knn

January 18, 2024

1 Assignment 1: KNN

For this part of assignment, you are tasked to implement KNN algorithm and test it on the a subset of CIFAR10 dataset.

You sould run the whole notebook and answer the question in the notebook.

TO SUBMIT: PDF of this notebook with all the required outputs and answers.

[2]:
```
# Import Packages
import numpy as np
import matplotlib.pyplot as plt
```

1.1 Prepare Dataset

Since CIFAR10 is a relative large dataset, and KNN is quite time-consuming method, we only a small sub-set of CIFAR10 for KNN part

[3]:
```
from utils.data_processing import get_cifar10_data

# Use a subset of CIFAR10 for KNN assignments
dataset = get_cifar10_data(subset_train=5000, subset_val=250, subset_test=500)

print(dataset.keys())
print("Training Set Data  Shape: ", dataset["x_train"].shape)
print("Training Set Label Shape: ", dataset["y_train"].shape)
```

```
dict_keys(['x_train', 'y_train', 'x_val', 'y_val', 'x_test', 'y_test'])
Training Set Data  Shape:  (5000, 3072)
Training Set Label Shape:  (5000,)
```

1.2 Implementation (60%)

You need to implement the KNN method in `algorithms/knn.py`. You need to fill in the prediction function(since the training of KNN is just remembering the training set).

For KNN implementation, you are tasked to implement two version of it.

- Two Loop Version: use one loop to iterate through training samples and one loop to iterate through test samples

- One Loop Version: use one loop to iterate through test samples and use broadcast (https://numpy.org/doc/stable/user/basics.broadcasting.html) feature of numpy to calculate all the distance at once

Note: It is possible to build a Fully Vectorized Version without explicit for loop to calculate the distance, but you do not have to do it in this assignment. You could use the fully vectorized version to replace the loop versions as well.

For distance function, in this assignment, we use Eucliean distance between samples.

[4]:
```
from algorithms import KNN

knn = KNN(num_class=10)
knn.train(
    x_train=dataset["x_train"],
    y_train=dataset["y_train"],
    k=5,
)
```

1.2.1 Compare the time consumption of different method

In this section, you will test your different implementation of KNN method, and compare their speed.

[5]:
```
from utils.evaluation import get_classification_accuracy
```

Two Loop Version:

[6]:
```
import time

c_t = time.time()
prediction = knn.predict(dataset["x_test"], loop_count=2)
print("Two Loop Prediction Time:", time.time() - c_t)

test_acc = get_classification_accuracy(prediction, dataset["y_test"])
print("Test Accuracy:", test_acc)
```

```
Two Loop Prediction Time: 39.563470125198364
Test Accuracy: 0.278
```

One Loop Version

[7]:
```
import time

c_t = time.time()
prediction = knn.predict(dataset["x_test"], loop_count=1)
print("One Loop Prediction Time:", time.time() - c_t)

test_acc = get_classification_accuracy(prediction, dataset["y_test"])
print("Test Accuracy:", test_acc)
```

```
One Loop Prediction Time: 36.486475706100464
Test Accuracy: 0.278
```

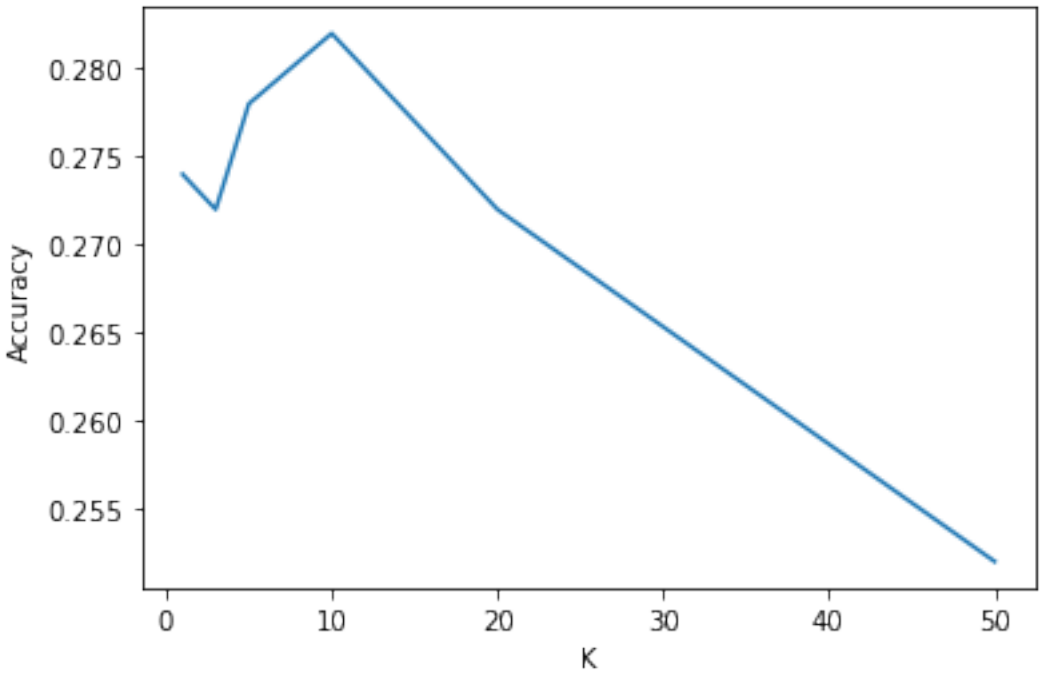
Your different implementation should output the exact same result

1.3 Test different Hyper-parameter (20%)

For KNN, there is only one hyper-parameter of the algorithm: How many nearest neighbour to use(**K**).

Here, you are provided the code to test different k for the same dataset.

[8]:
```
accuracies = []

k_candidates = [1, 3, 5, 10, 20, 50]
for k_cand in k_candidates:
    prediction = knn.predict(x_test=dataset["x_test"], k=k_cand)
    acc = get_classification_accuracy(prediction, dataset["y_test"])
    accuracies.append(acc)
plt.ylabel("Accuracy")
plt.xlabel("K")
plt.plot(k_candidates, accuracies)
plt.show()
```



1.3.1 Inline Question 1:

Please describe the output result you get, and provide some explanation as well.

1.3.2 Your Answer:

This output result makes sense and is in line with what the professor talked about in lecture 2 about KNN classifiers and their effect on the output. This is known as the bias-variance tradeoff in machine learning, as with a low bias but high variance, we get a more flexible model that may be too sensitive to the outliers, until we reach an optimal value of k, which here is just above 10 reaching an accuracy of just over 0.28. Afterwards, the model becomes oversimplified with too much bias and low variance.

1.4 Try different feature representation (19%)

Since machine learning method rely heavily on the feature extraction, you will see how different feature representation affect the performance of the algorithm in this section.

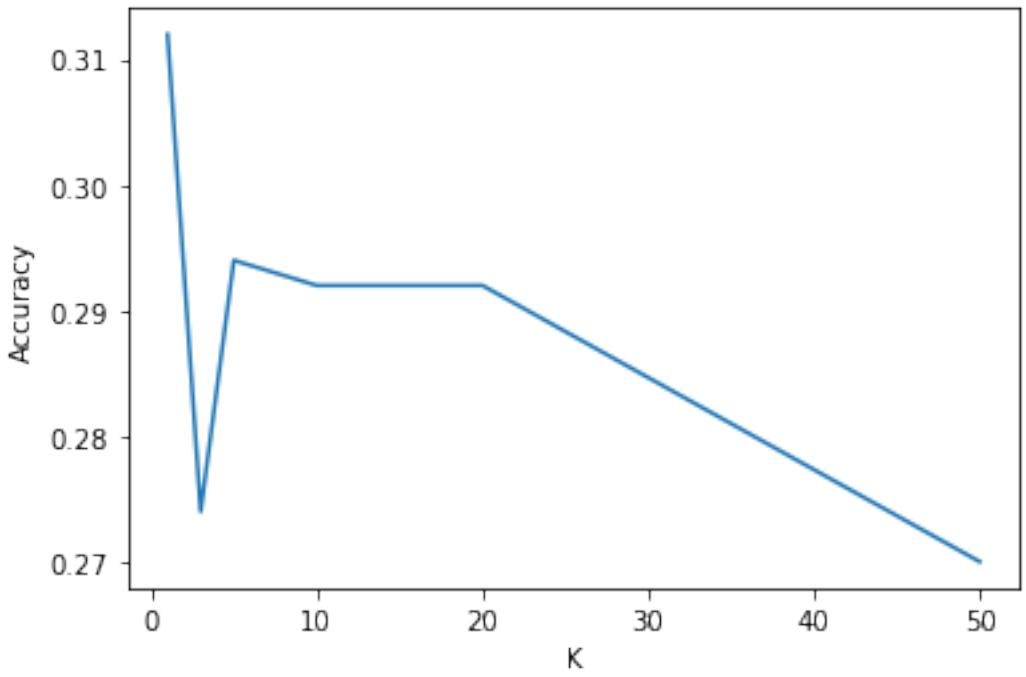
You are provided the code about using **HOG** descriptor to represent samples in the notebook.

[9]:
```
from utils.data_processing import get_cifar10_data
from utils.data_processing import HOG_preprocess
from functools import partial

# Delete previous dataset to save memory
del dataset
del knn

# Use a subset of CIFAR10 for KNN assignments
hog_p_func = partial(
    HOG_preprocess,
    orientations=9,
    pixels_per_cell=(4, 4),
    cells_per_block=(1, 1),
    visualize=False,
    multichannel=True,
)
dataset = get_cifar10_data(
    feature_process=hog_p_func, subset_train=5000, subset_val=250,␣
 ↪subset_test=500
)
```

```
Start Processing
Processing Time: 9.997384309768677
```

[10]:
```
knn = KNN(num_class=10)
knn.train(
    x_train=dataset["x_train"],
    y_train=dataset["y_train"],
    k=5,
)
accuracies = []
```

```
k_candidates = [1, 3, 5, 10, 20, 50]
for k_cand in k_candidates:
    prediction = knn.predict(x_test=dataset["x_test"], k=k_cand)
    acc = get_classification_accuracy(prediction, dataset["y_test"])
    accuracies.append(acc)

plt.ylabel("Accuracy")
plt.xlabel("K")
plt.plot(k_candidates, accuracies)
plt.show()
```



1.4.1 Inline Question 2:

Please describe the output result you get, compare with the result you get in the previous section, and provide some explanation as well.

1.4.2 Your Answer:

The use of HOG descriptors allows for the significant gains in terms of efficiency. We can see this firsthand as our processing times went from 30-40 seconds all the way down to under 10 seconds. This is because the HOG descriptors focus on gradients and edge orientations, allowing for simplified and imformative image representations. This also leads to a lower optimal value for k when finding KNN, as we can see on the plot. It is also good to note that the peak accuracy is higher, although that may not always hold true.

1.5 Survey (1%)

1.5.1 Question:

How many hours did you spend on assignment 1?

1.5.2 Your Answer:

4 hours