28	29	30	31	
5/6	8,5/9	5/5	nicht bearbeitet	18,5/20

Okäse

blatt_10_nitschke_grisard

January 10, 2019

0.1 Aufgabe 28

gut für Ubungsz

a)

0,5P.

$$A = \begin{pmatrix} 1-\varepsilon & \varepsilon \\ \varepsilon & 1-\varepsilon \end{pmatrix}$$ ✓ steht ja unten bei

b)

$$\vec{f} = A^{-1}\vec{g} = \frac{5}{4}\frac{1}{1-2\varepsilon}\begin{pmatrix} (1-\varepsilon)g_1 - \varepsilon g_2 \\ (1-\varepsilon)g_2 - \varepsilon g_1 \end{pmatrix}$$ ✓ ✓ 0,5P.

c) 0,5P.

$$V_f = \begin{pmatrix} (1-\varepsilon)^2\sqrt{g_1} + \varepsilon^2\sqrt{g_2} & -\varepsilon(1-\varepsilon)\left(\sqrt{g_1}+\sqrt{g_2}\right) \\ -\varepsilon(1-\varepsilon)\left(\sqrt{g_1}+\sqrt{g_2}\right) & (1-\varepsilon)^2\sqrt{g_2} + \varepsilon^2\sqrt{g_1} \end{pmatrix}$$ das ist hier nicht richtig aufgeschrieben

d) aber unten berechnet ihr die Matrix richtig 0,5P.

$\left(\frac{5}{4}\right)^2 \frac{1}{(1-2\varepsilon)}$ (das oben nur ohne Wurzeln)

```
In [1]: import numpy as np
        from scipy import linalg
        from numpy import linalg as la
```

1P.

```
In [2]: def A(e):
            return 0.8 * np.matrix([[1-e,e],[e,1-e]])

        g = np.array([200,169])
```

```
In [3]: def V_f(e):
            a = A(e)
            V_g = np.diag(g)
            return np.linalg.inv(a) @ V_g @ np.linalg.inv(a).T

        def f(g, e):
            return np.linalg.inv(A(e))@g

        def Korr(matrix_A):
            return matrix_A[0,1]/np.sqrt(matrix_A[0,0]*matrix_A[1,1])
```

```
In [4]: print(f'Fall 1, e = 0.1:', '\n', f' f =  {f(g,0.1)}', '\n', f' V_f = {V_f(0.1)}')
        print(f'Korrelationskoeffizient: {Korr(V_f(0.1))}')

Fall 1, e = 0.1:
  f =  [[254.84375 206.40625]]
  V_f = [[399.63378906 -81.07910156]
 [-81.07910156 339.08691406]]
Korrelationskoeffizient: -0.22025324331796559
```

2P.

-0,5P. Fehler von f_1 und f_2 fehlen

e)

```
In [5]: print(f'Fall 1, e = 0.4:', '\n', f' f =  {f(g, 0.4)}', '\n', f' V_f = {V_f(0.4)}')
        print(f'Korrelationskoeffizienten: {Korr(V_f(0.4))}')

Fall 1, e = 0.4:
  f =  [[327.5  133.75]]
  V_f = [[ 3868.75   -3459.375 ]
 [-3459.375   3626.5625]]
Korrelationskoeffizienten: -0.9235591729158679
```

0,5P.

f)

Für $\varepsilon = 0.5$ ist die Matrix A nicht invertierbar. Somit besitzt das Problem in diesem Fall keine analytische Lösung.

0,5P.

+ W'keit Ergebnis richtig oder falsch zu klassifizieren wird jeweils 50%.

0.2 Aufgabe 29

a)

```
In [6]: import matplotlib.pyplot as plt

In [7]: def A_matrix(n,e):
            A = np.zeros([n,n])
            for i in range(n-1):
                A[i, i+1] = e
                A[i+1, i] = e
            A[0,0] = A[n-1,n-1] = 1-e
            for i in range(n-2):
                A[i+1,i+1] = 1-2*e
            return A
```

1P.

A beschreibt einen Messprozess, bei dem mit einer Wahrscheinlichkeit von ε Ereignisse einem der Nachbarbins falsch zugeordnet werden. Der erste und letzte Bin der Diagonalen besitzen nur einen Nachbarn, weshalb hier nur $1 - \varepsilon$ steht.

b)

1P.

```
In [8]: f = [193, 485, 664, 763, 804, 805, 779, 736, 684, 626,
        566, 508, 452, 400, 351, 308, 268, 233, 202, 173]
        A = A_matrix(20,0.23)
        g = A@f
        np.random.seed(42)
        g_mess = np.array([np.random.poisson(lam=n) for n in g])

        print(g_mess)
```

```
[254 474 616 758 826 770 759 729 691 610 563 487 459 407 341 318 247 223
 194 181]
```

c)

$$g = Af = UDU^{-1}fU^{-1}g = DU^{-1}fc = Db \text{ (Transformierte Gleichung)}$$

Die neu eingeführten Vektoren c und b hängen über eine diagonale Matrix D miteinander zusammen. Dies bieten den Vorteil, dass die Einträge unabhängig transformiert werden (Multiplikation mit Eigenwerten von A). Berechne nun die Eigenwerte/Eigenvektoren von A und sortiere nach GröSSe absteigend (nutze hierzu die Funktion np.linalg.eigh(), welche die Eigenwerte in aufsteigender Reihenfolge ermittelt):

```
In [9]: Eigenvalues = la.eigh(A)[0][::-1]
        Eigenvectors = la.eigh(A)[1]
        U = Eigenvectors[:, ::-1]
        D = np.diag(Eigenvalues)
```

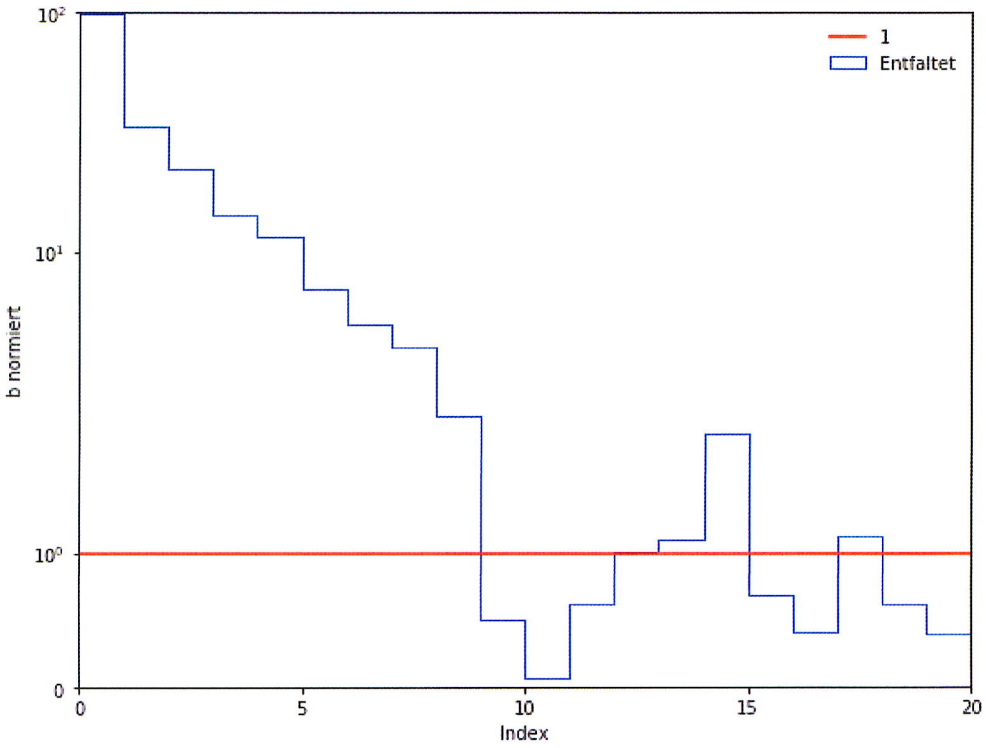
```
In [10]: c = la.inv(U)@g_mess
         V_g = np.diag(g_mess)
         V_c = la.inv(U)@V_g@la.inv(U).T
```

```
In [11]: b = la.inv(D)@c
         B = la.inv(D)
         V_b = B @ V_c @ B.T
         sigma_b = np.sqrt(np.diag(V_b))
         #b_j = np.array([b[i]/n for i,n in enumerate(sigma_b)])
         b_j = np.abs(b)/sigma_b
```

```
In [12]: import matplotlib.pyplot as plt
         plt.figure(figsize = (9, 7))
         bin_edges = np.linspace(0, 20, 21)

         plt.clf()
         plt.fill_between(bin_edges,np.concatenate(([0],b_j)),
                          step="pre",
                         linestyle = '-', facecolor = '',
                         edgecolor = 'b', label = 'Entfaltet')
```

```
plt.axhline(y = 1, color = 'r', label = '1')
plt.legend()
plt.xlabel('Index')
plt.ylabel('b normiert')
plt.xticks([0, 5, 10, 15, 20])
plt.ylim(0, 100)
plt.xlim(0, 20)
plt.yscale('symlog')
plt.show()
```



Hier hättet noch die wahre Verteilung mit zeigen können

ansonsten sieht es gut aus

0,5P.

Bei allen Werten die unterhalb der roten Linie liegen, für die also $b_j < 1$ gilt, liegen die Werte auSSerhalb der 1σ-Umgebung. Diese Koeffizienten sind mit 0 verträglich und enthalten keine Information.

d)

```
In [13]: f_unreg = U@b
         V_f = U@V_b@U.T

         # Regularisierung
         b_reg = np.copy(b)
```

```
b_reg[9:] = 0
V_b_reg = np.copy(V_b)

for i in range(9,20):
    V_b_reg[i,i] = 0

f_reg = U@b_reg
V_f_reg = U@V_b_reg@U.T
```

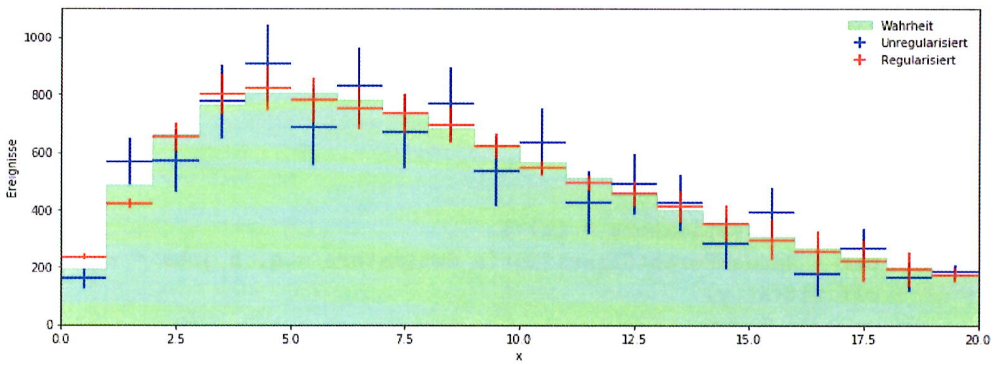
1 P.

In [14]:
```
plt.figure(figsize = (14, 5))
plt.fill_between(bin_edges,np.concatenate(([0],f)), step="pre", alpha = 0.3,
                 color = 'g', label = 'Wahrheit')

plt.errorbar(x = (bin_edges[1:] + bin_edges[:-1])*0.5, y = f_unreg,
             xerr = np.diff(bin_edges)*0.5, yerr = np.sqrt(np.diag(V_f)),
             linestyle = '', label = 'Unregularisiert', color = 'b')

plt.errorbar(x = (bin_edges[1:] + bin_edges[:-1])*0.5, y = f_reg,
             xerr = np.diff(bin_edges)*0.5, yerr = np.sqrt(np.diag(np.abs(V_f_reg))),
             linestyle = '', label = 'Regularisiert', color = 'r')

plt.ylabel('Ereignisse')
plt.xlabel('x')
plt.legend()
plt.ylim(0,1100)
plt.xlim(0,20)
plt.show()
```



sieht gut aus

1P.

Aufgrund der Regularisierung können die im unregularisierten Fall auftretenden Oszillationen vermindert werden. Zudem sind die Varianzen geringer.

0.3 Aufgabe 30: Data Mining Anwendung

a)

```
In [15]: from pandas import DataFrame
         import pandas as pd
         from sklearn.model_selection import train_test_split
         from sklearn.ensemble import RandomForestClassifier
         from sklearn.model_selection import cross_val_score
         from sklearn.metrics import roc_auc_score
         from sklearn.metrics import make_scorer
         from sklearn.metrics import roc_curve
```

```
/home/stefan/.local/anaconda3/lib/python3.7/site-packages/sklearn/ensemble/weight_boosting.py:
  from numpy.core.umath_tests import inner1d
```

```
In [16]: data = pd.read_hdf('image_parameters_smd_reduced.hdf5')
```

Erzeuge Vektor y mit den Labels für Hadronen- und Gamma-Ereignissen:

```
In [17]: y = np.zeros(len(data.corsika_run_header_particle_id))
         y[data.corsika_run_header_particle_id == 1] = 1
```

Für die Analyse dürfen nur Parameter verwendet werden, die auch einer Messung zugänglich wären. Werfe daher alle Lables und insbesondere auch die wahre Gesamtenergie weg.

```
In [18]: X = data.drop(columns=['run_id', 'event_num',
                                'corsika_event_header_total_energy',
                                'corsika_run_header_particle_id', ])
```

Erzeuge einen Trainings- und einen Test-Datensatz:

```
In [19]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,
                                                             train_size = 0.8,
                                                             random_state = 42)
```

b)

```
In [20]: for n in [1, 10, 100]:
             print(f'n_estimators = {n}')
             clf = RandomForestClassifier(n_estimators = n, n_jobs = -1)
             clf.fit(X, y)

             roc_score = cross_val_score(clf, X, y,
                                         scoring=make_scorer(roc_auc_score),
                                         cv = 5, n_jobs = -1)
             print(f'Roc Auc Score: {roc_score.mean():.4f} +/- {roc_score.std():.4f}\n')
```

```
n_estimators = 1
Roc Auc Score: 0.5932 +/- 0.0036

n_estimators = 10
Roc Auc Score: 0.6316 +/- 0.0011
```

```
n_estimators = 100
Roc Auc Score: 0.6561 +/- 0.0037
```

✓ 1P.

Die besten Werte werden mit 100 Bäumen erreicht.

c)

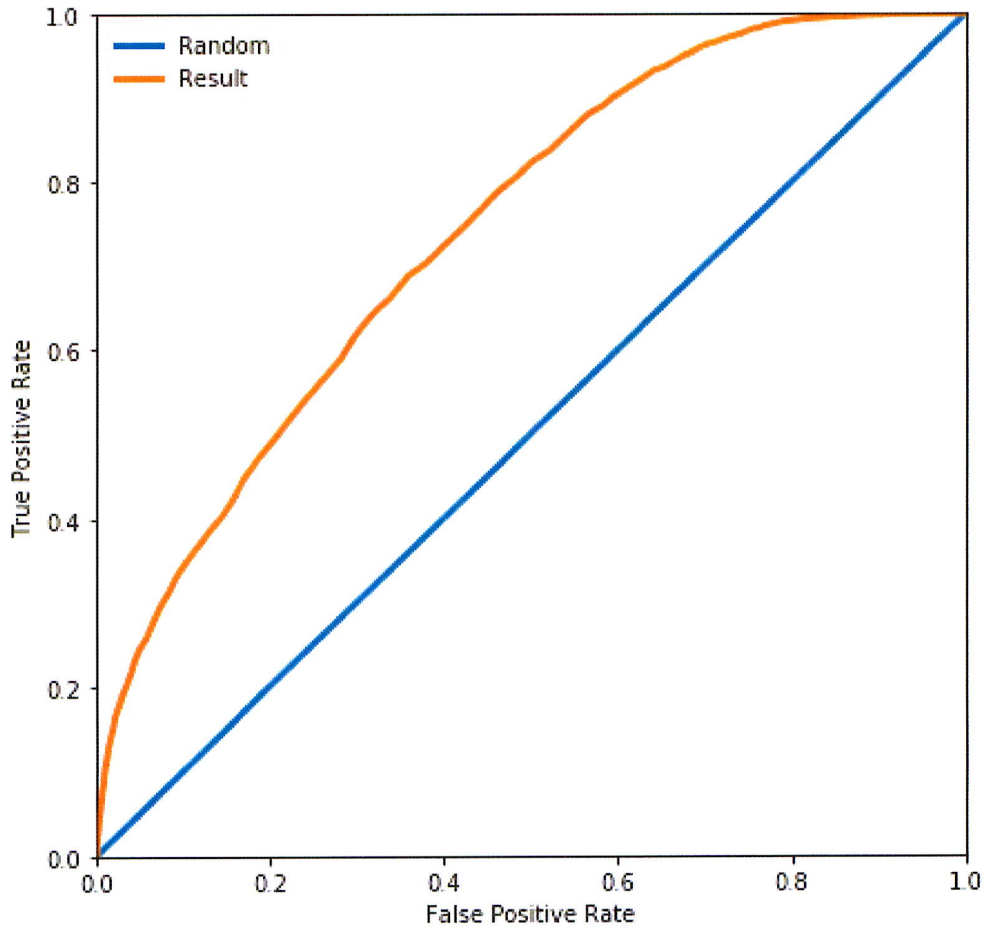
```
In [21]: clf = RandomForestClassifier(n_estimators = 100, n_jobs = -1)
         clf.fit(X_train, y_train)
         prediction = clf.predict_proba(X_test)[:, 1]
```

✓ 0,5 P.

d) Evaluiere den Klassifizierer mit der ROC-Kurve:

```
In [22]: plt.figure(figsize = (7, 7))
         plt.xlim(0, 1)
         plt.ylim(0, 1)
         plt.xlabel('False Positive Rate')
         plt.ylabel('True Positive Rate')
         fpr, tpr, t = roc_curve(y_test, prediction)
         plt.plot([0, 1], label = 'Random', linewidth = 3)
         plt.plot(fpr, tpr, label = 'Result', linewidth = 3)
         plt.legend()

Out[22]: <matplotlib.legend.Legend at 0x7f44ada6fc50>
```



Berechne die Fläche unter der Kurve:

```
In [23]: score = roc_auc_score(y_test, prediction)
         print(f'Fläche: {score}.')
```

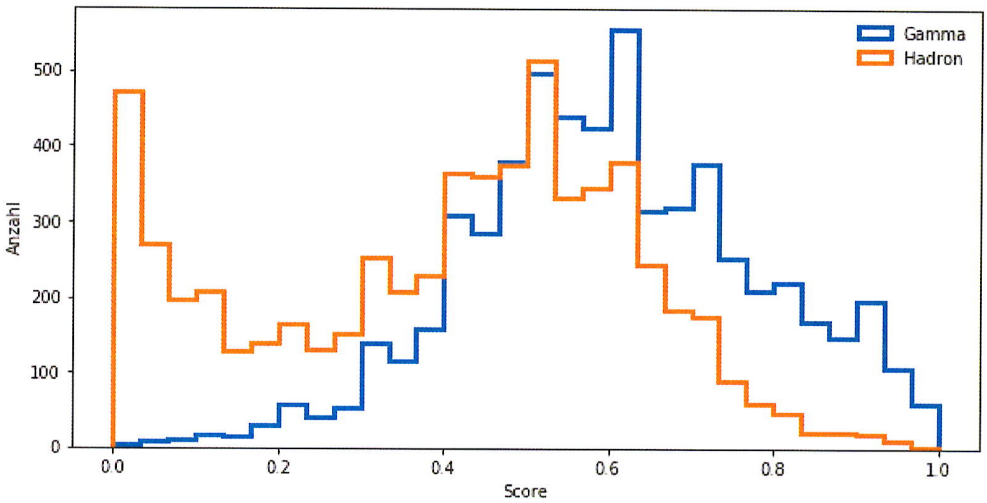
```
Fläche: 0.7416305956378468.
```

Für einen idealen Classifier wäre die Fläche 1.
Trenne die 'Prediction Propabilities' nach den wahren Gamma und Hadron Ereignissen:

```
In [24]: y_predict_gamma = prediction[y_test == 1]
         y_predict_hadron = prediction[y_test == 0]
```

```
In [25]: plt.figure(figsize = (10, 5))
         plt.hist(y_predict_gamma, histtype = 'step',
```

```
            bins = 30, label = 'Gamma',
            range = (0, 1), linewidth = 3)
plt.hist(y_predict_hadron, histtype = 'step',
            bins = 30, label = 'Hadron',
            range = (0, 1), linewidth = 3)
plt.xlabel('Score')
plt.ylabel('Anzahl')
plt.legend()
plt.show()
```



Kommentar: Ein idealer Classifier würde alle Hadron-Ereignisse bei 0 einordnen und alle Gamma-Ereignisse bei 1. Der plot zeigt, dass man die beiden Populationen nicht gut voneinader trennen kann, da der Überlapp der Verteilungen recht groSS ist.

2,5P.