features

October 9, 2023

1

notebook

[1]:
```
import random
import numpy as np
from daseCV.data_utils import load_CIFAR10
import matplotlib.pyplot as plt


%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'

# for auto-reloading extenrnal modules
# see http://stackoverflow.com/questions/1907993/
 ↪autoreload-of-modules-in-ipython
%load_ext autoreload
%autoreload 2
```

1.1

CIFAR-10

[2]:
```
from daseCV.features import color_histogram_hsv, hog_feature

def get_CIFAR10_data(num_training=49000, num_validation=1000, num_test=1000):
    # Load the raw CIFAR-10 data
    cifar10_dir = 'daseCV/datasets/cifar-10-batches-py'

    # Cleaning up variables to prevent loading data multiple times (which may␣
 ↪cause memory issue)
    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)

    # Subsample the data
    mask = list(range(num_training, num_training + num_validation))
    X_val = X_train[mask]
    y_val = y_train[mask]
    mask = list(range(num_training))
    X_train = X_train[mask]
    y_train = y_train[mask]
    mask = list(range(num_test))
    X_test = X_test[mask]
    y_test = y_test[mask]

    return X_train, y_train, X_val, y_val, X_test, y_test

X_train, y_train, X_val, y_val, X_test, y_test = get_CIFAR10_data()
```

1.2

Histogram of oriented gradient HOG HSV

HOG

hog_feature color_histogram_hsv extract_features

[3]:
```
from daseCV.features import *

num_color_bins = 10 # Number of bins in the color histogram
feature_fns = [hog_feature, lambda img: color_histogram_hsv(img,␣
 ↪nbin=num_color_bins)]
X_train_feats = extract_features(X_train, feature_fns, verbose=True)
X_val_feats = extract_features(X_val, feature_fns)
X_test_feats = extract_features(X_test, feature_fns)

# Preprocessing: Subtract the mean feature
mean_feat = np.mean(X_train_feats, axis=0, keepdims=True)
X_train_feats -= mean_feat
X_val_feats -= mean_feat
X_test_feats -= mean_feat

# Preprocessing: Divide by standard deviation. This ensures that each feature
# has roughly the same scale.
```

```
std_feat = np.std(X_train_feats, axis=0, keepdims=True)
X_train_feats /= std_feat
X_val_feats /= std_feat
X_test_feats /= std_feat

# Preprocessing: Add a bias dimension
X_train_feats = np.hstack([X_train_feats, np.ones((X_train_feats.shape[0], 1))])
X_val_feats = np.hstack([X_val_feats, np.ones((X_val_feats.shape[0], 1))])
X_test_feats = np.hstack([X_test_feats, np.ones((X_test_feats.shape[0], 1))])
```

```
Done extracting features for 1000 / 49000 images
Done extracting features for 2000 / 49000 images
Done extracting features for 3000 / 49000 images
Done extracting features for 4000 / 49000 images
Done extracting features for 5000 / 49000 images
Done extracting features for 6000 / 49000 images
Done extracting features for 7000 / 49000 images
Done extracting features for 8000 / 49000 images
Done extracting features for 9000 / 49000 images
Done extracting features for 10000 / 49000 images
Done extracting features for 11000 / 49000 images
Done extracting features for 12000 / 49000 images
Done extracting features for 13000 / 49000 images
Done extracting features for 14000 / 49000 images
Done extracting features for 15000 / 49000 images
Done extracting features for 16000 / 49000 images
Done extracting features for 17000 / 49000 images
Done extracting features for 18000 / 49000 images
Done extracting features for 19000 / 49000 images
Done extracting features for 20000 / 49000 images
Done extracting features for 21000 / 49000 images
Done extracting features for 22000 / 49000 images
Done extracting features for 23000 / 49000 images
Done extracting features for 24000 / 49000 images
Done extracting features for 25000 / 49000 images
Done extracting features for 26000 / 49000 images
Done extracting features for 27000 / 49000 images
Done extracting features for 28000 / 49000 images
Done extracting features for 29000 / 49000 images
Done extracting features for 30000 / 49000 images
Done extracting features for 31000 / 49000 images
Done extracting features for 32000 / 49000 images
Done extracting features for 33000 / 49000 images
Done extracting features for 34000 / 49000 images
Done extracting features for 35000 / 49000 images
Done extracting features for 36000 / 49000 images
Done extracting features for 37000 / 49000 images
```

```
Done extracting features for 38000 / 49000 images
Done extracting features for 39000 / 49000 images
Done extracting features for 40000 / 49000 images
Done extracting features for 41000 / 49000 images
Done extracting features for 42000 / 49000 images
Done extracting features for 43000 / 49000 images
Done extracting features for 44000 / 49000 images
Done extracting features for 45000 / 49000 images
Done extracting features for 46000 / 49000 images
Done extracting features for 47000 / 49000 images
Done extracting features for 48000 / 49000 images
Done extracting features for 49000 / 49000 images
```

1.3 SVM

SVM SVM

[4]:
```
#

from daseCV.classifiers.linear_classifier import LinearSVM

learning_rates = [1e-9, 1e-8, 1e-7]
regularization_strengths = [5e4, 5e5, 5e6]

results = {}
best_val = -1
best_svm = None

################################################################################
#    :
#
#    SVM
#        best_svm
#           bins
#          0.44
################################################################################
# *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
iters = 2000

for learn_rate in learning_rates:
    for regularization_strength in regularization_strengths:
        svm = LinearSVM()
        svm.train(X_train_feats, y_train, learning_rate=learn_rate,␣
 ↪reg=regularization_strength, num_iters=iters)

        y_train_pred = svm.predict(X_train_feats)
        accuracy_train = np.mean(y_train == y_train_pred)
        y_val_pred = svm.predict(X_val_feats)
```

```
        accuracy_val = np.mean(y_val == y_val_pred)

        results[(learn_rate, regularization_strength)] = (accuracy_train,␣
 ↪accuracy_val)

        if best_val < accuracy_val:
            best_val = accuracy_val
            best_svm = svm

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

# Print out results.
for lr, reg in sorted(results):
    train_accuracy, val_accuracy = results[(lr, reg)]
    print('lr %e reg %e train accuracy: %f val accuracy: %f' % (
                lr, reg, train_accuracy, val_accuracy))

print('best validation accuracy achieved during cross-validation: %f' %␣
 ↪best_val)
```

```
lr 1.000000e-09 reg 5.000000e+04 train accuracy: 0.100367 val accuracy: 0.110000
lr 1.000000e-09 reg 5.000000e+05 train accuracy: 0.085469 val accuracy: 0.068000
lr 1.000000e-09 reg 5.000000e+06 train accuracy: 0.357878 val accuracy: 0.359000
lr 1.000000e-08 reg 5.000000e+04 train accuracy: 0.108571 val accuracy: 0.109000
lr 1.000000e-08 reg 5.000000e+05 train accuracy: 0.414245 val accuracy: 0.417000
lr 1.000000e-08 reg 5.000000e+06 train accuracy: 0.413980 val accuracy: 0.408000
lr 1.000000e-07 reg 5.000000e+04 train accuracy: 0.414082 val accuracy: 0.417000
lr 1.000000e-07 reg 5.000000e+05 train accuracy: 0.404490 val accuracy: 0.401000
lr 1.000000e-07 reg 5.000000e+06 train accuracy: 0.376653 val accuracy: 0.395000
best validation accuracy achieved during cross-validation: 0.417000
```

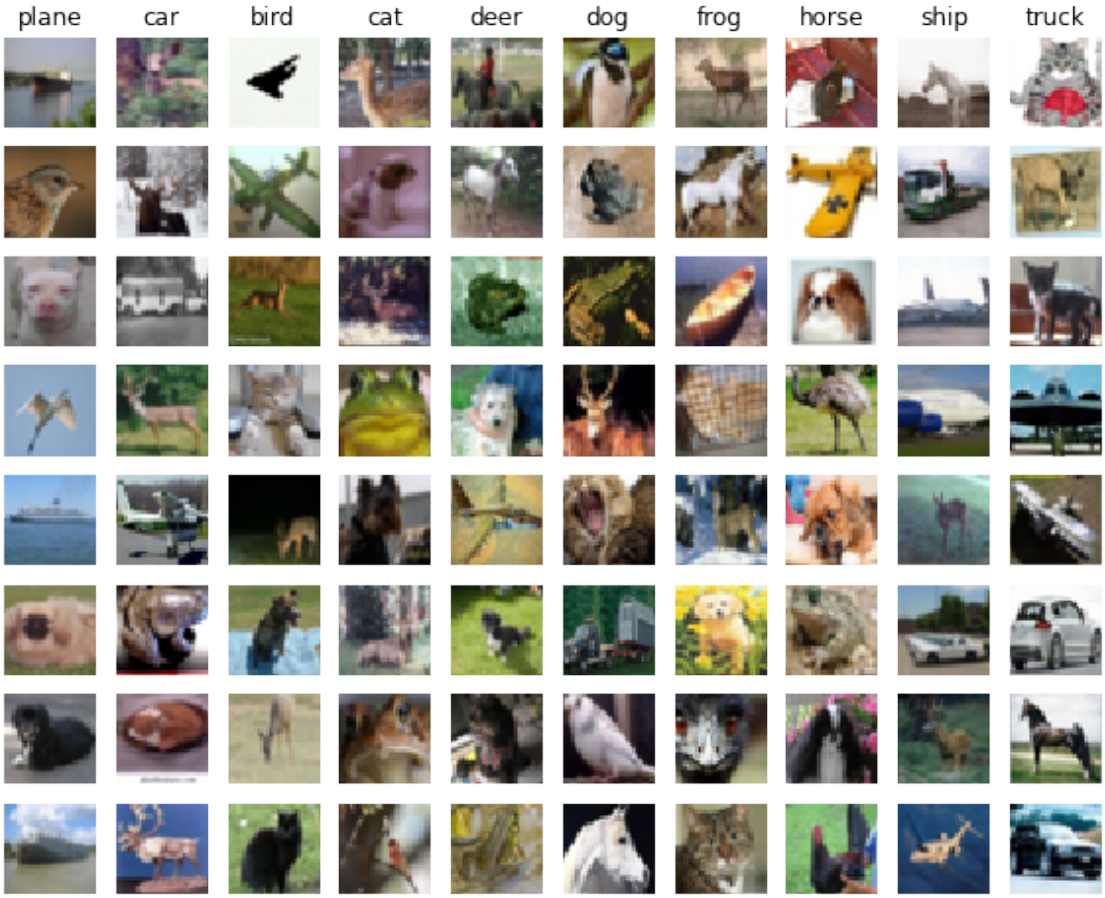
[5]:
```
# Evaluate your trained SVM on the test set
y_test_pred = best_svm.predict(X_test_feats)
test_accuracy = np.mean(y_test == y_test_pred)
print(test_accuracy)
```

```
0.412
```

[6]:
```
#
#
#          " plane"     " plane"

examples_per_class = 8
classes = ['plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse',␣
 ↪'ship', 'truck']
for cls, cls_name in enumerate(classes):
    idxs = np.where((y_test != cls) & (y_test_pred == cls))[0]
```

```
    idxs = np.random.choice(idxs, examples_per_class, replace=False)
    for i, idx in enumerate(idxs):
        plt.subplot(examples_per_class, len(classes), i * len(classes) + cls +␣
↪1)
        plt.imshow(X_test[idx].astype('uint8'))
        plt.axis('off')
        if i == 0:
            plt.title(cls_name)
plt.show()
```



1:

:

1.4

55 60

[7]:
```
# Preprocessing: Remove the bias dimension
# Make sure to run this cell only ONCE
print(X_train_feats.shape)
X_train_feats = X_train_feats[:, :-1]
X_val_feats = X_val_feats[:, :-1]
X_test_feats = X_test_feats[:, :-1]

print(X_train_feats.shape)
```

```
(49000, 155)
(49000, 154)
```

[8]:
```
from daseCV.classifiers.neural_net import TwoLayerNet

input_dim = X_train_feats.shape[1]
hidden_dim = 500
num_classes = 10
best_acc = 0.0

net = TwoLayerNet(input_dim, hidden_dim, num_classes)
best_net = None

################################################################################
# TODO:
#
#      best_net
################################################################################
# *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****

iters_num = 2000
learning_rates = [5e-2,6e-2,7e-2,8e-2,9e-2] ##
regularization_strengths = [3e-3,3.5e-3,5e-3,5.5e-3,7e-3]

for lr in learning_rates:
    for reg in regularization_strengths:
        stats = net.train(X_train_feats, y_train, X_val_feats, y_val,
                          num_iters=iters_num,batch_size=200,
                          learning_rate=lr,learning_rate_decay=0.95,
                          reg=reg, verbose=False)
        val_acc = (net.predict(X_val_feats) == y_val).mean()
        if val_acc > best_acc:
            best_acc = val_acc
            best_net = net
        print ("lr ",lr, "reg ", reg, "val accuracy:", val_acc)
print ("best validation accuracyachieved during cross-validation: ", best_acc)

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

```
lr  0.05 reg  0.003 val accuracy: 0.506
lr  0.05 reg  0.0035 val accuracy: 0.533
lr  0.05 reg  0.005 val accuracy: 0.551
lr  0.05 reg  0.0055 val accuracy: 0.562
lr  0.05 reg  0.007 val accuracy: 0.571
lr  0.06 reg  0.003 val accuracy: 0.581
lr  0.06 reg  0.0035 val accuracy: 0.595
lr  0.06 reg  0.005 val accuracy: 0.607
lr  0.06 reg  0.0055 val accuracy: 0.598
lr  0.06 reg  0.007 val accuracy: 0.599
lr  0.07 reg  0.003 val accuracy: 0.608
lr  0.07 reg  0.0035 val accuracy: 0.608
lr  0.07 reg  0.005 val accuracy: 0.609
lr  0.07 reg  0.0055 val accuracy: 0.604
lr  0.07 reg  0.007 val accuracy: 0.596
lr  0.08 reg  0.003 val accuracy: 0.607
lr  0.08 reg  0.0035 val accuracy: 0.604
lr  0.08 reg  0.005 val accuracy: 0.607
lr  0.08 reg  0.0055 val accuracy: 0.606
lr  0.08 reg  0.007 val accuracy: 0.614
lr  0.09 reg  0.003 val accuracy: 0.608
lr  0.09 reg  0.0035 val accuracy: 0.599
lr  0.09 reg  0.005 val accuracy: 0.61
lr  0.09 reg  0.0055 val accuracy: 0.61
lr  0.09 reg  0.007 val accuracy: 0.602
best validation accuracyachieved during cross-validation:  0.614
```

[9]:
```
#               55

test_acc = (best_net.predict(X_test_feats) == y_test).mean()
print(test_acc)
```

```
0.59
```