blatt04_nitschke_grisard

November 29, 2018

1 Blatt 4

1.1 Aufgabe 10: Zwei Populationen

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

In [2]:
```
mux0 = 0
muy0 = 3
sigx0 = 3.5
sigy0 = 2.6
cor0 = 0.9
cov0 = cor0 * sigx0 * sigy0

cov_mat0 = np.array([[sigx0**2, cov0], [cov0, sigy0**2]])
```

In [3]:
```
population0_10000 = np.random.multivariate_normal([mux0, muy0], cov_mat0, 10000)
```

In [4]:
```
mux1 = 6
sigx1 = 3.5
a = -0.5
b = 0.6
var_yx = 1

muy1 = a + b * mux1
sigy1 = np.sqrt(b**2 * sigx1**2 + var_yx)
cor1 = np.sqrt(b**2 * sigx1**2 / sigy1**2)

cov1 = cor1 * sigx1 * sigy1
cov_mat1 = np.array([[sigx1**2, cov1], [cov1, sigy1**2]])

print(muy1, sigy1, cor1)
```

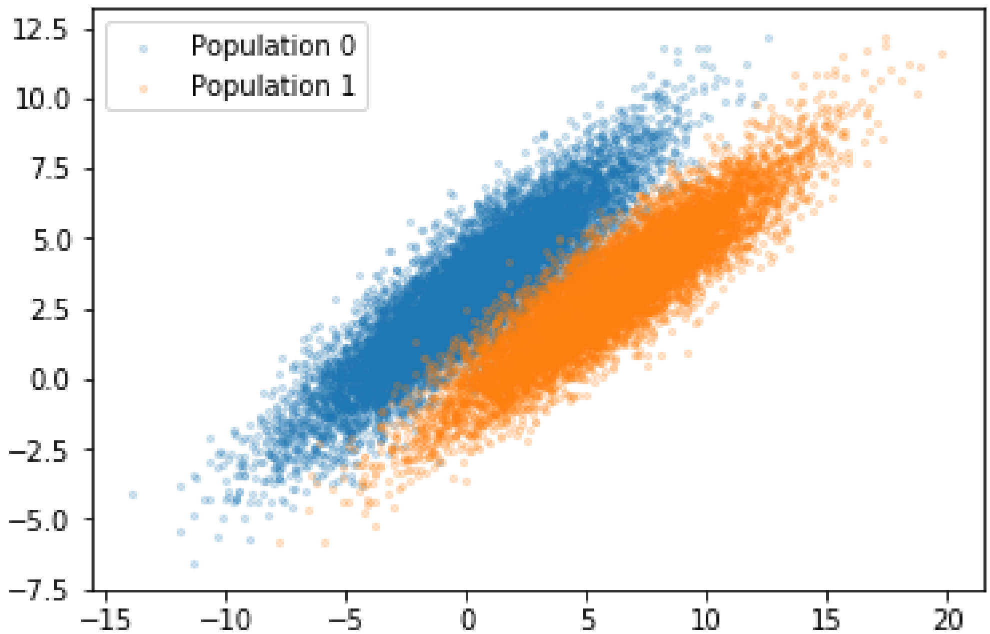
```
3.0999999999999996 2.3259406699226015 0.9028605188239304
```

In [5]:
```
population1 = np.random.multivariate_normal([mux1, muy1], cov_mat1, 10000)
```

b) Zeichne Scatter-Plots:

In [6]:
```
plt.scatter(population0_10000[:, 0], population0_10000[:, 1], s=5, alpha=0.2, label = 
plt.scatter(population1[:, 0], population1[:, 1], s=5, alpha=0.2, label = 'Population 1
plt.legend()
```

Out[6]: <matplotlib.legend.Legend at 0x7f0a670799e8>



In [7]:
```
population0_1000 = np.random.multivariate_normal([mux0, muy0], cov_mat0, 1000)

population0_10000_df = pd.DataFrame({
    'x': population0_10000[:, 0],
    'y': population0_10000[:, 1]
})

population0_1000_df = pd.DataFrame({
    'x': population0_1000[:, 0],
    'y': population0_1000[:, 1]
})

population1_df = pd.DataFrame({
    'x': population1[:, 0],
    'y': population1[:, 1]
})

population0_10000_df.to_hdf('sample.hdf5', key = 'population0_10000')
```

```
population0_1000_df.to_hdf('sample.hdf5', key = 'population0_1000')
population1_df.to_hdf('sample.hdf5', key = 'population1')
```

1.2 Aufgabe 11: Fisher-Diskriminante: Per Hand

siehe Abgabe

1.3 Aufgabe 12: Fisher-Diskriminante: Implementierung

Lade Daten:

```
In [8]: import pandas as pd
        P_0 = pd.read_hdf('sample.hdf5', key='population0_10000')
        P_1 = pd.read_hdf('sample.hdf5', key='population1')
        P_0.head()
```

```
Out[8]:           x         y
        0 -1.218659  3.137841
        1 -6.903642 -1.974535
        2  1.571834  3.171044
        3  0.575122  4.127489
        4 -2.033564  0.298584
```

a) Berechne Mittelwerte:

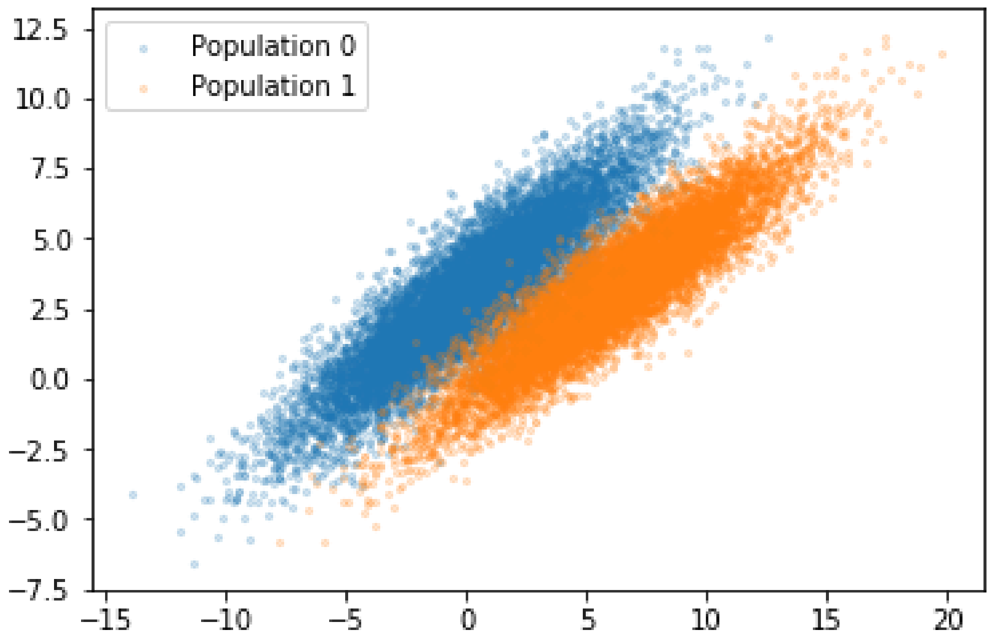
```
In [9]: mu0 = np.matrix([P_0.x.mean(), P_0.y.mean()]).T
        mu1 = np.matrix([P_1.x.mean(), P_1.y.mean()]).T
```

b) Berechne Kovarianzmatrizen:

```
In [10]: V_0 = P_0.cov()
         V_1 = P_0.cov()
```

```
In [11]: plt.scatter(P_0.x, P_0.y, s = 5, alpha = 0.2, label = 'Population 0')
         plt.scatter(P_1.x, P_1.y, s = 5, alpha = 0.2, label = 'Population 1')
         plt.legend()
```

```
Out[11]: <matplotlib.legend.Legend at 0x7f0a66fd9390>
```



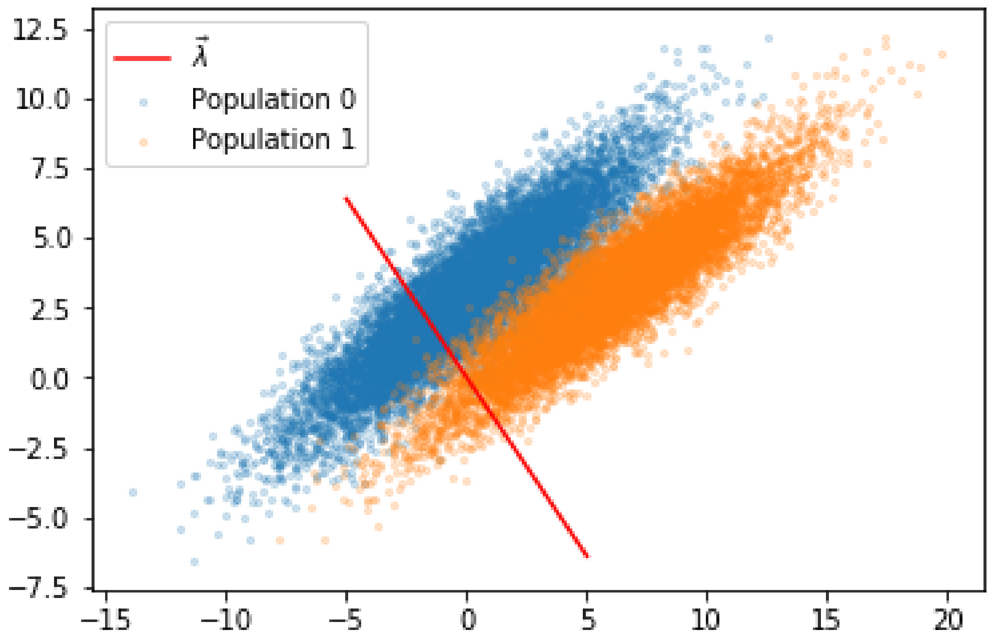
c) Konstruiere $\vec{\lambda}$:

```
In [12]: S_0 = np.sum([(xi.T - mu0) * (xi.T - mu0).T for xi in np.matrix(P_0)], axis = 0)
         S_1 = np.sum([(xi.T - mu1) * (xi.T - mu1).T for xi in np.matrix(P_1)], axis = 0)
         S_W = np.matrix(S_0 + S_1)

In [13]: lam = S_W.I * (mu1 - mu0)
         normed_lam = lam /  np.sqrt(lam[0]**2 + lam[1]**2)
         lam_array = np.array([lam[0], lam[1]])[:, 0]
         print(lam_array[1] / lam_array[0])

[-1.27746743]
```

Die Geradengleichung lautet:

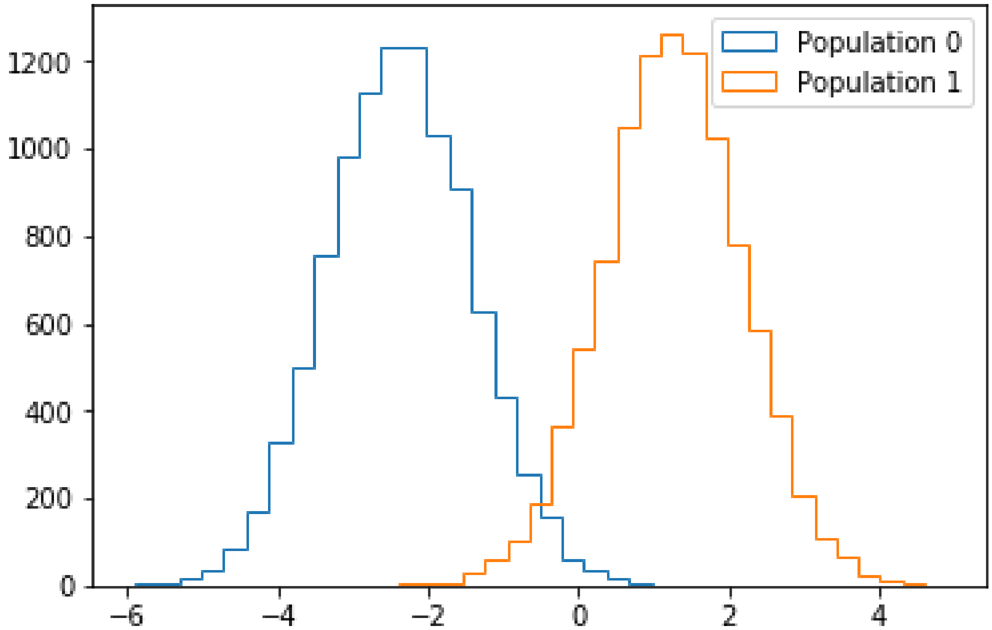
$$y(x) \approx -1.269 \cdot x$$

```
In [14]: xplot = np.linspace(-5, 5, 100)
         plt.plot(xplot, lam_array[1] / lam_array[0] * xplot,
                  color = 'red', label = r'$\vec{\lambda}$')

         plt.scatter(P_0.x, P_0.y, s = 5, alpha = 0.2, label = 'Population 0')
         plt.scatter(P_1.x, P_1.y, s = 5, alpha = 0.2, label = 'Population 1')
         plt.legend()

Out[14]: <matplotlib.legend.Legend at 0x7f0a61e2b7b8>
```



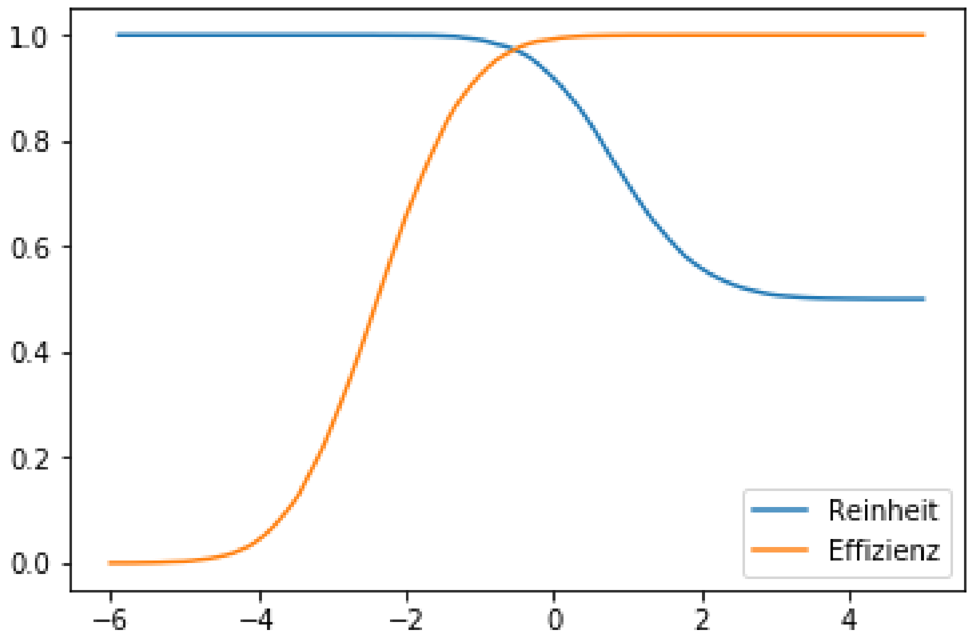
d) Stelle die Projektionen in einem Histogramm dar:

```
In [15]: projection_0 = np.array([(xi * normed_lam)[0, 0] for xi in np.matrix(P_0)])
         projection_1 = np.array([(xi * normed_lam)[0, 0] for xi in np.matrix(P_1)])

In [16]: plt.hist(projection_0, histtype = 'step', label = 'Population 0', bins = 25)
         plt.hist(projection_1, histtype = 'step', label = 'Population 1', bins = 25)
         plt.legend()

Out[16]: <matplotlib.legend.Legend at 0x7f0a61e04208>
```



e)

```
In [17]: def precision(signal, noise, cut):
             true_pos  = np.array([len(signal[signal < cut]) for cut in cut])
             false_pos = np.array([len(noise[noise < cut]) for cut in cut])
             return true_pos / (true_pos + false_pos)

In [18]: def recall(signal, noise, cut):
             true_pos  = np.array([len(signal[signal < cut]) for cut in cut])
             false_neg = np.array([len(signal[signal > cut]) for cut in cut])
             return true_pos / (true_pos + false_neg)

In [19]: signal = projection_0
         noise = projection_1

         lam_cut = np.linspace(-6, 5, 100)
         plt.plot(lam_cut, precision(signal, noise, lam_cut),
                  label = 'Reinheit')
         plt.plot(lam_cut, recall(signal, noise, lam_cut),
                  label = 'Effizienz')
         plt.legend()

/home/stefan/.local/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:4: RuntimeWarn
  after removing the cwd from sys.path.


Out[19]: <matplotlib.legend.Legend at 0x7f0a61f621d0>
```

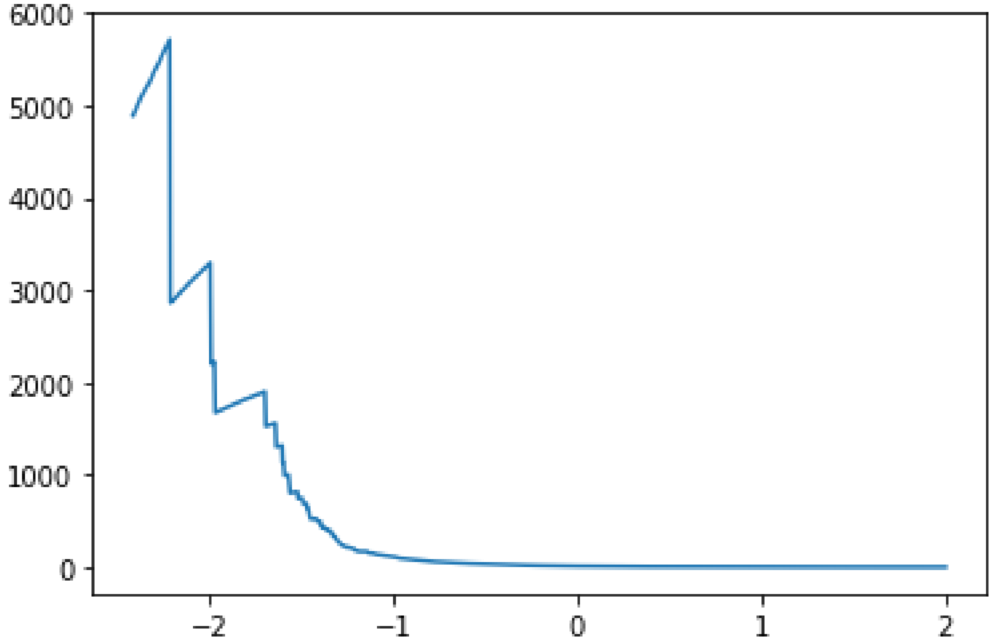


f) Untersuche Signal-Untergrundverhältnis:

```
In [20]: def signal_noise_ratio(signal, noise, cut):
             return np.array([len(signal[signal <= cut]) / len(noise[noise <= cut]) for cut in
```

```
In [21]: lam_cut = np.linspace(min(noise), 2, 1000)
         plt.plot(lam_cut, signal_noise_ratio(signal, noise, lam_cut),
                  label = 'Signal-Untergrundverhältnis')

         lam_cut[np.argmax(signal_noise_ratio(signal, noise, lam_cut))]
```

```
Out[21]: -2.211143186757372
```

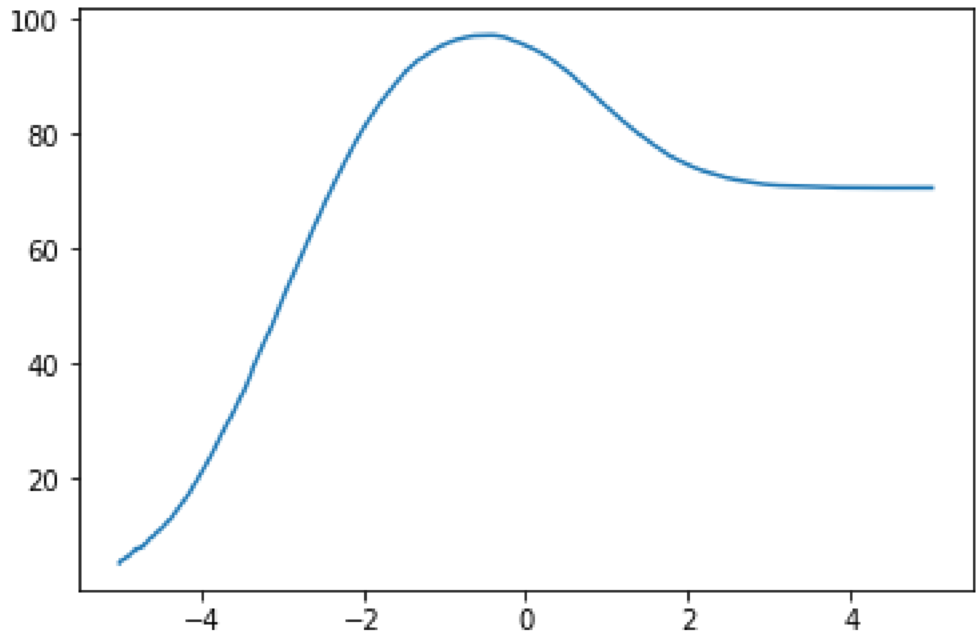


g) Untersuche Signifikanz:

```
In [22]: def sig(signal, noise, cut):
             return np.array([len(signal[signal <= cut]) /
                              np.sqrt(len(noise[noise <= cut]) + len(signal[signal <= cut])) fo
```

```
In [23]: lam_cut = np.linspace(-5, 5, 1000)
         plt.plot(lam_cut, sig(signal, noise, lam_cut),
                  label = 'Signifikanz')

         lam_cut[np.argmax(sig(signal, noise, lam_cut))]
```

```
Out[23]: -0.4554554554554553
```



1.3.1 Nun alles nochmal mit der kleineren Population

Lade Daten:

In [24]:
```
import pandas as pd
P_0 = pd.read_hdf('sample.hdf5', key='population0_1000')
P_1 = pd.read_hdf('sample.hdf5', key='population1')
P_0.head()
```

Out[24]:

	x	y
0	2.372637	2.539309
1	-0.395452	1.695328
2	-0.053511	3.521346
3	1.259727	4.528268
4	-2.598622	0.832334

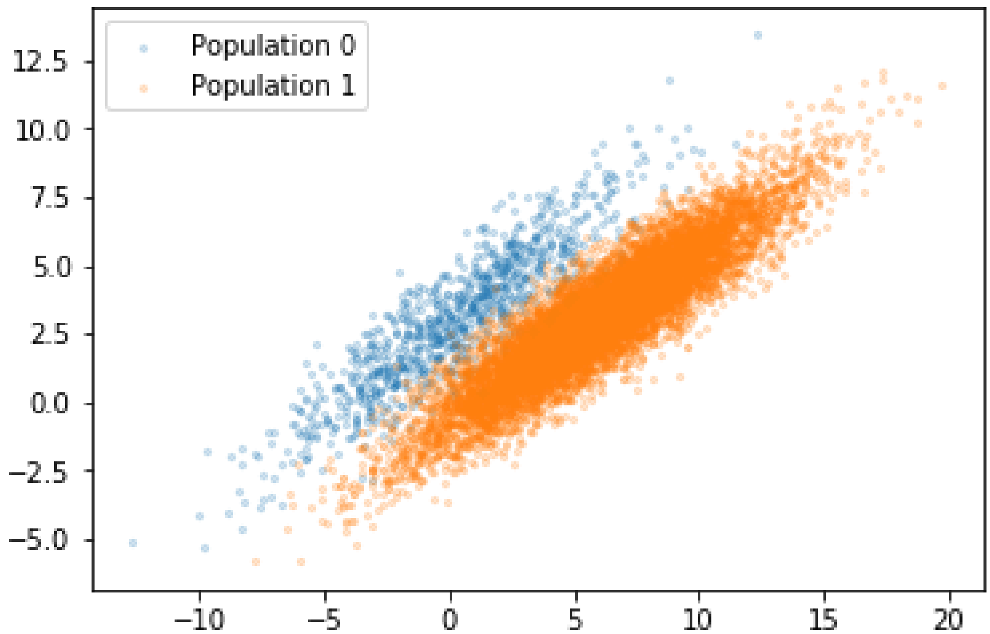
a) Berechne Mittelwerte:

In [25]:
```
mu0 = np.matrix([P_0.x.mean(), P_0.y.mean()]).T
mu1 = np.matrix([P_1.x.mean(), P_1.y.mean()]).T
```

b) Berechne Kovarianzmatrizen:

In [26]:
```
V_0 = P_0.cov()
V_1 = P_0.cov()
```

```
In [27]: plt.scatter(P_0.x, P_0.y, s = 5, alpha = 0.2, label = 'Population 0')
         plt.scatter(P_1.x, P_1.y, s = 5, alpha = 0.2, label = 'Population 1')
         plt.legend()

Out[27]: <matplotlib.legend.Legend at 0x7f0a61e9d978>
```



c) Konstruiere $\vec{\lambda}$:

```
In [28]: S_0 = np.sum([(xi.T - mu0) * (xi.T - mu0).T for xi in np.matrix(P_0)], axis = 0)
         S_1 = np.sum([(xi.T - mu1) * (xi.T - mu1).T for xi in np.matrix(P_1)], axis = 0)
         S_W = np.matrix(S_0 + S_1)

In [29]: lam = S_W.I * (mu1 - mu0)
         normed_lam = lam /  np.sqrt(lam[0]**2 + lam[1]**2)
         lam_array = np.array([lam[0], lam[1]])[:, 0]
         print(lam_array[1] / lam_array[0])

[-1.35197869]
```

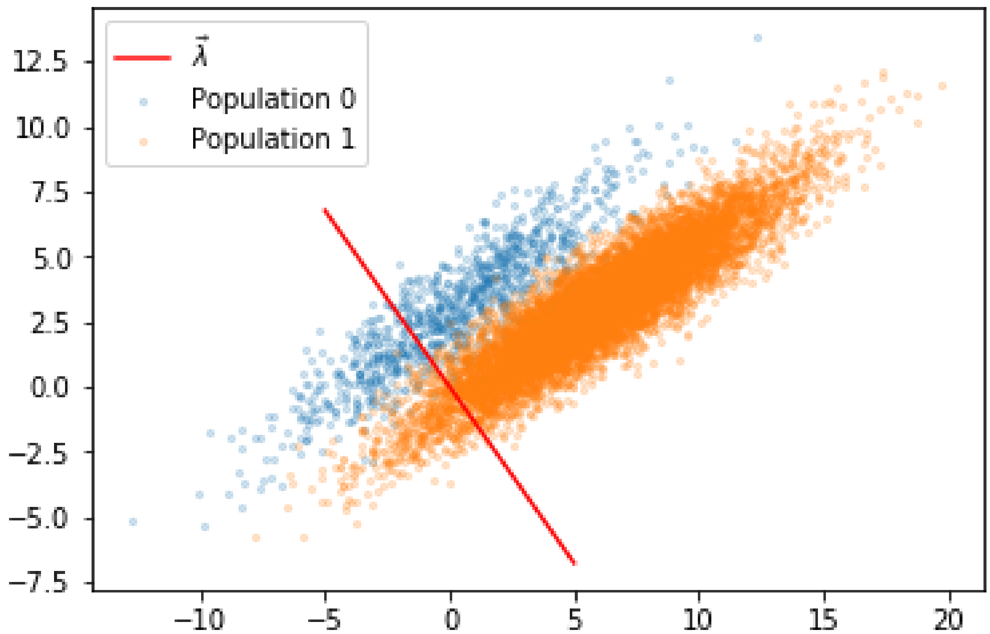
Die Geradengleichung lautet:

$$y(x) \approx -1.329 \cdot x$$

```
In [30]: xplot = np.linspace(-5, 5, 100)
         plt.plot(xplot, lam_array[1] / lam_array[0] * xplot,
                  color = 'red', label = r'$\vec{\lambda}$')
```

```
plt.scatter(P_0.x, P_0.y, s = 5, alpha = 0.2, label = 'Population 0')
plt.scatter(P_1.x, P_1.y, s = 5, alpha = 0.2, label = 'Population 1')
plt.legend()
```

Out[30]: <matplotlib.legend.Legend at 0x7f0a61f38198>

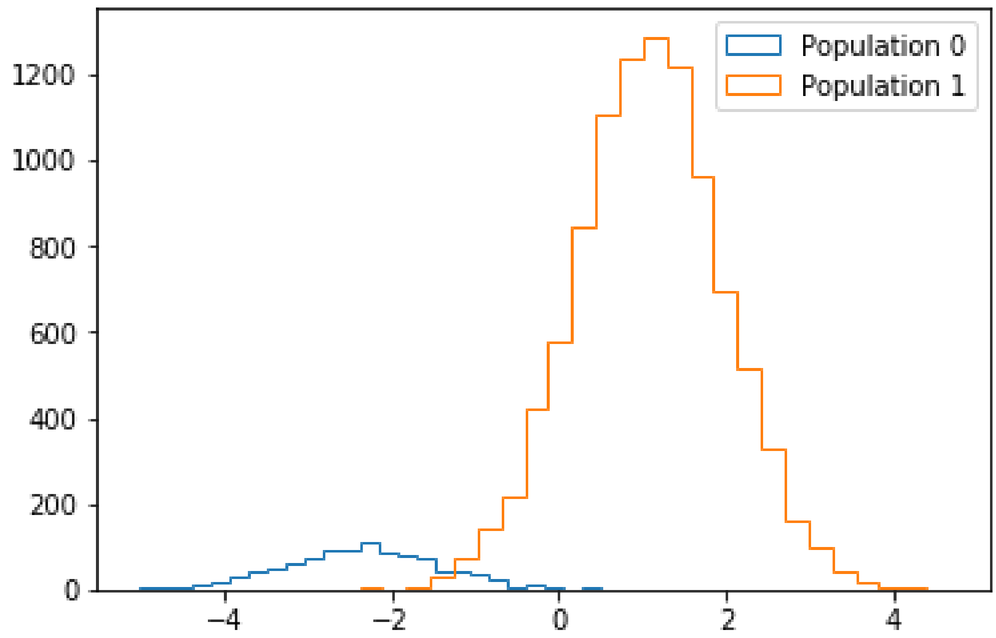


d) Stelle die Projektionen in einem Histogramm dar:

In [31]:
```
projection_0 = np.array([(xi * normed_lam)[0, 0] for xi in np.matrix(P_0)])
projection_1 = np.array([(xi * normed_lam)[0, 0] for xi in np.matrix(P_1)])
```

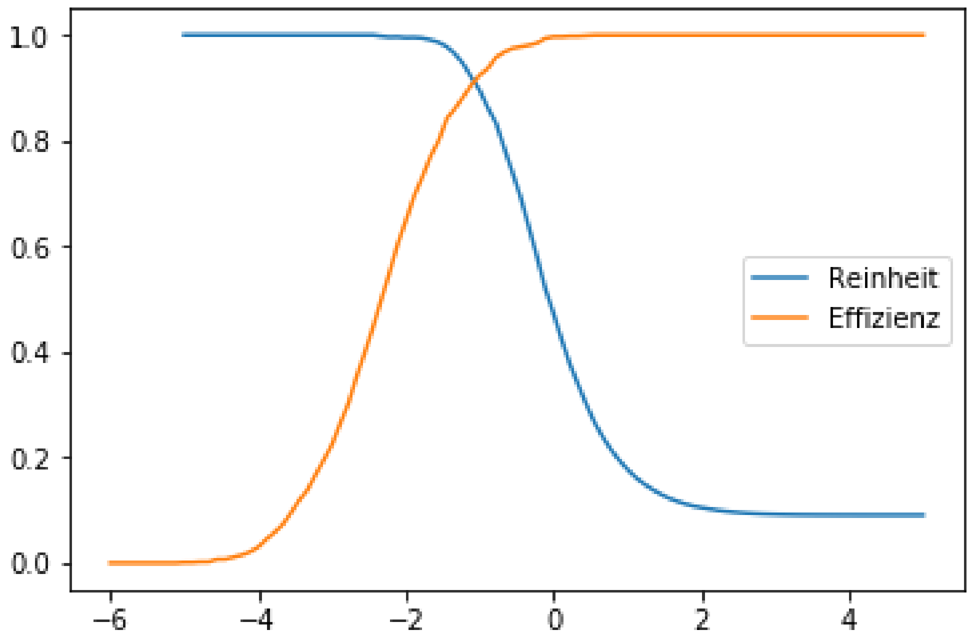
In [32]:
```
plt.hist(projection_0, histtype = 'step', label = 'Population 0', bins = 25)
plt.hist(projection_1, histtype = 'step', label = 'Population 1', bins = 25)
plt.legend()
```

Out[32]: <matplotlib.legend.Legend at 0x7f0a61f474e0>



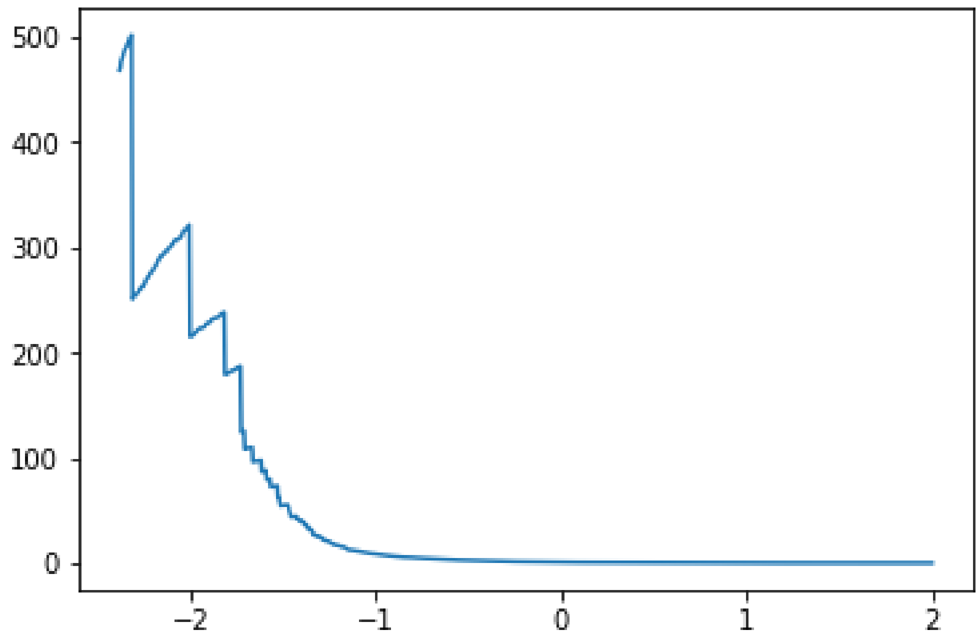
e)

```
In [33]: def precision(signal, noise, cut):
             true_pos  = np.array([len(signal[signal < cut]) for cut in cut])
             false_pos = np.array([len(noise[noise < cut]) for cut in cut])
             return true_pos / (true_pos + false_pos)

In [34]: def recall(signal, noise, cut):
             true_pos  = np.array([len(signal[signal < cut]) for cut in cut])
             false_neg = np.array([len(signal[signal > cut]) for cut in cut])
             return true_pos / (true_pos + false_neg)

In [35]: signal = projection_0
         noise = projection_1

         lam_cut = np.linspace(-6, 5, 100)
         plt.plot(lam_cut, precision(signal, noise, lam_cut),
                  label = 'Reinheit')
         plt.plot(lam_cut, recall(signal, noise, lam_cut),
                  label = 'Effizienz')
         plt.legend()

/home/stefan/.local/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:4: RuntimeWarn
  after removing the cwd from sys.path.


Out[35]: <matplotlib.legend.Legend at 0x7f0a61d92518>
```



f) Untersuche Signal-Untergrundverhältnis:

```
In [36]: def signal_noise_ratio(signal, noise, cut):
             return np.array([len(signal[signal <= cut]) / len(noise[noise <= cut]) for cut in

In [37]: lam_cut = np.linspace(min(noise), 2, 1000)
         plt.plot(lam_cut, signal_noise_ratio(signal, noise, lam_cut),
                  label = 'Signal-Untergrundverhältnis')

         lam_cut[np.argmax(signal_noise_ratio(signal, noise, lam_cut))]

Out[37]: -2.31935304024229
```

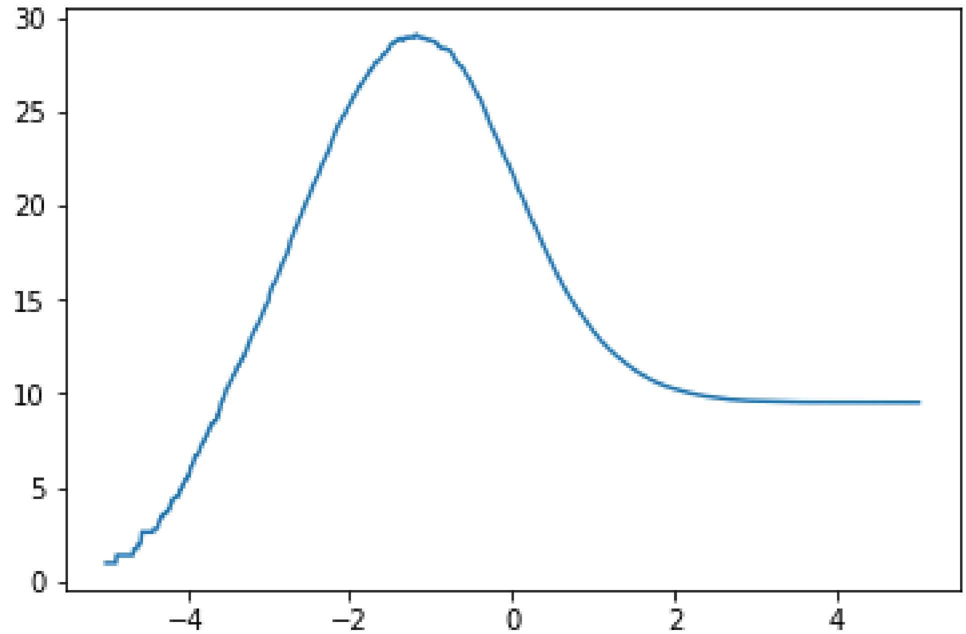


g) Untersuche Signifikanz:

```
In [38]: def sig(signal, noise, cut):
             return np.array([len(signal[signal <= cut]) /
                              np.sqrt(len(noise[noise <= cut]) + len(signal[signal <= cut])) fo
```

```
In [39]: lam_cut = np.linspace(-5, 5, 1000)
         plt.plot(lam_cut, sig(signal, noise, lam_cut),
                  label = 'Signifikanz')

         lam_cut[np.argmax(sig(signal, noise, lam_cut))]
```

```
Out[39]: -1.1761761761761762
```




```
In [ ]:
```