**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. Data Manipulation**

Instead of downloading the datasets on websites, I loaded the datasets using Keras datasets package. Because we are doing clustering, so we need to combine all the training feature data and test feature data together. Meanwhile, we still need to combine all the training label data and test label data together.

After that, I found the feature datasets MNIST, CIFAR-10, and LFW are all more than 2 dimensional data. Their shapes are shown below:

*Table 1: Shapes Before Reshaping*

|  |  |  |  |
| --- | --- | --- | --- |
| *Dataset* | MNIST | CIFAR-10 | LFW |
| *Shape* | (70000, 28, 28) | (60000, 32, 32, 3) | (13233, 64, 64) |

If I want to run K-means on those datasets, I need to reshape them into 2 dimensions. As a result, I used numpy to reshape those datasets to make them into 2 dimensions. And after reshaping, the shapes are shown below:

*Table 2: Shapes After Reshaping*

|  |  |  |  |
| --- | --- | --- | --- |
| *Dataset* | MNIST | CIFAR-10 | LFW |
| *Shape* | (70000, 784) | (60000, 3072) | (13233, 4096) |

In the end, I got the data for clustering.

**2. Feature Selection**

I only used the pixel feature in the raw data to do the clustering. However, there are still some other features that I can extract from the raw datasets. I decided to try them later on.

**3. Implementing K-means**

In this part, I tried two K-means methods. One is written by myself, another one is in the scikit-learn. Below are the results of the two methods which include the running time, loss function as well as purity.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Table 2: Results on K-mean Implemented by Sklearn and Me | | | | | | |
| **Methods** | MNIST | | CIFAR-10 | | LFW | |
| Value | Time | Value | Time | Value | Time |
| **K-means in klearn** | 5.9e+6 | 109s | 7.2e+6 | 472s | 5.6e+7 | 123s |
| **My K-means** | 6.0e+6 | 1021s | 8.0e+5 | 2230s | 4.3e+7 |  |

From the results above, we can see that, K-means in sklearn is much faster than mine. I tried to optimized my K-means using a more efficient way to compute the distances, but I still cannot beat the K-means in sklearn. As for the objective function values, mine are relatively lower than those of sklearn k-means.

**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. Data Manipulation**

Instead of downloading the datasets on websites, I loaded the datasets using Keras datasets package. Because we are doing clustering, so we need to combine all the training feature data and test feature data together. Meanwhile, we still need to combine all the training label data and test label data together.

After that, I found the feature datasets MNIST are all more than 2 dimensional data. So I transformed all the data into 2 dimensions.

During the data pre-processing, I standardize features by removing the mean and scaling to range of each column.

As for the feature, I used the pixel-level feature just like the problem 1.

**2. Implemeting Models**

For the above classifiers, I used K-fold validation and grid search to get the optimized parameters. As for the K-fold validation, I set K to 10. As for the algorithms, I employed the codes in scikit-learn which is very effective.

**2. Results Analysis**

**Linear Classifier**

I just used the 1-layer with hidden layer sizes range from 1 to 15. You can also try different layers with just simple hidden layers. However, usually, just one layer is enough. And I did the grid search on hidden layer size in [1,15]. In the end, I got the test error rate: 13.2%, which is fine.

**K-nearest-neighbors**

As for KNN, the time complexity is so high, and it will take a very long time if I ran all the dataset. To solve this problem, there are two methods: 1. Sampling, 2. Do clustering on the datasets, and find the distance between each centroid. I choose the second method, because in problem 1 I did the clustering for the datasets, which only took 110 seconds.

And I did the grid search on number of neighbors in [3, 4, 5, 6]. The test error rate is 5.65% which is very low.

**SVM with Gaussian Kernel**

I did the grid search on regularization parameter in [0.01, 0.1, 0.3, 0.8 1.0]. In the end, I found regularization parameter set to 0.8 is the best. And the test error rate I got is 4.47 % which is very low. And it’s the best model compare to the above tow.

**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?

**Answer:** Based on Friedman ranking, the best classifier is parallel random forest. Meanwhile, the none-parallel version can usually get high performance. The two random forest approaches are very good for practical use. Moreover, LibSVM with Gaussian kernel and SVM with polynomial kernel are also very good methods for practical use.

**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.

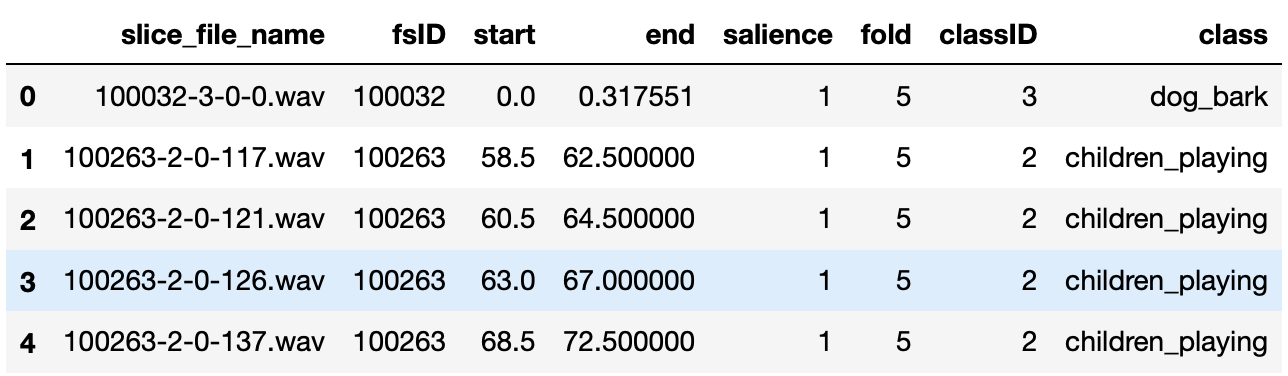
**UrbanSound8K – Classification**

**1. Introduction on datasets**

This dataset contains 8732 labeled sound excerpts (<=4s) of urban sounds from 10 classes: air\_conditioner, car\_horn, children\_playing, dog\_bark, drilling, enginge\_idling, gun\_shot, jackhammer, siren, and street\_music. The classes are drawn from the urban sound taxonomy. For a detailed description of the dataset and how it was compiled please refer to our paper. All excerpts are taken from field recordings uploaded to [www.freesound.org](http://www.freesound.org). The files are pre-sorted into ten folds (folders named fold1-fold10) to help in the reproduction of and comparison with the automatic classification results reported in the article above.

In addition to the sound excerpts, a CSV file containing metadata about each excerpt is also provided.

**Details on datasets**

****

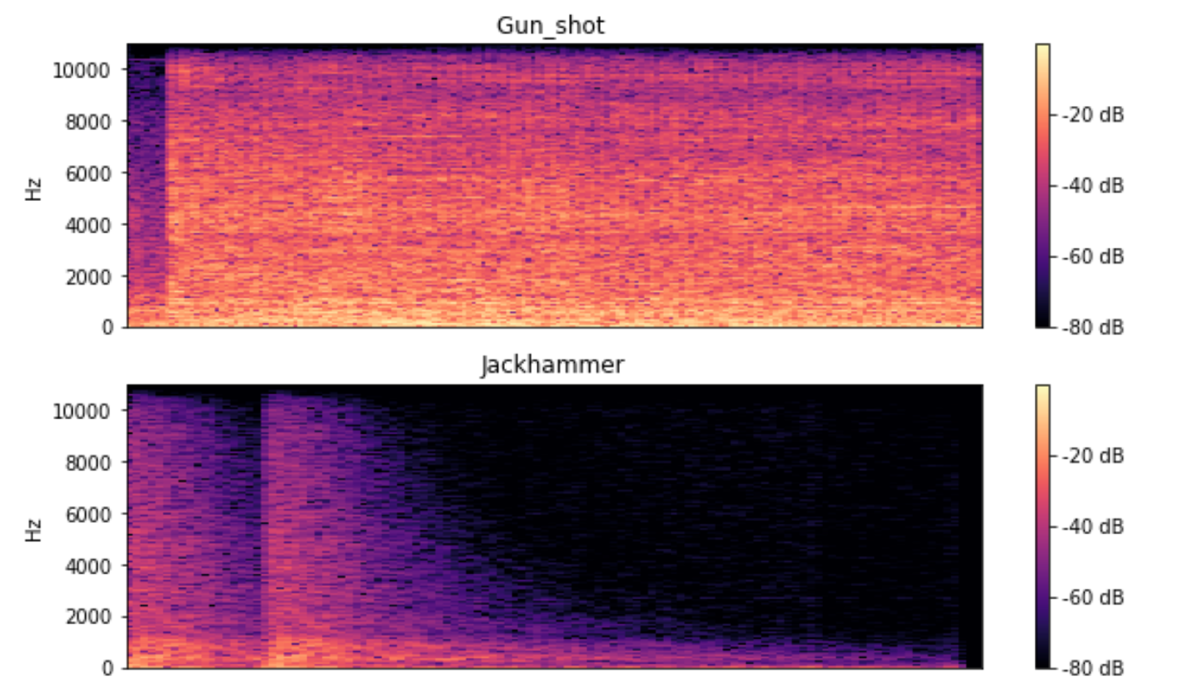
Above is a grasp ofCSV file, below is the details of each cloumns.

--Column Names

* slice\_file\_name: The name of the audio file. The name takes the following format: [fsID]-[classID]-[occurrenceID]-[sliceID].wav, where: [fsID] = the Freesound ID of the recording from which this excerpt (slice) is taken [classID] = a numeric identifier of the sound class (see description of classID below for further details) [occurrenceID] = a numeric identifier to distinguish different occurrences of the sound within the original recording [sliceID] = a numeric identifier to distinguish different slices taken from the same occurrence
* fsID: The Freesound ID of the recording from which this excerpt (slice) is taken
* start The start time of the slice in the original Freesound recording
* end: The end time of slice in the original Freesound recording
* salience: A (subjective) salience rating of the sound. 1 = foreground, 2 = background.
* fold: The fold number (1-10) to which this file has been allocated.
* classID: A numeric identifier of the sound class: 0 = air\_conditioner 1 = car\_horn 2 = children\_playing 3 = dog\_bark 4 = drilling 5 = engine\_idling 6 = gun\_shot 7 = jackhammer 8 = siren 9 = street\_music
* class: The class name: air\_conditioner, car\_horn, children\_playing, dog\_bark, drilling, engine\_idling, gun\_shot, jackhammer, siren, street\_music.

**Using random samples to observe difference in waveforms.**

I also usedLibrosa package to visualize different classes. We can see that, different class has different waveforms.



**2. Methodology**

1. There are 3 basic methods to extract features from audio file : a) Using the mffcs data of the audio files b) Using a spectogram image of the audio and then converting the same to data points (As is done for images). This is easily done using mel\_spectogram function of Librosa c) Combining both features to build a better model. (Requires a lot of time to read and extract data).

2. The labels have been converted to categorical data for classification.

3. CNN has been used as the primary layer to classify data

**2. Feature Extraction and Database Building**

--Method

1. I have used Librosa to extract features. For features, I tried both Mel-spectrogram and MFCC. I went through each fold and extracted the data for each file.

2.After reshaping and cleaning the data, I employed 10-fold cross validation. The splitting was based on the ten folders.

3. Labels were converted to Categorically Encoded Data.

Note : Extracting features may take upto 45 minutes depending on your hardware since it has to extract spectogram data for 8732 audio files. As a result, I did this step on remote server which took me 10 minutes.

**5.Model selection(CNN)**

Creating Keras Model and Testing

1. CNN 2D with 64 units and tanh activation.
2. MaxPool2D with 2\*2 window.
3. Dropout Layer with 0.5 drop probability.
4. CNN 2D with 64 units and tanh activation.
5. MaxPool2D with 2\*2 window.
6. Dropout Layer with 0.5 drop probability.
7. DL with 128 units and tanh activation.
8. Dropout Layer with 0.5 drop probability.
9. DL 10 units with softmax activation.
10. Adam optimizer with categorical\_crossentropy loss function.
11. 100 epochs have been used.

**6.Model update**

Adam Optimizer and Cross Entropy loss are used to update the model.

**7. Result analysis.**

The average accuracy is 53.49% with MFCC. While the results of each fold vary a lot (43% to 60%), the accuracy may be highly related to the samples. Moreover, the accuracy with MFCC is higher than that of Mel-spectrogram, the accuracy is related to the features, too.