Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

2024 FS CAS PML - Supervised Learning
4 Validierung (und mehr)
4.1 Sampling und Resampling

Werner Dähler 2024

4 Validierung und mehr

- zum Lernen gehört Testen, als ...
  - Motivation für die Lernenden
  - Selbstkontrolle der Lernenden
  - Rückmeldung an den Trainer, ob er seinen Job ordentlich gemach hat

- im Rahmen von ML etwas problematisch, da
  - Testen im Prinzip erst möglich, wenn neue Daten vorliegen, für welche Voraussagen gemacht werden, welche erst später mit den effektiven Ergebnissen verglichen werden können
  - Kunstgriff: Validierung

- im bisherigen Verlauf des Kurses (Klassifikation, Regression) wurden einige Aspekte von Validierung bereits vorneweg genommen
  - Train - Test - Split (für Holdout Validierung)
  - Performance Metriken (accuracy bei Klassifikation, r2 bei Regression

4 Validierung und mehr

- in diesem Kapitel sollen daher die folgenden Aspekte noch etwas vertieft resp. ergänzt werden

AGENDA

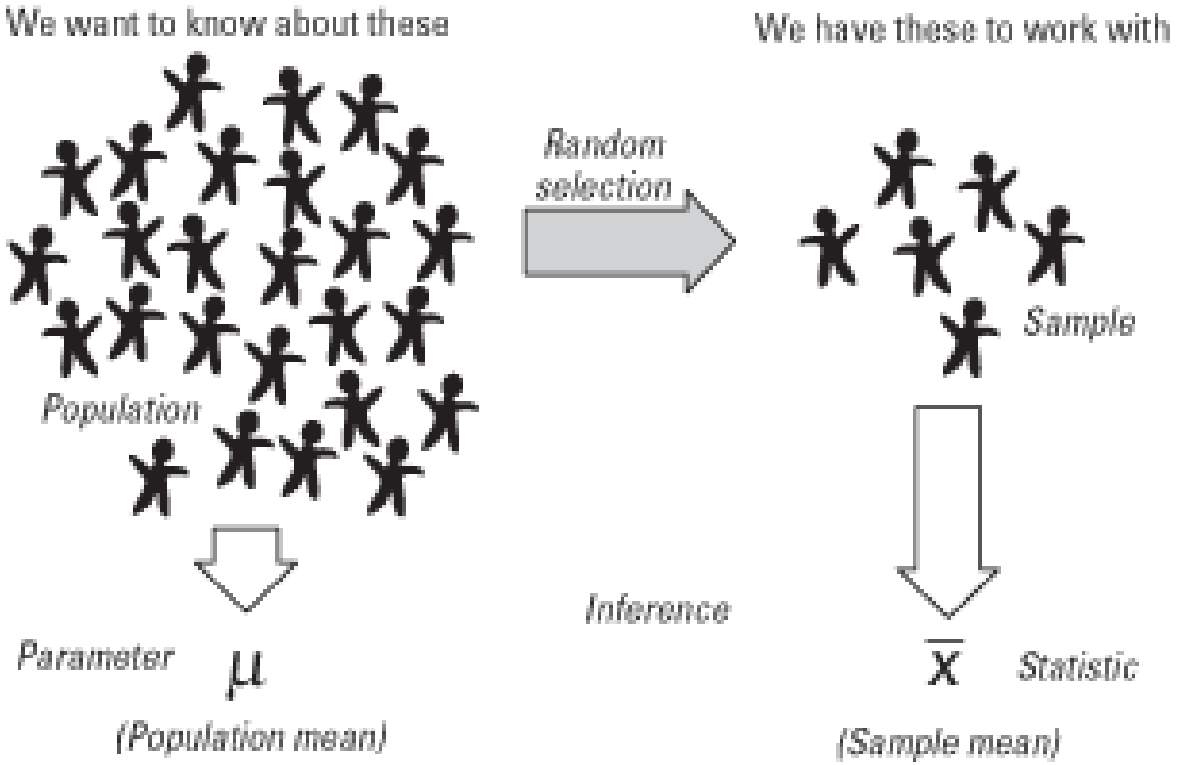
41. Sampling und Resampling
42. Validierungstechniken
43. Grid Search und Random Search
44. Performance Metriken
45. Umgang mit unbalancierten Daten

- einige Aspekte in diesem Kapitel haben weniger mit Validierung aber umso mehr mit Sampling zu tun, daher der Postfix dieses Kapitels "... und mehr"

- die hinterlegten Links wurden am 27.04.2024 abgegriffen

4.1 Validierung und mehr - Sampling und Resampling

- in Anlehnung an Statistische Datenanalyse wird jedes Dataset a priori als (Zufalls-) Stichprobe einer *Grundgesamtheit* oder *Population* aufgefasst
- daher auch die Bezeichnung *Zufallsvariable* für Merkmal in der statistischen Literatur
- aufgrund dieser Stichprobe sollen quantitative Aussagen über die Population gemacht werden können:
  - natürlich nur annäherungsweise
  - mit zusätzlicher Information über den Fehlerbereich (Vertrauens- oder Konfidenzintervall)
- Begründung für dieses Vorgehen: eine Vollerhebung, d.h. Berücksichtigung der gesamten Population ist
  - zu aufwendig / zu kostenintensiv
  - aus praktischen Gründen nicht möglich



Darstellung : CliffNotes

4.1 Validierung und mehr - Sampling und Resampling

- auch ausgehend von einem vorliegenden Dataset, das gemäss obiger Darstellung schon eine Stichprobe darstellt, kann es verschiedene Gründe geben, wiederum eine Zufallsstichprobe davon zu erstellen:
  - reduzieren der Daten aus Ressource-Gründen (Speicherplatz, Zeitbedarf)
  - testen von Vorhersagemodellen, somit die Hauptmotivation der hier besprochenen Aktivitäten
- wenn also im Folgenden von Population die Rede ist, dann ist jeweils die Ausgangsmenge für ein Random Sample gemeint (und nicht die dahinterliegende Grundgesamtheit)
- dabei stellt sich die Frage nach einem konventionellen Vorgehen zum Erstellen eines derartigen Sub-Samples
- z.B. mit Excel, "ziehen" von n Beobachtungen aus einer Tabelle, wobei jede Zeile eine Beobachtung (Instanz) enthält:
  - jeder Beobachtung wird eine Zufallszahl (z.B. uniform verteilt zwischen 0 und <1) zugewiesen
  - die Daten werden danach nach dieser Zufallszahl sortiert
  - die n ersten Beobachtungen werden als Zufallsstichprobe (random sample) betrachtet

4.1 Validierung und mehr - Sampling und Resampling

4.1.1 Zufallszahlen

- bei allen im Folgenden diskutierten Sampling Verfahren spielen Zufallszahlen eine wichtige Rolle
- entweder durch direkte Erzeugung derselben wie bei
  - Manuellem Sampling (4.1.2.1)
  - Sequenziellem Sampling (4.1.5.1)
  - Reservoir Sampling 4.1.5.2)
- oder aber intern
  - Sampling mit .sample() oder train_test_split() (4.1.3)
  - Geschichtetem Sampling (4.1.4)

4.1 Validierung und mehr - Sampling und Resampling

4.1.1 Zufallszahlen

- bei manuellem Sampling können die benötigten Zufallszahlen z.B. unter Verwendung der entsprechenden Funktion aus numpy wie folgt erzeugt werden

```
import numpy as np
n = 10
print(np.random.rand(n))
```

```
[0.37297087 0.18020772 0.00110873 0.25236517 0.55657484 0.01416197
 0.41637511 0.51408065 0.34707682 0.55884705]
```

- das Argument n enthält die Anzahl Zufallszahlen, welcher erzeugt werden sollen
- das Ergebnis ist ein Array (numpy.ndarray)
- die erzeugten Werte liegen zwischen 0 (inklusive) und 1 (exklusive) und sind uniform verteilt (mathematisch: rechtsoffenes Intervall)
- **der oben gezeigte Code führt jeweils zu einer anderen Sequenz von Zufallszahlen, das Ergebnis ist also nicht reproduzierbar!**

4.1 Validierung und mehr - Sampling und Resampling

4.1.1 Zufallszahlen

- für Test- und Ausbildungszwecke kann es angebracht sein, reproduzierbare Sequenzen von Zufallszahlen zu erzeugen
- dazu kann der Random Seed mit einem willkürlichen Wert festgelegt werden

```
np.random.seed(1234); print(np.random.rand(n))
np.random.seed(1234); print(np.random.rand(n))
```

```
[0.19151945 0.62210877 0.43772774 0.78535858 0.77997581 0.27259261
 0.27646426 0.80187218 0.95813935 0.87593263]
[0.19151945 0.62210877 0.43772774 0.78535858 0.77997581 0.27259261
 0.27646426 0.80187218 0.95813935 0.87593263]
```

- bei vielen der bisher gesehenen Funktionen oder Methoden, wo Zufallsprozesse eine Rolle spielen, kann dies auch mit dem Argument `random_state` erreicht werden

4.1 Validierung und mehr - Sampling und Resampling

4.1.1 Zufallszahlen

der Random Number Generator (RNG)

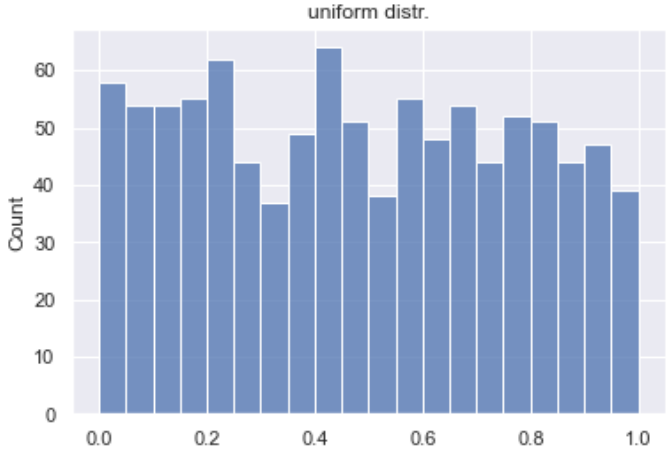
- eigentlich Pseudorandom Number Generator (PRNG)
- es existiert eine ganze Reihe derartiger Generatoren
- Standard bei Python (und auch z.B. bei R) ist Mersenne Twister (MT 19937) mit einer Periodizität von 2^{19937}-1
  - zum Vergleich: könnte man pro Sekunde 10^6 Zufallszahlen generieren, würde es 10^{5988} Jahre dauern, bis der Anfangszustand wieder erreicht wäre
  - das Universum ist gerade mal ca. 10^{11} Jahre alt
- Basis bei MT 19937 ist der Seed key (Startwert), ein Vektor mit 624 32-Bit Integer Werten
- ohne spezielle Vorkehrungen wird der Seed beim ersten Aufruf initialisiert, normalerweise durch die Systemzeit des OS
- MT 19937 gilt in der Literatur zwar als wenig zuverlässig, dies aber nur mit Sicht auf kryptographische Aufgaben

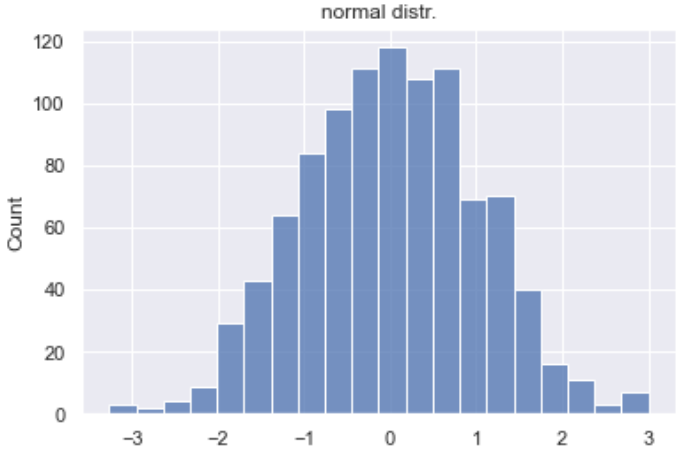
4.1 Validierung und mehr - Sampling und Resampling

4.1.1 Zufallszahlen

- Zufallszahlen können nach gewünschten Modellverteilungen generiert werden, zwei davon

uniform verteilt	**normal** verteilt
wird hauptsächlich verwendet, um Zufallsstichproben (random samples) einer bestimmten Grösse aus Grundgesamtheiten (Populationen) zu "ziehen"	wird hauptsächlich verwendet zum Simulieren von Werten einer Variable (Feature) um die Wirksamkeit von Methoden zu untersuchen - Datenanalyse (np.random.randn())
`np.random.rand()`	`np.random.randn()`





4.1 Validierung und mehr - Sampling und Resampling

4.1.1 Zufallszahlen

- Eigenschaften von normal verteilten Zufallszahlen
  - Mittelwert = 0
  - Standardabweichung = 1
- Eigenschaften von uniform verteilten Zufallszahlen
  - Wertebereich: zwischen 0 und 1
  - die Frequenzen für gleich grosse Intervalle sollen dabei (annähernd) gleich gross sein, abgesehen von "Rauschen"
- für nicht uniforme Verteilungen werden normalerweise uniform verteilte Zahlen erzeugt, welche dann entsprechen den Anforderungen in die Zielverteilung transformiert werden

- python verfügt auch über ein eigenes Modul: https://docs.python.org/3.8/library/random.html
- ist im Gebrauch aber etwas komplizierter als die oben aus numpy eingesetzten Aufrufe

4.1.2 Einfaches Sampling - 4.1.2.1 Implementierungsmöglichkeit mit Zufallszahlen

- Zufallszahlen können wie folgt eingesetzt werden, um aus einem Data Frame eine Zufallsstichprobe einer beliebigen Grösse zu "ziehen"
- die einzelnen Schritte (Code: vgl. [ipynb])
  1. festlegen, wie gross die Stichprobe sein soll: `n`
  2. ergänzen des Data Frame mit einer Hilfsspalte mit Zufallszahlen: z.B. `rnd_col`
  3. sortieren des Data Frame nach `rnd_col`, die Zeilen des Data Frame sind danach zufällig durchmischt (shuffled)
  4. entfernen der Spalte mit den Zufallszahlen
  5. die n ersten Zeilen aus dem durchmischten Data Frame auswählen
- das Verfahren lässt sich leicht in beliebigen Applikationen implementieren, welche sowohl tabellarische Datenstrukturen wie auch Zufallszahlen unterstützen (z.B. Excel)

4.1 Validierung und mehr - Sampling und Resampling

4.1.2 Einfaches Sampling - 4.1.2.2. Implementierungsmöglichkeit mit einem Index

- die Funktion `numpy.random.choice()` erzeugt eine Zufallsstichprobe aus einem gegebenen 1-D-Array
- die einzelnen Schritte
  1. festlegen der Grösse der zu erzeugenden Stichprobe

```
prop = 2 / 3
smpl_size = int(len(data) * prop)
```

`int()` ist hier notwendig, da die einzusetzende Funktion .choice() für die Grösse nur Integerwerte akzeptiert

  2. erstellen eines Range über alle Zeilenindices des Ausgangs-Data Frame

```
idx = range(len(data))
```

  3. erstellen einer Zufallsstichprobe der gewünschten Grösse aus dem Index-Range

```
idx_smpl = np.random.choice(idx, smpl_size, replace=False)
```

4.1.2 Einfaches Sampling - 4.1.2.2. Implementierungsmöglichkeit mit einem Index

`replace=False` ist hier notwendig, da sonst eine Stichprobe mit Zurücklegen erstellt würde, d.h. ein Element aus der Population kann mehrmals in der Stichprobe erscheinen

4. verwenden der Index-Stichprobe zum Filtern auf den Ausgangsdaten

```
data_smpl = data.iloc[idx_smpl, ]
```

- ein disjunktes Testset kann mit derselben Index-Stichprobe bedarfsweise wie folgt erstellt werden

```
data_ramain = data.iloc[data.index.difference(idx_smpl)]
```

der Aufruf verwendet zum Filtern alle Index-Werte welche **nicht** in `idx_smpl` enthalten sind

4.1 Validierung und mehr - Sampling und Resampling

4.1.2 Einfaches Sampling - 4.1.2.3 Mit der Methode `.sample()` von Pandas Data Frame

- pandas Data Frame stellt mit `.sample()` eine Methode zur Verfügung, welche das obige mit einem einzigen Aufruf erledigt

```
data_smpl = data.sample(frac=2/3, random_state=1234)
print(data_smpl.shape)
```

```
(6579, 21)
```

- wird der Parameter frac auf 1 eingestellt, dann resultiert der gesamte Data Frame, aber zufällig durchmischt

Sampling mit Zurücklegen

- bis jetzt: Sampling **ohne** Zurücklegen, eine einmal für die Stichprobe ausgewählte Beobachtung kann nicht ein weiteres Mal ausgewählt werden, vgl. Ziehung bei Lotterien
- für das Argument `replace` der Methode `.sample()` wurde der Defaultwert: `False` übernommen

4.1 Validierung und mehr - Sampling und Resampling

4.1.2 Einfaches Sampling - 4.1.2.3 Mit der Methode `.sample()` von Pandas Data Frame

- mittels `replace=True` wird das Sampling mit Zurücklegen durchgeführt

```
data_smpl = data.sample(
    frac=2/3,
    replace=True,
    random_state=1234
## check
print(data_smpl.index.value_counts())
```

```
5828    6
1896    6
8338    5
6761    5
7049    5
       ..
6838    1
9105    1
6318    1
8500    1
9629    1
Length: 4857, dtype: int64
```

- eine einzelne Beobachtung kann somit mehrmals in der Stichprobe vorkommen

- Anwendungsmöglichkeit: bei Bootstrapvalidierung wird eine Stichprobe gezogen, welche gleich gross ist wie die Population, aber mit Zurücklegen (vgl. Kap. 4.2.3)

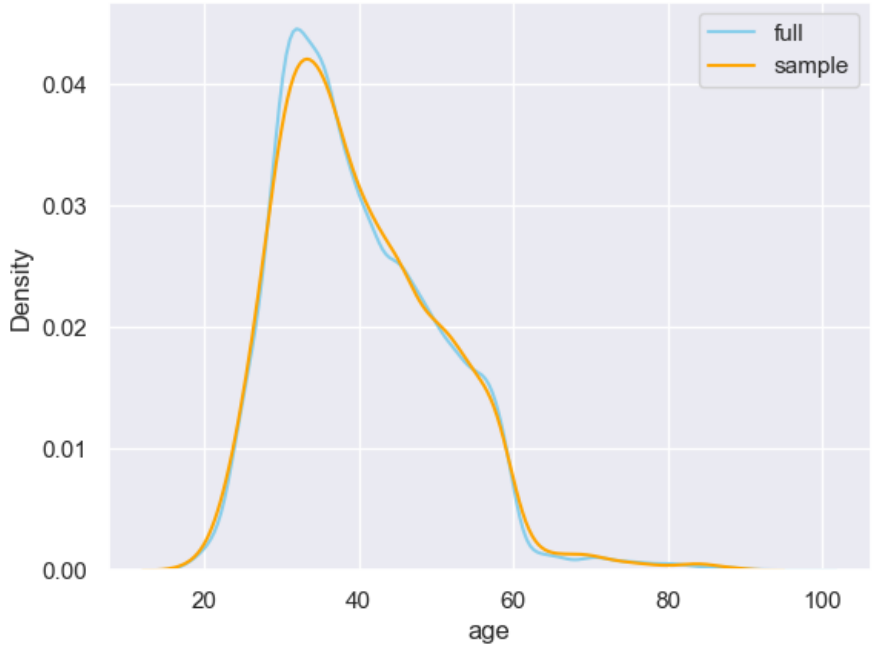
4.1 Validierung und mehr - Sampling und Resampling

4.1.2 Einfaches Sampling

- die oben erwähnte Eigenschaft, dass Zufallsstichproben die Charakteristiken der Ausgangspopulation widerspiegeln - wenn sie nur gross genug sind - soll hier kurz experimentell aufgezeigt werden

- nebenstehend die Dichteverteilung von age in bank-additional-full.csv (41188 Beobachtungen) und einem Random Sample mit 10% der Ausgangsdaten

- allerdings: bei Fragestellungen, bei welchen Extremwerte im Fokus stehen, ist Sampling dagegen keine sehr gute Idee



4.1 Validierung und mehr - Sampling und Resampling

4.1.3 Train - Test - Split

- in den Kapiteln zu Klassifikation und Regression (Kap. 2/3) wurde zur Erstellung von Trainings- und Testset die Funktion `train_test_split()` aus dem Modul `sklearn.model_selection` eingesetzt:

```
from sklearn.model_selection import train_test_split
train, test = train_test_split(data, train_size=2/3, random_state=1234)
```

- der obige Aufruf übernimmt ein Positionsargument (`data`) und und gibt zwei Ergebnisse zurück (`train` und `test`)
- es können aber auch mehrere Positionsargumente übergeben werden, Bedingung ist dann aber, dass es sich um gleich lange vektorartige Objekte handelt

```
X = data.drop('y', axis=1)
y = data['y']
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test, = train_test_split(
    X, y, train_size=2/3, random_state=1234)
```

4.1 Validierung und mehr - Sampling und Resampling

4.1.3 Train - Test - Split

- `X` und `y`, welche nach den vorbereitenden Aufrufen (erste zwei Zeilen) zwei gleich lange Objekte sind werden somit wie folgt gesplittet
  - `X` → `X_train` und `X_test`
  - `y` → `y_train` und `y_test`

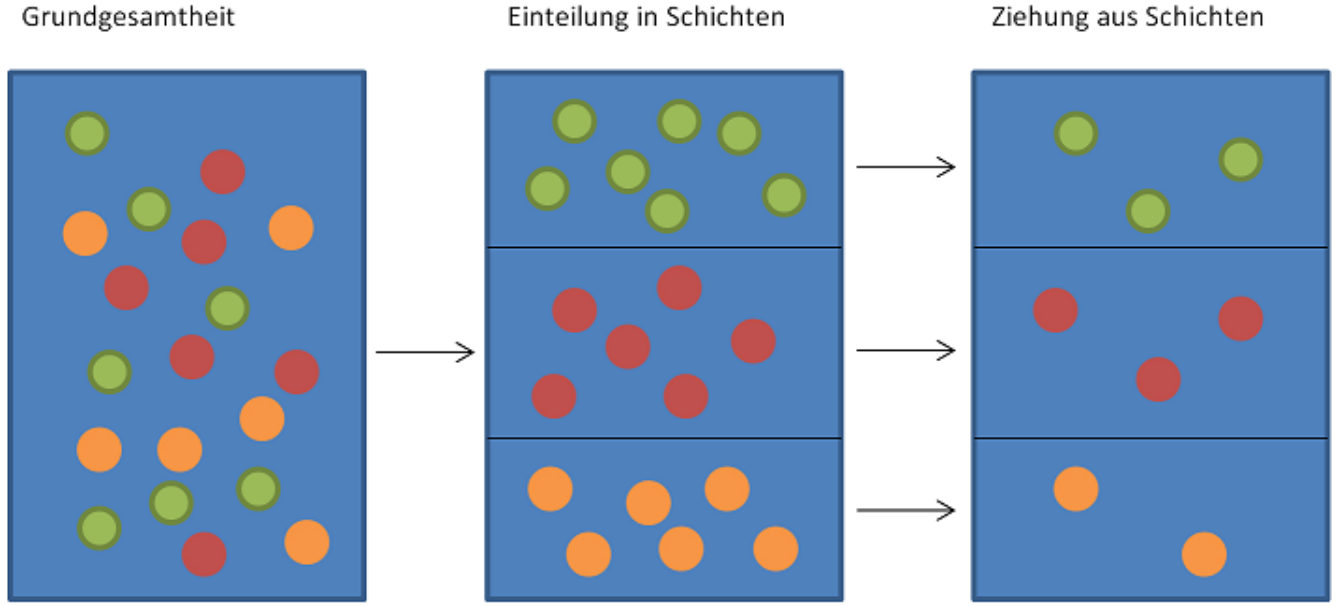
  und zwar so, dass die korrespondierenden X und y Werte nach dem Split dieselben Indices aufweisen

4.1.4 Geschichtetes Sampling

(aka "Stratified Sampling"), eine Spezialform der Stichprobenerstellung

- Motivation / Beispiel: Meinungsumfragen im Vorfeld von Volksabstimmungen, geschichtet nach
  - Geschlecht (2 Levels)
  - Altersklasse (3 Levels)
  - Sprachregion (2 Levels)
  - Stadt/Land (2 Levels)

  das ergibt insgesamt 2*2*2*3=24 verschiedene Kombinationen



- um zu verhindern, dass einzelne Kombinationen zu wenig Treffer erhalten, wird die Stichprobe der zu Befragenden geschichtet

4.1.4 Geschichtetes Sampling

Unterscheidungen

- Proportionale Schichtung:
  Von einer proportional geschichteten Zufallsstichprobe spricht man, wenn die Umfänge der den verschiedenen Schichten entnommenen Stichproben **proportional** zum Anteil der Schicht an der Grundgesamtheit sind: So ist jede Schicht in der Stichprobenauswahl in gleicher Relation wie in der Grundgesamtheit vertreten
- Disproportionale Schichtung:
  Im einfachsten Fall werden aus allen Schichten etwa gleich grosse Zufallsstichproben gezogen
  Ein Motiv für ein solches Vorgehen kann z. B. sein, dass die zu entnehmende Zufallsstichprobe für eine sehr kleine Schicht bei proportionaler Schichtung und vertretbarem Aufwand für die Gesamterhebung zu klein für eine sinnvolle statistische Auswertung wäre [Wikipedia]

4.1.4 Geschichtetes Sampling

Vorgehen mit sklearn:

- sklearn.model_selection.train_test_split bietet zu diesem Zweck den Parameter `stratify` an, welcher beim Train - Test - Split ein proportionales Sampling herbeiführt
- ein Vergleich der Klassenhäufigkeiten der Target-Werte vor dem Split, nach Split (stratify=None) und nach dem Split (stratify='y') zeigt untenstehende Zusammenstellung (vgl. [ipynb])

	vor Split	stratify=None	stratify='y'
no	0.529287	0.527364	0.52934
yes	0.470713	0.472636	0.47066

- mit stratify='y' werden dieselben Mengenverhältnisse erreicht wie vor dem Split (Differenzen wegen runden auf Integerwerten),
ohne stratifizieren dagegen andere Mengenverhältnisse

4.1.5 Externes Sampling

4.1.5.1 Sequentielles Sampling

- Spezialfall: die Datenmenge ist zu gross, als dass sie ins Memory geladen werden könnte
- sortieren verlangt aber genau dies, da externes Sortieren viel zu aufwendig wäre
- Vorgehen
  - vorab festlegen, wie gross der relative Anteil der Beobachtungen ist, welche ins Sample fliessen soll
  - für jede Beobachtung, die sequentiell gelesen wird, wird eine **uniform verteilte** Zufallszahl generiert
  - diese wird anschliessend mit einem Schwellenwert verglichen und entschieden, ob die Beobachtung ins Sample kommt oder nicht

4.1 Validierung und mehr - Sampling und Resampling

4.1.5 Externes Sampling - 4.1.5.1 Sequentielles Sampling

- im untenstehende Codebeispiel wird dazu die Funktion `reader()` aus dem Modul `csv` verwendet, um die Datei zeilenweise lesen zu können (`pandas.read_csv()` liest die ganze Datei auf einmal ein)

```
import numpy as np
import csv
smpl_prop = 0.1 ## proportion of pop for sample
rows = []
with open('bank-additional-full.csv', 'r') as f:
    csv_reader = csv.reader(f, delimiter=';', quoting=csv.QUOTE_NONE)
    header = next(csv_reader) ## pick header
    for row in csv_reader: ## iterate over each row after the header in the csv
        if np.random.rand() < smpl_prop: ## conditinal add row to rows
            rows.append(row)
data = pd.DataFrame(rows, columns=header) ## rows to pandas.DataFrame
print(data.shape)
```

```
(4179, 21)
```

4.1.5 Externes Sampling - 4.1.5.1 Sequentielles Sampling

- dieses Verfahren lässt sich auch einfach mit anderen Programmiersprachen (meist einfach) realisieren, z.B. SQL (hier SQLite)

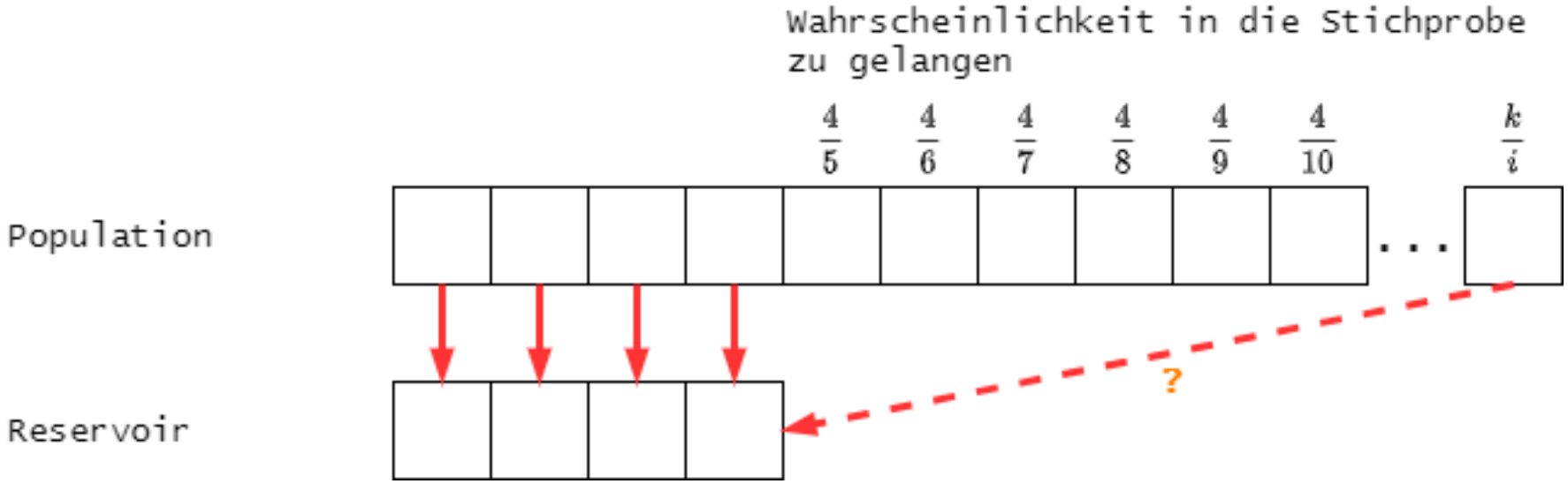
```
select abs(random()) % (10) as rnd, *
from t1
where rnd < 1
```

- (random() von SQLite gibt einen ganzzahligen Wert zwischen -2^{63} und $+2^{63}$ zurück, daher hier der Einsatz der Modulo Funktion "%", was einen Anteil von ca. 10% der Population zurückgibt)

4.1 Validierung und mehr - Sampling und Resampling

4.1.5 Externes Sampling - 4.1.5.2 Reservoir Sampling

- eine Gruppe von Randomisierungs-Algorithmen, um k Objekte aus einer Liste mit n Elementen zufällig auszuwählen, wobei n entweder sehr gross oder zu Beginn unbekannt ist:
  - n ist so gross, dass die Liste nicht im Memory gehalten werden kann
  - n ist anfangs unbekannt, wie beim Behandeln eines Datenstroms (z.B. Sensordaten)



4.1 Validierung und mehr - Sampling und Resampling

4.1.5 Externes Sampling - 4.1.5.2 Reservoir Sampling

- Vorgehen prinzipiell
  - festlegen der Stichprobengrösse k
  - bereitstellen eines Reservoirs der Grösse k, welches die Stichprobe aufnehmen wird
  - übertragen der ersten k Elemente aus der Population direkt ins Reservoir an identischer Position
  - ab dem Element k+1 der Population müssen folgende Entscheidungen getroffen werden:
    - soll das Element ins Reservoir übernommen werden?
    - wenn ja, an welche Position?

- Voraussetzung:
  - jedes Element der Population (Stream) soll dieselbe Chance bekommen, in das Reservoir (Stichprobe) zu gelangen
  - die Position, wo ein ausgewähltes Element im Reservoir eingefügt wird, soll ebenfalls zufällig sein

4.1 Validierung und mehr - Sampling und Resampling

4.1.5 Externes Sampling - 4.1.5.2 Reservoir Sampling

- die beiden genannten Entscheidungen (ob und wohin) können mit einem eleganten Algorithmus gefällt werden
- für jedes Element E_i der Population ($i > k$) wird eine ganzzahlige Zufallszahl r generiert so dass $1 \leq r \leq i$
- wenn $r \leq k$
  - dann wird E_i gerade an der Position r ins Reservoir übernommen
  - sonst verworfen

- ein ausprogrammiertes Beispiel befindet sich hier
- in scikit-learn steht folgende Klasse zur Verfügung: sklearn.utils.random.sample_without_replacement, welche aus einem gewünschten Range von 0 bis n k Elemente nach dieser Methode herauszieht
- um sie z.B. auf einen Data Frame anzuwenden, kann nach Erstellen des Sample dieses als Filterindex verwendet werden (vgl. [ipynb])

4.1 Validierung und mehr - Sampling und Resampling

Workshop 12

Gruppen zu 2 bis 4, Zeit: 30'

- das Original des Bankkundendatasets (bank-additional-full.csv) enthält 41187 Beobachtungen
- vergleichen Sie die Mengenverhältnisse von yes und no bei Samples von
  - 1000 bis 40000 in Schritten von 1000