```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].
            # ================================================================ #
                diff = X[i] - self.X_train[j]
                dists[i][j] = norm(diff)
            # ================================================================ #
            # 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.
        # ================================================================ #
        M = np.dot(X, (self.X_train).T) # dot procuct between testing and training data
        train_data_squares = np.square(self.X_train).sum(axis = 1) # sums across cols the squares of each entry in train data
        # print(train_data_squares.shape)
        test_data_squares = np.square(X).sum(axis = 1) # sums across cols the squares of each entry in test data
        # print(X)
```

```python
        # print("Test Data Squares size and entries")
        # print(test_data_squares.shape)
        # print(test_data_squares)
        test_columnar = test_data_squares.reshape((num_test, 1)) #transforms the sum into a vector form of N,1
        # print("Test Broadcast size and entries")
        # print(test_broadcast.shape)
        # print(test_broadcast)
        dists = np.sqrt(test_columnar + train_data_squares - 2 * M) # Formula from notes
        # ================================================================ #
        # END YOUR CODE HERE
        # ================================================================ #
        # print(dists)
        return dists


    def predict_labels(self, dists, k=1):
        """
        Given a matrix of distances between test points and training points,
        predict a label for each test point.

        Inputs:
        - dists: A numpy array of shape (num_test, num_train) where dists[i, j]
          gives the distance betwen the ith test point and the jth training point.

        Returns:
        - y: A numpy array of shape (num_test,) containing predicted labels for the
          test data, where y[i] is the predicted label for the test point X[i].
        """
        num_test = dists.shape[0]
        y_pred = np.zeros(num_test)
        for i in range(num_test):
            # A list of length k storing the labels of the k nearest neighbors to
            # the ith test point.

            closest_y = []

            # ================================================================ #
            # START YOUR CODE HERE
            # ================================================================ #
            #   Use the distances to calculate and then store the labels of
            #   the k-nearest neighbors to the ith test point.  The function
            #   numpy.argsort may be useful.
            #
            #   After doing this, find the most common label of the k-nearest
            #   neighbors.  Store the predicted label of the ith training example
            #   as y_pred[i].  Break ties by choosing the smaller label.
            # ================================================================ #
            sorted_dists = np.argsort(dists[i,:])
            closest_ys= sorted_dists[:k]
```

```python
            labels = self.y_train[closest_ys]
            y_pred[i] = np.bincount(labels).argmax()
            # ================================================================ #
            # END YOUR CODE HERE
            # ================================================================ #
        return y_pred

```