Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

2024 FS CAS PML - Supervised Learning
4 Validierung (und mehr)
4.5 Unbalancierte Daten

Werner Dähler 2024

4 Validierung und mehr - AGENDA

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

4.5 Validierung und mehr - Unbalancierte Daten

4.5.1 Motivation und Vorbereitung

- das bisher verwendete Dataset bank.csv wurde aus didaktischen Gründen aus einem im Web verfügbaren Dataset etwas adaptiert:
  - Missing Values, um den Umgang mit denselben diskutieren zu können
  - Sampling, um die Datenmenge etwas zu reduzieren
    dabei wurde aber eine besondere Sampling Technik verwendet, die im Folgenden vorgestellt und begründet wird
- eine Untersuchung zur Verteilung der Target-Werte im Originaldataset (bank-additional-full.csv) ergibt folgendes [ipynb]:

```
print('dim =', bank_full_df.shape)
```

```
dim = (41188, 21)
```

```
print(bank_full_df.y.value_counts(normalize=True))
```

```
no     0.887346
yes    0.112654
```

4.5 Validierung und mehr - Unbalancierte Daten

4.5.1 Motivation und Vorbereitung

- das Dataset umfasst also 41188 Beobachtungen
- die Verteilung des Targets (y) ist extrem einseitig - unbalanciert: knapp 89% aller Beobachtungen weisen den Wert "no" auf
- ein möglichst einfacher Learner, z.B. DecisionTreeClassifier ohne Split, gibt als Prediction den Modalwert des Targets zurück, was zu einer Accuracy von 0.887 führen würde
- solche unbalancierten Datasets haben die Tendenz, bei den meisten Learner zu einem systematischen Fehler (Bias) zugunsten der dominierenden Klasse zu führen
- am offensichtlichsten ist dieses Verhalten wohl bei NaiveBayes, da dort die Mengenverhältnisse der Targetklassen als Apriori Wahrscheinlichkeit in die Prediction einfliessen
- um derartige Verzerrungen ausgleichen zu können, stehen verschiedene Möglichkeiten zur Verfügung
  - over-sampling und under-sampling
  - verwenden von Gewichten beim Trainieren

4.5 Validierung und mehr - Unbalancierte Daten

4.5.1 Motivation und Vorbereitung

nomenklatorisches:

- Data Frame mit dem kompletten Dataset:
  - bank_full_df
- Features und Target des kompletten Datasets:
  - X_full, y_full, mit minimalem Feature Engineering: One Hot Encoding auf allen nicht numerischen Features von X
- Features und Target nach Train - Test - Split
  - X_full_train, X_full_test, y_full_train, y_full_test
- Features und Target von train nach resampling
  - X_resampled_train, y_resampled_train

- üblicherweise werden diese Arten des Resampling nur auf den Trainingsdaten durchgeführt

4.5 Validierung und mehr - Unbalancierte Daten

4.5.1 Motivation und Vorbereitung

- Vorbereiten der Daten (vgl. [ipynb])
  - laden in Data Frame
  - Features - Target - Split
  - One Hot Encoding auf alle nicht numerischen Features
  - Test - Train - Split

- damit im Folgenden die Auswirkungen der verschiedenen gezeigten Methoden effizient und ohne allzu viel redundanten Code verglichen werden können, wird vorab eine Funktion definiert, welche
  - Features und Targets von Train- und Testset als Parameter entgegennimmt
  - RandomForestClassifier auf den Trainingsdaten trainiert
  - folgendes zurückgibt:
    - den internen Score Wert (accuracy) bei Anwendung des Modells auf die Testdaten
    - die relativen Mengenverhältnisse der Trainingsdaten nach dem Resampling

4.5 Validierung und mehr - Unbalancierte Daten

4.5.1 Motivation und Vorbereitung

- Definition der Funktion

```
from sklearn.ensemble import RandomForestClassifier
def getResampledRfScore(X_train, y_train, X_test, y_test):
    model = RandomForestClassifier(random_state=1234)
    model.fit(X_train, y_train)
    print('score ', model.score(X_test, y_test))
    print(y_train.value_counts(normalize=True))
```

- und aufgerufen auf den nicht gesampelten Daten

```
getResampledRfScore(X_full_train, y_full_train, X_full_test, y_full_test)
```

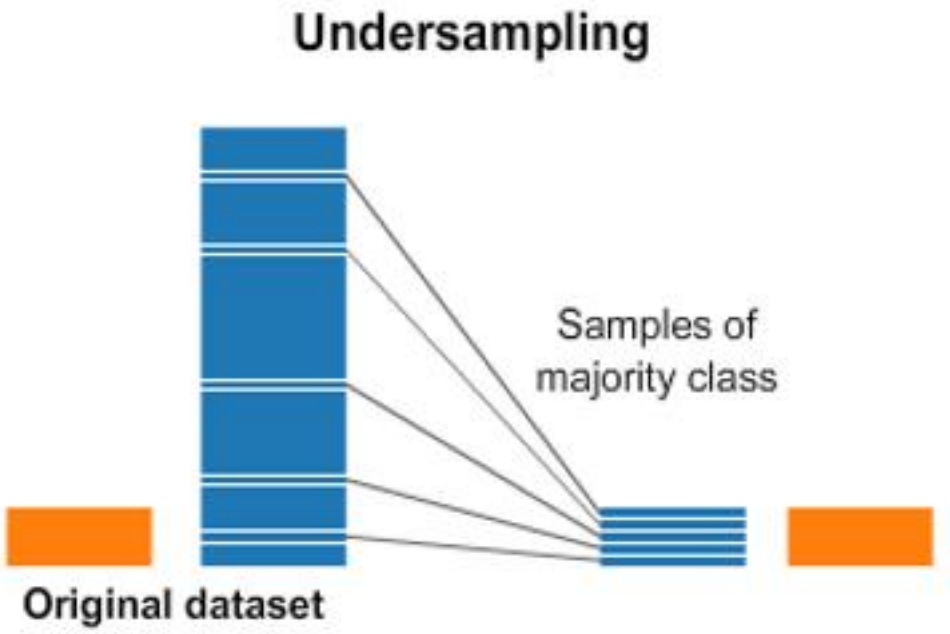
```
score  0.9131828113619811
no     0.886773
yes    0.113227
```

4.5 Validierung und mehr - Unbalancierte Daten

4.5.2 Random under-sampling

- es wird eine Zufallsstichprobe (ohne oder mit Zurücklegen) der Klasse mit mehr Instanzen erzeugt, welche gerade (oder annähernd) der Anzahl Instanzen der Gruppe mit weniger Instanzen entspricht
- dieses Vorgehen ist relativ einfach verständlich und auch programmierbar
- andere noch folgende Verfahren sind dagegen etwas anspruchsvoller, daher wird im Folgenden auf die Library imblearn zurückgegriffen, (welche aus den Notebook heraus nachinstalliert werden kann:)

```
pip install -U imbalanced-learn
```



[Quelle für die Visualisierungen in diesem Kapitel]

4.5 Validierung und mehr - Unbalancierte Daten

4.5.2 Random under-sampling

- aufrufen von imblearn.under_sampling.RandomUnderSampler und sichten des Ergebnisses mit der oben definierten Funktion

```
from imblearn.under_sampling import RandomUnderSampler
rus = RandomUnderSampler(random_state=1234)
X_resampled_train, y_resampled_train =\
    rus.fit_resample(X_full_train, y_full_train)
getResampledRfScore(...
```

```
score  0.8481427530954115
yes    0.5
no     0.5
```

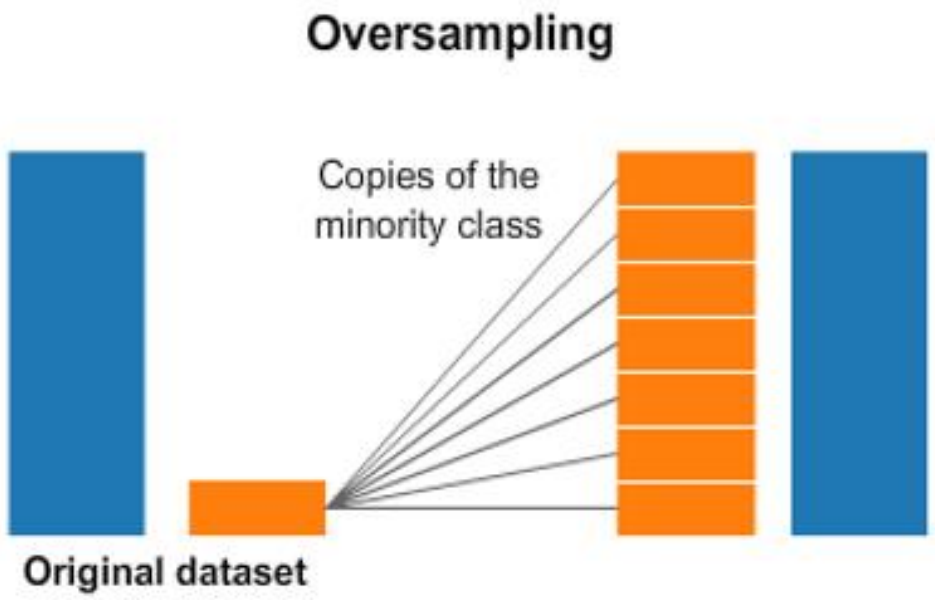
4.5 Validierung und mehr - Unbalancierte Daten

4.5.3 Random over-sampling

- aus der Klasse mit weniger Instanzen wird eine Zufallsstichproben (**mit Zurücklegen!**) erzeugt, welche gleich gross ist wie die Gruppe mit mehr Instanzen
- aufrufen der Funktion

```
from imblearn.over_sampling import\
    RandomOverSampler
ros = RandomOverSampler(random_state=1234)
X_resampled_train, y_resampled_train =\
    ros.fit_resample(X_full_train, y_full_train)
getResampledRfScore(...
```

```
score  0.9043699927166788
no     0.5
yes    0.5
```



4.5 Validierung und mehr - Unbalancierte Daten

4.5.3 Random over-sampling

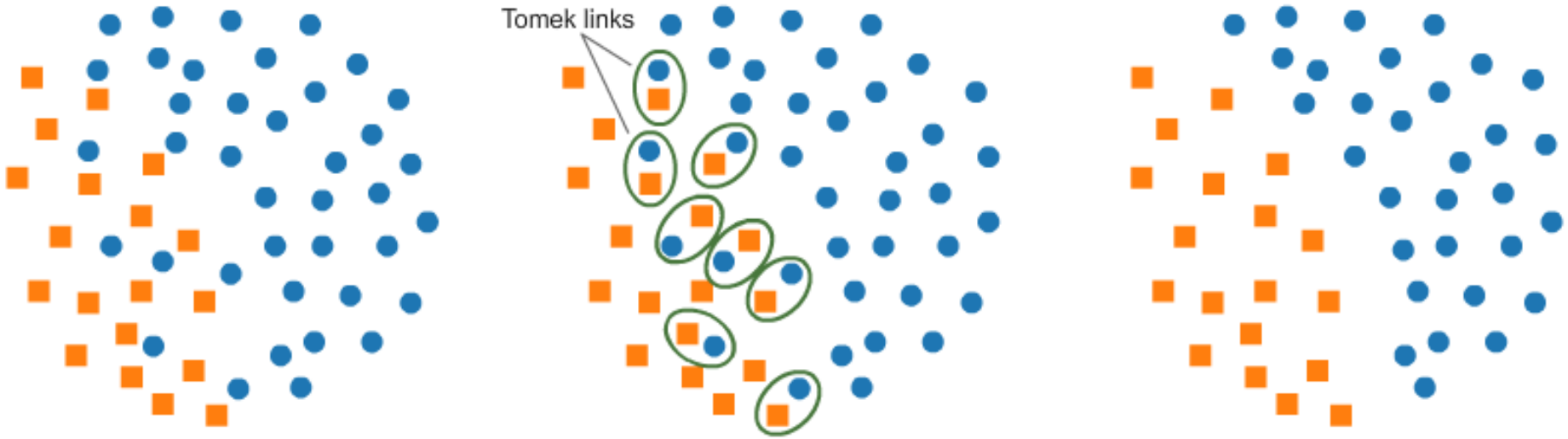
Fazit des bisherigen:

- over-sampling ist zwar leicht besser als full, möglicherweise wird hier aber der Bias noch verstärkt
- under-sampling ist deutlich schlechter, allerdings aus theoretischen Überlegungen glaubwürdiger

4.5 Validierung und mehr - Unbalancierte Daten

4.5.4 Undersampling mit Tomek Links

- Tomek Links sind Paare von Beobachtungen unterschiedlicher Klasse, die sich aber ansonsten ähnlich sind
- der Algorithmus entfernt bei solchen Paaren das Objekt der Mehrheitsklasse, was zu einer besseren Klassifikationsgrenze führen kann (!)



4.5 Validierung und mehr - Unbalancierte Daten

4.5.4 Undersampling mit Tomek Links

- Ausführung

```
from imblearn.under_sampling import TomekLinks
tl = TomekLinks()
X_resampled_train, y_resampled_train = tl.fit_resample(
    X_full_train, y_full_train)
getResampledRfScore(...
```

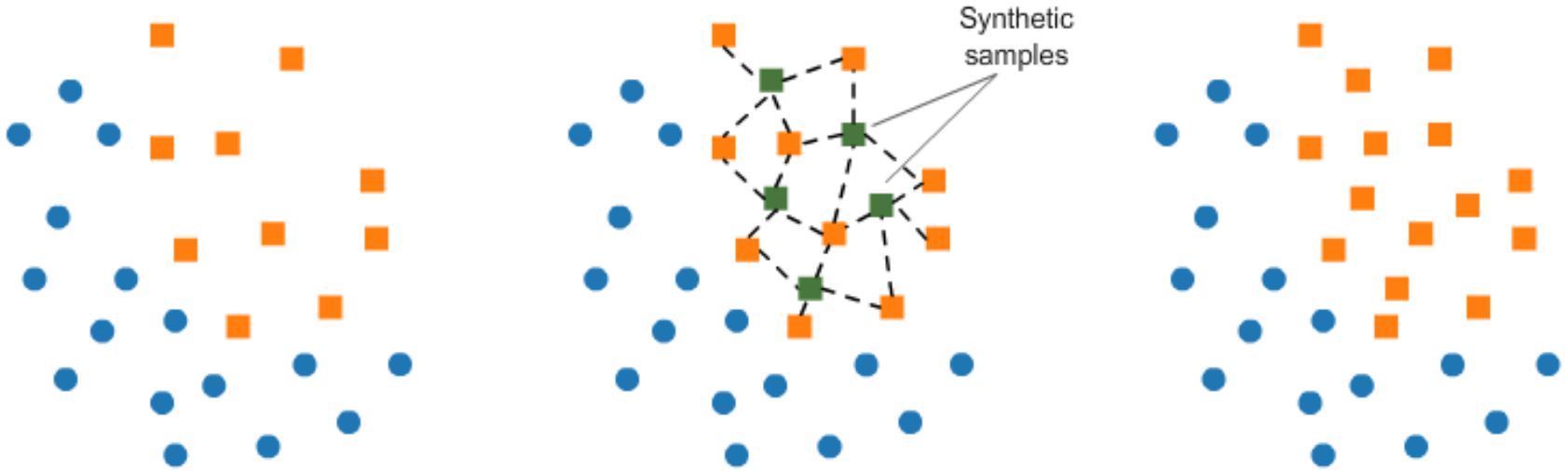
```
score  0.9114348142753096
no     0.883054
yes    0.116946
```

- eine nachträgliche Untersuchung nach den Mengenverhältnissen zeigt, dass die Majority-Klasse bloss von 24349 auf 23476 reduziert wurde und noch sehr weit von der Anzahl der Minority-Klasse entfernt liegt
  (hier muss möglicherweise die resampling_strategy noch überdacht werden)

4.5 Validierung und mehr - Unbalancierte Daten

4.5.5 Oversampling mit SMOTE

- SMOTE (Synthetic Minority Oversampling Technique) generiert synthetische Beobachtungen in der Nähe von existierenden Beobachtungen der Minderheitsklasse



4.5 Validierung und mehr - Unbalancierte Daten

4.5.5 Oversampling mit SMOTE

- Ausführung:

```
from imblearn.over_sampling import SMOTE
sm = SMOTE()
X_resampled_train, y_resampled_train = sm.fit_resample(
    X_full_train, y_full_train)
print(getResampledRfScore(...
```

```
score  0.9058266569555717
no     0.5
yes    0.5
```

4.5 Validierung und mehr - Unbalancierte Daten

4.5.5 Oversampling mit SMOTE

Fazit

- eine Zusammenstellung der erreichten Resultate mit den verschiedenen Methoden zeigt untenstehende Tabelle

Method	Accuracy	no	yes
ohne	0.9132	0.8868	0.1132
Random under-sampling	0.8481	0.5000	0.5000
Random over-sampling	0.9044	0.5000	0.5000
Undersampling mit Tomek Links	0.9114	0.8831	0.1169
Oversampling mit SMOTE	0.9058	0.5000	0.5000

- Tomek Links und SMOTE zeigen auf diesem Beispiel (mit Standard Parametrisierung) kaum eine Wirkung
- es kann aber durchaus sein, dass dies auf einer anderen Datenlage eine Verbesserung bringen mag

(in imbalanced-learn stehen noch einige weitere Methoden zur Verfügung, vgl. Doku)

4.5 Validierung und mehr - Unbalancierte Daten

4.5.6 Weights beim Trainieren

- zwar kein Resampling, aber eine weitere Möglichkeit zum Umgang mit nicht balancierten Daten, welche von vielen Klassifikatoren intern unterstützt wird
- z.B. von RandomForestClassifier:

```
model = RandomForestClassifier(n_estimators = 100, class_weight='balanced',
    random_state=1234)
model.fit(X_train, y_train)
print(model.score(X_test, y_test))
```

```
0.8801736613603474
```

- wobei gemäss Dokumentation der Modus 'balanced' die Gewichtungen von y umgekehrt proportional zu den Frequenzen der vorliegenden Klassen berechnet

$$weights = \frac{n\_samples}{n\_classes * np\_bincount(y)}$$