```
from tqdm import tqdm
import numpy as np
import pandas as pd
import anndata
from anndata import read_h5ad
import warnings
warnings.filterwarnings("ignore")
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.neighbors import KernelDensity
import torch
import random
import torch.nn as nn
from torch.optim import Adam
from torch.utils.data import Dataset, DataLoader
import torch.nn.functional as F
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as sts
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

```
class simdatset(Dataset):
    def __init__(self, X, Y, use_noise=False):
        self.X = X
        self.Y = Y
        self.use_noise = use_noise
        if use_noise:
            self.noise =
np.random.normal(self.X.mean(axis=0),self.X.std(axis=0),size=self.X.shap
e)
            self.noiseX = self.X + np.abs(self.noise)

    def __len__(self):
        return len(self.X)

    def __getitem__(self, index):
        if self.use_noise is False:
            x = torch.from_numpy(self.X[index]).float().to(device)
            y = torch.from_numpy(self.Y[index]).float().to(device)
            return x, y
        else:
            x = torch.from_numpy(self.X[index]).float().to(device)
            noise_x =
torch.from_numpy(self.noiseX[index]).float().to(device)
```

```
            y = torch.from_numpy(self.Y[index]).float().to(device)
            return x, y, noise_x



class AutoEncoder(nn.Module):
    def __init__(self, input_dim, output_dim):
        super().__init__()
        self.name = 'ae'
        self.state = 'train' # or 'test'
        self.inputdim = input_dim
        self.outputdim = output_dim
        self.encoder = nn.Sequential(nn.Dropout(),
                                     nn.Linear(self.inputdim, 512),
                                     nn.CELU(),

                                     nn.Dropout(),
                                     nn.Linear(512, 256),
                                     nn.CELU(),

                                     nn.Dropout(),
                                     nn.Linear(256, 128),
                                     nn.CELU(),

                                     nn.Dropout(),
                                     nn.Linear(128, 64),
                                     nn.CELU(),

                                     nn.Linear(64, output_dim),
                                     )




        self.decoder = nn.Sequential(nn.Linear(self.outputdim, 64,
bias=False),
                                     nn.Linear(64, 128, bias=False),
                                     nn.Linear(128, 256, bias=False),
                                     nn.Linear(256, 512, bias=False),
                                     nn.Linear(512, self.inputdim,
bias=False))


    def encode(self, x):
        return self.encoder(x)

    def decode(self, z):
        return self.decoder(z)

    def refraction(self,x):
        x_sum = torch.sum(x, dim=1, keepdim=True)
        return x/x_sum


    def sigmatrix(self):
        w0 = self.decoder[0].weight.T
        w1 = self.decoder[1].weight.T
        w2 = self.decoder[2].weight.T
        w3 = self.decoder[3].weight.T
        w4 = self.decoder[4].weight.T
        w01 = (torch.mm(w0, w1))
        w02 = (torch.mm(w01, w2))
        w03 = (torch.mm(w02, w3))
        w04 = F.hardtanh(torch.mm(w03, w4),0,1)
        return w04

    def forward(self, x):
        sigmatrix = self.sigmatrix()
        z = self.encode(x)

        if self.state == 'train':
            pass
        elif self.state == 'test':

            z = F.hardtanh(z,0,1)
            z = self.refraction(z)

        x_recon = torch.mm(z, sigmatrix)
        return x_recon, z, sigmatrix


def L1error(pred, true):
    return np.mean(np.abs(pred - true))


def CCCscore(y_pred, y_true):
    # pred: shape{n sample, m cell}
    ccc_value = 0
    for i in range(y_pred.shape[1]):
        r = np.corrcoef(y_pred[:, i], y_true[:, i])[0, 1]
        # print(r)
        # Mean
        mean_true = np.mean(y_true[:, i])
        mean_pred = np.mean(y_pred[:, i])
        # Variance
        var_true = np.var(y_true[:, i])
        var_pred = np.var(y_pred[:, i])
        # Standard deviation
        sd_true = np.std(y_true[:, i])
        sd_pred = np.std(y_pred[:, i])
        # Calculate CCC
        numerator = 2 * r * sd_true * sd_pred
        denominator = var_true + var_pred + (mean_true - mean_pred) ** 2
        ccc = numerator / denominator
        # print(ccc)
        ccc_value += ccc
    return ccc_value / y_pred.shape[1]

def score(pred, label):
    new_pred = pred.reshape(pred.shape[0]*pred.shape[1],1)
    new_label = label.reshape(label.shape[0]*label.shape[1],1)
    distance = L1error(new_pred, new_label)
    ccc = CCCscore(new_pred, new_label)
    return distance, ccc

def showloss(loss):
    plt.figure()
    plt.plot(loss)
    plt.xlabel('iteration')
    plt.ylabel('loss')
    plt.show()

def plot_scatter(model,test_loader):
    for data, label in test_loader:
        ori = label
        pred, x_recon, l1 = model(data)
        break
    ori = ori.cpu().detach().numpy()
    pred = pred.cpu().detach().numpy()
    fig, ax =
plt.subplots(pred.shape[1],sharex='col',sharey='row',figsize=
(3,15),dpi=100)
```

```
    fig.suptitle('Results')
    cccValue = 0
    for i in range(pred.shape[1]):
        y = pred[:,i]
        x = ori[:,i]
        ax[i].scatter(x, y, s=10)
        z1 = np.polyfit(x, y, 1)
        p1 = np.poly1d(z1)
        x = x.reshape(-1,1)
        y = y.reshape(-1,1)
        ccc = CCCscore(x,y)
        cccValue += ccc
        ax[i].text(0.1,0.8,str(ccc))
        yvals=p1(x)
        ax[i].set_xlim(0,1)
        ax[i].set_ylim(0,1)
        ax[i].plot(x, yvals, 'r',label='polyfit values')
    print(cccValue/pred.shape[1])
    plt.show()

def reproducibility(seed=1):
    torch.manual_seed(seed)
    random.seed(seed)
    np.random.seed(seed)
    if torch.cuda.is_available():
        torch.cuda.manual_seed_all(seed)
        torch.backends.cudnn.deterministic = True


def training_stage(model, train_loader, optimizer, epochs=10):
    model.train()
    model.state = 'train'
    loss = []
    recon_loss = []
    for i in tqdm(range(epochs)):
        for k, (data, label, noised) in enumerate(train_loader):
            optimizer.zero_grad()
            x_recon, cell_prop, sigm = model(data)
            batch_loss = F.l1_loss(x_recon,
data)+torch.mean(torch.abs((cell_prop-label/(label+1))))# +
F.l1_loss(cell_prop, label)
            batch_loss.backward()
            optimizer.step()
            loss.append(F.l1_loss(cell_prop,
label).cpu().detach().numpy())
            recon_loss.append(F.l1_loss(x_recon,
data).cpu().detach().numpy())

    return model, loss, recon_loss

def train_model(train_x, train_y,
                batch_size=128, iteration=5000, seed=0):
    reproducibility(seed)
    train_loader = DataLoader(simdatset(train_x, train_y, True),
batch_size=batch_size, shuffle=True)
    model = AutoEncoder(train_x.shape[1], train_y.shape[1]).to(device)
    optimizer = Adam(model.parameters(), lr=1e-4)#weight_decay=5e-6
    model, loss, reconloss = training_stage(model, train_loader,
optimizer, epochs=int(iteration /(len(train_x)/batch_size)))
    print('prediction loss is:')
    showloss(loss)
    print('reconstruction loss is:')
    showloss(reconloss)
    return model


def testTAPE(train_x, train_y, test_x, test_y, seed=0):
    reproducibility(seed)
    model = train_model(train_x, train_y, seed=seed)
    model.eval()
    model.state = 'test'
    data = torch.from_numpy(test_x).float().to(device)
    _, pred, _ = model(data)
    pred = pred.cpu().detach().numpy()
    a,b = score(pred,test_y)
    print(a,b)
    return pred
```

```
def preprocess(trainingdatapath, testx=None, testy=None,
testlabel='seq'):
    if (testx is not None) and (testy is not None):
        pbmc = read_h5ad(trainingdatapath)
        donorA = pbmc[pbmc.obs['ds']=='donorA']
        donorC = pbmc[pbmc.obs['ds']=='donorC']
        data6k = pbmc[pbmc.obs['ds']=='data6k']
        data8k = pbmc[pbmc.obs['ds']=='data8k']
        train_data = anndata.concat([donorA, data8k])
        test_x = pd.read_csv(testx, sep='\t')
        test_y = pd.read_csv(testy)
        intersection_genes =
list(test_x.index.intersection(train_data.var.index))
        test_x = test_x.loc[intersection_genes]
        simuvar = list(train_data.var.index)
        intersection_gene_position = []
        for gene in intersection_genes:
            intersection_gene_position.append(simuvar.index(gene))
        selected = np.zeros((len(intersection_genes),
len(train_data.X)))
        for i in range(selected.shape[0]):
            selected[i] = train_data.X.T[intersection_gene_position[i]]
        train_x = selected.T
        index_name = test_y.index
        intersection_cell =
list(test_y.columns.intersection(train_data.obs.columns))
        train_y = train_data.obs[intersection_cell].values
        ### re
        for i, values in enumerate(train_y):
            r_sum = np.sum(values)
            if r_sum == 0:
                pass
            else:
                train_y[i] = train_y[i] / r_sum
        ###
        test_y = test_y[intersection_cell]
        test_x = test_x.T
        test_x = test_x.values
        test_y = test_y.values
        ### re
        for i, values in enumerate(test_y):
            r_sum = np.sum(values)
            if r_sum == 0:
                pass
            else:
                test_y[i] = test_y[i] / r_sum
        ###
        assert test_x.shape[1] == train_x.shape[1]
        assert test_y.shape[1] == train_y.shape[1]
        return train_x, train_y, test_x, test_y, index_name,
intersection_cell

    else:
        pbmc = read_h5ad(trainingdatapath)

        pbmc1 = pbmc[pbmc.obs['ds']=='sdy67']
        microarray = pbmc[pbmc.obs['ds']=='GSE65133']

        donorA = pbmc[pbmc.obs['ds']=='donorA']
        donorC = pbmc[pbmc.obs['ds']=='donorC']
        data6k = pbmc[pbmc.obs['ds']=='data6k']
        data8k = pbmc[pbmc.obs['ds']=='data8k']

        if testlabel == 'seq':
            test = pbmc1
            train = anndata.concat([data8k])

        elif testlabel == 'microarray':
            test = microarray
            train = anndata.concat([data8k,pbmc1])


        train_y = train.obs.iloc[:,:-2].values
        test_y = test.obs.iloc[:,:-2].values

        #### variance cut off

        label = train.X.var(axis=0) > 0.1
        train = train[:,label]
        label = test.X.var(axis=0) > 0.01
        test = test[:,label]
        inter = test.var.index.intersection(train.var.index)
        train = train[:,inter]
        test = test[:,inter]



        ####
        return train.X, train_y, test.X, test_y,
test.obs.iloc[:,:-2].index, test.obs.iloc[:,:-2].columns
```

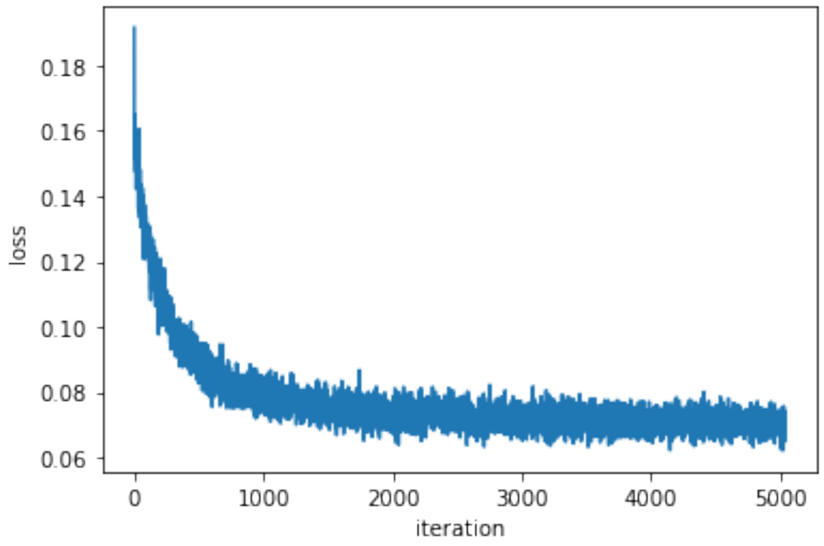
```
train_x, train_y, test_x, test_y, index, celltypes =
preprocess('pbmc_data.h5ad',testlabel='seq')
train_x = np.log(train_x + 1)
test_x = np.log(test_x + 1)

test = test_x[:,np.std(test_x,axis=0) > 0]
```

```
train = train_x[:,np.std(test_x,axis=0) > 0]
test = test[:,np.std(train,axis=0) > 0]
train = train[:,np.std(train,axis=0) > 0]


mms = MinMaxScaler()
test_x = mms.fit_transform(test_x.T).T
train_x = mms.fit_transform(train_x.T).T
```

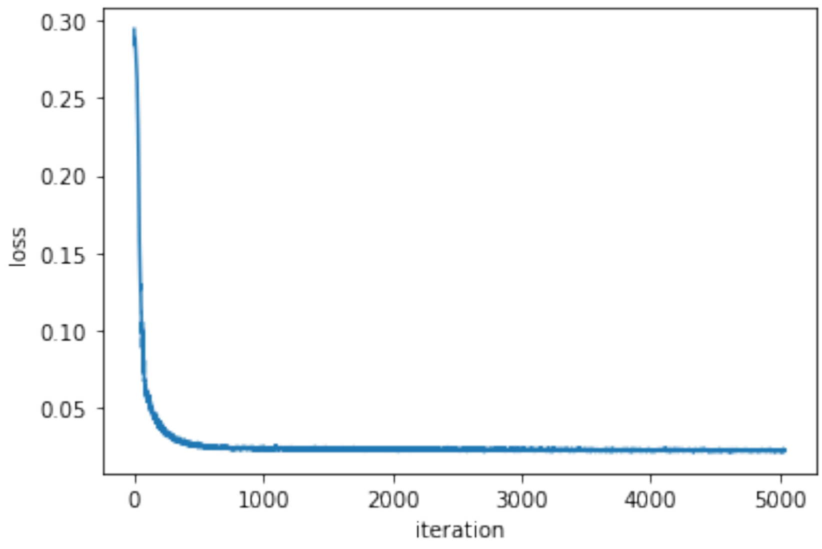
```
print(torch.__version__)
print(device)

for i in range(5):
    pred = testTAPE(train_x, train_y, test_x, test_y, seed=i)
    pred = pd.DataFrame(pred,index=index,columns=celltypes)
```

```
1.10.2+cu113
cuda
```

```
100%|██████████| 80/80 [01:40<00:00,  1.26s/it]

prediction loss is:
```

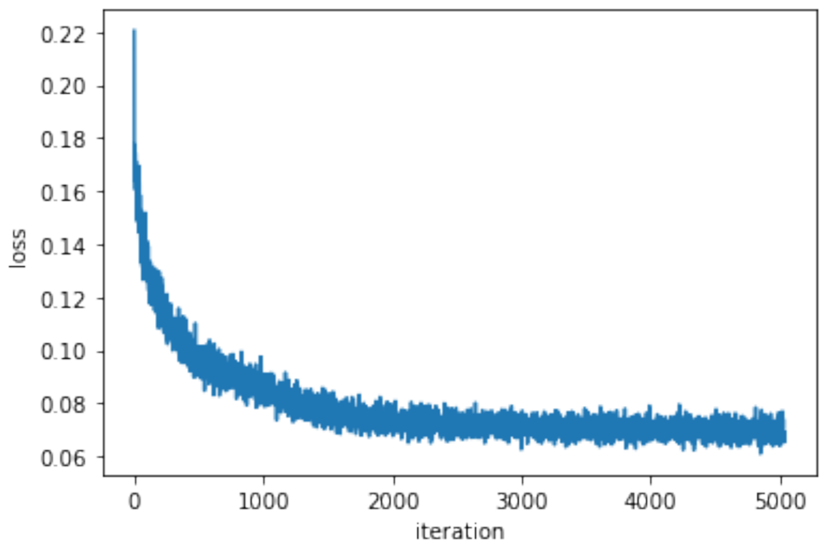


```
reconstruction loss is:
```

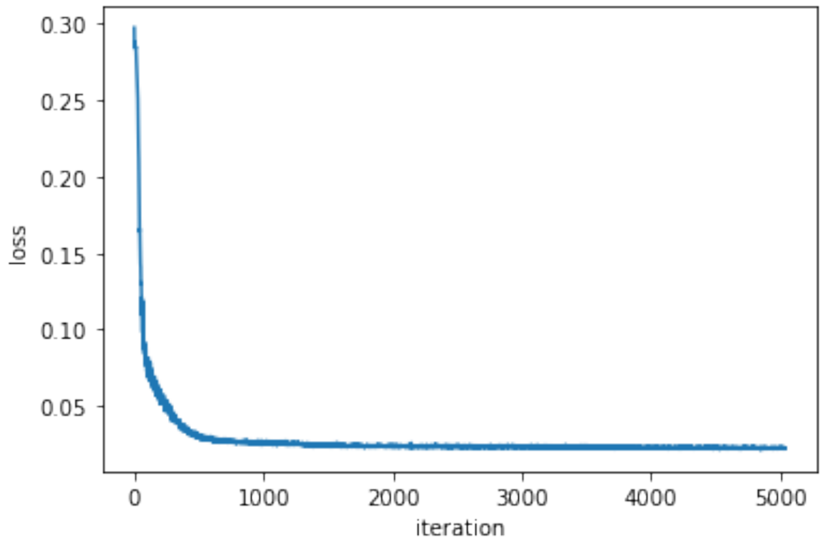


```
0.07734104049870642 0.6798772319374665
```

```
100%|██████████| 80/80 [01:40<00:00,  1.26s/it]

prediction loss is:
```

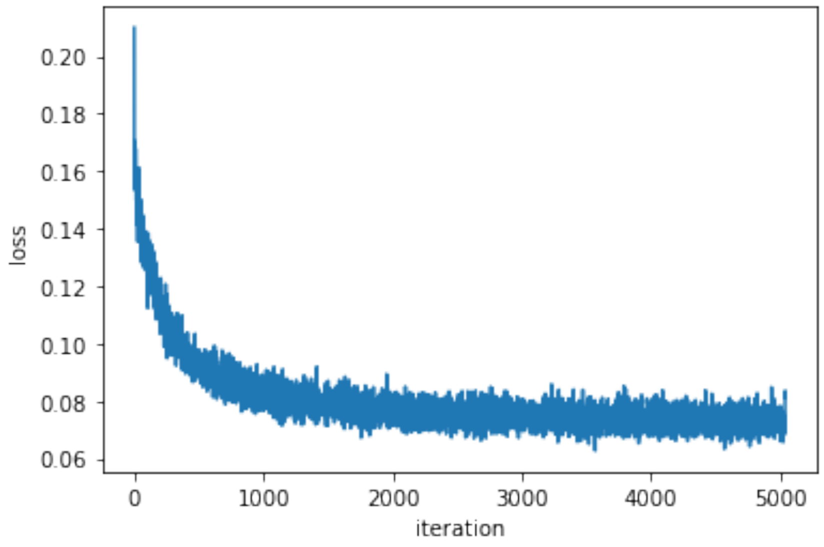


```
reconstruction loss is:
```

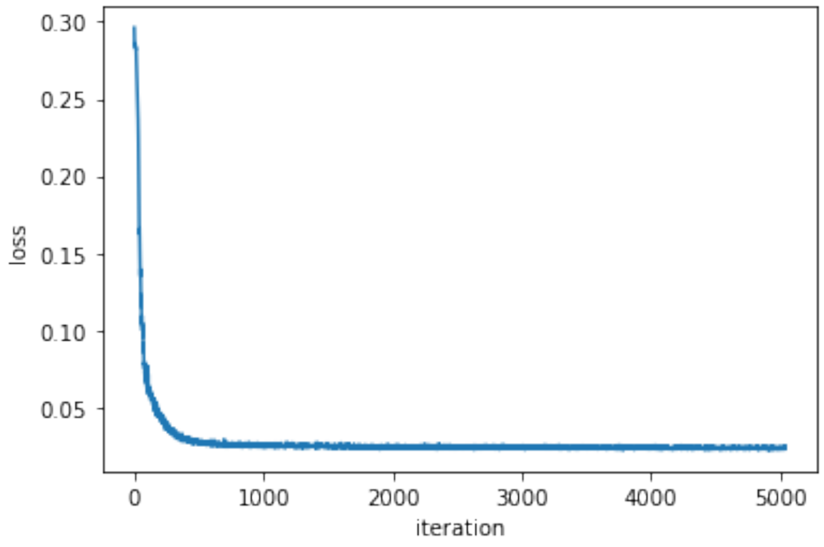


```
0.06512235086825159 0.7543057977405571
```

```
100%|██████████| 80/80 [01:39<00:00,  1.24s/it]

prediction loss is:
```

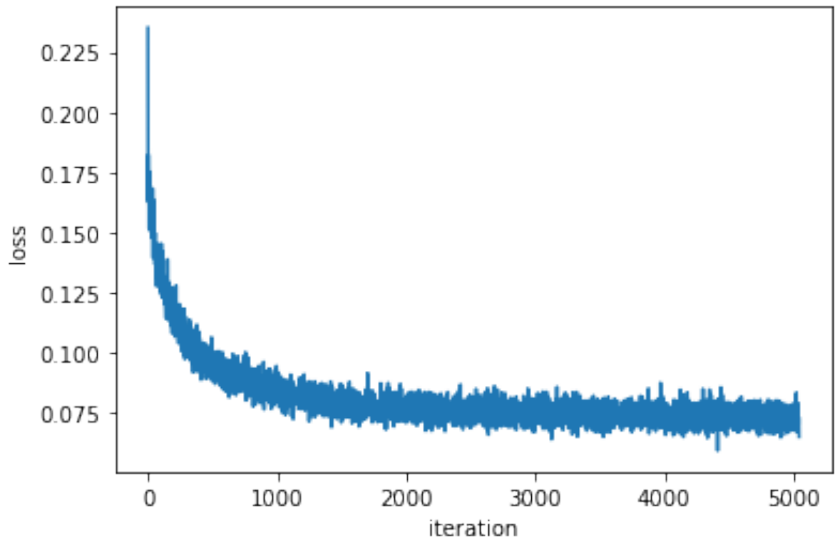


```
reconstruction loss is:
```

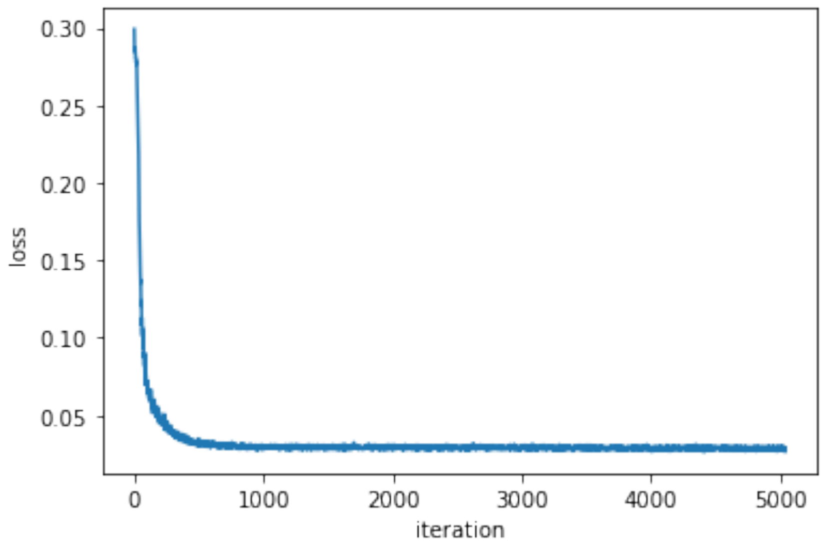


```
0.0669786099156158 0.7734624563816469
```

```
100%|██████████| 80/80 [01:39<00:00,  1.24s/it]

prediction loss is:
```

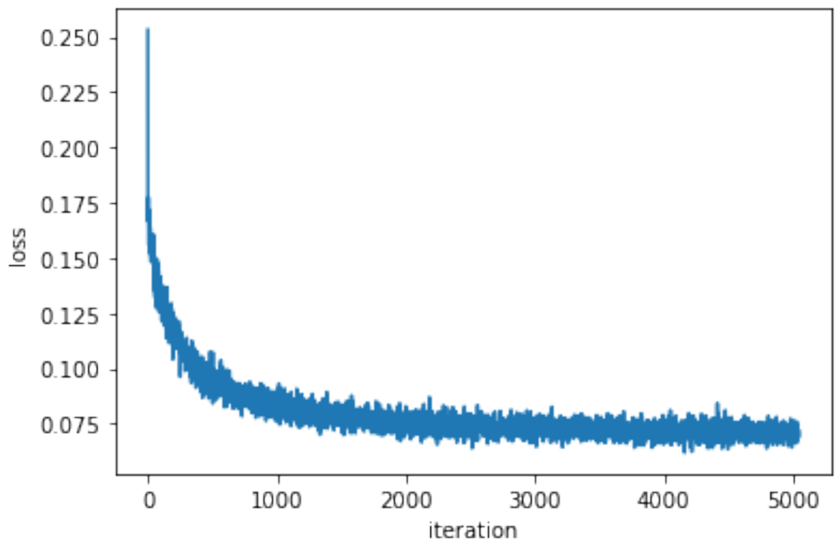


```
reconstruction loss is:
```

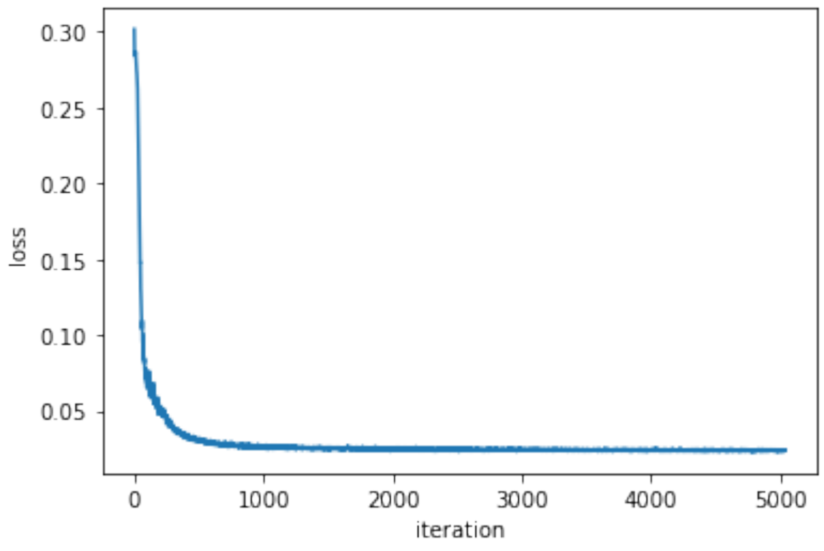


```
0.060259370505768385 0.7587068508933345
```

```
100%|██████████| 80/80 [01:39<00:00,  1.24s/it]

prediction loss is:
```



```
reconstruction loss is:
```

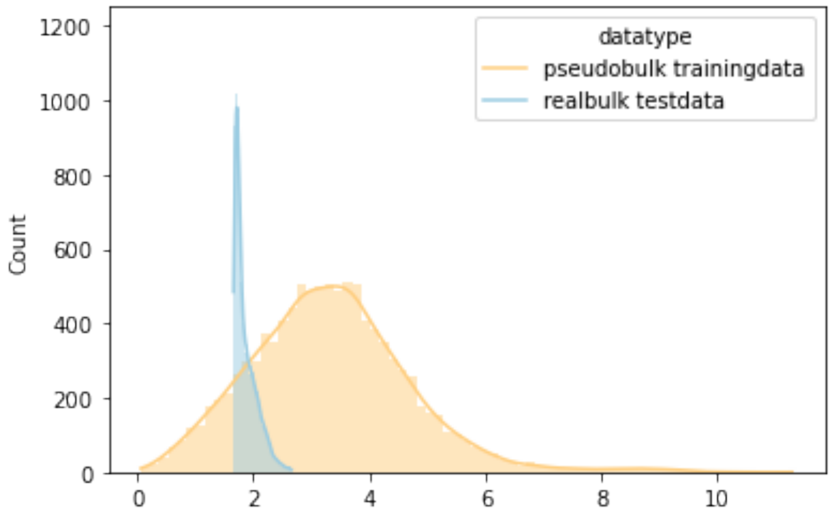


```
0.05720946095422971 0.8402414476637183
```

```
# # train_x, train_y, test_x, test_y =
preprocess('pbmc_data.h5ad','TPMPBMC.txt','PBMC2.csv')
# train_x, train_y, test_x, test_y, index, celltypes =
preprocess('pbmc_data.h5ad',testlabel='microarray')
```

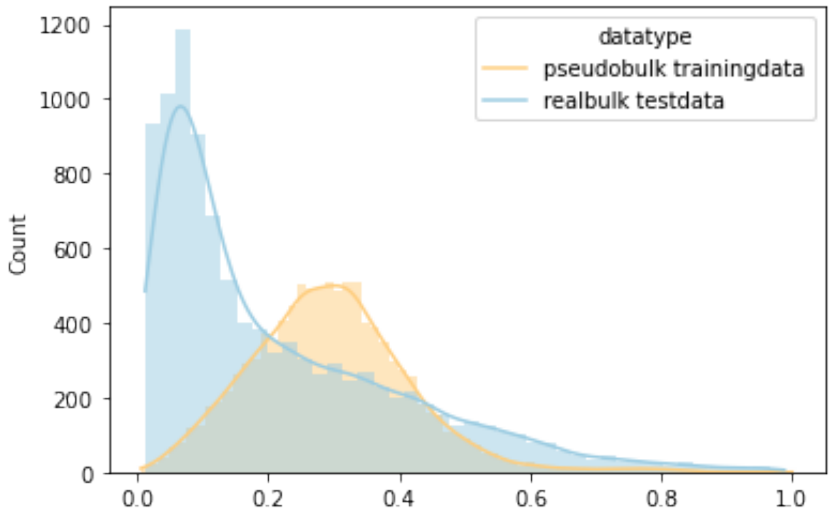
```
# train_x = np.log(train_x + 1)
# test_x = np.log(test_x + 1)
# fig = plt.figure()
#
sns.histplot(data=np.mean(train_x,axis=0),kde=True,color=colors[3],edgec
olor=None)
#
sns.histplot(data=np.mean(test_x,axis=0),kde=True,color=colors[7],edgeco
lor=None)
# plt.legend(title='datatype',labels=['pseudobulk
trainingdata','realbulk testdata'])
# fig.savefig('./figures/microarray_raw.eps',dpi=300)
# plt.show()
# test = test_x[:,np.std(test_x,axis=0) > 0]
# train = train_x[:,np.std(test_x,axis=0) > 0]
# test = test[:,np.std(train,axis=0) > 0]
# train = train[:,np.std(train,axis=0) > 0]
# print(np.sum(np.std(test,axis=0)/np.mean(test,axis=0)))
# print(np.sum(np.std(train,axis=0)/np.mean(train,axis=0)))
# print(np.mean(np.std(test,axis=1)/np.mean(test,axis=1)))
# print(np.mean(np.std(train,axis=1)/np.mean(train,axis=1)))



# mms = MinMaxScaler()
# test_x = mms.fit_transform(test_x.T).T
# train_x = mms.fit_transform(train_x.T).T
# fig = plt.figure()
#
sns.histplot(data=np.mean(train_x,axis=0),kde=True,color=colors[3],edgec
olor=None)
#
sns.histplot(data=np.mean(test_x,axis=0),kde=True,color=colors[7],edgeco
lor=None)
# plt.legend(title='datatype',labels=['pseudobulk
trainingdata','realbulk testdata'])
# fig.savefig('./figures/microarray_mms.eps',dpi=300)
# plt.show()
# test = test_x[:,np.std(test_x,axis=0) > 0]
# train = train_x[:,np.std(test_x,axis=0) > 0]
# test = test[:,np.std(train,axis=0) > 0]
# train = train[:,np.std(train,axis=0) > 0]
# print(np.sum(np.std(test,axis=0)/np.mean(test,axis=0)))
# print(np.sum(np.std(train,axis=0)/np.mean(train,axis=0)))
# print(np.mean(np.std(test,axis=1)/np.mean(test,axis=1)))
# print(np.mean(np.std(train,axis=1)/np.mean(train,axis=1)))




# print(np.sum(np.isnan(test_x)))
# print('train_x shape',train_x.shape)
# print('train_y shape',train_y.shape)
# print('test_x shape',test_x.shape)
# print('test_y shape',test_y.shape)
```

```
The PostScript backend does not support transparency; partially
transparent artists will be rendered opaque.
```



```
232.77246
1802.7124
0.10684365
0.4356013
```

```
The PostScript backend does not support transparency; partially
transparent artists will be rendered opaque.
```

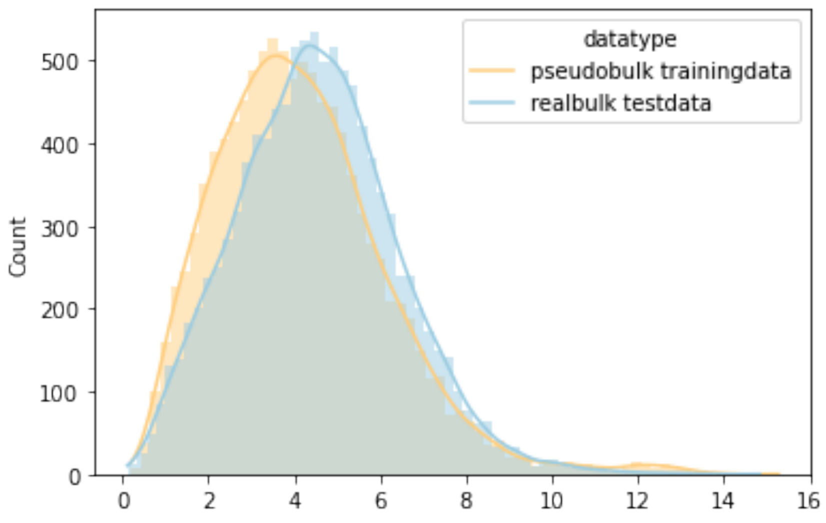


```
3218.82
1731.368
0.9050175
0.43560132
0
train_x shape (8012, 10193)
train_y shape (8012, 6)
test_x shape (20, 10193)
test_y shape (20, 6)
```

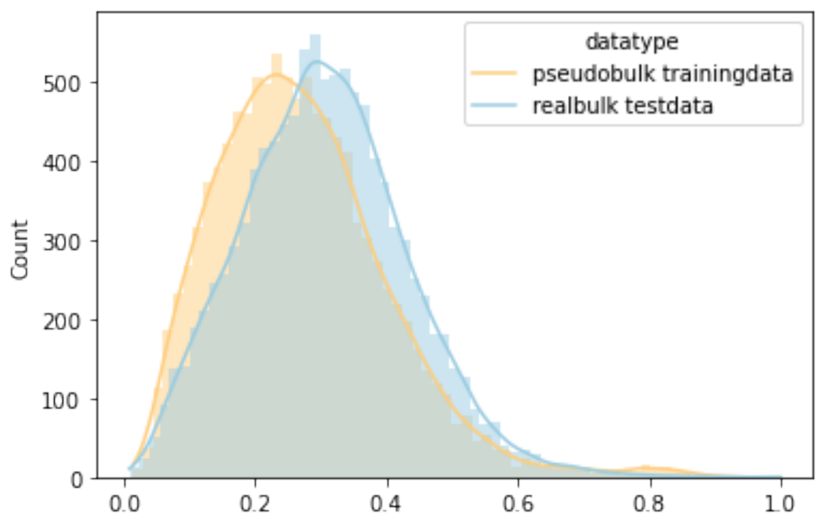
```
# print(torch.__version__)
# print(device)
# microarray = np.zeros((50,2))
# for i in range(5):
#     pred = testTAPE(train_x, train_y, test_x, test_y, seed=42)
#     pred = pd.DataFrame(pred,index=index,columns=celltypes)
```

```
# train_x, train_y, test_x, test_y, index, celltypes =
preprocess('pbmc_data.h5ad','monaco_pbmc.txt','monaco_pbmc_truth.csv')
# train_x = np.log2(train_x + 1)
# test_x = np.log2(test_x + 1)

# fig = plt.figure()
#
sns.histplot(data=np.mean(train_x,axis=0),kde=True,color=colors[3],edgec
olor=None)
#
sns.histplot(data=np.mean(test_x,axis=0),kde=True,color=colors[7],edgeco
lor=None)
# plt.legend(title='datatype',labels=['pseudobulk
trainingdata','realbulk testdata'])
# fig.savefig('./figures/monaco_raw.eps',dpi=300)
# plt.show()
# test = test_x[:,np.std(test_x,axis=0) > 0]
# train = train_x[:,np.std(test_x,axis=0) > 0]
# test = test[:,np.std(train,axis=0) > 0]
# train = train[:,np.std(train,axis=0) > 0]


# mms = MinMaxScaler()
# test_x = mms.fit_transform(test_x.T).T
# train_x = mms.fit_transform(train_x.T).T
# fig = plt.figure()
#
sns.histplot(data=np.mean(train_x,axis=0),kde=True,color=colors[3],edgec
olor=None)
#
sns.histplot(data=np.mean(test_x,axis=0),kde=True,color=colors[7],edgeco
lor=None)
# plt.legend(title='datatype',labels=['pseudobulk
trainingdata','realbulk testdata'])
# fig.savefig('./figures/monaco_mms.eps',dpi=300)
# plt.show()
# test = test_x[:,np.std(test_x,axis=0) > 0]
# train = train_x[:,np.std(test_x,axis=0) > 0]
# test = test[:,np.std(train,axis=0) > 0]
# train = train[:,np.std(train,axis=0) > 0]
```

```
The PostScript backend does not support transparency; partially
transparent artists will be rendered opaque.
```



```
The PostScript backend does not support transparency; partially
transparent artists will be rendered opaque.
```



```
# print(torch.__version__)
# print(device)

# for i in range(5):
#     pred = testTAPE(train_x, train_y, test_x, test_y, seed=i)
#     pred = pd.DataFrame(pred,index=index,columns=celltypes)
```

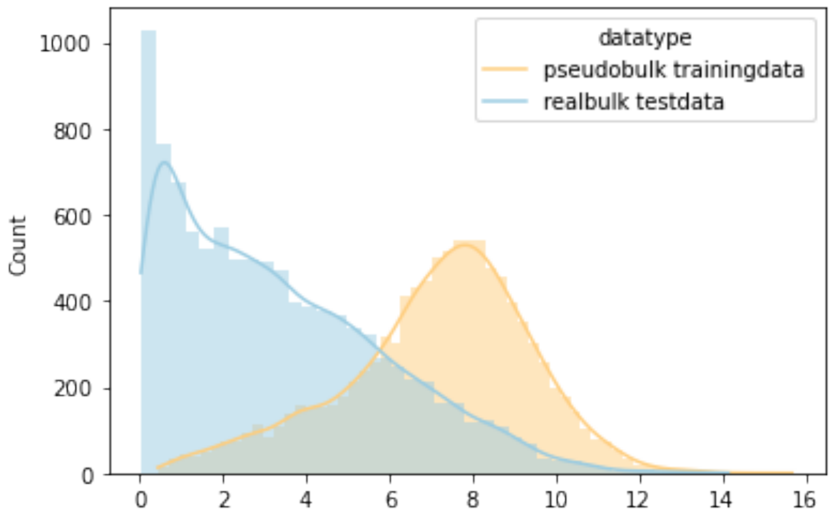
```
# pred.to_csv('monaco_pred.csv')
```

```
# def preprocess(trainingdatapath, testx, testy):
#     # mouse brain dataset
#     trainset = read_h5ad(trainingdatapath)
#     print(trainset.obs.columns)
#     testset = pd.read_csv(testx,index_col=0)
#     #display(testset)
#     testlabel = pd.read_csv(testy,index_col=0)

#     #display(testlabel)
#     test_y = testlabel.values
#     ### refraction
#     for i, values in enumerate(test_y):
#         r_sum = np.sum(values)
#         if r_sum == 0:
#             pass
#         else:
#             test_y[i] = test_y[i] / r_sum
#     ### find intersect genes
#     intersection_genes =
list(testset.index.intersection(trainset.var.index))
#     print(len(intersection_genes))
#     test_x = testset.loc[intersection_genes]
#     test_x = test_x.T.values
#     simuvar = list(trainset.var.index)
#     intersection_gene_position = []
#     for gene in intersection_genes:
#         intersection_gene_position.append(simuvar.index(gene))
#     selected = np.zeros((len(intersection_genes), len(trainset.X)))
#     for i in range(selected.shape[0]):
#         selected[i] = trainset.X.T[intersection_gene_position[i]]
#     train_x = selected.T


#     # merge ex&in-neurons
#     #trainset.obs['Neurons'] =
trainset.obs['ExNeurons']+trainset.obs['InNeurons']
#     # find intersect cell proportions
#     intersection_cell =
testlabel.columns.intersection(trainset.obs.columns)
#     print(intersection_cell)
#     print(testlabel.columns)
#     train_y = trainset.obs[intersection_cell].values
#     ### refraction
#     for i, values in enumerate(train_y):
#         r_sum = np.sum(values)
#         if r_sum == 0:
#             pass
#         else:
#             train_y[i] = train_y[i] / r_sum
#     ### variance cutoff
#     label = test_x.var(axis=0) > 0.1
#     test_x_new = np.zeros((test_x.shape[0],np.sum(label)))
#     train_x_new = np.zeros((train_x.shape[0],np.sum(label)))
#     k = 0
#     for i in range(len(label)):
#         if label[i] == True:
#             test_x_new[:,k] = test_x[:,i]
#             train_x_new[:,k] = train_x[:,i]
#             k += 1

#     return train_x_new, train_y, test_x_new, test_y, testlabel.index,
testlabel.columns
```

```
# train_x, train_y, test_x, test_y, index, celltypes =
preprocess(trainingdatapath='mouse_brain.h5ad',
#
testx='ROSMAP_mouse2human_GEP.csv',
#
testy='ROSMAP_IHC_fractions.csv')
# train_x = np.log2(train_x + 1)
# test_x = np.log2(test_x + 1)
# fig = plt.figure()
#
sns.histplot(data=np.mean(train_x,axis=0),kde=True,color=colors[3],edgec
olor=None)
#
sns.histplot(data=np.mean(test_x,axis=0),kde=True,color=colors[7],edgeco
lor=None)
# plt.legend(title='datatype',labels=['pseudobulk
trainingdata','realbulk testdata'])
# fig.savefig('./figures/ROSMAPm_raw.eps',dpi=300)
# plt.show()
# test = test_x[:,np.std(test_x,axis=0) > 0]
# train = train_x[:,np.std(test_x,axis=0) > 0]
# test = test[:,np.std(train,axis=0) > 0]
# train = train[:,np.std(train,axis=0) > 0]
# print(np.sum(np.std(test,axis=0)/np.mean(test,axis=0)))
# print(np.sum(np.std(train,axis=0)/np.mean(train,axis=0)))
# print(np.mean(np.std(test,axis=1)/np.mean(test,axis=1)))
# print(np.mean(np.std(train,axis=1)/np.mean(train,axis=1)))

# mms = MinMaxScaler()
# test_x = mms.fit_transform(test_x.T)
# test_x = test_x.T
# train_x = mms.fit_transform(train_x.T)
# train_x = train_x.T
# fig = plt.figure()
#
sns.histplot(data=np.mean(train_x,axis=0),kde=True,color=colors[3],edgec
olor=None)
#
sns.histplot(data=np.mean(test_x,axis=0),kde=True,color=colors[7],edgeco
lor=None)
# plt.legend(title='datatype',labels=['pseudobulk
trainingdata','realbulk testdata'])
# fig.savefig('./figures/ROSMAPm_mms.eps',dpi=300)
# plt.show()
# test = test_x[:,np.std(test_x,axis=0) > 0]
# train = train_x[:,np.std(test_x,axis=0) > 0]
# test = test[:,np.std(train,axis=0) > 0]
# train = train[:,np.std(train,axis=0) > 0]
# print(np.sum(np.std(test,axis=0)/np.mean(test,axis=0)))
# print(np.sum(np.std(train,axis=0)/np.mean(train,axis=0)))
# print(np.mean(np.std(test,axis=1)/np.mean(test,axis=1)))
# print(np.mean(np.std(train,axis=1)/np.mean(train,axis=1)))

# print('train_x shape',train_x.shape)
# print('train_y shape',train_y.shape)
# print('test_x shape',test_x.shape)
# print('test_y shape',test_y.shape)
# print(torch.__version__)
# print(device)

# # mouse = np.zeros((50,2))
# # for i in range(5):
# #     pred = testTAPE(train_x, train_y, test_x, test_y, seed=i)
# #     pred = pd.DataFrame(pred,columns=celltypes,index=index)
```

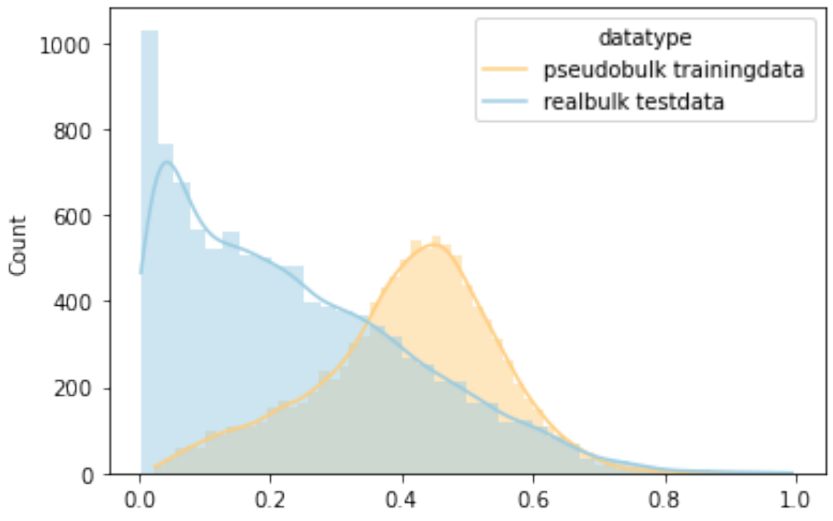
```
Index(['Neurons', 'Astrocytes', 'VLMC', 'Tanycytes', 'OPC',
'Oligodendrocytes',
       'Ependymal', 'Unknown', 'Endothelial', 'Microglia', 'NFO', 'ds',
       'batch'],
      dtype='object')
10836
Index(['Astrocytes', 'Endothelial', 'Microglia', 'Neurons',
       'Oligodendrocytes'],
      dtype='object')
Index(['Astrocytes', 'Endothelial', 'Microglia', 'Neurons',
       'Oligodendrocytes'],
      dtype='object')
```

```
The PostScript backend does not support transparency; partially
transparent artists will be rendered opaque.
```



```
7698.044263706695
6171.861243550134
0.8298702702337726
0.403497424427778
```

```
The PostScript backend does not support transparency; partially
transparent artists will be rendered opaque.
```



```
7652.56512728712
3869.8214382526744
0.8298702702337769
0.403497424427778
train_x shape (30000, 10184)
train_y shape (30000, 5)
test_x shape (41, 10184)
test_y shape (41, 5)
1.10.2+cu113
cuda
```

```
# pred.to_csv('ROSMAPm_selected_pred.csv')
```

```
# def preprocess(trainingdatapath, testx, testy):
#     # mouse brain dataset
#     trainset = read_h5ad(trainingdatapath)
#     print(trainset.obs.columns)
#     testset = pd.read_csv(testx,index_col=0)
#     #display(testset)
#     testlabel = pd.read_csv(testy,index_col=0)
#     #display(testlabel)
#     test_y = testlabel.values
#     ### refraction
#     for i, values in enumerate(test_y):
#         r_sum = np.sum(values)
#         if r_sum == 0:
#             pass
#         else:
#             test_y[i] = test_y[i] / r_sum
#     ### find intersect genes
#     intersection_genes =
list(testset.index.intersection(trainset.var.index))
#     print(len(intersection_genes))
#     test_x = testset.loc[intersection_genes]
#     test_x = test_x.T.values
#     simuvar = list(trainset.var.index)
#     intersection_gene_position = []
#     for gene in intersection_genes:
#         intersection_gene_position.append(simuvar.index(gene))
#     selected = np.zeros((len(intersection_genes), len(trainset.X)))
#     for i in range(selected.shape[0]):
```

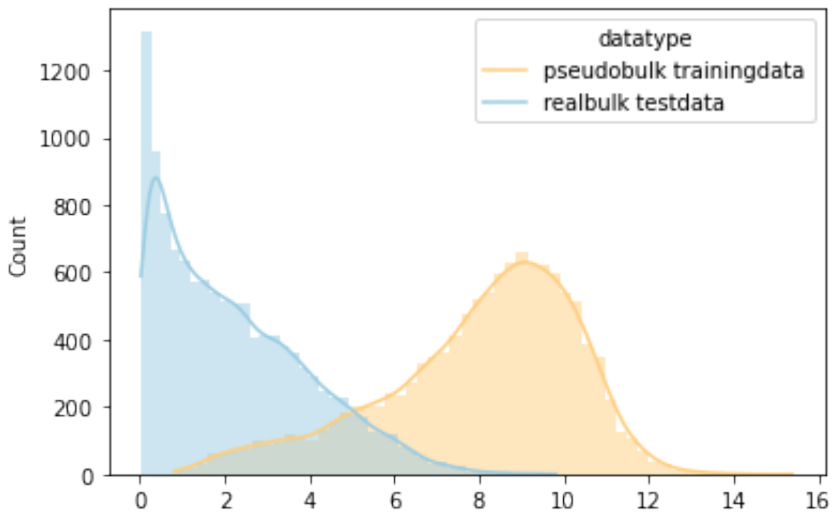
```
#         selected[i] = trainset.X.T[intersection_gene_position[i]]
#     train_x = selected.T


#     # merge ex&in-neurons
#     trainset.obs['Neurons'] =
trainset.obs['ExNeurons']+trainset.obs['InNeurons']
#     # find intersect cell proportions
#     intersection_cell =
testlabel.columns.intersection(trainset.obs.columns)
#     print(intersection_cell)
#     print(testlabel.columns)
#     train_y = trainset.obs[intersection_cell].values
#     ### refraction
#     for i, values in enumerate(train_y):
#         r_sum = np.sum(values)
#         if r_sum == 0:
#             pass
#         else:
#             train_y[i] = train_y[i] / r_sum
#     ### variance cutoff
#     label = test_x.var(axis=0) > 0.1
#     test_x_new = np.zeros((test_x.shape[0],np.sum(label)))
#     train_x_new = np.zeros((train_x.shape[0],np.sum(label)))
#     k = 0
#     for i in range(len(label)):
#         if label[i] == True:
#             test_x_new[:,k] = test_x[:,i]
#             train_x_new[:,k] = train_x[:,i]
#             k += 1

#     return train_x_new, train_y, test_x_new, test_y, testlabel.index,
testlabel.columns
```

```
# train_x, train_y, test_x, test_y, index, celltypes =
preprocess(trainingdatapath='../data/tape/humanbrain_ref.h5ad',
#
testx='ROSMAP_human_GEP.csv',
#
testy='ROSMAP_IHC_fractions.csv')
# train_x = np.log(train_x + 1)
# test_x = np.log(test_x + 1)
# fig = plt.figure()
```

```
#
sns.histplot(data=np.mean(train_x,axis=0),kde=True,color=colors[3],edgec
olor=None)
#
sns.histplot(data=np.mean(test_x,axis=0),kde=True,color=colors[7],edgeco
lor=None)
# plt.legend(title='datatype',labels=['pseudobulk
trainingdata','realbulk testdata'])
# fig.savefig('./figures/ROSMAPh_raw.eps',dpi=300)
# plt.show()
# test = test_x[:,np.std(test_x,axis=0) > 0]
# train = train_x[:,np.std(test_x,axis=0) > 0]
# test = test[:,np.std(train,axis=0) > 0]
# train = train[:,np.std(train,axis=0) > 0]
# print(np.sum(np.std(test,axis=0)/np.mean(test,axis=0)))
# print(np.sum(np.std(train,axis=0)/np.mean(train,axis=0)))
# print(np.mean(np.std(test,axis=1)/np.mean(test,axis=1)))
# print(np.mean(np.std(train,axis=1)/np.mean(train,axis=1)))

# mms = MinMaxScaler()
# test_x = mms.fit_transform(test_x.T)
# test_x = test_x.T
# train_x = mms.fit_transform(train_x.T)
# train_x = train_x.T
# fig = plt.figure()
#
sns.histplot(data=np.mean(train_x,axis=0),kde=True,color=colors[3],edgec
olor=None)
#
sns.histplot(data=np.mean(test_x,axis=0),kde=True,color=colors[7],edgeco
lor=None)
# plt.legend(title='datatype',labels=['pseudobulk
trainingdata','realbulk testdata'])
# fig.savefig('./figures/ROSMAPh_mms.eps',dpi=300)
# plt.show()
# test = test_x[:,np.std(test_x,axis=0) > 0]
# train = train_x[:,np.std(test_x,axis=0) > 0]
# test = test[:,np.std(train,axis=0) > 0]
# train = train[:,np.std(train,axis=0) > 0]
# print(np.sum(np.std(test,axis=0)/np.mean(test,axis=0)))
# print(np.sum(np.std(train,axis=0)/np.mean(train,axis=0)))
# print(np.mean(np.std(test,axis=1)/np.mean(test,axis=1)))
# print(np.mean(np.std(train,axis=1)/np.mean(train,axis=1)))
# print('train_x shape',train_x.shape)
# print('train_y shape',train_y.shape)
# print('test_x shape',test_x.shape)
# print('test_y shape',test_y.shape)
# print(torch.__version__)
# print(device)

# # for i in range(5):
# #     pred = testTAPE(train_x, train_y, test_x, test_y, seed=i)
# #     pred = pd.DataFrame(pred,columns=celltypes,index=index)
```

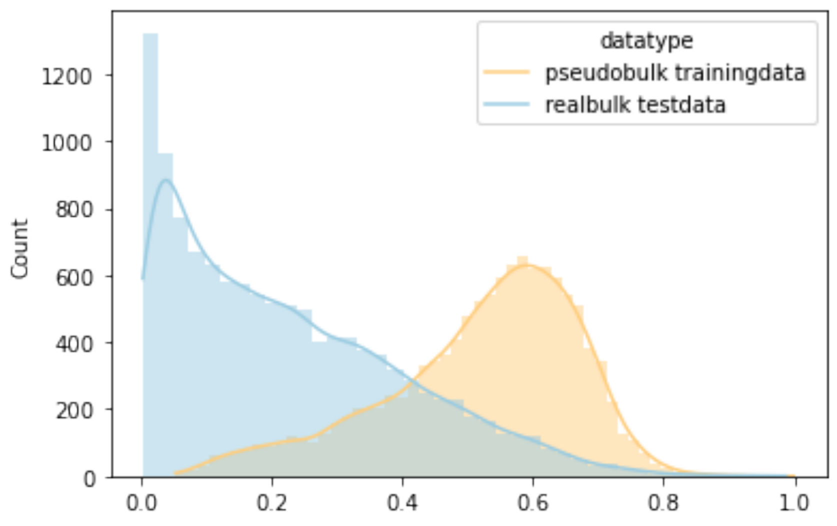
```
Index(['Astrocytes', 'Endothelial', 'ExNeurons', 'InNeurons',
'Microglia',
       'OPC', 'Oligodendrocytes', 'Unknown'],
      dtype='object')
12905
Index(['Astrocytes', 'Endothelial', 'Microglia', 'Neurons',
       'Oligodendrocytes'],
      dtype='object')
Index(['Astrocytes', 'Endothelial', 'Microglia', 'Neurons',
       'Oligodendrocytes'],
      dtype='object')
```

```
The PostScript backend does not support transparency; partially
transparent artists will be rendered opaque.
```



```
9538.103188433588
1577.0344694189534
0.8595327937042089
0.3092190243958002
```

```
The PostScript backend does not support transparency; partially
transparent artists will be rendered opaque.
```



```
9472.11858335965
1583.3697784675433
0.8595327937042135
0.3092201795676371
train_x shape (10000, 11897)
train_y shape (10000, 5)
test_x shape (41, 11897)
test_y shape (41, 5)
1.10.2+cu113
cuda
```

```
# pred.to_csv('ROSMAPh_selected_pred.csv')
```