Name: Rezwan Ahmad ID:20-44273-3 Section: C  
KNN-HW Assignment

KNN\_assignment.py:

import numpy as np

import matplotlib.pyplot as plt

import data\_utils

import download

def download\_data():

    url = "https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz"

    download\_dir = "./data"

    download.maybe\_download\_and\_extract(url, download\_dir)

# Class to initialize and apply K-nearest neighbour classifier

class KNearestNeighbor(object):

    def \_\_init\_\_(self):

        pass

    # Method to initialize classifier with training data

    def train(self, X, y):

        self.X\_train = X

        self.y\_train = y

    # Method to predict labels of test examples using 'compute\_distances' and 'predict\_labels' methods.

    def predict(self, X, k=1, num\_loops=0):

        if num\_loops == 0:

            dists = self.compute\_distances(X)

        else:

            raise ValueError('Invalid value %d for num\_loops' % num\_loops)

        return self.predict\_labels(dists, k=k)

    # Method to compute Euclidean distances from each text example to every training example

    def compute\_distances(self, X):

        num\_test = X.shape[0]

        num\_train = self.X\_train.shape[0]

        dists = np.zeros((num\_test, num\_train))

        # Compute distances from each test example (in argument 'X' of this method) to every training example and store distances in

        # dists variable given above. For each row, i, dist[i] should contain distances between test example i and every training example.

        for i in range(num\_test):

            dists[i] = np.sqrt(np.sum(np.square(self.X\_train - X[i]), axis=1))

        return dists

    # Method to predict labels of test examples using chosen value of k given Euclidean distances obtained from 'compute\_distances' method.

    def predict\_labels(self, dists, k=1):

        num\_test = dists.shape[0]

        y\_pred = np.zeros(num\_test)

        # Given dists computed using 'compute\_distances' method above, obtain k closest distances to training examples for each test example

        # dists[i]. Use k closest distances obtained to predict label of each dists[i]. Label of each dists[i] should be stored in y\_pred[i].

        for i in range(num\_test):

            closest\_y = self.y\_train[np.argsort(dists[i])[:k]]

            y\_pred[i] = np.argmax(np.bincount(closest\_y))

        return y\_pred

def visualize\_data(X\_train, y\_train):

    classes = ['plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']

    num\_classes = len(classes)

    samples\_per\_class = 7

    for y, cls in enumerate(classes):

        idxs = np.flatnonzero(y\_train == y)

        idxs = np.random.choice(idxs, samples\_per\_class, replace=False)

        for i, idx in enumerate(idxs):

            plt\_idx = i \* num\_classes + y + 1

            plt.subplot(samples\_per\_class, num\_classes, plt\_idx)

            plt.imshow(X\_train[idx].astype('uint8'))

            plt.axis('off')

            if i == 0:

                plt.title(cls)

    plt.show()

if \_\_name\_\_ == "\_\_main\_\_":

    # download\_data()

    cifar10\_dir = 'D:\\AIUB\\Academics\\Semester 10\\ML\\Mid Assignment\\Assignment 1\\KNN-HW\\cifar-10-batches-py'

    X\_train, y\_train, X\_test, y\_test = data\_utils.load\_CIFAR10(cifar10\_dir)

    print('Training data shape: ', X\_train.shape)

    print('Training labels shape: ', y\_train.shape)

    print('Test data shape: ', X\_test.shape)

    print('Test labels shape: ', y\_test.shape)

    visualize\_data(X\_train, y\_train)

    num\_training = 8000

    mask = list(range(num\_training))

    X\_train = X\_train[mask]

    y\_train = y\_train[mask]

    num\_test = 2000

    mask = list(range(num\_test))

    X\_test = X\_test[mask]

    y\_test = y\_test[mask]

    X\_train = np.reshape(X\_train, (X\_train.shape[0], -1))

    X\_test = np.reshape(X\_test, (X\_test.shape[0], -1))

    print(X\_train.shape, X\_test.shape)

    classifier = KNearestNeighbor()

    classifier.train(X\_train, y\_train)

    y\_test\_pred = classifier.predict(X\_test, k=5)

    num\_correct = np.sum(y\_test\_pred == y\_test)

    accuracy = float(num\_correct) / num\_test

    print('Got %d / %d correct with k=5 => accuracy: %f' % (num\_correct, num\_test, accuracy))

    num\_folds = 5

    k\_choices = [1, 3, 5, 8, 10]

    X\_train\_folds = np.array\_split(X\_train, num\_folds)

    y\_train\_folds = np.array\_split(y\_train, num\_folds)

    k\_to\_accuracies = {}

    for k in k\_choices:

        k\_to\_accuracies[k] = []

        for i in range(num\_folds):

            print("debug")

            X\_train\_cv = np.vstack(X\_train\_folds[:i] + X\_train\_folds[i + 1:])

            y\_train\_cv = np.hstack(y\_train\_folds[:i] + y\_train\_folds[i + 1:])

            X\_val\_cv = X\_train\_folds[i]

            y\_val\_cv = y\_train\_folds[i]

            classifier\_cv = KNearestNeighbor()

            classifier\_cv.train(X\_train\_cv, y\_train\_cv)

            dists = classifier\_cv.compute\_distances(X\_val\_cv)

            y\_val\_pred = classifier\_cv.predict\_labels(dists, k=k)

            num\_correct = np.sum(y\_val\_pred == y\_val\_cv)

            accuracy = float(num\_correct) / len(y\_val\_cv)

            k\_to\_accuracies[k].append(accuracy)

    print("Printing our 5-fold accuracies for varying values of k:")

    print("working")

    for k in sorted(k\_to\_accuracies):

        for accuracy in k\_to\_accuracies[k]:

            print('k = %d, accuracy = %f' % (k, accuracy))

    for k in sorted(k\_to\_accuracies):

        print('k = %d, avg. accuracy = %f' % (k, sum(k\_to\_accuracies[k]) / num\_folds))

    for k in k\_choices:

        accuracies = k\_to\_accuracies[k]

        plt.scatter([k] \* len(accuracies), accuracies)

    accuracies\_mean = np.array([np.mean(v) for k, v in sorted(k\_to\_accuracies.items())])

    accuracies\_std = np.array([np.std(v) for k, v in sorted(k\_to\_accuracies.items())])

    plt.errorbar(k\_choices, accuracies\_mean, yerr=accuracies\_std)

    plt.title('Cross-validation on k')

    plt.xlabel('k')

    plt.ylabel('Cross-validation accuracy')

    plt.savefig('cross-validation\_accuracy.jpg')

    best\_k = max(k\_to\_accuracies, key=lambda k: np.mean(k\_to\_accuracies[k]))

    classifier\_best\_k = KNearestNeighbor()

    classifier\_best\_k.train(X\_train, y\_train)

    y\_test\_pred\_best\_k = classifier\_best\_k.predict(X\_test, k=best\_k)

    num\_correct = np.sum(y\_test\_pred\_best\_k == y\_test)

    accuracy = float(num\_correct) / num\_test

    print('Got %d / %d correct on test data => accuracy: %f' % (num\_correct, num\_test, accuracy))

Results: (excluded multiple ‘debug’s as it was used to check if the code was working or not)

PS D:\AIUB\Academics\Semester 10\ML\Mid Assignment\Assignment 1\KNN-HW> & C:/Users/rez1b/AppData/Local/Programs/Python/Python312/python.exe "d:/AIUB/Academics/Semester 10/ML/Mid Assignment/Assignment 1/KNN-HW/KNN\_assignment.p

y"

Training data shape: (50000, 32, 32, 3)

Training labels shape: (50000,)

Test data shape: (10000, 32, 32, 3)

Test labels shape: (10000,)

(8000, 3072) (2000, 3072)

Got 546 / 2000 correct with k=5 => accuracy: 0.273000

Debug

.

.

.

debug

Printing our 5-fold accuracies for varying values of k:

working

k = 1, accuracy = 0.273125

k = 1, accuracy = 0.280000

k = 1, accuracy = 0.288125

k = 1, accuracy = 0.268125

k = 1, accuracy = 0.278750

k = 3, accuracy = 0.260625

k = 3, accuracy = 0.274375

k = 3, accuracy = 0.276875

k = 3, accuracy = 0.244375

k = 3, accuracy = 0.261250

k = 5, accuracy = 0.281875

k = 5, accuracy = 0.276875

k = 5, accuracy = 0.288750

k = 5, accuracy = 0.269375

k = 5, accuracy = 0.266250

k = 8, accuracy = 0.284375

k = 8, accuracy = 0.280000

k = 8, accuracy = 0.286875

k = 8, accuracy = 0.265625

k = 8, accuracy = 0.266875

k = 10, accuracy = 0.280000

k = 10, accuracy = 0.290000

k = 10, accuracy = 0.291250

k = 10, accuracy = 0.282500

k = 10, accuracy = 0.262500

k = 1, avg. accuracy = 0.277625

k = 3, avg. accuracy = 0.263500

k = 5, avg. accuracy = 0.276625

k = 8, avg. accuracy = 0.276750

k = 10, avg. accuracy = 0.281250

Got 551 / 2000 correct on test data => accuracy: 0.275500

**Report:**

**Introduction:**

This report summarizes the implementation and results of a K-nearest neighbor (KNN) classifier assignment. The assignment involved the development of a Python script named KNN\_assignment.py to implement the KNN algorithm and its application on the CIFAR-10 dataset.

**Implemented Code Overview:**

The KNN\_assignment.py script consists of several components:

1. Data Downloading: The script includes a function download.data() to download the CIFAR-10 dataset if it is not already downloaded. This function utilizes the download module to handle the download process. (data downloading was not needed as it was downloaded manually).
2. K Nearest Neighbor Class: This class initializes and applies the K-nearest neighbor classifier. It includes methods to train the classifier (train()), compute distances (compute\_distances()), predict labels (predict\_labels()), and make predictions (predict()).
3. Data Visualization: The script contains a function visualize\_data() to visualize sample images from the CIFAR-10 dataset.
4. Main Function: The main function of the script loads the CIFAR-10 dataset, preprocesses the data, trains the KNN classifier, performs cross-validation to tune hyperparameters, and evaluates the classifier's performance on the test dataset.

**Results and Observations:**

Upon running the script, the following results and observations were obtained:

Data Shapes: The training and test data shapes were printed, indicating 50000 training samples and 10000 test samples, each with a size of 32x32x3.

Data Preprocessing: The script then preprocesses the data by selecting a subset of samples for training and testing, reshaping the data into a suitable format for the classifier.

Model Training: The KNN classifier is trained on the pre-processed training data.

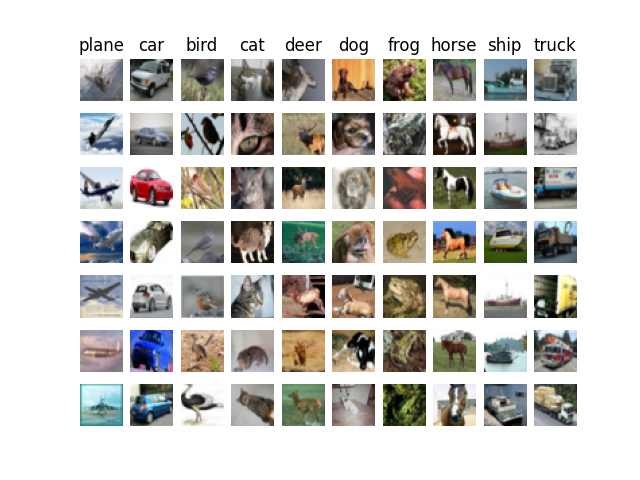
Model Evaluation: The accuracy of the trained classifier is evaluated on the test dataset. With k=5, an accuracy of 27.3% was achieved.

Cross-Validation: Cross-validation was performed with 5-fold validation for different values of k (1, 3, 5, 8, 10). The average accuracies for each value of k were calculated, with k=10 achieving the highest average accuracy of 28.1%.

Test Data Evaluation: The trained classifier with the best value of k (10) was evaluated on the test dataset, achieving an accuracy of 27.55%.

def visualize\_data(X\_train, y\_train):

    classes = ['plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']



A graph of a number of dots and lines

Description automatically generated with medium confidence

**Conclusion:**

In conclusion, the KNN classifier implemented achieved moderate accuracy on the CIFAR-10 dataset, with further optimization possible through hyperparameter tuning and potentially exploring alternative machine learning algorithms. The assignment demonstrates practical skills in implementing and evaluating machine learning models using Python.