HW3 Report

Gong Lixue

November 5, 2017

1 Neural Networks

The implementation of feedforward and backpropagation process is shown in code folder. At the end of training iterations, loss = 1.717e - 01, accuracy = 0.9400, which is not the optimal actually. And for testing, loss = 2.522e - 01, and accuracy = 0.9230.

2 K-Neareast Neighbor

(a) boundary figures is shown below:

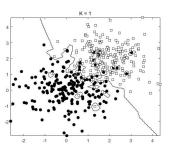


Figure 1: K = 1

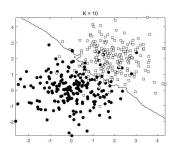


Figure 2: K = 10

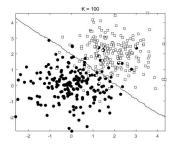


Figure 3: K = 100

43430	11877	26611	46380	20703
img_0.jpg	img_1.jpg	img_2.jpg	img_3.jpg	img_4.jpg
35526	66833	45424	45543	48616
img_5.jpg	img_6.jpg	img_7.jpg	img_8.jpg	img_9.jpg
33770	37843	43676	58400	80616
img_10.jpg	img_11.jpg	img_12.jpg	img_13.jpg	img_14.jpg
47637	78843	36764	31837	24001
img_15.jpg	img_16.jpg	img_17.jpg	img_18.jpg	img_19.jpg
66450	61523	55687	58751	16672
img_20.jpg	img_21.jpg	img_22.jpg	img_23.jpg	img_24.jpg
80185	05818	11642	63485	52067
img_25.jpg	img_26.jpg	img_27.jpg	img_28.jpg	img_29.jpg
18808	03062	40333	55156	86362
img_30.jpg	img_31.jpg	img_32.jpg	img_33.jpg	img_34.jpg
44383	75508	81333	68814	36845
img_35.jpg	img_36.jpg	img_37.jpg	img_38.jpg	img_39.jpg
00348	73752	51343	28264	40570
img_40.jpg	img_41.jpg	img_42.jpg	img_43.jpg	img_44.jpg
17402	02565	50486	81702	85865
img_45.jpg	img_46.jpg	img_47.jpg	img_48.jpg	img_49.jpg
00608	15422	31034	35107	41821
img_50.jpg	img_51.jpg	img_52.jpg	img_53.jpg	img_54.jpg
28412	34245	38318	23472	60633
img_55.jpg	img_56.jpg	img_57.jpg	img_58.jpg	img_59.jpg
12610	70531	84364	00187	04251
img_60.jpg	img_61.jpg	img_62.jpg	img_63.jpg	img_64.jpg
88324	08443	18008	38778	02282
img_65.jpg	img_66.jpg	img_67.jpg	img_68.jpg	img_69.jpg
22207	31802	45730	51553	34866

Figure 4: images fetched by python spider

- (b) One of methods is that choosing a proper K by Cross-Validation. Compute the validation error on validation set using different values for K. And pick the optimal K with the lowest validation error.
- (c) Firstly, I wrote a simple python script to fetch and save check code images from this website automatically. (See Figure 4)

And then, label these images by hand. I use 100 images of them. The raw labels are recored in file $./knn/hack_py/label_100img.txt$. Each row in this file means the actual codes that corresponding image represents.

Before recognizing a check code image, we should generate a .mat file which is used for training in knn. Run $gen_hack_data.m$ to generate a .mat file. Finally, we can test the algorithm to recognize a check code image(see $knn_exp.m$ Part2 and Figure 5).



Figure 5: show_image



Figure 6: result digits

3 Decision Tree and ID3

The decision tree and respective information gain is illustrated as the graph below:

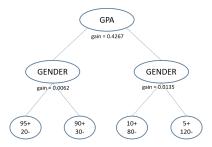


Figure 7: decision tree

4 K-Means Clustering

(a) The visualization of process with smallest SD:

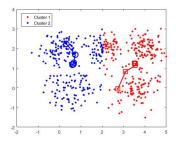


Figure 8: kmeans process with smallest ${\rm SD}$

The visualization of process with largest SD:

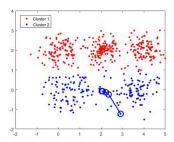


Figure 9: kmeans process with largest SD

- (b) We can run it from multiple starting points, and pick the solution with the smallest SD.
- (c) The visualization of centroid is illustrated below. And we can find that the cluster center can represent the patterns in dataset.



Figure 10: k=10



Figure 11: k=50



Figure 12: k=50

(d) This is the original image:



Figure 13: original image

And wen can obtain the compressed images after runing vq.m:



Figure 14: K = 8



Figure 15: K = 16



Figure 16: K = 32

We can observe that when we set K to 64, these is no obvious change comparing to original image. When K=64, each pixel can be represented with log(K) = 8 bits. So the data is not efficiently compressed actually.