Intelligente adaptive Systeme

Nearest-Neighbor-Methoden für Klassifikation

Team:

Lisa-Marie Mai *87751*

Andreas Lay *87952*

Marius Schenzle *87937*

22.11.2019

Inhaltsverzeichnis

1 Implementierung einer k-Nearest-Neighbors-Suche in Python

1.1 Implementierung der Python-Funktion *getKNearestNeighbor(X, x, k=1)*

1.2 Test der implementierten Python-Funktion

1.3 Angabe der Rechenschritte in O-Notation

2 Python-Modul für k-Nearest-Neighbor-Klassifikatoren

2.1 Aufbau des Moduls *V1A2_ Classifier.py*

2.1.1 Beschreibung der Klassen im Modul

2.1.2 Betrachtung der Basis-Klasse *Classifier*

2.2 Die Klasse *KNNClassifier*

2.2.1 Wie lernt ein k-NN-Klassifikator?

Der k-NN Klassifikator bestimmt die euklidische Distanz der k nächsten, bereits klassifizierten Nachbarn zu einem Merkmalsvektor. Der Merkmalsvektor wird der Klasse zugewiesen, welche unter der k Nachbarn am häufigsten vorkommt.

***fit(self,X,T)* prüft, ob die Matrizen *X(Trainingsdaten)* und *T(Klassenlabels)* die richtigen Dimensionen haben und speichert die Anzahl der Labels in *self.C*.**

2.2.2 Implementierung der Methode *getKNearestNeighbors(self, x, k=None, X=None)*

2.2.3 Implementierung der Methode *predict(self, x, k=None)*

2.3 Test der vollständig implementierten Klasse KNNClassifier

2.3.1 Ergebnisse der Test

2.3.2 Warum sollte man für C = 2 Klassen immer ungerades k wählen?

Da wenn man bei C = 2 z.B. k = 2 wählt, es sein kann, dass einer der Nachbarn Label 0 und der andere Label 1 hat und der Merkmalsvektor somit beiden Klassen zu 50% zugeordnet wird. Bei ungeradem k wird somit eine klare Klassentrennung gewährleistet.

2.4 Vervollständigung der von *KNNClassifier* abgeleiteten Klasse *FastKNNClassifier*

2.4.1 Die Methode *fit(self, X, T)*

2.4.2 Die Methode *getKNearestNeighbors(self, x, k=None)*

2.4.3 Test der Klasse *FastKNNClassifier*

3 Kreuzvalidierung und Effizienz des k-NN-Klassifikators

3.1 Allgemeine Fragen zur Evaluation eines Klassifikators

3.1.1 Klassifikationsfehlerwahrscheinlichkeiten

3.1.2 Diskussion des Ergebnis der Klassifikationsfehlerwahrscheinlichkeit

3.1.3 Möglichkeiten, um einen realistischen Schätzwert des Generalisierungsfehlers zu erhalten

3.2 Code-Review der Methode *Classifier.crossvalidate(self, S, X, T)*

Was bedeutet der Parameter S?

Die Daten werden in S Daten aufgeteilt. Es wird immer ein Teil zum validieren zurückgehalten.

Welche Rolle spielen die Variablen perm sowie Xp und Tp?

perm: eine zufällige Permutation der Zahlen 0 bis N-1 wird in einem np.array abgelegt. Liefert zufällige Indizes für Xp und Tp. *Xp und Tp*: Daten aus X werden in zufälliger Reihenfolge in Xp abgelegt. Daten aus T werden in zufälliger Reihenfolge n Tp abgelegt.

Welche Rolle spielt idxS?

Was bewirkt die äußere Schleife for idxTest in idxS? Durchläuft alle möglichen Datasets und holt sich die Indizes für Trainingsdaten und speichert diese in *X_learn* und *T_learn*. Die restlichen Daten sind Testdaten und werden in *X_test* und *T_test* gespeichert.

Was passiert für S=1?

Für S=1 wird das gesamte Dataset zum Lernen und Trainieren verwendet.

Was bewirkt die innere Schleife for i in range(len(X_test)): ...?

Durchläuft alle Datenvektoren in *X_test* und klassifiziert diese, holt sich aus *T_test* das zugehörige „richtige“ Label und schreibt in die Matrix nC die Häufigkeit, wie oft das jeweilige Label aufkommt. Später wird die Anzahl der Fehler ausgewertet.

Was bedeuten die Ergebnisse der Kreuzvalidierung pClassError und pConfErrors?

In pClassError wird aufsummiert, wie oft die Testdaten nicht korrekt klassifiziert wurden. Die Anzahl wird dann noch durch die Gesamtheit aller Datenvektoren geteilt, um die Wahrscheinlichkeit eines Klassifikationsfehlers zu erhalten. In der Matrix pConfErrors ist auf der Hauptdiagonalen die Anzahl der Fälle, in denen die Testdaten korrekt klassifiziert wurden. Alle anderen Elemente der Matrix enthalten die Anzahl der Fälle, in denen die Testdaten nicht korrekt klassifiziert wurden. Alle Elemente der Matrix werden dann noch durch die Wahrscheinlichkeit für das Auftreten der jeweiligen Klasse geteilt. So erhält man die Wahrheitsmatrix.

3.3 Betrachtung eines 2-Klassen-Problems für Gauß-verteilte Datenvektoren $x_n \in \mathbb{R}^D$ mit D-dim. Dichtefunktion

Wozu benötigt man den Befehl *from V1A2_Classifier import* $*$?

Um die Funktionen die in *V1A2_Classifier.py* definiert und implementiert wurden, in *V1A3_CrossVal_KNN.py* nutzen zu können.

Mit welchem Befehl werden die Gauß-verteilten Datenvektoren erzeugt? Wie viele Datenpunkte werden generiert? Was bedeuten die Variablen N, N1, N2?

Mit dem Befehl $numpy.random.multivariate_normal(mean, cov[, size, check_valid, tol])$ N1 und N2: Anzahl der Datenvektoren für die zwei Klassen. N: Anzahl der Datenvektoren von X1 und X2 insgesamt, also 1000.

Welche klassenspezifischen Verteilungen haben die Daten?

Klasse 1 Mittelwert: [1,1] Klasse 2 Mittelwert: [3,1] Klasse 1 Kovarianzmatrix: [[1,0.5] [0.5,1]] Klasse 2 Kovarianzmatrix: [[1,0.5] [0.5,1]]

Welche Bedeutung haben die Variablen pE_naive, pCE_naive und t_naive?

pE_naive: Wahrscheinlichkeit eines Klassifikationsfehlers
pCE_naive: Wahrheitsmatrix
t_naive: Berechnungszeit

3.4 Test der Kreuzvalidierung bzw. Klassifikationsleistung

3.4.1 Bestimmung der Klassifikationsfehler und Verwechselwahrscheinlichkeiten

3.4.2 Bestimmung der Klassenverteilung für drei weitere Testpunkte

3.5 Vergleich der Effizienz beider k-NN-Klassifikatoren

4 k-NN-Klassifikation von Satellitenbilder japanischer Wälder

4.1 Erstellen eines k-Nearest-Neighbor-Klassifikator für die Wald-Daten

4.2 Test des Klassifikators durch Kreuzvalidierung

4.3 Optimierung der Klassifikationsleistung

4.4 Zusatzfrage