

NNW-Übung 2

Bemerkungen zu Spyder

Im Editor können Sie (in Analogie zu MATLAB) mit der Kommentarzeile `%%` Blöcke („cells“) definieren, die einzeln abgearbeitet werden können: Start über die beiden Icons *neben* dem grünen Pfeil oder **Strg-Enter** (aktuellen Block ausführen) bzw. **Shift-Enter** (aktuellen Block ausführen und zum nächsten springen). Markierte Zeilen können durch Drücken von **F9** ausgeführt werden.

1 Supervised Learning 1: Klassifikation

Ein sehr umfangreiches und weit verbreitetes Machine Learning Paket in Python ist *scikit-learn* (scikit-learn.org, Package-Name: `sklearn`). Die Webseite von *scikit-learn* enthält zudem (in der Mehrzahl) sehr gute Einführungen in verschiedene Themen des Maschinellen Lernens mit vielen Code-Beispielen. Nur Neuronale Netze sind *nicht* die Stärke von *scikit-learn* – es geht zwar, aber nur für einfache Beispiele.¹ Es gibt aber Wrapper für z.B. PyTorch (`skorch`) und Keras (`scikeras`).

Algorithmen in *scikit-learn* sind als Klassen realisiert. Deswegen legt man typischerweise als Erstes ein Objekt der Klasse des gewünschten Algorithmus an, zum Beispiel für eine lineare *Support Vector Machine* (SVM) von der Klasse `sklearn.svm.LinearSVC` (Linear Support Vector Classification):

```
from sklearn.svm import LinearSVC # andere Art des imports als bei numpy
clf = LinearSVC()
```

Die meisten Algorithmen, insbesondere Lernverfahren, haben eine Methode `fit` zum Anpassen an die Daten (also Lernen anhand der Daten `X`). `fit` hat mindestens einen Parameter `X` für die Input-Daten, bei supervised Lernverfahren zusätzlich (mindestens) noch einen weiteren Parameter `y` für die zugehörigen Zielwerte.

- In der Vorlesung wird für die Zielwerte der Bezeichner `T` für *targets* verwendet.
- `y` steht in der Vorlesung für die Ausgabe eines Algorithmus – die ja, je nach Lernzustand/-fortschritt/-erfolg, nicht unbedingt identisch mit dem gewünschten Zielwert `T` sein wird.

Das eigentliche Lernen/Trainieren besteht also aus nur einer Zeile:

```
clf.fit(X, T)
```

Um das Gelernte auf (neue) Daten anzuwenden, gibt es die Methode `predict(X)`, die die Ergebnisse, die der trainierte Algorithmus zu den Inputs `X` berechnet, zurückliefert:

```
y = clf.predict(X) # Hier bei Anwendung auf die Trainingsdaten -
                  # spannender wären natürlich neue Daten.
```

Die Daten können (bzw. beim Training: müssen) mehrere Beobachtungen/Beispiele enthalten. Für *scikit-learn* (und später auch für TensorFlow/Keras, PyTorch, JAX/Flax) gilt:

- Für jede Beobachtung/Beispiel (sample) gibt es eine *Zeile* (numpy-Array-Dimension 0).
- Die Merkmale (features) `hat` stehen in den Spalten (numpy-Array-Dimension 1).² Die Merkmalsvektoren je Sample sind also *Zeilenvektoren*!

¹Die (wiedermal ziemlich gute) Einführungsseite von *scikit-learn* zu Neuronalen Netzen startet entsprechend mit einem Warnhinweis: „This implementation is not intended for large-scale applications.“ Quelle: scikit-learn.org/stable/modules/neural_networks_supervised.html.

²Genaugenommen: Wenn es nicht noch weitere, z.B. bei Bildern räumliche Dimensionen oder bei Zeitreihen zeitliche Dimensionen gibt. Bei *scikit-learn* gibt es das nicht, aber bei den Deep-Learning Frameworks. Dazu kommen wir dann später in der Vorlesung.

- a) Lassen Sie den Iris-Datensatz einlesen – siehe das erste Aufgabenblatt. Aber Achtung: Stehen die features und samples in der richtigen Dimension? Entsprechen die Dimensionen denen, die, wie auf dem ersten Aufgabenblatt angekündigt, in der Vorlesung verwendet werden? Sprechen Sie mich darauf an.
- b) Trainieren Sie wie oben beschrieben eine lineare SVM auf den Iris-Daten: Verwenden Sie nur die *ersten beiden Merkmale* (so wie auf dem ersten Aufgabenblatt). Es kann sein, dass Sie eine Warnung erhalten – diese ist aber nicht weiter schlimm, sondern sogar für dieses Aufgabenblatt nicht unerwünscht.
- c) Lassen Sie die Trainingsdaten vorhersagen und schauen Sie sich das Ergebnis an.
- d) Überlegen Sie, wie Sie mit wenigen **numpy**-Berechnungen zählen können (keine Spezialfunktionen aus **sklearn** verwenden!), wie oft das Ergebnis richtig ist. (Die benötigten Funktionen haben Sie auf dem letzten Aufgabenblatt kennengelernt, aber in anderem Zusammenhang, eine beginnt mit `uni...`)
- e) Nun soll das Ergebnis geplottet werden – aber nicht nur die Daten selbst, sondern welche Bereiche des *Merkmalsraums*³ von dem trainierten Klassifikator zu welcher Klasse zugeordnet werden.
Auf der Moodle-Seite finden Sie die Datei `nnwplot.py`. Importieren Sie diese mit `import nnwplot` (die Datei in dasselbe Verzeichnis kopieren wie Ihr Skript) und rufen Sie die darin enthaltenen Funktion `nnwplot.plotTwoFeatures(X, T, clf.predict)` auf, wobei `X` wie beschrieben nur zwei Merkmale je Blüte (aber alle Blüten!) enthalten darf.
Beachten Sie: Es wird tatsächlich `clf.predict` übergeben – *ohne* Klammern! Das ist *kein* Aufruf der Methode `predict`, sondern die Methode selbst wird übergeben! Wenn Sie aus der Java-Welt kommen, ist das sehr ungewöhnlich (oder?). Es entspricht in Java in etwa der Übergabe eines Objekts – in der Tat ist in Python (fast) alles ein Objekt, auch eine Methode wie `clf.predict`. Probieren Sie: `isinstance(clf.predict, object)` – `object` ist in Python (wie in Java `Object`) die oberste Klasse, von der alle Klassen (direkt oder meist indirekt) erben.
Und warum wird `clf.predict` als Methode (bzw. Funktion) an `nnwplot.plotTwoFeatures` übergeben? `nnwplot.plotTwoFeatures` ruft die übergebene Funktion für gleichmäßig über den Plot-Bereich verteilte Punkte, insgesamt 300×300 Punkte, auf um diese dann entsprechend der vorhergesagten Klasse einfärben zu können.
- f) Lassen Sie das Ganze noch für eine (nicht-linearen) SVC ablaufen – die Klasse heißt einfach `SVC`.
- g) Wie unterscheiden sich die Plots von linearem und nicht-linearem SVC?
- h) Versuchen Sie es auch noch mit den Merkmalen 0 und 3 (für beide Klassifikatoren). Wie wählen Sie aus den vier Merkmalen diese beiden (0 und 3) in *einer* Python-Zeile aus? Falls Sie das nicht hinkriegen, fragen Sie mich!
- i) Probieren Sie als weiteren Algorithmus `sklearn.neighbors.KNeighborsClassifier`.

2 Unsupervised Learning: Clustering

Clustering ist eine Form von Unsupervised Learning: Als Daten werden nur die Merkmale der Beobachtungen/Beispiele (samples) gebraucht (also `X`), *keine* Zielwerte (also *kein* `T`), daher spricht man auch von *unlabeled data*. Ziel bei Clustering ist Gruppen (die Cluster) von ähnlichen Daten zu finden.

In *scikit-learn* sind die Algorithmen wieder als Klassen realisiert (zusätzlich gibt es Funktionen). Das Unterpackage für Clustering-Algorithmen heißt `sklearn.cluster`. Das „Training“ ist dabei vollkommen identisch zur vorigen Teilaufgabe – nur das bei dem Aufruf von `fit` kein `y` (bzw. `T`) übergeben werden muss (man kann es sogar übergeben, es wird dann aber ignoriert). Die Abfrage des Ergebnisses, also die Zuordnung der Daten zu (den durchnummerierten) Clustern, gilt:

³Den Begriff haben Sie in der Vorlesung kennengelernt.

- Das Ergebnis der Zuordnung steht in der Variablen `clustering.labels_`, wobei `clustering` das von Ihnen angelegte Objekt des Verfahrens sein soll. Zusätzlich gibt es die Methode `fit_predict(X)`, die quasi `fit` und `predict` auf den (Trainings-)Daten `X` durchführt – nur dass das nicht hintereinander, sondern gleichzeitig passiert.
- Manche Algorithmen können aus prinzipiellen Gründen *nicht* auf anderen Daten als den Trainingsdaten angewendet werden. Daher gibt es für diese auch keine Methode `predict(X)` geben, denn hier könnten für `X` ja irgendwelche Daten eingesetzt werden. Andere Algorithmen können auch auf andere Daten als den Trainingsdaten verwendet werden, diese bieten dann eine Methode `predict(X)` an – völlig analog zu den Verfahren für Supervised Learning.

Verwenden Sie die **Merkmale 1 und 3** (ab 0 gezählt) des Iris-Datensatzes.

- Probieren Sie das Verfahren **KMeans** auf den Iris-Daten aus, in dem Sie das Skript der vorherigen Aufgabe zuerst kopieren und dann anpassen. k steht dabei für die Anzahl der zu suchenden Cluster. Versuchen Sie verschiedene Werte für k . In der Aufgabe 1.d haben Sie die Anzahl der „richtigen“ Klassifikationen berechnen lassen. Hat diese hier eine Bedeutung? Wenn ihr(e) Nachbar(in) andere Ergebnisse bekommt als Sie ... gehen siehe zur nächsten Teilaufgabe über.
- Lassen Sie das Skript mehrfach ablaufen. Was fällt auf? Wenn Sie wissen wollen, warum (oder auch, wie das Verfahren überhaupt funktioniert), fragen Sie mich oder lesen Sie über die Initialisierung des Verfahrens nach.
- Hatten die Farben der geplotteten Punkte für das Training des Verfahrens eine Bedeutung?
- Probieren Sie als weiteres Verfahren DBSCAN. Nicht wundern, Sie bekommen eine Fehlermeldung. Warum?

Optional: Der Fehler lässt sich umgehen, wenn Sie folgenden Code verwenden:

```
from sklearn.neighbors import KNeighborsClassifier
clustering = DBSCAN()
y = clustering.fit_predict(X)

clf = KNeighborsClassifier(1)
clf.fit(X, y) # Kein Tippfehler, wirklich y - warum?

nnwplot.plotTwoFeatures(X, T, clf.predict)
```

Was passiert hier? Ich helfe Ihnen gerne weiter.

3 Supervised Learning 2: Regression

Von *Regression* spricht man im Kontext von Machine Learning typischerweise dann, wenn die Zielwerte `T` *kontinuierlich* sind – im Unterschied zur *Klassifikation*, bei dem die Zielwerte *diskret* (also gerade nicht-kontinuierlich) sind, nämlich z.B. die Nummern der Klassen.

Eigentlich ist diese Verwendung des Begriffs Regression nicht ganz korrekt, weil es auch Regressionsverfahren gibt, die für diskrete Zielwerte sind, z.B. die logistische Regression (logistic regression). Zudem können im Prinzip alle Regressionsverfahren auch für Klassifikation verwendet werden – denn diskrete Zielwerte (sozusagen Datentyp `int`) sind Spezialfälle kontinuierlicher Zielwerte (sozusagen Datentyp `float`) und jedes `int` kann man verlustfrei in einem `float` speichern, nicht aber umgekehrt.⁴ Liefert das Regressionsverfahren kontinuierliche Werte, wird, um daraus ein Klassifikationsverfahren zu machen, noch eine *Entscheidungsregel* (decision rule) gebraucht, die die endgültige

⁴Bis auf Rundungsfehler.

Entscheidung, welche (diskrete) Klasse(nnummer) es denn nun ist, anhand der kontinuierlichen Werten trifft.

Regressionsverfahren in *scikit-learn* benutzt man exakt genauso wie Klassifikationsverfahren: Klasse anlegen, Training mit `fit`, Anwendung durch Aufruf von `predict`.

Zunächst soll es um die Daten gehen, die durch den folgenden Code erzeugt und geplottet werden:

```
import matplotlib.pyplot as plt

X = np.linspace(0, 2)
np.random.seed(42) # vereinfachte Variante, siehe np.random

X = np.sort(np.random.rand(50)*2)
T = np.sin(5*X) + X**2 + np.random.randn(len(X))/10

plt.plot(X, T, '.')
plt.xlabel('x')
plt.ylabel('t')
plt.show()
```

- a) Wie viele Merkmale gibt es bei diesen Daten? Welche Dimension hat also der *Merkmalsraum*?
- b) Als erstes Verfahren soll eine lineare Regression trainiert werden. Bei einer linearen Regression werden, wie der Name schon sagt, nur lineare Operationen durchgeführt, also die Eingabewerte mit vom Trainings-Algorithmus gewählten Zahlen multipliziert und konstante Werte addiert.

Die *scikit-learn*-Klasse ist: `sklearn.linear_model.LinearRegression`. Nennen Sie die Variable für das Objekt `reg`.

Damit `fit` und `predict` funktioniert, muss `X.reshape(-1,1)` statt `X` selbst übergeben werden – warum?

- c) Plotten Sie das Ergebnis als durchgezogene Linie in dasselbe Diagramm wie die Ausgangsdaten. Dazu einfach nochmal `plt.plot` aufrufen und `plt.show()` hinter diesen zweiten `plt.plot` Aufruf verschieben.

Das Ergebnis ist natürlich nicht besonders gut, weil die Funktion ja lineare Funktion (also hier: keine Gerade) ist.

- d) Probieren Sie stattdessen `sklearn.tree.DecisionTreeRegressor` und `sklearn.svm.SVR`.
- e) Probieren Sie nun `sklearn.svm.SVR` auf den Iris-Daten – auch das geht, denn Klassifikation ist gewissermaßen ein Spezialfall der Regression. Kopieren Sie dafür am besten Ihr Skript der ersten Teilaufgabe.

Das Plot mittels `nnwplot.plotTwoFeatures` sieht anders aus als bei der Aufgabe zur Klassifikation – warum?

- f) Damit das Plot wie bei einer Klassifikation aussieht, brauchen wir noch eine *Entscheidungsregel* (decision rule), siehe oben. Fügen Sie vor der Berechnung der Anzahl der richtigen Klassifikationen und dem Plot Folgendes ein:

```
decision = lambda x : np.round(reg_iris.predict(x)) # kurze Funktionsdefinition
y = decision(X)
```

Übergeben Sie zudem die neue Funktion `decision` an `nnwplot.plotTwoFeatures`.