```python
import numpy as np
import pdb

"""
This code was based off of code from cs231n at Stanford University, and modified for CS145 at UCLA.
"""

class KNN(object):

    def __init__(self):
        pass

    def train(self, X, y):
        """
        Inputs:
        - X is a numpy array of size (num_examples, D)
        - y is a numpy array of size (num_examples, )
        """
        # ========================================================== #
        # START YOUR CODE HERE
        # ========================================================== #
        #   Hint: KNN does not do any further processsing, just store the training
        #   samples with labels into as self.X_train and self.y_train
        # ========================================================== #
        self.X_train = X
        self.y_train = y
        # ========================================================== #
        # END YOUR CODE HERE
        # ========================================================== #

    def compute_distances(self, X, norm=None):
        """
        Compute the distance between each test point in X and each training point
        in self.X_train.

        Inputs:
        - X: A numpy array of shape (num_test, D) containing test data.
        - norm: the function with which the norm is taken.

        Returns:
        - dists: A numpy array of shape (num_test, num_train) where dists[i, j]
          is the Euclidean distance between the ith test point and the jth training
          point.
        """
        if norm is None:
            norm = lambda x: np.sqrt(np.sum(x**2)) #norm = 2

        num_test = X.shape[0]
        num_train = self.X_train.shape[0]
        dists = np.zeros((num_test, num_train))
        for i in np.arange(num_test):

            for j in np.arange(num_train):
            #
========================================================== #
```

```
            # START YOUR CODE HERE
            #
================================================================ #
            #   Compute the distance between the ith test point and the jth

            #   training point using norm(), and store the result in dists[i,
j].
            #
================================================================ #
                dists[i,j] = norm(X[i] - self.X_train[j])
            #
================================================================ #
            # END YOUR CODE HERE
            #
================================================================ #

        return dists

    def compute_L2_distances_vectorized(self, X):
        """
        Compute the distance between each test point in X and each training
point
        in self.X_train WITHOUT using any for loops.

        Inputs:
        - X: A numpy array of shape (num_test, D) containing test data.

        Returns:
        - dists: A numpy array of shape (num_test, num_train) where dists[i,
j]
          is the Euclidean distance between the ith test point and the jth
training
          point.
        """
        num_test = X.shape[0]
        num_train = self.X_train.shape[0]
        dists = np.zeros((num_test, num_train))

        # ================================================================ #
        # START YOUR CODE HERE
        # ================================================================ #
        #   Compute the L2 distance between the ith test point and the jth

        #   training point and store the result in dists[i, j].  You may
        #    NOT use a for loop (or list comprehension).  You may only use
        #     numpy operations.
        #
        #     HINT: use broadcasting.  If you have a shape (N,1) array and
        #   a shape (M,) array, adding them together produces a shape (N, M)
        #   array.
        # ================================================================ #
        # I referenced https://stackoverflow.com/questions/27948363/numpy-
broadcast-to-perform-euclidean-distance-vectorized
        # to solve this part
        X_square = np.square(X).sum(axis = 1)
        X_train_square = np.square(self.X_train).sum(axis = 1)
        X_square_reshape = X_square.reshape((num_test, 1))
        element_wise_prod = 2 * (np.dot(X, (self.X_train).T))
        dists = np.sqrt(X_square_reshape - element_wise_prod +
X_train_square)
```

```
103         # ========================================================== #
104         # END YOUR CODE HERE
105         # ========================================================== #
106 
107         return dists
108 
109 
110     def predict_labels(self, dists, k=1):
111         """
112         Given a matrix of distances between test points and training points,
113         predict a label for each test point.
114 
115         Inputs:
116         - dists: A numpy array of shape (num_test, num_train) where dists[i,
    j]
117           gives the distance betwen the ith test point and the jth training
    point.
118 
119         Returns:
120         - y: A numpy array of shape (num_test,) containing predicted labels
    for the
121           test data, where y[i] is the predicted label for the test point
    X[i].
122         """
123         num_test = dists.shape[0]
124         y_pred = np.zeros(num_test)
125         for i in range(num_test):
126             # A list of length k storing the labels of the k nearest
    neighbors to
127             # the ith test point.
128 
129             closest_y = []
130 
131             #
    ========================================================== #
132             # START YOUR CODE HERE
133             #
    ========================================================== #
134             #   Use the distances to calculate and then store the labels of
135             #   the k-nearest neighbors to the ith test point.  The function
136             #   numpy.argsort may be useful.
137             #
138             #   After doing this, find the most common label of the k-nearest
139             #   neighbors.  Store the predicted label of the ith training
    example
140             #   as y_pred[i].  Break ties by choosing the smaller label.
141             #
    ========================================================== #
142             sorted_indices = np.argsort(dists[i])
143             common_labels = self.y_train[sorted_indices[:k]]
144             y_pred[i] = np.bincount(common_labels).argmax()
145             #
    ========================================================== #
146             # END YOUR CODE HERE
147             #
    ========================================================== #
148         return y_pred
149 
```