

# Studienarbeit

## Maximilian Gaul

In [1]:

```
import math
import pandas as pd
import numpy as np
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score
import matplotlib.pyplot as plt
%matplotlib inline
```

## Aufgabe 1

Laden Sie die Daten aus `adult.data` in einen Pandas DataFrame

In [2]:

```
hdr = ["age", "workclass", "fnlwgt", "education", "education_num",
       "marital_status", "occupation", "relationship",
       "race", "sex", "capital_gain", "capital_loss",
       "hours_per_week", "native_country", "income"]
df = pd.read_csv("adult.data", skipinitialspace=True, names=hdr)
```

In [3]:

```
df
```

Out[3]:

	age	workclass	fnlwgt	education	education_num	marital_status	occupation	relations
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-fa
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husb
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-fa
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husb
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	\
...	...	...	...	...	...	...	...	...
32556	27	Private	257302	Assoc-acdm	12	Married-civ-spouse	Tech-support	\
32557	40	Private	154374	HS-grad	9	Married-civ-spouse	Machine-op-inspct	Husb
32558	58	Private	151910	HS-grad	9	Widowed	Adm-clerical	Unmar
32559	22	Private	201490	HS-grad	9	Never-married	Adm-clerical	Own-c
32560	52	Self-emp-inc	287927	HS-grad	9	Married-civ-spouse	Exec-managerial	\

32561 rows × 15 columns



## Aufgabe 2

1) In den nominalen Daten sind noch unbekannte Werte, gekennzeichnet durch `?`, vorhanden. Bereinigen Sie die Daten, indem Sie alle Zeilen entfernen, die unbekannte Werte enthalten.

Das Query selektiert alle Zeilen die keinen Spalteneintrag mit `?` haben.

In [4]:

```
def query_builder(hdr, val):
    query = ''
    for h in hdr:
        query += '{0} != "{1}" &'.format(str(h), str(val))
    return query[:-1]

df = df.query(query_builder(hdr, "?"), inplace=False)
```

## 2) Entfernen Sie die Spalten `fnlwgt` und `income` als Features

In [5]:

```
X = df.copy()
X = X.drop(columns=["fnlwgt", "income"])
X
```

Out[5]:

	age	workclass	education	education_num	marital_status	occupation	relationship	race
0	39	State-gov	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White
1	50	Self-emp-not-inc	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White
2	38	Private	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White
3	53	Private	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black
4	28	Private	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black
...	...	...	...	...	...	...	...	...
32556	27	Private	Assoc-acdm	12	Married-civ-spouse	Tech-support	Wife	White
32557	40	Private	HS-grad	9	Married-civ-spouse	Machine-op-inspct	Husband	White
32558	58	Private	HS-grad	9	Widowed	Adm-clerical	Unmarried	White
32559	22	Private	HS-grad	9	Never-married	Adm-clerical	Own-child	White
32560	52	Self-emp-inc	HS-grad	9	Married-civ-spouse	Exec-managerial	Wife	White

30162 rows × 13 columns

3) Als Target soll das Feature `income` dienen, jedoch kommt nicht jeder Algorithmus mit nominalen Features klar. Konvertieren Sie das Target daher, sodass `income` den Wert 1 annimmt, falls das `income` ursprünglich den Wert `>50K` hat und 0 andernfalls

In [6]:

```
y = pd.DataFrame(  
    {"income": np.where(df['income'] == ">50K", 1, 0)}  
)  
y
```

Out[6]:

	income
0	0
1	0
2	0
3	0
4	0
...	...
30157	0
30158	1
30159	0
30160	0
30161	1

30162 rows × 1 columns

**4) Wieviel Prozent der Personen haben ein Einkommen von mehr als 50.000\$ ?**

In [7]:

```
total = len(y.index)  
rich = len(y.query('income == 1').index)  
print((rich / total) * 100, "% haben ein Einkommen von mehr als 50.000$")
```

24.892248524633644 % haben ein Einkommen von mehr als 50.000\$

**5) Was ist die Genauigkeit eines naiven Modells, welches unabhängig von den tatsächlichen Features immer weniger als 50.000\$ Einkommen zuweist? Dies ist das Mindestmaß an Genauigkeit, an dem sich ihre späteren Modelle messen müssen**

In [8]:

```
tn = total - rich  
print("total:", total)  
print("tn:", tn)
```

total: 30162

tn: 22654

Die Genauigkeit ist definiert als  $\frac{tp+tn}{tp+tn+fp+fn}$

$tp$  ist die Anzahl der Personen, die das Modell als  $>50k$  einstuft und die auch tatsächlich  $>50k$  verdienen. Im Fall des naiven Modells ist  $tp = 0$ .

$tn$  ist die Anzahl der Personen, die das Modell als  $\leq 50k$  einstuft und die auch tatsächlich  $\leq 50k$  verdienen. Im Fall des naiven Modells ist  $tn = 22654$ .

Der Wert des Nenners  $tp + tn + fp + fn$  entspricht der Gesamtzahl an Datensätzen. Damit lässt sich die Genauigkeit des naiven Modells berechnen:

$$acc = \frac{0+22654}{30162} = 75.11\%$$

## Aufgabe 3

Schreiben Sie eine Methode `transform(X)` welche einen Feature-Dataframe `X` als Parameter erhält und einen transformierten `DataFrame` zurückgibt.

Die im Wertebereich verzerrten Features `capital_gain` und `capital_loss` sollten durch Logarithmierung normalisiert werden. Verwenden Sie dafür die Funktion

$$f : \mathbb{R} \rightarrow \mathbb{R}, f(x) = \log(x + 1)$$

In [9]:

```
f = lambda x: math.log(x + 1)
```

In [10]:

```
def transform(X):
    _X = X.copy()
    _X["capital_gain"] = _X["capital_gain"].apply(f)
    _X["capital_loss"] = _X["capital_loss"].apply(f)

    numeric_headers = [
        "age", "education_num", "capital_gain",
        "capital_loss", "hours_per_week"
    ]

    nominal_headers = [
        "workclass", "education", "marital_status",
        "occupation", "relationship",
        "race", "sex", "native_country"
    ]

    for nmh in numeric_headers:
        min_max_scaler = preprocessing.MinMaxScaler()
        _X[nmh] = min_max_scaler.fit_transform(_X[nmh].values.reshape(-1, 1))

    _X = pd.get_dummies(_X, columns=nominal_headers)

    return _X
```

In [11]:

```
X_trans = transform(X)
X_trans
```

Out[11]:

	age	education_num	capital_gain	capital_loss	hours_per_week	workclass_Federal-gov
0	0.301370	0.800000	0.667492	0.0	0.397959	0
1	0.452055	0.800000	0.000000	0.0	0.122449	0
2	0.287671	0.533333	0.000000	0.0	0.397959	0
3	0.493151	0.400000	0.000000	0.0	0.397959	0
4	0.150685	0.800000	0.000000	0.0	0.397959	0
...	...	...	...	...	...	...
32556	0.136986	0.733333	0.000000	0.0	0.377551	0
32557	0.315068	0.533333	0.000000	0.0	0.397959	0
32558	0.561644	0.533333	0.000000	0.0	0.397959	0
32559	0.068493	0.533333	0.000000	0.0	0.193878	0
32560	0.479452	0.533333	0.835363	0.0	0.397959	0

30162 rows × 103 columns

**Splitten Sie den Datensatz in einen Trainings- und Testdatensatz, wobei der Testdatensatz eine relative Größe von 20% haben soll. Verwenden Sie für die Reproduzierbarkeit einen `random_state = 0`**

In [12]:

```
X_train, X_test, y_train, y_test = train_test_split(X_trans, y, test_size=0.2, random_state
```

## Aufgabe 4

**Wählen Sie drei verschiedene, in der Vorlesung behandelte Modelltypen und trainieren Sie die entsprechenden Modelle mit den Standardparametern (keine Parameter). Geben Sie zu jedem Modell die Genauigkeit auf dem Testdatensatz aus**

Da es sich um ein (binäres) Klassifizierungsproblem handelt werden nur Modelltypen verwendet, die zur Klassifikation von Daten geeignet sind.

Gleichzeitig ist nicht bekannt, ob die Daten linear separierbar sind, d.h. einfache Neuronale Netze wie Perzeptron oder Adaline sind ungeeignet.

Daher werden folgende Modelle trainiert:

In [13]:

```
def model_to_accuracy(model, X_test, y_test):
    model_predict = model.predict(X_test)
    cf = confusion_matrix(
        y_true=y_test, y_pred=model_predict, labels=[1, 0]
    )

    TP = cf[0][0]
    FN = cf[0][1]
    FP = cf[1][0]
    TN = cf[1][1]

    return (TP + TN) / (TP + TN + FP + FN)
```

## Logistic Regression

Logistische Regression ist hier ein potentiell geeignetes Modell, da es sich nur um ein binäres Klassifikationsproblem handelt.

Für das Modell wird `max_iter=1000` gesetzt, da es ansonsten zu einer Fehlermeldung kommt (Modell konvergiert nicht).

Die mehrdimensionale Logistische Regression kann durch einen Basiswechsel auch mit Daten umgehen, die nicht linear trennbar sind.

In [14]:

```
lr = LogisticRegression(max_iter=1000).fit(X_train, y_train.values.ravel())
lr_accuracy = model_to_accuracy(lr, X_test, y_test)
print("Genauigkeit LogisticRegression:", lr_accuracy)
```

Genauigkeit LogisticRegression: 0.8395491463616774

## K-Nearest-Neighbors

K-Nearest-Neighbors ist in der Lage, auch Mehrklassen-Klassifikationsprobleme zu lösen, kommt aber genauso gut mit binärer Klassifikation zurecht.

In [15]:

```
knn = KNeighborsClassifier().fit(X_train, y_train.values.ravel())
knn_accuracy = model_to_accuracy(knn, X_test, y_test)
print("Genauigkeit KNN:", knn_accuracy)
```

Genauigkeit KNN: 0.8168407094314603

## Support Vector Klassifikation

Aufgrund seiner Abweichung von perfekt separierbaren Daten indem Trainingsdatenpunkte auch innerhalb des Margins bzw. auf der falschen Seite liegen dürfen und das Modell robuster machen, eignet sich die Support Vector Machine potentiell für dieses Problem.

In [16]:

```
svc = SVC().fit(X_train, y_train.values.ravel())
svc_accuracy = model_to_accuracy(svc, X_test, y_test)
print("Genauigkeit SVC:", svc_accuracy)
```

Genauigkeit SVC: 0.8392176363334991

## Aufgabe 5

**Wählen Sie das im vorherigen Schritt beste Modell und tunen Sie die Hyperparameter um möglichst eine noch bessere Performance zu bekommen**

Das Modell mit der besten Genauigkeit ist das `LogisticRegression` -Modell.

Dieses Modell wird nun anhand der Hyperparameter `C`, `tol`, `solver` und `penalty` mit einer Rastersuche über viele Wertekombinationen getuned.

Bestimmte penalties kommen nur mit bestimmten solvern zurecht (und anders herum), darauf wird entsprechend Rücksicht genommen (siehe sklearn-Dokumentation).

*C*: Inverser Bestrafungsfaktor für ausufernde Parameterwerte

*tol*: Toleranz des Stoppkriteriums

*solver*: Der für das Optimierungsproblem verwendete Algorithmus

*penalty*: Art der Bestrafung für große Parameterwerte (z.B. Ridge-Regression)



In [17]:

```
C = [0.01, 0.05, 0.8, 2.0, 5.0, 10.0]
tol = [1e-5, 1e-4, 1e-3, 1e-2]
solver = ["newton-cg", "lbfgs", "liblinear", "sag", "saga"]
penalties = ["l1", "l2", "elasticnet"]

combinations = len(C) * len(tol) * len(solver) * len(penalties)
print("Trying", combinations, "combinations...")

i = np.arange(
    0, combinations, 1
)
combination_dict = []

vals = []
for c in C:
    for t in tol:
        for s in solver:
            for p in penalties:
                l1_ratio = None
                if p == "elasticnet":
                    s = "saga"
                    l1_ratio = 0.5
                elif s in ["newton-cg", "lbfgs", "sag"]:
                    p = "l2"
                model = LogisticRegression(
                    max_iter=1500, C=c, tol=t, penalty=p, solver=s, l1_ratio=l1_ratio
                )
                model.fit(X_train, y_train.values.ravel())
                acc = model_to_accuracy(model, X_test, y_test)
                vals.append(acc)
                combination_dict.append({
                    "C": c,
                    "tol": t,
                    "solver": s,
                    "penalty": p,
                    "l1_ratio": l1_ratio
                })
            if(len(vals) % 10 == 0):
                print("We are at", len(vals))
```

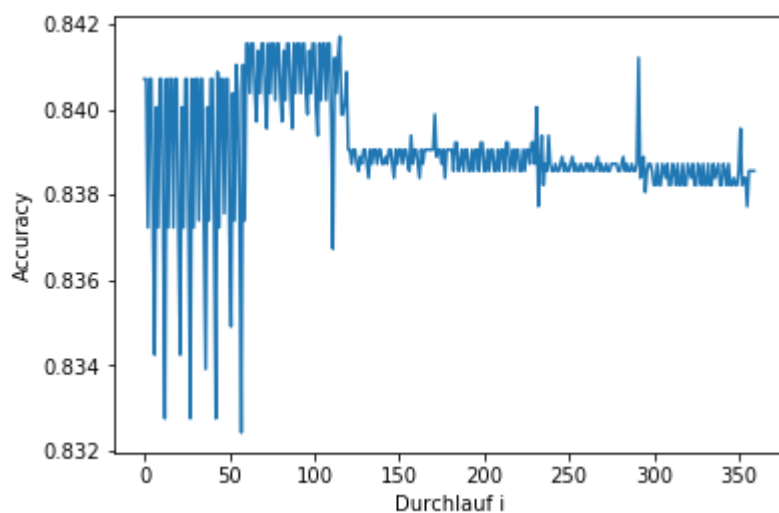
Trying 360 combinations...

We are at 10  
We are at 20  
We are at 30  
We are at 40  
We are at 50  
We are at 60  
We are at 70  
We are at 80  
We are at 90  
We are at 100  
We are at 110  
We are at 120  
We are at 130  
We are at 140  
We are at 150  
We are at 160  
We are at 170

We are at 180  
We are at 190  
We are at 200  
We are at 210  
We are at 220  
We are at 230  
We are at 240  
We are at 250  
We are at 260  
We are at 270  
We are at 280  
We are at 290  
We are at 300  
We are at 310  
We are at 320  
We are at 330  
We are at 340  
We are at 350  
We are at 360

In [18]:

```
plt.plot(i, vals)
plt.xlabel("Durchlauf i")
plt.ylabel("Accuracy")
plt.show()
```



Liniendiagramm in dem die Genauigkeit für jeden der 360 Durchläufe dargestellt ist.

In [19]:

```
best_comb = combination_dict[np.argmax(vals)]
print("Maximale Genauigkeit", np.array(vals).max()*100, "% für ", best_comb)
```

Maximale Genauigkeit 84.17039615448367 % für {'C': 0.05, 'tol': 0.01, 'solver': 'sag', 'penalty': 'l2', 'l1\_ratio': None}

Nach 360 Optimierungsversuchen konnte die Genauigkeit von 83.95 % auf 84.17 % des Logistischen Regressionsmodells gesteigert werden.

In [20]:

```
best_model = LogisticRegression(  
    C=best_comb["C"],  
    tol=best_comb["tol"],  
    solver=best_comb["solver"],  
    penalty=best_comb["penalty"]  
)  
best_model.fit(X_train, y_train.values.ravel())
```

Out[20]:

```
LogisticRegression(C=0.05, class_weight=None, dual=False, fit_intercept=True,  
    intercept_scaling=1, l1_ratio=None, max_iter=100,  
    multi_class='auto', n_jobs=None, penalty='l2',  
    random_state=None, solver='sag', tol=0.01, verbose=0,  
    warm_start=False)
```

## Aufgabe 6

Erstellen Sie einen DataFrame mit Werten für eine erfundene Person

In [21]:

```
person = pd.DataFrame({  
    "age": 23,  
    "workclass": "Self-emp-inc",  
    "education": "Bachelors",  
    "education_num": 12,  
    "marital_status": "Never-married",  
    "occupation": "Tech-support",  
    "relationship": "Unmarried",  
    "race": "White",  
    "sex": "Male",  
    "capital_gain": 5,  
    "capital_loss": 0,  
    "hours_per_week": 20,  
    "native_country": "Germany"  
}, index=[0])  
person
```

Out[21]:

	age	workclass	education	education_num	marital_status	occupation	relationship	race
0	23	Self-emp-inc	Bachelors	12	Never-married	Tech-support	Unmarried	White

Transformieren Sie diese Person ebenfalls mit Hilfe der `transform`-Methode. Da die Normierung nur auf größeren Datensätzen Sinn macht, vereinen Sie den ursprünglichen DataFrame und die neue Person und transformieren das Gesamtpaket

In [22]:

```
Xp = pd.concat((X, person))
Xp_trans = transform(Xp)
```

In [23]:

```
tail = Xp_trans.tail(1)
tail
```

Out[23]:

	age	education_num	capital_gain	capital_loss	hours_per_week	workclass_Federal- gov	wc
0	0.082192	0.733333	0.15563	0.0	0.193878		0

1 rows × 103 columns

Die erfundene Person ganz am Ende des zusammengeführten Datensatzes.

**Machen Sie mit Hilfe des Modells eine Vorhersage. Würde die Person als potentieller Spender ausgewählt werden?**

In [24]:

```
p = best_model.predict(Xp_trans)[-1]
```

In [25]:

```
p
```

Out[25]:

```
0
```

Die Person würde nicht als potentieller Spender ausgewählt werden da die Vorhersage die Klasse 0 zurückliefert ( <=50k ).