Final_Report

Li Yuan

July 25, 2021

1 Backgound and Introduction

We will initialize two main methods appearing in Neural Network.

1. The first one is a new method in Image Data Augmentation.
2. The second one is a new method in redesigning Committee Vote.

Currently, most image data augmentation techniques are only using one input to shift, rotate, add white noise. However, my new method will incorporate random multiple inputs to extract their common obvious features by committee votes based on each class. Our method performs well on small size data.

The present mature method of committee vote is to choose mode or median as the final decision, but those methods don't consider probability and weights. However, I will come up with a method which make use of probability distribution and appearing frequency as weights to redesign my new committee vote.

1.1 Introduce our Python3.8 environment to reproduce my results

[24]:
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.pyplot import imshow
from keras.datasets import mnist

from keras.preprocessing.image import ImageDataGenerator
import os

import sklearn
from sklearn.utils import shuffle

from tensorflow import keras
from tensorflow.keras import layers

import copy
%config InlineBackend.figure_format = 'pdf'

import warnings
warnings.filterwarnings('ignore')
```

2 The current general image augmentation method review

2.1 Load Digit Written MNIST database as our new method experimental data

[25]:
```
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

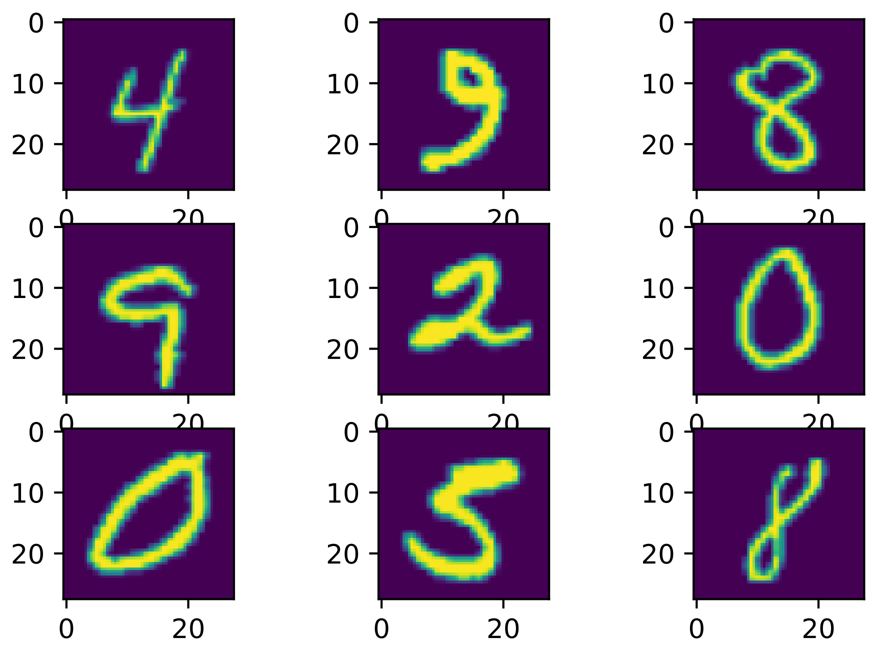
2.2 reshape to be [samples][width][height][channels] and change to float type

[26]:
```
X_train = X_train.reshape((X_train.shape[0], 28, 28, 1))
X_train = X_train.astype('float32')
# Set seed
np.random.seed(12345)
```

2.3 Feature Standardization

This method will centralize and standardize the data. The raw data sometimes will benefit from this manipulation because of some outliers and different scales.

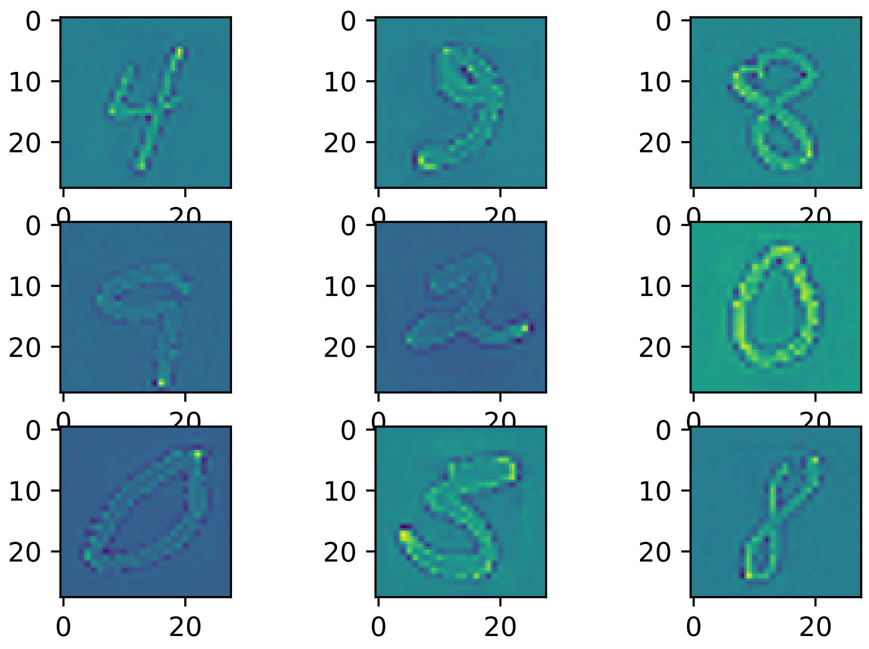
[27]:
```
# define data preparation
datagen = ImageDataGenerator(featurewise_center=True,␣
 ↪featurewise_std_normalization=True)
# fit parameters from data
datagen.fit(X_train)
# configure batch size and retrieve one batch of images
for X_batch, y_batch in datagen.flow(X_train, y_train, batch_size=9, seed=123):
    # create a grid of 3x3 images
    for i in range(0, 9):
        plt.subplot(330 + 1 + i)
        plt.imshow(X_batch[i].reshape(28, 28))
    # show the plot
    plt.show()
    break
```



2.4 ZCA Whitening

This technique is used to linearly transform raw data such that covariance matrix of it is identical and make sure to decrease the correlation in data.

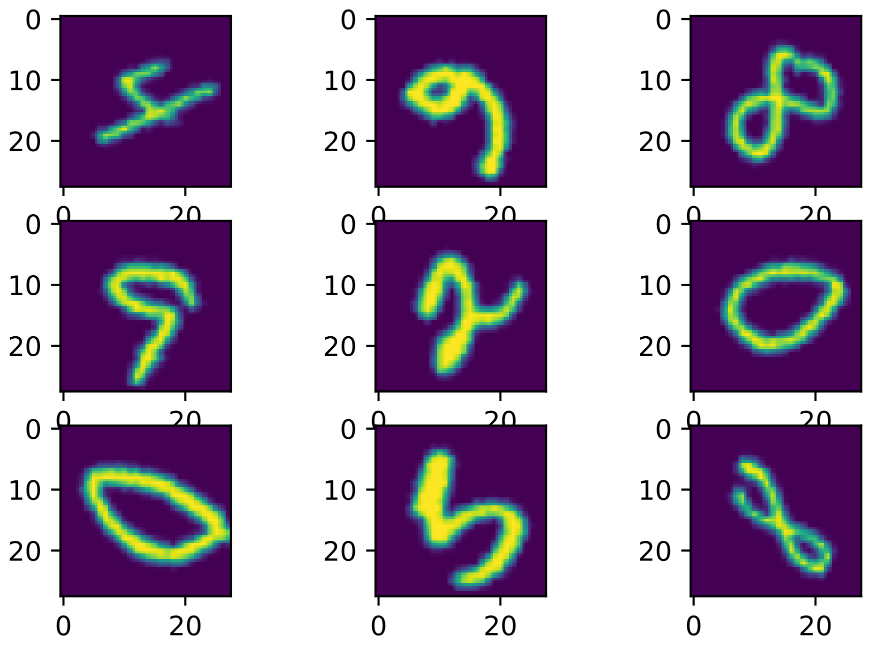
[28]:
```
datagen = ImageDataGenerator(zca_whitening=True)
datagen.fit(X_train)
for X_batch, y_batch in datagen.flow(X_train, y_train, batch_size=9, seed=123):
    for i in range(0, 9):
        plt.subplot(330 + 1 + i)
        plt.imshow(X_batch[i].reshape(28, 28))
    plt.show()
    break
```



2.5 Random Rotations

This method is very straightforward. Some of images are randomly chosen to rotate 90 degree.

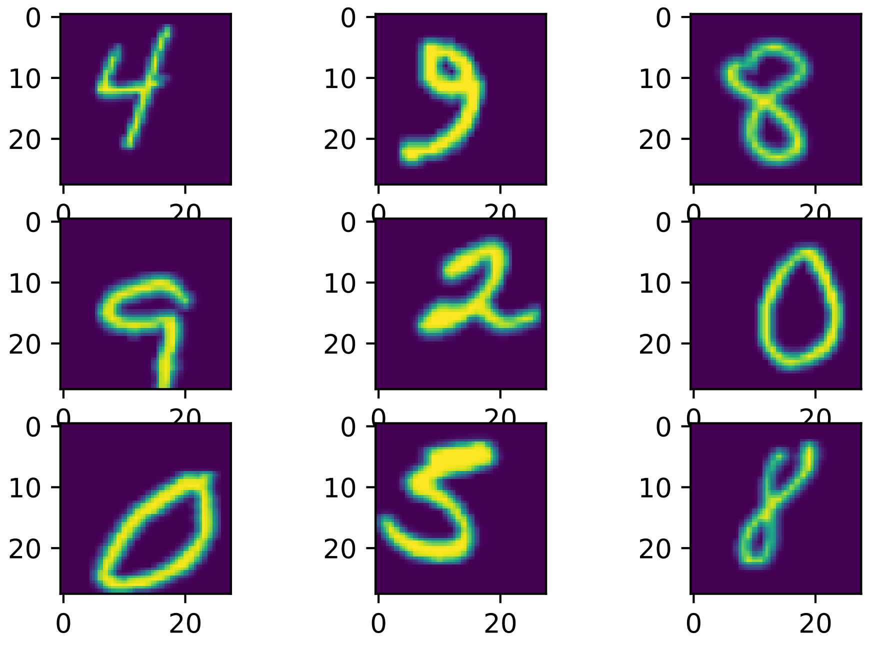
```
[29]: datagen = ImageDataGenerator(rotation_range=90)
```



2.6 Random Shifts

This method is also very straightforward, some images are randomly shifted vertically or horizontally by different values.

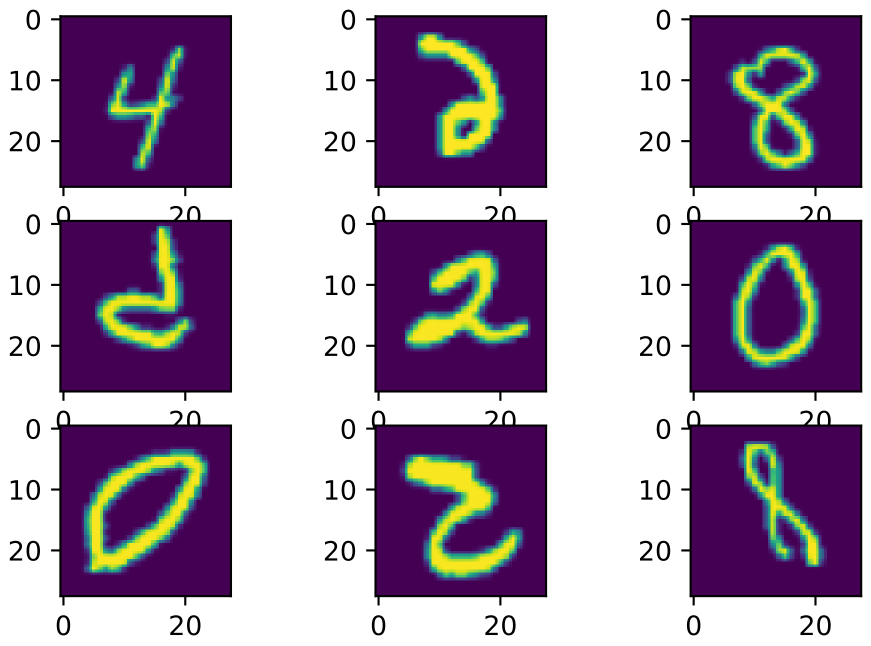
[31]:
```
# define data preparation
shift = 0.2
datagen = ImageDataGenerator(width_shift_range=shift, height_shift_range=shift)
```



2.7 Random Flips

This method is just to flip the random images and keep the remaining the same.

[33]:
```
datagen = ImageDataGenerator(horizontal_flip=True, vertical_flip=True)
```



2.8 My new method

All above proposed methods only use one input image. The drawback of this method is that one new image can't learn common characteristics of this class due to one picture. I want to leverage the idea of the committee vote to select multiple input images to generate new images. This committee vote can learn the obvious pattern of this class and spit out this pattern in new image. I will step by step to illustrate my method.

2.8.1 Set a parameter S, size of committee vote, then randomly select S images from the class(instead of the whole data) we want to generate

[ ]:
```
cla = dataset
np.random.shuffle(cla) # shuffle the class
sub_index = np.random.choice(cla.shape[0], S, False) # random subset S data
basis = cla[sub_index, :, :, :] # form the basis
```

2.8.2 Iterate each pixel of image shape, for example we use (28, 28, 1)

[ ]:
```
for i in range(28):
    for j in range(28):
        count = []
```

2.8.3 Iterate each image on the selected pixel in this committee vote

```
[ ]: S = 9
     for q in range(S):
```

2.8.4 Set a parameter thre, threshold of each pixel, then judge if the pixel value of each image is greater than this threshold

```
[ ]: if basis[q, i, j, 0] >= thre:
         count.append(basis[q, i, j, 0])
```

2.8.5 Set a parameter p, percentage of committee vote needed to calculate the average of values on this pixel

```
[ ]: if len(count) >= S * p:
         new[i, j, 0] = np.mean(count)
```

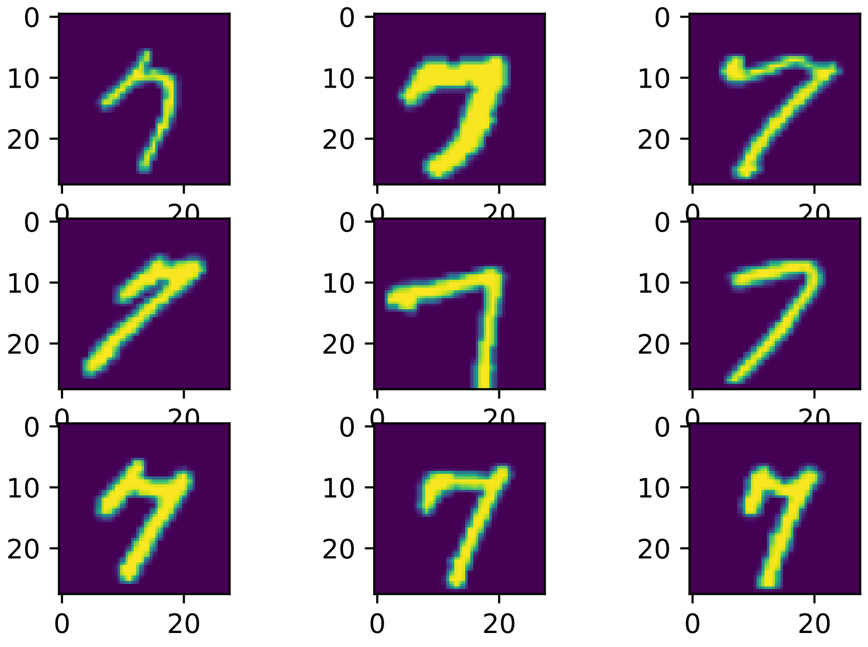
2.9 After these steps, we will combine them together to define a function to process a new image based on multiple inputs

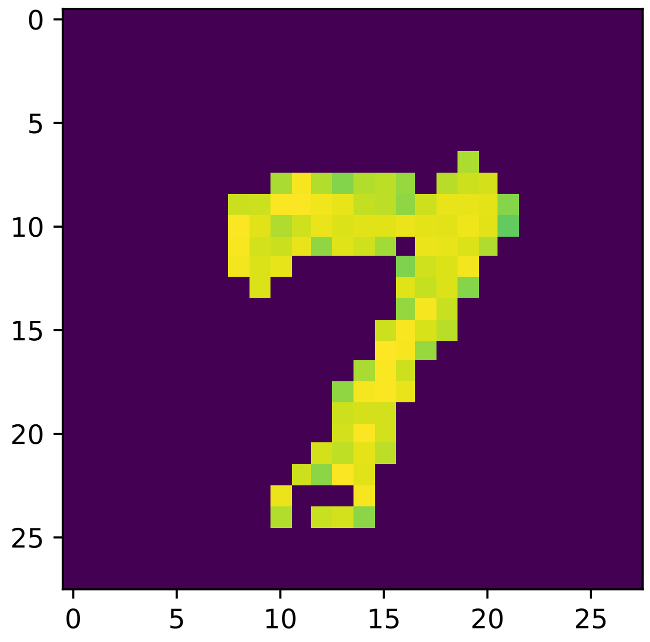
```
[35]: def generator(dataset, num = 0, S = 50, p = 0.25, thre = 140):
          '''
          by tuning paramaters we generate one new digit image
          '''
          cla = dataset
          np.random.shuffle(cla) # shuffle the class
          sub_index = np.random.choice(cla.shape[0], S, False) # random subset S data
          basis = cla[sub_index, :, :, :] # form the basis

          new = np.zeros_like(basis[1])
          for i in range(new.shape[0]):
              for j in range(new.shape[1]):
                  count = []
                  for q in range(S):
                      if basis[q, i, j, 0] >= thre:
                          count.append(basis[q, i, j, 0])
                  if len(count) >= S * p:
                      new[i, j, 0] = np.mean(count)
          return new
```

2.10 Next we will use the variant of above function to tune suitable parameters and compare new generated image with committee vote images

2.10.1 For example, we used 7 as our first trial number, 9 as our committee size, 40% as our percentage to pass, 150 as each pixel threshold value

[39]:
```
generator1(num = 7, S = 9, p = 0.4, thre = 150)
```

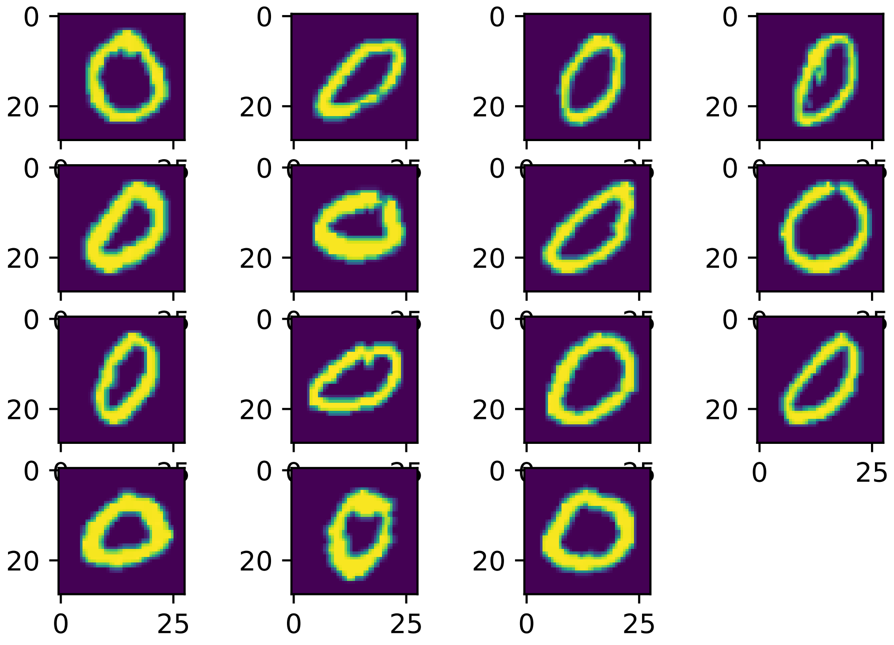


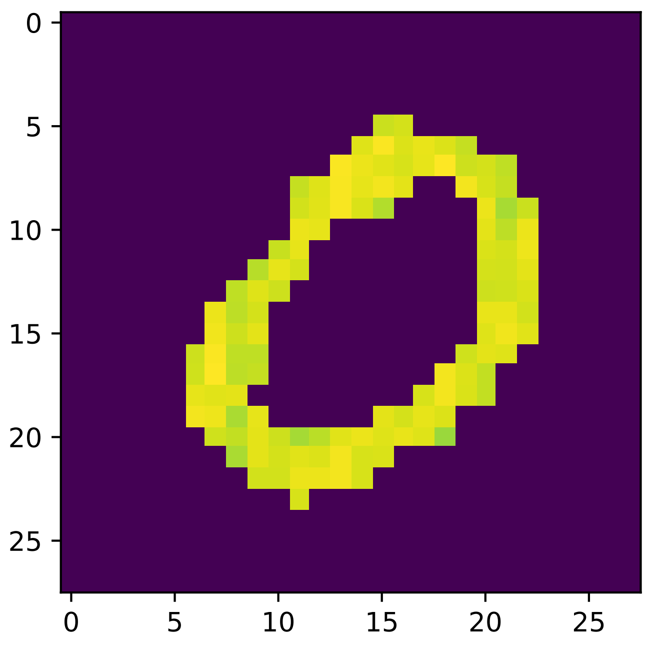


As we see from number = 7, the generated new image capture most common part of digit 7 in this hand-written 9 images. If you want to get better results, you can tune more parameters to find the best combination of it. These three parameters, *S*, *p*, *thre*, are well represented the most features we can manually tune to generate an artificial image based on what we had.

2.10.2 Now we change our number class to 0 and S = 15, p = 0.6, thre = 170

[41]:
```
generator1(num = 0, S = 15, p = 0.6, thre = 140)
```





This 0 image performs better than the previous one 7, which means we find a better combination of parameters to capture its characteristic. Actually, for some classes, the more complex the class is, the harder new image can capture.

2.11 Compare the performance of this method with original raw data

2.11.1 Create a small subset data from MNIST

2.11.2 Split data based on each class from 0 thru 9

[42]:
```
# the data, split between train and test sets
(X_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()

# Scale images to the [0, 1] range
x_test = x_test.astype("float32") / 255
# Make sure images have shape (28, 28, 1)
X_train = np.expand_dims(X_train, -1)
x_test = np.expand_dims(x_test, -1)

x0 = X_train[y_train == 0]
x1 = X_train[y_train == 1]
x2 = X_train[y_train == 2]
x3 = X_train[y_train == 3]
x4 = X_train[y_train == 4]
x5 = X_train[y_train == 5]
x6 = X_train[y_train == 6]
x7 = X_train[y_train == 7]
```

```
x8 = X_train[y_train == 8]
x9 = X_train[y_train == 9]
```

2.11.3 Randomly choose a small subset of each class from 0 thru 9

[44]:
```
p = 50
np.random.seed(123)
x0_ind = np.random.choice(x0.shape[0], x0.shape[0]//p, False)
x1_ind = np.random.choice(x1.shape[0], x1.shape[0]//p, False)
x2_ind = np.random.choice(x2.shape[0], x2.shape[0]//p, False)
x3_ind = np.random.choice(x3.shape[0], x3.shape[0]//p, False)
x4_ind = np.random.choice(x4.shape[0], x4.shape[0]//p, False)
x5_ind = np.random.choice(x5.shape[0], x5.shape[0]//p, False)
x6_ind = np.random.choice(x6.shape[0], x6.shape[0]//p, False)
x7_ind = np.random.choice(x7.shape[0], x7.shape[0]//p, False)
x8_ind = np.random.choice(x8.shape[0], x8.shape[0]//p, False)
x9_ind = np.random.choice(x9.shape[0], x9.shape[0]//p, False)

x0_train = x0[x0_ind, :, :, :]
x1_train = x1[x1_ind, :, :, :]
x2_train = x2[x2_ind, :, :, :]
x3_train = x3[x3_ind, :, :, :]
x4_train = x4[x4_ind, :, :, :]
x5_train = x5[x5_ind, :, :, :]
x6_train = x6[x6_ind, :, :, :]
x7_train = x7[x7_ind, :, :, :]
x8_train = x8[x8_ind, :, :, :]
x9_train = x9[x9_ind, :, :, :]
```

2.11.4 Combine each small subset class and shuffle them with their labels

[45]:
```
new_train = np.concatenate([x0_train, x1_train, x2_train, x3_train, x4_train,␣
 ↪x5_train, x6_train, x7_train, x8_train, x9_train], axis = 0)

new_y_train = np.array([0] * x0_train.shape[0] +
[1] * x1_train.shape[0] +
[2] * x2_train.shape[0] +
[3] * x3_train.shape[0] +
[4] * x4_train.shape[0] +
[5] * x5_train.shape[0] +
[6] * x6_train.shape[0] +
[7] * x7_train.shape[0] +
[8] * x8_train.shape[0] +
[9] * x9_train.shape[0])

train, y = sklearn.utils.shuffle(new_train, new_y_train, random_state=0)
```

[ ]:
```

```