```
import numpy as np
import pdb
import heapq
import copy
"""
This code was based off of code from cs231n at Stanford University, and modified
for ECE C147/C247 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, )
    """
    self.X_train = X
    self.y_train = y

  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):

      temp = []
      for j in np.arange(num_train):
                # ================================================================
#
        # 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].
        # ================================================================ #
        temp.append(norm(X[i]-self.X_train[j]))
      dists[i] = temp


                # ================================================================
#
                # 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))

        # ========================================================== #
        # 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.
        # ========================================================== #

    test_sqr = np.sum(X * X, axis = 1)
    test_sqr_ext = np.asarray([test_sqr] * num_train).T
    train_sqr = np.sum(self.X_train * self.X_train, axis = 1)
    train_sqr_ext = np.asarray([train_sqr] * num_test)
    test_train = X.dot(self.X_train.T)
    dists = np.sqrt(test_sqr_ext + train_sqr_ext - 2 * test_train)


        # ========================================================== #
        # END YOUR CODE HERE
        # ========================================================== #

    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].
    """

    def max_list(lt):
      temp = 0
      for i in lt:
        if lt.count(i) > temp:
          max_str = i
          temp = lt.count(i)
```

```
      return max_str

    num_test = dists.shape[0]
    y_pred = []
    for i in np.arange(num_test):
      # A list of length k storing the labels of the k nearest neighbors to
      # the ith test point.
      for i1 in range(k):
        temp_ls = copy.deepcopy(dists[i]).tolist()
        min_number = heapq.nsmallest(k,temp_ls)
        min_index = []
        for t in min_number:
          index = temp_ls.index(t)
          min_index.append(index)
          temp_ls[index] = 1000000
      y_pred.append(max_list(self.y_train[min_index].tolist()))
    y_pred = np.asarray(y_pred)
          # ========================================================================= #
          # 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.
          # ========================================================================= #


          # ========================================================================= #
          # END YOUR CODE HERE
          # ========================================================================= #

    return y_pred
```