II Übung zur Vorlesung Deep Learning

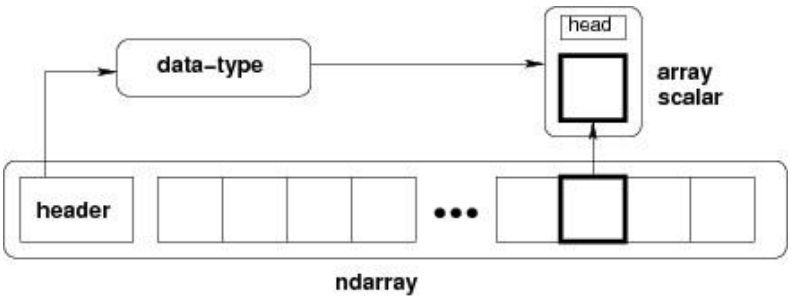
Einführung in NumPy und Scikit-Learn und Matplotlib

Einführung in NumPy

NumPy (Numerical Python) ist ein Python-Paket/eine Bibliothek, die aus mehrdimensionalen Array-Objekten und einer Sammlung von Routinen zur Verarbeitung von Arrays besteht. NumPy bietet unter anderem Operationen zur mathematischen und logischen Handhabe von auf Arrays, Fourier-Transformationen und Routinen zur Formmanipulation, Operationen im Zusammenhang mit linearer Algebra und Zufallszahlengenerierung. NumPy ist eng verwandt mit dem Deep Learning Framework PyTorch und ein Kernpaket für das Entwickeln und Trainieren von Deep Neural Networks.

NumPy Arrays

Das wichtigste in NumPy definierte Objekt ist ein N-dimensionaler Array-Typ namens ndarray. Es beschreibt die Sammlung von Elementen des gleichen Typs. Auf Elemente in der Sammlung kann über einen Null-basierten Index zugegriffen werden. Jedes Element in einem ndarray nimmt die gleiche Größe des Blocks im Speicher an. Jedes Element in ndarray ist ein Objekt vom Datentyp Objekt (genannt dtype). Jedes Element, das aus dem ndarray-Objekt extrahiert wird (durch Slicing), wird durch ein Python-Objekt eines der skalaren Array-Typen dargestellt. Das folgende Diagramm zeigt eine Beziehung zwischen ndarray, Datentyp-Objekt (dtype) und Array-Skalartyp.



Im Folgenden findest du einige Beispiele für das Erstellen und das Nutzen von NumPy Arrays.

Arrays erstellen

```
import numpy as np

a = np.array([1, 2, 3])
b = np.array([[1, 2], [3, 4]])
c = np.array([1, 2, 3, 4, 5], ndmin=2)
d = np.array([1, 2, 3], dtype=complex)

print(a)
>>> [1  2  3]

print(b)
>>> [[1  2]
     [3  4]]

print(c)
>>> [[1  2  3  4  5]]

print(d)
>>> [1.+0.j  2.+0.j  3.+0.j]
```

```
import numpy as np

a = np.zeros(4)
b = np.ones((3, 2))
c = np.zeros((5,), dtype=np.int)
d = np.zeros((2,2), dtype=[('x', 'i4'), ('y', 'i4')])

print(a)
>>> [0.  0.  0.  0.]

print(b)
>>> [[1. 1.]
     [1. 1.]
     [1. 1.]]

print(c)
>>> [0  0  0  0  0]

print(d)
>>> [[(0, 0) (0, 0)]
     [(0, 0) (0, 0)]]

a = np.arange(5)
b = np.arange(10, 20, 2)  # (von, bis, Schrittweite)
c = np.linspace(10, 20, 5, endpoint=True)  # (von, bis, Anzahl)

print(a)
>>> [1  2  3  4  5]

print(b)
>>> [10  12  14  16  18]

print(c)
>>> [10.  12.5  15.  17.5  20.]
```

In Arrays umwandeln

```
import numpy as np

x = [1, 2, 3]
b = np.asarray(x, dtype=float)

print(b)
>>> [1.  2.  3.]

x = (1, 2, 3)
c = np.asarray(x)
print(c)
>>> [1  2  3]

x = [(1, 2, 3) (4, 5)]
d = np.asarray(x)

print(d)
>>> [(1, 2, 3)  (4, 5)]
```

Arrays indexieren und slicen

Auf den Inhalt des ndarray-Objekts kann durch Indizieren oder Slicen zugegriffen werden. Wie bereits erwähnt, folgen Elemente im ndarray-Objekt dem nullbasierten Index. Es stehen drei Arten von Indexierungsmethoden zur Verfügung - Feldzugriff, einfaches Slicing und erweiterte Indexierung. Basic Slicing ist eine Erweiterung von Pythons Grundkonzept des Schneidens auf n Dimensionen. Ein Python-Slice-Objekt wird konstruiert, indem der integrierten Slice-Funktion Start-, Stopp- und Step-Parameter übergeben werden. Dieses Slice-Objekt wird an das Array übergeben, um einen Teil des Arrays zu extrahieren. Es gibt zwei Arten der erweiterten Indizierung - **Integer** und **Boolean**.

Beispiele:

```
import numpy as np

a = np.arange(10)
s = slice(2, 7, 2)

print(a[s])
>>> [2  4  6]

a = np.arange(10)
b = a[2:7:2]

print(b)
>>> [2  4  6]

a = np.arange(10)

print(a[2:])
>>> [2  3  4  5  6  7  8  9]

a = np.array([[1, 2, 3], [3, 4, 5], [4, 5, 6]])

print(a[1:])
>>> [[3 4 5]
     [4 5 6]]
```

Es ist möglich, eine Auswahl aus einem Array zu treffen, die eine Nicht-Tupel-Sequenz, ein ndarray-Objekt vom ganzzahligen oder booleschen Datentyp oder ein Tupel mit mindestens einem Element als Sequenzobjekt ist. Die erweiterte Indexierung gibt immer eine Kopie der Daten zurück. Im Gegensatz dazu stellt das Slicing nur eine Ansicht dar.

Beispiele:

```
import numpy as np

x = np.array([[1, 2], [3, 4], [5, 6]])
y = x[[0, 1, 2], [0, 1, 0]]

print(y)
>>> [1  4  5]

x = np.array([[0,  1,  2], [3,  4,  5], [6,  7,  8], [9, 10, 11]])
rows = np.array([[0, 0], [3, 3]])
cols = np.array([[0, 2], [0, 2]])
y = x[rows, cols]
```

```
print(y)
>>> [[0  2]
     [9 11]]

y = x[1:4, [1, 2]]

print(y)
>>> [[4  5]
     [7  8]
     [10 11]]

print(x[x > 5])
>>> [6  7  8  9  10  11]
```

Im folgenden Beispiel werden NaN-Elemente (keine Zahl) durch die Verwendung von ~ (Komplementoperator) weggelassen.

```
a = np.array([np.nan, 1, 2, np.nan, 3, 4, 5])

print(a[~np.isnan(a)])
>>> [ 1.   2.   3.   4.   5.]
```

NumPy bietet viele weitere Möglichkeiten Daten in Arrays zu handhaben, wie zum Beispiel das Iterieren über Arraystrukturen, das Manipulieren der Form und der Dimensionalitäten einzelner Arrays, das Zusammenführen mehrere oder das Aufteilen einzelner Arrays und das Hinzufügen oder Löschen einzelner Elemente eines Arrays. Darüber hinaus bietet NumPy die Anwendung binärer und arithmetischer Operatoren, mathematischer Funktionen, wie beispielsweise Sinus und Cosinus, oder einfache statistische Funktionen wie das Ermitteln von Mittelwerten, Varianzen, usw. Mehr zu NumPy und den zusätzlichen Funktionalitäten findest du unter https://www.tutorialspoint.com/numpy/index.htm.

Übungsaufgaben – Einführung in Scikit-Learn und Matplotlib

In der vorliegenden Übung soll das aus der Vorlesung bekannte Iris Flower dataset analysiert werden. Dabei werden zwei Bibliotheken (Scikit-Learn und Matplotlib) eingeführt, die beim Erstellen und Trainieren der Lernmodelle und bei der Visualisierung der Ergebnisse helfen. Die Dokumentation beider Bibliotheken findest du unter https://scikit-learn.org/stable/documentation.html und https://matplotlib.org/contents.html.

Aufgabe 1 – Iris Flower Dataset

a) Lade den Iris Flower Datensatz und mache dich mit seinen Eigenschaften vertraut. Der Datensatz ist in Scikit-Learn verfügbar.

```
from sklearn import datasets
iris = datasets.load_iris()
print(iris.keys())
```

Untersuche das Objekt, welches die Daten beinhaltet und beantworte folgende Fragen mit ein paar Zeilen Code:

- Wie viele Klassen gibt es?
- Welche Namen haben die Klassen?
- Wie viele Datenpunkte pro Klasse gibt es?
- Wie viele Features gibt es?
- Welche Namen haben die Feature?

- Wie sind die einzelnen Features ausgeprägt? (Minimalwert, Maximalwert, Mittelwert, Standardabweichung)

b) Visualisiere die Ausprägungen der Features in einem Histogramm. Benutze für die Visualisierung Matplotlib. Das folgende Minimalbeispiel zeigt die Nutzung der Funktion.

```
import numpy as np
import matplotlib.pyplot as plt

data = np.random.normal(0, 1, 1000)
plt.hist(data)
plt.show()
```

c) Visualisiere die Korrelationen zwischen den einzelnen Features in einem Scatterplot. Trage jedes Feature gegen jedes auf und verschaffe dir einen Überblick, welche Features für eine Vorhersage ausschlaggebend sein könnten. Benutze für die Visualisierung wieder Matplotlib. Das folgende Minimalbeispiel zeigt die Nutzung der Scatterplot Funktion.

```
import numpy as np
import matplotlib.pyplot as plt

data = np.random.normal(0, 1, (100, 2))
plt.scatter(data[:, 0], data[:, 1])
plt.show()
```

Aufgabe 2 – Vorhersagen für neue Iris Flowers

a) Bereite die Daten vor, um ein Klassifizierungsmodell trainieren und evaluieren zu können. Reduziere zunächst die Anzahl an Features von 4 auf 2, sodass die Ergebnisse später leicht visualisiert werden können. Führe dazu eine Principal Component Analysis durch und berücksichtige für die weitere Analyse nur die ersten beiden Principal Components. PCA ist direkt in Scikit-Learn verfügbar. Das folgende Minimalbeispiel zeigt die Nutzung der Funktion.

```
import numpy as np
from sklearn.decomposition import PCA

X = np.random.normal(42, 2.0, (100, 7))
pca = PCA(n_components=2)
X_transformed = pca.fit_transform(X)
```

b) Nach der Transformation, teile die Daten in einen Trainings- und Testdatensatz auf. Verwende dafür die in Scikit-Learn implementierte Funktion. Das folgende Minimalbeispiel zeigt die Nutzung der Funktion.

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
```

c) Erstellen ein K-Nearest-Neighbor Klassifizierungsmodell und trainiere das Modell mit den Trainingsdaten. Validiere die Modellgenauigkeit anhand der Testdaten. Variiere die Anzahl an nächsten Nachbarn und finde heraus, für welchen Parameter das Modell am besten funktioniert. Das folgende Minimalbeispiel zeigt die Erstellung, das Training und das Testen des Modells.

```
from sklearn.neighbors import KNeighborsClassifier as KNN
from sklearn.model_selection import cross_validate
```

```
clf = KNN(n_neighbors=3)
clf.fit(X_train, y_train)
score = clf.score(X_test, y_test)
scores_cv = cross_validate(clf, iris.data, iris.target, cv=5)
```

d) Mache eine Vorhersage für einen Satz von neuen Iris Flowers. Erstelle zunächst einen kleinen Datensatz an neuen Datenpunkten mit vier Features, z.B.

```
X_new = np.array([[5.83, 3.15, 3.99, 1.01],
                  [5.79, 2.89, 3.73, 1.53],
                  [5.42, 3.75, 3.71, 1.23],
                  [5.46, 3.23, 3.35, 0.91]])
```

Bedenke, dass dein Modell im reduzierten feature space trainiert wurde. Reduziere zunächst deine neuen Datenpunkte mit dem bereits trainierten PCA Modell:

```
X_new_trans = pca.transform(X_new)
```

Mache nun die Vorhersage für deine neuen Datenpunkte:

```
y_new = clf.predict(X_new_trans)
```

Welche der drei Klassen hast du beim Erstellen der neuen Datenpunkte gewählt?

e) Visualisiere das Ergebnis des KNN Klassifizierers, sowie die Vorhersage für die neuen Datenpunkte. Nutze für die Visualisierung die folgende vorgegebene Funktion:

```
def visualize(model, X, y, X_new=None, y_new=None):
    # create a mesh grid
    h = 0.02
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))

    # save the decision boundary of the model.
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    # add some random noise to the data for plotting purposes only so that
      overlapping point can be identified.
    noise = np.random.normal(0, 0.02, X.shape)
    X = X + noise

    # create a custom colormap that maps a specific color to each flower
      class
    cmap = plt.cm.coolwarm
    colors = cmap(np.linspace(0, 1, 3, endpoint=True))

    # set the size of the figure
    plt.rcParams["figure.figsize"] = (10, 6)

    # plot the decision boundary of the model in color code
    plt.contourf(xx, yy, Z, cmap=cmap, alpha=0.5)

    # plot the data points for each flower class
    for i_class in np.unique(y):
        plt.scatter(X[:, 0][y == i_class], X[:, 1][y == i_class],
                    s=64, c=colors[i_class], edgecolor='k',
                    cmap=plt.cm.coolwarm, alpha=0.7,
                    label=iris['target_names'][i_class])

    if X_new is not None and y_new is not None:
        for i_class in np.unique(y_new):
            plt.scatter(X_new[:, 0][y_new == i_class],
```

```
                    X_new[:, 1][y_new == i_class],
                    marker='^', s=96, c=colors[i_class],
                    edgecolor='k', cmap=plt.cm.coolwarm, alpha=0.7)

# set plotting parameters such as axis labels, ranges,
  a title and the legend.
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.title('KNN Classification')
plt.legend()

# show the plot
plt.show()
```