m1-hw2

January 27, 2024

1 Problem 2 (30 points)

1.1 Problem Description

Here, you will perform *weighted* KNN regression.

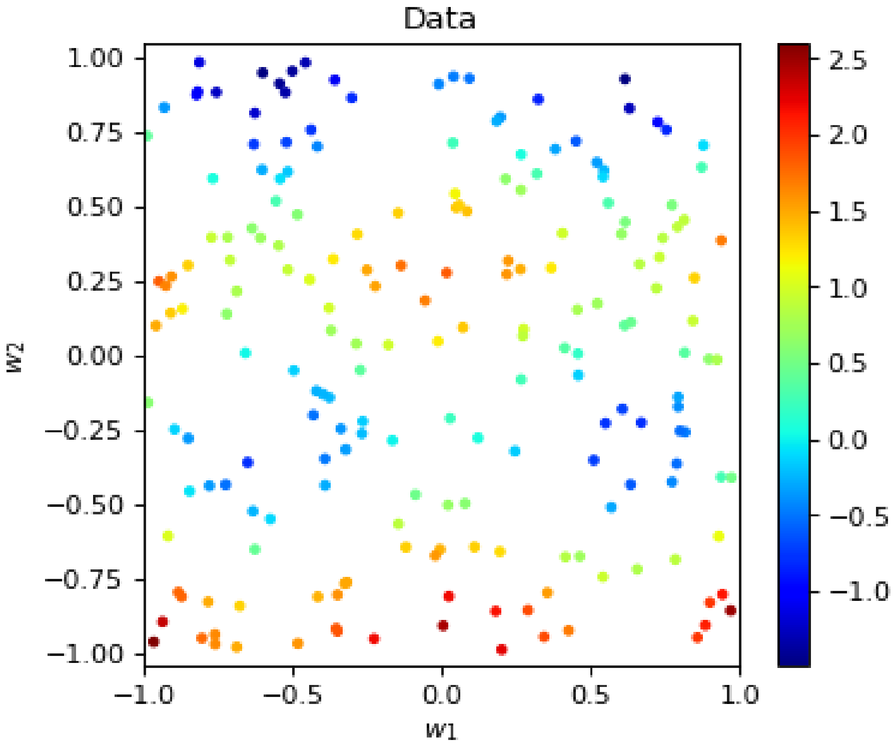
After you write your own code for weighted KNN regression, you will also try out sklearn's built-in KNN regressor.

Fill out the notebook as instructed, making the requested plots and printing necessary values.

Summary of deliverables: Functions: - `weighted_knn(w1, w2, k)`

Plots: - 3 plots of by-hand KNN results - 3 plots of sklearn.

[352]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsRegressor

# Data generation -- don't change
np.random.seed(42)
N = 200
w1_data = np.random.uniform(-1,1,N)
w2_data = np.random.uniform(-1,1,N)
L_data = np.cos(4*w1_data) + np.sin(5*w2_data) + 2*w1_data**2 - w2_data/2
# (end of data generation)

plt.figure(figsize=(5,4.2),dpi=80)
plt.scatter(w1_data,w2_data,s=10,c=L_data,cmap="jet")
plt.colorbar()
plt.axis("equal")
plt.xlabel("$w_1$")
plt.ylabel("$w_2$")
plt.xlim(-1,1)
plt.ylim(-1,1)
plt.title("Data")
plt.show()
```



1.2 Weighted KNN function

Here, define a function, `weighted_knn(w1, w2, k)`, which takes in a point at [`w1`, `w2`] and a `k` value, and returns the weighted KNN prediction.

- As in the lecture activity, data is in the variables `w1_data`, `w2_data`, and `L_data`.
- You can create as many helper functions as you want
- The key difference between unweighted and weighted KNN is summarized below:

Unweighted KNN 1. Find the k data points closest to the target point w 2. Get the output values at each of these points 3. Average these values together: this is the prediction at w

Weighted KNN 1. Find the k data points closest to the target point w 2. Compute the proximity of each of these points as $\text{prox}_i = 1/(\text{distance}(w, w_i) + 1e - 9)$ 3. For each w_i, multiply prox_i by the output value at w_i, and divide by the sum of all k proximities 4. Add all k of these results together: this is the prediction at w

[353]:
```
def weighted_knn(w1, w2, k):
    # YOUR CODE GOES HERE
    # distance calculation
    distance = np.sqrt((w1_data - w1)**2 + (w2_data - w2)**2)
    # k nearest neighbors
    k = np.argsort(distance)[:k]
```

```
    # proximity calculation
    proximity = 1/(distance[k] + 1e-9)
    # weighted average
    prediction = np.sum(proximity * L_data[k])/np.sum(proximity)
    return prediction
```

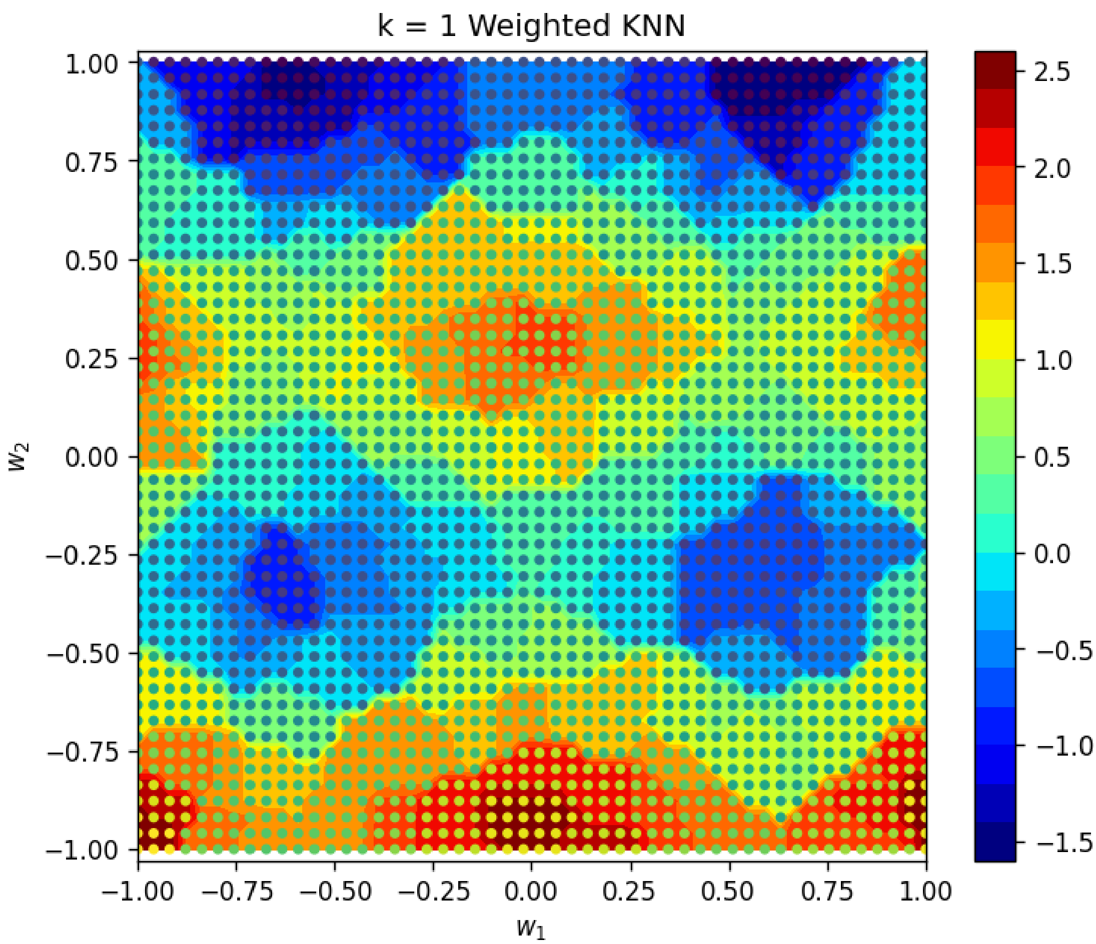
1.3 Plotting

Now create 3 plots showing KNN regressor predictions for k values [1, 5, 25].

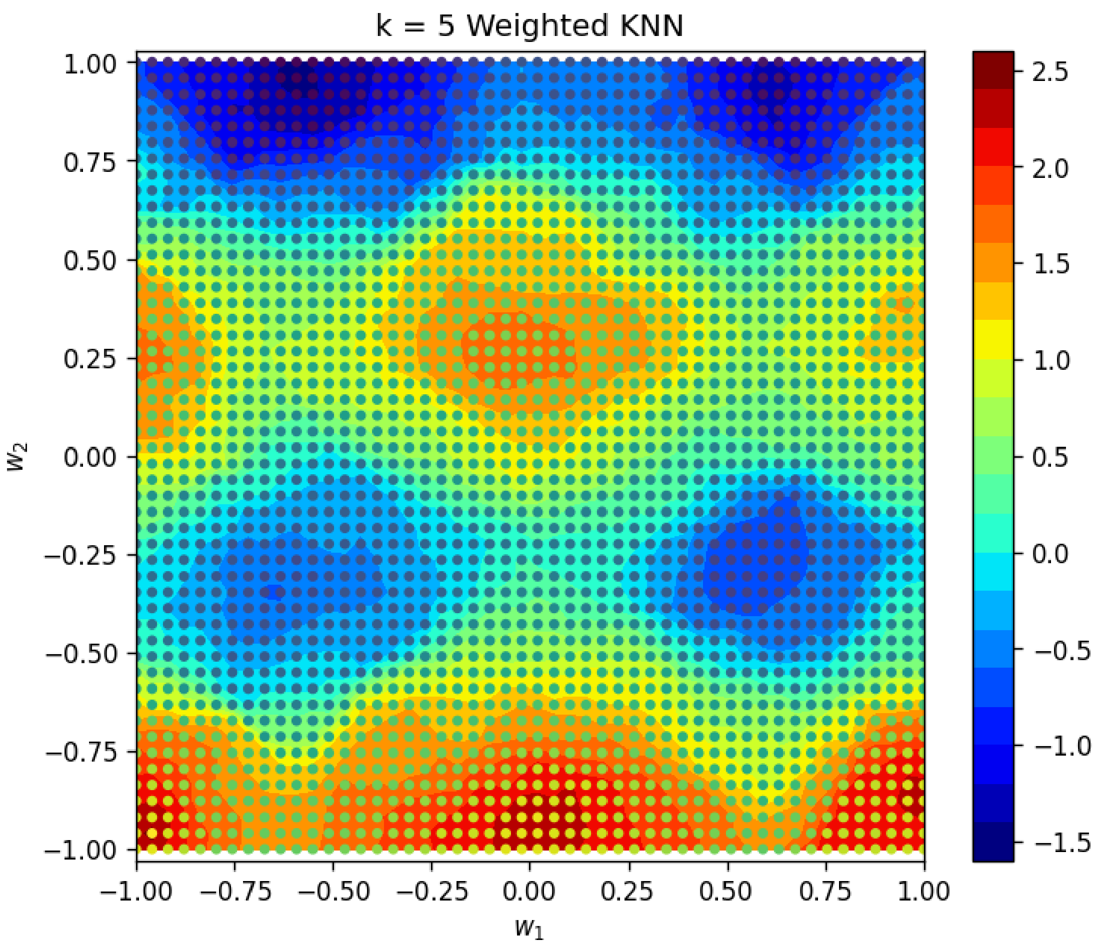
You should plot a 50x50 grid of points on a grid for `w1` and `w2` values between -1 and 1. Consult the lecture activity for how to do this.

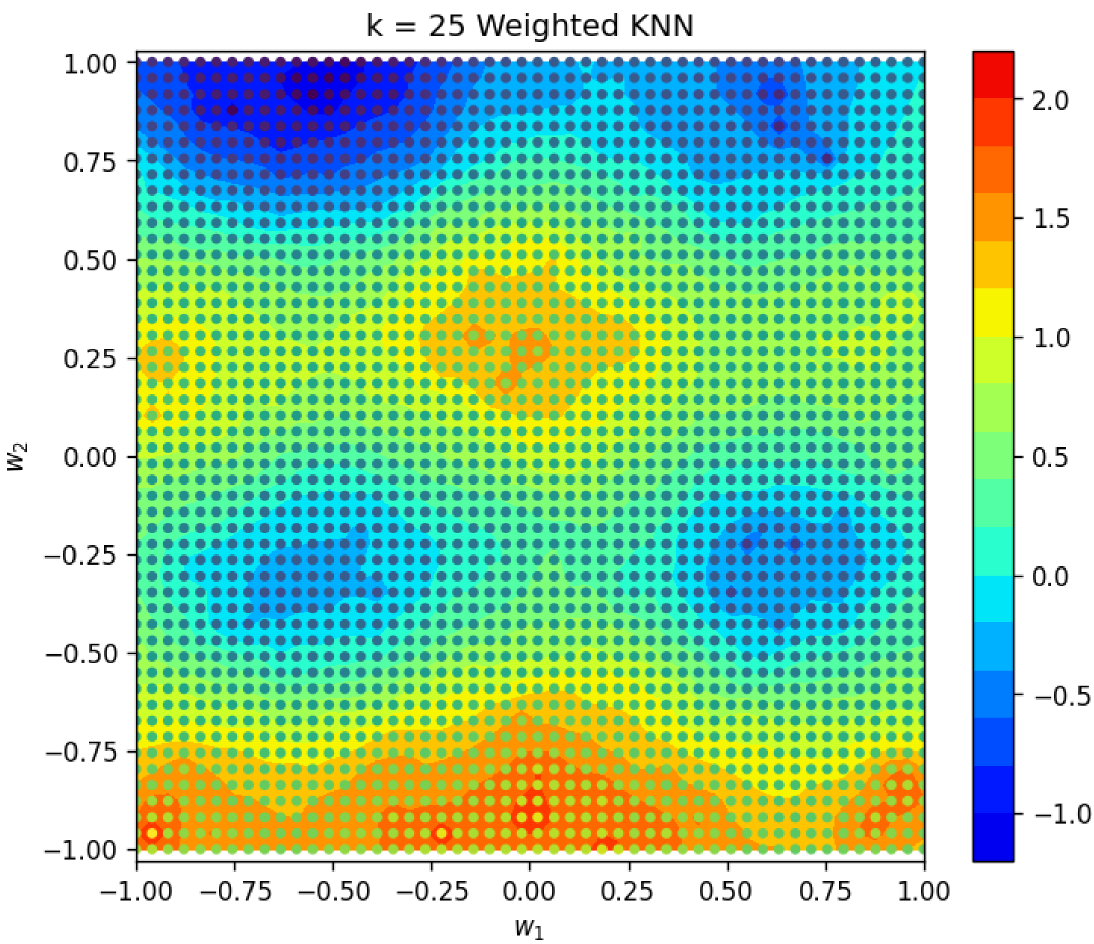
We recommend creating a function, e.g. `plot(k)`, so that you need to rewrite less code.

[354]:
```
# YOUR CODE GOES HERE
# Visualize results for k = 1, 5, and 25

def plot(k):
    w1 = np.linspace(-1, 1, 50)
    w2 = np.linspace(-1, 1, 50)
    w1s, w2s = np.meshgrid(w1, w2)
    w1_grid, w2_grid = w1s.flatten(), w2s.flatten()
    L_grid = np.zeros_like(w1_grid)
    for i in range(len(w1_grid)):
        L_grid[i] = weighted_knn(w1_grid[i], w2_grid[i], k)
    plt.figure(figsize=(7,5.8),dpi=120)
    contour = plt.contourf(w1s, w2s, L_grid.reshape(w1s.shape), 20, cmap="jet",␣
 ↪vmin=-1.5, vmax=2.5)
    plt.colorbar(contour, ticks = [-1.5, -1 , -0.5, 0, 0.5, 1, 1.5, 2, 2.5])
    plt.scatter(w1_grid, w2_grid, c=L_grid, s=10)
    plt.axis("equal")
    plt.xlabel("$w_1$")
    plt.ylabel("$w_2$")
    plt.xlim(-1,1)
    plt.ylim(-1,1)
    plt.title("k = " + str(k) + " Weighted KNN")
    plt.show()

# Visualize results for k = 1
plot(1)
# Visualize results for k = 5
plot(5)
# Visualize results for k = 25
plot(25)
```



k = 1 Weighted KNN
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
w_2
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
w_1
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5


k = 5 Weighted KNN
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
w_2
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
w_1
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5



1.4 Using SciKit-Learn

We can also use sklearn's `KNeighborsRegressor()`, which is a very efficient implementation of KNN regression.

The code to do this has been done for one case below. First, make note of how this is done.

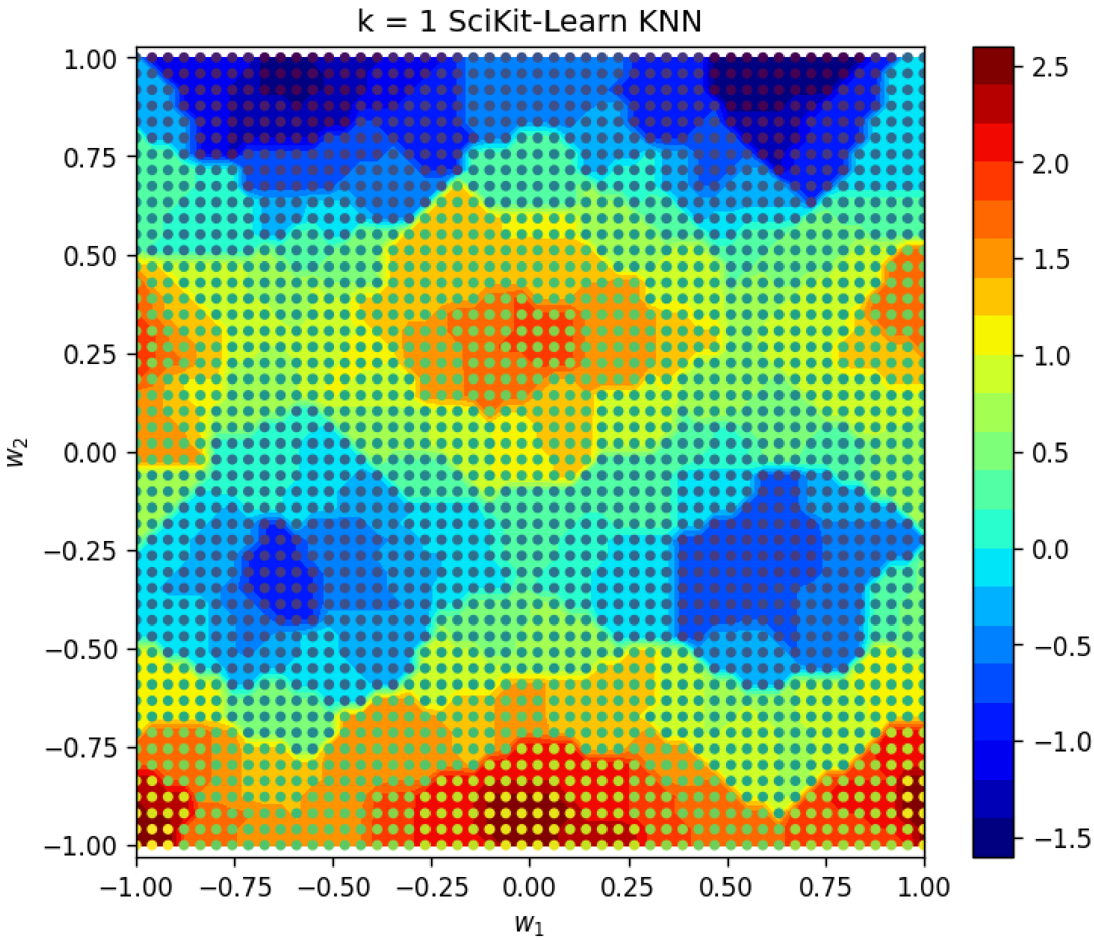
[355]:
```
model = KNeighborsRegressor(n_neighbors = 1, weights="distance")
X = np.vstack([w1_data,w2_data]).T
model.fit(X, L_data)

# Get a prediction at a point (0, 0):
print(model.predict(np.array([[0,0]])))
```

```
[1.19743607]
```

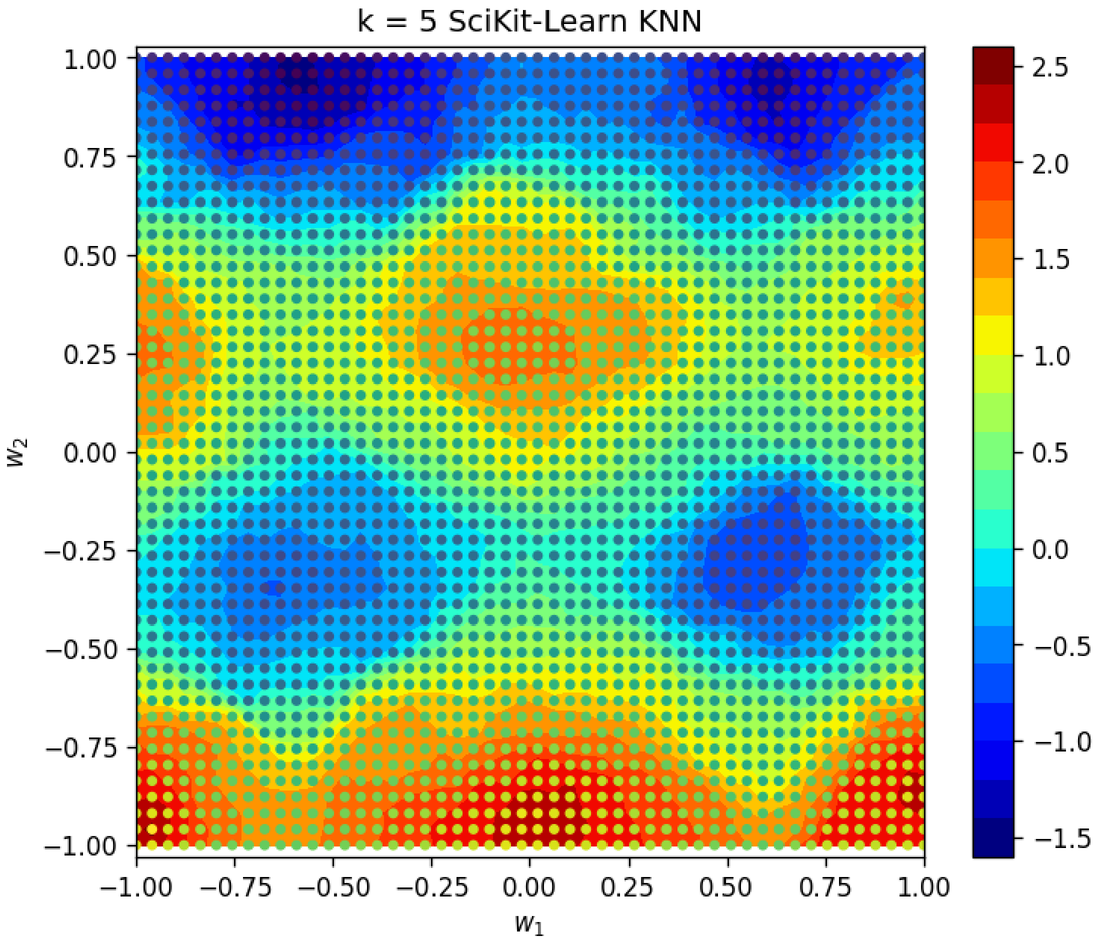
Now create 3 plots for the same values of k as before, using this KNN implementation instead. You can make sure these are visually the same as your from-scratch KNN regressor.

[356]:

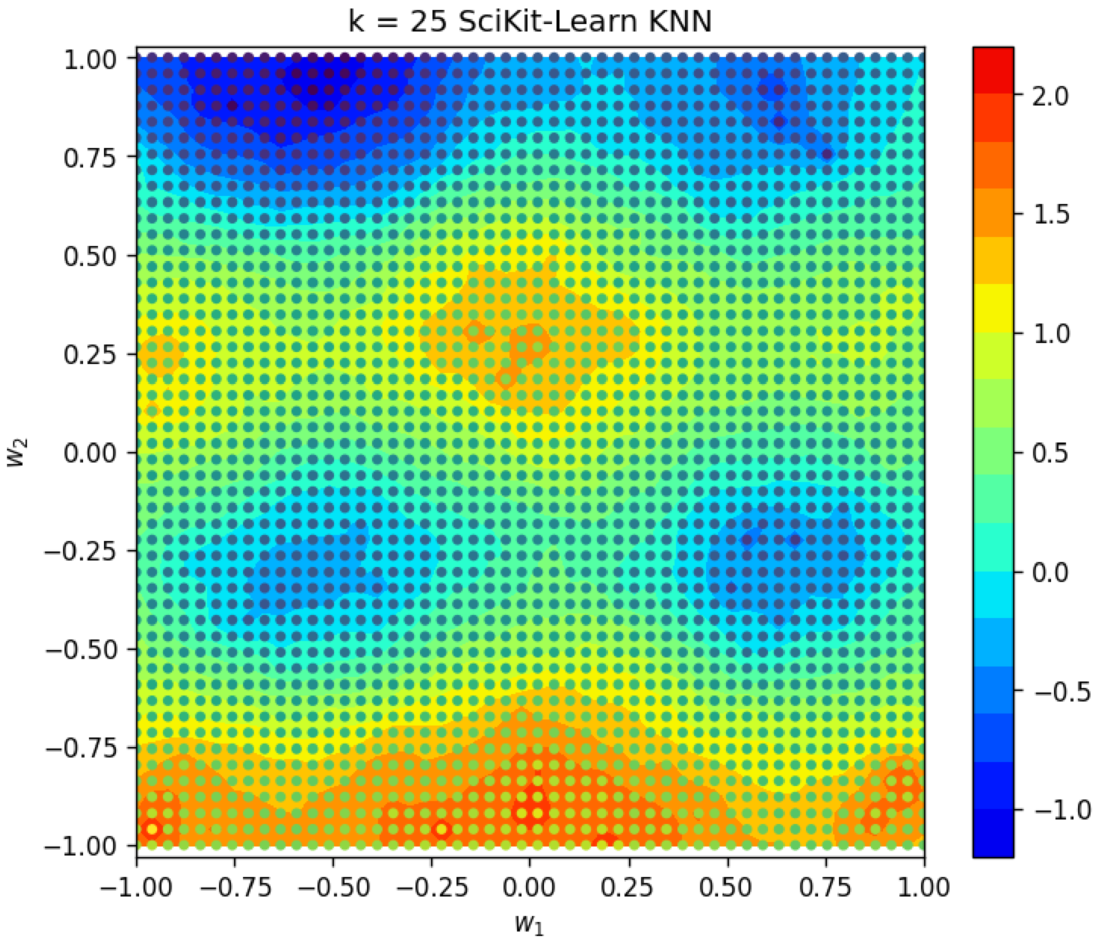
```
# YOUR CODE GOES HERE
# Visualize sklearn results for k = 1, 5, and 25
def plot_sklearn(k):
    w1 = np.linspace(-1, 1, 50)
    w2 = np.linspace(-1, 1, 50)
    w1s, w2s = np.meshgrid(w1, w2)
    w1_grid, w2_grid = w1s.flatten(), w2s.flatten()
    L_grid = np.zeros_like(w1_grid)
    for i in range(len(w1_grid)):
        L_grid[i] = weighted_knn(w1_grid[i], w2_grid[i], k)
    plt.figure(figsize=(7,5.8),dpi=120)
    contour = plt.contourf(w1s, w2s, L_grid.reshape(w1s.shape), 20, cmap="jet",␣
↪vmin=-1.5, vmax=2.5)
    plt.colorbar(contour, ticks = [-1.5, -1 , -0.5, 0, 0.5, 1, 1.5, 2, 2.5])
    plt.scatter(w1_grid, w2_grid, c=L_grid, s=10)
    plt.axis("equal")
    plt.xlabel("$w_1$")
    plt.ylabel("$w_2$")
    plt.xlim(-1,1)
    plt.ylim(-1,1)
    plt.title("k = " + str(k) + " SciKit-Learn KNN")
    plt.show()

# Visualize sklearn results for k = 1
plot_sklearn(1)
# Visualize sklearn results for k = 5
plot_sklearn(5)
# Visualize sklearn results for k = 25
plot_sklearn(25)
```



k = 1 SciKit-Learn KNN
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
w_2
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
w_1
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5



k = 5 SciKit-Learn KNN
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
w_2
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
w_1
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5



k = 25 SciKit-Learn KNN
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
w_2
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00
w_1
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0