

Radon Cumulative Distribution Transform

This notebook implements the Radon Cumulative Distribution Transform (Radon-CDT) and its inverse and showcases an example application of this transformation.

CDT Definition: ¶

Consider two nonzero probability density functions I_0 and I_1 defined on $X, Y \subset \mathbb{R}$. Considering I_0 to be a pre-determined 'reference' density, one can use the following relation,

$$\int_{\inf(Y)}^{f(x)} I_1(\tau) d\tau = \int_{\inf(X)}^x I_0(\tau) d\tau \quad (1)$$

to uniquely associate $f_1 : X \rightarrow Y$ to the given density I_1 . We use this relationship to define the Cumulative Distribution Transform (CDT) of I_1 (denoted as $\hat{I}_1 : X \rightarrow \mathbb{R}$), with respect to the reference I_0 :

$$\hat{I}_1(x) = (f_1(x) - x) \sqrt{I_0(x)}. \quad (2)$$

Now let $J_0 : X \rightarrow [0, 1]$ and $J_1 : Y \rightarrow [0, 1]$ be the corresponding cumulative distribution functions for I_0 and I_1 , that is: $J_0(x) = \int_{\inf(X)}^x I_0(\tau) d\tau$, $J_1(y) = \int_{\inf(Y)}^y I_1(\tau) d\tau$. Equation (1) can then be rewritten as,

$$J_0(x) = J_1(f_1(x)) \Rightarrow f_1(x) = J_1^{-1}(J_0(x)). \quad (3)$$

For continuous cumulative distribution functions J_0 and J_1 , f_1 is a continuous and monotonic function. If f_1 is differentiable, Equation (3) can be rewritten as

$$I_0(x) = f_1'(x) I_1(f_1(x)). \quad (4)$$

Inverse-CDT Definition:

The Inverse-CDT of \hat{I}_1 is defined as:

$$I_1(y) = \frac{d}{dy} J_0(f_1^{-1}(y)) = (f_1^{-1})' I_0(f_1^{-1}(y))$$

where $f_1^{-1} : Y \rightarrow X$ refers to the inverse of f_1 (i.e. $f_1^{-1}(f_1(x)) = x$), $f(x) = \hat{I}_1(x) / \sqrt{I_0(x)} + x$, and where $I_1 : Y \rightarrow \mathbb{R}$ as before. Naturally, the equation above holds for points where J_0 and f are differentiable. By the construction above, f will be differentiable except for points where I_0 and I_1 are discontinuous.

Radon-CDT Definition:

Radon-CDT extends the concept of CDT to higher dimensional probability distributions, by projecting the high-dimensional pdf into a set of one-dimensional pdfs via Radon transform and applying CDT to these projections. To define Radon-CDT we first define the Radon transform. The d -dimensional Radon transform \mathcal{R} maps a function $I \in L^1(\mathbb{R}^d)$ where $L^1(\mathbb{R}^d) := \{I : \mathbb{R}^d \rightarrow \mathbb{R} \mid \int_{\mathbb{R}^d} |I(x)| dx \leq \infty\}$ into the set of its integrals over the hyperplanes of \mathbb{R}^n and is defined as,

$$\mathcal{R}I(t, \theta) := \int_{\mathbb{R}} I(t\theta + \gamma\theta^\perp) d\gamma$$

here θ^\perp is the subspace or unit vector orthogonal to θ . Note that $\mathcal{R} : L^1(\mathbb{R}^d) \rightarrow L^1(\mathbb{R} \times \mathbb{S}^{d-1})$, where \mathbb{S}^{d-1} is the unit sphere in \mathbb{R}^d . We note that the Radon transform is an invertible, linear transform and we denote its inverse as \mathcal{R}^{-1} . For brevity we do not define the inverse Radon transform here, but the details can be found in

(Natterer, Frank. The mathematics of computerized tomography. Society for Industrial and Applied Mathematics, 2001.).

Given a template distribution $I_0 \in L^1(\mathbb{R}^d)$, the Radon-CDT of the probability distribution $I \in L^1(\mathbb{R}^d)$ is then defined as:

$$\tilde{I}(t, \theta) = (f(t, \theta) - t) \sqrt{\mathcal{R}I_0(t, \theta)}$$

where $f(t, \theta)$ satisfies:

$$\int_{-\infty}^{f(t, \theta)} \mathcal{R}I_1(\tau, \theta) d\tau = \int_{-\infty}^t \mathcal{R}I_0(\tau, \theta) d\tau, \forall \theta \in \mathbb{S}^{d-1}$$

Inverse Radon-CDT Definition:

The inverse Radon-CDT is defined as:

$$I = \mathcal{R}^{-1}(\det(Dg) \mathcal{R}I_0(g))$$

where $\det(Dg)$ is the determinant of Jacobian of g and $g : \mathbb{R} \times \mathbb{S}^{d-1} \rightarrow \mathbb{R} \times \mathbb{S}^{d-1}$ is defined as, $g(t, \theta) = (f^{-1}(t, \theta), \theta)$.

Now we implement Radon-CDT in python

```
In [1]: import numpy as np
import matplotlib.pyplot as plt
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as
LDA
from skimage.transform import radon, iradon
from scipy import interp
from skimage.io import imread
from scipy.ndimage import filters
from sklearn.svm import LinearSVC
from matplotlib import colors
from skimage.color import rgb2gray
%matplotlib inline
```

```
/Users/skolouri/anaconda/lib/python2.7/site-packages/matplotlib/font_ma
nager.py:273: UserWarning: Matplotlib is building the font cache using
fc-list. This may take a moment.
  warnings.warn('Matplotlib is building the font cache using fc-list. T
his may take a moment.')
```

Define CDT and its inverse

```

In [2]: def CDT(I0,I1):
    assert I0.shape==I1.shape
    assert not((1.0*I0<0).sum() and (1.0*I1<0).sum())
    I0=I0/I0.sum()
    I1=I1/I1.sum()
    cI0=np.cumsum(I0)
    cI1=np.cumsum(I1)
    x=np.asarray(range(len(I0)))
    xtilde=np.linspace(0,1,len(I0))
    XI0 = interp(xtilde,cI0, x)
    XI1 = interp(xtilde,cI1, x)
    u = interp(x,XI0,XI0-XI1)
    f = x - u
    return u,f
def iCDT(u,I0):
    x=np.asarray(range(len(I0)))
    f=x-u
    fprime=np.gradient(f)
    I1 = interp(x,f, I0/fprime)
    return I1
def RadonCDT(I1,theta=np.asarray(range(180))):
    R1 = radon(I1, theta=theta, circle=False)
    R0 = np.ones_like(R1)
    rcdt=[]
    for i in range(len(theta)):
        u,f=CDT(R0[:,i],R1[:,i])
        rcdt.append(u)
    rcdt=np.asarray(rcdt).T
    return rcdt
def iRadonCDT(rcdt,theta=np.asarray(range(180))):
    J0=np.ones([rcdt.shape[0]])
    J0=J0/J0.sum()
    R=[]
    for i in range(rcdt.shape[1]):
        R.append(iCDT(rcdt[:,i],J0))
    R=np.asarray(R).T
    I=iradon(R,theta=theta)
    return I

```

To demonstrate the nonlinear nature of our transformation we simply take the average of two 2D images in the image space and in the Radon-CDT space

```

In [17]: I1=np.zeros((128,128))
x,y=np.random.uniform(20,108,(2,)).astype('int')
I1[x,y]=1
I1=filters.gaussian_filter(I1,sigma=10)
I2=np.zeros((128,128))
x,y=np.random.uniform(20,108,(2,)).astype('int')
I2[x,y]=1
I2=filters.gaussian_filter(I2,sigma=10)

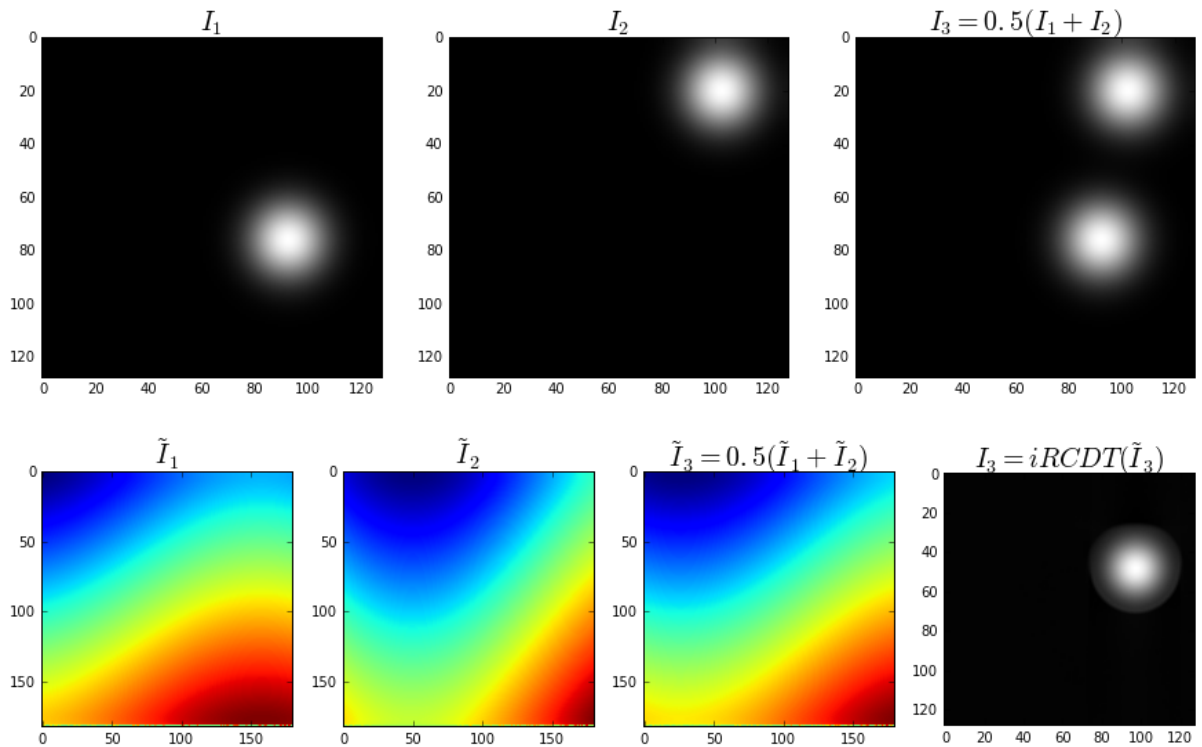
```

```
In [18]: RI1=RadonCDT(I1)
         RI2=RadonCDT(I2)
```

```
In [19]: I3=0.5*(I1+I2)
         RI3=0.5*(RI1+RI2)
         iRI3=iRadonCDT(RI3)
```

```
In [20]: fig,[ax1,ax2,ax3]=plt.subplots(1,3,figsize=(15,15))
         ax1.imshow(I1,cmap='gray')
         ax1.set_title('$I_1$',fontsize=20)
         ax2.imshow(I2,cmap='gray')
         ax2.set_title('$I_2$',fontsize=20)
         ax3.imshow(I3,cmap='gray')
         ax3.set_title('$I_3=0.5(I_1+I_2)$',fontsize=20)
         plt.show()
         fig,[ax1,ax2,ax3,ax4]=plt.subplots(1,4,figsize=(15,15))
         ax1.imshow(RI1)
         ax1.set_title('$\tilde{I}_1$',fontsize=20)
         ax2.imshow(RI2)
         ax2.set_title('$\tilde{I}_2$',fontsize=20)
         ax3.imshow(RI3)
         ax3.set_title('$\tilde{I}_3=0.5(\tilde{I}_1+\tilde{I}_2)$',fontsize=20)
         ax4.imshow(iRI3,cmap='gray')
         ax4.set_title('$I_3=iRCDT(\tilde{I}_3)$',fontsize=20)

         plt.show()
```



```
In [7]: N=1000
C1=np.zeros((N,128,128))
C2=np.zeros((N,128,128))
C3=np.zeros((N,128,128))
K,L=RadonCDT(1+C1[0,:,:]).shape

RC1=np.zeros((N,K,L))
RC2=np.zeros((N,K,L))
RC3=np.zeros((N,K,L))

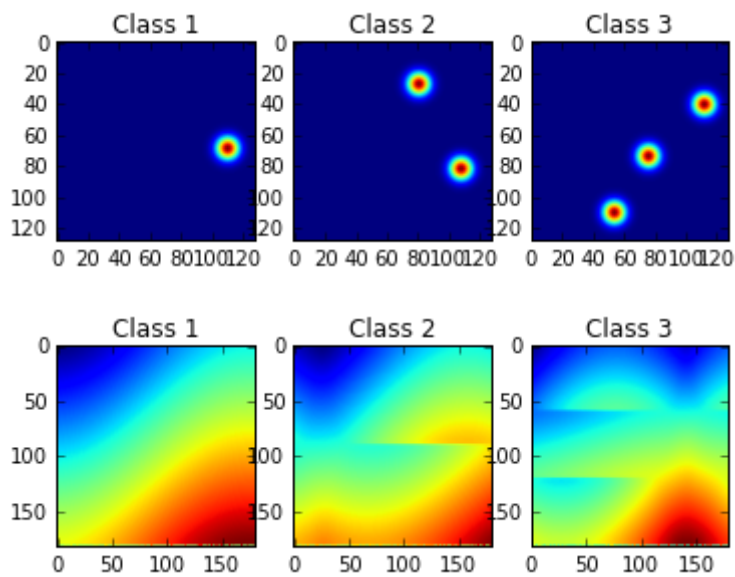
for i in range(N):
    x,y=np.random.uniform(10,118,(2,)).astype('int')
    C1[i,x,y]=1
    C1[i,:,:]=filters.gaussian_filter(C1[i,:,:],sigma=5)

    x,y=np.random.uniform(10,118,(2,)).astype('int')
    C2[i,x,y]=0.5
    x,y=np.random.uniform(10,118,(2,)).astype('int')
    C2[i,x,y]=0.5
    C2[i,:,:]=filters.gaussian_filter(C2[i,:,:],sigma=5)

    x,y=np.random.uniform(10,118,(2,)).astype('int')
    C3[i,x,y]=0.333
    x,y=np.random.uniform(10,118,(2,)).astype('int')
    C3[i,x,y]=0.333
    x,y=np.random.uniform(10,118,(2,)).astype('int')
    C3[i,x,y]=0.333
    C3[i,:,:]=filters.gaussian_filter(C3[i,:,:],sigma=5)

    RC1[i,:,:]=RadonCDT(C1[i,:,:])
    RC2[i,:,:]=RadonCDT(C2[i,:,:])
    RC3[i,:,:]=RadonCDT(C3[i,:,:])
```

```
In [8]: fig,[ax1,ax2,ax3]=plt.subplots(1,3)
ax1.imshow(C1[0,:,:])
ax1.set_title('Class 1')
ax2.imshow(C2[0,:,:])
ax2.set_title('Class 2')
ax3.imshow(C3[0,:,:])
ax3.set_title('Class 3')
plt.show()
fig,[ax1,ax2,ax3]=plt.subplots(1,3)
ax1.imshow(RC1[0,:,:])
ax1.set_title('Class 1')
ax2.imshow(RC2[0,:,:])
ax2.set_title('Class 2')
ax3.imshow(RC3[0,:,:])
ax3.set_title('Class 3')
plt.show()
```



```
In [9]: X=np.concatenate((np.concatenate((np.reshape(C1,
[N,128*128]),np.reshape(C2,[N,128*128])),0),np.reshape(C3,
[N,128*128])),0)
Xhat=np.concatenate((np.concatenate((np.reshape(RC1,
[N,K*L]),np.reshape(RC2,[N,K*L])),0),np.reshape(RC3,[N,K*L])),0)
label=np.concatenate((np.concatenate((np.zeros(N),np.ones(N)),0),2*np.on
es(N)),0)
```

```
In [10]: lda=LDA(n_components=2)
```

```

In [11]: n=N
# Apply LDA and show classification boundaries in Signal Space
Xlda=lda.fit_transform(X,label)
svm=LinearSVC()
svm.fit(Xlda,label)

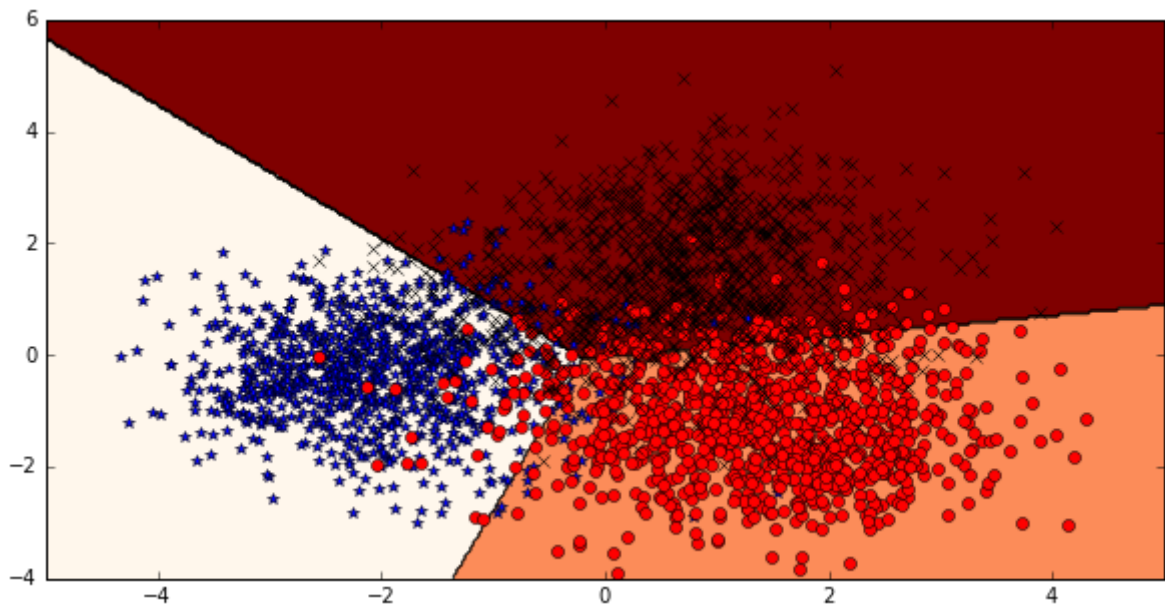
plt.figure(figsize=(10,5))
plt.plot(Xlda[:n,0],Xlda[:n,1],'b*')
plt.plot(Xlda[n:2*n,0],Xlda[n:2*n,1],'ro')
plt.plot(Xlda[2*n:,0],Xlda[2*n:,1],'kx')
x_min, x_max = plt.xlim()
y_min, y_max = plt.ylim()
nx, ny = 400, 200
xx, yy = np.meshgrid(np.linspace(x_min, x_max, nx),
                     np.linspace(y_min, y_max, ny))
Z = svm.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z[:].reshape(xx.shape)
plt.pcolormesh(xx, yy, Z, cmap='OrRd')
plt.contour(xx, yy, Z, linewidths=.5, colors='k')
plt.show()

```

```

/Users/skolouri/anaconda/lib/python2.7/site-packages/sklearn/discrimina
nt_analysis.py:387: UserWarning: Variables are collinear.
  warnings.warn("Variables are collinear.")

```

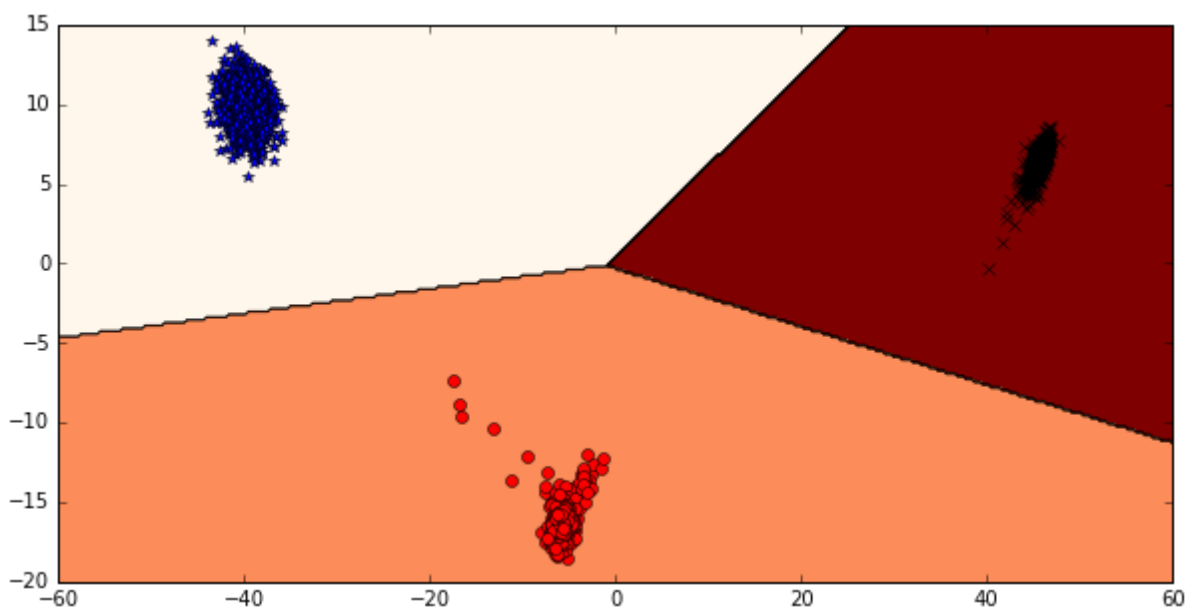



```

In [12]: # Apply LDA and show classification boundaries in the transform Space
Xhatlda=lda.fit_transform(Xhat,label)
svm=LinearSVC()
svm.fit(Xhatlda,label)

plt.figure(figsize=(10,5))
plt.plot(Xhatlda[:n,0],Xhatlda[:n,1],'b*')
plt.plot(Xhatlda[n:2*n,0],Xhatlda[n:2*n,1],'ro')
plt.plot(Xhatlda[2*n:,0],Xhatlda[2*n:,1],'kx')
x_min, x_max = plt.xlim()
y_min, y_max = plt.ylim()
nx, ny = 400, 200
xx, yy = np.meshgrid(np.linspace(x_min, x_max, nx),
                      np.linspace(y_min, y_max, ny))
Z = svm.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z[:].reshape(xx.shape)
plt.pcolormesh(xx, yy, Z, cmap='OrRd')
plt.contour(xx, yy, Z, linewidths=.5, colors='k')
plt.show()

```



In []: