Laborversuch 1

Versuch	**Nearest-Neighbor-Methoden und Kernel-MLP für Klassifikation**
Fach	Intelligente Lernende Systeme
Semester	WS 2023/24
Fachsemester	TIN5/ITS5/WIN5
Labortermine	09.11.2023 16.11.2023
Abgabe bis spätestens	24.11.2023

Versuchsteilnehmer

Name: Vorname:

Semester: Matrikelnummer:

Bewertung des Versuches

Aufgabe:	1	2	3	4	5 (ZA)
Punkte maximal:	10	15	35	35	15 (ZP)
Punkte erreicht:					

Gesamtpunktezahl: /95 Note: Zeichen:

Anmerkungen:

Allgemeine Vorbemerkungen zu den Praktikumsaufgaben

- Sie sollten mit Python-Grundlagen vertraut sein (z.B. List-Comprehensions, Dicts, Numpy-Arrays, Definition von Funktionen, Definition von Klassen, Vererbung, etc.)
- Aus Zeitgründen wird bei den meisten Praktikums-Aufgaben schon ein Programmgerüst vorgegeben, das Sie nur noch vervollständigen müssen.
- Achten Sie hierbei auf Anweisungen in den Kommentarzeilen, z.B. `# INSERT ...` oder `# REPLACE ... !`
- Versuchen Sie trotzdem auch die anderen bereits vorgegebenen Progammteile zu verstehen !
- Sie finden die Programmgerüste und Datensammlungen im Vorlesungsverzeichnis: I-Platte, Unterverzeichnis Praktikum/PRAKTIKUMSVERZEICHNIS.
- Die Grobstruktur eines Python-Moduls `myModule.py` können Sie mit dem Shell-Kommando `pydoc myModule` ansehen. Hierbei werden Klassen, Funktionen, Variablen sowie die Docstrings des Moduls ausgegeben sowie deren Docstrings (=Kommentare unter Funktions/Klassennamen mit dreifachen Anführungszeichen ‘‘ ‘‘ ‘‘) angezeigt.
- Es wird (meist) dieselbe Notation wie in der Vorlesung verwendet (siehe Skript) !
- Z.B. bezeichnet `X` die Daten-Matrix `X`, deren Zeilen `X[i]` die Datenvektoren sind. Ähnlich enthalten die Zeilen der Design-Matrix `PHI` die Merkmalsvektoren `PHI[i]` und die Zeilen der Zielwerte-Matrix `T` die Zielvektoren bzw. Labels `T[i]`.
- `N`, `M` und `D` sind üblicherweise Anzahl der Datenvektoren, Dimension der Merkmalsvektoren und Dimension der Datenvektoren.
- Details zu Python/Numpy-Funktionen finden Sie am einfachsten in den Online-Dokus. Um z.B. Informationen über die Funktion `numpy.argsort(.)` zu finden googeln Sie “numpy argsort”.
- Lesen Sie die Hinweise zu den Aufgaben !

Hinweise zur Abgabe der Ausarbeitungen:

- Erstellen Sie eine elektronische Ausarbeitung (als pdf) welche die Antworten zu allen Praktikumsaufgaben enthält.
- Die Ausarbeitung soll außerdem Snapshots der relevanten Programm-Teile bzw. Grafiken enthalten.
- Bitte laden Sie eine zip-Datei aller notwendigen Dateien auf Ilias hoch.
- Werfen Sie außerdem einen Ausdruck Ihrer Ausarbeitung in das Fach von Prof.Knoblauch.
- Abgabedatum steht jeweils auf dem Deckblatt des Aufgabenblatts.

Aufgabe 1: (4+2+4 = 10 Punkte)

Thema: Erzeugung Gauß-verteilter synthetischer gelabelter Daten für Klassifikation und Darstellung mit Matplotlib und IVISIT

a) Schreiben Sie eine Python-Funktion
`X,T=getGaussData2D(N,mu1,mu2,Sig11,Sig22,Sig12,t=0,C=2,flagOneHot=0)`
welche zweidimensionale Gauß-verteilte gelabelte Daten erzeugt (siehe Skript, Anhang **??**, Gleichung (**??**) und Abb. **??**). Die Funktion soll Datenmatrix $\mathbf{X}$ und Zielwertematrix (oder -vektor) $\mathbf{T}$ erzeugen.

Hinweise:

- Sie können dazu das Programmgerüst `GaussDataGeneration.py` aus dem Vorlesungsverzeichnis vervollständigen. Dort werden auch die Parameter genauer beschrieben.
- Zur Daten-Erzeugung können Sie die Numpy-Funktion `np.random.multivariate_normal` verwenden.

b) Testen Sie Ihre Implementierung indem Sie Daten für zwei Klassen mit Parametern

$$\text{Klasse 1:} \quad N = 10, \quad \mu := \begin{pmatrix} 1 \\ 2 \end{pmatrix}, \quad \mathbf{\Sigma} := \begin{pmatrix} 1 & 0.1 \\ 0.1 & 2 \end{pmatrix}$$
$$\text{Klasse 1:} \quad N = 15, \quad \mu := \begin{pmatrix} 2 \\ 1 \end{pmatrix}, \quad \mathbf{\Sigma} := \begin{pmatrix} 2 & 0.2 \\ 0.2 & 1 \end{pmatrix}$$

generieren und die Matrizen $\mathbf{X}$ und $\mathbf{T}$ auf dem Bildschirm ausgeben. Verwenden Sie den Seed $s := 13$ für den Zufallszahlengenerator. Schätzen Sie außerdem Mittelwerts-Vektor μ und Kovarianzmatrix Σ der Gesamtdaten Sie können hierfür die Schätzer (**??**) von Anh. **??** im Skript verwenden.

Hinweise: Für die Summanden im Kovarianzen-Schätzer von (**??**) können Sie z.B. `np.outer` verwenden. Zum Setzen des Seeds verwenden Sie `np.random.seed`.

c) Integrieren Sie ihre Implementierung in eine IVISIT-Simulation. Testen Sie Ihre Implementierung indem Sie Daten für zwei Klassen generieren und die Matrizen $\mathbf{X}$ und $\mathbf{T}$ auf dem Bildschirm ausgeben:

$$\text{Klasse 1:} \quad N = 10, \quad \mu := \begin{pmatrix} 1 \\ 2 \end{pmatrix}, \quad \mathbf{\Sigma} := \begin{pmatrix} 1 & 0.1 \\ 0.1 & 2 \end{pmatrix}$$
$$\text{Klasse 1:} \quad N = 15, \quad \mu := \begin{pmatrix} 2 \\ 1 \end{pmatrix}, \quad \mathbf{\Sigma} := \begin{pmatrix} 2 & 0.2 \\ 0.2 & 1 \end{pmatrix}$$

Hinweise:

- Für IVISIT siehe Einführung zum Praktikum und die Dokumentation auf ILIAS (Tutorials).
- Sie können das Programmgerüst `ivisit_GaussDataGeneration.py` aus dem Vorlesungsverzeichnis vervollständigen.
- Starten z.B. aus der Kommandozeile mit `python ivisit_GaussDataGeneration.py ivisit_GaussDataGeneration.db`

Aufgabe 2: (8+4+3 = 15 Punkte)

Thema: Naive Implementierung einer k-Nearest-Neighbors-Suche in Python

a) Schreiben Sie eine Python-Funktion `getKNearestNeighbor(X, x, K=1)` welche zum Testvektor `x` eine Index-Liste der K Nearest-Neighbors in der Datenmatrix `X` zurückgibt.

- Hierbei soll der Daten-Vektor `x` als Array aus D Zahlen und die Datenmatrix `X` als Array aus N Daten-Vektoren dargestellt werden.
- Verwenden Sie als Abstandsmaß zwischen den Vektoren die euklidische Distanz, d. h. $\|\mathbf{x} - \mathbf{x_n}\|^2 = \sum_{i=1}^{D}(x_i - x_{n,i})^2$.

b) Testen Sie Ihre Implementierung mit einer Datenmatrix aus dreidimensionalen Datenvektoren $\mathbf{x}_1 = \begin{pmatrix}1 & 2 & 3\end{pmatrix}^{\mathrm{T}}$, $\mathbf{x}_2 = \begin{pmatrix}2 & 3 & 4\end{pmatrix}^{\mathrm{T}}$, $\mathbf{x}_3 = \begin{pmatrix}3 & 4 & 5\end{pmatrix}^{\mathrm{T}}$, $\mathbf{x}_4 = \begin{pmatrix}4 & 5 & 6\end{pmatrix}^{\mathrm{T}}$ und dem Test-Datenvektor $\mathbf{x} = \begin{pmatrix}1.5 & 3.6 & 5.7\end{pmatrix}^{\mathrm{T}}$:
Geben Sie auf dem Bildschirm zunächst die Euklidischen Abstände von $\mathbf{x}$ zu allen Datenvektoren in der Datenbank aus. Bestimmen Sie dann die $K = 2$ Nearest-Neighbors zu $\mathbf{x}$ und geben Sie diese auf dem Bilschirm aus.

c) Laufzeit/Komplexitätsanalyse: Wie viele Rechenschritte in Abhängigkeit von $N :=$`len(X)`, $D :=$`len(x)` und K braucht Ihr Programm ungefähr, um eine Anfrage zu bearbeiten (Angabe in O-Notation reicht)? Wie könnte man das Verfahren beschleunigen?

Hinweise:

- Vervollständigen Sie das Programmgerüst `KNearestNeighborSearch.py` aus dem Vorlesungsverzeichnis.
- Die Euklidische Distanz ||.|| können Sie entweder "von Hand" oder mit Hilfe der Numpy-Funktion `numpy.linalg.norm(.)` berechnen.
- Um die Indexe der K Nearest Neighbors zu bestimmen können Sie die Distanzen z.B. mit Hilfe von `numpy.argsort(.)` sortieren.

Aufgabe 3: (6+15+5+9 = 35 Punkte)

Thema: Python-Modul für k-Nearest-Neighbor-Klassifikatoren

Da wir im folgenden verschiedene Klassifikations-Verfahren implementieren wollen lohnt es sich ein eigenes Python-Modul dafür zu erstellen (siehe Programmgerüst `V1A2_Classifier.py`).

a) Versuchen Sie zunächst den Aufbau des Moduls `V1A2_Classifier.py` zu verstehen:

- Betrachten Sie den Aufbau des Moduls durch Eingabe von `pydoc V1A2_Classifier`. Welche Klassen gehören zu dem Modul und welchen Zweck haben sie jeweils?
- Betrachten Sie nun die Basis-Klasse `Classifier` im Quelltext: Wozu dienen jeweils die Methoden `__init__(self,C)`, `fit(self,X,T)`, `predict(self,x)` und `crossvalidate(self,S,X,T)` ?

b) Die von `Classifier` abgeleitete Klasse `KNNClassifier` soll einen einfachen k-NN-Klassifikator implementieren. Beantworten Sie die folgenden Fragen bzw. vervollständigen Sie die Klasse `KNNClassifier` entsprechend:

- Wie "lernt" ein k-NN-Klassifikator? Beschreiben Sie kurz was die Methode `fit(self,X,T)` macht.
- Implementieren Sie die Methode `getKNearestNeighbors(self, x, k=None, X=None)`, indem Sie Ihren Code aus Aufgabe 1 kopieren und anpassen.
- Implementieren Sie die Methode `predict(self,x,k=None)`:
  Der Rückgabewert `prediction` soll die wahrscheinlichste Klasse für Testvektor `x` enthalten. Die Rückgabewerte `pClassPosteriori[i]`:= $\frac{\#\text{NN mit Klassen-Label i}}{k}$ sollen die A-Posteriori-Wahrscheinlichkeiten aller Klassen $0, 1, \ldots, C-1$ enthalten.

c) Testen Sie Klasse `KNNClassifier` für Daten `X = [[1,2,3],[2,3,4],[3,4,5],[4,5,6]]` mit Labels `T=[0,1,0,1]` (siehe Hauptprogramm/Modultest). Welche Ergebnisse erhalten Sie für Test-Vektor `x=[1.5,3.6,5.7]`? Geben Sie für $k = 1$, $k = 2$ und $k = 3$ jeweils die k Nearest-Neighbors, die wahrscheinlichste Klasse sowie die A-posteriori-Klassen-Verteilung an.
Warum sollte man für $C = 2$ Klassen immer ungerades k wählen?

d) Vervollständigen Sie die von `KNNClassifier` abgeleitete Klasse `FastKNNClassifier` um einen effizienteren k-NN-Klassifikator zu implementieren:

- Vervollständigen Sie die Methode `fit(self,X,T)` um einen KD-Tree für die Daten `X` aufzubauen.
- Vervollständigen Sie die Methode `getKNearestNeighbors(self, x, k=None)` um mit dem KD-Tree die K Nearest-Neighbor zu bestimmen.
- Vervollständigen Sie den Modultest (im Hauptprogramm) und testen Sie damit den `FastKNNClassifier` ähnlich wie in (c).

Hinweise:

- Vervollständigen Sie das Programmgerüst `V1A2_Classifier.py` aus dem Vorlesungsverzeichnis.
- Falls in `predict(.)` mehrere Klassen maximale Wahrscheinlichkeit haben, können Sie die `prediction` mit Hilfe der Python-Funktion `random.randint(<min>,<max>)` zufällig auswählen.
- Für den KD-Tree können Sie die SciPy-Klasse `scipy.spatial.KDTree` benutzen (siehe Online-Doku).
- Zum Bestimmen der k Nearest Neighbors mit dem KD-Tree können Sie dessen Methode `query(x)` benutzen, welche die k minimalen Distanzen und die zugehörigen Indexe liefert (siehe Online-Doku).

Aufgabe 4: (4+6+6+10+9 = 35 Punkte)

Thema: Kreuzvalidierung und Effizienz des k-NN-Klassifikators

a) Allgemeine Fragen zur Evaluation eines Klassifikators:

- Welche Klassifikationsfehlerwahrscheinlichkeit erhält man, wenn man einen k-NN-Klassifikator für $k = 1$ auf der gespeicherten Lern-Daten-Menge `X` testet?
- Bedeutet dies, dass der k-NN-Klassifikator auch auf neuen Datenvektoren `x` (welche nicht gespeichert sind) immer korrekt klassifiziert?
- Was kann man tun um einen realistischen Schätzwert des Generalisierungsfehlers (d.h. der Klassifikationsfehlerwahrscheinlichkeit für neue Daten) zu erhalten ?
- Erklären Sie kurz den Begriff Kreuzvalidierung und ihren Zweck! Lesen Sie hierzu im Skript (siehe Folien zu Kapitel 2).

b) Code Review: Betrachten Sie das Python-Modul `V1A2_Classifier.py` aus Aufgabe 2. Versuchen Sie die Implementierung der Methode `Classifier.crossvalidate(self,S,X,T)` zu verstehen:

- Was bedeutet der Parameter `S`?
- Welche Rolle spielen die Variablen `perm` sowie `Xp` und `Tp`?
- Welche Rolle spielt `idxS`?
- Was bewirkt die äußere Schleife `for idxTest in idxS: ...`?
- Welche Rolle haben die Variablen `X_learn` und `T_learn` bzw. `X_test` und `T_test`?
- Was passiert für `S=1`?
- Was bewirkt die innere Schleife `for i in range(len(X_test)): ...`?
- Was bedeuten die Ergebnisse der Kreuzvalidierung `pClassError` und `pConfErrors`?

c) Wir betrachten nun ein 2-Klassen-Problem für Gauß-verteilte Datenvektoren $\mathbf{x}_n \in \mathbb{R}^D$ mit D-dimensionalen Dichtefunktionen

$$\mathcal{N}_i(\mathbf{x}|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) = \frac{1}{\sqrt{(2\pi)^D |\boldsymbol{\Sigma}_i|}} \exp\left\{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_i)^{\mathrm{T}} \boldsymbol{\Sigma}_i^{-1} (\mathbf{x}-\boldsymbol{\mu}_i)\right\}$$

wobei $\boldsymbol{\mu}_i$ der Mittelwert und $\boldsymbol{\Sigma}_i$ die $D \times D$ Kovarianzmatrix für Klasse $i \in \{0, 1\}$ ist. D.h. ähnlich wie bei der 1-dimensionalen Gauß-Dichte (siehe Skript) ist $\boldsymbol{\mu}_i$ der Punkt mit höchster Wahrscheinlichkeit, und $\boldsymbol{\Sigma}_i$ definiert wie weit um $\boldsymbol{\mu}_i$ die Datenpunkte streuen. Versuchen Sie das Programmgerüst `V1A3_CrossVal_KNN.py` zu verstehen und zu vervollständigen:

- Wozu benötigt man den Befehl `from V1A2_Classifier import *` ?
- Mit welchem Befehl werden die Gauß-verteilten Datenvektoren erzeugt? Wieviele Datenpunkte werden generiert? Was bedeuten die Variablen `N`, `N1`, `N2` ?
- Welche Klassenspezifischen Verteilungen haben die Daten? Geben Sie für jede Klasse Mittelwert und Kovarianzmatrix an!
- Welche Bedeutung haben die Variablen `pE_naive`, `pCE_naive` und `t_naive` ?
- Der Programmteil unter `# (ii.a) test of naive KNN classifier` erstellt und testet einen `KNNClassifier`. Vervollständigen Sie ähnlich den Code unter `# (ii.b) test of KD-tree KNN classifier` unter Verwendeung eines `FastKNNClassifier` aus dem Modul `V1A2_Classifier.py`.

d) Testen Sie die Kreuzvalidierung bzw. Klassifikationsleistung Ihrer Implementierung aus Teilaufgabe (c):

- Bestimmen Sie Klassifikationsfehler und Verwechslungswahrscheinlichkeiten für die gegebenen Daten bei Kreuzvalidierung mit dem k-NN-Klassifikator für $k = 1$, $k = 5$, $k = 11$ kombiniert mit $S = 1$, $S = 2$, $S = 5$?
- Bestimmen Sie die Klassenverteilungen für drei weitere Testpunkte $(2, 1)$, $(5, 1)$, $(-1, 1)$ für $k = 1$, $k = 5$, $k = 11$, $k = 111$, $k = 511$. Was ist jeweils die wahrscheinlichste Klasse?

Sind die Ergebnisse für den `KNNClassifier` und den `FastKNNClassifier` gleich?
Stellen Sie die Ergebnisse für einen der beiden Klassifikatoren jeweils in einer Tabelle dar.

e) Vergleichen Sie systematisch die Effizienz der beiden k-NN-Klassifikatoren mit $k = 5$, indem Sie die Laufzeiten von Kreuzvalidierungen mit $S = 1$ und $S = 5$ für eine zunehmende Anzahl $N := N1 + N2 \in \{10, 20, 50, 100, 200, 500, 1000, 2000, 5000, 10000\}$ von Datenvektoren messen. Stellen Sie die gemessenen Laufzeiten als Funktion von N in einer Tabelle bzw. einem Schaubild dar.

Hinweise:

- Sie können das Programmgerüst `V1A3_CrossVal_KNN.py` aus dem Praktikumsverzeichnis verwenden.
- Für Teilaufgabe (e) bietet sich ein automatisierter Test an, wobei Sie eine Schleife über alle gewünschten Werte von N laufen lassen. Sie können hierfür das Programmgerüst `V1A3e_TestEfficiency_KNN.py` aus dem Praktikumsverzeichnis verwenden.

Zusatzaufgabe 5 4+3+6+2 = 15 Punkte

Thema: k-NN-Klassifikation von Satellitenbilder Japanischer Wälder

Implementieren Sie in einem neuen Python-Skript einen Nearest-Neighbor-Klassifikator zur Klassifikation verschiedener (japanischer) Wald-Typen auf Satellitenbildern. Hierfür enthält die Datensammlung `ForestTypesData.csv` (siehe Praktikumsverzeichnis) $N = 524$ Datensätze, wobei jeder Datensatz aus $D = 27$ Bild-Merkmalen der Satellitenaufnahmen besteht (deren genaue Bedeutung uns hier nicht näher zu interessieren braucht). Zusätzlich enthält jeder Datensatz in der ersten Spalte ein Klassenlabel, welches den Datensatz einer von $C = 4$ Klassen zuordnet. Hierbei bedeuten: 's'=Sugi-Zypressenwald, 'h'=Hinoki-Zypressenwald, 'd'=Mischlaubwald, 'o'=unbewaldet.

Versuchen Sie zunächst den Aufbau des Programmgerüsts `V1A4_ForestClassification.py` zu verstehen:

- Laden der Wald-Daten mit der Pandas-Funktion `read_csv` (oder `read_table`; siehe Pandas-Tutorial).
- Die eingelesenen Daten werden wie üblich in Daten-Matrix `X` und Labels `T` umgewandelt. Beachten Sie hierzu:
  - Nach dem Laden z.B. mit `forestdata = pd.read_csv(filename)` liefert `forestdata.values` ein Daten-Array welches die Daten der Tabelle `ForestTypesData.csv` enthält.
  - Die Klassenlabels stehen in der ersten Spalte von `ForestTypesData.csv`, die Bild-Merkmale in den restlichen 27 Spalten.
  - Sie können die Umwandlung der Text-Label 's','h','d','o' in numerische Klassenlabels 0,1,2,3 etwa mit Hilfe einer List-Comprehension erledigen.

Lösen/beantworten Sie nun die folgenden Fragen:

a) Erstellen Sie einen k-Nearest-Neighbor-Klassifikator für die Wald-Daten. Importieren Sie hierfür wieder das Modul `V1A2_Classifier.py` aus Aufgabe 2 und verwenden Sie eine geeignete Klasse für den Klassifikator.

b) Testen Sie den Klassifikator für $k = 3$ durch Kreuzvalidierung mit $S = 5$. Geben Sie die Klassifikationsfehlerwahrscheinlichkeit und Verwechslungsmatrix an.

c) Versuchen Sie die Klassifikationsleistung zu optimieren, z.B. in dem Sie k variieren. Für welches k ist der Klassifikationsfehler am kleinsten? Ändert sich Ihr Ergebnis für verschiedene $S \in \{1, 2, 3, 5, 10, 100\}$?

d) Theoretische Zusatzfrage: Bis auf k gelten k-NN-Verfahren als parameterfrei, d.h. unabhängig von irgendwelchen Annahmen über die Datenverteilung. Ist diese Annahme wirklich gerechtfertigt?

Hinweise:

- Zur Programmieraufgabe: Sie können das Programmgerüst `V1A4_ForestClassification.py` aus dem Praktikumsverzeichnis verwenden.
- Zur Theorie-Frage: Was ist z.B. falls sich der Wertebereich verschiedener Merkmalsdimensionen um mehrere Größenordnungen unterscheiden? Wie könnte man hier abhelfen?