MARCH 23, 2020

COSI126A _HW2

CHUYUE WU

INSTRUCTOR: DR. HONGFU LIU

SCHOOL: BRANDEIS UNIVERSITY

Course Name: Introduction to Data Mining

Problem 1 (30 points)

Compare your K-means clustering code with the public scikit-learn one in terms of objective function value and execution time on the following data sets.

- MNIST. http://yann.lecun.com/exdb/mnist/
- CIFAR-10. http://www.cs.toronto.edu/~kriz/cifar.html
- LFW. http://vis-www.cs.umass.edu/lfw/

The above three data sets are image data sets. You can use the pixel level features or other well extracted features with the true cluster number. Modify your code and try to beat the scikit-learn one in a fair setting.

(1)MNIST

For the MINIST dataset, I downloaded all the data the website provides and used some codes to read and transform data. The training images dataset has 60000 pieces of handwritten digits images, each of images is consisted with 28*28 pixels. After transformation, the training images dataset has been organized to a new dataset of 60000 rows and 784 columns.

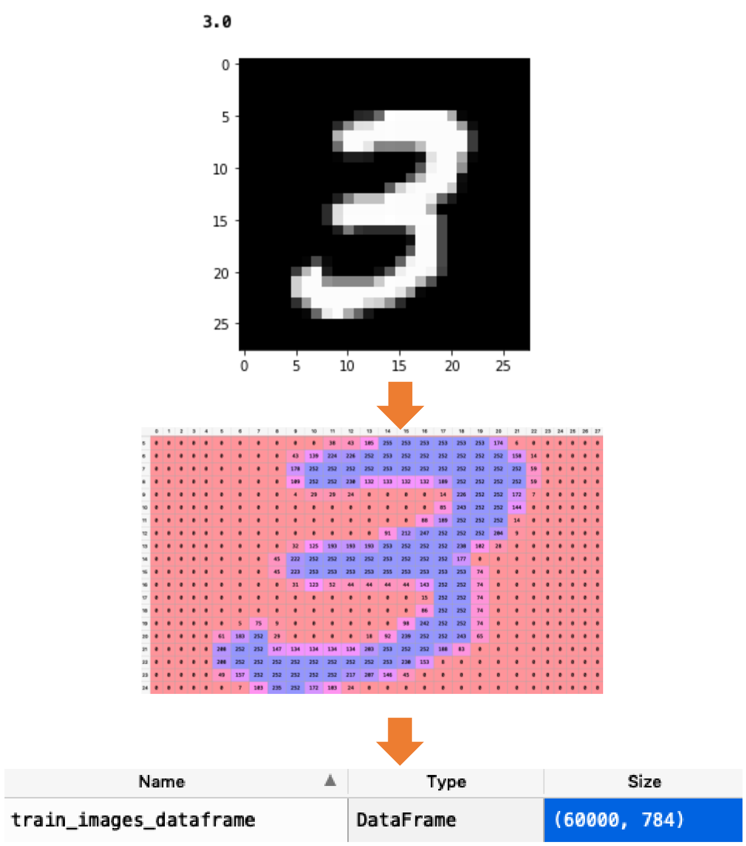

figure1: Transformation of MNNIST dataset

Since we want to do k-means clustering for this dataset, we could regard each row as a vector and put all the dataset into k-means function directly. Test dataset seems to have no use here. In addition, because scikit-learn kmeans has a default setting of initializing with k-means++ method, I also used my own kmeans++ kmeans method to run the model.
Because the dataset is too big, I selected 300 rows at first and run it locally. The results are shown below. It seems both the execution time and objective function value are very close. My own kmeans method even has a better accuracy.

k-means	execution time	objective function value	accuracy
scikit-learn-kmeans	0	6.8484e+08	0.606667
myown_kmeans	1	7.07757e+08	0.61

figure2: Methods comparation for 300 rows MNNIST dataset sample

Then I use ssh to connect to the remote computer, which has a better performance. I run 60000 rows data and the results are shown below. Although the execution time seems to have a little difference when I run 300 rows in my computer, there is a very obvious difference when it comes to big-size dataset. It shows that writing a good package is not an easy thing since there are many factors needing to be considered, including efficiency and applicability. Fortunately, objective function values and accuracy of two methods are still close. My own kmeans method is not very efficiency, but it is accurate at least.

k-means	execution time	objective function value	accuracy
scikit-learn-kmeans	169	1.52993e+11	0.59085
myown_kmeans	1713	1.53715e+11	0.598717

figure3: Methods comparation for 60000 rows MNNIST training dataset

(2) CIFAR-10

CIFAR dataset consists of 60000 32x32 color images in 10 classes. The logic of this dataset is same as the MNIST, the only different is that because the images are colorful, each of images has three channel (red, green, blue) values. After combine training datasets together and manipulating, each row has 32*32*3 = 3072 columns. We also ignore test dataset because it's useless for kmeans method.

Name	Type	Size
images_train	uint8	(50000, 3072)

figure4: Transformation results of CIFAR-10 training dataset

I select 300 rows firstly and run it in my own computer, the results are shown here. It's not very nice because the execution time of my kmeans seems too long. It's only 300 rows. Things must be worse when it moves to 50000 rows.

k-means	execution time	objective function value	accuracy
scikit-learn-kmeans	0	2.23648e+09	0.263333
myown_kmeans	4	2.2264e+09	0.28

figure5: Methods comparation for 300 rows CIFAR-10 dataset sample

k-means	execution time	objective function value	accuracy
scikit-learn-kmeans	453	3.9481e+11	0.22122
myown_kmeans	2679	3.94207e+11	0.22079

figure6: Methods comparation for 60000 rows CIFAR-10 training dataset

(3) LFW

For LFW dataset, because there are only 13233 images but 5749 people, if we set k equal to 5749, the accuracy of the results must be extremely low. Fortunately, I found scikit-learn package also provides sorted LFW data. By using fetch_lfw_people function, we get a dataset of 1140 rows and 1850 columns. Each row is extracted from an face images, and all of these face images only belong to 5 different people. In this way, we could set the k equal to 5. It's a better dataset which could be used by kmeans cluster method.

Name	Type	Size	Value
lfw_people_data	float32	(1140, 1850)	[[85.666664 81.666664 53. ... 110.666664 117.333336 181.66667 ...
lfw_people_target	int64	(1140,)	[2 3 1 ... 4 2 4]
lfw_people_target_names	str544	(5,)	ndarray object of numpy module

figure7: LFW cluster dataset obtained by fetch_lfw_people function

We run the models with this dataset, and find the results below. Because this dataset is relative smaller and the number of cluster is also less when compared to previous dataset, the execution time is not very long. But the gap of time between two methods is still very large. Ojective function value and accuracy are close. Accuracy can reach nearly 50% in this case, which is better than k-means methods for CIFAR-10 dataset. It might be explained by the fact that the well selected features of people face are easier recognized than just color features of different objects inCIFAR-10 dataset.

k-means	execution time	objective function value	accuracy
scikit-learn-kmeans	2	2.22631e+09	0.464912
myown_kmeans	16	2.23601e+09	0.475439

figure8: Methods comparation for 1140 rows LFW dataset

Problem 2 (25 points)

Let us re-use the above MNIST data set for classification. Here we focus on the none-preprocessing category, that means the pixel-level feature. Each image is represented by a 1 × 784 vector. Reimplement linear classifier (1-layer NN) (Test error rate: 12%), K-nearest- neighbors (Test error rate: 5%), SVM + Gaussian Kernel (Test error rate: 1.4%).
You can use scikit-learn codes. Report the parameter settings of the above classifiers and how you get them.

(1)1-layer NN
I manipulated data as I did in problem1 and scaled the images data. Then I used two-fold cross validation to train model and choose the best number of nodes for the hidden layer.

```
scores = cross_val_score(mlp, X_train, Y_train, cv=2, scoring='accuracy')
```

For parameter settings, since we need to use linear function, the activation should choose "identity", which returens f(x) = x. Solver 'sgd' refers to stochastic gradient descent. Alpha = 0.0001 is the default setting. I put all the parameters below.

```
MLPClassifier(activation='identity', alpha=0.0001, batch_size='auto',
          beta_1=0.9, beta_2=0.999, early_stopping=False, epsilon=1e-08,
          hidden_layer_sizes=(nodes,), learning_rate='constant',
          learning_rate_init=0.1, max_iter=10, momentum=0.9,
          n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
          random_state=1, shuffle=True, solver='sgd', tol=0.0001,
          validation_fraction=0.1, verbose=10, warm_start=False)
```

The results for loop shows here:

```
nn_execution_time: 1124
1 accuracy 0.33503333333333335
2 accuracy 0.60285
3 accuracy 0.7842666666666667
4 accuracy 0.8479666666666668
5 accuracy 0.8734
6 accuracy 0.8731833333333333
7 accuracy 0.8856333333333334
8 accuracy 0.88785
9 accuracy 0.8891166666666667
10 accuracy 0.8917333333333333
11 accuracy 0.897
12 accuracy 0.8852
13 accuracy 0.8857666666666666
14 accuracy 0.8927166666666666
15 accuracy 0.89225
16 accuracy 0.8894166666666666
17 accuracy 0.8763666666666666
18 accuracy 0.89165
19 accuracy 0.8802333333333334
20 accuracy 0.8840333333333333
21 accuracy 0.8744666666666667
22 accuracy 0.8874
23 accuracy 0.88475
24 accuracy 0.8655333333333333
25 accuracy 0.8764000000000001
26 accuracy 0.8760333333333333
27 accuracy 0.8884666666666667
28 accuracy 0.86485
29 accuracy 0.8901
30 accuracy 0.86385
31 accuracy 0.8739666666666667
32 accuracy 0.8722
33 accuracy 0.8806166666666667
34 accuracy 0.8748
35 accuracy 0.8623833333333333
36 accuracy 0.8426166666666667
37 accuracy 0.8598166666666667
38 accuracy 0.8687333333333334
39 accuracy 0.8784333333333334
40 accuracy 0.8686166666666666
41 accuracy 0.8748833333333333
42 accuracy 0.8355166666666667
43 accuracy 0.8706
44 accuracy 0.8673166666666667
45 accuracy 0.8015000000000001
46 accuracy 0.8617333333333334
47 accuracy 0.8600833333333333
48 accuracy 0.84765
49 accuracy 0.8696
50 accuracy 0.8567333333333333
51 accuracy 0.8481666666666667
52 accuracy 0.8553666666666666
53 accuracy 0.8783833333333333
54 accuracy 0.8640666666666666
55 accuracy 0.8353666666666667
56 accuracy 0.8635
57 accuracy 0.8486333333333334
58 accuracy 0.8644166666666666
59 accuracy 0.86695
60 accuracy 0.8675333333333333
61 accuracy 0.8546666666666667
62 accuracy 0.85495
63 accuracy 0.8745833333333333
64 accuracy 0.8638833333333333
65 accuracy 0.8710666666666667
66 accuracy 0.8654
67 accuracy 0.8627333333333334
68 accuracy 0.8581166666666666
69 accuracy 0.8735666666666666
70 accuracy 0.8537166666666667
71 accuracy 0.8614166666666667
72 accuracy 0.8305833333333333
73 accuracy 0.8141166666666667
74 accuracy 0.8463666666666667
75 accuracy 0.8458666666666668
76 accuracy 0.8384
77 accuracy 0.85305
78 accuracy 0.8487666666666667
79 accuracy 0.8523000000000001
80 accuracy 0.8530666666666666
81 accuracy 0.8753833333333334
82 accuracy 0.8597333333333333
83 accuracy 0.85245
84 accuracy 0.8643166666666666
85 accuracy 0.8276333333333333
86 accuracy 0.8494166666666667
87 accuracy 0.8584
88 accuracy 0.8539833333333333
89 accuracy 0.87965
90 accuracy 0.86895
91 accuracy 0.8466166666666667
92 accuracy 0.84745
93 accuracy 0.8583333333333334
94 accuracy 0.8433333333333333
95 accuracy 0.85855
96 accuracy 0.84785
97 accuracy 0.8549
98 accuracy 0.8592666666666666
99 accuracy 0.8497333333333333
100 accuracy 0.8440166666666666
```

figure9: Cross Validation accuracy for number of nodes from 1 to 100

According to the results, I decided to choose 11 nodes model, ran the model, and computed scores of training set and test set. The test set score is a slightly lower than training set score. The results are very closely to the 12% error rate on the question given and even better than 12%.

```
Training set score: 0.896017 Test set score: 0.891300
```

figure10: Training set score and test set score for 11 nodes 1-layer model

(2)K-nearest- neighbors

For KNN model, I also used two-fold cross validation to train model and choose the best k for the model. I tried k from 3 to 39 and skipped all the even numbers.

```
3 accuracy 0.9351
5 accuracy 0.9350666666666667
7 accuracy 0.9333333333333333
9 accuracy 0.9312
11 accuracy 0.9298666666666666
13 accuracy 0.9282833333333333
15 accuracy 0.9264166666666667
17 accuracy 0.9250166666666666
19 accuracy 0.9226666666666667
21 accuracy 0.9216166666666666
23 accuracy 0.9205666666666666
25 accuracy 0.9189333333333334
27 accuracy 0.9183833333333333
29 accuracy 0.9172
31 accuracy 0.9154
33 accuracy 0.9141333333333334
35 accuracy 0.9126833333333333
37 accuracy 0.9119166666666667
39 accuracy 0.9110666666666667
```

figure11: Cross Validation accuracy for number of k from 3 to 39(odd number)

From results above, k equals to 3 seems to be the best choose. So I used this model and computed final results of training set scores and testing set scores. The training set score are pretty good, and the testing set score is close to 95%, which is consistent with 5% error rate. So it’s also good.

```
Training set score: 0.972767 Test set score: 0.944100
```

figure12: Training set score and test set score for KNN of k = 3

(3)SVM + Gaussian Kernel

For svm classification, I set regularization parameter C to be 100, gamma to be 0.03, and kernel = ‘rbf’ which represents Gaussian Kernel. The results are shown here. This is the best one in these three classifications.

```
Training set score: 0.990847 Test set score: 0.985789
```

figure13: Training set score and test set score for SVM + Gaussian Kernel

Problem 3 (5 points)

Read this paper titled Do we Need Hundreds of Classifiers to Solve Real World Classification Problems?. What are the recommended classifiers for practical use?

According to this paper, the best results are achieved by the parallel random forest, which achieves 94.1% of the maximum accuracy. The SVM with Gaussian kernel is also good to use, and it achieves 92.3% of the maximum accuracy.
To be general, the random forest is clearly the best family of classifiers (3 out of 5 bests classifiers are RF), followed by SVM (4 classifiers in the top-10), neural networks and boosting ensembles (5 and 3 members in the top-20, respectively).
So my conclusion is that for practical use, random forest, SVM, neural networks and boosting ensembles are the recommended classifiers.

Problem 4 (40 points)

Kaggle project (https://www.kaggle.com/chrisfilo/urbansound8k). Provide a detailed report on the audio classification including feature extraction, dataset building, model selection, model update and result analysis.

Feature Extraction
The audio file can be manipulated by librosa package. At first, I select five files randomly to see their patterns. It is obviously that drilling sound is noisier while dog bark is more distinct. After exploring the patterns of wav file, I decided to use the mel_spectogram function to extract the spectogram data as a numpy array as other people posted online.

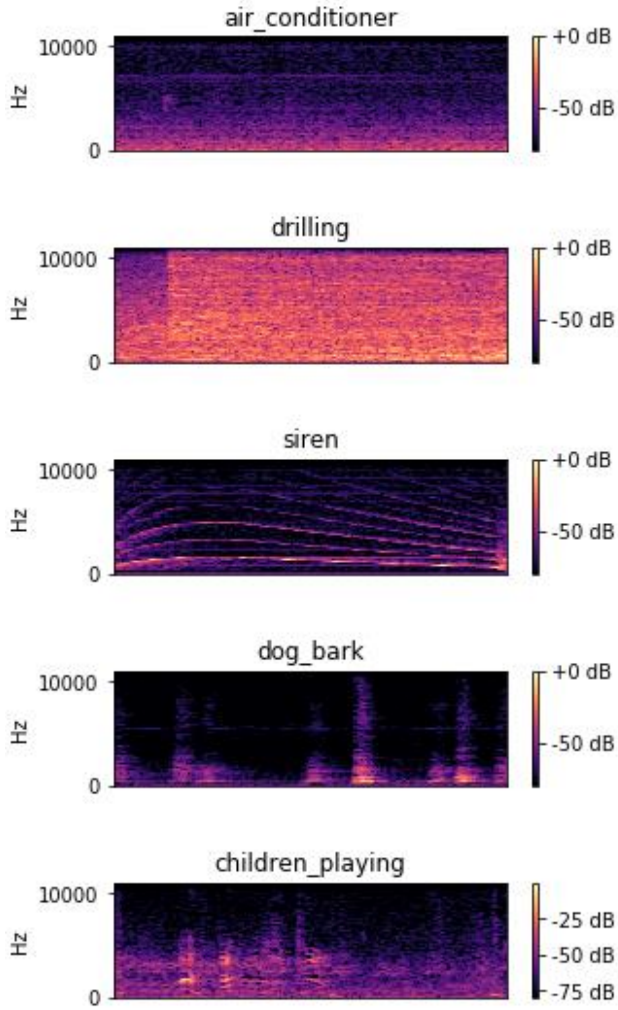

figure14: Select five files randomly and see their patterns

Dataset Building
Then, I used the same way to read all the 8732 files and applied mel_spectogram function to extract all the data. I also used to_categorical function to distribute data of different class into different category. I set test size to 0.25 to split the dataset.

Model Selection
As we know, CNN is a very classical way of sound classification as it is observed that similar type of sounds have similar spectrogram. A spectrogram is a visual representation of the spectrum of frequencies of sound or another signal as they vary with time. And thus we can train a CNN network that takes these spectrogram images as input and using it tries to generalize patterns and hence classify them.

Model Update

```
Train on 6549 samples, validate on 2183 samples
Epoch 1/100
6549/6549 [==============================] - 6s 874us/sample - loss: 1.5575 - accuracy: 0.4755 - val_loss: 1.2482 - val_accuracy: 0.5630
Epoch 2/100
6549/6549 [==============================] - 5s 753us/sample - loss: 1.1525 - accuracy: 0.6103 - val_loss: 1.1686 - val_accuracy: 0.5873
Epoch 3/100
6549/6549 [==============================] - 5s 731us/sample - loss: 0.9841 - accuracy: 0.6743 - val_loss: 0.9873 - val_accuracy: 0.6803
Epoch 4/100
6549/6549 [==============================] - 5s 811us/sample - loss: 0.8805 - accuracy: 0.7047 - val_loss: 0.8759 - val_accuracy: 0.7100
Epoch 5/100
6549/6549 [==============================] - 6s 977us/sample - loss: 0.7761 - accuracy: 0.7464 - val_loss: 0.8565 - val_accuracy: 0.7109
```

.

.

.

100 times

```
Epoch 95/100
6549/6549 [==============================] - 6s 840us/sample - loss: 0.0425 - accuracy: 0.9847 - val_loss: 0.8905 - val_accuracy: 0.8699
Epoch 96/100
6549/6549 [==============================] - 5s 737us/sample - loss: 0.0450 - accuracy: 0.9843 - val_loss: 0.8358 - val_accuracy: 0.8727
Epoch 97/100
6549/6549 [==============================] - 5s 807us/sample - loss: 0.0433 - accuracy: 0.9856 - val_loss: 0.8681 - val_accuracy: 0.8704
Epoch 98/100
6549/6549 [==============================] - 5s 759us/sample - loss: 0.0474 - accuracy: 0.9827 - val_loss: 0.9050 - val_accuracy: 0.8635
Epoch 99/100
6549/6549 [==============================] - 5s 722us/sample - loss: 0.0450 - accuracy: 0.9832 - val_loss: 0.9286 - val_accuracy: 0.8722
Epoch 100/100
6549/6549 [==============================] - 5s 737us/sample - loss: 0.0549 - accuracy: 0.9832 - val_loss: 0.9685 - val_accuracy: 0.8548
```

figure15: Model training process

I tried CNN code from the Internet and run the model 100 times to adjust and update model. From the process above we could see the both the loss and validation loss of each epoch is decreasing and both the model accuracy and validation accuracy are increasing with each epoch. As an end, for validation dataset, the loss decreases to 0.9685 and the accuracy reaches 0.8548. The we export the predicted class for test data into a csv file called ’Results.csx’.

Result Analysis

To figure why the accuracy is not very high, I compared predicted class and true label and tried to find some index whose data has been divided into a wrong prediction.

Index	label	class
0	7	7
1	0	0
2	4	4
3	3	3
4	3	2
5	3	3
6	5	5
7	7	7
8	4	4
9	5	5
10	3	3
11	8	8
12	2	6
13	7	7
14	1	1

figure16: Compare classification results and true labels

I viewed the first 15 rows and found index 4 and 12 have wrong predictions. So I plotted the figures of these two index's data. I also plotted the patterns of two successful predicted sound record, whose indexes are 443 and 1081. It is obviously the right prediction has a more reasonable spectrogram since it can distinguish the discrete tune of human voice and elongated car horn.

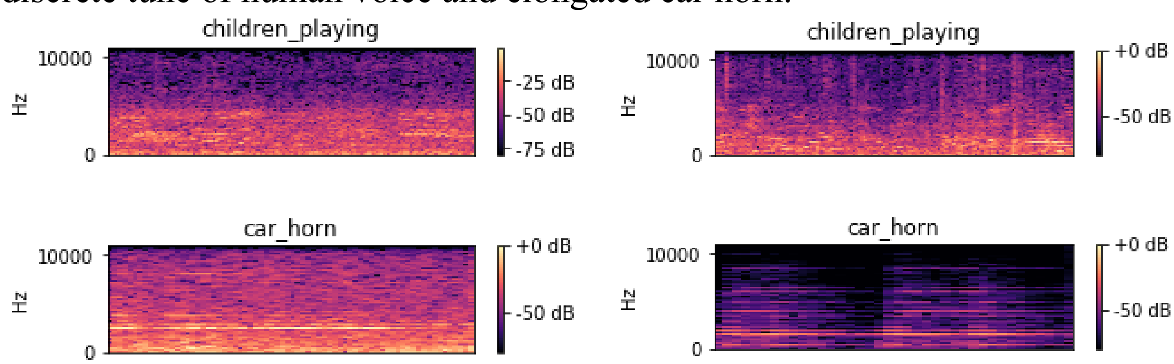

figure17: Dig into why there are some wrong results
(left: the files with wrong results   right: the files with right results)

To explored the problem deeper, I found the files of each figure and listened to them. (index4: 100263-2-0-137.wav, index12: 100648-1-3-0.wav, index443: 105415-2-0-1.wav, index1081: 125520-1-2-0.wav). I could hear the boy speaking in index443 file clearly, and I could hear two clear car horn in index 1081 file. However, as a human, I can also tell that there are children playing and laughing in the noisy environment of the recording index4, and there is a long and loud whistle in the index12 file. So there are still long ways to explore for human to make maching learning as smart as human brains.