Data Mining und Maschinelles Lernen

Prof. Kristian Kersting
Steven Lang
Johannes Czech

TECHNISCHE UNIVERSITÄT DARMSTADT

Sommersemester 2021
19. April 2021
Übungsblatt 1

Diese Übung wird am **22.04.2020** um **13:30 Uhr** besprochen und **nicht** bewertet.

Benötigte Dateien
Alle benötigten Datensätze und Skriptvorlagen finden Sie in unserem Moodle-Kurs:
https://moodle.informatik.tu-darmstadt.de/course/view.php?id=1058

1.1 Einführung in Python

In der Übung zu Data Mining und Maschinelles Lernen werden wir für Programmieraufgaben die Skriptsprache Python (Version ≥ 3.5) verwenden. Ein schneller Weg, Python mitsamt nützlichen Standardbibliotheken zu installieren, ist die Anaconda Plattform (Open-Source Distribution):

- https://www.anaconda.com/products/individual#Downloads

Anaconda beinhaltet nach Auswahl ebenfalls die interaktiven Entwicklungsumgebungen *spyder* sowie *jupyter*.

Alternativ können Sie Python auch direkt installieren:

- https://www.python.org/downloads/

und ggf. die Entwicklungsumgebung *Pycharm* (Community Edition) kostenfrei herunterladen:

- https://www.jetbrains.com/pycharm/download/

Zusätzliche Programmbibliotheken installieren Sie in Python über den Paketinstaller *pip*. Rufen Sie hierfür eine Konsole auf und ersetzen `Paket-Name` mit dem jeweiligen Paketnamen.

```
$ pip3 install <Paket-Name>
```

a) **Installation**
Installieren Sie die Programmbibliotheken *numpy*, *matplotlib* und *scikit-learn*.

b) **Einstieg in Python**
Zu Beginn der ersten Übungsstunde wird eine kurze Einführung in Python und *numpy* stattfinden.
Wenn Sie mit der Skriptsprache Python noch nicht vertraut sind, empfehlen wir Ihnen das Jupyter-Notebook `python_intro.ipynb` zu untersuchen oder der Einführung auf
https://cs231n.github.io/python-numpy-tutorial/
zu folgen.

1.2 Überblick zum maschinellen Lernen

Das maschinelle Lernen (engl. Machine Learning) kommt in vielen Gebieten des täglichen Lebens zum Einsatz.

Im Setup des **Supervised Learning** (dt. überwachten maschinellen Lernens) gibt es einen Datensatz $D = (\mathbf{X}, \mathbf{y})$, mit Datenpunkten $\mathbf{x}_i \in \mathbf{X}$ und einer entsprechenden Zielvariablen $y_i \in \mathbf{y}$. Überwachte Algorithmen des maschinellen

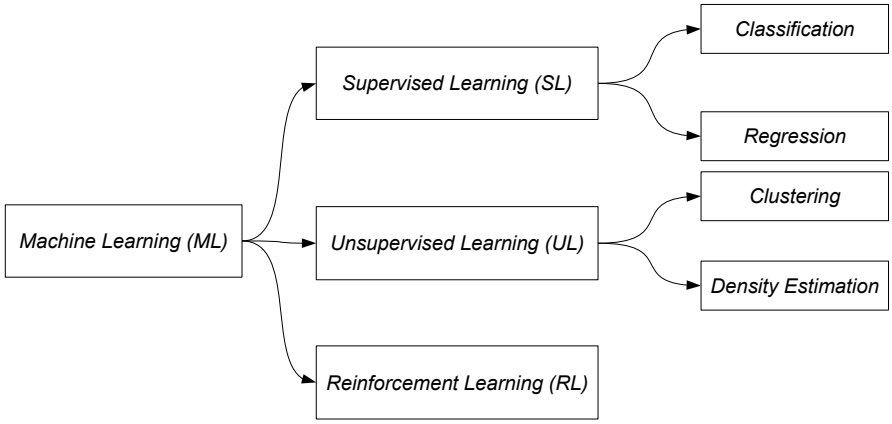

Abbildung 1: Übersicht einer groben Hierarchie der Begriffe zum maschinellen Lernen.

Lernens zielen darauf ab, ein Modell f zu finden, welches $f(\mathbf{x_i}) \approx y_i$ für alle Datenpunkte annähert. Dieses Modell kann anschließend verwendet werden, um den Wert der Zielvariablen für ungesehene Daten vorherzusagen. Wenn die Zielvariable kategorisch ist, z.B. $y_i \in \{Katze, Hund\}$, wird die Aufgabe **Klassifikation** genannt. Wenn die Zielvariable kontinuierlich ist, z.B. $y_i \in \mathbb{R}$, wird die Aufgabe **Regression** genannt.

Auf der anderen Seite gibt es auch das sogenannte **Unsupervised Learning** (dt. unüberwachte Lernen), bei dem es keine Zielvariable gibt. In diesem Fall interessieren wir uns für Aufgaben wie **Clustering**, d.h. das Finden von Gruppen in Datenpunkten, oder die **Dichteschätzung**, die versucht, die Wahrscheinlichkeitsverteilungen der Eingangsvariablen zu modellieren.

Weiterhin gibt es den Begriff des **Reinforcement Learning** (dt. selbstverstärkendes Lernen), bei welchem ein Agent eine Strategie erlernt, um eine Belohnung zu maximieren. Reinforcement Learning wird innerhalb dieser Vorlesung nicht näher behandelt.

a) **Beispiele**

Nennen Sie drei Gebiete mit je einem Anwendungsbeispiel, bei welchem maschinelles Lernen genutzt wird.

Gesundheitswesen: *Vorhersage, ob ein Patient, der wegen eines Herzinfarktes ins Krankenhaus eingeliefert wurde, einen zweiten Herzinfarkt erleiden wird. Die Vorhersage soll auf demographischen und klinischen Messungen für diesen Patienten basieren.*

Gesundheitswesen: *Schätzen der Menge an Glukose im Blut eines Diabetikers anhand des Infrarot-Absorptionsspektrums des Blutes dieser Person.*

Gesundheitswesen: *Identifikation der Risikofaktoren für Prostatakrebs, basierend auf klinischen und demographischen Merkmalen.*

Finanzwesen: *Prognostizieren eines Aktienkurses in 6 Monaten auf der Grundlage von Leistungskennzahlen und Wirtschaftsdaten des Unternehmens.*

Postwesen: *Identifikation der Nummern in einer handgeschriebenen Postleitzahl aus einem digitalisierten Bild.*

b) **Einordnung**

Ordnen Sie die Begriffe **Klassifikation**, **Unsupervised Learning (UL)**, **Dichteschätzung**, **Reinforcement Learning (RL)**, **maschinelles Lernen (ML)**, **Regression**, **Supervised Learning (SL)**, **Clustering** in der Abbildung 1 ein.

1.3 Klassifikation

Bei einer Klassifizierungsaufgabe (mit einer Zielvariable) stammt die Zielvariable aus einer Menge von diskreten Werten (Klassen): $y_i \in \{c_0, \ldots, c_{I-1}\}$. Jeder Datenpunkt $\mathbf{x_i} \in \mathbf{X}$ ist mit seinem entsprechenden Klassenwert y_i assoziiert.

Es gibt zwei Hauptansätze für das Klassifizierungsproblem:

1. **Entscheidungsgrenzen**: Bei dieser Methode wird versucht, die Daten in Regionen zu trennen, in denen die Datenpunkte zur gleichen Klasse gehören, wenn sie in der gleichen Region liegen, und in verschiedene Klassen, wenn sie in verschiedenen Regionen liegen (in den obigen Beispielen verwendeter Ansatz).
2. **Bayes'sche Entscheidungstheorie**: Modellierung der A-Posteriori-Wahrscheinlichkeit, dass ein Datenpunkt $\mathbf{x_i}$ zur Klasse C_k gehört:

$$P\left(C_k|\mathbf{x_i}\right) = \frac{P\left(\mathbf{x_i}|C_k\right) P\left(C_k\right)}{P\left(\mathbf{x_i}\right)}. \tag{1}$$

   Eine Entscheidungsregel kann man erhalten werden, indem man die Klasse findet, die die maximale Wahrscheinlichkeit für den gegebenen Datenpunkt x_i ergibt:

$$C_i = \text{argmax}_k P\left(C_k|\mathbf{x_i}\right). \tag{2}$$

Das Hauptziel des maschinellen Lernens besteht darin, Vorhersagefunktionen zu finden, die gut zu den Daten passen und weiter auf ungesehene Daten verallgemeinern. Da die handliche Herstellung dieser Funktionen mühsam ist (man nennt dies **Expertensysteme**), geht es beim maschinellen Lernen darum, die Vorhersagefunktionen aus den Daten zu erlernen. Das bedeutet, dass ein Modell des maschinellen Lernens die Daten durchsieht und einen eigenen Satz von Regeln entwickelt, wie man y_i erhält, wenn man den Datenpunkt $\mathbf{x_i}$ angibt.

a) **Datensatzerstellung**

Erstellen Sie einen synthetischen Datensatz zweier gaußverteilten Cluster. Der Datensatz soll dabei insgesamt 500 Stichproben besitzen und die Clusterzentren in den Punkten $(-1.5, -1.5)$ und $(1.5, 1.5)$ liegen.

Sie können hierfür die Funktion `sklearn.datasets.make_blobs` verwenden.

```python
def create_blob_dataset() -> [np.ndarray, np.ndarray]:
    """Create blobs of independent Gaussian distributions with centers at (-1.5,-1.5) and (1.5,1.5)
    """
    return datasets.make_blobs(
        n_samples=500, random_state=0, centers=[[-1.5, -1.5], [1.5, 1.5]]
    )
```

b) **Einfache Vorhersage**

Unser Ziel ist es zu bestimmen, ob ein Datenpunkt der Klasse A ($\hat{y} = 0$) oder B ($\hat{y} = 1$) angehört. Aus der Betrachtung der Daten könnten wir einige handgeschriebene Regeln ableiten. Weisen Sie einen Datenpunkt der Klasse A in der Funktion `predict_1(x)` zu, wenn er sich im dritten Quadranten befindet. Alle anderen Punkte werden der Klasse B zugewiesen.

```python
def predict_1(x: np.ndarray) -> int:
    """Predict the class based on wheter it is in the third quardrant."""
    if x[0] < 0 and x[1] < 0:
        return 0  # Class A
    else:
        return 1  # Class B
```

c) **Genauigkeit**

Implementieren Sie die Funktion `accuracy()`, um die Genauigkeit unserer Vorhersagefunktion zu bestimmen.

```python
def accuracy(y: np.ndarray, y_pred: np.ndarray) -> float:
    """Calculate the accuracy of the prediction y_pred w.r.t. the true labels y."""
    num_correct = 0
    num_total = y.shape[0]
    for y_i, y_i_pred in zip(y, y_pred):
        # Count correct predictions
        if y_i == y_i_pred:
            num_correct += 1
```

```
    # Calculate accuracy
    return num_correct / num_total * 100
```

```
def accuracy2(y: np.ndarray, y_pred: np.ndarray) -> float:
    """Calculate the accuracy of the prediction y_pred w.r.t. the true labels y."""
    return (y == y_pred).sum() / len(y) * 100
```

d) **Verbesserte Vorhersage**

Wir können nun die Vorhersagefunktion verfeinern und z.B. sagen, dass alles unterhalb der Diagonalen ($-x_0 = x_1$) zur Klasse A gehört und sonst zur Klasse B. Implementieren Sie diese Regel in der Funktion `predict_2(x)`.

```
def predict_2(x: np.ndarray) -> int:
    """Predict the class based on whether it is below the line -x_0 = x_1."""
    if -x[0] > x[1]:
        return 0  # Class A
    else:
        return 1  # Class B
```

e) **Grenzpunkte**

Diese neue Vorhersagefunktion funktioniert besser als die erste, da sie der Grenze zwischen den beiden Klassenclustern näher kommt.

Im folgenden Beispiel können Sie versuchen, die Entscheidungsgrenze selbst festzulegen und die Datenpunkte in den beiden Monddatensätzen zu klassifizieren:

Passen Sie die Grenzwerte in der Funktion `get_boundaries()` an, um eine Genauigkeit $\geq 75\%$ zu erzielen.

```
def get_boundaries() -> list:
    """Returns a list of boundary points e.g.
    [[-5, 0],[-2, 2],[-1,1],[1, 0.9],[4, 2],]
    """
    return [[-5, 0],
            [-0.8, -0.5],
            [-0.1, 0.8],
            [1.2, -0.3],
            [5, 5]]
```

1.4 Regression

Bei einer Regressionsaufgabe (mit einer Zielvariable) ist die Zielvariable kontinuierlich ($y_i \in \mathbb{R}$). Dies unterscheidet sich sehr von der früheren Klassifizierungsstruktur, da wir jetzt nicht einfach unsere Daten in Klassen trennen und Entscheidungsgrenzen finden können, um die Klassen zu trennen.

Ein einfaches Beispiel ist die lineare Regression, bei der wir annehmen, dass die Daten $\mathbf{x_i}$ linear mit der Zielvariablen y_i korreliert sind, wobei wir Koeffizienten $\beta_0, \ldots, \beta_K \in \mathbb{R}$ verwenden:

$$y_i = \beta_0 + \beta_1 x_i^1 + \beta_2 x_i^2 + \cdots + \beta_K x_i^K \tag{3}$$

und der Superskript den jeweiligen Eingangsvariablen-Index beschreibt.

Für den zweidimensionalen Fall reduziert sich dies auf den klassischen Fall $y_i = b + m x_i$ (mit $b = c_0$ von oben).

Die Regressionskoeffizienten können über folgende Formal bestimmt werden:

$$\hat{\beta} = \left(\mathbf{X'}^T \mathbf{X'}\right)^{-1} \mathbf{X'}^T \mathbf{y}\,, \tag{4}$$

wobei $\mathbf{X}$ um eine Spalte mit Einsen ergänzt wird ($\mathbf{X'} = [\mathbf{1}, \mathbf{X}]$), um die Verschiebung an der y-Achse β_0 zu berücksichtigen.

Neue Vorhersagen können wir nun wie folgt erhalten:

$$\hat{\mathbf{y}} = \mathbf{X}'\hat{\beta}\,. \tag{5}$$

Zur Bewertung, wie sehr die ermittelte Regressionskurve den Datensatz annähert, wird häufig die mittlere quadratische Abweichung ermittelt:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}\left(\mathbf{y_i} - \hat{\mathbf{y}}_\mathbf{i}\right)^2\,, \tag{6}$$

wobei n der Anzahl der Datenpunkten entspricht.

a) **Datensatzvisualisierung**

Visualisieren Sie in der Methode `visualize_data()` den synthetisch generierten Datensatz als Scatterplot. Verwenden Sie dabei die *matplotlib* Bibliothek und beschriften Sie die x- und y-Achse.

```python
def visualize_data(X: np.ndarray, y: np.ndarray) -> None:
    """Visualizes the data points in a scatter plot."""
    plt.scatter(X, y, label="Data")
    plt.xlabel("$x$")
    plt.ylabel("$y$")
    plt.legend()
    plt.show()
```

b) **Regressionsermittlung**

Berechnen Sie in `perform_linear_regression()` die Regressionskoeffizienten $\hat{\beta}$ anhand von (4) an.

```python
def perform_linear_regression(Xp: np.ndarray, y: np.ndarray) -> np.ndarray:
    """Computes the regression coefficient beta_hat."""
    return np.linalg.inv(Xp.T @ Xp) @ Xp.T @ y
```

c) **Vorhersage**

Geben Sie in `compute_predictions()` für jeden Datenpunkt $\mathbf{x_i}$ den zugehörigen Punkt auf der ermittelten Regressionsline $\hat{\mathbf{y}}_\mathbf{i}$ an. Nutzen Sie hierfür (5) und die zuvor bestimmten Koeffizienten $\hat{\beta}$.

```python
def compute_predictions(Xp: np.ndarray, beta_hat: np.ndarray) -> np.ndarray:
    """Computes the corresponding prediction on the regression line for Xp."""
    return Xp @ beta_hat
```

d) **Fehlerbewertung**

Berechnen Sie in der Funktion `compute_mse()` die mittlere quadratische Abweichung (6) für die ermittelte Funktion zu den Datenpunkten.

```python
def compute_mse(y: np.ndarray, y_pred: np.ndarray) -> float:
    """Computes the mean squared error."""
    return float(np.mean((y - y_pred) ** 2))
```

```python
def compute_mse2(y: np.ndarray, y_pred: np.ndarray) -> float:
    """Computes the mean squared error."""
    return 1/len(y) * float(np.sum((y - y_pred) ** 2))
```