

9. Workshop der Reihe Softwaretechnik

Maschinelles Lernen

Weiterbildung für Informatik-Lehrkräfte

Freitag, 27. September 2019

Grundlagen

Worum geht es?

	TV	radio	newspaper	sales
<i>observations</i>	230,1	37,8	69,2	22,1
	44,5	39,3	45,1	10,4
	17,2	45,9	69,3	9,3
	151,5	41,3	58,5	18,5
	180,8	10,8	58,4	12,9

	<i>features</i>			<i>target</i>

- Ziel: Mit Hilfe der Investitionen in Fernseh-, Radio- und Zeitungswerbung den Produktverkauf schätzen/vorhersagen.

Algorithmen

Für Prognosen gibt es eine Vielzahl von Verfahren wie

- Lineare Regression
- Random-Forests
- Gradient-Boosting
- Neuronale Netze

...

Prognosen

“Prognosen sind schwierig, besonders wenn sie die Zukunft betreffen.”

Carl Valentin

- Wie groß ist der Fehler?
- Was bedeutet überhaupt ‘Fehler’

Fehler

- Reellwertige Funktion: $F(y, y')$
- Wobei y der tatsächliche Wert und y' der Schätzwert des Targets ist.
- y und y' können einzelne Werte oder Vektoren von Zahlen sein.

Medium Average Error

$$\text{MAE} = \frac{1}{n} \sum_{j=1}^n |y_j - \hat{y}_j|$$

Die mittlere Abweichung erscheint als der 'natürliche' Fehler.

Nachteil: Einige Algorithmen minimieren Fehler mit Hilfe der Differenzialrechnung. Der Betrag ist *nicht* differenzierbar.

Rooted Mean Squared Error

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{j=1}^n (y_j - \hat{y}_j)^2}$$

Der RMSE wird sehr häufig genutzt. Es gibt weitere Fehlerfunktionen.

1. Versuch

Mittelwert bisheriger sales-Werte als Vorhersage nutzen.

Demo mit Excel

Fehler

- Wenn für die Prognose der *Mittelwert* genutzt wird, ergibt sich ein RMSE von 5.2.
- Diese erste Schätzung gibt uns einen Vergleichswert für Prognosen mit anderen Algorithmen.
- Die Qualität der Algorithmen wird mit Hilfe der Fehler beurteilt.
- Es ist durchaus üblich den *Mittelwert* als *erste Schätzung* zu nehmen. Er ist die 'baseline'.
- Es gibt Probleme, bei denen es schwer ist, diese Baseline deutlich zu verbessern.

2. Versuch

- In die Schätzung mit Hilfe des Durchschnitts gehen die Features tv, news und radio bisher gar nicht ein.
- Eine Schätzung, in die mindestens eines dieser Features eingeht, hat *vermutlich* einen geringeren Fehler.

Idee (Lineare Regression):

- Finde Parameter a, b, c und d mit

$$\text{sales} = a * \text{tv} + b * \text{newspaper} + c * \text{radio} + d$$

Lineare Regression

Für welche Parameter a , b , c und d wird der RMSE für die Vorhersage

$$\text{sales} = a * \text{tv} + b * \text{newspaper} + c * \text{radio} + d$$

minimiert?

Kann man die minimierenden Parameter überhaupt explizit angeben?

Die Methode der kleinsten Quadrate



Ja, man kann die Parameter explizit angeben!

(Gauss um 1800)

Lineare Regression

Excel hilft uns bei der linearen Regression mit *einer* Variablen.

Demo mit Excel

Lineare Regression

Es ergibt sich ein RMSE von 3.24, der deutlich besser als der Fehler aus dem Mittelwert (5.2) ist.

Modelle

Im Maschinellen Lernen (ML) ist der Begriff ‚Schätzfunktionen‘ unüblich.
Man spricht vielmehr von *Modellen*.

Der Algorithmus (hier die Methode der kleinsten Quadrate) ermittelt die Modellparameter a , b und c .

Lineare Regression

Excel unterstützt lineare Regression nur mit *einer* Variablen. Allgemein werden Machine Learning Verfahren in *Programmiersprachen* genutzt. Insbesondere

- Python
- R

Die Funktionalität beider Sprachen ist durch Pakete beliebig erweiterbar. Die ML-Algorithmen müssen nicht selbst implementiert werden.

Python ist im ML weiter verbreitet als R.

Die Entwicklungsumgebung

Als Entwicklungsumgebung hat sich Python mit der IDE PyCharm (Community Edition reicht) bewährt.

Die Installation ist einfach.

- Vorher sicherstellen, dass python3 installiert ist: Terminal öffnen und Befehl `python3` eintippen. Gegebenenfalls installieren.
- Danach PyCharm installieren

Die Entwicklungsumgebung

Python wird erst durch geeignete Pakete zu einem leistungsstarken ML-System. Hier werden die folgenden Pakete benötigt:

- scikit-learn
- pandas
- xlrd
- light-gbm

Die Installation wird in den Begleitvideos gezeigt.

Lineare Regression

```
import numpy as np
from sklearn.linear_model import LinearRegression
import pandas as pd
from sklearn.metrics import mean_squared_error
filename = '../data/Advertising.xlsx'
advertising = pd.read_excel(filename)
target = advertising.iloc[:, -1]
features = advertising.iloc[:, :-1]
regressor = LinearRegression()
regressor.fit(features, target)
predictions = regressor.predict(features)
rmse = np.sqrt(mean_squared_error(target, predictions))
print(rmse)
```

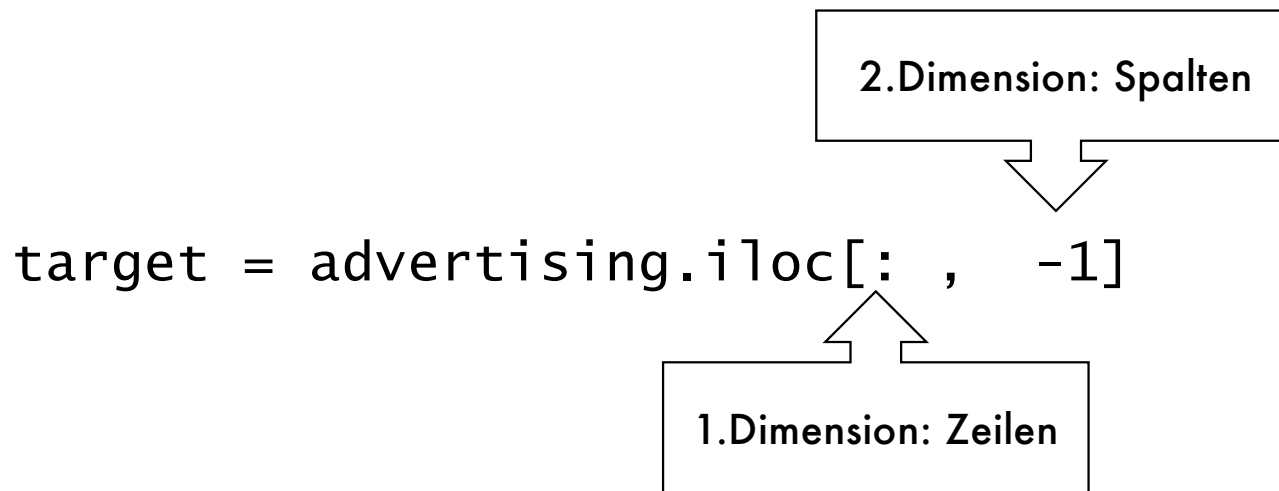
Pakete

Das Paket scikit-learn enthält viele ML-Algorithmen

Eine weiteres wichtiges Paket ist pandas: Tabellarische Daten können

- bequem aus Dateien gelesen werden und
- mit pandas-Funktionen ähnlich wie Tabellen mit SQL verarbeitet werden.

Pandas



Die Variable `target` enthält aus allen Zeilen die letzte Spalte.

Pandas

```
features = advertising.iloc[:, :-1]
```

Die Variable `features` enthält aus allen Zeilen alle bis auf die letzte Spalte.

Ergebnis

Es ergibt sich ein RMSE von 1.67, der deutlich besser als der Fehler aus der linearen Regression mit *einer* Variablen (3.24) ist.

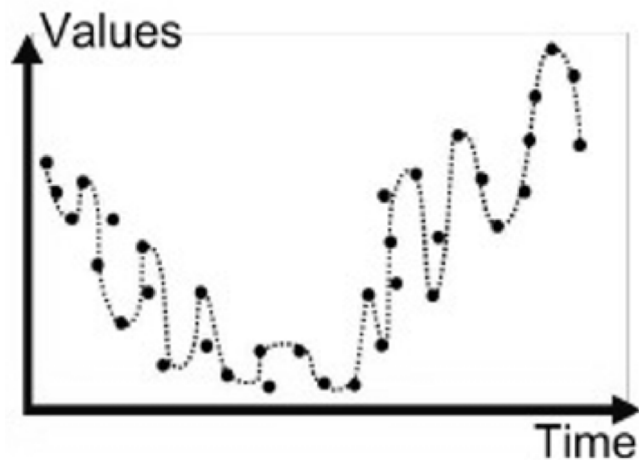
Vorsicht!

- Das Modell wurde mit Daten entwickelt.
- Die Vorhersagen wurden mit *den gleichen Daten* ermittelt.
- Das Modell ‚kennt‘ also schon die Targets der Daten, für die es Vorhersagen machen soll.

Überanpassung - Overfitting

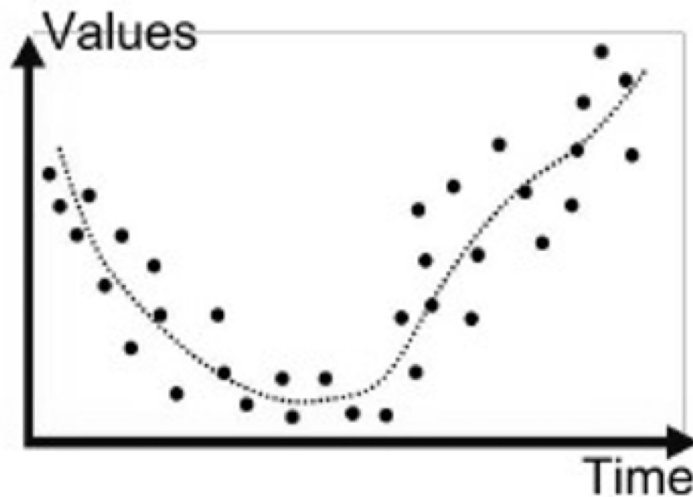
- Sehr flexible Algorithmen lernen bei der Modellbildung die Daten 'auswendig'. Die resultierenden Fehler sind dann sehr klein.
- Nimmt man unbekannte Daten, ergeben sich viel größere Fehler

Überanpassung - Overfitting



- Hier ist das Modell sehr genau an die Daten angepasst.
- Für ähnliche aber abweichende Daten ist es aber vermutlich nicht geeignet.

Kein Overfitting



- Bei diesem Modell wird bewusst ein größerer Fehler *des Modells* in Kauf genommen.
- Dafür ist der Fehler für unbekannte Daten aber vermutlich geringer.

Wie erkennt man Overfitting?

- Die Daten werden jetzt in Trainings- und Testdaten getrennt (80%/20%).
- Das Modell wird aus den Trainingsdaten entwickelt und der resultierende Fehler berechnet.
- Für die Testdaten sind die Werte des Targets bekannt. Man kann mit Hilfe des Modells daher ebenfalls den Fehler berechnen.
- Ist der Fehler für die Trainingsdaten deutlich geringer als für die Testdaten, ist das ein Indiz für Overfitting.

Test- und Trainingsdaten getrennt

Python unterstützt die Trennung von Test- und Trainingsdaten. Am Programm muss im wesentlichen eine Zeile geändert werden.

Test- und Trainingsdaten getrennt

```
...
from sklearn.model_selection import train_test_split
filename= '../data/Advertising.xlsx'
advertising = pd.read_excel(filename)
target=advertising.iloc[:,-1]
features=advertising.iloc[:,-1]
train_features, test_features, train_target,test_target =train_test_split(
    features, target, test_size=0.2, random_state=4711)
regressor = LinearRegression()
regressor.fit(train_features, train_target)
predictions=regressor.predict(test_features)
rmse=np.sqrt(mean_squared_error(test_target, predictions))
print(rmse)
```

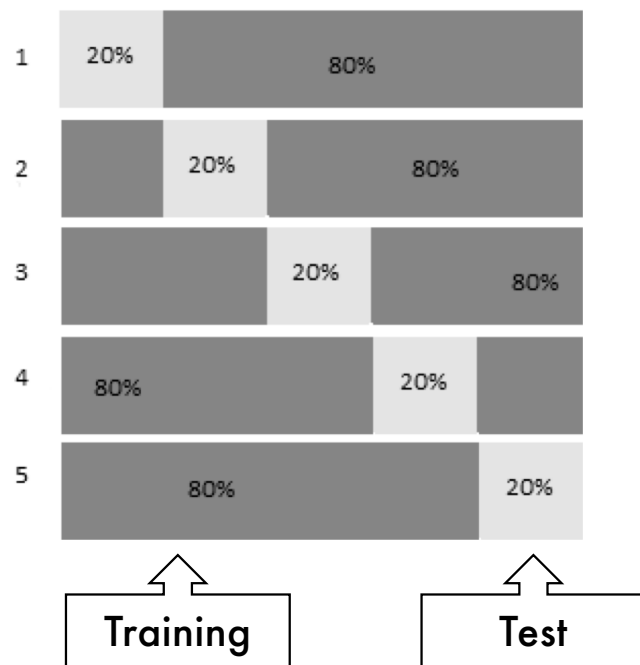
Ein überraschendes Ergebnis

- RMSE für die Testdaten: 1.2
- Vorher wurde der Fehler 1.67 für *alle* Daten, also ohne Trennung ermittelt.
- Erwartung: Fehler für Trainingsdaten mindestens genauso gut wie für Testdaten.

Ein überraschendes Ergebnis

- Im Beispiel ist der Testfehler überraschenderweise kleiner als der Trainingsfehler.
- Solche Ergebnisse sind möglich, weil die Testdaten *zufällig* ermittelt werden.

Kreuzvalidierung – Cross Validation



- Besser ist es nach dem abgebildeten Schema Versuche mit mehreren Modellen auszuführen.
- Man erhält so mehrere Fehler, die insgesamt aussagekräftiger oder weniger vom Zufall abhängen.

Kreuzvalidierung

```
...
from sklearn.model_selection import cross_val_score
filename= '../data/Advertising.xlsx'
advertising = pd.read_excel(filename)
advertising=advertising.sample(frac=1, random_state=4711)
target=advertising.iloc[:,-1]
features=advertising.iloc[:,-1]
regressor = LinearRegression()
scores = cross_val_score(
    regressor, features, target, cv=5, scoring='neg_mean_squared_error')
print(np.sqrt (-scores))
print(np.sqrt (-scores).mean())
```

Ergebnis

CV-Scores: [1.20 1.99 1.84 1.72 1.80]

Mittelwert: 1.71

- Man sieht, dass der Fehler für einzelne Testdaten *zufällig* sehr gut sein *kann*.
- Kreuzvalidierung (CV) ist in aller Regel unabdingbar!

3. Versuch: Light Gradient Boosting

- *Lineare Regression* ist ein vergleichsweise einfaches, klassisches Verfahren, das in vielen Fällen vollkommenen ausreicht.
- *Light Gradient Boosting* (2017) ist ein sehr leistungsstarkes Verfahren, das oft mit verblüffender Präzision prognostiziert.
- LightGBM ist als Paket für Python verfügbar und muss nachinstalliert werden. Die Handhabung ist aber wie bei der linearen Regression.

Light Gradient Boosting

```
...
from lightgbm import LGBMRegressor
filename = '../data/Advertising.xlsx'
advertising = pd.read_excel(filename)
advertising = advertising.sample(frac=1, random_state=4711)
target = advertising.iloc[:, -1]
features = advertising.iloc[:, :-1]
regressor = LGBMRegressor(n_estimators=1000, learning_rate=0.01)
scores = cross_val_score(
    regressor, features, target, cv=5, scoring='neg_mean_squared_error')
print(np.sqrt(-scores))
print(np.sqrt(-scores).mean())
```

Ergebnis

CV-Scores: [0.70 1.51 1.08 0.62 0.88]

Mittelwert: 0.96

- Der Fehler ist *deutlich* niedriger als bei der linearen Regression
- Ohne Kreuzvalidierung ist der Fehler für die *Trainingsdaten* 0.51 noch niedriger. Hieran und an der Varianz in der Kreuzvalidierung erkennt man Overfitting.

Hyperparameter Optimierung

In der Code-Zeile

```
LGBMRegressor(n_estimators=1000, learning_rate=0.01)
```

treten zwei so genannte Hyperparameter (`n_estimators` und `learning_rate`) auf.

In LightGBM gibt es Dutzende solcher Parameter. Durch geschickte Tuning-Maßnahmen, kann die Qualität der Prognose weiter verbessert und Overfitting vermieden werden.

Interessante Daten?

Daten, die Auswirkungen von Werbemaßnahmen auf den Produktabsatz aufzeigen, sind gute einführende Beispiele.

Es gibt aber erheblich spannendere Datensätze.

Kaggle

- Viele Datensätze findet man auf
www.kaggle.com
- Die Daten werden teilweise in Form von Wettbewerben angeboten, für die 6-7-stellige Dollarbeträge als Siegprämie ausgelobt werden.
- In sogenannten Kernels demonstrieren einige Teilnehmer auch Programmcode und weitere Einsichten, die sie für ihre Lösung entwickelt haben.
- Kaggle ist ein exzellentes Werkzeug, um seine Kenntnisse in Machine Learning zu vertiefen.
- Die folgenden Datensätze wurden auch intensiv auf Kaggle diskutiert.

Wer hat auf der Titanic überlebt?

Beispieldaten

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	0	3	Braund, Mr. Owen Harris	male	22	1	0	A/5 21171	7.25		S
2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Thayer)	female	38	1	0	PC 17599	71.2833	C85	C
3	1	3	Heikkinen, Miss. Laina	female	26	0	0	STON/O2. 3101282	7.925		S
4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35	1	0	113803	53.1	C123	S
5	0	3	Allen, Mr. William Henry	male	35	0	0	373450	8.05		S
6	0	3	Moran, Mr. James	male		0	0	330877	8.4583		Q
7	0	1	McCarthy, Mr. Timothy J	male	54	0	0	17463	51.8625	E46	S
8	0	3	Palsson, Master. Gosta Leonard	male	2	3	1	349909	21.075		S
9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27	0	2	347742	11.1333		S
10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14	1	0	237736	30.0708		C

- Survived ist offenbar Target.
- SibSp: Anzahl der Ehegatten oder Geschwister an Bord.
- Parch: Anzahl der Elternteile oder Kinder an Bord
- Embarked: Der Hafen der Einschiffung

Features

- Einige Features sind Texte.
- Verfahren wie Lineare Regression oder LightGBM können aber nur Zahlen verarbeiten.
- Wir werden sinnvolle Techniken zur Transformation von Texte in Zahlen kennen lernen.
- Einige Spalten wie PassengerId, Name, Ticket oder Cabin scheinen nicht so aussagekräftig zu sein und werden entfernt.

Feature Engineering I

```
filename = '../data/titanic.csv'  
titanic = pd.read_csv(filename)  
titanic = titanic.drop(columns="PassengerId")  
titanic = titanic.drop(columns="Name")  
titanic = titanic.drop(columns="Ticket")  
titanic = titanic.drop(columns="Cabin")
```

One-Hot-Encoding

- In der Spalte Embarked findet man die Werte C, Q, S und T. Die Funktion

```
pd.get_dummies(embarked)
```

- überführt die Spalte in die vier *neuen* Spalten C, Q, S und T.
- Für einen Datensatz ist als Wert ist in der zu C, Q, S oder T gehörenden 1 eingetragen

One-Hot-Encoding

Embarked
S
C
S
S
S
S
Q
S
S
S



C	Q	S	T
0	0	1	0
1	0	0	0
0	0	1	0
0	0	1	0
0	0	1	0
0	1	0	0
0	0	1	0
0	0	1	0
0	0	1	0

Feature Engineering II

```
...
embarked = titanic['Embarked']
embarked = pd.get_dummies(embarked)
titanic = pd.concat([titanic, embarked], axis=1)
titanic = titanic.drop(columns="Embarked")

sex = titanic['Sex']
sex = pd.get_dummies(sex)
titanic = pd.concat([titanic, sex], axis=1)
titanic = titanic.drop(columns="Sex")

titanic = titanic.fillna(titanic.mean())
```

Unbekannte Werte

- Einige Tabelleneinträge sind leer. Die zugehörigen Werte sind unbekannt.
- Die meisten Algorithmen können unbekannte Werten nicht verwenden.

Unbekannte Werte

Man kann

- die zugehörige Spalte entfernen
- den zugehörigen Datensatz entfernen oder
- den Wert schätzen.

Keine der drei Vorgehensweise ist perfekt!

Wir gehen den dritten Weg: Die letzte Zeile

```
titanic = titanic.fillna(titanic.mean())
```

ersetzt unbekannte Werte durch den Mittelwert der zugehörigen Spalte.

Die Baseline

891 Datensätze

549 Tote

342 Überlebende

- Schätzt man, dass niemand überlebt hat, erhält man eine Genauigkeit von $549/891=0.616$
- Bei Klassifikationsproblemen wird oft die Mehrheitsklasse für die Schätzung der Baseline herangezogen.

Klassifikation

- Beim ersten Problem sollte der Umsatz (sales) mit Hilfe von tv, news und radio geschätzt werden.
- Dieser Problemtyp wird als *Regression* bezeichnet.
- Bei den aktuellen Daten soll die Zugehörigkeit zu einer Klasse (survived=0 oder 1) geschätzt werden.
- Dieser Problemtyp wird als *Klassifikation* bezeichnet.
- Entsprechend arbeiten wir mit dem Datentyp `LGBMClassifier` und nicht wie bisher mit `LGBMRegressor`.

Klassifikation

```
...
titanic = titanic.sample(frac=1, random_state=4711)
target = titanic.iloc[:, 0]
features = titanic.iloc[:, 1:]
classifier = LGBMClassifier(n_estimators=1000, learning_rate=0.01)
scores = cross_val_score(
    classifier, features, target, cv=5, scoring='accuracy')
print(scores)
print(scores.mean())
```

Ergebnis

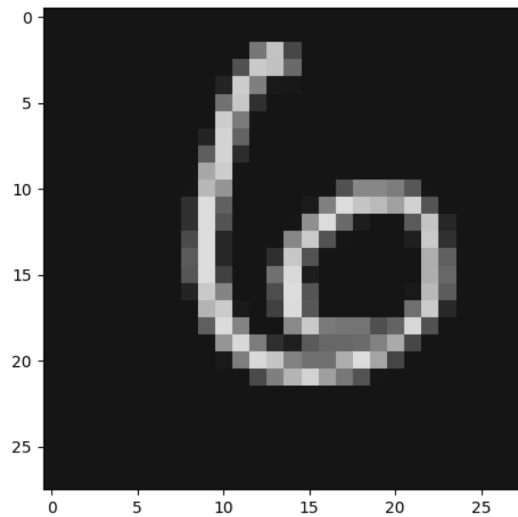
CV-Scores: [0.810 0.78 0.84 0.80 0.77]

Mittelwert: 0.80

- Dies ist besser als die Baseline (0.616).
- Durch ein sorgfältigeres Feature Engineering erreicht man Werte jenseits der 90%.

Handgeschriebene Ziffern erkennen

MNIST



- Die MNIST Datenbank ist eine Sammlung von ca 60.000 handgeschriebenen Ziffern.
- Jede Ziffer tritt gleich oft auf.
- Man leitet sich eine Baseline von 10% her, indem man immer eine ‚0‘ schätzt.
- Es gibt Algorithmen, die deutlich über 99% der Ziffern korrekt erkennen.

Woher kommen die Features?

- Die Bilder liegen in Form von quadratischen Matrizen vor.
- Jeder der 28x28 Einträge entspricht einem Pixel.
- Die Bilder werden in Arrays mit $28 * 28 = 784$ Features transformiert.
- Target ist jeweils die Ziffer.
- Hier arbeiten wir nur mit 10.000 verschiedenen Bildern, also einem kleinen Teil der Gesamtdaten.
- Die Datensätze wurden bereits mit Werkzeugen in eine Datei im csv-Format geschrieben.

Die Daten

Die Bilder werden in einer Comma-Separated-Values-Datei (csv) mit $1 + 28 \times 28$ Spalten – also das Target und ein Wert je Pixel – eingetragen.

```
label,1x1,1x2, ..., 28x27,28x28  
7,0,0,...,0,0  
1,0,0,...,0,0  
...
```

Warum sind die hier gezeigten Pixel-Werte alle 0?

Ziffern lesen und darstellen

```
...
import matplotlib.pyplot as plt
filename = '../data/mnist_small.csv'
mnist = pd.read_csv(filename )
print(mnist.shape)
target = mnist.iloc[:, 0]
features = mnist.iloc[:, 1:]
def show_digit(index):
    pixels = features.iloc[index, :].values.reshape(28, 28)
    print("The below image should be a ", target[index])
    plt.imshow(pixels)
    plt.show()
show_digit(100)
```

Klassifikation

```
classifier = LGBMClassifier(n_estimators=100, learning_rate=0.1)
scores = cross_val_score(
    classifier, features, target, cv=2, scoring='accuracy')
print(scores)
print(scores.mean())
```

Ergebnis

Um das Verfahren abzukürzen, wurde die Kreuzvalidierung mit 50% Trainings- und 50% Testdaten, also in zwei Versuchen, durchgeführt.

CV-Scores: [0.88 0.94]

Mittelwert: 0.91

Literatur

