

## פעילות 12 - אלגוריתם KNN ככלי לפיתוח מכונה לומדת

מקורות:

<https://medium.com/datadriveninvestor/knn-algorithm-and-implementation-from-scratch-b9f9b739c28f>

<https://www.youtube.com/watch?v=aMtckmWAZDg&t=393s>

<https://github.com/muzzart/knn-cifar10/blob/master/KNN%20-%20CIFAR10.ipynb>

בפעילות 9 כתבנו קוד המממש פרספטרון בודד במטרה להבין ולתרגל את עקרון הפעולה של מכונה לומדת העושה שימוש בנוירון אחד. בהמשך למדנו לעשות שימוש במספר פרספטרונים כדי לבנות רשת של נוירונים מלאכותיים - Artificial neural networks - ANN. חשוב להדגיש, אלגוריתמים למימוש רשתות נוירונים מלאכותיות אינם הכלי היחיד לפיתוח מכונות לומדות. קיימים מספר אלגוריתמים המיישמים מכונות לומדות חלק מהאלגוריתמים האלה, בתנאים מסוימים, מצליחים לפתור בעיות בתחום למידת המכונה באופן יעיל יותר מאלגוריתמים של רשתות נוירונים מלאכותיים.

בפעילות זו נדגים אלגוריתם בשם K-Nearest Neighbors algorithm או בקיצור K-NN. אלגוריתם זה הוא אחד הפתרונות לפיתוח מכונות לומדות. במסגרת הפעילות נלמד ש-KNN מבוסס בין היתר על מדידת מרחק אוקלידי בין נקודות במרחב או בשם הפשוט יותר שלו, שימוש פיתגורס כדי לחשב את היתר במשולש ישר זווית.

בסוף פעילות 11 במדריך זה השתמשנו ברשת נוירונים מלאכותיים במבנה (3-5-4) כדי לסווג פרחי אירוס ל-3 משפחות. למדנו שבחלק מהמקרים תוצאות הסיווג של רשת הנוירונים היו יותר קרובות לניסיון לסווג מספרים אקראיים מאשר לסווג משפחות של אירוסים. בסוף פעילות זו נחזור ונבחן את יעילות האלגוריתם KNN על סיווג משפחות האירוסים ונבחן האם לא נכון יותר לבצע את הסיווג תוך שימוש ב-KNN על פני שימוש ב-ANN.

כדי להבין את עקרון הפעולה של אלגוריתם KNN נחקור את מערך הנקודות 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] ])
```

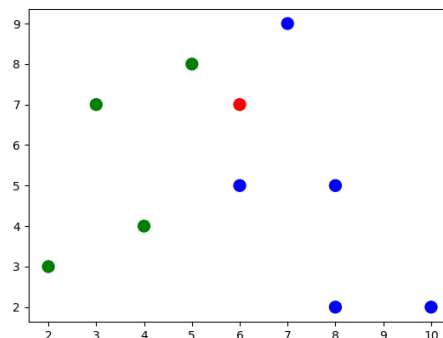
```
categories = np.array([0,1,1,1,1,2,2,2,2,2])
```

```
colormap = np.array(['r', 'g', 'b'])
```

```
plt.scatter(data[:,0], data[:,1], s=100, c=colormap[categories])
```

```
plt.show()
```

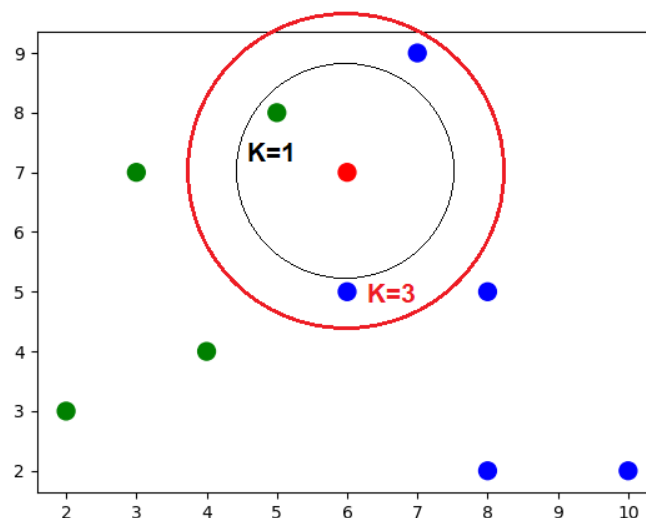
נקבל את הפלט הבא:



קיבלנו אם כן מערך הכולל 3 סוגי נקודות. נקודות כחולות, נקודות ירוקות ונקודה אחת בצבע אדום. בהנחה שיש קשר כלשהו בין כל הנקודות הכחולות וקשר אחר בין כל הנקודות הירוקות. נשאלת השאלה האם הנקודה האדומה שייכת לקבוצת הנקודות הכחולות או לקבוצת הירוקות?

על פי אלגוריתם KNN אנו יכולים לשער לאיזו קבוצה שייכת הנקודה האדומה על ידי כך שנבדוק מי השכנים שלה וכמה שכנים יש לאותה נקודה מכל סוג (כחולים וירוקים) לפי עקרון זה ניתן לשער שככל שיש יותר נקודות מסוג מסוים באותו האזור שאנו בודקים, הנקודה הנבדקת שייכת לאותה קבוצה.

נשאלת השאלה מה גודל האזור? נבחן את הסוגייה על ידי האיור הבא:



כאשר הגדרנו באיור את האזור בעיגול שחור מצאנו שבתוך אותו השטח קיימת רק נקודה אחת בצבע ירוק. כאשר הגדרנו את האזור בעיגול אדום קיבלנו בשטח שיש 2 נקודות כחולות ונקודה ירוקה אחת.

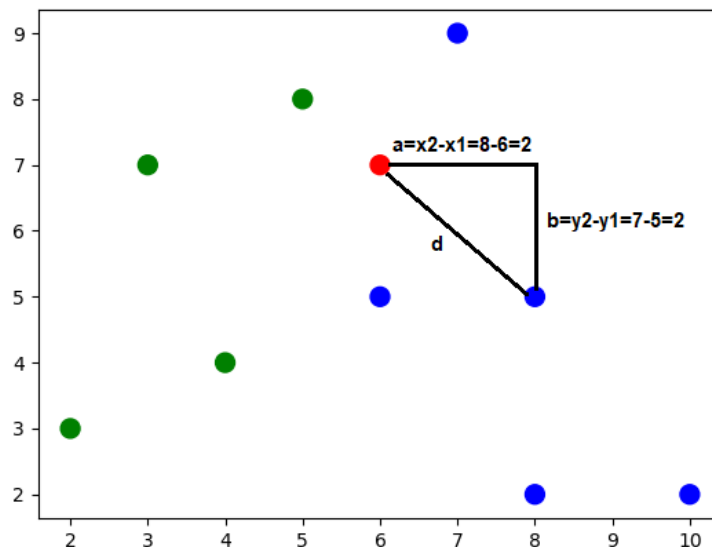
גודל השטח נקרא  $K$  והוא מספר המייצג את מספר הפריטים הכי קרובים לאותה נקודה שאנו בודקים. כאשר  $K=1$  אנו בודקים מי הנקודה הכי קרובה לנקודה הנבדקת. כאשר  $K=3$  אנו בודקים מי שלוש הנקודות הכי קרובות לנקודה הנבדקת. באופן זה ניתן לחזות האם הנקודה שאנו בודקים שייכת לקבוצה הכחולה או לקבוצה הירוקה.

מהדוגמה שבאיור אנו כבר מקבלים בעייה בסיווג הנקודה האדומה. כי כאשר  $K=1$  הנקודה הנבדקת שייכת לקבוצה הירוקה. אך כאשר  $K=3$  הנקודה הנבדקת שייכת לקבוצה הכחולה.

מדוגמה זו ניתן להבין לצד הפשטות של השיטה גם את הקושי שלה. כי שינוי מספר הקרובים  $K$  משפיע על קביעת הסיווג.

### מרחק אוקלידי בין נקודות במרחב

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



המרחק  $d$  בין 2 הנקודות באיור מחושב על ידי המשוואה הבא:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

נדגים את עקרון מדידת מרחק דרך דוגמת הקוד הבא:

```
import numpy as np

data = np.array([ [ 6.0 , 7.0],
                  [ 2.0 , 3.0],
                  [ 3.0 , 7.0],
```

```
[ 4.0 , 4.0],  
[ 5.0 , 8.0],  
[ 6.0 , 5.0],  
[ 7.0 , 9.0],  
[ 8.0 , 5.0],  
[ 8.0 , 2.0],  
[10.0 , 2.0]])
```

```
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)  
  
print("\nAll Euclidean Distance from point: ",data[0], "\n")  
for point in data:  
    distance = euclidean_distance(data[0], point)  
    print(distance)
```

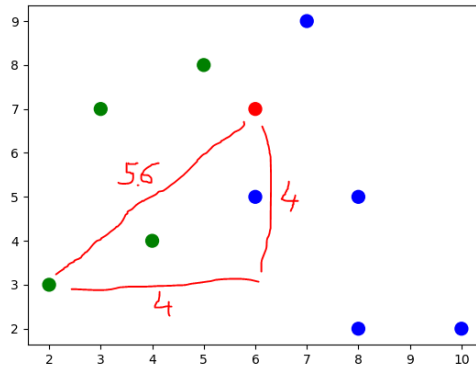
נקבל את הפלט הבא:

```
All Euclidean Distance from point: [6. 7.]  
  
0.0  
5.656854249492381  
3.0  
3.605551275463989  
1.4142135623730951  
2.0  
2.23606797749979  
2.8284271247461903  
5.385164807134504  
6.4031242374328485
```

נבחן את שלוש התוצאות הראשונות בפלט:

הערך הראשון הוא 0 כי המרחק בין נקודה לעצמה הוא אפס.

הערך השני הוא 5.656 כי המרחק בין נקודה (6,7) לנקודה (2,3) הוא שורש של 4 בריבוע ועוד 4 בריבוע.



הערך השלישי הוא 3 כי המרחק בין נקודה (6,7) לנקודה (3,7) הוא פשוט 3 כי יש שינוי רק על ציר x.

### מרחק אוקלידי על גבי מערכת מרובת צירים

נכתוב מחדש את הפעולה euclidean\_distance כדי שתתאים לחישוב מרחק אוקלידי ביותר מ-2 מימדים. להלן דוגמת קוד המחשבת מרחקים מ-9 נקודות על פני מערכת צירים בעלת 3 מימדים, כאשר המרחק מחושב מהנקודה (5,5,5):

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np

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] ])

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

print("\nAll Euclidean Distance from point: ",data[0], "\n")
for point in data:
    distance = euclidean_distance(data[0], point)
    print(distance)

categories = np.array([0,1,1,1,1,2,2,2,2,2])
colormap = np.array(['r', 'g', 'b'])
fig = plt.figure()
ax = Axes3D(fig)
ax.scatter(data[:,0], data[:,1],data[:,2], s=150, c=colormap[categories])
plt.show()

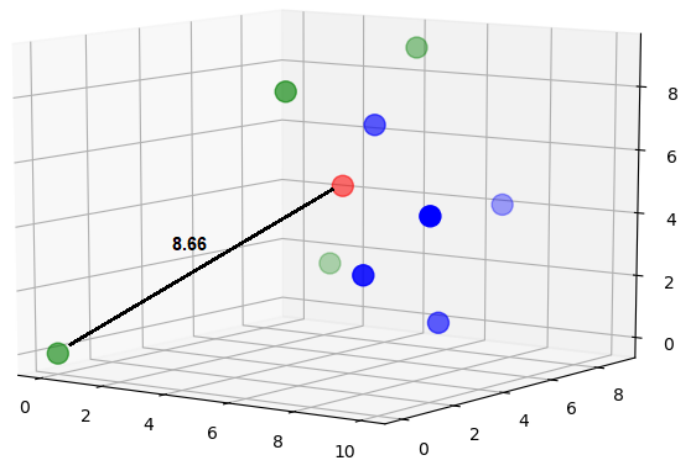
```

נקבל את הפלט הבא:

```

All Euclidean Distance from point: [5. 5. 5.]
0.0
8.660254037844387
4.123105625617661
3.3166247903554
5.0
2.23606797749979
4.58257569495584
5.0
4.69041575982343
5.830951894845301

```



ניתן לראות שהפעולה חישובה נכון את המרחק בין נקודה (0,0,0) לבין הנקודה (5,5,5) כ- 8.66 כי:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

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

השלב הבא בכתיבת קוד יהיה לכתוב פעולה בשם KNN. הפעולה תקבל מערך הנקודות כפרמטר בשם train, את הנקודה שאליה אנו רוצים לחפש את הנקודות הקרובות ביותר בשם test ופרמטר נוסף בשם K שקובע את מספר הנקודות אותה הפעולה תחזיר.

פעולה זו תבצע את ההוראות הבאות:

- 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 ואת ערכי הנקודה שנבדקה.
- לאחר סיום מדידת כל הנקודות נמייין את המערך על פי המרחקים.
- נמחק את האיבר הראשון במערך בגלל שהוא מודד מרחק לנקודה test כך שברור שהערך תמיד יהיה אפס.
- לבסוף נעבור על הלולאה הממוינת ונחליץ ממנה את K האיברים שאותם אנו רוצים לקבל.

הלהן מימוש הקוד:

```
import matplotlib.pyplot as plt
import numpy as np

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

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
```

```

def KNN(train, test, K):
    distances = []
    NN = []
    for p in train:
        dist = euclidean_distance(test, p)
        distances.append((p, dist))
    distances.sort(key=lambda dist: dist[1])
    distances = np.delete(distances,[0], axis=0)
    for i in range(K):
        NN.append(distances[i][0])
    return NN

neighbors = KNN(data, data[0], 3)
for neighbor in neighbors:
    print(neighbor)

```

נקבל את הפלט הבא עבור  $K=3$ :

```

[5. 8.]
[6. 5.]
[7. 9.]

```

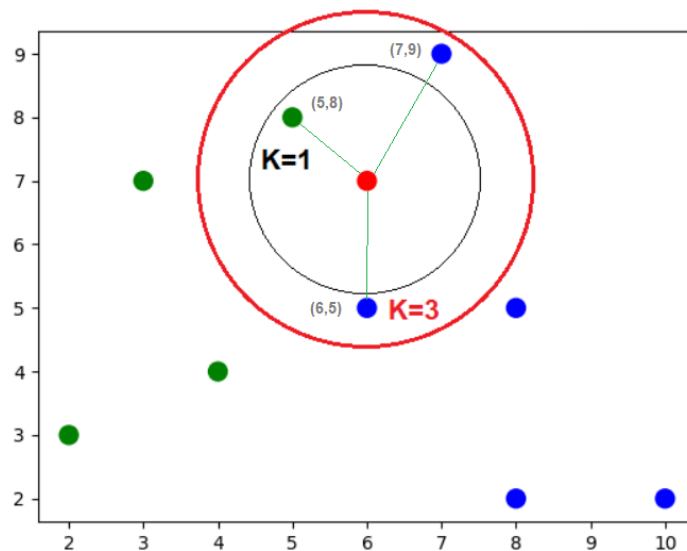
ואת פלט הבא עבור  $K=1$ :

```

[5. 8.]

```

נבחן את התוצאות מול התרשים הבא:





השלב האחרון בכתיבת קוד ליישום אלגוריתם KNN במכונה לומדת יהיה לכתוב את הפעולה predict. הפעולה 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 ופרמטר נוסף בשם K שקובע את מספר הנקודות שעל פיהם נקבע לאיזו קבוצה שייכת הנקודה test. הפעולה תחזיר פרמטר אחד שהוא מספר הקבוצה.

### פעולה זו תבצע את השלבים הבאים:

נזמן את הפעולה KNN כדי לקבל את שלוש הנקודות הכי קרובות לנקודה test להלן צילום מסך של הערכים שמקבלים מהפעולה KNN:

```

(array([5., 8.]), 1.4142135623730951, 1), (array...
✓ 0: (array([5., 8.]), 1.4142135623730951,
  > 0: array([5., 8.])
  1: 1.4142135623730951
  > 2: 1
  __len__: 3
✓ 1: (array([6., 5.]), 2.0, 2)
  > 0: array([6., 5.])
  1: 2.0
  > 2: 2
  __len__: 3
✓ 2: (array([7., 9.]), 2.23606797749979, 2)
  > 0: array([7., 9.])
  1: 2.23606797749979
  > 2: 2
  __len__: 3
  __len__: 3
  
```

על פי האיור ניתן לראות שאנו מקבלים מערך הכולל 3 איברים שבו כל איבר בנוי ממערך נוסף הכולל את ערכי הנקודה, המרחק ומספר הקבוצה שאליה שייכת הנקודה.

השלב הבא יהיה לספור כמה נקודות יש לכל קבוצה ולהציג את הקבוצה שיש לה הכי הרבה נקודות:

```

out = [row[-1] for row in neighbors]
return max(set(out), key=out.count)
  
```

```

return nn
def predict(train, test, lbl, K):
    neighbors = find_neighbors(test, train, K)
    out = [row[-1] for row in neighbors]
    return max(set(out), key=out.count)
  
```

ניתן לראות שערך המערך out שווה לערכים [1,2,2]

בשלב האחרון הפעולה max תחזיר את הספרה שמופיעה הכי הרבה פעמים, במקרה שלנו 2.

**הערה דידיקטית:** כדי לקצר את קוד הפעולה השתמשתי בלולאה מקוצרת העוברת על האיבר האחרון (-1) של הרשימה neighbor. כמו כן השתמשתי בפעולה max הכוללת פרמטר בשם key=out.count הסוכם את כמות האיברים במערך.

נוסף על כתיבת הפעולה predict יש צורך לשנות את הפעולה KNN כך שתקבל גם את מערך התגיות lbl. כמו כן נעשה שינוי במבנה המערך distances שיכיל גם את התוויות של כל אחד מהנקודות. להלן מימוש הפעולה KNN לאחר השינויים הנדרשים בתרגיל:

```
def KNN(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=lambda dist: dist[1])
    # distances = np.delete(distances,[0], axis=0)
    NN = []
    for i in range(K):
        NN.append(distances[i])
    return NN
```

להלן קוד התוכנית המלא:

```
import matplotlib.pyplot as plt
import numpy as np

train_data = np.array( [
    [ 2.0 , 3.0 ],
    [ 3.0 , 7.0 ],
    [ 4.0 , 4.0 ],
    [ 5.0 , 8.0 ],
    [ 6.0 , 5.0 ],
    [ 7.0 , 9.0 ],
    [ 8.0 , 5.0 ],
    [ 8.0 , 2.0 ],
    [10.0 , 2.0 ] ])

train_lbl = np.array( [
    [ 1 ],
    [ 1 ],
```

```
[ 1],  
[ 1],  
[ 2],  
[ 2],  
[ 2],  
[ 2],  
[ 2 ] ])
```

```
test_data = np.array( [[ 6.0 , 7.0 ]])
```

```
def euclidean_distance(p1, p2):
```

```
    d = 0.0
```

```
    for i in range(len(p1[0])):
```

```
        a = float(p1[0,i])
```

```
        b = float(p2[i])
```

```
        d += np.power((a-b),2)
```

```
    d = np.sqrt(d)
```

```
    return d
```

```
def KNN(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=lambda dist: dist[1])
```

```
    NN = []
```

```
    for i in range(K):
```

```
        NN.append(distances[i])
```

```
    return NN
```

```
def predict(train, test, lbl, K):
```

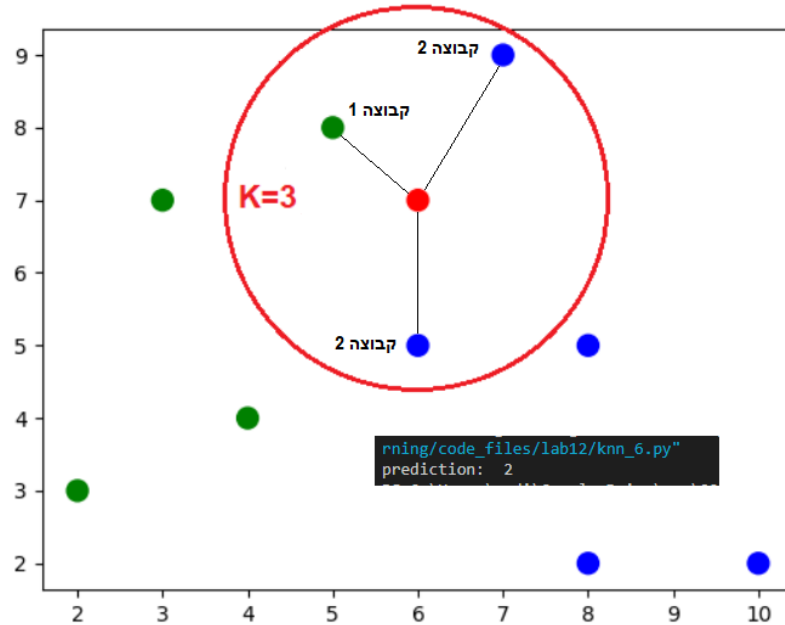
```
    neighbors = KNN(train, test, lbl, K)
```

```
    out = [row[-1] for row in neighbors]
```

```
    return max(set(out), key=out.count)
```

```
prediction = predict(train_data, test_data, train_lbl, 3)
print("prediction: ", prediction)
```

נקבל את הפלט הבא:

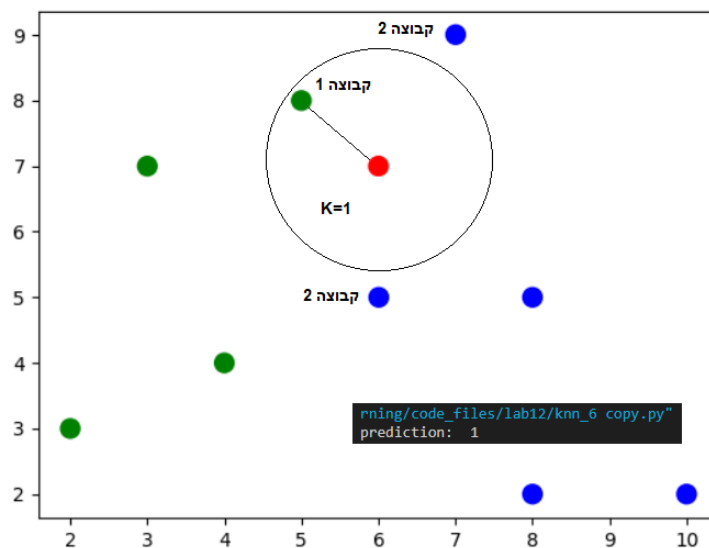


קיבלנו שהנקודה האדומה שייכת לקבוצה מספר 2.

### תרגיל

שנו את הפרמטר  $K=3$  ל-  $K=1$  ובדקו האם גם במצב זה הנקודה האדומה שייכת לקבוצה 2. נמקו את תשובתכם.

### פתרון



קיבלנו הפעם שיוך לקבוצה 1 כי עבור  $K=1$  יש רק נקודה אחת מקבוצה 1 שהיא בעצם הנקודה הכי קרובה.

## במשימה 1 - מכונה לומד לסיווג פרחים תוך שימוש באלגוריתם KNN.

בפעילות הקודמת כתבנו קוד העושה שימוש ברשת נוירונים מלאכותית ANN במטרה לבצע סיווג של תמונות פרחים ל-3 סוגי אירוסים. למדנו שתצאות הסיווג לא היו ממש מדויקות והיה צורך לבצע מספר אימונים לרשת כדי לקבל תוצאות סבירות. על כן עולה החשד שאולי קיים אלגוריתם אחר שיכול בצורה פשוטה יותר לבצע את משימת סיווג הפרחים. במשימה זו נבחן את אלגוריתם KNN כדי לסווג פרחים.

להלן חזרה על מבנה הנתונים לסיווג הפרחים:

מערך הנתונים הכולל מידע מתויג על 3 סוגים שונים של אירוסים. להלן תמונות של שלשות האירוסים:

Iris Setosa 1 0 0



Iris Versicolor 0 1 0



Iris Virginica 0 0 1

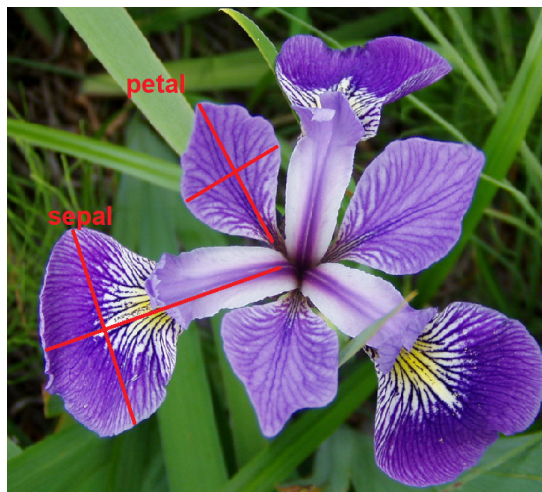


מקור התמונה: [https://commons.wikimedia.org/wiki/Iris\\_Iridaceae](https://commons.wikimedia.org/wiki/Iris_Iridaceae) (לשימוש חוזר עם אפשרות לשינויים)

מאגר הנתונים כולל נתונים מתוקפים מחקרית המסווגים שלושת פרחים על פי 4 מאפיינים שהם:

- אורך ה- petal (אורך עלי כותרת של איריס)
- רוחב ה- petal (רוחב עלי כותרת של איריס)
- אורך ה- sepal (אורך עלי הגביע של איריס)
- רוחב ה- sepal (רוחב עלי הגביע של איריס)

להלן תמונה שתסביר את המאפיינים כל גבי תמונה של הפרח



ניתן להוריד את קובץ הנתונים ישירות מהקישור הבא:

[https://www.neuraldesigner.com/files/datasets/iris\\_flowers.csv](https://www.neuraldesigner.com/files/datasets/iris_flowers.csv)

נפתח את קובץ הנתונים על ידי תוכנת Excel ונקבל את הנתונים הבאים:

	A	B	C	D	E
1	sepal_length	sepal_width	petal_length	petal_width	class
2	5.1	3.5	1.4	0.2	iris_setosa
3	4.9	3	1.4	0.2	iris_setosa
4	4.7	3.2	1.3	0.2	iris_setosa
5	4.6	3.1	1.5	0.2	iris_setosa
6	5	3.6	1.4	0.2	iris_setosa
7	5.4	3.9	1.7	0.4	iris_setosa
8	4.6	3.4	1.4	0.3	iris_setosa
9	5	3.4	1.5	0.2	iris_setosa
10	4.4	2.9	1.4	0.2	iris_setosa
11	4.9	3.1	1.5	0.1	iris_setosa
12	5.4	3.7	1.5	0.2	iris_setosa
13	4.8	3.4	1.6	0.2	iris_setosa
14	4.8	3	1.4	0.1	iris_setosa
15	4.3	3	1.1	0.1	iris_setosa
16	5.8	4	1.2	0.2	iris_setosa
17	5.7	4.4	1.5	0.4	iris_setosa
18	5.4	3.9	1.3	0.4	iris_setosa
19	5.1	3.5	1.4	0.3	iris_setosa
20	5.7	3.8	1.7	0.3	iris_setosa
97	5.7	3	4.2	1.2	iris_versicolor
98	5.7	2.9	4.2	1.3	iris_versicolor
99	6.2	2.9	4.3	1.3	iris_versicolor
100	5.1	2.5	3	1.1	iris_versicolor
101	5.7	2.8	4.1	1.3	iris_versicolor
102	6.3	3.3	6	2.5	iris_virginica
103	5.8	2.7	5.1	1.9	iris_virginica
104	7.1	3	5.9	2.1	iris_virginica
105	6.3	2.9	5.6	1.8	iris_virginica
106	6.5	3	5.8	2.2	iris_virginica
107	7.6	3	6.6	2.1	iris_virginica
108	4.9	2.5	4.5	1.7	iris_virginica
109	7.3	2.9	6.3	1.8	iris_virginica
110	6.7	2.5	5.8	1.8	iris_virginica

קיבלנו קובץ המכיל דגימות של 150 פרחים (50 מכל סוג) כאשר לכל פרח מדדו את 4 המאפיינים שקבענו כדי להבדיל בין השלושה. כמובן הקובץ מכיל עמודה חמישית הכוללת את שם הפרח שאותו מדדו. מכאן שיש לנו 150 שורות של מידע מתויג.

בשלב הבא נקלוט את נתוני הקובץ לתוך מערך numpy שאנו מכירים. להלן דוגמת הקוד:

```
import numpy as np
from termcolor import colored

vir_iris_data = np.genfromtxt('iris_for_ML.csv', delimiter=',')
print(vir_iris_data)
```

נקבל את הפלט הבא:

```
[5.1 3.5 1.4 0.2 1. ]
[4.9 3. 1.4 0.2 1. ]
[4.7 3.2 1.3 0.2 1. ]
[4.6 3.1 1.5 0.2 1. ]
[5. 3.6 1.4 0.2 1. ]
[5.4 3.9 1.7 0.4 1. ]
[4.6 3.4 1.4 0.3 1. ]
```

בשלב הבא יש צורך להתאים את מערך הנתונים כדי שיכנס לתוך אלגוריתם KNN באופן הבא:

- נשנה את שם הפרח למספרים 1 עד 3 בהתאמה.
- שנערבב את סוגי הפרחים.
- נחלק את המערך ל-4 מערכים על פי הפירוט הבא:
  - a. מערך אימונים הכולל רק את הנתונים.
  - b. מערך הכולל את התגיות של מערך האימונים.
  - c. מערך בדיקה הכולל רק את הנתונים.
  - d. מערך הכולל את התגיות של מערך הבדיקה.

להלן מימוש הדברים בקוד:

```
import numpy as np
from termcolor import colored

vir_iris_data = np.genfromtxt('iris_for_ML.csv', delimiter=',')
random_iris_data = np.random.permutation(vir_iris_data)
test_data = random_iris_data[0:10,:4]
train_data = random_iris_data[10:,:4]
test_lbl = random_iris_data[0:10,4:]
train_lbl = random_iris_data[10:,4:]
print("\nrandom_iris_data: \n",colored(random_iris_data, 'red'),"\n")
print("\ntrain_data: \n",colored(train_data, 'blue'),"\n")
print("\ntrain_lbl: \n",colored(train_lbl, 'blue'),"\n")
print("\ntest_data: \n",colored(test_data, 'green'),"\n")
print("\ntest_lbl: \n",colored(test_lbl, 'green'),"\n")
```

נבחן את הנתונים שקיבלנו:

```
random_iris_data:      test_data:      train_data:      test_lbl:      train_lbl:
[[5.7 3.8 1.7 0.3 1. ]  [[5.7 3.8 1.7 0.3]  [[5.7 2.6 3.5 1. ]  [[1.]  [[2.]
[6.  2.9 4.5 1.5 2. ]  [6.  2.9 4.5 1.5]  [5.5 2.4 3.7 1. ]  [2.]  [2.]
[6.9 3.1 4.9 1.5 2. ]  [6.9 3.1 4.9 1.5]  [6.6 2.9 4.6 1.3]  [2.]  [2.]
[5.9 3.  5.1 1.8 3. ]  [5.9 3.  5.1 1.8]  [7.1 3.  5.9 2.1]  [3.]  [3.]
[6.3 2.7 4.9 1.8 3. ]  [6.3 2.7 4.9 1.8]  [5.  3.3 1.4 0.2]  [3.]  [1.]
[5.4 3.7 1.5 0.2 1. ]  [5.4 3.7 1.5 0.2]  [7.7 3.  6.1 2.3]  [1.]  [3.]
[4.6 3.2 1.4 0.2 1. ]  [4.6 3.2 1.4 0.2]  [4.9 2.5 4.5 1.7]  [1.]  [3.]
[5.1 3.8 1.9 0.4 1. ]  [5.1 3.8 1.9 0.4]  [4.7 3.2 1.3 0.2]  [1.]  [1.]
[4.9 3.1 1.5 0.1 1. ]  [4.9 3.1 1.5 0.1]  [4.9 2.4 3.3 1. ]  [1.]  [2.]
[5.1 3.4 1.5 0.2 1. ]  [5.1 3.4 1.5 0.2]] [5.7 2.8 4.5 1.3]  [1.]]  [2.]
[5.7 2.6 3.5 1.  2. ]  [7.9 3.8 6.4 2. ]  [3.]
[5.5 2.4 3.7 1.  2. ]  [4.9 3.1 1.5 0.1]  [1.]
[6.6 2.9 4.6 1.3 2. ]  [6.1 2.8 4.7 1.2]  [2.]
[7.1 3.  5.9 2.1 3. ]  [4.8 3.4 1.9 0.2]  [1.]
[5.  3.3 1.4 0.2 1. ]  [6.9 3.2 5.7 2.3]  [3.]
[7.7 3.  6.1 2.3 3. ]  [1.]
[4.9 2.5 4.5 1.7 3. ]  [1.]
[4.7 3.2 1.3 0.2 1. ]  [1.]
```

נשלב את קוד הכנת מבנה הנתונים יחד עם שלוש הפעולות שכתבנו למימוש אלגוריתם KNN ונקבל את הקוד הבא:

```
import numpy as np
from termcolor import colored

vir_iris_data = np.genfromtxt('iris_for_ML.csv', delimiter=',')
random_iris_data = np.random.permutation(vir_iris_data)
test_data = random_iris_data[0:10,:4]
train_data = random_iris_data[10:,:4]
test_lbl = random_iris_data[0:10,4:]
train_lbl = random_iris_data[10:,4:]

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
```



```

def KNN(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=lambda dist: dist[1])
    NN = []
    for i in range(K):
        NN.append(distances[i])
    return NN

def predict(train, test, lbl, K):
    neighbors = KNN(train, test, lbl, K)
    out = [row[-1] for row in neighbors]
    return max(set(out), key=out.count)

for i in range(len(test_data)):
    p = predict(train_data, test_data[i], train_lbl, 1)
    print("test_lbl:" , colored(test_lbl[i], 'green') , "prediction: " , colored(p, 'green') )

```

נקבל את הפלט הבא:

```

test_lbl: [1.] prediction: 1.0
test_lbl: [2.] prediction: 2.0
test_lbl: [3.] prediction: 3.0
test_lbl: [1.] prediction: 1.0
test_lbl: [3.] prediction: 3.0
test_lbl: [1.] prediction: 1.0
test_lbl: [2.] prediction: 2.0
test_lbl: [1.] prediction: 1.0
test_lbl: [3.] prediction: 3.0
test_lbl: [1.] prediction: 1.0

```

קיבלנו התאמה מלאה בין מערך התגיות של הפרחים שבדקנו לבין פלט האלגוריתם עבור כל אחד מהם. ראינו כיצד אלגוריתם KNN שלכל הדעות, אחרי שהבנו כל אחד מהם, פשוט יותר מרשת נירונים מלאכותיים ANN ומצליח לזהות פרחים ביעילות מושלמת!

מפעילות זו למדנו כיצד לממש אלגוריתם KNN למדנו שניתן בתנאים משויימים ליישם מכונות לומדות באופן יעיל יותר תוך שימוש באלגוריתם כמו KNN על פני ANN.

ניתן לסכם ש-KNN הוא אלגוריתם עצלן לימודית כי בניגוד ANN שקודם לומד ולאחר מכן יכולה לסווג. אלגוריתם KNN לומד ומסווג רק כאשר מתקבלת בקשה לסיווג. מכאן שזו גם אחת החסרונות של KNN.

אלגוריתם זה יתקשה לתפקד כאשר מדובר על כמויות גדולות של אימון (למידה). בקיצור אלגוריתם עצלן!  
לומד תוך כדי הבחינה?!

## תנאי השימוש

תנאי השימוש במסמך זה הם לפי הסטנדרט הבא:

You are free:

to Share – to copy, distribute and transmit the material  
to Remix – to adapt the material

Under the following conditions:

Attribution — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.

NonCommercial — You may not use the material for commercial purposes.

ShareAlike — If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.