Istanbul Technical University- Spring 2018
computer vision
Assignment 1

Aydin Ayanzadeh
student number: 504161503
ayanzadeh17@itu.edu.tr

March 6, 2018

Note: Please notice that K-fold algorithm and evaluation part of code are saved in KNN.py file. you can run it through the terminal.

For plotting 5 examples of each CIFAR-10 class ,we have used subplot function from built-in function of the python library. The code can be seen in KNN.py code which simply plots 5 examples of each class in Fig1.

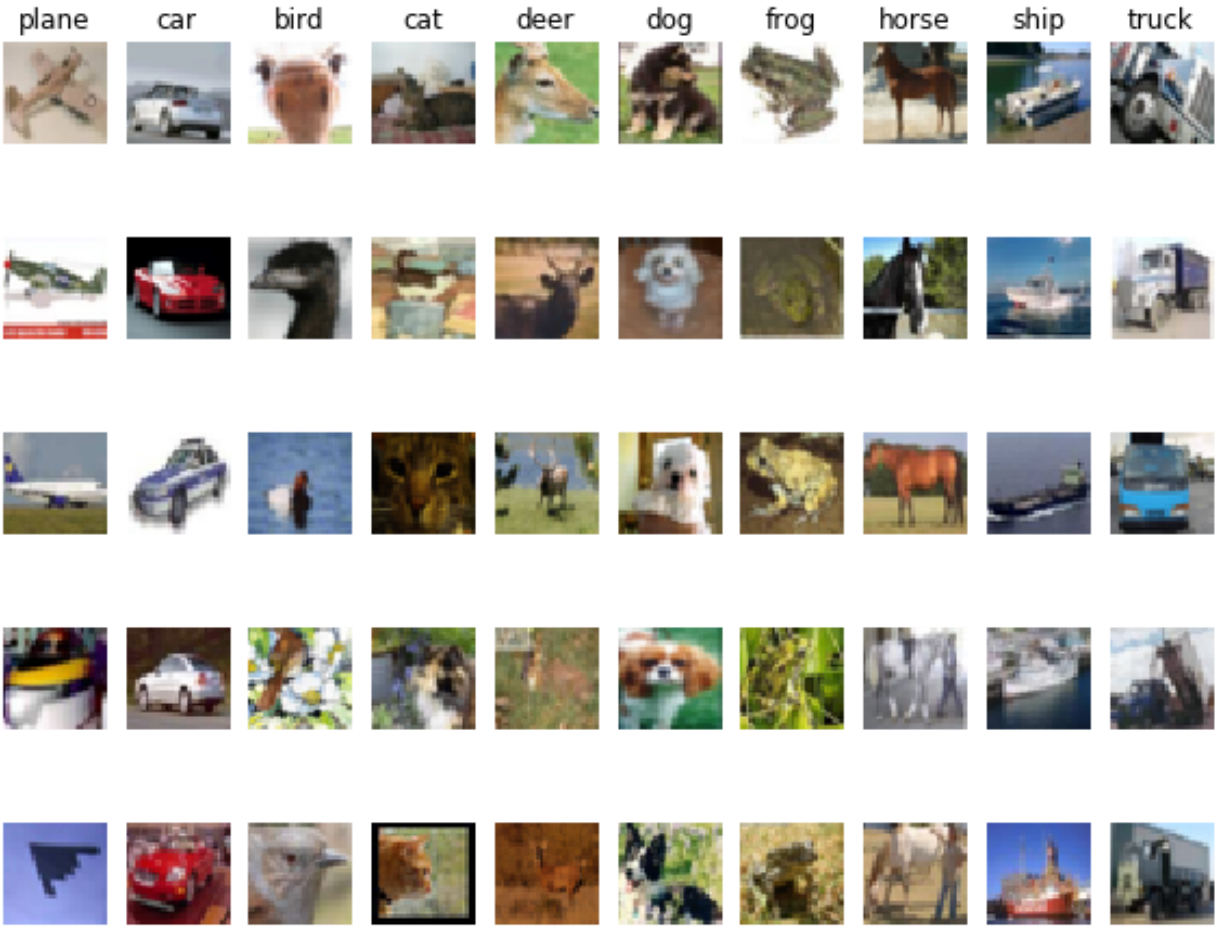

Figure 1: Sample images of CIFAR-10 dataset. There are 5 samples from each 10 classes of this dataset.

Sampling from main dataset: In order to create a new dataset containing 5000 train and 500 test samples. As can be seen in following figures number of samples in different classes are not equal. But number of samples of each class are close and can be said that data is almost balanced. Following figures show histogram of new sampled train and test dataset in Figure2.

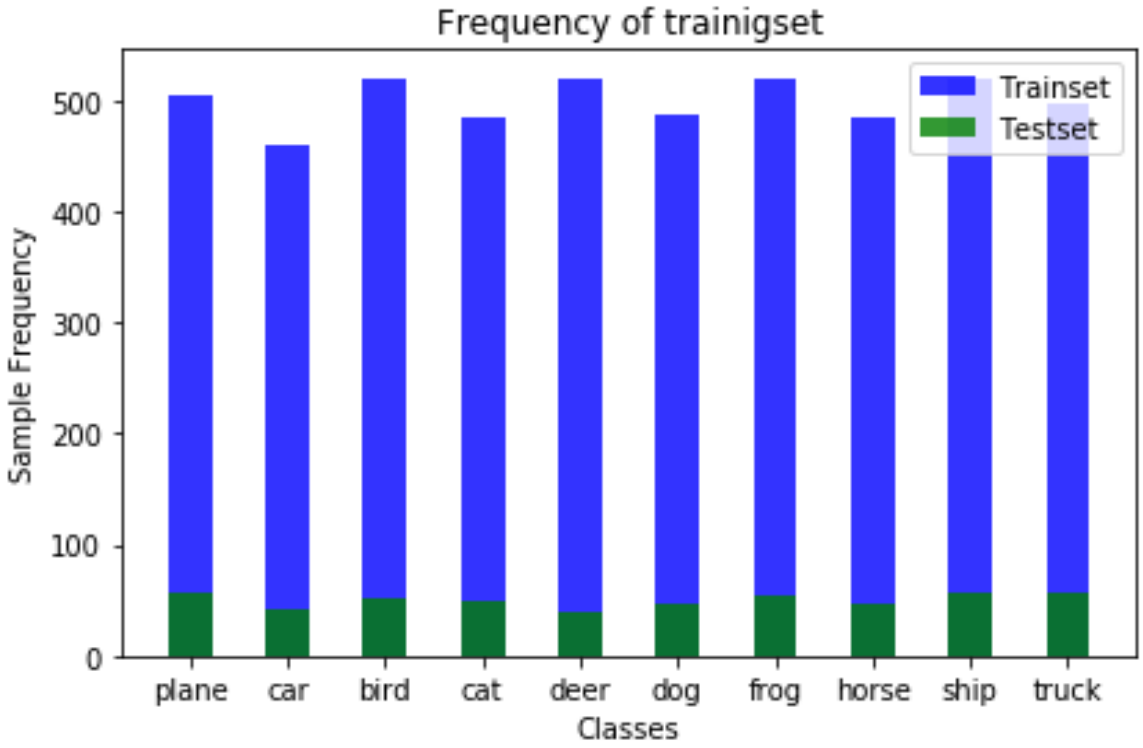

Figure 2: Class distribution of sampled training set and testset from CIFAR10 .

Implementing Euclidean distance:

Two loops: For implementing with two loops, This code calculate the distance between each instance in testset and trainset iteratively. Finally, it creates a matrix of distances in which shows distance between test instance i and train instance j. **kNearestNeighbor.py**

One loop: For implementing with only one loop, we use broadcasting technique. Actually, in this part we use a for loop in order to go through each instance of test data and in each iteration i. subtract one instance of test data from whole train matrix. Then, we use sum function along axis 1 to add al of the subtracted elements. Finally after finishing the for loop we again will have a matrix of distances.**kNearestNeighbor.py**

No loop: For this part, we use matrix multiplication to find a formula in order to calculate the Euclidean distance. After multiplying the test matrix with transpose of training matrix, each element of this new matrix is result of vector multiplication of one instance from train set and one instance form test set. Actually, in the detail, first we should take

square from $(X_train)and(X_test)$, afterwards,each of X of test ant train matrices are summed along row, besides we calculating the multiplication of $X_trainandX_test$ and multiply the result of them to negative two X_test). At the end with the help of vectorization feature in numpy. we should take the square of the the result of $(-2 * (X_train * X_test)$) at the end step.**kNearestNeighbor.py**

- **Elapsed Time:**
  In case of computation time of these three different ways of implementing I used Time package(Tic-Toc) for measuring the time computation of given methods. Computation time for no loop, one loop and two loops implementations was respectively 0.291302s, 59.508242s and 26.03s Not surprisingly no loop implementation is best implementation and one loop has the worst performance.For evaluating the performance of these, we used the computer that has the following information: Intel Core i7-4702MQ 2.2 GHz and 8 GB RAM and this is captured based on this system.

  Two loops implementation took 26.03 seconds
  One loop implementation took 59.508242 seconds
  No loop implementation took 0.291302 seconds

visualization of the Euclidean distance matrix : Figure 3 shows the Euclidean distance between the training and the test set.

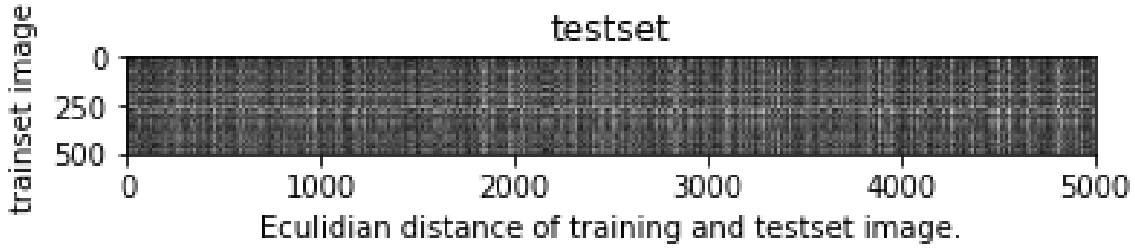

Figure 3: Euclidean distance between the training and the test set.

predicting the labels:
for predicting the labels of the instances in testset according to their distance, firstly, we sort them according to their distances using *np.argsort* an then I choose k instances with best distances and then I use *np.bincount* to counts number of occurrences of each label. Finally using argmax I find the closest instance to each query instance.
k-fold cross validation
In this part, we split our training data and labels into 5 folds and test the k for 1, 3, 5, 8, 10, 12, 15, 20, 50, 100. based on the k-cross validation technique, in each iteration we choose the 4 out of 5 fold as training set and the rest one is chosen as validation set. Following, table1

Table 1: Average Accuracy for K-fold cross validation

k	k=1	k=3	k=5	k=8	k=10	k=15	k=20	k=50	k=100
mean-Accuracy	0.2656	0.2496	0.2732	0.27599	**0.2802**	0.0.275	0.279	0.2744	0.26159

shows the mean accuracy of for each k of running the algorithm for different k parameters. Please note that all of them have done using 5-fold cross validation and these results are average accuracy(calculate the average of folds in each k). Figure 3 also shows this idea that k=10 has led to best results. By using k=10 on the sampled dataset which contains 5000 training and 500 testing samples algorithm could gain 0.2820 average accuracy. At the end you can see the results of accuracy of each k in 5-fold cross validation.

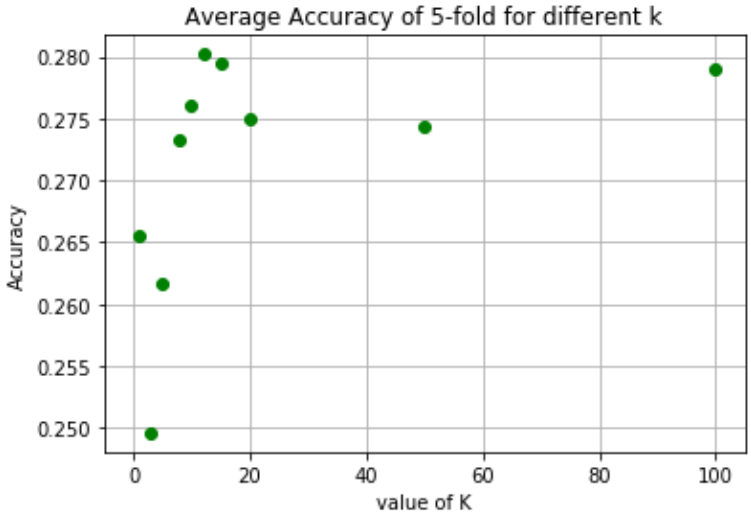

Figure 4: Impact of changing K parametr in KNN algorithm on average accuracy of classifier.

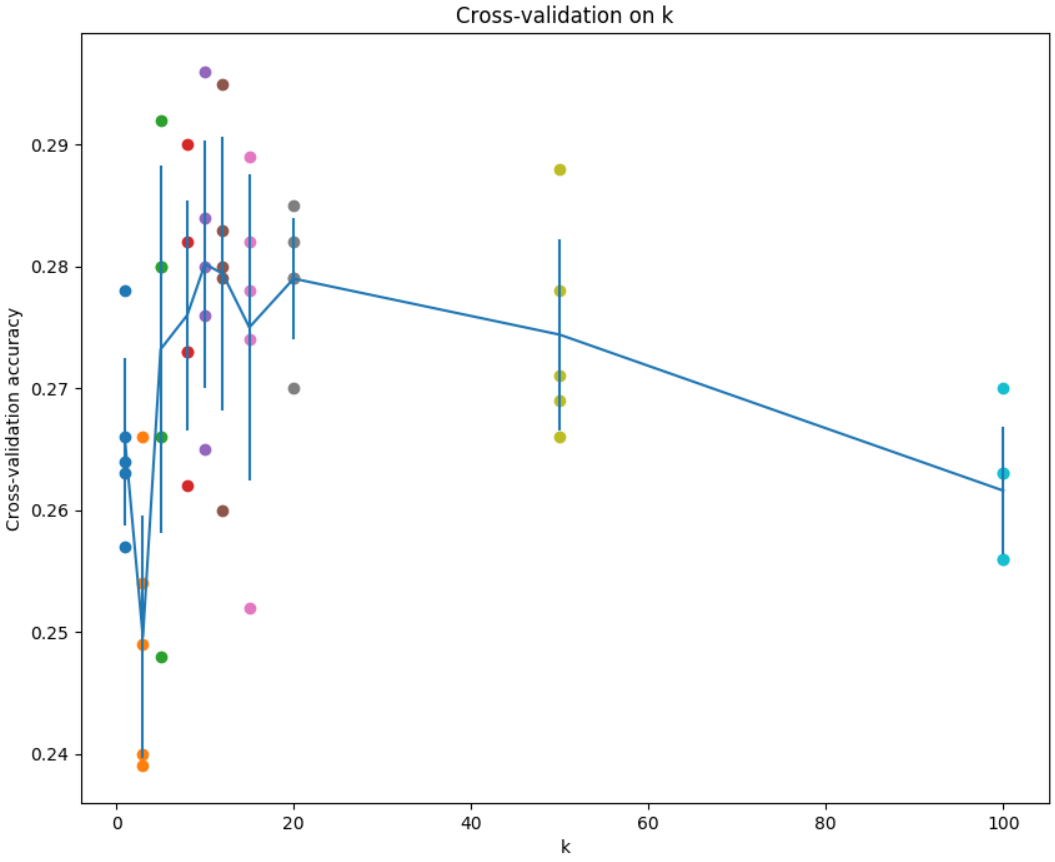

Figure 5: Impact of changing K parameter in KNN algorithm on accuracy of classifier in k-fold cross validation.