Image Classification with the K-Nearest-Neighbors Algorithm

Jacky Dam
University of California San Diego
San Diego, California
jdam@ucsd.edu

Abstract

In this assignment, I utilized python to create an algorithm based on k-nearest-neighbors to classify data sets containing images of numbers and the correct label for each number. What makes this algorithm special is the fact that it classifies images of numbers from a test set that the algorithm has never seen before by learning distinct characteristics from the training set. After implementing this algorithm in python, I found that the algorithm performed adequately with an error of about 9% evaluated on the test set. Additionally, I found that after performing dimensionality reduction using projections which drastically reduced runtime, the algorithm performed with an error of about 29% evaluated on the test set.

1 Data Set:

http://cseweb.ucsd.edu/classes/wi21/cse151A-a/homeworks.html

2 Evaluation:

Classifier:	k	Training Error:	Validation Error:
C_1	1	0	0.082
C_5	5	0.055	0.099
C_9	9	0.068	0.105
C_{15}	15	0.093	0.106

Test Error of C_1 : 0.094

3 Evaluation w/ Projection:

Classifier:	k	Training Error:	Validation Error:
C_1	1	0	0.32
C_5	5	0.1905	0.296
C_9	9	0.2285	0.29
C_{15}	15	0.2565	0.291

Test Error of C_9 : 0.29

4 Summary:

After implementing the algorithm in part 1, I found that C_1 performed the best on the validation set with an error of about 8%. I then utilized this classifier to evaluate the algorithm using the test set and found that with the test set, the algorithm performed with an error of about 9%.

In part 2, I utilized the same algorithm but utilized dimenionality reduction by projecting each image onto some projection matrix. This drastically reduced the runtime of the code by more than half, but I found that the accuracy of the algorithm was greatly reduced as the algorithm had an error of about 29% on the test set. Although the accuracy of the algorithm was drastically reduced by dimenionality reduction, it is evident that there is a trade off between computing power and performance.

```python
import numpy as np

def extract_data(file):
    data = []
    with open(file) as f:
        for line in f.readlines():
            data.append([int(i) for i in line.strip().split(" ")])

    return data

def extract_double_data(file):
    data = []
    with open(file) as f:
        for line in f.readlines():
            data.append([float(i) for i in line.strip().split(" ")])

    return data

def find_distance(vector1, vector2):
    vector = np.square(np.subtract(vector1, vector2))
    distance = np.sqrt(np.sum(vector))

    return distance

def find_neighbors(train_set, input_vector, k):
    distance_set = []
    neighbors_set = []
    for train_vector in train_set:
        distance = find_distance(train_vector[:784], input_vector)
        distance_set.append((train_vector, distance))
    distance_set.sort(key=lambda tup: tup[1])

    for i in range(k):
        neighbors_set.append(distance_set[i][0])

    return neighbors_set

train_set = extract_data("pa1train.txt")
val_set = extract_data("pa1validate.txt")
test_set = extract_data("pa1test.txt")
proj_matrix = extract_double_data("projection.txt")

# This chunk of code evaluates the error of the KNN algorithm (part 1).

output_labels = []
actual_labels = []
error_count = 0

for j in range(len(test_set)):
    neighbors = find_neighbors(train_set, test_set[j][:784], 1)
    output = [int(i[-1]) for i in neighbors]
    label = max(set(output), key=output.count)

    if label != test_set[j][-1]:
        error_count += 1

print(error_count/len(test_set))
```

```
58
59
60  # This chunk of code utilizes projections to evaluate the error of the KNN algorithm (part 2).
61
62  train_label = [i[-1] for i in train_set]
63  val_label = [i[-1] for i in val_set]
64  test_label = [i[-1] for i in test_set]
65
66  train_set = np.column_stack((np.matmul(np.dot([i[:784] for i in train_set], proj_matrix),
    np.transpose(proj_matrix))/np.linalg.norm(proj_matrix), train_label))
67  val_set = np.column_stack((np.matmul(np.dot([i[:784] for i in val_set], proj_matrix),
    np.transpose(proj_matrix))/np.linalg.norm(proj_matrix), val_label))
68  test_set = np.column_stack((np.matmul(np.dot([i[:784] for i in test_set], proj_matrix),
    np.transpose(proj_matrix))/np.linalg.norm(proj_matrix), test_label))
69
70  output_labels = []
71  actual_labels = []
72  error_count = 0
73
74  for j in range(len(test_set)):
75      neighbors = find_neighbors(train_set, test_set[j][:784], 9)
76      output = [int(i[-1]) for i in neighbors]
77      label = int(max(set(output), key=output.count))
78
79      if label != test_set[j][-1]:
80          error_count += 1
81
82  print(error_count/len(test_set))
83
```