knn

October 9, 2023

[1]:
```
%cd daseCV/datasets/
!bash get_datasets.sh
%cd ../..
```

```
/home/public/10215501437-838-161/daseCV/datasets
--2023-09-30 14:39:33--  http://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
Resolving www.cs.toronto.edu (www.cs.toronto.edu)… 128.100.3.30
Connecting to www.cs.toronto.edu (www.cs.toronto.edu)|128.100.3.30|:80…
connected.
HTTP request sent, awaiting response… 200 OK
Length: 170498071 (163M) [application/x-gzip]
Saving to: ‘cifar-10-python.tar.gz’

cifar-10-python.tar 100%[===================>] 162.60M  24.1KB/s    in 2h 27m

2023-09-30 17:07:10 (18.8 KB/s) - ‘cifar-10-python.tar.gz’ saved
[170498071/170498071]

cifar-10-batches-py/
cifar-10-batches-py/data_batch_4
cifar-10-batches-py/readme.html
cifar-10-batches-py/test_batch
cifar-10-batches-py/data_batch_3
cifar-10-batches-py/batches.meta
cifar-10-batches-py/data_batch_2
cifar-10-batches-py/data_batch_5
cifar-10-batches-py/data_batch_1
/home/public/10215501437-838-161
```

1 K- (kNN)

kNN :

-
- , kNN k
- k

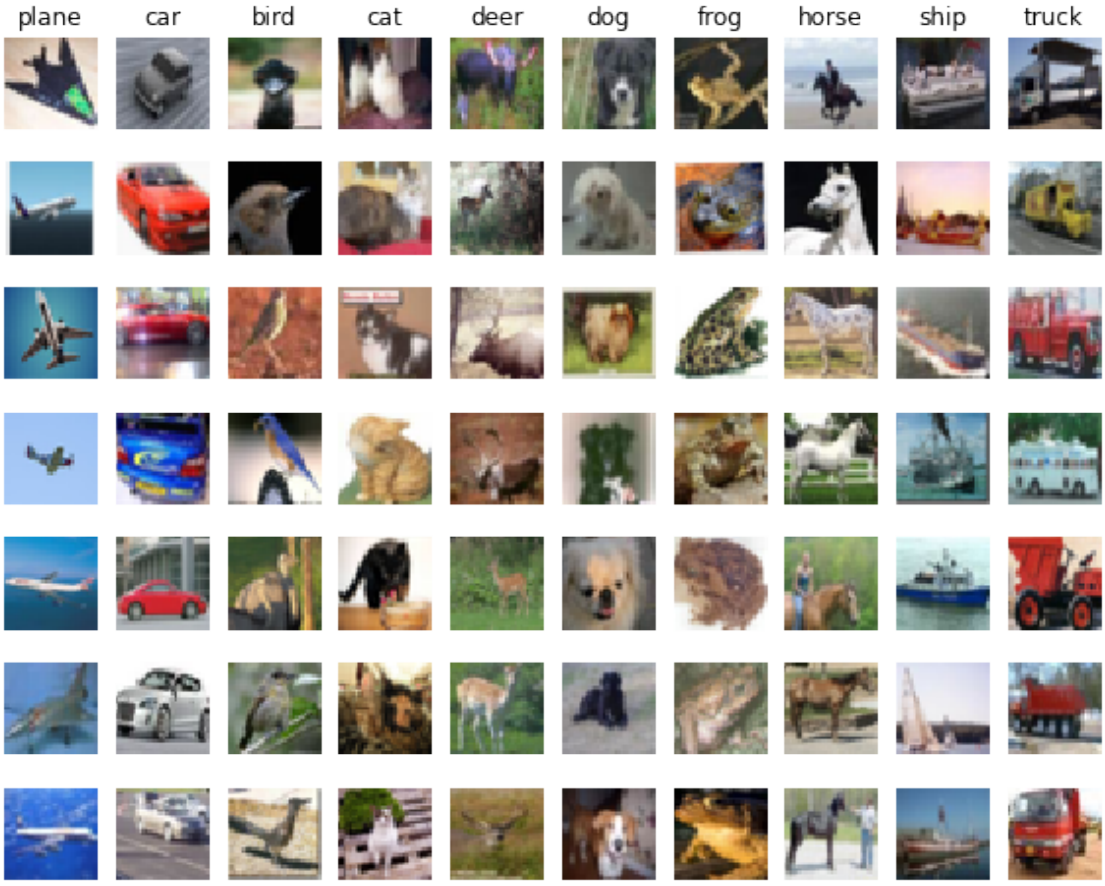
[2]:
```
#  notebook

import random
import numpy as np
from daseCV.data_utils import load_CIFAR10
import matplotlib.pyplot as plt

#  matplotlib
%matplotlib inline
plt.rcParams['figure.figsize'] = (10.0, 8.0) # set default size of plots
plt.rcParams['image.interpolation'] = 'nearest'
plt.rcParams['image.cmap'] = 'gray'

#     notebook    python ;
#   http://stackoverflow.com/questions/1907993/autoreload-of-modules-in-ipython
%load_ext autoreload
%autoreload 2
```

[3]:
```
#     CIFAR-10  .
cifar10_dir = 'daseCV/datasets/cifar-10-batches-py'

#
try:
   del X_train, y_train
   del X_test, y_test
   print('Clear previously loaded data.')
except:
   pass

X_train, y_train, X_test, y_test = 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)
```

```
Training data shape:  (50000, 32, 32, 3)
Training labels shape:  (50000,)
Test data shape:  (10000, 32, 32, 3)
Test labels shape:  (10000,)
```

[4]:
```
#
#
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) # flatnonzero                y
    idxs = np.random.choice(idxs, samples_per_class, replace=False) #␣
 ↪replace
    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()
```



[5]:
```
#
num_training = 5000
mask = list(range(num_training))
X_train = X_train[mask]
```

```
y_train = y_train[mask]

num_test = 500
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)
```

```
(5000, 3072) (500, 3072)
```

[6]:
```
from daseCV.classifiers import KNearestNeighbor

#   kNN
#   kNN
#
classifier = KNearestNeighbor()
classifier.train(X_train, y_train)
```

kNN

1.
2. k

Ntr **Nte** , **Nte x Ntr** (i,j) i j

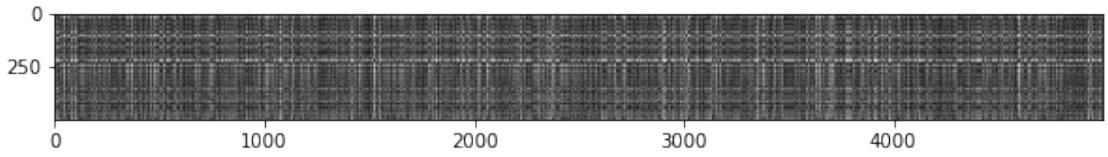
: **notebook** **numpy** **np.linalg.norm()**

daseCV/classifiers/k_nearest_neighbor.py compute_distances_two_loops

[7]:
```
#   daseCV/classifiers/k_nearest_neighbor.py
# compute_distances_two_loops.

#    :
dists = classifier.compute_distances_two_loops(X_test)
print(dists.shape)
```

```
(500, 5000)
```

[8]:
```
#
plt.imshow(dists, interpolation='none')
plt.show()
```



1

-
- ?

: 1.
2.

[9]:
```
#    predict_labels

#   k = 1
y_test_pred = classifier.predict_labels(dists, k=1)

#
num_correct = np.sum(y_test_pred == y_test)
accuracy = float(num_correct) / num_test
print('Got %d / %d correct => accuracy: %f' % (num_correct, num_test, accuracy))
```

```
Got 137 / 500 correct => accuracy: 0.274000
```

27% k, k = 5:

[10]:
```
y_test_pred = classifier.predict_labels(dists, k=5)
num_correct = np.sum(y_test_pred == y_test)
accuracy = float(num_correct) / num_test
print('Got %d / %d correct => accuracy: %f' % (num_correct, num_test, accuracy))
```

```
Got 139 / 500 correct => accuracy: 0.278000
```

k = 1

2

L1

I_k (i,j) $p_{ij}^{(k)}$

μ

$$\mu = \frac{1}{nhw}\sum_{k=1}^{n}\sum_{i=1}^{h}\sum_{j=1}^{w} p_{ij}^{(k)}$$

μ_{ij}

$$\mu_{ij} = \frac{1}{n}\sum_{k=1}^{n} p_{ij}^{(k)}.$$

σ σ_{ij}

L1 1. μ $(\tilde{p}_{ij}^{(k)} = p_{ij}^{(k)} - \mu.)$ 2. μ_{ij} $(\tilde{p}_{ij}^{(k)} = p_{ij}^{(k)} - \mu_{ij}.)$ 3. μ σ. 4. μ_{ij} σ_{ij}. 5.

: 1 2

: 1 2 L1 L1 .

[11]:
```
#
#    compute_distances_one_loop

dists_one = classifier.compute_distances_one_loop(X_test)

#
#                Frobenius
#      Frobenius
#

difference = np.linalg.norm(dists - dists_one, ord='fro')
print('One loop difference was: %f' % (difference, ))
if difference < 0.001:
    print('Good! The distance matrices are the same')
else:
    print('Uh-oh! The distance matrices are different')
```

```
One loop difference was: 0.000000
Good! The distance matrices are the same
```

[12]:
```
#   compute_distances_no_loops
dists_two = classifier.compute_distances_no_loops(X_test)

#
difference = np.linalg.norm(dists - dists_two, ord='fro')
print('No loop difference was: %f' % (difference, ))
if difference < 0.001:
    print('Good! The distance matrices are the same')
else:
    print('Uh-oh! The distance matrices are different')
```

```
No loop difference was: 0.000000
Good! The distance matrices are the same
```

[13]:
```
#
def time_function(f, *args):
    """
    Call a function f with args and return the time (in seconds) that it took␣
 ↪to execute.
    """
    import time
    tic = time.time()
    f(*args)
    toc = time.time()
    return toc - tic

two_loop_time = time_function(classifier.compute_distances_two_loops, X_test)
print('Two loop version took %f seconds' % two_loop_time)

one_loop_time = time_function(classifier.compute_distances_one_loop, X_test)
print('One loop version took %f seconds' % one_loop_time)

no_loop_time = time_function(classifier.compute_distances_no_loops, X_test)
print('No loop version took %f seconds' % no_loop_time)

#

#
#
```

```
Two loop version took 34.205247 seconds
One loop version took 53.223486 seconds
No loop version took 0.784877 seconds
```

1.0.1

kNN k = 5

[14]:
```
num_folds = 5
k_choices = [1, 3, 5, 8, 10, 12, 15, 20, 50, 100]

X_train_folds = []
y_train_folds = []
################################################################################
#     :
#          X_train_folds y_train_folds    num_folds
#  y_train_folds [i] X_train_folds [i]
#    numpy array_split
################################################################################
# *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****

X_train_folds = np.array_split(X_train, num_folds)
```

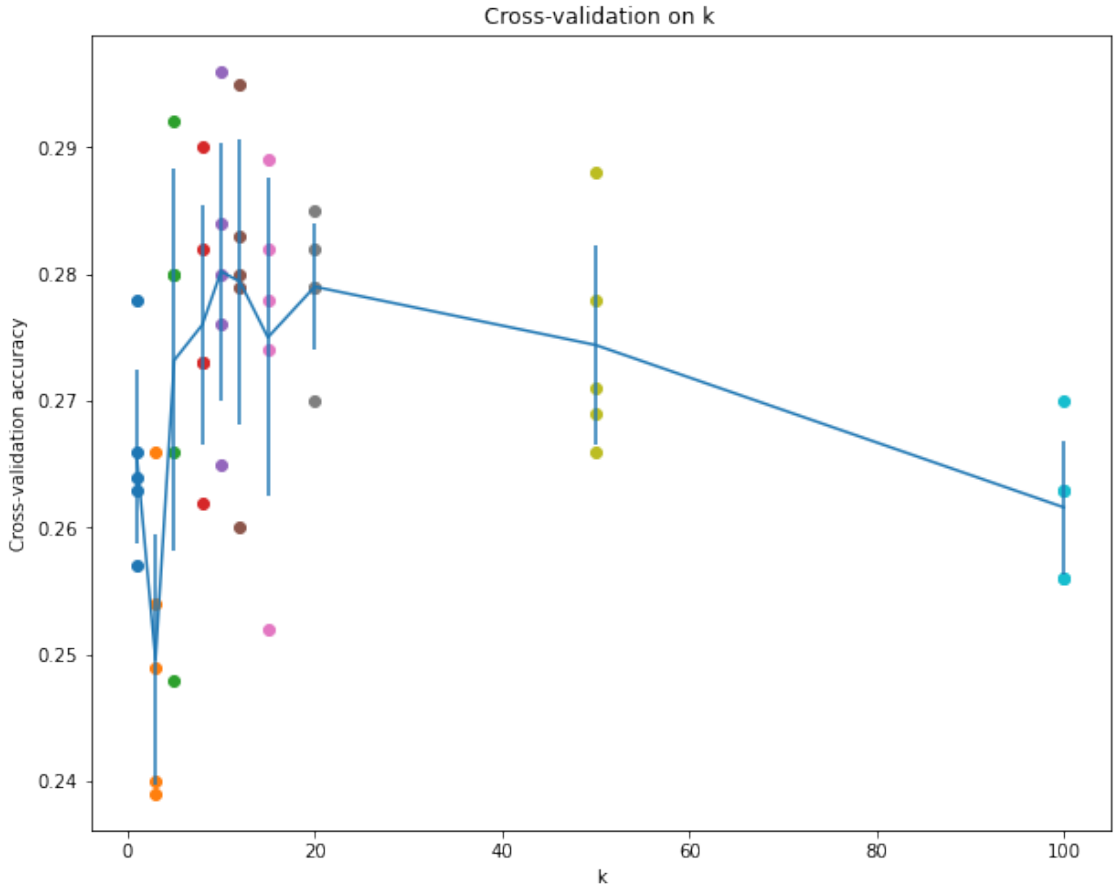
```
y_train_folds = np.array_split(y_train, num_folds)

# *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****

# A dictionary holding the accuracies for different values of k that we find␣
 ↪when running cross-validation.
#           k
#     k_to_accuracies[k]   num_folds    k
k_to_accuracies = {}


################################################################################
#     :
#  k     k
#     k  k-    num_folds
#
#        k_to_accuracies[k]
################################################################################
# *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****

#
#
#   1000        5  4    1
#      4    1      5
#

for k in k_choices:
    k_to_accuracies[k] = []
    for i in range(num_folds):
        # prepare training data for the current fold
        X_train_fold = np.concatenate([ fold for j, fold in␣
 ↪enumerate(X_train_folds) if i != j ])
        y_train_fold = np.concatenate([ fold for j, fold in␣
 ↪enumerate(y_train_folds) if i != j ])

        # use of k-nearest-neighbor algorithm
        classifier.train(X_train_fold, y_train_fold)
        y_pred_fold = classifier.predict(X_train_folds[i], k=k, num_loops=0)

        # Compute the fraction of correctly predicted examples
        num_correct = np.sum(y_pred_fold == y_train_folds[i])
        accuracy = float(num_correct) / X_train_folds[i].shape[0]
        k_to_accuracies[k].append(accuracy)


# *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
```

```
#
for k in sorted(k_to_accuracies):
    for accuracy in k_to_accuracies[k]:
        print('k = %d, accuracy = %f' % (k, accuracy))
```

```
k = 1, accuracy = 0.263000
k = 1, accuracy = 0.257000
k = 1, accuracy = 0.264000
k = 1, accuracy = 0.278000
k = 1, accuracy = 0.266000
k = 3, accuracy = 0.239000
k = 3, accuracy = 0.249000
k = 3, accuracy = 0.240000
k = 3, accuracy = 0.266000
k = 3, accuracy = 0.254000
k = 5, accuracy = 0.248000
k = 5, accuracy = 0.266000
k = 5, accuracy = 0.280000
k = 5, accuracy = 0.292000
k = 5, accuracy = 0.280000
k = 8, accuracy = 0.262000
k = 8, accuracy = 0.282000
k = 8, accuracy = 0.273000
k = 8, accuracy = 0.290000
k = 8, accuracy = 0.273000
k = 10, accuracy = 0.265000
k = 10, accuracy = 0.296000
k = 10, accuracy = 0.276000
k = 10, accuracy = 0.284000
k = 10, accuracy = 0.280000
k = 12, accuracy = 0.260000
k = 12, accuracy = 0.295000
k = 12, accuracy = 0.279000
k = 12, accuracy = 0.283000
k = 12, accuracy = 0.280000
k = 15, accuracy = 0.252000
k = 15, accuracy = 0.289000
k = 15, accuracy = 0.278000
k = 15, accuracy = 0.282000
k = 15, accuracy = 0.274000
k = 20, accuracy = 0.270000
k = 20, accuracy = 0.279000
k = 20, accuracy = 0.279000
k = 20, accuracy = 0.282000
k = 20, accuracy = 0.285000
k = 50, accuracy = 0.271000
k = 50, accuracy = 0.288000
```

```
k = 50, accuracy = 0.278000
k = 50, accuracy = 0.269000
k = 50, accuracy = 0.266000
k = 100, accuracy = 0.256000
k = 100, accuracy = 0.270000
k = 100, accuracy = 0.263000
k = 100, accuracy = 0.256000
k = 100, accuracy = 0.263000
```

[15]:
```
#
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.show()
```



[16]:
```
#        k
#                28

best_k = k_choices[accuracies_mean.argmax()]

classifier = KNearestNeighbor()
classifier.train(X_train, y_train)
y_test_pred = classifier.predict(X_test, k=best_k)

# Compute and display the accuracy
num_correct = np.sum(y_test_pred == y_test)
accuracy = float(num_correct) / num_test
print('Got %d / %d correct => accuracy: %f' % (num_correct, num_test, accuracy))
```

```
Got 141 / 500 correct => accuracy: 0.282000
```

3

k-NN k

1. k-NN

2. 1-NN 5-NN
3. 1-NN 5-NN
4. k-NN
5.

: 4

:

1. KNN
2.
4.k-NN

1.0.2 Data for leaderboard

X leaderborad

---

[17]:
```
# leaderboard
X = np.load("./input/X_3072.npy")
################################################################################
#     :
#    classifer
#    best_k
#                 classifer   best_k.
################################################################################
# *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
best_k = k_choices[accuracies_mean.argmax()]
# print(best_k)
classifer_leaderboard = KNearestNeighbor()
classifer_leaderboard.train(X_train, y_train)
preds = classifer_leaderboard.predict(X, k=best_k)
```

submit leaderboard phase1 leaderboard

[18]:
```
import os
#
def output_file(preds, phase_id=1):
    path=os.getcwd()
    if not os.path.exists(path + '/output/phase_{}'.format(phase_id)):
        os.mkdir(path + '/output/phase_{}'.format(phase_id))
    path=path + '/output/phase_{}/prediction.npy'.format(phase_id)
    np.save(path,preds)
def zip_fun(phase_id=1):
    path=os.getcwd()
    output_path = path + '/output'
    files = os.listdir(output_path)
    for _file in files:
```

```
        if _file.find('zip') != -1:
            os.remove(output_path + '/' + _file)
    newpath=path+'/output/phase_{}'.format(phase_id)
    os.chdir(newpath)
    cmd = 'zip ../prediction_phase_{}.zip prediction.npy'.format(phase_id)
    os.system(cmd)
    os.chdir(path)
output_file(preds)
zip_fun()
```

[ ]: