The following code shows the analyses of genomap and genoNet for five different tasks on four sets of scRNA-seq data

In [1]:
```
# For the tasks below, four datasets analysed in the manuscript will be automatically
loaded.
# However, you can upload your own dataset, create genomaps and train the supervised or
unsupervised
# genoNet models for different tasks.
# Our data were saved as .mat file to reduce the data size (normally .csv file needs
more disk space).
# However, .csv files can also be loaded in the way shown in the third section
```

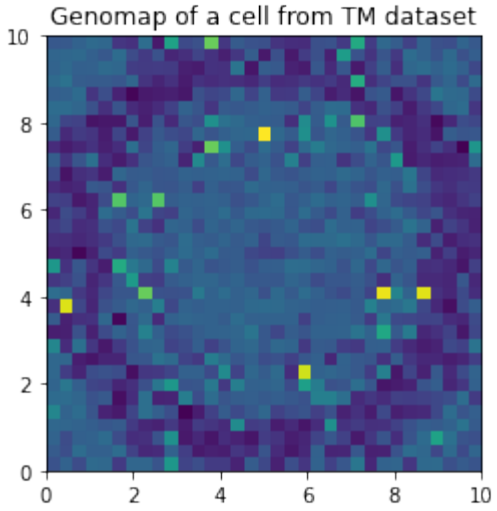
In [2]:
```
# Load all necessary python packages needed for the reported analyses
# in our manuscript

%matplotlib inline

import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import scipy
import scipy.io as sio
import numpy as np
import scipy.io as sio
import genoNet as gNet
import os
import torch
from torch.utils.data import DataLoader, Dataset
from pytorch_grad_cam import GradCAM
from pytorch_grad_cam.utils.model_targets import ClassifierOutputTarget
from tqdm import tqdm
import sklearn
import phate
from sklearn.manifold import TSNE
from pyclustering.cluster.xmeans import xmeans
from pyclustering.cluster.center_initializer import kmeans_plusplus_initializer
from pyclustering.cluster import cluster_visualizer

from genoNet import geneDataset,_get_device,load_genoNet,predict,traingenoNet,rescale
from genomap import construct_genomap

# For reproducibility
torch.manual_seed(1)
torch.use_deterministic_algorithms(True)
```

Create genomaps from Tabula Muris (TM) scRNA-seq dataset

In [3]:
```
# First, we load the TM data. Data should be in cells X genes format,
# i.e., each row should correspond to gene expressions of a cell and each column
# should represent the expression value of a specific gene.
# The original TM data is very large with 54,865 cells and 19,791 genes.
# We select the 1089 most variable genes from the data to reduce its size.
# The reason behind chosing 1089 is that it is a square number (33*33)
# However, you can choose any other number. The following commented 4-lines code
```

```
# allows one to select the most variable genes (numGene=1089 was used in our anlaysis)
# varX=np.var(dataFull,axis=0)
# idxV=np.argsort(varX)
# idxVX=np.flip(idxV)
# dataReduced=dataFull[:,idxVX[0:numGene]]

data = pd.read_csv('../data/TMdata.csv', header=None,
                   delim_whitespace=False)

# Creation of genomaps
# Selection of row and column number of the genomaps
# To create square genomaps, the row and column numbers are set to be the same.
colNum=33
rowNum=33
# Normalization of the data
dataNorm=scipy.stats.zscore(data,axis=0,ddof=1)
# Construction of genomaps
genoMaps=construct_genomap(dataNorm,rowNum,colNum,epsilon=0.0,num_iter=200)

# Visualization of the constructed genomaps:
# The "genoMaps" array: All the constructed genomaps are saved in the array. This
# array has four indices. The first one indexes the series of genomaps.
# The second and third ones denote the row and column number in a genomap.
# The fourth index is introduced to facillitate the caclulation using Pytorch or
Tensorflow
# to visualize  the i-th genomap, set the first index variable to i (i=10 here)
findI=genoMaps[10,:,:,:]
plt.figure(1)
plt.imshow(findI, origin = 'lower',  extent = [0, 10, 0, 10], aspect = 1)
plt.title('Genomap of a cell from TM dataset')
```

Out[3]:`Text(0.5, 1.0, 'Genomap of a cell from TM dataset')`



Train and test the genoNet on the genomaps of TM dataset for cell recognition

In [4]:
```
# Load ground truth cell labels of the TM dataset
gt_data = sio.loadmat('../data/GT_TM.mat')
GT = np.squeeze(gt_data['GT'])
```

```
    # Load the indices of training and test data (70% training, 30% testing data)
    index_data = sio.loadmat('../data/index_TM.mat')
    indxTest = np.squeeze(index_data['indxTest'])
    indxTrain = np.squeeze(index_data['indxTrain'])

    GT = GT - 1 # to ensure the labels begin with 0 to conform with PyTorch

    # Split the data for training and testing
    dataMat_CNNtrain = genoMaps[indxTrain-1,:,:,:] # to ensure the index starts at 0 to
    conform with
    # python indexing
    dataMat_CNNtest = genoMaps[indxTest-1,:,:,:]
    groundTruthTest = GT[indxTest-1]
    groundTruthTrain = GT[indxTrain-1]
    classNum = len(np.unique(groundTruthTrain))

    # Preparation of training and testing data for PyTorch computation
    XTrain = dataMat_CNNtrain.transpose([0,3,1,2])
    XTest = dataMat_CNNtest.transpose([0,3,1,2])
    yTrain = groundTruthTrain
    yTest = groundTruthTest


    # Train the network in PyTorch framework
    miniBatchSize = 128
    net = gNet.traingenoNet(XTrain, yTrain, maxEPOCH=150, batchSize=miniBatchSize,
    verbose=True)
    torch.save(net.state_dict(),'../results/genoNet_TM.pt')

    # Process the test data
    testset = gNet.geneDataset(XTest, yTest)
    testloader = gNet.DataLoader(testset, batch_size=miniBatchSize, shuffle=False)
    device = gNet._get_device()

    # Perform cell classification/reocognition
    prob_test = gNet.predict(net, testloader, device)
    pred_test = np.argmax(prob_test, axis=-1)

    # Compute the accuracy of cell classification/reocognition
    print('Classification accuracy of genomap+genoNet for TM
    dataset:'+str(np.sum(pred_test==yTest) / pred_test.shape[0]))
```

```
0: 1.0263555290493063
10: 0.11414843579240613
20: 0.07318498558479677
30: 0.035157931777122634
40: 0.04290943306853741
50: 0.0321918042904374
60: 0.0308683146599697
70: 0.0171414141255217
80: 0.009432878830661398
90: 0.007115020575732816
100: 0.01441603327154411
110: 0.022488392139677182
120: 0.01007968801951752
130: 0.021043601193864756
140: 0.007578769824992251
```

```
Classification accuracy of genomap+genoNet for TM dataset:0.9155425932677117
```

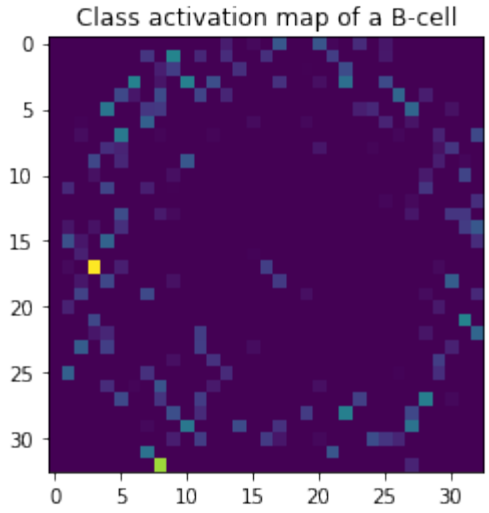
Discovery of the cell- and class-specific gene sets from TM dataset

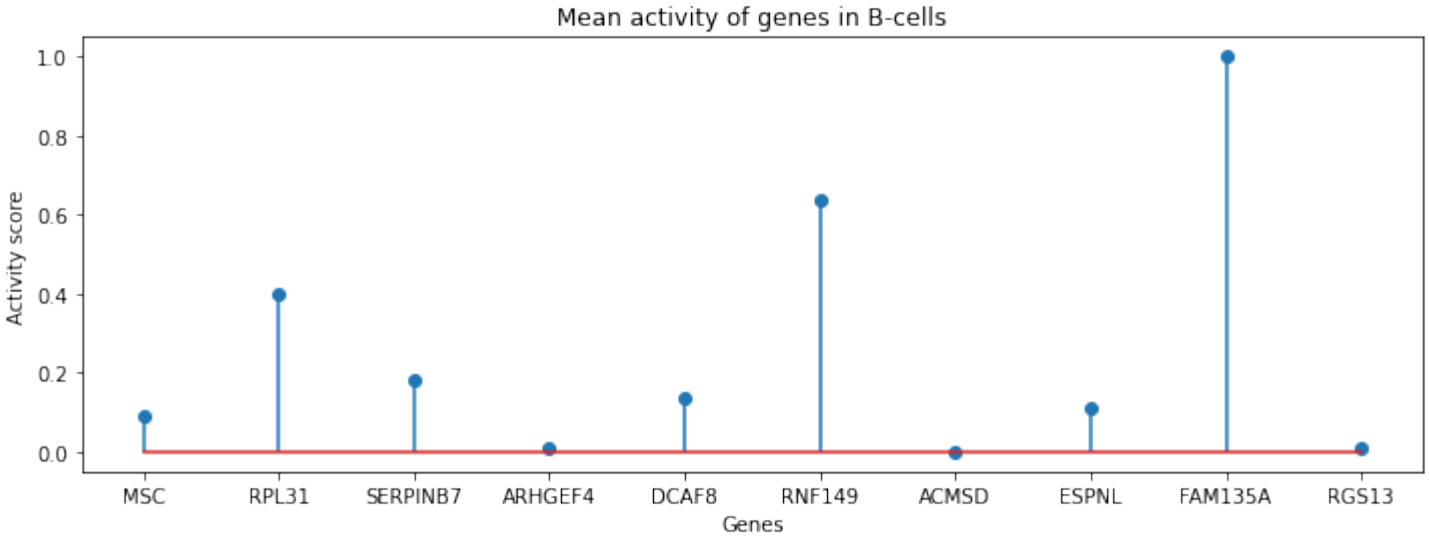
In [5]:
```
# The trained genoNet in the last section is used here to find the cell- and class-
specific
# gene sets. We use the GRAD-CAM technique to compute the class activation maps
# of the trained genoNet for given genomaps.
# The class activation map shows which pixels in the genomaps are important
# for decision-making of the genoNet.
# We emphasize that each genomap pixel represents
# a gene and the class activation value of the pixel
# denotes the importance of that particular gene in cell classification
target_layers = [net.conv1]
# Grad cam object
cam = GradCAM(model=net, target_layers=target_layers, use_cuda=False)

# We compute the class activation maps of all the cells
# of a specific cell type to find the class-specific important genes.
# Load the indices of different cell types
gt_data = sio.loadmat('../data/index_GRADCAM.mat')

# Compute class activation maps of B-cells using GRAD-CAM
indxcell=gt_data['idxBcell']
# Create a numpy array to store the grad CAM results
grayscale_cam=np.zeros([indxcell.shape[0],colNum*rowNum])

for i in range(indxcell.shape[0]):
    # Looping over all B-cells to compute the class activation maps
    grad_cam_img = torch.Tensor(dataMat_CNNtrain[i,:,:,:].transpose(2,0,1)
                            .reshape((1,1,colNum,rowNum))).to(device)
    pred_label = torch.argmax(net(grad_cam_img),axis=-1)
    targets = [ClassifierOutputTarget(pred_label)]
    A = cam(input_tensor=grad_cam_img, targets=targets)
    B=np.reshape(A.T,(1,colNum*rowNum))
    grayscale_cam[i,:]=B

# Plot an example activation map of a B-cell, which highlights the genes that are
important
# for the decision made by the genoNet.
plt.figure(2)
plt.imshow(np.squeeze(A))
plt.title('Class activation map of a B-cell')

# Compute the mean of gene activation values across all the B-cells
# to find the average activity of all the genes in B-cells
graySmeanB=np.mean(grayscale_cam,axis=0)


# Repeat the same procedure for T cell, natural killer cells,
# Monocytes, Pneumoctyes and Hematopoitic cells

indxcell=gt_data['idxTcell']
grayscale_cam=np.zeros([indxcell.shape[0],colNum*rowNum])
for i in range(indxcell.shape[0]):
```

```
        grad_cam_img = torch.Tensor(dataMat_CNNtrain[i,:,:,:].transpose(2,0,1)
                                .reshape((1,1,colNum,rowNum))).to(device)
        pred_label = torch.argmax(net(grad_cam_img),axis=-1)
        targets = [ClassifierOutputTarget(pred_label)]
        A = cam(input_tensor=grad_cam_img, targets=targets)
        B=np.reshape(A.T,(1,colNum*rowNum))
        grayscale_cam[i,:]=B

    graySmeanT=np.mean(grayscale_cam,axis=0) # Activity of genes in T-cells

    indxcell=gt_data['idxNK']
    grayscale_cam=np.zeros([indxcell.shape[0],colNum*rowNum])
    for i in range(indxcell.shape[0]):

        grad_cam_img = torch.Tensor(dataMat_CNNtrain[i,:,:,:].transpose(2,0,1)
                                .reshape((1,1,colNum,rowNum))).to(device)
        pred_label = torch.argmax(net(grad_cam_img),axis=-1)
        targets = [ClassifierOutputTarget(pred_label)]
        A = cam(input_tensor=grad_cam_img, targets=targets)
        B=np.reshape(A.T,(1,colNum*rowNum))
        grayscale_cam[i,:]=B

    graySmeanNK=np.mean(grayscale_cam,axis=0) # Activity of genes in natural killer cells

    indxcell=gt_data['idxMono']
    grayscale_cam=np.zeros([indxcell.shape[0],colNum*rowNum])
    for i in range(indxcell.shape[0]):

        grad_cam_img = torch.Tensor(dataMat_CNNtrain[i,:,:,:].transpose(2,0,1)
                                .reshape((1,1,colNum,rowNum))).to(device)
        pred_label = torch.argmax(net(grad_cam_img),axis=-1)
        targets = [ClassifierOutputTarget(pred_label)]
        A = cam(input_tensor=grad_cam_img, targets=targets)
        B=np.reshape(A.T,(1,colNum*rowNum))
        grayscale_cam[i,:]=B

    graySmeanMono=np.mean(grayscale_cam,axis=0) # Activity of genes in Monocytes


    indxcell=gt_data['idxPne']
    grayscale_cam=np.zeros([indxcell.shape[0],colNum*rowNum])
    for i in range(indxcell.shape[0]):

        grad_cam_img = torch.Tensor(dataMat_CNNtrain[i,:,:,:].transpose(2,0,1)
                                .reshape((1,1,colNum,rowNum))).to(device)
        pred_label = torch.argmax(net(grad_cam_img),axis=-1)
        targets = [ClassifierOutputTarget(pred_label)]
        A = cam(input_tensor=grad_cam_img, targets=targets)
        B=np.reshape(A.T,(1,colNum*rowNum))
        grayscale_cam[i,:]=B
    grayscale_cam=np.nan_to_num(grayscale_cam)
    graySmeanPne=np.mean(grayscale_cam,axis=0) # Activity of genes in Pneumoctyes


    indxcell=gt_data['idxHema']
    grayscale_cam=np.zeros([indxcell.shape[0],colNum*rowNum])
    for i in range(indxcell.shape[0]):
```

```
        grad_cam_img = torch.Tensor(dataMat_CNNtrain[i,:,:,:].transpose(2,0,1)
                                    .reshape((1,1,colNum,rowNum))).to(device)
        pred_label = torch.argmax(net(grad_cam_img),axis=-1)
        targets = [ClassifierOutputTarget(pred_label)]
        A = cam(input_tensor=grad_cam_img, targets=targets)
        B=np.reshape(A.T,(1,colNum*rowNum))
        grayscale_cam[i,:]=B

    graySmeanHema=np.mean(grayscale_cam,axis=0) # Activity of genes in Hematopoitic cells


    # Stack all the average gene activity values in different types of cells for plotting
    activityMatrix=np.vstack((graySmeanB,graySmeanHema,graySmeanMono,graySmeanNK,graySmeanT,


    # Load the gene names for plotting
    geneNames = pd.read_csv('../data/gene_interest_ROI_ImXreXB.csv', header=None,
                        delim_whitespace=False)


    # The following are the 10 most variable genes in sci-ATAC-seq dataset
    genes=np.array(('MSC','RPL31','SERPINB7','ARHGEF4','DCAF8','RNF149','ACMSD','ESPNL','FAM


    # Plot the activity of the 10 most variable genes in sci-ATAC-seq dataset
    idxS=np.zeros(genes.shape[0])
    for ii in range(genes.shape[0]):

        idx=[geneNames==genes[ii]]
        idxN=np.squeeze((np.array(idx)))
        idxS[ii]=np.squeeze(np.asarray(np.where(idxN==True)))

    idxSs=idxS.astype(int)
    actData=activityMatrix[:,idxSs]

    # Rescale the activity in each cell type from 0 to 1 for plotting
    actDataRe=actData
    for ik in range(6):
        actDataRe[ik,:]=rescale(actData[ik,:])

    plt.figure(figsize=(12,4))
    ax=plt.stem(genes,actDataRe[0,:]) # to see the gene activity in other cells, please
    change this
    # index to a value from 1 to 5. Default is 0 (B-cell). 1 is for T-cell, 2 for natural
    killer cells,
    # 3 for Monocytes, 4 for Pneumoctyes, and 5 for Hematopoitic cells.
    plt.xlabel("Genes")
    plt.ylabel("Activity score")
    plt.title('Mean activity of genes in B-cells')
```

Out[5]: `Text(0.5, 1.0, 'Mean activity of genes in B-cells')`





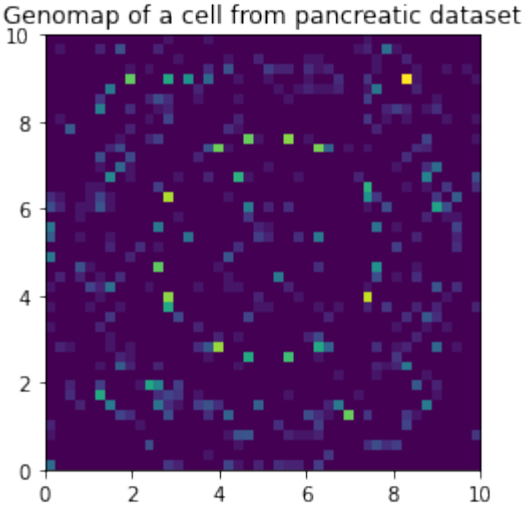
Multi-omic integration

In [6]:
```
# The pancreatic scRNA-seq data from 5 different technologies (Baron et al,
# Muraro et al., Xin et al., Wang et al., and Segerstolpe et al.) are first integrated
with Seurat.
# In Seurat, the parameter 'nfeatures' (denoting  the number of features) is set
# to 2000. The resulting Seurat output is loaded here
data = sio.loadmat('../data/outSeurat_pancORG.mat')
outSeurat=data['outSeurat']

# We now create a genomap for each cell in the data
# We select the nearest square number to 2000, which is 1936
# Thus the size of the genomap would be 44 by 44.
# Next, let us select data with 1936 most variable features
numRow=44
numCol=44
varX=np.var(outSeurat,axis=0)
idxV=np.argsort(varX)
idxVX=np.flip(idxV)
outSeurat1936=outSeurat[:,idxVX[0:numRow*numCol]]

# Construction of the genomaps for the pancreatic data from the five different
technologies
```

```
genoMaps=construct_genomap(outSeurat1936,numRow,numCol,epsilon=0.0,num_iter=200)


# Visualize a genomap (we show here the first one (i=0))
findI=genoMaps[0,:,:,:]
plt.figure(4)
plt.imshow(findI, origin = 'lower',  extent = [0, 10, 0, 10], aspect = 1)
plt.title('Genomap of a cell from pancreatic dataset')
# Load the data labels
gt_data = sio.loadmat('../data/GT_panc.mat')
GT = np.squeeze(gt_data['GT'])

# Load the index of the training and testing data
# Here training data are from Baron et al, Muraro et al., Xin et al., and Wang et al.
#  and the testing data is from Segerstolpe et al.
index_data = sio.loadmat('../data/index_panc.mat')
indxTest = np.squeeze(index_data['indxTest'])
indxTrain = np.squeeze(index_data['indxTrain'])
GT = GT - 1

# Prepare the data for genoNet training
dataMat_CNNtrain = genoMaps[indxTrain-1,:,:,:] # transfer from matlab indexing to
python indexing
dataMat_CNNtest = genoMaps[indxTest-1,:,:,:]
groundTruthTest = GT[indxTest-1]
groundTruthTrain = GT[indxTrain-1]

XTrain = dataMat_CNNtrain[:,:,:,:].transpose([0,3,1,2])
XTest = dataMat_CNNtest.transpose([0,3,1,2])
# Make labels begin with 0
yTrain = groundTruthTrain-1
yTest = groundTruthTest-1

miniBatchSize = 128
# Train the genoNet
torch.manual_seed(0)
net = traingenoNet(XTrain, yTrain, maxEPOCH=150, batchSize=miniBatchSize,
verbose=True)

# Test the genoNet
testset = geneDataset(XTest, yTest)
testloader = DataLoader(testset, batch_size=miniBatchSize, shuffle=False)
device = torch.device('cpu')
prob_test = predict(net, testloader, device)
pred_test = np.argmax(prob_test, axis=-1)

print('Label transfer accuracy of genomap+genoNet:'+ str(np.sum(pred_test==yTest) /
pred_test.shape[0]))
```

```
0: 0.9680222721412929
10: 0.04672816498769504
20: 0.013940912345659476
30: 0.03075548820550356
40: 0.005105792394449118
50: 0.00979785608466376
60: 0.0057228274748959305
70: 0.0021925929242496926
```

```
80: 0.0030763835010971393
90: 0.0027813530548681025
100: 0.004214840945972567
110: 0.0005964575616509146
120: 0.0015612584203439047
130: 0.0025263514307145595
140: 0.0013759825180950382
Label transfer accuracy of genomap+genoNet:0.9652092148566056
```



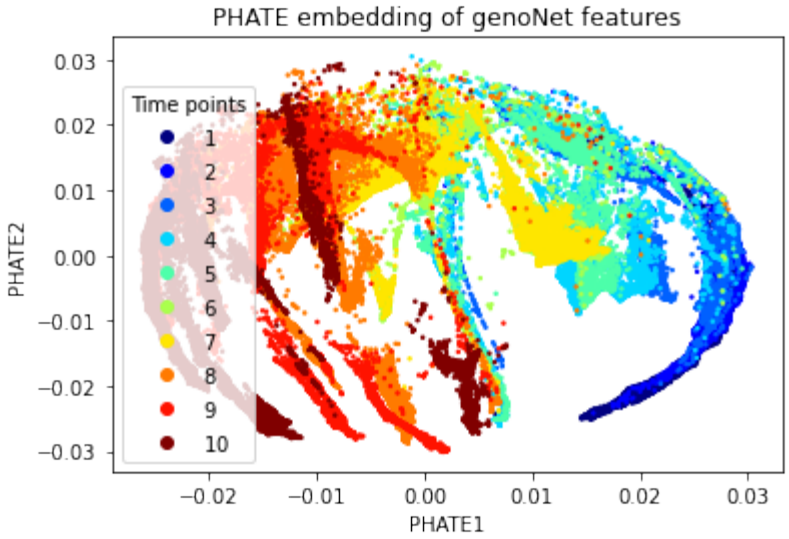
Genomap enables high quality cellular trajectory mapping

In [7]:
```
# As cellular trajectory mapping is an unsupervised task, we import the unsupervised
genoNet.
# The difference between a supervised and unsupervised
# genoNet is the size of the final fully connected (FFC) layer. As unsupervised
genoNet
# is used for feature extraction, its FFC layer needs to have much more neurons than
the
# supervised genoNet. We set the neuron number in the FFC layer of supervised
# genoNet to 100 and of unsupervised genoNet to 512.

# import the unsupervised genoNet
from genoNetus import genoNet,geneDataset,load_genoNet,predict,traingenoNet

# For the purpose of rapid demonstration, we have created/saved the genomaps for this
dataset
# The saved genomaps are uploaded here.
# However, one can also create the genomaps from data/data_proto.mat by
# following the process described in the third section.

# Load the genomaps and tabular data of proto-vertebrate dataset
data = sio.loadmat('../data/genoMaps_Proto.mat')
genoMaps = data['genoMaps']
dataX = sio.loadmat('../data/data_proto.mat')
dataVec = dataX['X']

# Load data labels for visualization
gt_data = sio.loadmat('../data/GT_proto.mat')
GrTruth = np.squeeze(gt_data['GT'])

# Use K-means++ for the initial clustering
```

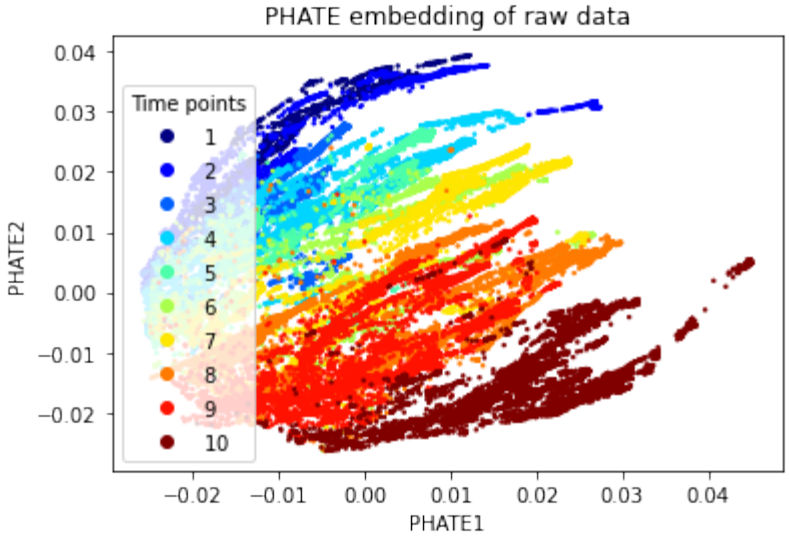
```
amount_initial_centers = 10
initial_centers = kmeans_plusplus_initializer(dataVec,
amount_initial_centers).initialize()
# Use x-means for clustering the data
xmeans_instance = xmeans(dataVec, initial_centers, 20)
xmeans_instance.process()
# Extract clustering results: clusters and their centers
clusters = xmeans_instance.get_clusters()
centers = xmeans_instance.get_centers()

clusIndex=np.zeros(dataVec.shape[0]);
for i in range(0,len(clusters)):
    a=(clusters[i])
    clusIndex[a]=i;


# Set up the training and testing data
dataMat_CNNtrain = genoMaps # For unsupervised genoNet, all the data are used for
training
dataMat_CNNtest = genoMaps

# Use the estimated labels by x-means clustering for training the unsupervised genoNet
groundTruthTest = clusIndex
groundTruthTrain = clusIndex
classNum = len(np.unique(groundTruthTrain))
XTrain = dataMat_CNNtrain[:,:,:,:].transpose([3,2,0,1])
XTest = dataMat_CNNtest.transpose([3,2,0,1])
yTrain = groundTruthTrain
yTest = groundTruthTest

# Train the genoNet
miniBatchSize = 128

# For the purpose of rapid demonstration, we have trained the genoNet model, which
# is uploaded here. But if one wants to train the
# genoNet on his/her dataset, please uncomment the following code:
# net = traingenoNet(XTrain, yTrain, maxEPOCH=30, batchSize=miniBatchSize,
verbose=True)

# Load the trained genoNet.
net = load_genoNet([1,27,27], 20, '../data/genoNet_PHATE_ZF.pt')

# Extract genoNet features from the final fully connected layer
# for PHATE analysis.
gx=np.reshape(XTrain,(XTrain.shape[0],1,XTrain.shape[2],XTrain.shape[3]))
device = _get_device()
t = torch.from_numpy(gx).to(device)
net=net.double()
dataAtFC=net.forwardX(t)
data=dataAtFC.cpu().detach().numpy()

# Run PHATE on the genoNet features
phate_op = phate.PHATE(random_state=1)
X_embedded = phate_op.fit_transform(data)

# Plot embeddings
plt.figure(11)
```

```
    scatter=plt.scatter(X_embedded[:,0], X_embedded[:,1],s=2,c=GrTruth,cmap="jet")
    plt.xlabel("PHATE1")
    plt.ylabel("PHATE2")
    plt.title('PHATE embedding of genoNet features')
    legend1 = plt.legend(*scatter.legend_elements(),
                         loc="lower left", title="Time points")
    plt.show()

    # Run PHATE on the raw data and compare with result from the proposed approach
    X_embedded = phate_op.fit_transform(dataVec)

    # Plot embeddings
    plt.figure(12)
    scatter=plt.scatter(X_embedded[:,0], X_embedded[:,1],s=2,c=GrTruth,cmap="jet")
    plt.xlabel("PHATE1")
    plt.ylabel("PHATE2")
    plt.title('PHATE embedding of raw data')
    legend1 = plt.legend(*scatter.legend_elements(),
                         loc="lower left", title="Time points")
    plt.show()
```

```
Calculating PHATE...
  Running PHATE on 90579 observations and 512 variables.
  Calculating graph and diffusion operator...
    Calculating PCA...
    Calculated PCA in 3.48 seconds.
    Calculating KNN search...
    Calculated KNN search in 355.90 seconds.
    Calculating affinities...
    Calculated affinities in 0.57 seconds.
  Calculated graph and diffusion operator in 360.04 seconds.
  Calculating landmark operator...
    Calculating SVD...
    Calculated SVD in 6.66 seconds.
    Calculating KMeans...
    Calculated KMeans in 7.78 seconds.
  Calculated landmark operator in 16.94 seconds.
  Calculating optimal t...
    Automatically selected t = 26
  Calculated optimal t in 1.33 seconds.
  Calculating diffusion potential...
  Calculated diffusion potential in 0.29 seconds.
  Calculating metric MDS...
  Calculated metric MDS in 4.13 seconds.
Calculated PHATE in 382.74 seconds.
```



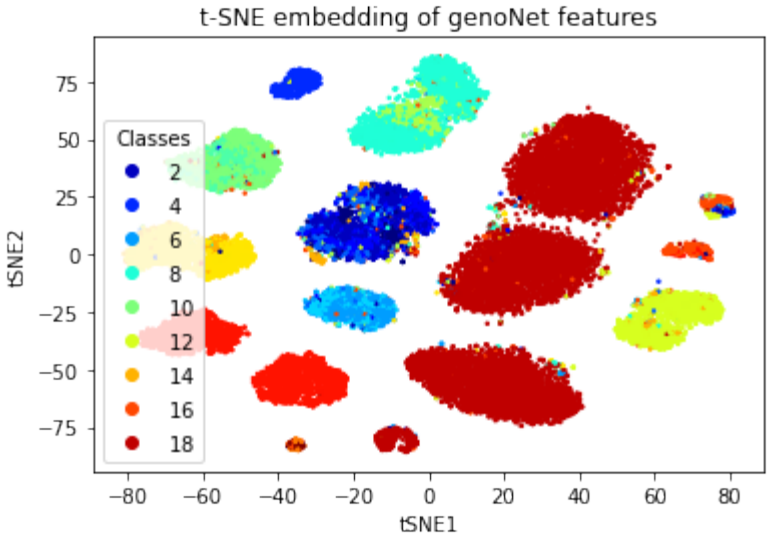

```
Calculating PHATE...
  Running PHATE on 90579 observations and 752 variables.
  Calculating graph and diffusion operator...
    Calculating PCA...
    Calculated PCA in 4.24 seconds.
    Calculating KNN search...
    Calculated KNN search in 448.82 seconds.
    Calculating affinities...
    Calculated affinities in 41.32 seconds.
  Calculated graph and diffusion operator in 494.50 seconds.
  Calculating landmark operator...
    Calculating SVD...
    Calculated SVD in 7.39 seconds.
    Calculating KMeans...
    Calculated KMeans in 7.02 seconds.
  Calculated landmark operator in 16.87 seconds.
  Calculating optimal t...
    Automatically selected t = 36
  Calculated optimal t in 1.31 seconds.
  Calculating diffusion potential...
  Calculated diffusion potential in 0.29 seconds.
  Calculating metric MDS...
  Calculated metric MDS in 4.03 seconds.
Calculated PHATE in 517.01 seconds.
```

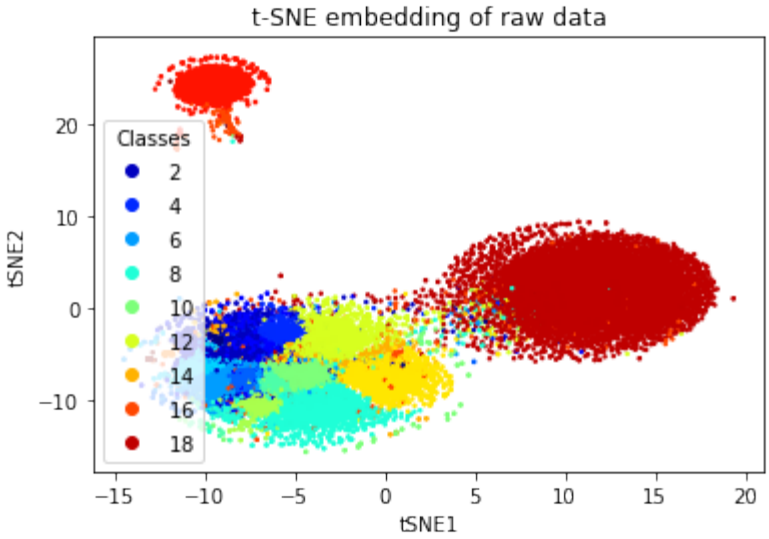


Dimensionality reduction, visualization and clustering

In [8]:
```
# We use unsupervised genoNet for dimensionality reduction.
# We create the genomaps of a dataset and then train the unsupervised genoNet
# using the genomaps. We then extract features from the final fully connected layer of
the genoNet, which
# have a lower dimensionality than the original data. We then apply
# t-SNE on the features to obtain the embeddings.

# Load the genomaps and tabular data of comprehensive classification of mouse retinal
bipolar cells
data = sio.loadmat('../data/genoMap_comClass.mat')
genoMaps = data['genoMaps']
dataX = sio.loadmat('../data/data_comClass.mat')
dataVec = dataX['X']

# Load cell labels for visualization
gt_data = sio.loadmat('../data/GT_comClass.mat')
GrTruth = np.squeeze(gt_data['GT'])

# Use k-means++ for initial clustering
amount_initial_centers = 10
initial_centers = kmeans_plusplus_initializer(dataVec,
amount_initial_centers).initialize()
# Use x-means for clustering the data
xmeans_instance = xmeans(dataVec, initial_centers, 20)
xmeans_instance.process()
# Extract clustering results: clusters and their centers
clusters = xmeans_instance.get_clusters()
centers = xmeans_instance.get_centers()

clusIndex=np.zeros(dataVec.shape[0]);
for i in range(0,len(clusters)):
    a=(clusters[i])
    clusIndex[a]=i;


# Set up the training and testing data
dataMat_CNNtrain = genoMaps # For unsupervised genoNet, all the data are used for
training
dataMat_CNNtest = genoMaps

# Use the estimated labels for genoNet training
groundTruthTest = clusIndex
groundTruthTrain = clusIndex
classNum = len(np.unique(groundTruthTrain))

# Prepare data for training
XTrain = dataMat_CNNtrain[:,:,:,:].transpose([3,2,0,1])
XTest = dataMat_CNNtest.transpose([3,2,0,1])
yTrain = groundTruthTrain
yTest = groundTruthTest

# Train the genoNet
miniBatchSize = 128
```

```
# If one wants to train the genoNet on his/her dataset, please uncomment the folloiwng
code line
# net = traingenoNet(XTrain, yTrain, maxEPOCH=30, batchSize=miniBatchSize,
verbose=True)

# Load the trained genoNet
net = load_genoNet([1,33,33], 19, '../data/genoNet_TSNE_ComClass.pt')
# Extract genoNet features from the final fully connected layer
gx=np.reshape(XTrain,(XTrain.shape[0],1,XTrain.shape[2],XTrain.shape[3]))
device = _get_device()
t = torch.from_numpy(gx).to(device)
net=net.double()
dataAtFC=net.forwardX(t)

# Run t-SNE on the genoNet features
X=dataAtFC.cpu().detach().numpy()
X_embedded = TSNE(n_components=2, learning_rate='auto',init='random').fit_transform(X)

plt.figure(13)
scatter=plt.scatter(X_embedded[:,0], X_embedded[:,1],s=2,c=GrTruth,cmap="jet")
plt.xlabel("tSNE1")
plt.ylabel("tSNE2")
plt.title('t-SNE embedding of genoNet features')
legend1 = plt.legend(*scatter.legend_elements(),
                     loc="lower left", title="Classes")
plt.show()


X_embedded = TSNE(n_components=2,
learning_rate='auto',init='random').fit_transform(dataVec)

plt.figure(14)
scatter=plt.scatter(X_embedded[:,0], X_embedded[:,1],s=2,c=GrTruth,cmap="jet")
plt.xlabel("tSNE1")
plt.ylabel("tSNE2")
plt.title('t-SNE embedding of raw data')
legend1 = plt.legend(*scatter.legend_elements(),
                     loc="lower left", title="Classes")
plt.show()
```




t-SNE embedding of raw data
Classes
2
4
6
8
10
12
14
16
18
tSNE1
tSNE2
−15
−10
−5
0
5
10
15
20
−10
0
10
20