

```

1 import numpy as np
2 import pdb
3
4
5 class KNN(object):
6
7     def __init__(self):
8         pass
9
10    def train(self, X, y):
11        """
12        Inputs:
13        - X is a numpy array of size (num_examples, D)
14        - y is a numpy array of size (num_examples, )
15        """
16        self.X_train = X
17        self.y_train = y
18
19    def compute_distances(self, X, norm=None):
20        """
21        Compute the distance between each test point in X and each training point
22        in self.X_train.
23
24        Inputs:
25        - X: A numpy array of shape (num_test, D) containing test data.
26        - norm: the function with which the norm is taken.
27
28        Returns:
29        - dists: A numpy array of shape (num_test, num_train) where dists[i, j]
30            is the Euclidean distance between the ith test point and the jth
31            training point.
32        """
33        if norm is None:
34            norm = lambda x: np.sqrt(np.sum(x**2))
35            #norm = 2
36
37        num_test = X.shape[0]
38        num_train = self.X_train.shape[0]
39        dists = np.zeros((num_test, num_train))
40
41        for i in np.arange(num_test):
42            for j in np.arange(num_train):
43                # ===== #
44                # YOUR CODE HERE:
45                #   Compute the distance between the ith test point and the jth
46                #   training point using norm(), and store the result in dists[i, j].
47
48                # ===== #
49
50                dist = norm(X[i] - self.X_train[j])
51                dists[i][j] = dist
52
53                # ===== #
54                # END YOUR CODE HERE
55                # ===== #
56
57        return dists

```

```

58 def compute_L2_distances_vectorized(self, X):
59     """
60     Compute the distance between each test point in X and each training point
61     in self.X_train WITHOUT using any for loops.
62
63     Inputs:
64     - X: A numpy array of shape (num_test, D) containing test data.
65
66     Returns:
67     - dists: A numpy array of shape (num_test, num_train) where dists[i, j]
68         is the Euclidean distance between the ith test point and the jth
69         training point.
70     """
71     num_test = X.shape[0]
72     num_train = self.X_train.shape[0]
73     dists = np.zeros((num_test, num_train))
74
75     # ===== #
76     # YOUR CODE HERE:
77     # Compute the L2 distance between the ith test point and the jth
78     # training point and store the result in dists[i, j]. You may
79     # NOT use a for loop (or list comprehension). You may only use
80     # numpy operations.
81     #
82     # HINT: use broadcasting. If you have a shape (N,1) array and
83     # a shape (M,) array, adding them together produces a shape (N, M)
84     # array.
85     # ===== #
86
87     X2 = np.sum(X**2, axis=1).reshape((num_test, 1)) # shape is (num_test, 1)
88     Y2 = np.sum(self.X_train**2, axis=1).reshape((1, num_train)) # shape is
89     (1, num_train)
90     XY = X.dot(self.X_train.T) # shape is (num_test, num_train)
91     dists = np.sqrt(X2 + Y2 - 2*XY) # shape is (num_test, num_train)
92
93     # ===== #
94     # END YOUR CODE HERE
95     # ===== #
96
97     return dists
98
99 def predict_labels(self, dists, k=1):
100     """
101     Given a matrix of distances between test points and training points,
102     predict a label for each test point.
103
104     Inputs:
105     - dists: A numpy array of shape (num_test, num_train) where dists[i, j]
106         gives the distance between the ith test point and the jth training
107         point.
108
109     Returns:
110     - y: A numpy array of shape (num_test,) containing predicted labels for
111         the test data, where y[i] is the predicted label for the test point X[i].
112     """
113     num_test = dists.shape[0]
114     y_pred = np.zeros(num_test)

```

```
114 for i in np.arange(num_test):
115     # A list of length k storing the labels of the k nearest neighbors to
116     # the ith test point.
117     closest_y = []
118     # ===== #
119     # YOUR CODE HERE:
120     # Use the distances to calculate and then store the labels of
121     # the k-nearest neighbors to the ith test point. The function
122     # numpy.argsort may be useful.
123     #
124     # After doing this, find the most common label of the k-nearest
125     # neighbors. Store the predicted label of the ith training example
126     # as y_pred[i]. Break ties by choosing the smaller label.
127     # ===== #
128
129     sortedIdxs = np.argsort(dists[i])
130     closest_y = self.y_train[sortedIdxs[:k]]
131     y_pred[i] = np.argmax(np.bincount(closest_y))
132
133     # ===== #
134     # END YOUR CODE HERE
135     # ===== #
136
137 return y_pred
138
```