Programmentwurf

TM40507 Maschinelles Lernen

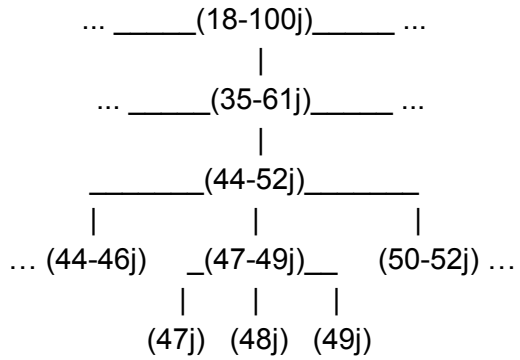
Aufgabenteil a)

Aufgabe: Geben Sie eine geeignete Konfiguration für ein Versionenraum Lernverfahren an, welches aus dem Datensatz die Gewichtseinschätzung erlernen soll.

Um Versionenraum Lernverfahren anzuwenden müssen zuerst alle Merkmale diskretisiert werden.

Die Merkmale Geschlecht und Betätigung sind bereits nominalskaliert. Diese können unverändert übernommen werden.

Für die Merkmale Größe, Alter und Gewicht sollten Merkmalsbäume erstellt werden. Je feiner diese Bäume unterteilt sind, umso genauer kann getrennt werden. Im Idealfall sollte der Baum für jeden möglichen Wert des Merkmals ein Blatt besitzen und dann immer jeweils 2-5 benachbarte Knoten der darunterliegenden Ebene zusammenfassen. Werden in den Blättern schon mehrere Werte zusammengefasst, besteht das Risiko unterschiedlich klassifizierten Beispielen dieselbe Merkmalskombination zuzuweisen. Beispiel eines möglichen Merkmalsbaumes für das Merkmal Alter:



Es wird davon ausgegangen dass nur ganzzahlige Ausprägungen existieren. Aus den zur Verfügung gestellten Daten ergeben sich folgende Wurzelknoten:

Größe:: (140-197cm)
Alter: (18-100j)
Gewicht: (19-150kg)

Die Reihenfolge der Merkmale entspricht dabei den Spalten der CSV-Datei:
(Geschlecht, Größe, Alter, Gewicht, Betätigung)

Beispiele für Tupel :
(w, 160, 25, 50, keinSport)
(m, 180, 35, 100, Kraftsport)

Beispiele für Hypothesen:
(*, 160-168, 60, 40, *)
(*, *, *, 20-22, *)

Es sind zwei Konzepte zu lernen: *istÜbergewichtig* und *istUntergewichtig*. Zuerst werden alle Beispiele für untergewichtige Personen in B+ und alle anderen Personen in B- gelegt um das Konzept *istUntergewichtig* zu lernen. Anschließend werden umgekehrt alle Beispiele für übergewichtige Personen in B+ und alle anderen Personen in B- gelegt um das Konzept *istÜbergewichtig* zu lernen. Mit den resultierenden Konzepten können neue Beispiele klassifiziert werden.

Durch Nutzung des AQ-Algorithmus können auch disjunktive Hypothesen für ein Konzept gelernt werden. Dies ist nötig, da Übergewicht und Untergewicht prinzipiell für jede Ausprägung jedes Merkmals möglich sind. Ein korrekt gelerntes Konzept muss also folgende Form haben:

(m ∧ 170-180cm ∧ …) ∨ (m ∧ 160-170cm ∧ …) ∨ (w ∧ 170-180cm ∧ …) ∨ ...

Aufgabenteil b)

Aufgabe: Konfigurieren Sie ein Neuronales Netz mit der gleichen Aufgabe, implementieren Sie dies geeignet und bewerten Sie das Lernergebnis.

Da vermutet wird, dass die zu lernende Funktion weder extrem komplex noch “mehrstufig” ist wird ein Multilayer-Perceptron mit einem Hidden-Layer genutzt.

Basierend auf den in den Daten vorhandenen Merkmalsausprägungen (siehe oben) werden folgende Input-Neuronen konfiguriert (Werte unter oder über den Schranken werden auf 0 bzw. 1 normiert):

Ein Input für Geschlecht, männlich = 0, weiblich = 1
Ein Input für Größe, normiert zwischen 0 (140 cm) und 1 (200 cm)
Ein Input für Alter, normiert zwischen 0 (18 Jahre) und 1 (100 Jahre)
Ein Input für Gewicht, normiert zwischen 0 (20kg) und 1 (150 kg)

Ein Input für Kraftsport, 0 oder 1
Ein Input für Ausdauersport, 0 oder 1

Das Input-Layer besitzt also 6 Neuronen. Um die Gewichtseinschätzung auszugeben werden 2 Ouput-Neuronen genutzt, jeweils eines für über- und untergewichtig. Ein Beispiel wird einer Klasse zugeordnet wenn der jeweilige Output-Wert über 0.5 liegt. Doppel-Klassifizierungen wären hier theoretisch möglich, sollten aber bei korrekten Daten nicht vorkommen. Alternativ können 3 Output-Neuronen für die 3 Klassen normal, unter- und übergewichtig genutzt werden, dies hat aber nicht zu besseren Ergebnissen geführt.

Die Parameter des Feed-Forward-Netzes wurden folgendermaßen gewählt:

Als Transferfunktion wird die Sigmoid-Funktion genutzt. Sie besitzt überall eine Ableitung ungleich null und ist damit für Backpropagation geeignet.

Die Zahl der Hidden-Neuronen hat Einfluss auf Lerngeschwindigkeit und Genauigkeit (Accuracy). Für die Gewichtseinschätzung scheinen 6 Hidden-Neuronen ausreichend. Werden zu viele Neuronen genutzt generalisiert das Netz zu wenig und die Genauigkeit bei der Klassifizierung der Testdaten sinkt. Außerdem dauert der Lernprozess länger. Mit unter 6 Hidden-Neuronen scheint das Netz die zugrunde liegende Funktion nicht mehr abbilden zu können und die Genauigkeit sinkt.

Das Epsilon steuert wie genau das Netz die einzelnen Beispiele lernen soll. Mit dem Wert 0.1 wurden gute Ergebnisse erzielt. Für das genau Nachbilden einer Funktion wäre dieser Wert relativ hoch, aber für die Trennung von Klassen innerhalb einer relativ unscharfen Datenmenge funktioniert der Wert gut. Niedrigere Werte führen lediglich zu längeren Trainingszeiten, aber nicht zwangsweise zu einer besseren Genauigkeit. Wird ein Wert höher als 0.3 gewählt lernt das Netz überhaupt nicht mehr.

Die Lernrate steuert wie stark das Netz in jedem Backpropagation-Schritt angepasst wird. Hier wurden mit dem Wert 0.3 gute Ergebnisse erzielt. Zu hohe Lernraten führen zu schlechteren Ergebnissen, niedrigere Lernrate verringern die Lerngeschwindigkeit.

Die Backpropagation-Runden (Epochen) wurden auf 15 begrenzt, da weitere Runden das Ergebnis kaum positiv beeinflussten.

Die 10000 gestellten Datensätze wurde in 7000 Trainingsbeispiele und 3000 Testbeispiele aufgeteilt. Andere Aufteilungen (6000/4000, 5000/5000, 7000/3000) beeinflussten die Genauigkeit nur minimal.

Ergebnisbewertung:

Die maximal erreichbare Genauigkeit mit einem auf den Vorlesungsunterlagen basierenden 6-6-2 Multilayer-Perceptron war ~88%. Mit einer Standard-ML-Bibliotheken (sklearn) und mehreren Hidden Layern konnten nur geringfügig bessere Werte erzielt werden (~90% mit 6-12-12-2). Variable Lernrate, Momentum oder ReLU als Transferfunktion konnten das Ergebnis nicht positiv beeinflussen. Wichtig für das schnelle Lernen von mehreren Tausend Beispielen waren ein hohes Epsilon, eine Toleranz von trotz hohem Epsilon falsch zugeordneten Trainingsbeispielen sowie eine performante Implementierung des Backpropagation-Algorithmus. Es hat sich außerdem gezeigt dass der Lernprozess in dieser Konfiguration schon nach wenigen Iterationen (Epochen) abgebrochen werden kann, da sich später trotz hohen Fehlerraten kaum noch Verbesserungen gefunden wurden.

Eine Betrachtung der falsch klassifizierten ~10% zeigt, dass die Fehler fast ausschließlich im Grenzbereich zwischen zwei Klassen auftreten. Oft ist einer der beiden Output-Werte sehr nah an der 0.5 Grenze. Dies lässt darauf schließen dass sich die Klassen teilweise überschneiden, die Daten also nicht linear separierbar sind.

Dokumentation des abgegeben Quellcodes:

Für das vorgegebene Szenario (Eingabe einer Datei von Trainingsdaten, Training, Eingabe einer Datei von Testdaten, Klassifikation + Bewertung) das Python-Skript *main.py* ausführen (Python 3).

Die eigentliche Implementierung befindet sich in der Datei *classifier.py*. Die Klassen darin können auch interaktiv genutzt werden.

Die Klasse *MultilayerPerceptron* modelliert ein neuronales Netz mit einem Hidden Layer. Die Implementierung ist angelehnt an die in der Vorlesung vorgestellte Datei *backpropagation.py*, jedoch werden numpy Vektor-Operationen genutzt, weswegen *MultilayerPerceptron* kleiner, einfacher und schneller ist. Außerdem wurde der Trainingsalgorithmus vereinfacht (nur noch ein for-loop). Die Funktionalität ist auf die zu lösende Aufgabe zugeschnitten. Zahl der Hidden-Neuronen, Epsilon und Lernrate sind variabel.

Die Klasse *WeightClassifier* nutzt das *MultilayerPerceptron* zur Gewichtseinschätzung. Die Hauptfunktionen sind Beispiele aus einer CSV-Datei zu laden (*load_data*) sowie mit diesen Beispiele das neuronale Netz zu trainieren (*train*) und zu testen (*test*). Auch die Parametrisierung des neuronalen Netzes geschieht in dieser Klasse.

Das (6-zeilige!) Skript *sklearn_comparsion.py* wurde genutzt um die Ergebnisse des selbst implementierten Netzes mit einer Standardlösung zu vergleichen.