest Neighbors algorithm

גדי הרמן

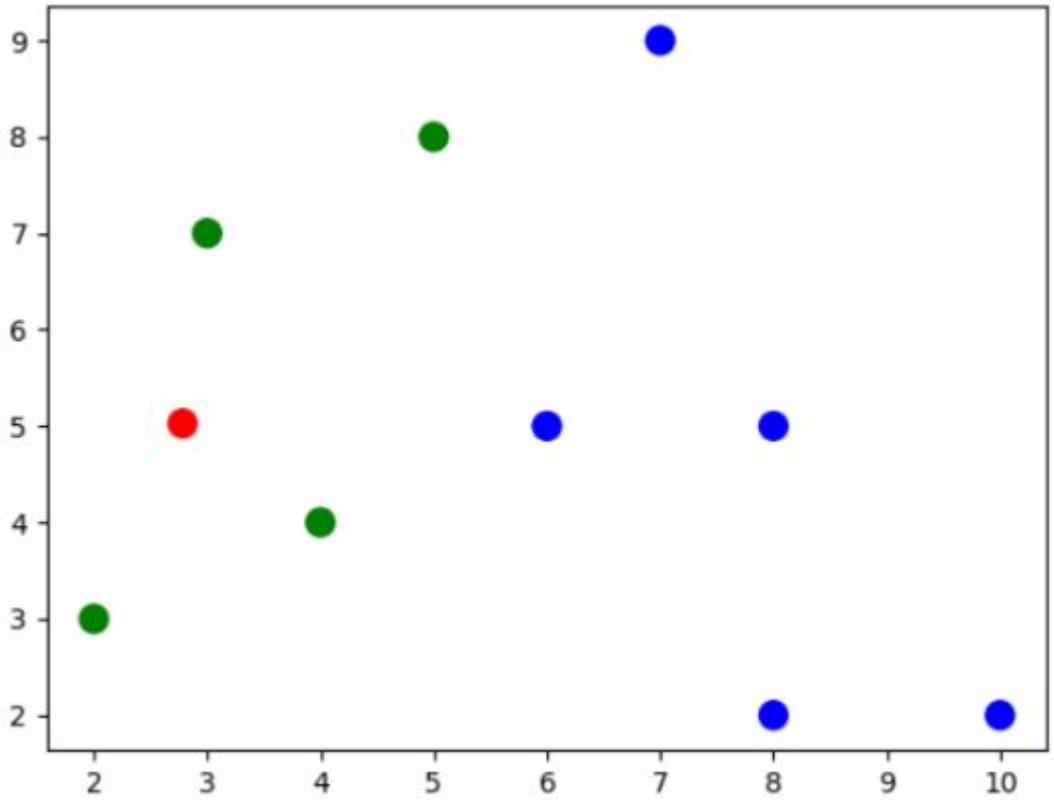
אלגוריתם K-Nearest Neighbors algorithm או בקיצור K-NN

K-NN מבוסס בין היתר על מדידת מרחק אוקלידי בין נקודות במרחב או בשם הפשוט יותר שלו, שימוש בפיתגורס כדי לחשב את היתר במשולש ישר זווית.

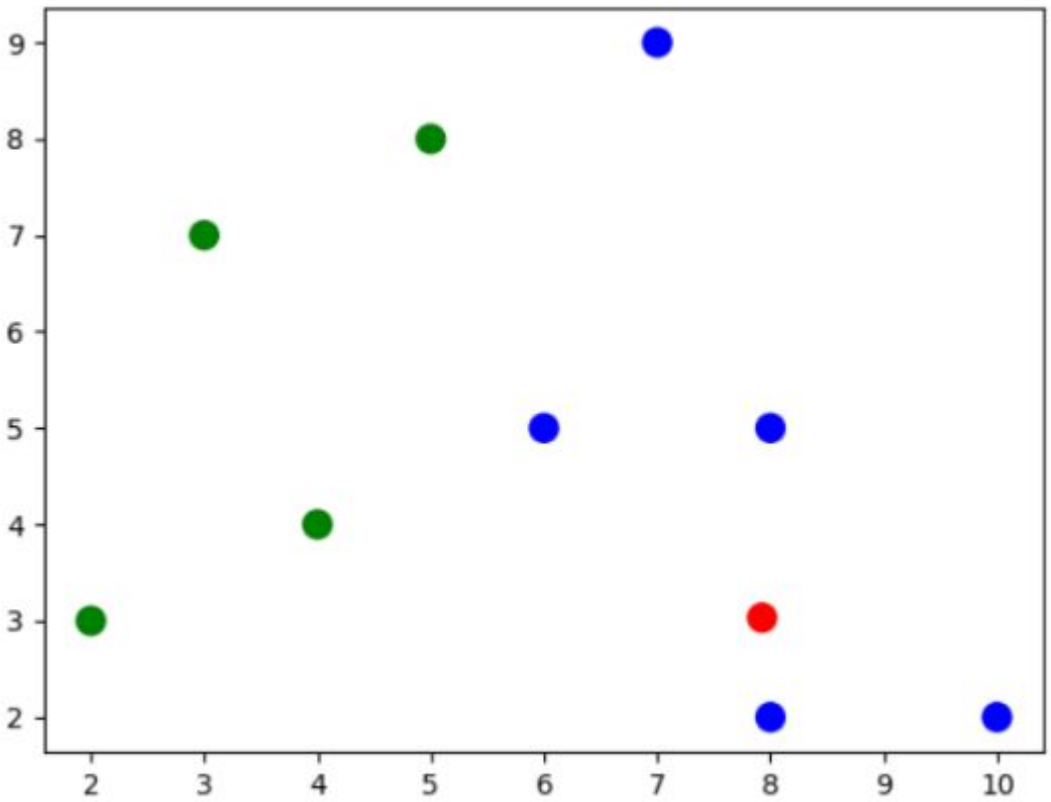




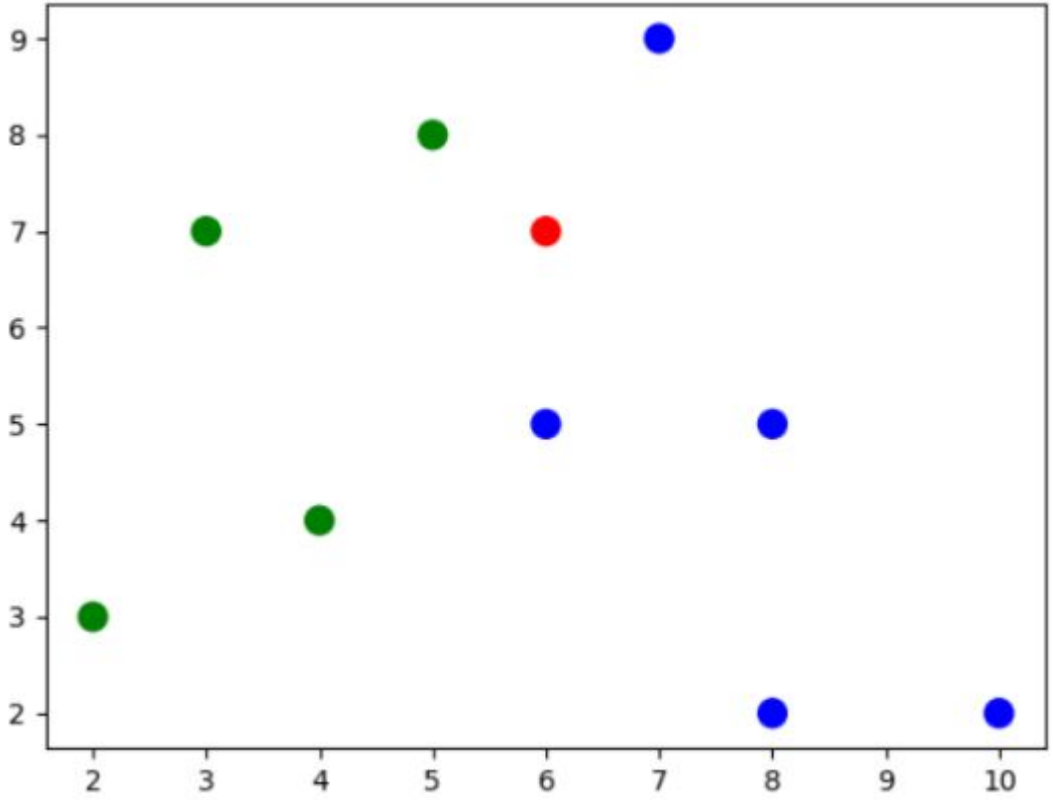
מי החברים שלי?



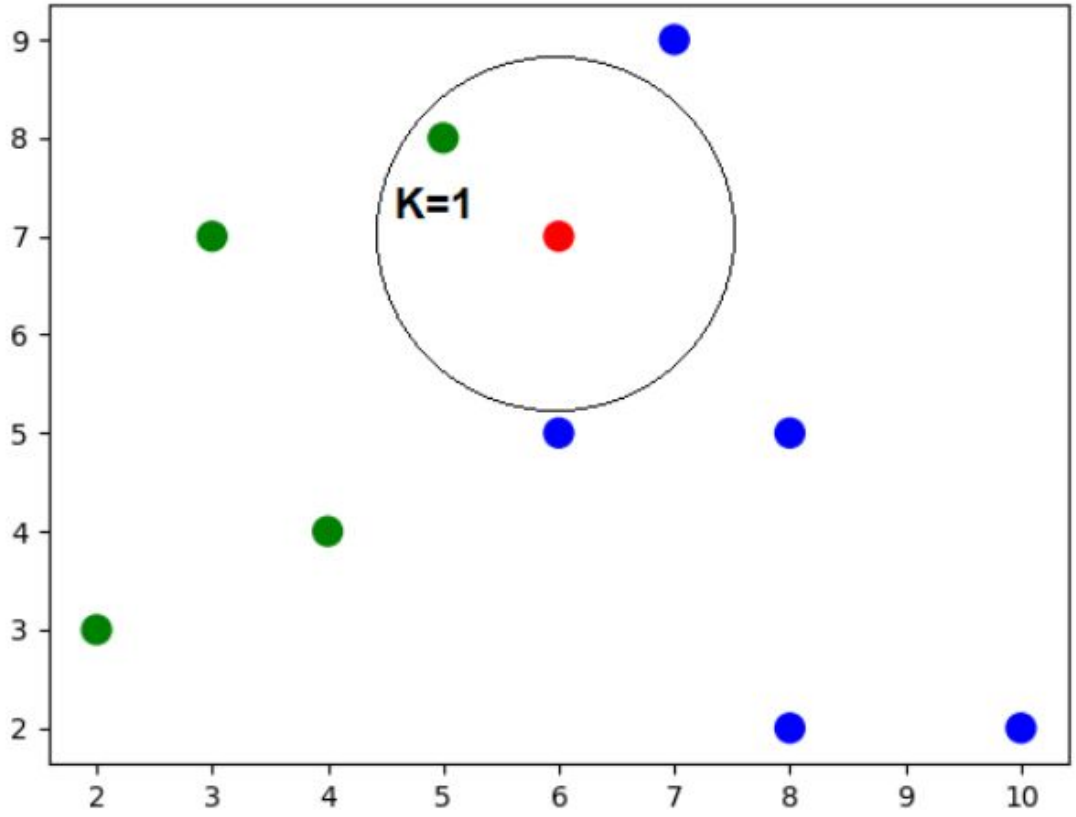
מי החברים שלי?



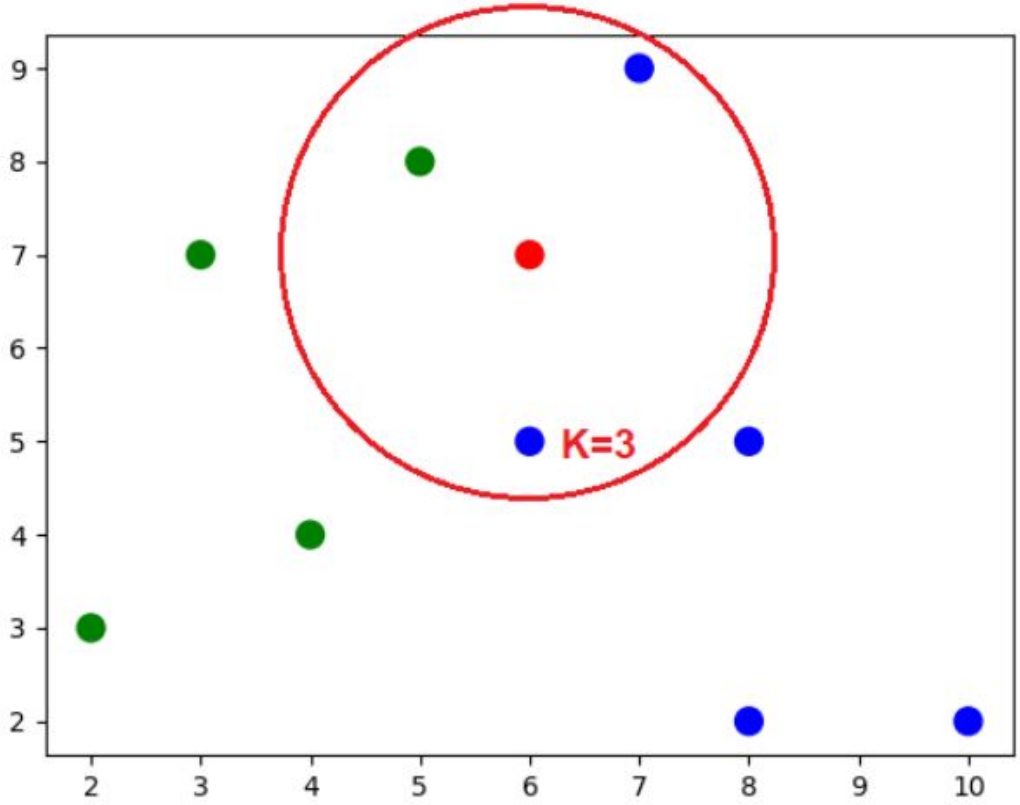
מי החברים שלי?



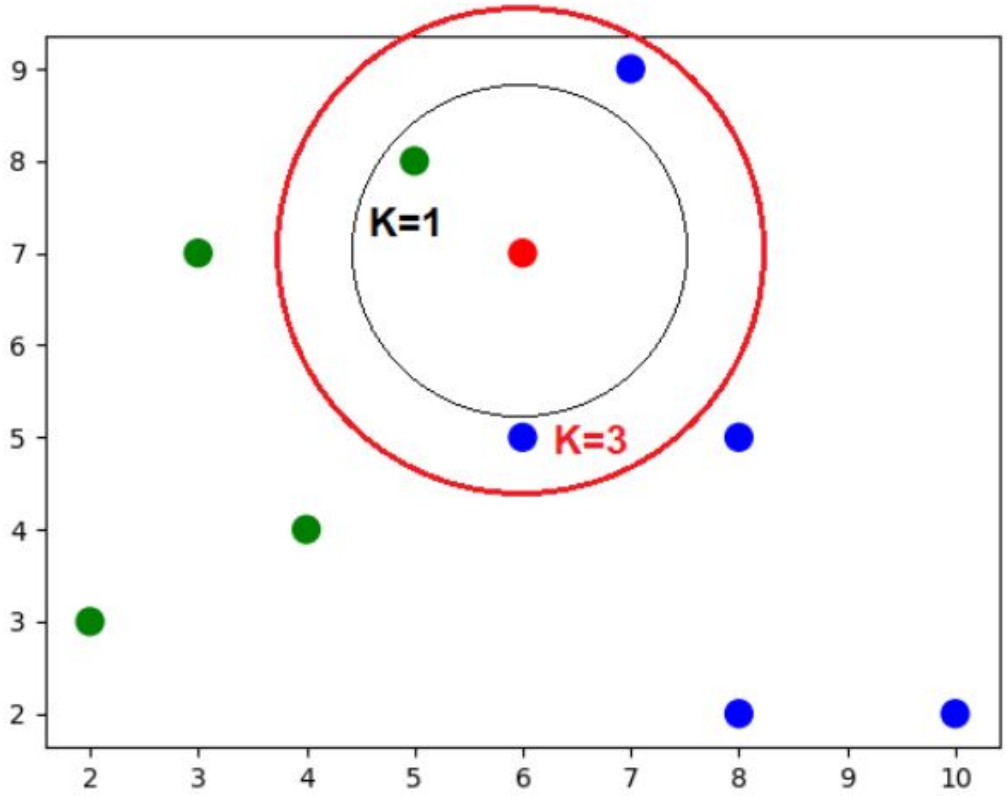
מי החברים שלי? K=1



מי החברים שלי? K=3

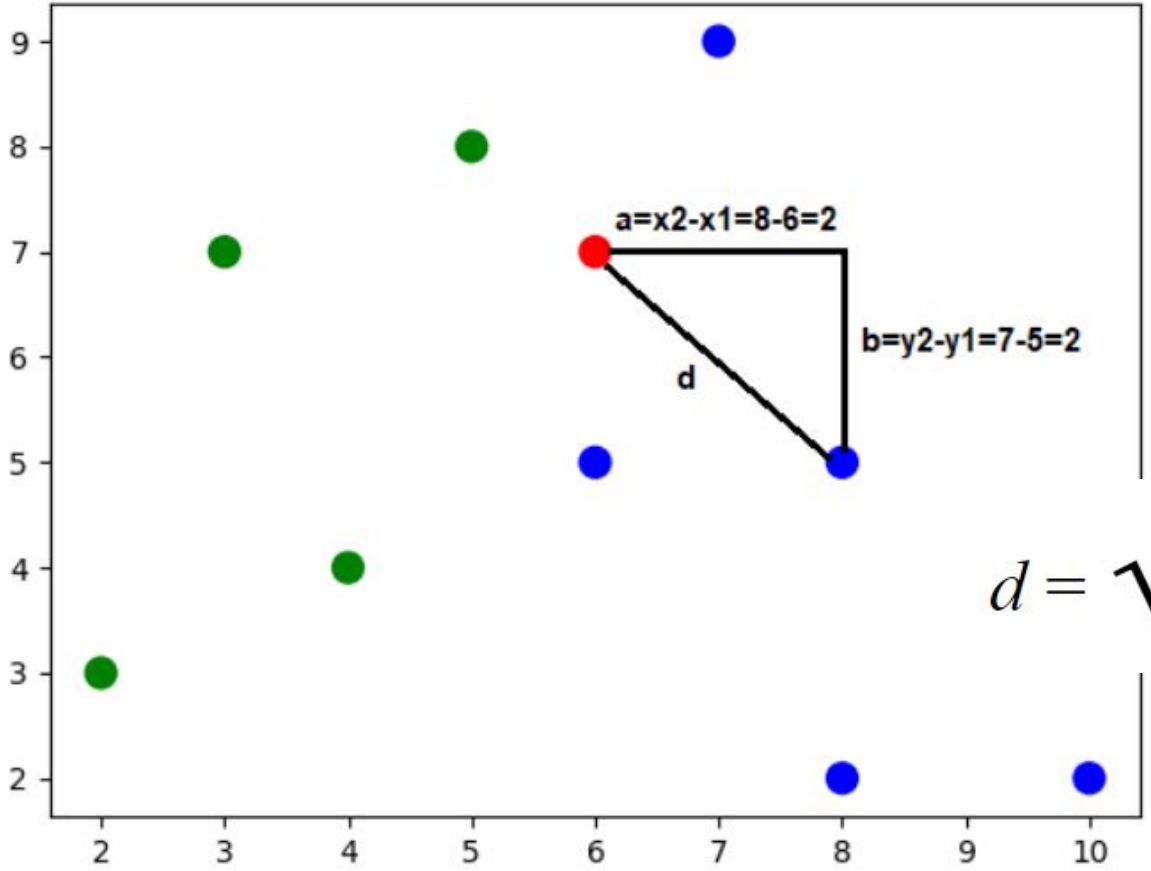


מי החברים שלי? K=1 או K=3



מרחק אוקלידי בין נקודות

כדי לממש את אלגוריתם KNN אנו זקוקים לנוסחה לחישוב מרחק בין נקודות. נדגים זאת על ידי מציאת המרחק בין 2 נקודות על פני מישור דו-מימדי.



$$d = \sqrt{(x2 - x1)^2 + (y2 - y1)^2}$$

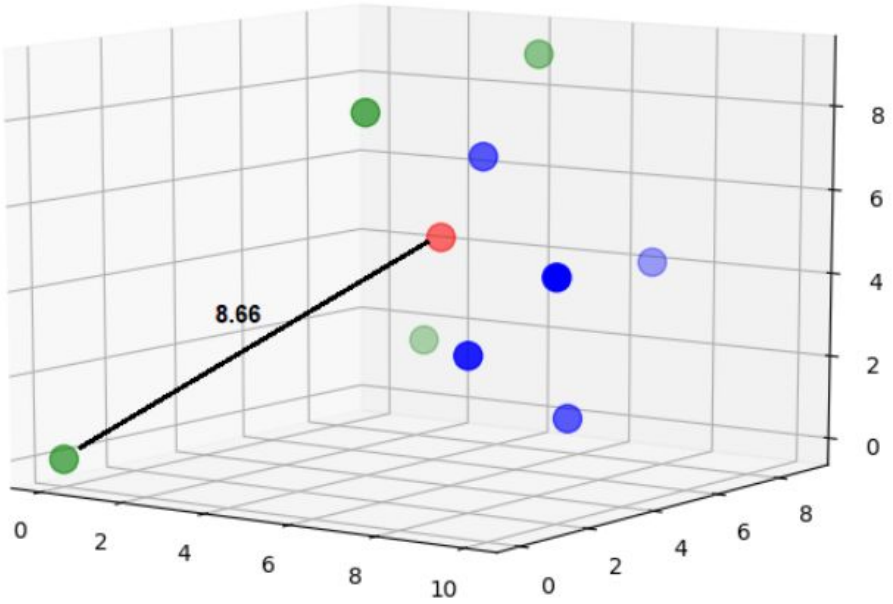
מימוש מרחק אוקלידי בין נקודות בקוד

$$d = \sqrt{(x2 - x1)^2 + (y2 - y1)^2}$$

```
def euclidean_distance(p1, p2):
    dx = float(p1[0]-p2[0])
    dy = float(p1[1]-p2[1])
    d = np.power(dx,2) + np.power(dy,2)
    return np.sqrt(d)
```

מרחק אוקלידי על גבי מערכת מרובת צירים

```
data = np.array( [      [ 5.0 , 5.0, 5.0],
                        [ 0.0 , 0.0, 0.0],
                        [ 3.0 , 7.0, 2.0],
                        [ 4.0 , 4.0, 8.0],
                        [ 5.0 , 8.0, 9.0],
                        [ 6.0 , 5.0, 7.0],
                        [ 7.0 , 9.0, 4.0],
                        [ 8.0 , 5.0, 1.0],
                        [ 8.0 , 2.0, 3.0],
                        [10.0 , 2.0, 5.0]    ])
```



$$d = \sqrt{(x2 - x1)^2 + (y2 - y1)^2 + (z2 - z1)^2}$$

$$d = \sqrt{(5 - 0)^2 + (5 - 0)^2 + (5 - 0)^2} = 8.66$$

מרחק אוקלידי על גבי מערכת מרובת צירים

נכתוב מחדש את הפעולה euclidean_distance כדי שתתאים לחישוב מרחק אוקלידי ביותר מ-2 מימדים.

חישוב מרחק ב-2 מימדים

```
def euclidean_distance(p1, p2):
    dx = float(p1[0]-p2[0])
    dy = float(p1[1]-p2[1])
    d = np.power(dx,2) + np.power(dy,2)
    return np.sqrt(d)
```

חישוב מרחק ב-N מימדים

```
def euclidean_distance(p1, p2):
    d = 0.0
    for i in range(len(p1)):
        a = float(p1[i])
        b = float(p2[i])
        d += np.power((a-b),2)
    d = np.sqrt(d)
    return d
```

מימוש הפעולה predict בתוך KNN

הפעולה predict

תקבל

- מערך נקודות כפרמטר בשם train.
- נקודה בודדת כפרמטר בשם test שאליה אנו רוצים לחפש את הנקודות הקרובות ביותר.
- מערך תגיות בשם lbl המציין לאיזו קבוצה שייכת כל נקודה במערך train.
- פרמטר בשם K שקובע את מספר הנקודות שעל פיהם נקבע לאיזו קבוצה שייכת הנקודה test.

הפעולה תבצע את האלגוריתם הבא:

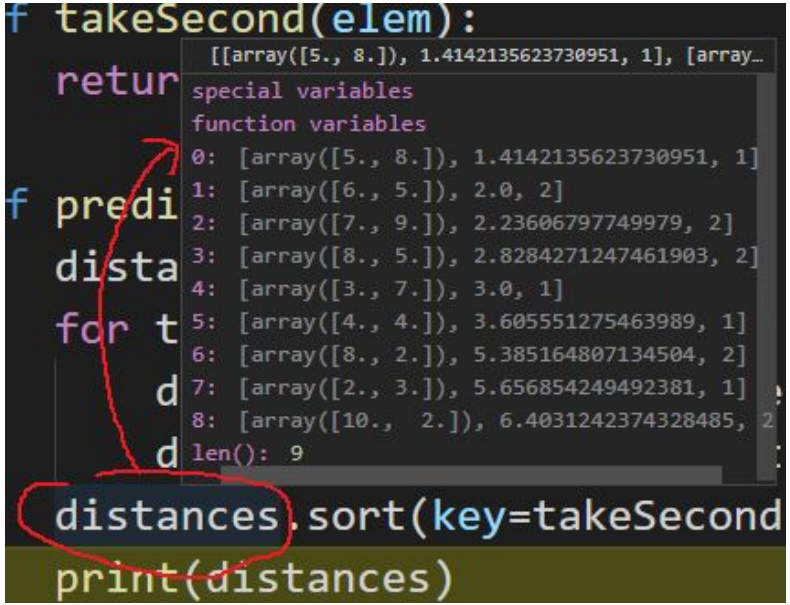
- תעבור בלולאה על כל הערכים במערך train ובכל פעם תזמן את הפעולה euclidean_distance
- הערך המוחזר מפעולה euclidean_distance נכנס למערך פנימי הכולל את המרחק, התגית ואת ערכי הנקודה שנבדקה.
- לאחר סיום מדידת כל הנקודות נמיין את המערך לפי המרחקים.
- לבסוף נעבור על הלולאה הממוינת ונחלץ ממנה את K האיברים שאותם אנו רוצים לקבל.
- הפעולה תחזיר פרמטר אחד שהוא מספר הקבוצה.

מימוש הפעולה predict בתוך KNN

```
def takeSecond(elem):
    return elem[1]

def predict(train, test, lbl, K):
    distances = []
    for t, l in zip(train,lbl):
        dist = euclidean_distance(test, t)
        distances.append([t, dist, l[0]])
    distances.sort(key=takeSecond)
    print(distances)
    neighbors = []
    for i in range(K):
        neighbors.append(distances[i])
    out = [row[-1] for row in neighbors]
    return max(out,key=out.count)
```

מימוש הפעולה predict בתוך KNN

```python
def takeSecond(elem):
    return elem[1]

def predict(train, test, lbl, K):
    distances = []
    for t, l in zip(train,lbl):
        dist = euclidean_distance(test, t)
        distances.append([t, dist, l[0]])
    distances.sort(key=takeSecond)
    print(distances)
    neighbors = []
    for i in range(K):
        neighbors.append(distances[i])
    out = [row[-1] for row in neighbors]
    return max(out,key=out.count)
```



מימוש הפעולה predict בתוך KNN

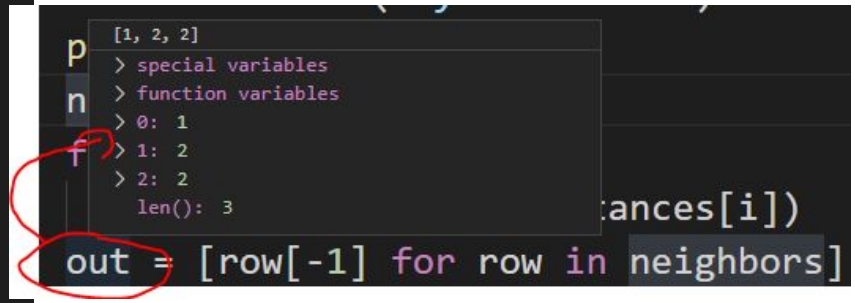
```python
def takeSecond(elem):
    return elem[1]

def predict(train, test, lbl, K):
    distances = []
    for t, l in zip(train,lbl):
        dist = euclidean_distance(test, t)
        distances.append([t, dist, l[0]])
    distances.sort(key=takeSecond)
    print(distances)
    neighbors = []
    for i in range(K):
        neighbors.append(distances[i])
    out = [row[-1] for row in neighbors]
    return max(out,key=out.count)
```

```python
    distances.append([t, dist, l[0]])
distance
print(di
neighbor
for i in
    neighbors.append(distances[i])
out = [row[-1] for row in neighbors]
```

```
[[array([5., 8.]), 1.4142135623730951, 1], [array…
special variables
function variables
0: [array([5., 8.]), 1.4142135623730951, 1]
1: [array([6., 5.]), 2.0, 2]
2: [array([7., 9.]), 2.23606797749979, 2]
len(): 3
```

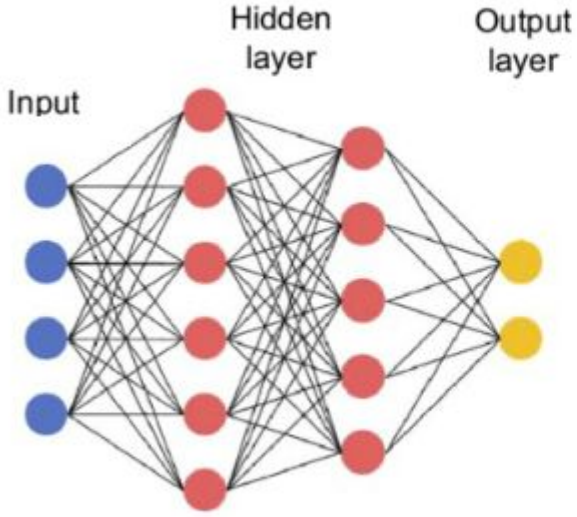
מימוש הפעולה predict בתוך KNN

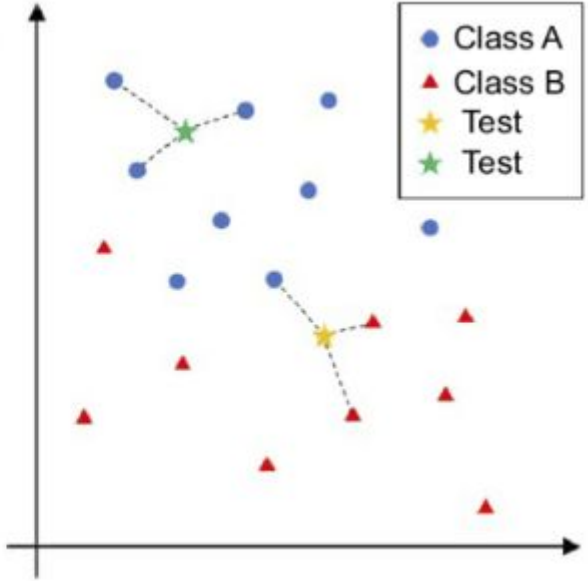
```python
def takeSecond(elem):
    return elem[1]

def predict(train, test, lbl, K):
    distances = []
    for t, l in zip(train,lbl):
        dist = euclidean_distance(test, t)
        distances.append([t, dist, l[0]])
    distances.sort(key=takeSecond)
    print(distances)
    neighbors = []
    for i in range(K):
        neighbors.append(distances[i])
    out = [row[-1] for row in neighbors]
    return max(out,key=out.count)
```



סיכום

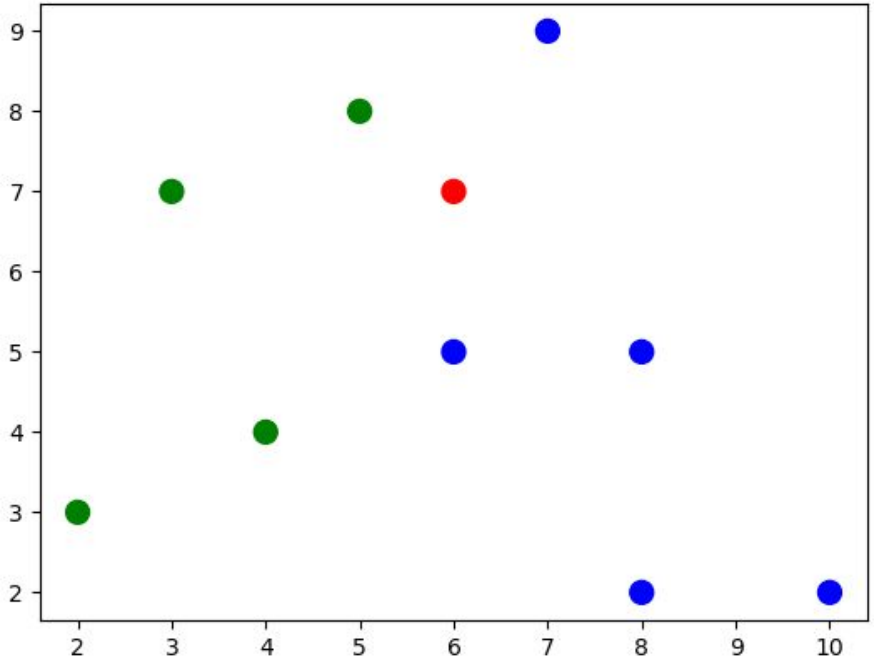
KNN הוא אלגוריתם עצלן לימודית כי בניגוד ל ANN שקודם לומד ולאחר מכן יכול לסווג. אלגוריתם KNN לומד ומסווג רק כאשר מתקבלת בקשה. מכאן שזה גם אחד החסרונות של KNN. אלגוריתם זה יתקשה לתפקד כאשר מדובר על כמויות גדולות של אימון (למידה). בקיצור אלגוריתם עצלן! כאילו התלמיד לומד תוך כדי הבחינה.





נבחן את הקוד הבא

```
import matplotlib.pyplot as plt
import numpy as np
data = np.array( [  [ 6 , 7],
                    [ 2 , 3],
                    [ 3 , 7],
                    [ 4 , 4],
                    [ 5 , 8],
                    [ 6 , 5],
                    [ 7 , 9],
                    [ 8 , 5],
                    [ 8 , 2],
                    [10 , 2]    ])
categories = np.array([0,1,1,1,1,2,2,2,2,2])
colormap = np.array(['r', 'g', 'b'])
plt.scatter(data[:,0], data[:,1], s=100, c=colormap[categories])
plt.show()
```



תרגול 8:

מימוש קוד KNN