# **kütüphaneleri import ediyoruz**

|  |
| --- |
| import numpy as np  import random as rand  import numpy.matlib as nm  import math |

NumPy, Python programlama dili için büyük, çok boyutlu dizileri ve matrisleri destekleyen, bu diziler üzerinde çalışacak üst düzey matematiksel işlevler ekleyen bir kütüphanedir.

Belirli bir aralıkta rastgele sayıların üretilmesine ihtiyaç duyarsanız Python’ın standart kütüphanesinde bulunan random adlı bir modülü kullanabilirsiniz.

Matplotlib en popüler görselleştirme kütüphanesidir.

Math kütüphanesi Python’un standart kütüphanelerinden biridir. Bazı matematiksel işlemleri yapmanıza kolaylık sağlamaktadır.

|  |
| --- |
| from scipy.sparse import coo\_matrix, isspmatrix, spdiags  from numpy.linalg import cholesky, LinAlgError, norm  from scipy.sparse.linalg import eigs, eigsh |

|  |
| --- |
| def CutonRatio(U,S,V,options):    if 'PCARatio' in options:  pass  else:  options['PCARatio'] = 1  eigvalue\_PCA = np.diag(S.toarray())  if options['PCARatio'] > 1:  idx = options['PCARatio']  if idx < len(eigvalue\_PCA):  U = U[:,0:idx]  V = V[:,0:idx]  S = np.diag(S.toarray()[0:idx,0:idx])  S = spdiags(S,0,len(S),len(S))  elif options['PCARatio'] < 1:  sumEig = sum(eigvalue\_PCA)  sumEig = sumEig\*options['PCARatio']  sumNow = 0  for idx in range(len(eigvalue\_PCA)):  sumNow = sumNow + eigvalue\_PCA[idx]  if sumNow >= sumEig:  break  U = U[:,0:idx]  V = V[:,0:idx]  S = np.diag(S.toarray()[0:idx,0:idx])  S = spdiags(S,0,len(S),len(S))  return U, S, V |

Eigenvalue değerinin büyüklüğü o PC doğrultusundaki varyansın bir ölçüsüdür. Yani eigenvalue ne kadar büyükse o doğrultuda veri kümenizin yayılımı o kadar fazladır. PCA ile indirgeme PC ve karşılık gelen eigenvalue'lara bağlı olarak veri kümesinin boyutunu indirgemeye denir. PCA ile indirgeme PC ve karşılık gelen eigenvalue’lara bağlı olarak veri kümesinin boyutunu indirgemeye denir.

numpy.diag Bir diyagonal ayıklayın veya bir diyagonal dizi oluşturun.

scipy.sparse.spdiags Köşegenlerden seyrek bir matris döndürün.

|  |
| --- |
| def mySVD(X,nargout):    MAX\_MATRIX\_SIZE = 1600  EIGVECTOR\_RATIO = 0.1    reducedDim = 0  nSmp, mFea = np.shape(X)    if (mFea/nSmp > 1.0713):  # This is an efficient method which computes the eigvectors of  # of A\*A^T (instead of A^T\*A) first, and then convert them back to  # the eigenvectors of A^T\*A.  ddata = np.dot(X,X.transpose())  ddata = np.maximum(ddata,ddata.transpose())    dimMatrix = len(ddata)  if(reducedDim > 0 & dimMatrix > MAX\_MATRIX\_SIZE & reducedDim < dimMatrix\*EIGVECTOR\_RATIO):  eigvalue, U = np.linalg.eig(ddata)  index = np.argsort(-eigvalue)  eigvalue = eigvalue[index]  U = U[:,index]  eigvalue = np.array(eigvalue[0:reducedDim])  U = U[:,0:reducedDim]  else:  if isspmatrix(ddata):  ddata = ddata.toarray()  eigvalue, U = np.linalg.eig(ddata)  index = np.argsort(-eigvalue)  eigvalue = eigvalue[index]  U = U[:,index]  maxEigValue = max(np.abs(eigvalue))  eigIdx = np.where(abs(eigvalue)/maxEigValue > 1e-10)[0]## 'where' return tuple,with [0] gettting array  eigvalue = eigvalue[eigIdx]  U = U[:,eigIdx]    if reducedDim > 0 & reducedDim < len(eigvalue):  eigvalue = eigvalue[0:reducedDim]  U = U[:,0:reducedDim]    eigvalue\_Half = np.sqrt(eigvalue)  S = spdiags(eigvalue\_Half,0,len(eigvalue\_Half),len(eigvalue\_Half))  if nargout >= 3:  eigvalue\_MinusHalf = np.power(eigvalue\_Half,-1)  V = np.dot(np.array(X).transpose(),(np.array(U)\*nm.repmat(eigvalue\_MinusHalf,len(U),1)))  else:  ddata = np.dot(X.transpose(),X)  ddata = np.maximum(ddata,ddata.transpose())    dimMatrix = len(ddata)  if(reducedDim > 0 & dimMatrix > MAX\_MATRIX\_SIZE & reducedDim < dimMatrix\*EIGVECTOR\_RATIO):  eigvalue, V = np.linalg.eig(ddata)  index = np.argsort(-eigvalue)  eigvalue = eigvalue[index]  V = V[:,index]  eigvalue = np.array(eigvalue[0:reducedDim])  V = V[:,0:reducedDim]  else:  if isspmatrix(ddata):  ddata = ddata.toarray()  eigvalue, V = np.linalg.eig(ddata)  index = np.argsort(-eigvalue)  eigvalue = eigvalue[index]  V = V[:,index]  maxEigValue = max(np.abs(eigvalue))  eigIdx = np.where(abs(eigvalue)/maxEigValue > 1e-10)[0]## 'where' return tuple,with [0] gettting array  eigvalue = eigvalue[eigIdx]  V = V[:,eigIdx]    if reducedDim > 0 & reducedDim < len(eigvalue):  eigvalue = eigvalue[0:reducedDim]  V = V[:,0:reducedDim]    eigvalue\_Half = np.sqrt(eigvalue)  S = spdiags(eigvalue\_Half,0,len(eigvalue\_Half),len(eigvalue\_Half))    eigvalue\_MinusHalf = np.power(eigvalue\_Half,-1)  U = np.dot(np.array(X),(np.array(V)\*nm.repmat(eigvalue\_MinusHalf,len(V),1)))    return U,S,V |

Nargout o anda yürütülmekte olan işleve yapılan çağrıda belirtilen işlev çıktı bağımsız değişkenlerinin sayısını döndürür.

numpy.dot İki dizinin nokta çarpımı.

numpy. maximum İki diziyi karşılaştırır ve eleman bazında maksimumu içeren yeni bir dizi döndürür. Karşılaştırılan öğelerden biri bir NaN ise, o öğe döndürülür. Her iki öğe de NaN ise, ilki döndürülür. İkinci ayrım, gerçek veya sanal parçalardan en az birinin NaN olması olarak tanımlanan karmaşık NaN'ler için önemlidir. Net etki, NaN'lerin yayılmasıdır.

# numpy. Shape Bir dizinin şeklini döndürür.

numpy.transpose Bir dizinin eksenlerini tersine çevirin veya değiştirin; değiştirilen diziyi döndürür.İki eksenli bir a dizisi için devrik(a) matris devrik değerini verir.

numpy.matrix Sınıf gelecekte kaldırılabilir.Dizi benzeri bir nesneden veya bir veri dizisinden bir matris döndürür. Bir matris, işlemler yoluyla 2 boyutlu yapısını koruyan özel bir 2 boyutlu dizidir. \*(matris çarpımı) ve \*\*(matris gücü) gibi belirli özel operatörlere sahiptir .

reducedDims"mat" veya "matris" olarak ayarlanırsa, işlev nesneyi her bir hücre için girişleri olan bir matris olarak döndürür . reducedDims Aksi takdirde, tam nesneyi döndürür .

numpy.linalg.eig Bir kare dizinin özdeğerlerini ve sağ özvektörlerini hesaplayın.

numpy.argsort Bir diziyi sıralayacak dizinleri döndürür.*tür* anahtar sözcüğü tarafından belirtilen algoritmayı kullanarak verilen eksen boyunca dolaylı bir sıralama gerçekleştirin . Sıralanmış bir düzende verilen eksen boyunca o indeks verisi ile aynı şekle sahip *bir* indeks dizisini döndürür .

numpy.absolute Mutlak değeri eleman bazında hesaplayın.

# numpy.where Koşullara bağlı olarak x veya y arasından seçilen öğeleri döndürür .

# ## 'where' return tuple,with [0] gettting array

numpy.sqrt Bir dizinin negatif olmayan karekökünü eleman bazında döndürür.

scipy.sparse.spdiags Köşegenlerden seyrek bir matris döndürün.

numpy.power İlk dizi öğeleri, öğe bazında ikinci diziden güçlere yükseltildi.*x1'deki* her tabanı, *x2'deki* konumsal olarak karşılık gelen güce yükseltin . *x1* ve *x2* aynı şekle yayınlanabilir olmalıdır. Negatif bir tamsayı gücüne yükseltilmiş bir tamsayı türünün bir ValueError oluşturacağını unutmayın.

repmat Bir diziyi çoğaltın ve döşeyin

numpy.dot İki dizinin nokta çarpımı.

|  |
| --- |
| def LGE(W, D, options, data):    MAX\_MATRIX\_SIZE = 1600  EIGVECTOR\_RATIO = 0.1    reducedDim = 30  if 'reducedDim' in options:  reducedDim = options['reducedDim']  if 'Regu' not in options:  bPCA = 1  if 'PCARatio' not in options:  options['PCARatio'] = 1  else:  bPCA = 0  if 'ReguType' not in options:  options['ReguType'] = 'Ridge'  if 'ReguAlpha' not in options:  options['ReguAlpha'] = 0.1  if 'PCARatio' not in options:  options['PCARatio'] = 1  bD = 1    if(isinstance(D, (np.ndarray))):  lenD = len(D)  else:    lenD = len(D.toarray())    if lenD == 0:  bD = 0  nSmp,nFea = np.shape(data)  if W.shape[0] != nSmp:  print('W and data mismatch!')  if bD ==1 & (lenD != nSmp):  print('D and data mismatch!')    bChol = 0  if bPCA==1 & (nSmp > nFea) & (options['PCARatio'] >= 1):  if bD == 1:  DPrime = np.dot(np.dot(data.transpose(),D),data)  else:  DPrime = np.dot(data.transpose(),data)  DPrime = np.maximum(DPrime, DPrime.transpose())  try:  R = cholesky(DPrime)  p = True  except LinAlgError as err:  if 'Matrix is not positive definite' in str(err):  p = False  if p == True:  bPCA = 0  bChol = 1  if bPCA == 1:  U, S, V = mySVD(data,3)  U, S ,V = CutonRatio(U,S,V,options)  eigvalue\_PCA = np.diag(S.toarray())  if bD == 1:  data = np.dot(U,S)  eigvector\_PCA = V    DPrime = np.dot(np.dot(data.transpose(),D),data)  DPrime = np.maximum(DPrime, DPrime.transpose())  else:  data = U  K = spdiags(np.power(eigvalue\_PCA,-1), 0, len(eigvalue\_PCA), len(eigvalue\_PCA))  eigvector\_PCA = np.dot(V,K.toarray())  else:  if bChol == 0:  if bD == 1:  DPrime = np.dot(np.dot(data.transpose(),D.toarray()),data)  else:  DPrime = np.dot(data.transpose(),data)  if options['ReguType'] == 'Ridge':  for i in range(len(DPrime)):  DPrime[i,i] += options['ReguAlpha']  elif options['ReguType'] == 'Tensor':  DPrime = np.add(DPrime, options['ReguAlpha']\*options['regularizerR'])  elif options['ReguType'] == 'Custom':  DPrime = np.add(DPrime, options['ReguAlpha']\*options['regularizerR'])  else:  print('ReguType does not exist!')    DPrime = np.maximum(DPrime,DPrime.transpose())    WPrime = np.dot(np.dot(data.transpose(),W.toarray()),data)  WPrime = np.maximum(WPrime,WPrime.transpose())  dimMatrix = WPrime.shape[0]  if reducedDim > dimMatrix:  reducedDim = dimMatrix    if 'bEigs' in options:  bEigs = options['bEigs']  else:  if (dimMatrix > MAX\_MATRIX\_SIZE) & (reducedDim < dimMatrix\*EIGVECTOR\_RATIO):  bEigs = 1  else:  bEigs = 0    if bEigs == 1:  if bPCA ==1 and bD == 0:  eigvalue, eigvector = eigsh(WPrime, k=reducedDim, which='LA')  else:  if bChol ==1:  eigvalue, eigvector = eigsh(WPrime, k=reducedDim, M=R.transpose(), which='LA')  else:  eigvalue, eigvector = eigsh(WPrime, k=reducedDim, M=DPrime, which='LA')  else:  if bPCA==1 and bD == 0:  eigvalue, eigvector = eigsh(WPrime, k = len(WPrime)-1)  else:  eigvalue, eigvector = eigsh(DPrime - WPrime, k = len(WPrime)-1, M=DPrime)  index = np.argsort(eigvalue)  eigvalue = eigvalue[index]  eigvector = eigvector[:,index]    if reducedDim < eigvector.shape[1]:  eigvector = eigvector[:,0:reducedDim]  eigvalue = eigvalue[0:reducedDim]    if bPCA == 1:  eigvector = np.dot(eigvector\_PCA,eigvector)    for i in range(eigvector.shape[1]):  eigvector[:,i] = eigvector[:,i]/norm(eigvector[:,i])    return eigvector, eigvalue |

isinstance(), nesnenin (ilk argüman) classinfo sınıfının bir örneği veya alt sınıfı (ikinci argüman) olup olmadığını kontrol eder.

numpy.ndarray Bir dizi nesnesi, çok boyutlu, homojen bir sabit boyutlu öğeler dizisini temsil eder. İlişkili bir veri türü nesnesi, dizideki her öğenin biçimini (bayt sırası, bellekte kaç bayt kapladığı, bir tamsayı mı, kayan noktalı sayı mı yoksa başka bir şey mi, vb.)

scipy.linalg.cholesky Bir matrisin Cholesky ayrıştırmasını hesaplayın.

Cholesky ayrışması Lineer cebirde, Cholesky ayrıştırması veya Cholesky çarpanlara ayırması (/ʃəˈlɛski/ shə-LES-kee olarak telaffuz edilir), bir Hermitian, pozitif tanımlı matrisin bir alt üçgen matrisin ürününe ve verimli için yararlı olan eşlenik devrik ürününe ayrıştırılmasıdır. sayısal çözümler, örneğin Monte Carlo simülasyonları. Gerçek matrisler için.

# Dprime d' veya duyarlılık indeksi olarak bilinen iki olasılık dağılımı arasındaki standartlaştırılmış mesafeyi elde edin.

numpy.linalg.LinAlgError Linalg işlevleri tarafından oluşturulan genel Python istisnasından türetilmiş nesne.Python'un istisna.İstisna sınıfından türetilen genel amaçlı istisna sınıfı, Lineer Cebir ile ilgili bir koşul fonksiyonun daha fazla doğru yürütülmesini engellediğinde linalg işlevlerinde programlı olarak oluşturulur.

scipy.sparse.linalg.eigsh Gerçek simetrik kare matris veya karmaşık Hermit matrisi A'nın k özdeğerlerini ve özvektörlerini bulun

# eigvalue, eigvector Özdeğerler ve özvektörler

numpy.linalg.norm Matris veya vektör normu.

|  |
| --- |
| def LPP(W, options, data):    nSmp,nFea = np.shape(data)  if W.shape[0] != nSmp:  print('W and data mismatch!')    if 'keepMean' in options:  pass  else:  if isspmatrix(data):  data = data.toarray()  sampleMean = np.mean(data, axis = 0)  data = (data - nm.repmat(sampleMean,nSmp,1))    D = np.sum(W,axis=1)    if 'Regu' not in options:  DToPowerHalf = np.power(D,0.5)  D\_mhalf = np.power(DToPowerHalf,-1)  #D\_mhalf = np.power(np.add(DToPowerHalf,0.01),-1)    if nSmp < 5000:  tmpD\_mhalf = nm.repmat(D\_mhalf,1,nSmp)  if isinstance(W, (np.ndarray)):  W = np.multiply(np.multiply(tmpD\_mhalf,W),tmpD\_mhalf.transpose())  else:  W = np.multiply(np.multiply(tmpD\_mhalf,W.toarray()),tmpD\_mhalf.transpose())  W = coo\_matrix(W)  else:  j\_idx, i\_idx = np.where(W.toarray() != 0)  v1\_idx = np.zeros(len(i\_idx))  LL = W.toarray()  for i in range(len(j\_idx)):  v1\_idx[i] = LL[i\_idx[i],j\_idx[i]]\*D\_mhalf[i\_idx[i]]\*D\_mhalf[j\_idx[i]]  W = coo\_matrix((v1\_idx,(i\_idx,j\_idx)), shape=(nSmp, nSmp))    W = np.maximum(W.toarray(),W.toarray().transpose())  W = coo\_matrix(W)  data = np.multiply(nm.repmat(DToPowerHalf,1,nFea),data)  eigvector, eigvalue = LGE(W, np.array([]), options, data)  else:  if 'ReguAlpha' not in options:  options['ReguAlpha'] = 0.1  options['ReguAlpha'] = options['ReguAlpha']\*sum(D)/len(D)  D = coo\_matrix((np.array(D).ravel(),(range(nSmp),range(nSmp))),shape = (nSmp,nSmp))  eigvector, eigvalue = LGE(W, D, options, data)    eigIdx = np.where(eigvalue > 1e-3)[0]  eigvalue = eigvalue[eigIdx]  eigvector = eigvector[:,eigIdx]    return eigvector, eigvalue |

numpy.mean Belirtilen eksen boyunca aritmetik ortalamayı hesaplayın.

numpy.multiply Argümanları eleman bazında çarpın.

numpy.zeros Verilen şekil ve türde sıfırlarla dolu yeni bir dizi döndürün.

|  |
| --- |
| def EuDist2(fea\_a,fea\_b,bSqrt):  if(len(fea\_b)==0):  squareFeaA = fea\_a\*fea\_a#Eleman eleman çarpimi  sumFeaAA = np.sum(squareFeaA,axis=1) ## satir elemanlarini toplar  sumFeaAB = np.dot(fea\_a,fea\_a.transpose())#matris carpimi    if(isspmatrix(np.array(sumFeaAA))):  sumFeaAA.toarray()  sumFeaAA = np.reshape(sumFeaAA,(-1,1)) #tek boyutlu diziyi iki boyutlu diziye çevirir    D = sumFeaAA[...,:]+sumFeaAA.transpose()-2\*sumFeaAB  D[D<0]=0  if(bSqrt==1):  D = np.sqrt(D)  D = np.maximum(D,D.transpose())  else:  squareFeaA = fea\_a\*fea\_a#Eleman eleman çarpimi  sumFeaAA = np.sum(squareFeaA,axis=1) ## satir elemanlarini toplar    squareFeaB = fea\_b\*fea\_b#Eleman eleman çarpimi  sumFeaBB = np.sum(squareFeaB,axis=1)    sumFeaAB = np.dot(fea\_a,fea\_b.transpose())  if(isspmatrix(np.array(sumFeaAA))):  sumFeaAA.toarray()  sumFeaBB.toarray()  sumFeaAA = np.reshape(sumFeaAA,(-1,1)) #tek boyutlu diziyi iki boyutlu diziye çevirir  sumFeaBB = np.reshape(sumFeaBB,(-1,1)) #tek boyutlu diziyi iki boyutlu diziye çevirir    D = sumFeaAA[...,:]+sumFeaBB.transpose()-2\*sumFeaAB  D[D<0]=0  if(bSqrt==1):  D = np.sqrt(D)    return D |

numpy.reshape Verilerini değiştirmeden diziye yeni bir şekil verir.

|  |
| --- |
| def NormalizeFea(fea,row):    if row==1:  nSmp = fea.shape[0]  feaNorm = np.maximum(1e-14,np.sum(fea\*\*2,1))  fea = spdiags(feaNorm\*\*-.5,0,nSmp,nSmp)\*fea  else:  nSmp = fea.shape[1]  feaNorm = np.maximum(1e-14,np.transpose(np.sum(fea\*\*2,0)))  fea = fea\*spdiags(feaNorm\*\*-.5,0,nSmp,nSmp)  return fea |

scipy.sparse.spdiags Köşegenlerden seyrek bir matris döndürün.

|  |
| --- |
| def constructWLPP(fea,options):    bSpeed = 1  if 'bNormalized' not in options:  options['bNormalized'] = 0    if 'NeighborMode' not in options:  options['NeighborMode'] = 'KNN'    if options['NeighborMode'] == 'KNN':  if 'k' not in options:  options['k'] = 5    elif options['NeighborMode'] == 'Supervised':  if 'bLDA' not in options:  options['bLDA'] = 0  if options['bLDA'] == 1:  options['bSelfConnected'] = 1  if 'k' not in options:  options['k'] = 0  if 'gnd' not in options:  print('Label(gnd) should be provided under ''Supervised'' NeighborMode!')  else:  print('NeighborMode does not exist!')    if 'WeightMode' not in options:  options['WeightMode'] = 'HeatKernel'    bBinary = 0  bCosine = 0    if options['WeightMode'] == 'Binary':  bBinary = 1  elif options['WeightMode'] == 'HeatKernel':  if 't' not in options:  nSmp = fea.shape[0]  if nSmp > 3000:  D = EuDist2(fea[rand.sample(range(nSmp), 3000)],[],1)  else:  D = EuDist2(fea,[],1)  options['t'] = np.mean(D)  elif options['WeightMode'] == 'Cosine':  bCosine = 1  else:  print('WeightMode does not exist!')    if 'bSelfConnected' not in options:  options['bSelfConnected'] = 0    if 'gnd' in options:  nSmp =len(options['gnd'])  else:  nSmp = fea.shape[0]    maxM = 62500000  blockSize = maxM//(nSmp\*3)    if options['NeighborMode'] == 'Supervised':  label = np.unique(options['gnd'])  nLabel = len(label)  if options['bLDA'] == 1:  G = np.zeros((nSmp,nSmp))  for idx in range(nLabel):  classIdx = np.where(options['gnd'] == label[idx])[0]  for i in classIdx:  for j in classIdx:  G[i,j] = 1/len(classIdx)  W = coo\_matrix(G)  return W    if options['WeightMode'] == 'Binary':  if options['k'] > 0:  G = np.zeros((nSmp\*(options['k']+1),3))  idNow = 0  for i in range(nLabel):  classIdx = np.where(options['gnd']==label[i])[0]  D = EuDist2(fea[classIdx],[],0)  idx = np.argsort(D, axis=1)#satir ici siralama  idx = idx[:,0:options['k']+1]    nSmpClass = len(classIdx)\*(options['k']+1)  G[idNow:nSmpClass+idNow,0] = nm.repmat(classIdx, 1, options['k']+1)  G[idNow:nSmpClass+idNow,1] = classIdx[idx.flatten('F')]  G[idNow:nSmpClass+idNow,2] = 1  idNow = idNow+nSmpClass    G = coo\_matrix((G[:,2],(G[:,0],G[:,1])),shape=(nSmp, nSmp))  G = np.maximum(G.toarray(),G.toarray().transpose())  G = coo\_matrix(G)  else:  G = np.zeros((nSmp,nSmp))  for i in range(nLabel):  classIdx = np.where(options['gnd']==label[i])[0]  for i in classIdx:  for j in classIdx:  G[i,j] = 1  if options['bSelfConnected'] == 0:  if isinstance(G, (np.ndarray)) :  pass  else:  G = G.toarray()  for i in range(G.shape[0]):  G[i,i] = 0  W = coo\_matrix(G)    elif options['WeightMode'] == 'HeatKernel':  if options['k'] > 0:  G = np.zeros((nSmp\*(options['k']+1),3))  idNow = 0  for i in range(nLabel):  classIdx = np.where(options['gnd']==label[i])[0]  D = EuDist2(fea[classIdx],[],0)  idx = np.argsort(D, axis=1)  D.sort(axis=1)  idx = idx[:,0:options['k']+1]  dump = D[:,0:options['k']+1]  dump = np.exp(-dump/(2\*np.power(options['t'],2)))    nSmpClass = len(classIdx)\*(options['k']+1)  G[idNow:nSmpClass+idNow,0] = nm.repmat(classIdx, 1, options['k']+1)  G[idNow:nSmpClass+idNow,1] = classIdx[idx.flatten('F')]  G[idNow:nSmpClass+idNow,2] = dump.flatten('F')  idNow = idNow+nSmpClass    G = coo\_matrix((G[:,2],(G[:,0],G[:,1])),shape=(nSmp, nSmp))  else:  G = np.zeros((nSmp,nSmp))  for i in range(nLabel):  classIdx = np.where(options['gnd']==label[i])[0]  D = EuDist2(fea[classIdx],[],0)  D = np.exp(-D/(2\*np.power(options['t'],2)))  for j,m in enumerate(classIdx):  for k,n in enumerate(classIdx):  G[m,n] = D[j,k]    if options['bSelfConnected'] == 0:  if isinstance(G, (np.ndarray)) :  pass  else:  G = G.toarray()  for i in range(G.shape[0]):  G[i,i] = 0    W = coo\_matrix(np.maximum(G,G.transpose()))  elif options['WeightMode'] == 'Cosine':  if options['bNormalized'] == 0:  fea = NormalizeFea(fea,1)  if options['k'] > 0:  G = np.zeros((nSmp\*(options['k']+1),3))  idNow = 0  for i in range(nLabel):  classIdx = np.where(options['gnd']==label[i])[0]  D = np.dot(fea[classIdx],np.transpose(fea[classIdx]))  idx = np.argsort(-D, axis=1)#satir ici siralama  D = -D  D.sort(axis=1)  idx = idx[:,0:options['k']+1]  dump = -D[:,0:options['k']+1]    nSmpClass = len(classIdx)\*(options['k']+1)  G[idNow:nSmpClass+idNow,0] = nm.repmat(classIdx, 1, options['k']+1)  G[idNow:nSmpClass+idNow,1] = classIdx[idx.flatten('F')]  G[idNow:nSmpClass+idNow,2] = dump.flatten('F')  idNow = idNow+nSmpClass  G = coo\_matrix((G[:,2],(G[:,0],G[:,1])),shape=(nSmp, nSmp))  else:  G = np.zeros((nSmp,nSmp))  for i in range(nLabel):  classIdx = np.where(options['gnd']==label[i])[0]  D = np.dot(fea[classIdx,:],np.transpose(fea[classIdx,:]))  for j,m in enumerate(classIdx):  for k,n in enumerate(classIdx):  G[m,n] = D[j,k]  if options['bSelfConnected'] == 0:  if isinstance(G, (np.ndarray)) :  pass  else:  G = G.toarray()  for i in range(G.shape[0]):  G[i,i] = 0  W = coo\_matrix(np.maximum(G,G.transpose()))  else:  print('WeightMode does not exist!')  return W  if (bCosine == 1 and options['bNormalized']==0):  Normfea = NormalizeFea(fea,1)  if options['NeighborMode'] == 'KNN' and options['k']>0:  if not (bCosine and options['bNormalized']):  G = np.zeros((nSmp\*(options['k']+1),3))  for i in range(math.ceil(nSmp/blockSize)):  if i == math.ceil(nSmp/blockSize) - 1:  smpIdx = np.arange(i\*blockSize,nSmp)  dist = EuDist2(fea[smpIdx,:],fea,0)  if bSpeed == 1:  nSmpNow = len(smpIdx);  dump = np.zeros((nSmpNow,options['k']+1))  idx = np.zeros((nSmpNow,options['k']+1))  for j in range(options['k']+1):  dump[:,j] = np.min(dist, axis=1)  idx[:,j] = np.argmin(dist, axis=1)  temp = idx[:,j]\*nSmpNow+np.arange(nSmpNow)  sizeTemp = dist.shape[0]  sizeTemp2 = dist.shape[1]  tempDist = dist.flatten('F')  for x in temp:  tempDist[int(x)] = np.exp(100)  dist = tempDist.reshape(sizeTemp2,sizeTemp)  dist = np.transpose(dist)  else:  idx = np.argsort(dist, axis=1)#satir ici siralama  dist.sort(axis=1)  dump = dist[:,0:options['k']+1]  idx = idx[:,0:options['k']+1]  if bBinary == 0:  if bCosine == 1:  dist = np.dot(Normfea[smpIdx,:],np.transpose(Normfea))  linidx = np.transpose(np.arange(0,len(idx))).reshape(-1,1)  sub2ind = idx\*len(dist)+linidx[:,[0]\*len(idx[0])]  tempDistance = dist.flatten('F')  for rr in range(len(sub2ind[0])):  dump[:,rr] = tempDistance[np.int\_(sub2ind[:,rr])]  else:  dump = np.exp(-dump/(2\*np.power(options['t'],2)))  G[np.arange(i\*blockSize\*(options['k']+1),nSmp\*(options['k']+1)),0] = nm.repmat(smpIdx.reshape(-1,1),options['k']+1,1).flatten('F')  G[np.arange(i\*blockSize\*(options['k']+1),nSmp\*(options['k']+1)),1] = idx.flatten('F')  if bBinary == 0:  G[np.arange(i\*blockSize\*(options['k']+1),nSmp\*(options['k']+1)),2] = dump.flatten('F')  else:  G[np.arange(i\*blockSize\*(options['k']+1),nSmp\*(options['k']+1)),2] = 1  else:  smpIdx = np.arange(i\*blockSize,(i+1)\*blockSize)  dist = EuDist2(fea[smpIdx,:],fea,0)    if bSpeed == 1:  nSmpNow = len(smpIdx);  dump = np.zeros((nSmpNow,options['k']+1))  idx = np.zeros((nSmpNow,options['k']+1))  for j in range(options['k']+1):  dump[:,j] = np.min(dist, axis=1)  idx[:,j] = np.argmin(dist, axis=1)  temp = idx[:,j]\*nSmpNow+np.arange(nSmpNow)  sizeTemp = dist.shape[0]  sizeTemp2 = dist.shape[1]  tempDist = dist.flatten('F')  for x in temp:  tempDist[int(x)] = np.exp(100)  dist = tempDist.reshape(sizeTemp2,sizeTemp)  dist = np.transpose(dist)  else:  idx = np.argsort(dist, axis=1)#satir ici siralama  dist.sort(axis=1)  dump = dist[:,0:options['k']+1]  idx = idx[:,0:options['k']+1]  if bBinary == 0:  if bCosine == 1:  dist = np.dot(Normfea[smpIdx,:],np.transpose(Normfea))  linidx = np.transpose(np.arange(0,len(idx))).reshape(-1,1)  sub2ind = idx\*len(dist)+linidx[:,[0]\*len(idx[0])]  tempDistance = dist.flatten('F')  for rr in range(len(sub2ind[0])):  dump[:,rr] = tempDistance[np.int\_(sub2ind[:,rr])]  else:  dump = np.exp(-dump/(2\*np.power(options['t'],2)))  G[np.arange(i\*blockSize\*(options['k']+1),(i+1)\*blockSize\*(options['k']+1)),0] = nm.repmat(smpIdx.reshape(-1,1),options['k']+1,1).flatten('F')  G[np.arange(i\*blockSize\*(options['k']+1),(i+1)\*blockSize\*(options['k']+1)),1] = idx.flatten('F')  if bBinary == 0:  G[np.arange(i\*blockSize\*(options['k']+1),(i+1)\*blockSize\*(options['k']+1)),2] = dump.flatten('F')  else:  G[np.arange(i\*blockSize\*(options['k']+1),(i+1)\*blockSize\*(options['k']+1)),2] = 1  W = coo\_matrix((G[:,2],(G[:,0],G[:,1])),shape=(nSmp, nSmp))  else:  G = np.zeros((nSmp\*(options['k']+1),3))  for i in range(math.ceil(nSmp/blockSize)):  if i == math.ceil(nSmp/blockSize) - 1:  smpIdx = np.arange(i\*blockSize,nSmp)  dist = np.dot(fea[smpIdx,:],np.transpose(fea))  if bSpeed == 1:  nSmpNow = len(smpIdx);  dump = np.zeros((nSmpNow,options['k']+1))  idx = np.zeros((nSmpNow,options['k']+1))  for j in range(options['k']+1):  dump[:,j] = np.max(dist, axis=1)  idx[:,j] = np.argmax(dist, axis=1)  temp = idx[:,j]\*nSmpNow+np.arange(nSmpNow)  sizeTemp = dist.shape[0]  sizeTemp2 = dist.shape[1]  tempDist = dist.flatten('F')  for x in temp:  tempDist[int(x)] = 0  dist = tempDist.reshape(sizeTemp2,sizeTemp)  dist = np.transpose(dist)  else:  idx = np.argsort(-dist, axis=1)#satir ici siralama  dist = -dist  dist.sort(axis=1)  idx = idx[:,0:options['k']+1]  dump = -dist[:,0:options['k']+1]  G[np.arange(i\*blockSize\*(options['k']+1),nSmp\*(options['k']+1)),0] = nm.repmat(smpIdx.reshape(-1,1),options['k']+1,1).flatten('F')  G[np.arange(i\*blockSize\*(options['k']+1),nSmp\*(options['k']+1)),1] = idx.flatten('F')  G[np.arange(i\*blockSize\*(options['k']+1),nSmp\*(options['k']+1)),2] = dump.flatten('F')  else:  smpIdx = np.arange(i\*blockSize,(i+1)\*blockSize)  dist = np.dot(fea[smpIdx,:],np.transpose(fea))  if bSpeed == 1:  nSmpNow = len(smpIdx);  dump = np.zeros((nSmpNow,options['k']+1))  idx = np.zeros((nSmpNow,options['k']+1))  for j in range(options['k']+1):  dump[:,j] = np.max(dist, axis=1)  idx[:,j] = np.argmax(dist, axis=1)  temp = idx[:,j]\*nSmpNow+np.arange(nSmpNow)  sizeTemp = dist.shape[0]  sizeTemp2 = dist.shape[1]  tempDist = dist.flatten('F')  for x in temp:  tempDist[int(x)] = 0  dist = tempDist.reshape(sizeTemp2,sizeTemp)  dist = np.transpose(dist)  else:  idx = np.argsort(-dist, axis=1)#satir ici siralama  dist = -dist  dist.sort(axis=1)  idx = idx[:,0:options['k']+1]  dump = -dist[:,0:options['k']+1]  G[np.arange(i\*blockSize\*(options['k']+1),(i+1)\*blockSize\*(options['k']+1)),0] = nm.repmat(smpIdx.reshape(-1,1),options['k']+1,1).flatten('F')  G[np.arange(i\*blockSize\*(options['k']+1),(i+1)\*blockSize\*(options['k']+1)),1] = idx.flatten('F')  G[np.arange(i\*blockSize\*(options['k']+1),(i+1)\*blockSize\*(options['k']+1)),2] = dump.flatten('F')  W = coo\_matrix((G[:,2],(G[:,0],G[:,1])),shape=(nSmp, nSmp))  if bBinary == 1:  W[W!=0]=1  if options['bSelfConnected'] == 0:  W = W - np.diag(np.diag(W.toarray()))  if 'bTrueKNN' in options:  pass  else:  W = np.maximum(W,np.transpose(W))  return W |

numpy.unique Bir dizinin benzersiz öğelerini bulun.

dimensionality reduction boyutluluk indirgeme

#NumPy, Python programlama dili için bir kütüphanedir;

#büyük, çok boyutlu diziler ve matrisler için destek ekler ve

#bu dizilerde çalışacak geniş bir üst düzey matematiksel

#işlev koleksiyonu sunar.

#SciPy, Python’un Numpy uzantısına dayanan matematiksel algoritmalar ve kolaylık işlevlerinin bir koleksiyonudur.

#linalg Linear algebra

# K-en yakın komşuların oylamasını uygulayan sınıflandırıcı.

# kneighbors\_graph ....'teki noktalar için k-Komşular'ın (ağırlıklı) grafiğini hesaplar

# NearestNeighborsDenetimsiz en yakın komşu öğrenmesini uygular.

# utils.check\_array(dizi[, kabul\_sparse, ...])

#Bir dizi, liste, seyrek matris veya benzeri üzerinde giriş doğrulaması.

#base.BaseEstimator scikit-learn'deki tüm tahmin ediciler için temel sınıf.

#base.TransformerMixin Scikit-learn'deki tüm transformatörler için mixin sınıfı.

#Self adında bir sınıf oluşturun, n\_components ve n\_neighbors

#için değerler atamak için \_\_init\_\_() işlevini kullanın:

# TODO: allow for precomputed weights

# YAPILACAKLAR: önceden hesaplanmış ağırlıklara izin ver

# TODO: allow for precomputed weights

# YAPILACAKLAR: yarıçap komşularına izin ver

#modeldeki bazı parametreleri tahmin etmektir.

#Bu fit()yöntemde uygulanmaktadır.

#\_compute\_weight ağırlık hesaplama

#check\_array Bir dizi, liste, seyrek matris veya benzeri girdi doğrulaması.

#Verileri denetimli veya denetimsiz bir şekilde filtrelemek veya değiştirmek için:

https://medium.com/@sddkal/cpp-eigen-k%C3%BCt%C3%BCphanesi-ile-temel-bile%C5%9Fen-analizi-pca-7cdaf37155ef