1. *Casting an image into vector form.* We have a 15×15 image that we would like to represent as a vector in $\mathbb{R}^d$.

   (a) If the image is greyscale, we can use one coordinate of the vector for each pixel. In this case, what is d?

   (b) On the other hand, for a color image we would need three coordinates for each pixel (to represent R,G,B values). In this case, what is d?

a. $15 \times 15 = 225,\ d = 225$

b. $225 \times 3 = 675,\ d = 675$

2. *Euclidean distance.* What is the Euclidean distance between the following two points in $\mathbb{R}^3$?

$$\begin{bmatrix} 1 \\ -2 \\ 1 \end{bmatrix}, \quad \begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix}$$

$\sqrt{(3-1)^2 + (2+2)^2 + (1-1)^2} = \sqrt{20}$

3. *Accuracy of a random classifier.* A particular data set has 4 possible labels, with the following frequencies:

Label	Frequency
A	50%
B	20%
C	20%
D	10%

   (a) What is the error rate of a classifier that picks a label (A,B,C,D) at random, each with probability 1/4?

   (b) One very simple type of classifier just returns the same label, always.

   - What label should it return?
   - What will its error rate be?

a. $1 - \frac{1}{4}(50\% + 20\% + 20\% + 10\%) = \frac{3}{4}$

b.

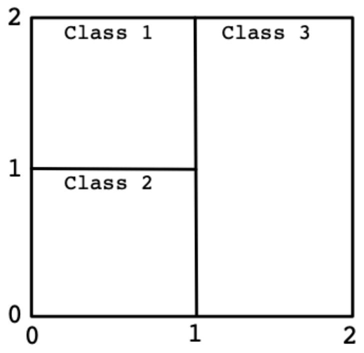
   i. Label "A" should be returned

   ii. $1 - 50\% = \frac{1}{2}$

4. *Decision boundary of the nearest neighbor classifier.* In this problem,

- The data space is $\mathcal{X} = [0,2]^2$: each point has two coordinates, and they lie between 0 and 2.
- The labels are $\mathcal{Y} = \{1,2,3\}$.
- The distance function is ℓ_2 (Euclidean distance).

The correct labels in different parts of $\mathcal{X}$ are as shown below.



(a) What is the label of point $\begin{bmatrix}0.5\\0.5\end{bmatrix}$?

Now suppose you have a training set consisting of just two points, located at

$$\begin{bmatrix}0.5\\0.5\end{bmatrix}, \begin{bmatrix}0.5\\1.5\end{bmatrix}.$$

(b) What label will the nearest neighbor classifier assign to point $\begin{bmatrix}1.5\\0.5\end{bmatrix}$?

(c) What label will the nearest neighbor classifier assign to point $\begin{bmatrix}2\\2\end{bmatrix}$?

(d) Which label will this classifier never predict?

(e) Now suppose that when the classifier is used, the test points are uniformly distributed over the square $\mathcal{X}$. What is the error rate of the 1-NN classifier?

a. label 2

b. $\sqrt{(1.5 - 0.5)^2 + (0.5 - 0.5)^2} = 1$

$\sqrt{(1.5 - 0.5)^2 + (0.5 - 1.5)^2} = \sqrt{2}$

$\sqrt{2} > 1$ and since point $\begin{bmatrix}0.5\\0.5\end{bmatrix}$ is label 2 and point $\begin{bmatrix}1.5\\0.5\end{bmatrix}$ is closer to point $\begin{bmatrix}0.5\\0.5\end{bmatrix}$, point $\begin{bmatrix}1.5\\0.5\end{bmatrix}$ will be label 2.

c. $\sqrt{(2 - 0.5)^2 + (2 - 0.5)^2} = \sqrt{4.5} = 2.12$

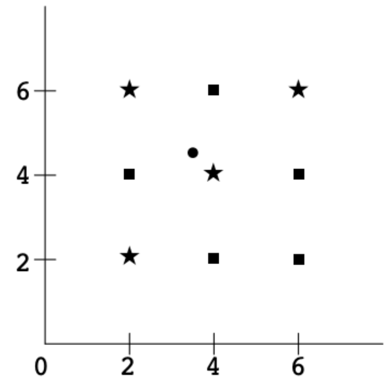
$\sqrt{(2 - 0.5)^2 + (2 - 1.5)^2} = 1.58$

$2.12 > 1.58$ and since point $\begin{bmatrix}0.5\\1.5\end{bmatrix}$ is label 1 and point $\begin{bmatrix}2\\2\end{bmatrix}$ is closer to point $\begin{bmatrix}0.5\\1.5\end{bmatrix}$, point $\begin{bmatrix}2\\2\end{bmatrix}$ will be label 1.

d. There's no training set in label 3 so the classifier will never predict label 3.

e. $1 - \frac{1}{2} = \frac{1}{2}$

5. In the picture below, there are nine training points, each with label either `square` or `star`. These will be used to guess the label of a query point at $\begin{bmatrix}3.5\\4.5\end{bmatrix}$, indicated by a circle.



Suppose Euclidean distance is used.

(a) How will the point be classified by 1-NN? The options are `square`, `star`, or `ambiguous`.

(b) By 3-NN?

(c) By 5-NN?

a. Will be classified as a star as it's the closest to the star.
b. Will be classified as a square because we're taking the top 3 closest points and 2 of them are squares.
c. $To\ (2, 2)\ star\text{:}\ \sqrt{(3.5 - 2)^2 + (4.5 - 2)^2} = 2.915$

   $To\ (4, 2)\ square\text{:}\ \sqrt{(3.5 - 4)^2 + (4.5 - 2)^2} = 2.549$

   $To\ (6, 6)\ star\text{:}\ \sqrt{(3.5 - 6)^2 + (4.5 - 6)^2} = 2.915$

   $To\ (6, 4)\ square\text{:}\ \sqrt{(3.5 - 6)^2 + (4.5 - 4)^2} = 2.549$

   Will be classified as a square because we're taking the top 5 closest points and 3 of them are squares, 2 of them are stars.

6. *Nearest neighbor on MNIST.* The Jupyter notebook `nn-mnist.ipynb` (in the archive file mentioned above) implements a basic 1-NN classifier for a subset of the MNIST data set. It uses a separate training and test set. Begin by going through this notebook, running each segment and taking care to understand exactly what each line is doing.

   Now do the following.

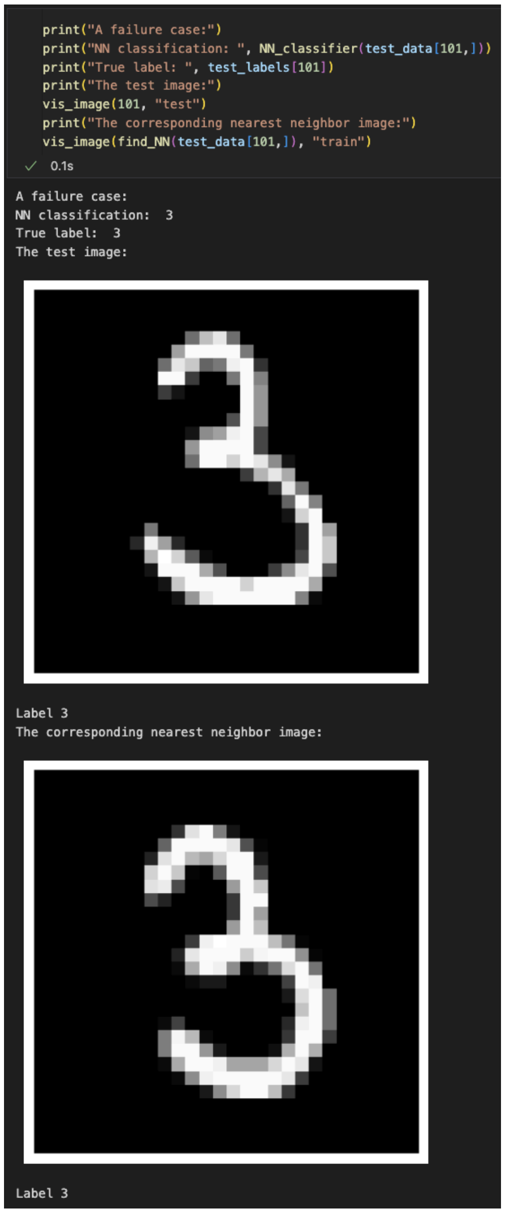
   (a) For test point #100 (which is actually the 101st point in the test set due to Python's zero-indexing), print its image as well as the image of its nearest neighbor in the training set. Put these images in your writeup. Is this test point classified correctly?

   (b) The *confusion matrix* for the classifier is a 10×10 matrix N_{ij} with $0 \leq i, j \leq 9$, where N_{ij} is the number of test points whose true label is i but which are classified as j. Thus, if all test points are correctly classified, the off-diagonal entries of the matrix will be zero.

   - Compute the matrix N for the 1-NN classifier and print it out.
   - Which digit is misclassified most often? Least often?

   (c) For each digit $0 \leq i \leq 9$: look at all training instances of image i, and compute their mean. This average is a 784-dimensional vector. Use the `show_digit` routine to print out these 10 average-digits.

a.



```
print("A failure case:")
print("NN classification: ", NN_classifier(test_data[101,]))
print("True label: ", test_labels[101])
print("The test image:")
vis_image(101, "test")
print("The corresponding nearest neighbor image:")
vis_image(find_NN(test_data[101,]), "train")
```

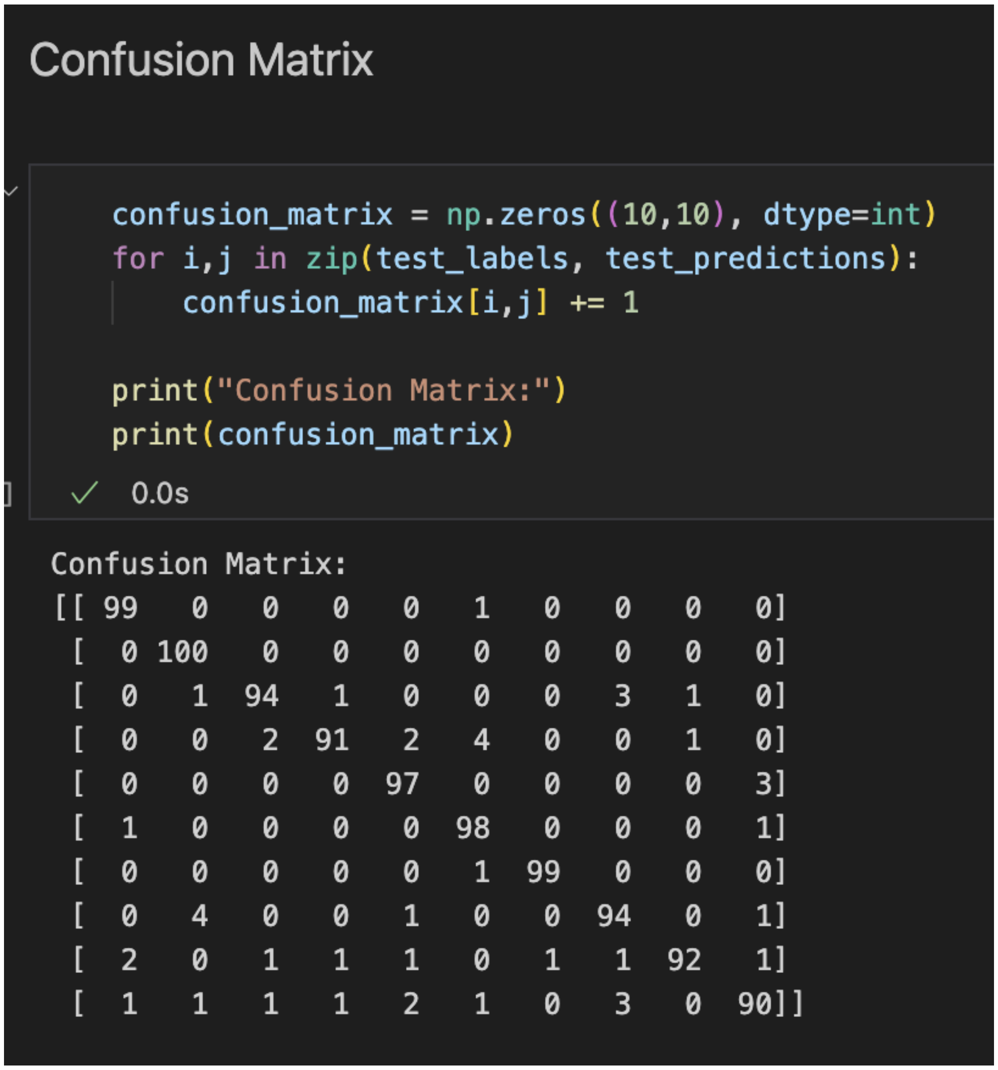
✓ 0.1s

```
A failure case:
NN classification:  3
True label:  3
The test image:
```

```
Label 3
The corresponding nearest neighbor image:
```

```
Label 3
```

b.



Confusion Matrix

```
confusion_matrix = np.zeros((10,10), dtype=int)
for i,j in zip(test_labels, test_predictions):
    confusion_matrix[i,j] += 1

print("Confusion Matrix:")
print(confusion_matrix)
```

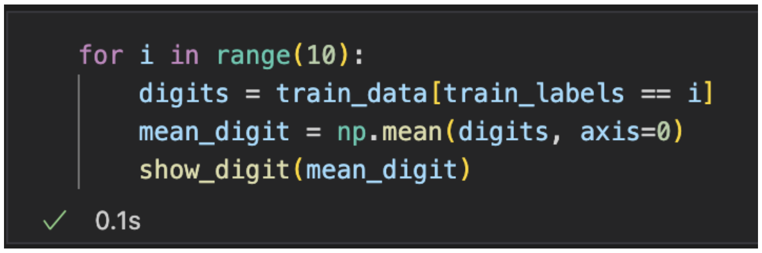
✓ 0.0s

```
Confusion Matrix:
[[ 99   0   0   0   0   1   0   0   0   0]
 [  0 100   0   0   0   0   0   0   0   0]
 [  0   1  94   1   0   0   0   3   1   0]
 [  0   0   2  91   2   4   0   0   1   0]
 [  0   0   0   0  97   0   0   0   0   3]
 [  1   0   0   0   0  98   0   0   0   1]
 [  0   0   0   0   0   1  99   0   0   0]
 [  0   4   0   0   1   0   0  94   0   1]
 [  2   0   1   1   1   0   1   1  92   1]
 [  1   1   1   1   2   1   0   3   0  90]]
```

i.

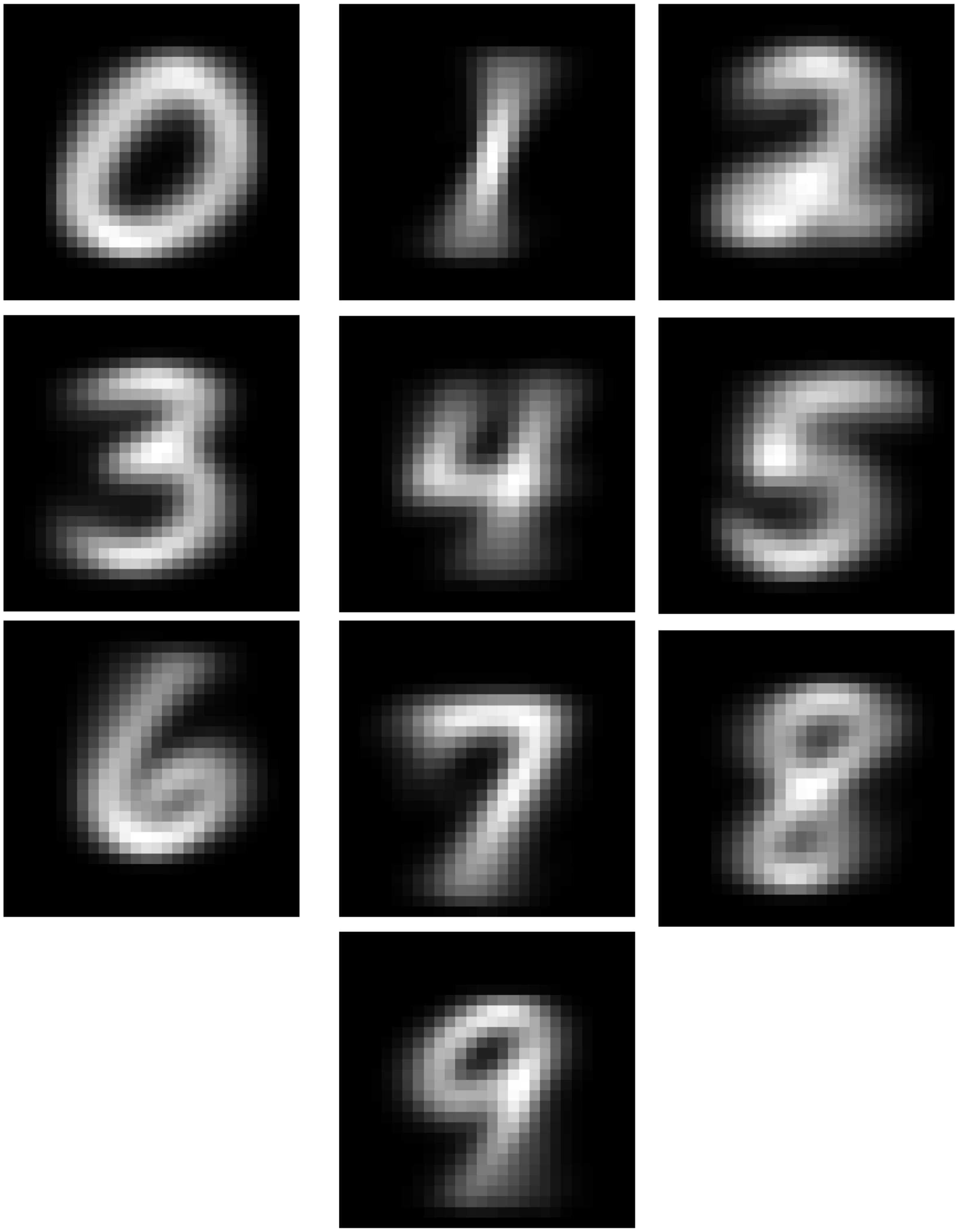
ii. Row represents the true labels and column represents the predict labels
Any numbers that appear outside of the diagonal entries of the matrix are misclassified.
Row 0: 1
Row 1: 0
Row 2: 1 + 1 +3 + 1 = 6
Row 3: 9
Row 4: 3
Row 5: 2
Row 6: 1
Row 7: 6
Row 8: 8
Row 9: 10
Therefore misclassified most often is digit 9 and least often is digit 1.

c.



```
for i in range(10):
    digits = train_data[train_labels == i]
    mean_digit = np.mean(digits, axis=0)
    show_digit(mean_digit)
```

```
✓ 0.1s
```



7. *Classifying back injuries.* In this problem, you will use nearest neighbor to classify patients' back injuries based on measurements of the shape and orientation of their pelvis and spine.

The data set contains information from 310 patients. For each patient, there are: six numeric features (the x) and a label (the y): 'NO' (normal), 'DH' (herniated disk), or 'SL' (spondilolysthesis). We will divide this data into a training set with 250 points and a separate test set of 60 points.

- Make sure you have the data set `spine-data.txt`. You can load it into Python using the following.

```
import numpy as np
# Load data set and code labels as 0 = 'NO', 1 = 'DH', 2 = 'SL'
labels = [b'NO', b'DH', b'SL']
data = np.loadtxt('spine-data.txt', converters={6:  lambda s:  labels.index(s)})
```

  This converts the labels in the last column into 0 (for 'NO'), 1 (for 'DH'), and 2 (for 'SL').
- Split the data into a training set, consisting of the *first* 250 points, and a test set, consisting of the remaining 60 points.
- Code up a nearest neighbor classifier based on this training set. Try both ℓ_2 and ℓ_1 distance. Recall that for $x, x' \in \mathbb{R}^d$:

$$\|x - x'\|_2 = \sqrt{\sum_{i=1}^{d}(x_i - x_i')^2}$$

$$\|x - x'\|_1 = \sum_{i=1}^{d}|x_i - x_i'|$$

Now do the following exercises, to be turned in.

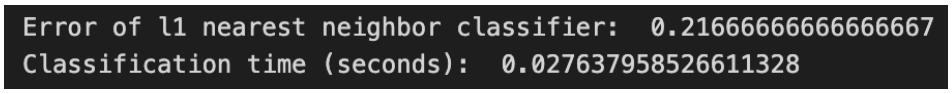
(a) What error rates do you get on the test set for each of the two distance functions?

(b) For each of the two distance functions, give the *confusion matrix* of the NN classifier. This is a 3×3 table of the form:

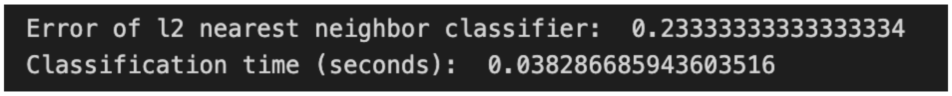
	NO	DH	SL
NO			
DH			
SL			

The entry at row DH, column SL, for instance, contains the number of test points whose correct label was DH but which were classified as SL.

a.

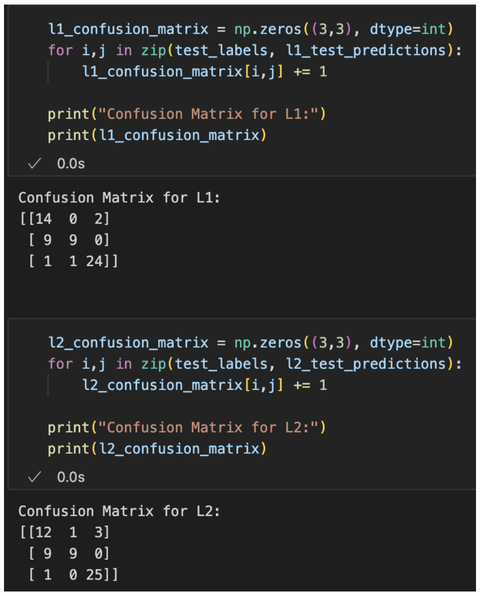


```
Error of l1 nearest neighbor classifier:  0.21666666666666667
Classification time (seconds):  0.027637958526611328
```



```
Error of l2 nearest neighbor classifier:  0.23333333333333334
Classification time (seconds):  0.038286685943603516
```

b.



```
l1_confusion_matrix = np.zeros((3,3), dtype=int)
for i,j in zip(test_labels, l1_test_predictions):
    l1_confusion_matrix[i,j] += 1

print("Confusion Matrix for L1:")
print(l1_confusion_matrix)
```

✓ 0.0s

```
Confusion Matrix for L1:
[[14  0  2]
 [ 9  9  0]
 [ 1  1 24]]
```

```
l2_confusion_matrix = np.zeros((3,3), dtype=int)
for i,j in zip(test_labels, l2_test_predictions):
    l2_confusion_matrix[i,j] += 1

print("Confusion Matrix for L2:")
print(l2_confusion_matrix)
```

✓ 0.0s

```
Confusion Matrix for L2:
[[12  1  3]
 [ 9  9  0]
 [ 1  0 25]]
```