Description of files

Data files:

- `PCPG.txt`: the $\log(\cdot+1)$ transformed count matrix of PCPG dataset. Rows are samples, and columns are genes.
- `PCPG_label.txt`: the one-hot encoded labels of samples, which is an $N \times K$ matrix, where N is the sample size, K is the number of classes.

Python files:

- `SDC_OP2.py`: includes the main function of SINC, `sdc()`, which builds and trains the neural network and evaluates the model with the test data provided.
- `scale_noise.py`: includes the function to augment train data for both scaling factors and technical noises.
- `OP_PCPG_2.py`: Run this file to conduct K-fold CV on PCPG dataset. It will show the CV accuracy on the original and augmented data and concordance rate.

Description of key functions

1. `sdc`

Build and train the neural network on the given training data and evaluate the model with the given test data.

Argurments:

- `expr_train`: the expression data of training sets on the log scale, which is a $n \times p$ matrix, where n is the sample size of training data and p is the number of genes (or the dimension of input layer).
- `labels_train`: the class labels (one-hot encoded) of training data, which is a $n \times K$ matrix, where n is the sample size of training data and $\mathrm{K}K$ is the number of classes.
- `expr_test`: which is a $m \times p$ matrix, where m is the sample size of test data and p is the number of genes (or the dimension of the input layer).
- `labels_test`: the class labels of test data, which is an $m \times K$ matrix, where m is the sample size of test data and K is the number of classes.
- `epoch`: the maximal number of training epochs, or the folds of augmentation.
- `min_epoch`: the minimal number of training epochs
- `lr`: the starting value of learning rate to train the network.
- `batch_size`: the batch size to train the network.
- `min_stop`: when the training epoch is larger than `min_stop`, and the change of loss is smaller than `min_stop`, stop the training.
- `display_step`: for every `display_step` of epochs, print the current number of epochs, loss value, and classification accuracy on the test data.
- `aug`: the folds of augmentation for test data to evaluate the accuracy on the augmented test data and concordance rate.

Values:

- `res0`: classification accuracy on the test data.
- `res1`: classification accuracy on the augmented test data.
- `OP`: the concordance rate of the predicted labels on the augmented test data.
- `clist`: a list that records the value of loss after each training epoch.
- `ct_list`: a list that records the test accuracy after each training epoch.
- `pred_test`: the output of the trained network on the test data.
- `pred_test`: an $m \times K$ matrix, where m is the sample size of test data, and K is the number of classes. The predicted labels of test data can be obtained by `np.argmax(pred_test)`.

Example:

```
from SDC_OP2 import sdc
```

```
acc_test,acc_aug,concordance,clist,ct_list,y_hat =
sdc(expr_train,labels_train,expr_test,labels_test)
```

2. `mean_std_est`

Estimate the mean expression level (on the log scale) and the within-class standard deviation of each gene. Fit the linear relationship between means and standard deviations.

Arguments:

- `expr`: an $n \times p$ expression matrix, where n is the number of samples, and p is the number of genes.
- *labels*: an $n \times K$ matrix, which contains the one-hot encoded class labels of samples. n is the number of samples, and K is the number of classes.

Values:

- `U`: a length-p vector, which is the mean expression level of genes (across classes).
- `std_h`: a length-p vector, which records the fitted within-group standard deviations.

2. `augment_noise_scale`

Augment expression data for both technical noises and scaling factors.

Arguments:

- `expr`: an $n \times p$ expression matrix, where n is the number of samples, and p is the number of genes.
- *labels*: an $n \times K$ matrix, which contains the one-hot encoded class labels of samples. n is the number of samples, and K is the number of classes.
- `Fold`: the times of augmentation, whose default value is 100.
- `alpha`: the proposed proportion of technical noises in standard deviations, whose default value is 0.2.

Values:

- `expr1`: the expression matrix after randomly generated technical noises are added to each sample.
- `SF`: the randomly generated scaling factors for samples.

3. `gene_filter`

Order the genes according to how variable they are.

Arguments:

- `expr`: an $n \times p$ expression matrix, where n is the number of samples, and pp is the number of genes.
- cl: an $n \times K$ matrix, which contains the one-hot encoded class labels of samples. n is the number of samples, and K is the number of classes.

Values:

- `orders`: Order the genes according to their p-values of variability. The top ordered genes have the smallest p-values, meaning they are highly variable.

Tutorial

Here we show an example of using PCPG data to conduct five-fold cross-validation.

1. Import all modules and functions that will be used.

```
import numpy as np
import random
from SDC_OP2 import sdc
from scipy import stats
from scale_noise import augment_noise_scale
```

2. Read the expression matrix and labels (one-hot encoded) of PCPG. Filter out Then we divide the data into five folds.

```
DATASET = "PCPG"
expr = np.loadtxt(DATASET+".txt")
cl = np.loadtxt(DATASET+"_label.txt")
## To do cross-validation, K is the number of folds that we set.
K = 5
expr = np.asarray(expr).T # Transpose the matrix, so that rows are samples and columns are
genes
cl =  cl.astype(int)
N = cl.shape[0] # N is the total number of cells/ sample size
np.random.seed(200)
indx = np.random.randint(0,K,N)
## filter out lowly expressed genes.
zerop = np.mean(expr==0,axis=0)
expr=expr[:,zerop<0.8]
```

3. Take the i-th fold as test data and the rest as training data. Select high variable genes only based on training data.

```
expr_train.append(expr[indx!=i,])
cl_train.append(cl[indx!=i,])
expr_test.append(expr[indx==i,])
cl_test.append(cl[indx==i,])
## orders of negative p values for genes
orders_t = gene_filter(expr=expr_train[i],cl=cl_train[i])
expr_train[i] =(expr_train[i][:,orders_t<=n_genes])
expr_test[i] = (expr_test[i][:,orders_t<=n_genes])
```

4. Train the network with user's choice of hyper-parameters. Obtain test accuracy and concordance rate.

```
## res_op is a vector recording the concordance rate of each sample.
res_aug_0[i],res_aug_1[i],res_op[indx==i],clist,ct_list,label_hat[indx==i,] =
sdc(expr_train=expr_train[i],labels_train=cl_train[i],expr_test=expr_test[i],labels_test=cl_te
st[i],lr=0.02,min_stop=0.05,epoch=nepoch,min_epoch=nepoch,aug=200,display_step=100,batch_size=
24)

## acc_aug_0 is used to the accuracy on the original test data
## acc_aug_1 is used to the accuracy on the augmented test data
acc_aug_0=acc_aug_0+res_aug_0[i]*expr_test[i].shape[0]
acc_aug_1=acc_aug_1+res_aug_1[i]*expr_test[i].shape[0]
```

4. After K (K=5) iterations, obtain the results of cross-validation, including accuracy on the original test data and the augmented test data, and the average concordance rate.

```
acc_aug_0 = acc_aug_0/expr.shape[0]
acc_aug_1 = acc_aug_1/expr.shape[0]
print("On the original test set the accuracy is",acc_aug_0,";The concordance rate
is:",np.mean(res_op),"; Accuracy on the augmented is ",acc_aug_1)
```