Laborversuch 2

Versuch	**Lineare Modelle und Regularisierung**
Fach	Int. Syst. u. Masch. Lernen (ISML)
Semester	WS 2023/24
Fachsemester	TIN5/ITS5/WIN5
Labortermine	07.12.2023
	14.12.2023
Abgabe bis spätestens	22.12.2023

Versuchsteilnehmer

Name: Vorname:

Semester: Matrikelnummer:

Bewertung des Versuches

Aufgabe:	1	2	3
Punkte maximal:	45	30	35 (ZP)
Punkte erreicht:			

Gesamtpunktezahl: /75 Note: Zeichen:

Anmerkungen:

Aufgabe 1: (10+9+14+12 = 45 Punkte)

Thema: Lineares Modell mit polynomiellen Basisfunktionen Wir wollen lineare Modellfunktionen der Form $y = \mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x})$ nach (4.2) bzw. $\mathbf{y} = \mathbf{W}\boldsymbol{\phi}(\mathbf{x})$ nach (4.1) im Skript implementieren. Für die Merkmalsvektoren $\phi(\mathbf{x})$ sollen dabei polynomielle Basisfunktionen vom Typ

$$\phi_j(\mathbf{x}) := x_1^{n_1} x_2^{n_2} \cdots x_d^{k_d}$$

wie in (4.5,B.7) verwendet werden. Betrachten Sie hierzu das Programmgerüst `polynomial_basis_functions.py` aus dem Praktikumsverzeichnis.

a) Versuchen Sie zunächst die Funktion `get_phi_polyD1(m)` für eindimensionale Inputs $\mathbf{x} := (x) \in \mathbb{R}$ zu verstehen und beantworten Sie die folgenden Fragen:

- Lambda-Formalismus in Python: Was bedeutet der Ausdruck `lambda x,n=n: x[0]**n` für $n = 3$?
- Welche Funktionen enthält also die Liste `phi` für $m = 5$?
- Was gibt die Funktion `get_phi_polyD1(m)` für $m = 5$ als Ergebnis zurück?
- Warum muss man in `phi = [lambda x,n=n: x[0]**n for n in range(m+1)]` den Exponenten n als "default parameter" `n=n` übergeben? Testen Sie was passiert wenn man dies nicht tut bzw. den Teil "`,n=n`" weglässt?
- Was ist der Unterschied zwischen der Liste `phi` und dem Ergebnis `lambda x: np.array([phi_j(x) for phi_j in phi])` in der letzten Zeile der Funktion?
- Berechnen Sie den Merkmalsvektor $\boldsymbol{\phi}(\mathbf{x})$ für $m = 5$ und Input $\mathbf{x} := \begin{pmatrix}2\end{pmatrix}^{\mathrm{T}}$.

b) Versuchen Sie nun die Funktion `get_phi_polyD2(m)` für zweidimensionale Inputs $\mathbf{x} \in \mathbb{R}^2$ zu verstehen und beantworten Sie die folgenden Fragen:

- Was bedeuten die Variablen `n`, `n0`, `n1`?
- Welche Funktionen enthält die Liste `phi` für $m := 4$?
- Wieviele Funktionen enthält die Liste `phi` für beliebiges m?
- Berechnen Sie $\boldsymbol{\phi}(\mathbf{x})$ für $m = 4$ und Input $\mathbf{x} := \begin{pmatrix}1 & 2\end{pmatrix}^{\mathrm{T}}$.

c) Implementieren Sie nun analog zu den vorigen Funktionen `get_phi_polyD3(m)` für dreidimensionale Inputs $\mathbf{x} \in \mathbb{R}^3$. Als Ergebnis soll wieder die polynomielle Merkmalsfunktion $\boldsymbol{\phi}(\mathbf{x})$ für Grad m zurückgegeben werden. Testen Sie Ihre Funktion indem Sie $\boldsymbol{\phi}(\mathbf{x})$ für $m = 3$ und $\mathbf{x} := \begin{pmatrix}1 & 2 & 3\end{pmatrix}^{\mathrm{T}}$ berechnen.

Hinweis: Das Ergebnis des Tests sollte der Merkmalsvektor $\boldsymbol{\phi}(\mathbf{x}) = (1, 3, 2, 1, 9, 6, 4, 3, 2, 1, 27, 18, 12, 8, 9, 6, 4, 3, 2, 1)^{\mathrm{T}}$ sein.

d) Implementieren Sie die entsprechende allgemeine Funktion `get_phi_poly(d,m)` für d-dimensionale Inputs $\mathbf{x} \in \mathbb{R}^d$ und polynomielle Merkmalsfunktion $\boldsymbol{\phi}(\mathbf{x})$ vom Grad m. Testen Sie Ihre Implementierung indem Sie $\boldsymbol{\phi}(\mathbf{x})$ für die folgenden Fälle berechnen:

- $d = 1$, $m = 4$ für $\mathbf{x} = \begin{pmatrix}2\end{pmatrix}^{\mathrm{T}}$
- $d = 2$, $m = 3$ für $\mathbf{x} = \begin{pmatrix}1 & 2\end{pmatrix}^{\mathrm{T}}$
- $d = 3$, $m = 2$ für $\mathbf{x} = \begin{pmatrix}1 & 2 & 3\end{pmatrix}^{\mathrm{T}}$
- $d = 4$, $m = 3$ für $\mathbf{x} = \begin{pmatrix}1 & 2 & 3 & 4\end{pmatrix}^{\mathrm{T}}$

Vergleichen Sie für $d \leq 3$ mit den Test-Ergebnissen der vorigen Teilaufgaben (die Ergebnisse sollten übereinstimmen!).

Hinweise: Am einfachsten importieren Sie das Python-Modul `itertools` und verwenden `itertools.product(.)` um das d-fache **Kartesische Produkt** $\{0, 1, 2, \ldots, m\}^d$ zu berechnen (Sie können für die Potenzierung mit d den Parameter `repeat=d` setzen). Für Kartesische Produkte siehe Skript Mathe1, Def. 1.6. Filtern Sie dann die resultierende Liste aller Tupel $(n_1, \ldots, n_d)$ mit einer List-Comprehension nach den **zulässigen Tupeln** mit $(n_1, \ldots, n_d) = n$ für $n = 0, 1, \ldots, m$ und erzeugen mit dem Lambda-Formalismus die zugehörigen Funktionen $x_1^{n_1} x_2^{n_2} \cdots x_d^{n_d}$. Speichern Sie diese wieder in einer Liste `phi`. Achten Sie darauf, dass die Funktionen in derselben Reihenfolge wie in den vorigen Teilaufgaben generiert werden, damit Sie beim Vergleichen dieselben Vektoren erhalten.

e) Schreiben Sie schließlich eine Funktion `evaluate_linear_model(W,phi,x)` um die Werte von linearen Modellfunktionen vom Typ $y = \mathbf{w}^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x})$ bzw. $\mathbf{y} = \mathbf{W} \boldsymbol{\phi}(\mathbf{x})$ zu berechnen. Testen Sie Ihre Implementierung für die folgenden Fälle:

- $d = 4$, $m = 3$ für $\mathbf{x} = \begin{pmatrix} 1 & 2 & 3 & 4 \end{pmatrix}^{\mathrm{T}}$ und $\mathbf{w} = \begin{pmatrix} \frac{2}{1} & \frac{3}{2} & \frac{4}{3} & \cdots \frac{36}{35} \end{pmatrix}$
- $d = 4$, $m = 3$ für $\mathbf{x} = \begin{pmatrix} 1 & 2 & 3 & 4 \end{pmatrix}^{\mathrm{T}}$ und $\mathbf{W} = \begin{pmatrix} \frac{2}{1} & \frac{3}{2} & \frac{4}{3} & \cdots \frac{36}{35} \\ \frac{36}{35} & \frac{35}{34} & \cdots & \frac{2}{1} \end{pmatrix}$

Hinweis: Die Ergebnisse sollten $y \approx 456.442833$ bzw. $\mathbf{y} \approx \begin{pmatrix} 456.442833 & 456.882633 \end{pmatrix}^{\mathrm{T}}$ sein.

Aufgabe 2: (10+9+14+12 = 45 Punkte)

Thema: Python-Modul für Least-Squares- und KNN-Regression

In Versuch 1 haben wir ein Python-Modul für Klassifikation erstellt. Nun wollen wir ein entsprechendes Modul für Regressions-Verfahren implementieren (siehe Programmgerüst `Regression.py`).

a) Versuchen Sie zunächst den Aufbau des Moduls `Regression.py` zu verstehen:

- Betrachten Sie den Aufbau des Moduls durch Eingabe von `pydoc Regression`. Welche Klassen gehören zu dem Modul und welchen Zweck haben sie jeweils?
- Betrachten Sie nun die Basis-Klasse `Regressifier` im Quelltext: Wozu dienen jeweils die Methoden `fit(self,X,T)`, `predict(self,x)` und `crossvalidate(self,S,X,T)` ?
- Worin unterscheidet sich `crossvalidate(.)` von der entsprechenden Methode für Klassifikation (siehe vorigen Versuch)?

b) Betrachten Sie die Klasse `LSRRegressifier`:

- Welche Art von Regressions-Modell soll diese Klasse implementieren?
- Wozu dienen jeweils die Parameter `lmbda`, `phi`, `flagSTD` und `eps` ?
- Welche Rolle spielt hier die Klasse `DataScaler`?
  In welchen Methoden und zu welchem Zweck werden die Daten ggf. umskaliert?
  Welches Problem kann auftreten wenn man dies nicht tut?
  Wozu braucht man die Variablen `Z` und `maxZ` in der Methode `fit(.)`?

- Vervollständigen Sie die Methoden `fit(self,X,T,...)` und `predict(self,x,...)`.

  Hinweis: Siehe im (4.15) mit (4.12,4.11) im Skript.

c) Betrachten Sie die Klasse `KNNRegressifier`:

- Welche Art von Regressions-Modell berechnet diese Klasse?
- Wozu dienen jeweils die Parameter `K` und `flagKLinReg`?
- Beschreiben Sie kurz in eigenen Worten (2-3 Sätze) auf welche Weise die Prädiktion $\mathbf{y}(\mathbf{x})$ berechnet wird.

d) Betrachten Sie abschließend den Modultest:

- Beschreiben Sie kurz was im Modultest passiert.
- Welche Gewichte W werden gelernt? Wie lautet also die gelernte Prädiktionsfunktion? Welche Funktion sollte sich idealerweise (für $N \to \infty$) ergeben?
- Welche Ergebnisse liefert die Kreuzvalidierung? Was bedeuten die Werte?
- Vergleichen und Bewerten Sie die Ergebnisse von Least Squares Regression gegenüber der KNN-Regression (nach Optimierung der Hyper-Parameter λ, K, ...).