

Solution 1

Solution 1 (a)**Step 1: Identify characteristics of the dataspace**

We are given 10 dimensional vectors where each element can be any real number ($x_i \in \mathbb{R}$):

\therefore we can express the dataspace χ as: $\chi = \mathbb{R}^{10}$

Solution 1 (b)**Step 1: Identify characteristics of the dataspace**

We are given 3 dimensional vectors where each element is zero or one ($x_i \in [0, 1]$):

\therefore we can express the dataspace χ as: $\chi = [0, 1]^3$

Solution 2

Solution 2 (a)

Step 1: Define Euclidean distance (ℓ_2)

$$\ell_2 = \|p - q\|_2 = \sqrt{\sum_{i=1}^n (p_i - q_i)^2}$$

Step 2: Compute ℓ_2

Let $p = 1$ and $q = 10$

$$\begin{aligned}\ell_2 &= \sqrt{\sum_{i=1}^n (p_i - q_i)^2} \\ \ell_2 &= \sqrt{\sum_{i=1}^1 (1 - 10)^2} \\ \ell_2 &= \sqrt{(-9)^2} \\ \ell_2 &= 9\end{aligned}$$

$\therefore \ell_2 = 9$

Solution 2 (b)

Step 1: Define Euclidean distance (ℓ_2)

$$\ell_2 = \|p - q\|_2 = \sqrt{\sum_{i=1}^n (p_i - q_i)^2}$$

Step 2: Compute ℓ_2

Let $p = \begin{bmatrix} -1 \\ 12 \end{bmatrix}$, $q = \begin{bmatrix} 6 \\ -12 \end{bmatrix}$

$$\begin{aligned}\ell_2 &= \sqrt{\sum_{i=1}^n (p_i - q_i)^2} \\ \ell_2 &= \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2} \\ \ell_2 &= \sqrt{(-1 - 6)^2 + (12 - (-12))^2} \\ \ell_2 &= \sqrt{(-7)^2 + (24)^2} \\ \ell_2 &= \sqrt{625} \\ \ell_2 &= 25\end{aligned}$$

$\therefore \ell_2 = 25$

Solution 2

Solution 2 (c)**Step 1: Define Euclidean distance (ℓ_2)**

$$\ell_2 = \|p - q\|_2 = \sqrt{\sum_{i=1}^n (p_i - q_i)^2}$$

Step 2: Compute ℓ_2

$$\text{Let } p = \begin{bmatrix} 1 \\ 5 \\ -1 \end{bmatrix}, q = \begin{bmatrix} 5 \\ 2 \\ 11 \end{bmatrix}$$

$$\begin{aligned}\ell_2 &= \sqrt{\sum_{i=1}^n (p_i - q_i)^2} \\ \ell_2 &= \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + (p_3 - q_3)^2} \\ \ell_2 &= \sqrt{(1 - 5)^2 + (5 - 2)^2 + (-1 - 11)^2} \\ \ell_2 &= \sqrt{(-4)^2 + (3)^2 + (-12)^2} \\ \ell_2 &= \sqrt{169} \\ \ell_2 &= 13\end{aligned}$$

$$\therefore \ell_2 = 13$$

Solution 3

Solution 3 (a)**Step 1: Normalize the vector x**

$$\text{Let } x = \begin{bmatrix} 10 \\ 15 \\ 25 \end{bmatrix}$$

$$\sum_{i=1}^3 x_i = x_1 + x_2 + x_3 = 10 + 15 + 25 = 50$$

Now, divide each entry by the total sum:

$$p = \frac{1}{50} \cdot x = \frac{1}{50} \begin{bmatrix} 10 \\ 15 \\ 25 \end{bmatrix} = \begin{bmatrix} 10/50 \\ 15/50 \\ 25/50 \end{bmatrix} = \begin{bmatrix} 0.2 \\ 0.3 \\ 0.5 \end{bmatrix}$$

 \therefore the result (p) of scaling vector x is the following:

$$p = \begin{bmatrix} 0.2 \\ 0.3 \\ 0.5 \end{bmatrix}$$

Solution 3 (b)**Step 1: Define dimension of the probability simplex**The dimension of vector p is 3 and $k = n - 1$ where k is the dimension of the probability simplex \therefore vector p lies in the probability simplex(Δ_2) for $k = 2$

Solution 4

Step 1: Define probability simplex Δ_2

For a point to be scalable to Δ_2 , after scaling it must satisfy:

- All components must be non-negative
- The sum of components must equal 1

Step 2: Give example that violates one of the rules in Step 1

Let $x = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$

The second component of x violates the first rule, all components for a point must be non-negative Δ_2 .

\therefore the point $x = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$ cannot be scaled to lie in Δ_2

Solution 5

Visualizing the Simplex Δ_3 in 2D Projections

Here are the three 2D views of the probability simplex Δ_3 . Each plot is a *shadow* of the 3D triangle, viewed along one of the principal axes.

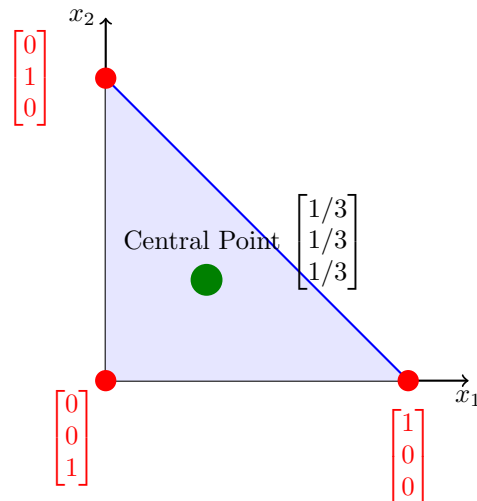


Figure 1: View 1: Projection onto the x_1 - x_2 plane.

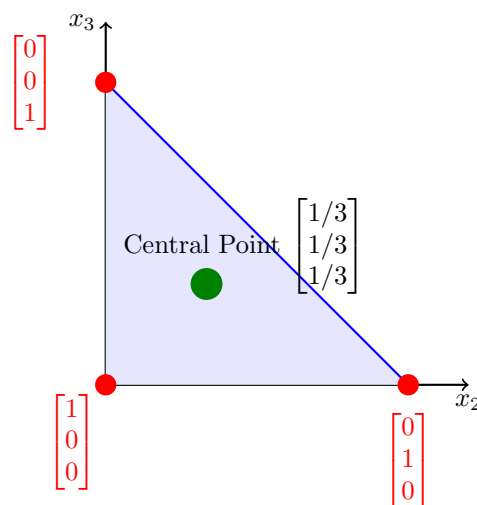


Figure 2: View 2: Projection onto the x_2 - x_3 plane.

Solution 5

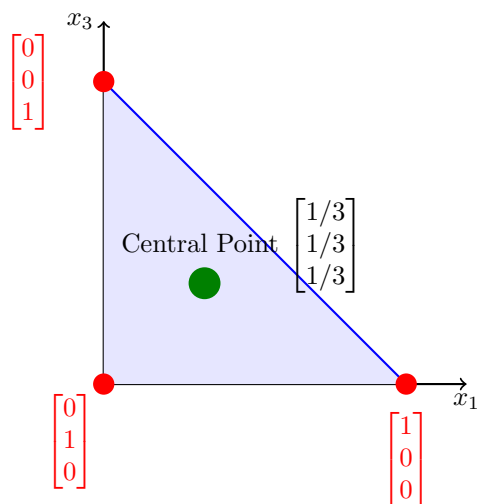


Figure 3: View 3: Projection onto the x_1 - x_3 plane.

Solution 6

6 (a): ℓ_1 for p and q

The ℓ_1 distance between two vectors $p, q \in \mathbb{R}^n$ is given by:

$$\|p - q\|_1 = \sum_{i=1}^n |p_i - q_i|$$

$$\text{Let } p = \begin{bmatrix} 1/2 \\ 1/4 \\ 1/8 \\ 1/8 \end{bmatrix} \text{ and } q = \begin{bmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{bmatrix}$$

$$\begin{aligned} \|p - q\|_1 &= \left| \frac{1}{2} - \frac{1}{4} \right| + \left| \frac{1}{4} - \frac{1}{4} \right| + \left| \frac{1}{8} - \frac{1}{4} \right| + \left| \frac{1}{8} - \frac{1}{4} \right| \\ &= \left| \frac{2}{4} - \frac{1}{4} \right| + |0| + \left| \frac{1}{8} - \frac{2}{8} \right| + \left| \frac{1}{8} - \frac{2}{8} \right| \\ &= \frac{1}{4} + 0 + \left| -\frac{1}{8} \right| + \left| -\frac{1}{8} \right| \\ &= \frac{1}{4} + \frac{1}{8} + \frac{1}{8} = \frac{2}{8} + \frac{1}{8} + \frac{1}{8} = \frac{4}{8} = \frac{1}{2} \end{aligned}$$

$$\therefore \ell_1 = \frac{1}{2}$$

Solution 6

6 (b): ℓ_1 for q and r

The ℓ_1 distance between two vectors $q, r \in \mathbb{R}^n$ is given by:

$$\|q - r\|_1 = \sum_{i=1}^n |q_i - r_i|$$

$$\text{Let } q = \begin{bmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{bmatrix} \text{ and } r = \begin{bmatrix} 1/2 \\ 0 \\ 1/4 \\ 1/4 \end{bmatrix}$$

$$\begin{aligned} \|q - r\|_1 &= \sum_{i=1}^4 |q_i - r_i| \\ &= \left| \frac{1}{4} - \frac{1}{2} \right| + \left| \frac{1}{4} - 0 \right| + \left| \frac{1}{4} - \frac{1}{4} \right| + \left| \frac{1}{4} - \frac{1}{4} \right| \\ &= \left| \frac{1}{4} - \frac{2}{4} \right| + \left| \frac{1}{4} \right| + |0| + |0| \\ &= \left| -\frac{1}{4} \right| + \frac{1}{4} + 0 + 0 \\ &= \frac{1}{4} + \frac{1}{4} \\ &= \frac{1}{2} \end{aligned}$$

$$\therefore \ell_1 = \frac{1}{2}$$

Solution 6

6 (c): KL divergence $K(p, q)$

The Kullback-Leibler (KL) divergence from a distribution p to a distribution q is defined as:

$$K(p, q) = \sum_i p_i \ln \frac{p_i}{q_i}$$

$$\text{Let } p = \begin{bmatrix} 1/2 \\ 1/4 \\ 1/8 \\ 1/8 \end{bmatrix} \text{ and } q = \begin{bmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{bmatrix}$$

$$\begin{aligned} K(p, q) &= \sum_{i=1}^4 p_i \ln \left(\frac{p_i}{q_i} \right) \\ &= p_1 \ln \left(\frac{p_1}{q_1} \right) + p_2 \ln \left(\frac{p_2}{q_2} \right) + p_3 \ln \left(\frac{p_3}{q_3} \right) + p_4 \ln \left(\frac{p_4}{q_4} \right) \\ &= \frac{1}{2} \ln \left(\frac{1/2}{1/4} \right) + \frac{1}{4} \ln \left(\frac{1/4}{1/4} \right) + \frac{1}{8} \ln \left(\frac{1/8}{1/4} \right) + \frac{1}{8} \ln \left(\frac{1/8}{1/4} \right) \\ &= \frac{1}{2} \ln(2) + \frac{1}{4} \ln(1) + \frac{1}{8} \ln \left(\frac{1}{2} \right) + \frac{1}{8} \ln \left(\frac{1}{2} \right) \\ &= \frac{1}{2} \ln(2) + \frac{1}{4}(0) - \frac{1}{8} \ln(2) - \frac{1}{8} \ln(2) \\ &= \frac{1}{2} \ln(2) - \frac{2}{8} \ln(2) \\ &= \left(\frac{1}{2} - \frac{1}{4} \right) \ln(2) \\ &= \frac{1}{4} \ln(2) \end{aligned}$$

$$\therefore K(p, q) = \frac{1}{4} \ln(2)$$

Solution 6

6 (d): KL divergence $K(q, r)$

The Kullback-Leibler (KL) divergence from a distribution q to a distribution r is defined as:

$$K(q, r) = \sum_i q_i \ln \frac{q_i}{r_i}$$

$$\text{Let } q = \begin{bmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{bmatrix} \text{ and } r = \begin{bmatrix} 1/2 \\ 0 \\ 1/4 \\ 1/4 \end{bmatrix}$$

Looking at the second component ($i = 2$). Here, $q_2 = \frac{1}{4} > 0$ while $r_2 = 0$. The corresponding term in the KL divergence sum, $q_2 \ln \left(\frac{q_2}{r_2} \right)$, involves division by zero.

Hence, the divergence will be infinite.

$$\therefore K(q, r) = \infty$$

Solution 7

Python Code

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 from extract_feature import compute_or_load_features
4 from sklearn.neighbors import KNeighborsClassifier
5
6 def run_nearest_neighbor(x_train, y_train, x_test, y_test):
7     # create classifier
8     nn_classifier = KNeighborsClassifier(n_neighbors=1, algorithm='auto')
9
10    # train
11    nn_classifier.fit(x_train, y_train)
12
13    # test and report accuracy
14    test_acc = nn_classifier.score(x_test, y_test)
15
16    print("Nearest neighbor accuracy on the test set: %f"%test_acc)
17
18    return nn_classifier
19
20
21 def analyze_nn(classifier, x_train, y_train, x_test, y_test, x_test_features, N,
22               model_type):
23     """
24     generalization to grab indices, predictions, and make plots
25     """
26     # list of image labels
27     CIFAR_CLASSES = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog',
28                     'horse', 'ship', 'truck']
29
30     # get predictions of classifier on test data
31     y_pred = classifier.predict(x_test_features)
32
33     # define bool condition where classifier made correct prediction
34     is_correct = (y_pred == y_test)
35
36     # get indices for correct and incorrect samples
37     correct_indices_test = np.where(is_correct)[0][:N]
38     incorrect_indices_test = np.where(~is_correct)[0][:N]
39
40     # make image plots
41     def plot_pairs(title, test_indices):
42         # check just in case of bad result.. nema nista
43         if len(test_indices) == 0:
44             return
45
46         # Get the feature vectors for the selected test images (flattened)
47         selected_test_features = x_test_features[test_indices]
48
49         # get nearest neighbor
50         # note: kneighbors -> (distances, indices).. only need the indices.
51         _, nn_train_indices_2D = classifier.kneighbors(X=selected_test_features,
52                                                       n_neighbors=1)
53
54         # 2D -> 1D
55         nn_train_indices = nn_train_indices_2D.flatten()
56
57         # get images and labels for the selected test points
58         test_images_sample = x_test[test_indices]
59         test_labels_sample = y_test[test_indices]
60
61         # get the RAW neighbor image and label from the training points
62         nn_images_sample = x_train[nn_train_indices]
63         nn_labels_sample = y_train[nn_train_indices]

```

```

63     # prediction is nearest neighbor label
64     y_pred_sample = nn_labels_sample
65
66     # reshape (N, C, H, W) to (N, H, W, C) for plot
67     test_images_plt = test_images_sample.transpose(0, 2, 3, 1)
68     nn_images_plt = nn_images_sample.transpose(0, 2, 3, 1)
69
70     N_plot = len(test_indices)
71
72     # just in case only 1 image is plotted (axes will be 1D instead of 2D)
73     if N_plot == 1:
74         fig, axes = plt.subplots(N_plot, 2, figsize=(6, 2))
75         axes = axes[np.newaxis, :]
76     else:
77         fig, axes = plt.subplots(N_plot, 2, figsize=(6, 2 * N_plot))
78
79     fig.suptitle(title, fontsize=14, y=1.02)
80
81     for i in range(N_plot):
82         is_correct = (test_labels_sample[i] == y_pred_sample[i])
83         pred_color = 'green' if is_correct else 'red'
84
85         # plot test image
86         axes[i, 0].imshow(test_images_plt[i] / 255.0)
87         axes[i, 0].set_title(f"Test ({CIFAR_CLASSES[test_labels_sample[i]]})",
88                             fontsize=10)
89         axes[i, 0].axis('off')
90
91         # plot nearest neighbor
92         axes[i, 1].imshow(nn_images_plt[i] / 255.0)
93         axes[i, 1].set_title(
94             f"NN ({CIFAR_CLASSES[nn_labels_sample[i]]})",
95             fontsize=10,
96             color=pred_color
97         )
98         axes[i, 1].axis('off')
99
100     plt.tight_layout(rect=[0, 0.03, 1, 0.98])
101     plt.show()
102     fig.savefig(f"{title.split(' ')[2].lower()}_{model_type}.png")
103
104     # plot correct and incorrect cases
105     plot_pairs(f"First {N} Correct Predictions ({model_type})", correct_indices_test)
106     plot_pairs(f"First {N} Incorrect Predictions ({model_type})",
107               incorrect_indices_test)
108
109 # raw pixel
110 raw_pixel_train_features, raw_pixel_test_features = compute_or_load_features(x_train,
111                                     x_test, "raw_pixel")
112 raw_pixel_knn_classifier = run_nearest_neighbor(raw_pixel_train_features, y_train,
113         raw_pixel_test_features, y_test)
114 analyze_nn(
115     classifier=raw_pixel_knn_classifier,
116     x_train=x_train,
117     y_train=y_train,
118     x_test=x_test,
119     y_test=y_test,
120     x_test_features=raw_pixel_test_features,
121     N=5,
122     model_type="raw_pixel"
123 )
124
125 # HoG
126 hog_train_features, hog_test_features = compute_or_load_features(x_train, x_test, "hog")
127 hog_knn_classifier = run_nearest_neighbor(hog_train_features, y_train,
128         hog_test_features, y_test)
129 analyze_nn(
130     classifier=hog_knn_classifier,
131     x_train=x_train,
132     y_train=y_train,
133     x_test=x_test,
134     y_test=y_test,
135     x_test_features=hog_test_features,

```

```

131     N=5,
132     model_type="hog"
133 )
134
135 # vgg-last-fc
136 pretrained_cnn_last_fc_train_features, pretrained_cnn_last_fc_test_features =
137     compute_or_load_features(x_train, x_test, "pretrained_cnn", "last_fc")
138 pretrained_cnn_last_fc_knn_classifier =
139     run_nearest_neighbor(pretrained_cnn_last_fc_train_features, y_train,
140         pretrained_cnn_last_fc_test_features, y_test)
141 analyze_nearest_neighbors_simple(
142     classifier=pretrained_cnn_last_fc_knn_classifier,
143     x_train=x_train,
144     y_train=y_train,
145     x_test=x_test,
146     y_test=y_test,
147     x_test_features=pretrained_cnn_last_fc_test_features,
148     N=5,
149     model_type='vgg-last-fc'
150 )

```

Part a: Dimensionality for each of the representations (raw pixel, HoG, VGG-last-fc, VGG-last-conv)

Feature Type	Dimensionality
Raw Pixel	3072
HoG	512
VGG-last-fc	4096
VGG-last-conv	512

Part b: Test accuracies for 1-nearest neighbor classification using the various representations (raw pixel, HoG, VGG-last-fc, VGG-last-conv, random-VGG-last-fc, random-VGG-last-conv).

Feature Type	1-NN test accuracy (%)
Raw Pixel	35.4
HoG	36.6
VGG-last-fc	92.1
VGG-last-conv	92.0
random VGG-last-fc	39.1
random VGG-last-conv	40.6

Solution 7

Part c: Raw Pixel correct/incorrect

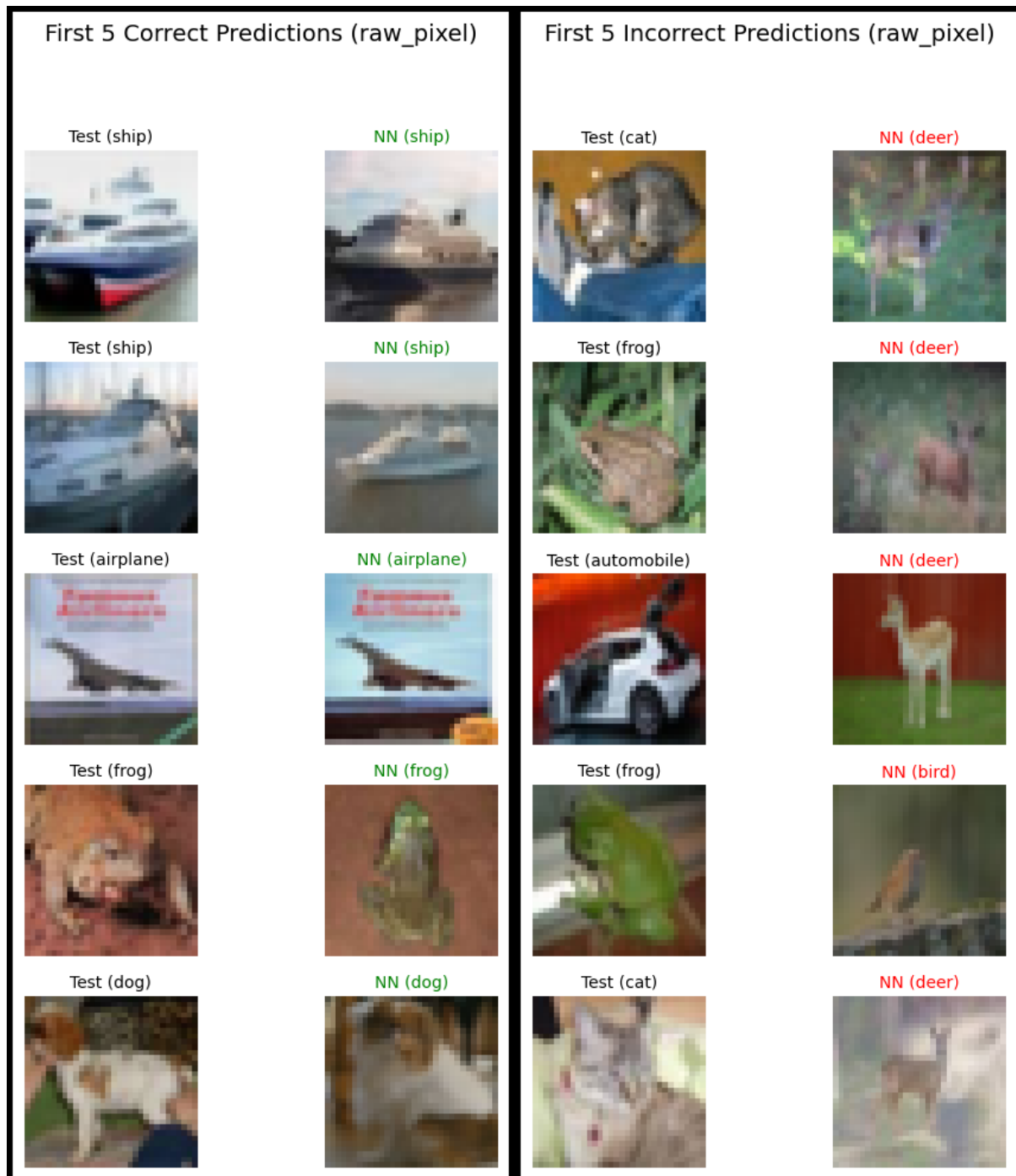


Figure 4: First five correct/incorrect images for Raw Pixel

Solution 7

Part c: HoG correct/incorrect

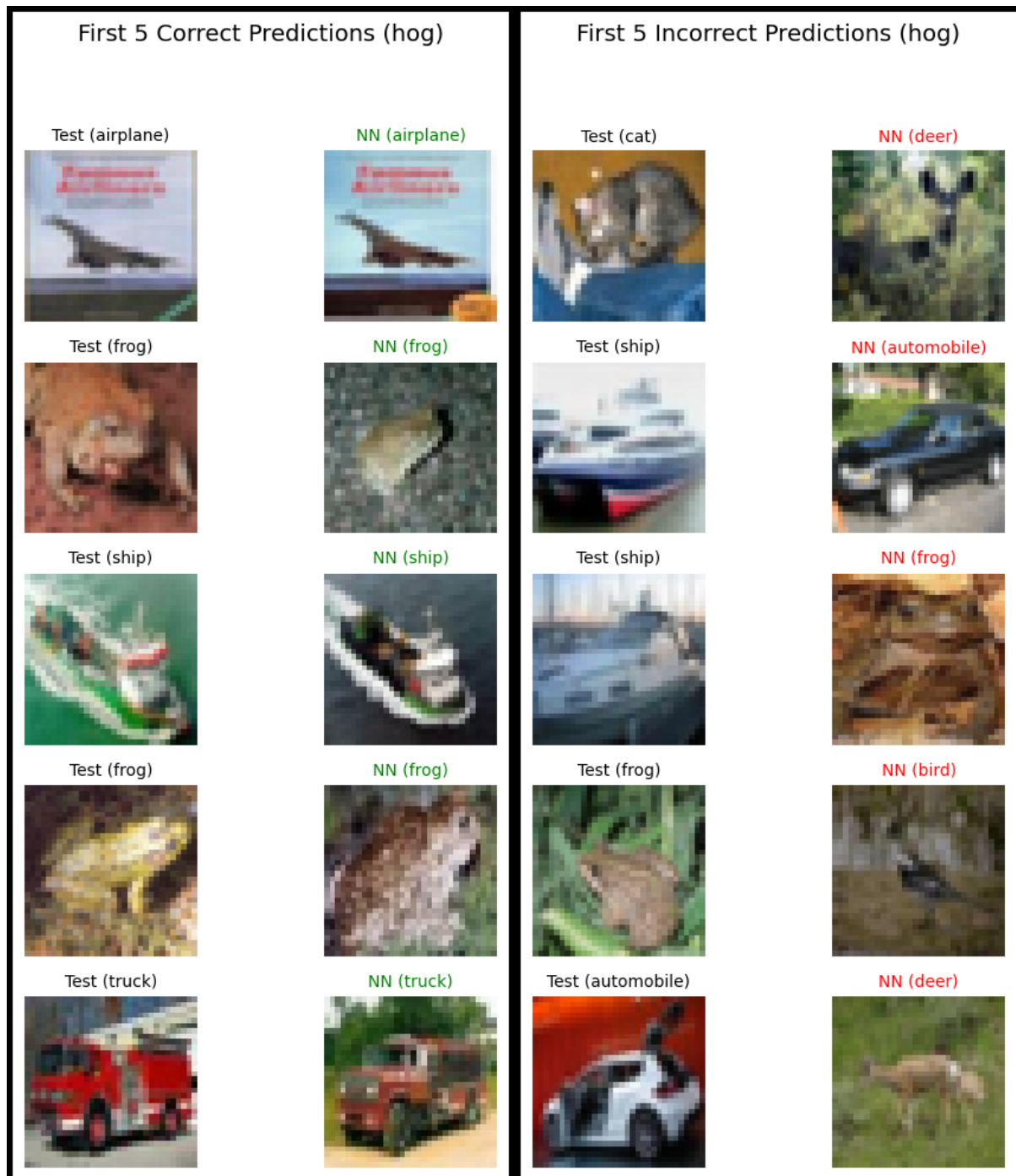


Figure 5: First five correct/incorrect images for HoG

Solution 7

Part c: VGG-last-fc correct/incorrect

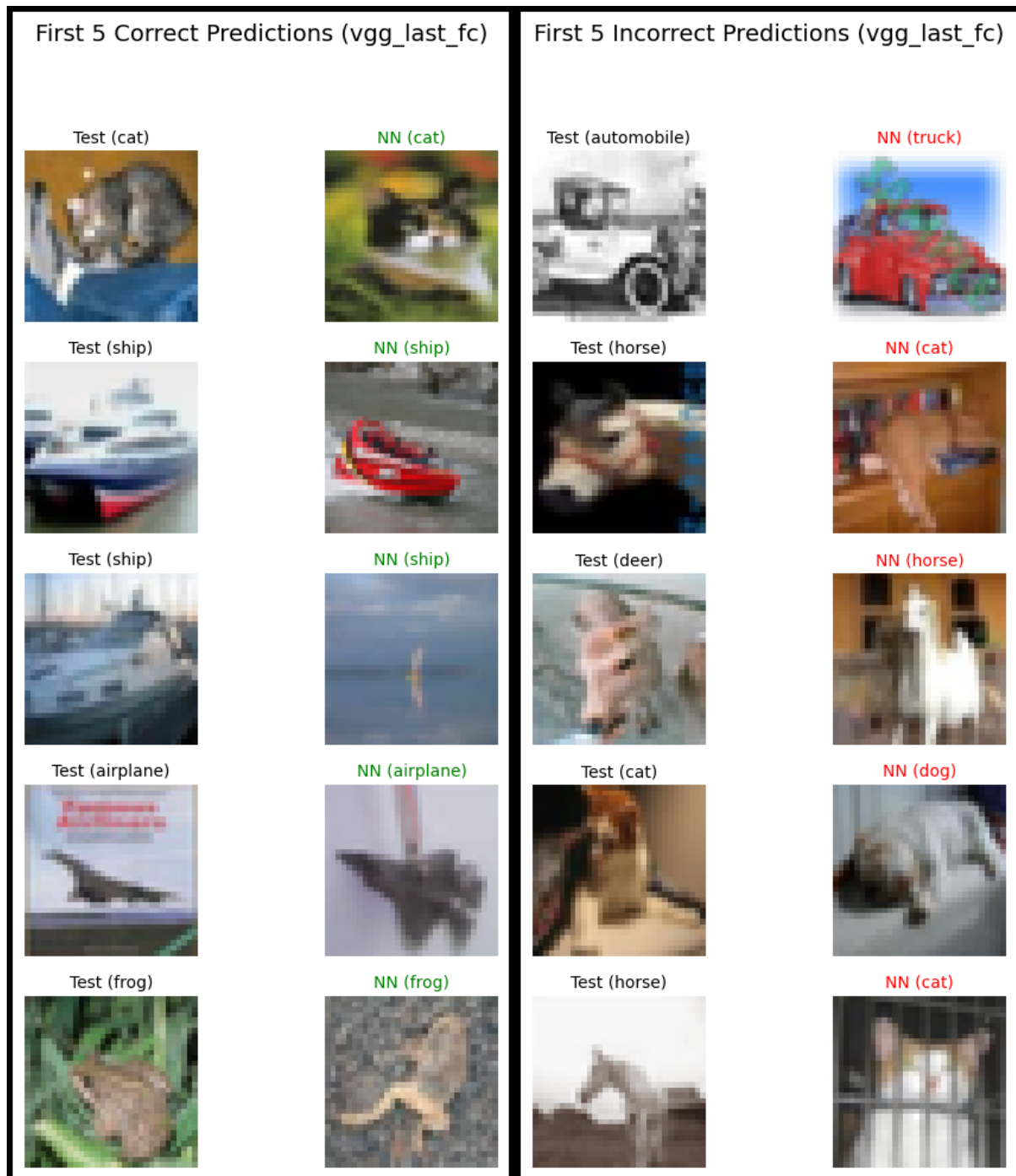


Figure 6: First five correct/incorrect images for VGG-last-fc

Solution 8

Python Code

```
1 import numpy as np
2 from sklearn.neighbors import NearestNeighbors
3
4 filename = 'glove.6B.300d.txt'
5 with open(filename) as f:
6     content = f.read().splitlines()
7
8 # initialize vecs and words 'containers'
9 n = len(content)
10 vecs = np.zeros((n, 300))
11 words = [" " for i in range(n)]
12 for index, rawline in enumerate(content):
13     line = rawline.split()
14     words[index] = line[0]
15     # need numpy speed
16     vecs[index] = np.array(line[1:], dtype=np.float32)
17
18 # make dict for access to word and index
19 word_to_index = {word: i for i, word in enumerate(words)}
20
21 # initialize target words
22 target_words = ['communism', 'africa', 'happy', 'sad', 'upset', 'computer', 'cat',
23                 'dollar']
24
25 # find 5 nearest neighbors (n).. remember k = n+1
26 n_neighbors = 6
27
28 # initialize and fit the nn model
29 nn_model = NearestNeighbors(n_neighbors=n_neighbors, metric='euclidean',
30                             algorithm='auto')
31 nn_model.fit(vecs)
32
33 # gracefully check for typo
34 try:
35     target_indices = [word_to_index[word] for word in target_words]
36 except KeyError as e:
37     print(f" error: word '{e.args[0]}' was not found.")
38     exit()
39
40 # extract the corresponding vectors for the target words from vecs
41 target_vectors = vecs[target_indices]
42
43 # find the nearest neighbors
44 distances, indices = nn_model.kneighbors(target_vectors)
45
46 # format and print results
47 results = {}
48 for i, word in enumerate(target_words):
49     neighbor_indices = indices[i][1:]
50     neighbor_words = [words[idx] for idx in neighbor_indices]
51     results[word] = neighbor_words
52
53 print(f"5 nearest neighbors in {filename} for {target_words}")
54 print(results)
```

Word Vectors: 5 closest words

Target Word	Five Closest Words
communism	['fascism', 'capitalism', 'nazism', 'stalinism', 'socialism']
africa	['african', 'continent', 'south', 'africans', 'zimbabwe']
happy	['glad', 'pleased', 'always', 'everyone', 'sure']
sad	['sorry', 'tragic', 'happy', 'pathetic', 'awful']
upset	['upsetting', 'surprised', 'upsets', 'stunned', 'shocked']
computer	['computers', 'software', 'technology', 'laptop', 'computing']
cat	['cats', 'dog', 'pet', 'feline', 'dogs']
dollar	['currency', 'dollars', 'euro', 'multibillion', 'weaker']

Solution 1: Scalability on the Probability Simplex

Step 1: Define the Probability Simplex Δ_2

The probability simplex Δ_k is the set of all k -dimensional vectors with non-negative components that sum to 1. For $k = 2$, a vector $p = \begin{bmatrix} p_1 \\ p_2 \end{bmatrix}$ is in Δ_2 if and only if it satisfies two conditions:

1. **Non-negativity:** $p_1 \geq 0$ and $p_2 \geq 0$.
2. **Sum-to-one:** $p_1 + p_2 = 1$.

The question asks if for any vector $p \in \mathbb{R}^2$ and scalar $c > 0$, the condition $c \cdot p \in \Delta_2$ implies that $p \in \Delta_2$.

Step 2: Analyze the Constraints under Scaling

Let $q = c \cdot p = \begin{bmatrix} cp_1 \\ cp_2 \end{bmatrix}$. We are given that $q \in \Delta_2$.

- **Non-negativity:** Since $c > 0$ and we are given $cp_1 \geq 0$ and $cp_2 \geq 0$, it must be that $p_1 \geq 0$ and $p_2 \geq 0$. This condition is satisfied for p .
- **Sum-to-one:** We are given that the components of q sum to 1: $cp_1 + cp_2 = 1$. Factoring out c , we get $c(p_1 + p_2) = 1$, which implies $p_1 + p_2 = \frac{1}{c}$.

For p to be in Δ_2 , its components must sum to 1, i.e., $p_1 + p_2 = 1$. This only holds if $\frac{1}{c} = 1$, which means $c = 1$. Since the statement must hold for any $c > 0$, we can find a counterexample by choosing $c \neq 1$.

Step 3: Construct a Counterexample

Let $c = 2$. Choose a point $q \in \Delta_2$, for example, $q = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}$. If $c \cdot p = q$, then $p = \frac{1}{c}q = \frac{1}{2} \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} = \begin{bmatrix} 0.25 \\ 0.25 \end{bmatrix}$.

Let's check if this p is in Δ_2 :

- **Non-negativity:** $p_1 = 0.25 \geq 0$ and $p_2 = 0.25 \geq 0$. (Satisfied)
- **Sum-to-one:** $p_1 + p_2 = 0.25 + 0.25 = 0.5 \neq 1$. (Not satisfied)

Since p does not satisfy the sum-to-one constraint, $p \notin \Delta_2$. Thus, the statement is false.

\therefore Final Answer

The statement is **false**. A counterexample is $p = \begin{bmatrix} 0.25 \\ 0.25 \end{bmatrix}$ and $c = 2$. Here, $c \cdot p = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} \in \Delta_2$, but $p \notin \Delta_2$ because its components sum to 0.5, not 1.

Graduate Level Explanation

The probability simplex Δ_k is an affine subspace of \mathbb{R}^k , specifically the intersection of the hyperplane $\sum x_i = 1$ and the non-negative orthant \mathbb{R}_+^k . While the non-negative orthant is a convex cone (closed under non-negative scalar multiplication), the hyperplane $\sum x_i = 1$ is not a linear subspace as it does not contain the origin. Scaling a vector p by $c \neq 1$ moves it off this hyperplane, thus violating the sum-to-one constraint. The set of vectors whose scaled versions lie on the simplex forms a cone over the simplex, but these vectors are not, in general, on the simplex themselves.

Explanation for a 5 year old

Imagine a recipe for one special juice drink says you need 1 cup of ingredients in total. This "1 cup total" rule is very important. You find a bottle of juice that follows the rule. Your friend says, "I have a different bottle, and if I pour out half of it, it's exactly the same as your juice." Your friend's bottle might follow the non-negativity rule (it has juice in it), but it must have had 2 cups of ingredients to begin with. So, your friend's original bottle did not follow the "1 cup total" rule.

Solution 2: Sketching the Probability Simplex Δ_3

Step 1: Define the Geometry of Δ_3

The probability simplex Δ_3 is the set of points $p = [p_1 \ p_2 \ p_3]^\top$ in \mathbb{R}^3 satisfying:

1. $p_1 \geq 0, p_2 \geq 0, p_3 \geq 0$ (it lies in the first octant).
2. $p_1 + p_2 + p_3 = 1$ (it lies on a plane).

The intersection of the plane $p_1 + p_2 + p_3 = 1$ with the first octant forms a bounded, closed shape. To identify the shape, we find its vertices.

Step 2: Identify the Vertices and the Central Point

The vertices of the shape are the points where the plane intersects the coordinate axes.

- Intersection with p_1 -axis ($p_2 = 0, p_3 = 0$): $p_1 = 1$. Vertex $v_1 = [1 \ 0 \ 0]^\top$.
- Intersection with p_2 -axis ($p_1 = 0, p_3 = 0$): $p_2 = 1$. Vertex $v_2 = [0 \ 1 \ 0]^\top$.
- Intersection with p_3 -axis ($p_1 = 0, p_2 = 0$): $p_3 = 1$. Vertex $v_3 = [0 \ 0 \ 1]^\top$.

Connecting these three vertices in 3D space forms an equilateral triangle. The "most central" point of this triangle is its barycenter (or centroid), which is the average of the coordinates of its vertices.

Step 3: Calculate the Centroid

The coordinates of the centroid p_c are:

$$p_c = \frac{v_1 + v_2 + v_3}{3} = \frac{1}{3} \left(\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \right) = \frac{1}{3} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix}$$

This point corresponds to the uniform probability distribution over three outcomes.

• Final Answer

Sketch Description: The probability simplex Δ_3 is an equilateral triangle in 3D space whose vertices are at the standard basis vectors $[1, 0, 0]^\top$, $[0, 1, 0]^\top$, and $[0, 0, 1]^\top$.

Most Central Point: The coordinates of the most central point (the barycenter) are $\begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix}^\top$.

Graduate Level Explanation

The standard k -simplex, Δ_k , is a $(k - 1)$ -dimensional convex polytope embedded in \mathbb{R}^k . For $k = 3$, this results in a 2-dimensional triangle. The vertices are the standard basis vectors e_1, e_2, e_3 , representing deterministic probability distributions. The barycenter of the simplex, $(1/k, \dots, 1/k)^\top$, corresponds to the uniform probability distribution. In information theory, this is the distribution with the maximum Shannon entropy, representing the state of maximum uncertainty.

Explanation for a 5 year old

Imagine a big glass cube. Now, imagine you slice it with a flat piece of glass. The slice starts at the number 1 on the 'x' line, goes to the number 1 on the 'y' line, and also to the number 1 on the 'z' line. The shape of this flat slice inside the corner of the cube is a perfect triangle. The very middle of that triangle is its balancing point. That special point is at $(1/3, 1/3, 1/3)$, which means it's an equal distance from all three number lines.

Solution 3: ℓ_1 Distance and KL Divergence**Step 1: Calculate the ℓ_1 Distance**

The ℓ_1 distance between two vectors $p, q \in \mathbb{R}^n$ is given by $\|p - q\|_1 = \sum_{i=1}^n |p_i - q_i|$. For $p = [1/2 \ 1/4 \ 1/8 \ 1/8]^\top$ and $q = [1/4 \ 1/4 \ 1/4 \ 1/4]^\top$:

$$\begin{aligned} \|p - q\|_1 &= \left| \frac{1}{2} - \frac{1}{4} \right| + \left| \frac{1}{4} - \frac{1}{4} \right| + \left| \frac{1}{8} - \frac{1}{4} \right| + \left| \frac{1}{8} - \frac{1}{4} \right| \\ &= \left| \frac{2}{4} - \frac{1}{4} \right| + |0| + \left| \frac{1}{8} - \frac{2}{8} \right| + \left| \frac{1}{8} - \frac{2}{8} \right| \\ &= \frac{1}{4} + 0 + \left| -\frac{1}{8} \right| + \left| -\frac{1}{8} \right| \\ &= \frac{1}{4} + \frac{1}{8} + \frac{1}{8} = \frac{2}{8} + \frac{1}{8} + \frac{1}{8} = \frac{4}{8} = \frac{1}{2} \end{aligned}$$

Step 2: Calculate the KL Divergence $K(p, q)$

The Kullback-Leibler (KL) divergence from q to p is $K(p, q) = \sum_{i=1}^n p_i \ln \frac{p_i}{q_i}$.

$$\begin{aligned} K(p, q) &= p_1 \ln \left(\frac{p_1}{q_1} \right) + p_2 \ln \left(\frac{p_2}{q_2} \right) + p_3 \ln \left(\frac{p_3}{q_3} \right) + p_4 \ln \left(\frac{p_4}{q_4} \right) \\ &= \frac{1}{2} \ln \left(\frac{1/2}{1/4} \right) + \frac{1}{4} \ln \left(\frac{1/4}{1/4} \right) + \frac{1}{8} \ln \left(\frac{1/8}{1/4} \right) + \frac{1}{8} \ln \left(\frac{1/8}{1/4} \right) \\ &= \frac{1}{2} \ln(2) + \frac{1}{4} \ln(1) + \frac{1}{8} \ln \left(\frac{1}{2} \right) + \frac{1}{8} \ln \left(\frac{1}{2} \right) \\ &= \frac{1}{2} \ln(2) + 0 - \frac{1}{8} \ln(2) - \frac{1}{8} \ln(2) \\ &= \left(\frac{1}{2} - \frac{1}{8} - \frac{1}{8} \right) \ln(2) = \left(\frac{4}{8} - \frac{2}{8} \right) \ln(2) = \frac{2}{8} \ln(2) = \frac{1}{4} \ln(2) \end{aligned}$$

Step 3: Calculate the KL Divergence $K(q, p)$

The KL divergence from p to q is $K(q, p) = \sum_{i=1}^n q_i \ln \frac{q_i}{p_i}$.

$$\begin{aligned} K(q, p) &= q_1 \ln \left(\frac{q_1}{p_1} \right) + q_2 \ln \left(\frac{q_2}{p_2} \right) + q_3 \ln \left(\frac{q_3}{p_3} \right) + q_4 \ln \left(\frac{q_4}{p_4} \right) \\ &= \frac{1}{4} \ln \left(\frac{1/4}{1/2} \right) + \frac{1}{4} \ln \left(\frac{1/4}{1/4} \right) + \frac{1}{4} \ln \left(\frac{1/4}{1/8} \right) + \frac{1}{4} \ln \left(\frac{1/4}{1/8} \right) \\ &= \frac{1}{4} \ln \left(\frac{1}{2} \right) + \frac{1}{4} \ln(1) + \frac{1}{4} \ln(2) + \frac{1}{4} \ln(2) \\ &= -\frac{1}{4} \ln(2) + 0 + \frac{1}{4} \ln(2) + \frac{1}{4} \ln(2) \\ &= \frac{1}{4} \ln(2) \end{aligned}$$

Final Answer

For the given probability distributions p and q :

- The ℓ_1 distance is $\|p - q\|_1 = \frac{1}{2}$.
- The KL divergence from q to p is $K(p, q) = \frac{1}{4} \ln(2)$.
- The KL divergence from p to q is $K(q, p) = \frac{1}{4} \ln(2)$.

Graduate Level Explanation

The ℓ_1 distance is a true metric satisfying symmetry and the triangle inequality; on the probability simplex, it is equivalent to twice the total variation distance. The Kullback-Leibler divergence, conversely, is not a metric. It is asymmetric ($K(p, q) \neq K(q, p)$ in general, although they coincide in this specific case) and does not satisfy the triangle inequality. It is a Bregman divergence generated by the negative entropy function, and it quantifies the expected inefficiency (in terms of information) of using a code optimized for distribution q to encode data from the true distribution p . By Gibbs' inequality, $K(p, q) \geq 0$ with equality if and only if $p = q$.

Explanation for a 5 year old

L1 Distance: Imagine you have two towers built from 4 kinds of colored blocks. Tower P has 4 red, 2 blue, 1 green, 1 yellow. Tower Q has 2 red, 2 blue, 2 green, 2 yellow. The "distance" is how many blocks you have to move to make Tower P look exactly like Tower Q. You need to take 2 red blocks away and add 1 green and 1 yellow. That's 4 moves in total. Our math gives an answer of 1/2, which is like a grown-up way of counting this.

KL Divergence: This is like a guessing game. Your bag of marbles has the colors mixed like in Tower P. Your friend thinks the colors are mixed like in Tower Q. The KL number measures how "surprised" your friend will be, on average, each time they pull a marble from your bag. A bigger number means more surprise! It's not usually the same amount of surprise as if you pulled from their bag, but for these special towers, it happens to be the same.