Eigene KI-Anwendungen programmieren

Bilderkennung und -generierung, ChatGPT, Neuronale Netze u.v.m.

» Hier geht's direkt zum Buch

DIE LESEPROBE

Kapitel 2
Installation

Bevor wir mit der eigentlichen Entwicklung beginnen, werden wir die hauptsächlich verwendeten Werkzeuge installieren und testen.

Worum geht es in diesem Kapitel?

Die wichtigsten Werkzeuge, die Sie für die Arbeit mit diesem Buch benötigen, werden installiert und getestet.

Die Einrichtung eines Entwicklungsrechners kann sehr zeitaufwendig sein. Nehmen Sie sich diese Zeit! Wir wollen uns später bei der Entwicklung nicht mit kuriosen Fehlermeldungen herumschlagen.

Betriebssystemversionen

Die folgenden Installationsanweisungen wurden auf diesen Maschinen getestet:

- Windows 11
- macOS Ventura 13.2 (Intel)
- Ubuntu LTS Jammy Jellyfish 22.04

2.1 Anaconda-Distribution

Mithilfe der Anaconda-Distribution werden wir Python sowie Jupyter Notebook installieren und die Installationen testen. Die Installationsdatei für die Betriebssysteme Windows, macOS und Linux können Sie auf der Website *www.anaconda.com/products/distribution* herunterladen. Über den Browser wird das Betriebssystem erkannt und die richtige Installationsdatei heruntergeladen.

2.1.1 Windows und macOS

Führen Sie die Installationsdatei aus, und folgen Sie den Anweisungen auf dem Bildschirm. Die empfohlenen Standardeinstellungen können beibehalten werden. Nach der Installation können Sie den Anaconda-Navigator starten (siehe Abbildung 2.1).

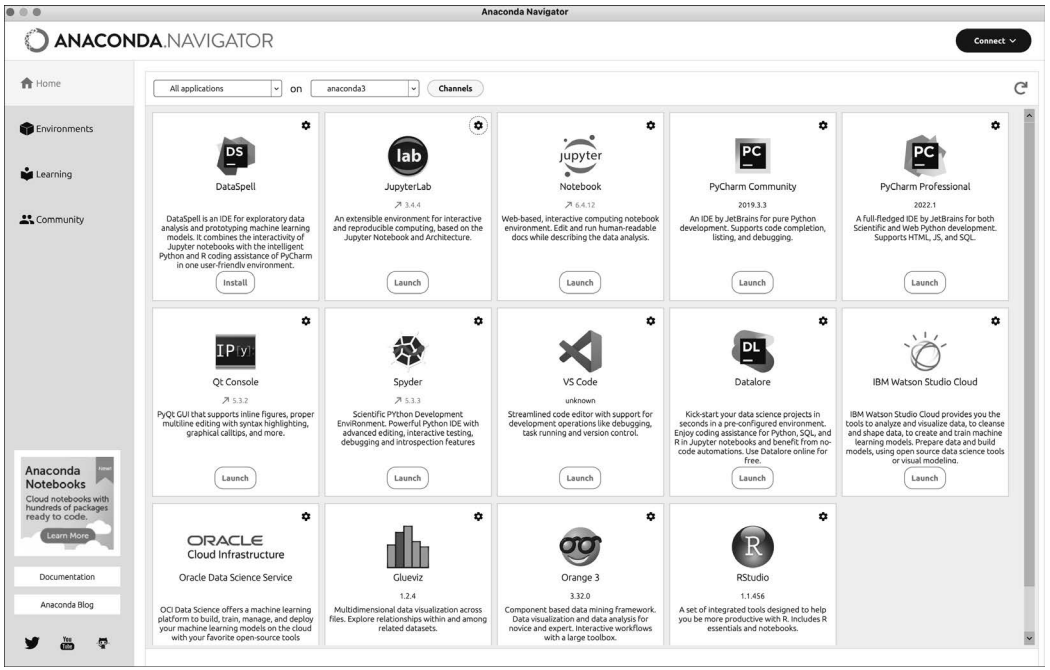


Abbildung 2.1 Home-Bildschirm des Anaconda-Navigators

Nun können wir uns der noch notwendigen Konfiguration des Entwicklungsrechners widmen.

2.1.2 Linux

Installationsanleitung für Anaconda

Eine sehr gute Installationsanleitung für Anaconda finden Sie unter *wiki.ubuntu-users.de/anaconda*.

Nachdem die Datei heruntergeladen und die Installationsroutine durchgeführt wurde, öffnen Sie ein neues Terminalfenster (damit die neuen Umgebungsvariablen wirken) und geben folgenden Befehl ein, um den Anaconda-Navigator zu starten:

```
anaconda-navigator
```

Damit ist die Installation schon abgeschlossen. Sie müssen aber noch einige Einstellungen vornehmen.

2.1.3 Konfiguration und Test

Nach erfolgreicher Installation können Sie im Home-Bereich des Anaconda-Navigators verschiedene Anwendungen über den jeweiligen Launch-Button starten. Wenn Sie nun hierüber Jupyter Notebook starten, kommen Sie auf die Jupyter-Notebook-Homepage (siehe Abbildung 2.2).

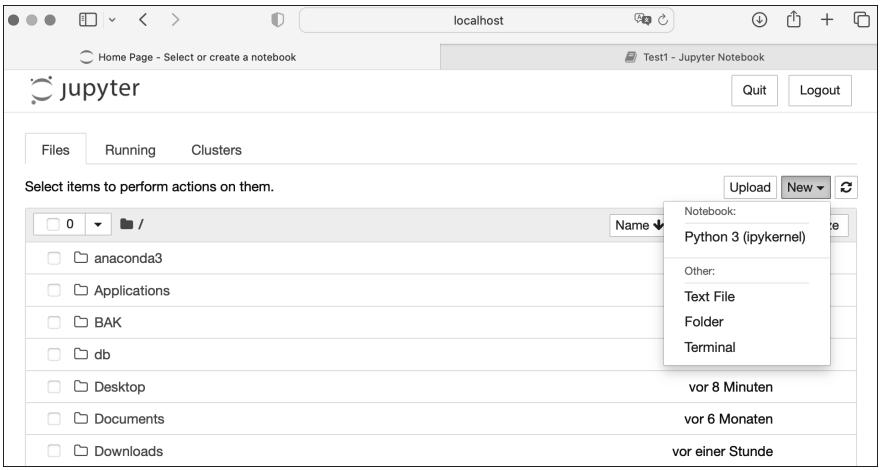


Abbildung 2.2 Jupyter-Notebook-Homepage

Hier gehen wir auf NEW • PYTHON3 (IPYKERNEL) und starten ein neues Notebook (siehe Abbildung 2.3).

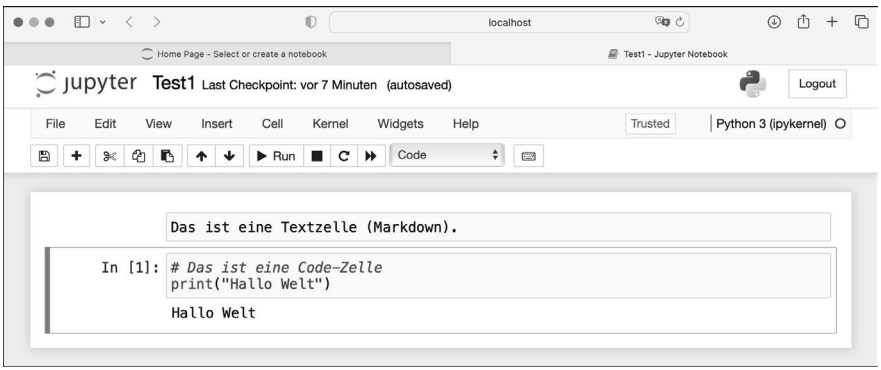


Abbildung 2.3 Erstes Programm in Jupyter Notebook

Das Programm ist in verschiedenen Zellen organisiert. Mit dem Button + können Sie eine neue Zelle hinzufügen und mit den Pfeil-Buttons die selektierte Zelle jeweils nach oben oder unten verschieben. Im Dropdown-Menü können Sie einstellen, ob die Zelle nur zur Dokumentation dient (Markdown), oder aber Quellcode enthält (Code). Der Play-Button führt die selektierte Codezelle, der Forward-Button alle Zellen aus. Mit File • Rename können Sie die Datei umbenennen, mit File • Save as angeben, wo die Datei gespeichert werden soll.

Im Anaconda-Navigator können Sie alternativ JupyterLab starten (siehe Abbildung 2.4), was ebenfalls Notebooks ausführen kann, aber mehr Funktionalität bietet.

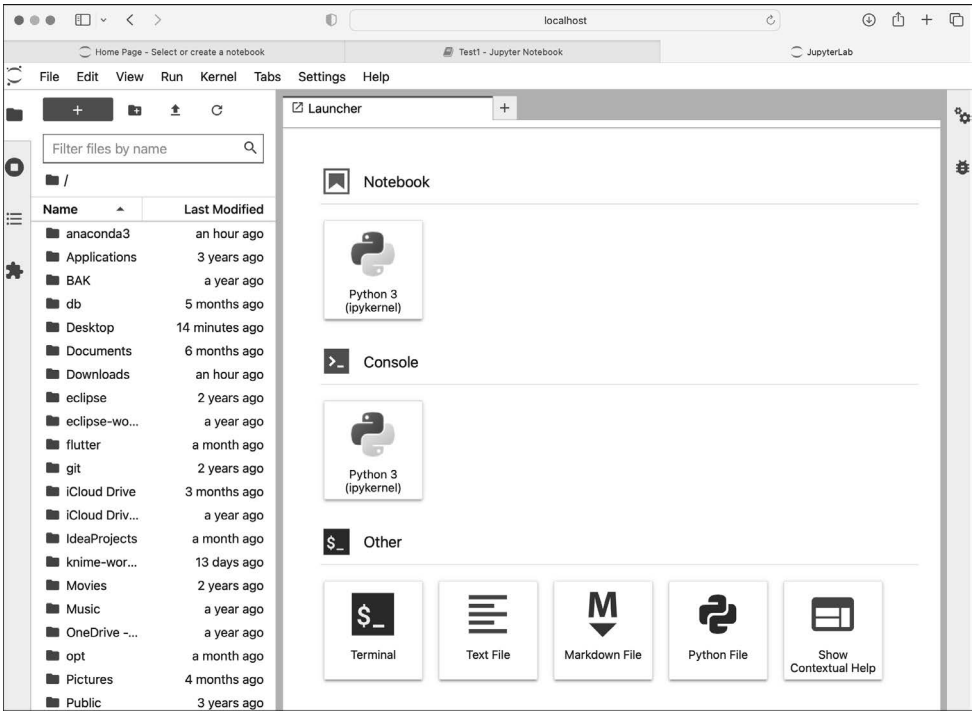


Abbildung 2.4 JupyterLab Launcher

Hier können Sie mit Notebook • Python 3 (ipykernel) ein neues Notebook starten (siehe Abbildung 2.5) oder aber z. B. die oben erstellte Datei *Test1.ipynb* öffnen. Die Verzeichnisse und Dateien sind links ausgelistet.

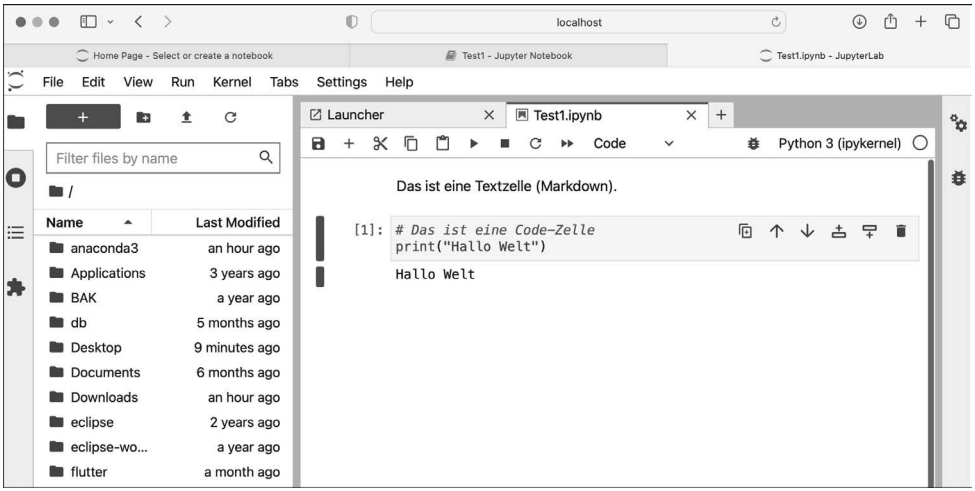


Abbildung 2.5 Erstes Programm mit JupyterLab

Google Colaboraty

Eine interessante Alternative zur Installation von Anaconda bzw. Jupyter Notebook bietet Google mit Colaboraty (*colab.research.google.com*). Bei dieser Lösung müssen Sie lokal gar nichts installieren. Google stellt Speicher- und Rechenkapazitäten kostenlos zur Verfügung. Sie benötigen lediglich einen Browser.

Da wir später KNIME für die visuelle Programmierung nutzen werden, brauchen Sie auch eine lokale Installation. Ich verwende Colab gerne bei rechenintensiven KI-Programmen, weil man hier recht einfach eine GPU anstatt der CPU nutzen kann. Dies bringt eine enorme Performancesteigerung. Die Einrichtung einer lokalen GPU-Unterstützung hängt von Ihrer Grafikkarte ab und ist mit einigem Aufwand verbunden.

Bei dieser Gelegenheit werden wir einige benötigte Module installieren. Mit der Installation von Python haben Sie nicht sofort Zugriff auf alle Funktionalitäten, die diese Programmiersprache bietet. Vielmehr ist Python so aufgebaut, dass Sie weitere benötigte Module bei Bedarf nachinstallieren können. Dieser modulare Aufbau sorgt dafür, dass das Programm nicht unnötig viel Speicherplatz einnimmt.

Starten Sie den Anaconda-Navigator, und gehen Sie in das Menü ENVIRONMENTS. Im Dropdown stellen Sie ALL ein, rechts steht Ihnen ein Suchfeld zur Verfügung (siehe Abbildung 2.6).

Installieren Sie die in Tabelle 2.1 aufgelisteten Module bzw. verifizieren Sie, dass diese installiert sind.

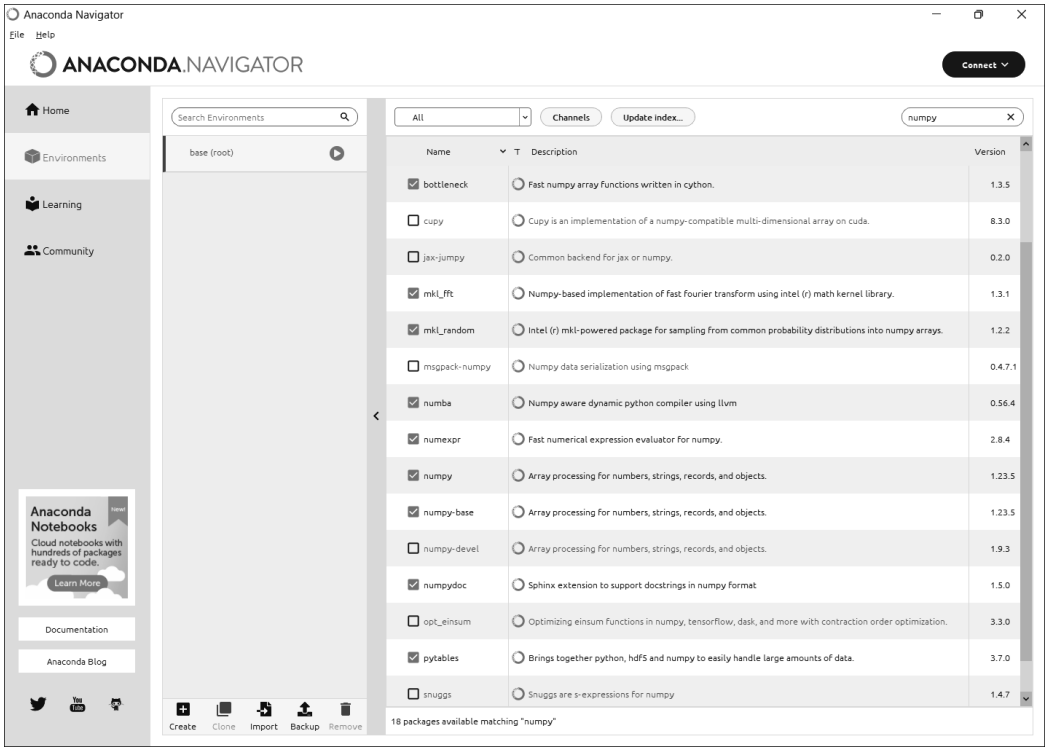


Abbildung 2.6 Modulsuche und -installation über den Anaconda-Navigator

Modul	Erläuterung
NumPy	NumPy bietet Datentypen und Funktionen für die einfachere Handhabung von komplexen Strukturen an, wie z. B. Vektoren und Matrizen.
pandas	pandas ist ebenfalls für komplexere Strukturen und deren einfache Handhabung entwickelt worden. Eine Stärke ist die große Funktionalität für Tabellenstrukturen.
Matplotlib	Matplotlib wird für visuelle Analysen und zum Plotten verwendet.
scikit-learn	Enthält viele ML-Algorithmen, die man sehr einfach im eigenen Programm verwenden kann.
Keras	Mit Keras können künstliche neuronale Netze aufgebaut werden.
TensorFlow	Erweitert Keras um weitere Funktionalitäten. Ist sehr performant bei großen und komplexen Datenstrukturen.

Tabelle 2.1 Zu installierende Module

Weitere Module werden wir nach Bedarf installieren. Jetzt wissen Sie, wie Sie die Installation von Modulen durchführen können.

Anaconda und Environments

In Python sind Environments isolierte Umgebungen, in denen Python-Quellcode ausgeführt werden kann. Jedes Environment kann verschiedene Pakete oder Versionen von Paketen installieren. Der Vorteil vom getrennten Environment ist, dass die Installationen in einem Environment nicht die anderen Environments beeinflussen.

Wenn Sie ein wenig Erfahrung mit Python bzw. Anaconda haben, können Sie ein eigenes Environment für Jupyter Notebook anlegen und die Pakete darin installieren. Sie können aber auch das Base-Environment verwenden.

2.2 KNIME

In Kapitel 11 werden wir einige Aufgabenstellungen mit KNIME lösen. Dabei werden Sie einen alternativen Ansatz, nämlich die visuelle Programmierung, kennenlernen. Wenn Sie nur mit der Programmiersprache Python arbeiten wollen, können Sie diesen Abschnitt überspringen. Ich empfehle Ihnen aber, sich mit dieser Art der KI-Programmierung vertraut zu machen.

2.2.1 Installation

Die Software *KNIME Analytics Plattform* können Sie unter *www.knime.com/downloads* für das bevorzugte Betriebssystem herunterladen. Bei Windows und macOS führen Sie das Installationsprogramm aus, die empfohlenen Einstellungen können beibehalten werden. Bei Linux müssen Sie nur das Archiv entpacken (in Ihrem Home-Verzeichnis).

Darstellung mit Linux Wayland

Beim Start von KNIME bekommen Sie einen Hinweis, dass Sie für eine optimale Darstellung Xorg nutzen sollten. Sie können es trotzdem mit Wayland probieren. Wenn die Darstellung nicht zufriedenstellend sein sollte, können Sie sich abmelden und im Anmeldefenster auf Xorg umstellen.

Nach dem Start des Programmes erscheint ein Fenster mit dem Hinweis, den Workspace (in dem Ihre Programme und Konfigurationen gespeichert werden) auszuwählen (siehe Abbildung 2.7). Setzen Sie den Haken bei USE THIS AS THE DEFAULT AND DO NOT ASK AGAIN, und klicken Sie auf LAUNCH.

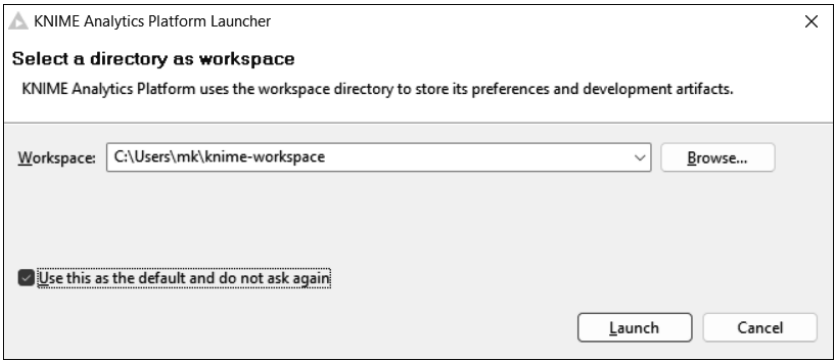


Abbildung 2.7 Auswahl des Workspace

Nachdem Sie den Workspace eingestellt bzw. bestätigt haben, erscheint die KNIME Workbench.

Neue GUI bei Version 5.1

Wenn Sie die aktuelle Version von KNIME herunterladen, ist das User Interface standardmäßig anders aufgebaut als in den Screenshots in diesem Buch dargestellt. Das ist aber überhaupt kein Problem. Sie müssen lediglich eine Einstellung vornehmen, um den Anleitungen in diesem Buch folgen zu können.

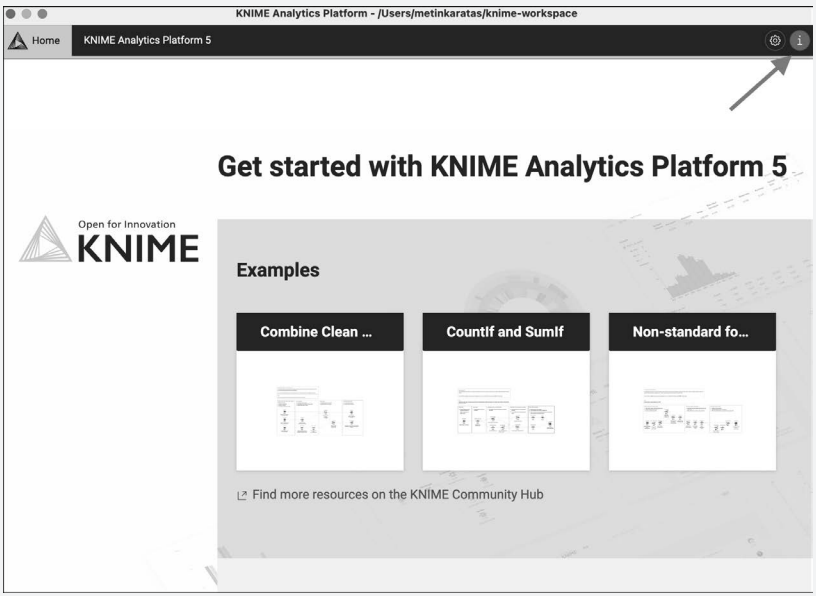


Abbildung 2.8 Neues User Interface

Auf der Startseite in Abbildung 2.8 klicken Sie oben rechts auf das Symbol und wählen SWITCH TO KNIME CLASSIC USER INTERFACE aus (siehe Abbildung 2.9).

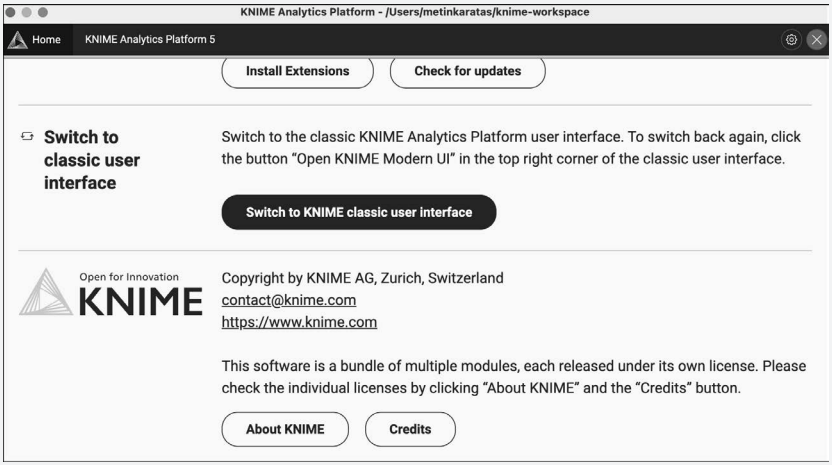


Abbildung 2.9 Umstellung auf klassische Ansicht

Verschaffen Sie sich einen Überblick über den Aufbau der Software. Wenn Sie noch nie mit Entwicklungsumgebungen zu tun hatten, kann das User Interface etwas komplex wirken (siehe Abbildung 2.10). Machen Sie sich keine Sorgen, mit der Zeit werden Sie routiniert Programme erstellen und Einstellungen vornehmen können.

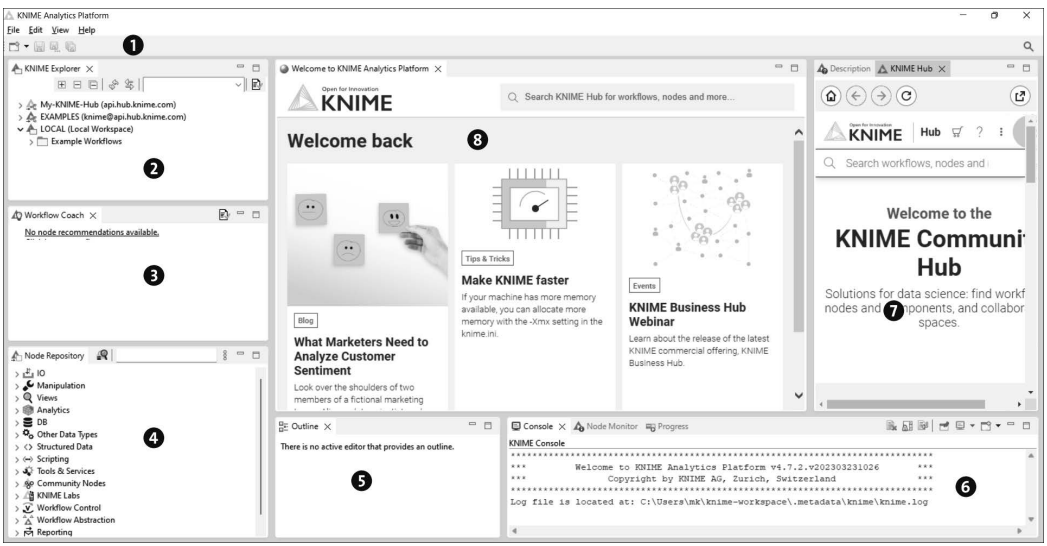


Abbildung 2.10 KNIME Workbench

❶ MenüLeiste: Hier können Sie z. B. zentrale Einstellungen vornehmen, Updates suchen und installieren usw.

❷ KNIME-Explorer: Zugriff auf die Projektstruktur

❸ Workflow-Coach: Wenn Sie dies mit einem Klick aktivieren, werden passende Nodes (Grafikelemente bzw. Bausteine) zu Ihrem Programm empfohlen. Aktivieren Sie es. Das spart Ihnen später viel Zeit bei der Entwicklung.

❹ Node-Repository: Übersicht über alle vorhandenen Nodes

❺ Outline: Bietet eine Gesamtübersicht über den Arbeitsbereich. Wird interessant, wenn Sie große Programme haben.

❻ Konsole und Node-Monitor: Hier erhalten Sie wichtige Ausgaben.

❼ Node-Beschreibung und Hub-Suche: Die Node-Beschreibung liefert Informationen zu selektierten Nodes. Mit der Hub-Suche können Sie z. B. auf fertige Programme (Workflows) der Community zugreifen und diese verwenden.

❽ Workflow-Editor: Das ist der Arbeitsbereich, in dem die Programmierung stattfindet.

2.2.2 Konfiguration

Installieren wir noch weitere benötigte Pakete. Gehen Sie dazu bitte in der Menüleiste auf Help • Install New Software. Bei Work With stellen Sie ein, dass auf allen Seiten gesucht werden soll. Suchen Sie anschließend nach Python und selektieren Sie, wie in Abbildung 2.11 gezeigt:

- KNIME Conda Integration
- KNIME Python Integration
- KNIME Python Scripting extension

Klicken Sie auf Next, akzeptieren Sie die Lizenzbedingungen, und bestätigen Sie mit Finish. Die selektierten Pakete werden heruntergeladen und installiert.

Auf die gleiche Art und Weise installieren Sie bitte folgende Pakete:

- KNIME Deep Learning – Keras Integration
- KNIME Deep Learning – TensorFlow Integration
- KNIME Deep Learning – TensorFlow2 Integration
- KNIME Text Processing
- KNIME Image Processing

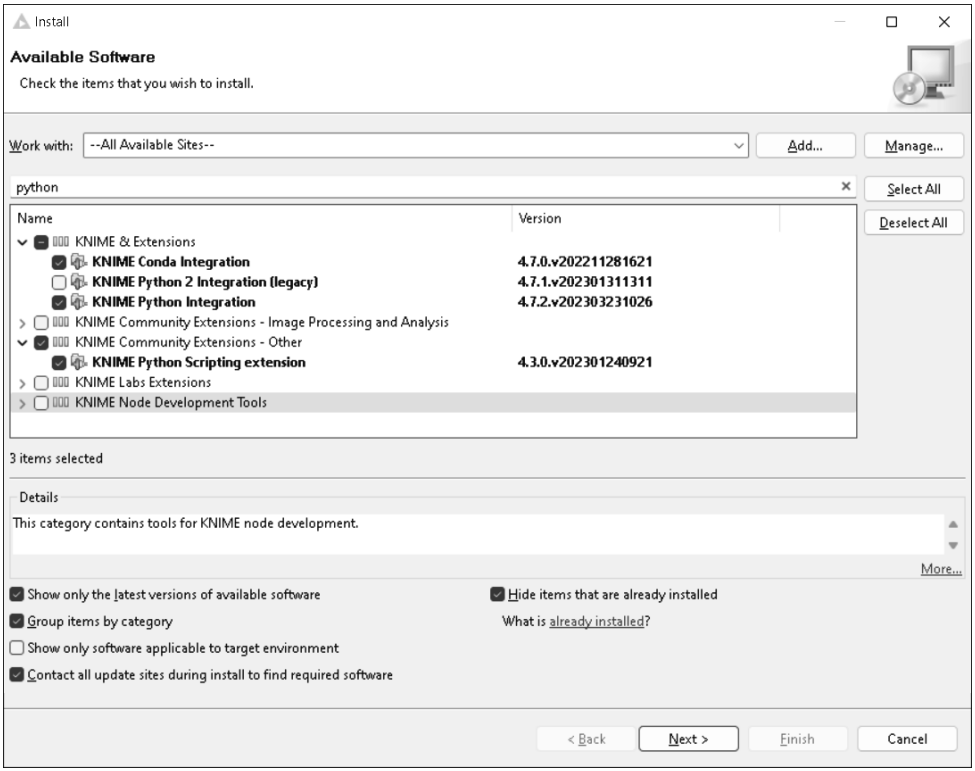


Abbildung 2.11 Installation weiterer Pakete

Wir müssen noch einige Einstellungen für Python, Keras und TensorFlow vornehmen. Dazu gehen Sie über die Menüleiste auf File • Preferences • KNIME • Python. Hier selektieren Sie Conda und warten kurz, bis KNIME alle Informationen zur Installation gesammelt hat. Sicherlich wird eine Warnung angezeigt, dass einige Pakete fehlen. Daher werden wir nun ein Environment anlegen, das alle benötigten Pakete enthält. Klicken Sie dazu auf New environment, und behalten Sie den vorgeschlagenen Namen für die Umgebung bei (hier *py3_knime*).

Damit später Programme mit künstlichen neuronalen Netzen erstellt werden können, muss auch hierfür ein Environment angelegt werden. Dazu gehen Sie wieder über die Menüleiste auf File • Preferences • KNIME • Python Deep Learning. Hier selektieren Sie Use special Deep Learning as defined below, Keras und Conda. Danach müssen Sie sich erneut kurz gedulden, bis KNIME Informationen über die Python-Installation gesammelt hat. Nun klicken Sie beim Punkt Keras auf New Environment, übernehmen den vorgeschlagenen Namen und klicken auf Create new CPU environment. Schließen Sie das Menü (Abbildung 2.13) mit Apply and Close.

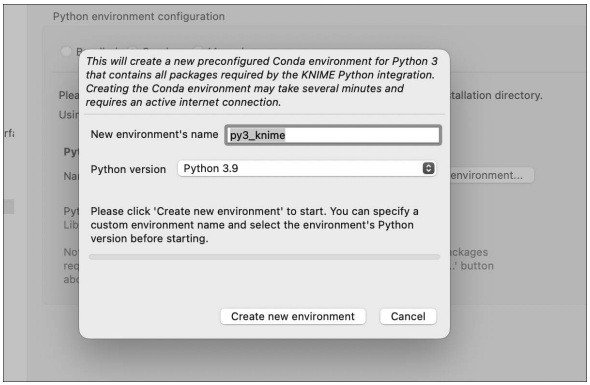


Abbildung 2.12 Einstellung für Python

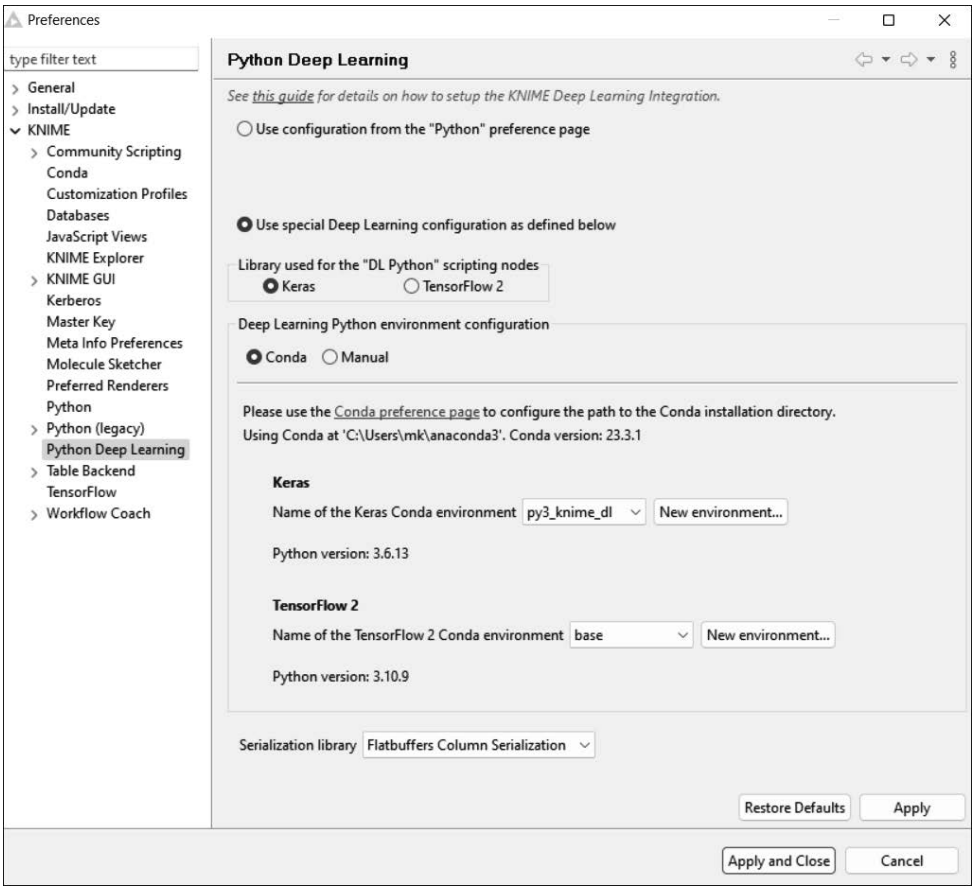


Abbildung 2.13 Einstellung für Python Deep Learning

Erstellen Sie auf die gleiche Art und Weise ebenfalls ein Environment für TensorFlow 2. Damit ist die Entwicklungsumgebung für die visuelle Programmierung vorbereitet.

Troubleshooting

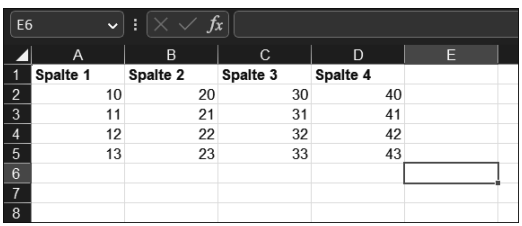
Das Anlegen von Environments über KNIME funktioniert leider nicht immer reibungslos und hängt von den Versionen des Betriebssystems, Anaconda-Navigator, Python und KNIME ab. Falls es zu Fehlermeldungen kommen sollte, probieren Sie erst, Anaconda zu aktualisieren. Taucht das Problem beim Anlegen des Environments für Python auf, können Sie im Dropdown-Menü, bevor Sie CREATE NEW ENVIRONMENT klicken, eine ältere Python-Version (z. B. 3.9) wählen.

Wenn beim Anlegen eines Environments für Python Deep Learning Probleme auftreten und Sie diese nicht selbst lösen können, ist das kein Weltuntergang. Widmen Sie sich erst den Python-Beispielen. Bis Sie sich zu Kapitel 11, »Visuelle Programmierung mit KNIME«, durchgearbeitet haben, gibt es bestimmt wieder aktuellere Versionen der benötigten Tools. Und falls es trotzdem nicht funktioniert: Ich werde Ihnen zeigen, wie Sie in KNIME Python-Bausteine verwenden können. Für die Datenverarbeitung kommen somit grafische Bausteine zum Einsatz, das künstliche neuronale Netz wird in einem Python-Baustein mit ein paar Zeilen Code aufgebaut. Dafür ist es notwendig (wenn Sie kein Environment für Deep Learning anlegen konnten), das Environment für Python in Base umzustellen, da die Pakete für Keras und TensorFlow in diesem Environment installiert wurden.

Es ist ratsam, bei Problemen rund um KNIME im KNIME-Forum (*https://forum.knime.com*) zu recherchieren bzw. nachzufragen.

2.2.3 Test

Die Installation werden wir mit einer Excel-Datei testen, welche Sie z. B. mit *MS Excel* oder *LibreOffice Calc* erstellen können (siehe Abbildung 2.14).



	A	B	C	D	E
1	**Spalte 1**	**Spalte 2**	**Spalte 3**	**Spalte 4**	
2	10	20	30	40	
3	11	21	31	41	
4	12	22	32	42	
5	13	23	33	43	
6					
7					
8					

Abbildung 2.14 Tabelle mit Zahlenwerten

Führen Sie im KNIME-Explorer einen Rechtsklick auf LOCAL aus, und wählen Sie im Kontextmenü New Workflow Group, benennen sie *Test*, und klicken Sie auf Finish.

Anschließend legen Sie in dieser Workflow-Gruppe (WORKFLOW-GROUP) einen Workflow mit dem Namen *Test-1* an. Die XLSX-Datei kopieren wir per Drag-and-Drop in die Workflow-Gruppe (das ist ein Ordner). Die Ordnerstruktur sollte wie in Abbildung 2.15 aussehen.

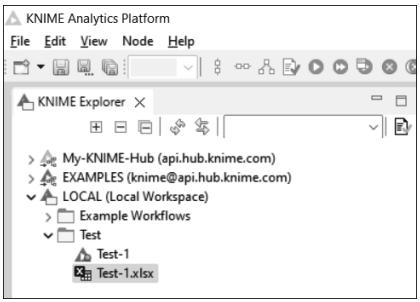


Abbildung 2.15 Dateistruktur im KNIME-Explorer

Jetzt können Sie wieder per Drag-and-Drop die XLSX-Datei in den Workflow-Editor ziehen. Es erscheint ein Anlege-Assistent, wie in Abbildung 2.16 dargestellt, der meistens alle Einstellungen richtig erkennt. Sie müssen nur noch mit OK bestätigen.

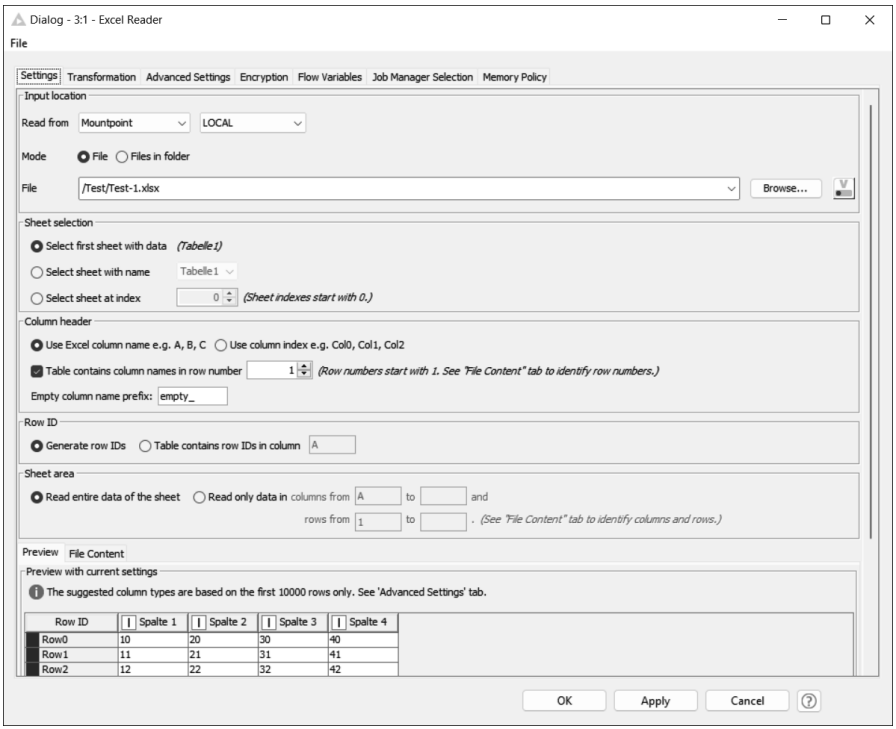


Abbildung 2.16 KNIME-Anlege-Assistent

Der Baustein bzw. Node Excel READER erscheint mit einem gelben Punkt. Mit einem Rechtklick und Execute führen Sie den Baustein aus. Der Punkt wechselt auf grün, und in der Ausgabe wird der Inhalt der Datei ausgegeben (siehe Abbildung 2.17). Mit Rechtsklick und Reset können Sie den Baustein zurücksetzen.

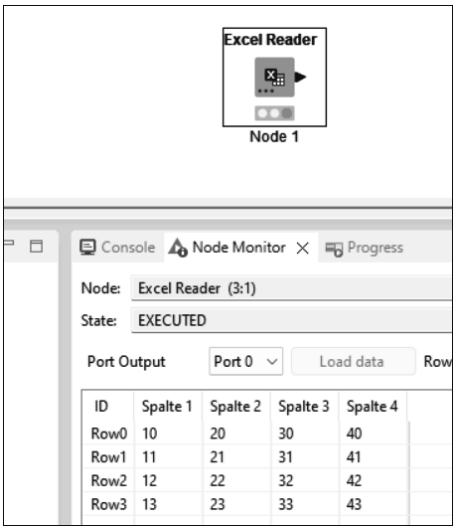


Abbildung 2.17 Ausgabe des Nodes »Excel Reader«

Ihre Entwicklungsmaschine ist somit konfiguriert, getestet und einsatzbereit. Jetzt können Sie mit der Entwicklung von KI-Modellen beginnen.

Kapitel 11
Visuelle Programmierung mit KNIME

Eine KI programmieren, ohne eine Programmiersprache zu lernen, oder aber optional Quellcode einbinden? KNIME macht es möglich.

Worum geht es in diesem Kapitel?

- Einführung in die visuelle Programmierung
- Regression und Klassifizierung mit künstlichen neuronalen Netzen
- Regression und Klassifizierung mit XGBoost
- Bildklassifizierung mit vortrainierten Modellen
- Transfer Learning
- Autoencoder
- Textklassifizierung
- automatische Erstellung von KI-Modellen mit AutoML
- Clusteranalyse
- Zeitreihenanalyse
- Textgenerierung

Mit KNIME können Sie verschiedenste KI-Modelle mithilfe von grafischen Bausteinen programmieren. Einer der Vorteile von visueller Programmierung ist die Übersichtlichkeit. Selbst wenn das Programm von jemand anderem entwickelt worden ist, können Sie sich relativ schnell einarbeiten, der Datenfluss ist einfach nachvollziehbar. Dabei ist KNIME alles andere als eine Spielerei. Viele Firmen und Forschende nutzen diese Software für professionelle Aufgaben.

Mit KNIME ist es auch sehr einfach, mit Daten aus verschiedenen Quellen zu arbeiten. Bei Programmiersprachen wie Python müssen Sie, je nach Datenquelle, verschiedene Module importieren und mithilfe dieser die Daten einlesen. Für Dateiformate wie Excel oder CSV gibt es eigene Module, die jeweils andere Funktionen zur Verfügung stellen. Wenn die Daten aus einer Datenbank eingelesen werden sollen, müssen Sie sich mit der Datenbanksprache SQL (Structured Query Language) vertraut machen. Bei KNIME gibt es jeweils verschiedene Bausteine für die Datenquellen, wie z. B. die Bausteine Excel

Reader oder CSV Reader. Sogar für Datenbanken gibt es Bausteine, sodass Sie ohne SQL-Kenntnisse die Daten einlesen können. Nachdem die Daten aus der jeweiligen Quelle eingelesen worden sind, kann man sie auf eine einheitliche Art und Weise verarbeiten und für die Entwicklung von KI-Modellen nutzen. Als KI-Entwicklerin bzw. KI-Entwickler müssen Sie sich nicht erst aufwändig mit der Anbindung der Datenquelle oder dem Import der Daten beschäftigen. Sie können auch Python-Quellcode einbinden, wo es Ihnen sinnvoll erscheint.

Python-Bausteine als Alternative für Keras- und TensorFlow-Bausteine

In Kapitel 2, »Installation«, wurden Keras und TensorFlow Environments für Python Deep Learning erstellt. Falls Sie an Ihrem Rechner Probleme bei der Konfiguration haben, stellen Sie alle Environments aus »Base« um, da in diesem Environment alle benötigten Module installiert wurden. Dann können Sie anstelle von Keras- und TensorFlow-Bausteinen einen Python-Baustein mit ein paar Zeilen Code verwenden. Das Laden und Verarbeiten von Daten kann hingegen mit anderen Bausteinen realisiert werden, wie Sie an den Beispielen in diesem Kapitel sehen werden.

Die Programme, die in diesem Buch vorgestellt werden, können Sie selbst programmieren oder aber die zur Verfügung gestellten Programme importieren. Nach dem Import muss der Pfad zur jeweiligen Datenquelle (CSV-Datei) aktualisiert werden. Der Import des Programmes erfolgt über Rechtsklick auf den Zielort im KNIME Explorer (*LOCAL* oder einem Unterordner, der ebenfalls mit Rechtsklick und Auswahl von New Workflow Group erstellt werden kann). Dann wollen wir mal loslegen.

11.1 Einfache künstliche neuronale Netze

Wenden wir das Erlernte über künstliche neuronale Netze (KNN) hier an. Vergleichen Sie die KNIME-Programme jeweils mit den Python-Programmen, die wir bereits in vorherigen Kapiteln erstellt haben.

11.1.1 Klassifizierung

Der erste Workflow soll mithilfe von einem KNN Schwertlilien klassifizieren. Anhand der Länge und Breite der Blätter soll die Unterart der Schwertlilie bestimmt werden. Für diese Problemstellung haben wir bereits mit KNN und XGBoost KI-Modelle in der Programmiersprache Python entwickelt. Bei diesem ersten Programm werden wir uns die verwendeten Bausteine (*Nodes*) genau anschauen (siehe Abbildung 11.1).

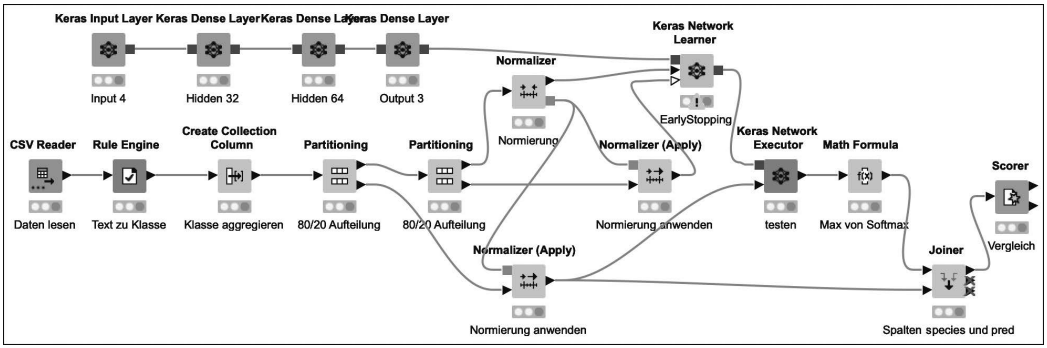


Abbildung 11.1 Das Programm »01-iris.knwf«

Datenvorbereitung

Fangen wir mit der Datenquelle an. Sie können den Baustein CSV-Reader in den Workflow-Editor ziehen und mit Rechtsklick und Auswahl von Configure die gewünschten Einstellungen vornehmen. Alternativ können Sie auch die CSV-Datei in den Workflow-Editor per Drag-and-Drop ablegen. Dann wird automatisch der passende Baustein eingefügt, und es erscheint das Einstellungsfenster (Abbildung 11.2).

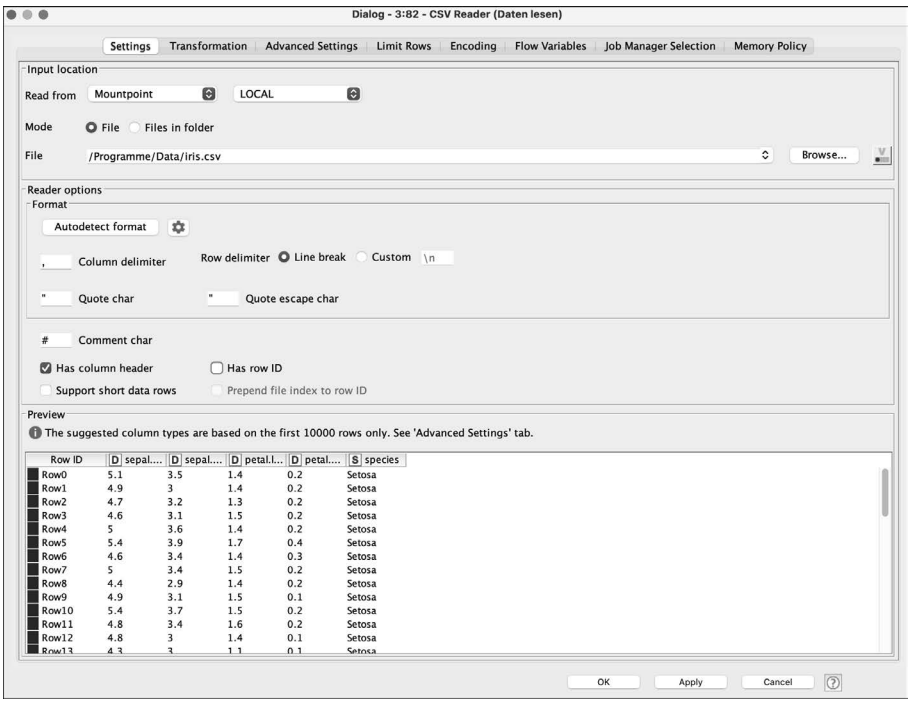


Abbildung 11.2 Das Einstellungsfenster für die CSV-Datei

Die wichtigsten Einstellungen, die vorzunehmen sind:

- der Pfad zur Datei (FILE)
- das Trennzeichen zwischen den Spalten (COLUMN DELIMITER)
- Angabe, ob die Tabelle Spaltenüberschriften hat (HAS COLUMN HEADER)
- Angabe, ob die Tabelle eine Spalte zur Indexierung hat (HAS ROW ID). Wenn die Tabelle selbst eine Spalte z. B. mit aufsteigender Nummerierung hat, um die Zeilen eindeutig zu kennzeichnen, kann diese Spalte verwendet werden. Sonst wir wird eine neue, temporäre Spalte mit eindeutigen Indizes erstellt.

Die Statusanzeige des Bausteins ist nach der Konfiguration gelb. Mit einem Rechtsklick und Auswahl von EXECUTE können Sie den Baustein ausführen, dann wechselt die Statusanzeige auf grün. Alternativ kann der Baustein selektiert und oben in der Menüleiste der PLAY-Button geklickt werden. Im Node-Monitor sehen Sie die Ausgabewerte des Bausteins (Abbildung 11.3).

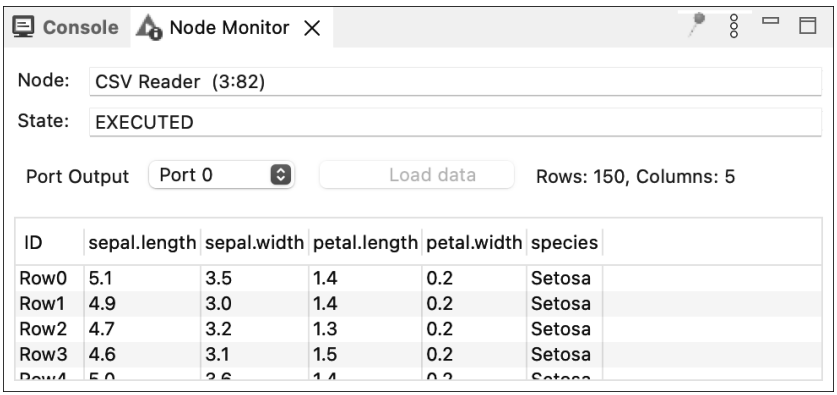


Abbildung 11.3 Ausgabe des Node-Monitors zu »CSV-Reader«

Mit den Daten am Ausgang des Bausteins CSV READER werden wir weiterarbeiten. Ab hier ist es egal, aus welcher Datenquelle die Daten stammen, sie liegen im Ergebnis immer in Tabellenform vor.

Damit das KNN mit den Eingangsdaten rechnen kann, müssen alle dafür benötigten, nicht nummerischen Spalten in Zahlen transformiert werden. Dies betrifft hier nur die Spalte *species*. In diesem Beispiel werden wir dafür den Baustein RULE ENGINE verwenden. Die Ausdrücke, die in diesem Baustein eingegeben werden können, haben eine ungewöhnliche Syntax, wie Sie in Abbildung 11.4 sehen.

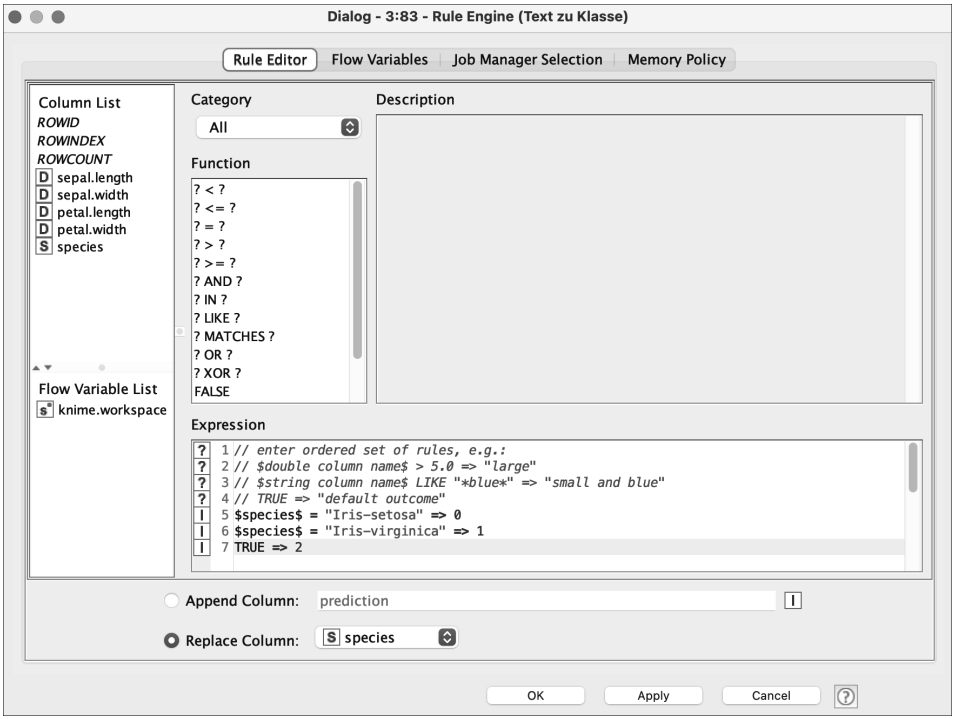


Abbildung 11.4 Das Einstellungsfenster von »Rule Engine«

Die ersten Zeilen enthalten einige auskommentierte Beispiele. Im linken Fenster sehen Sie die Variablen, die Sie verwenden können. Wichtig sind hier die Spaltenüberschriften, insbesondere *species*.

```
$species$ = "Iris-setosa" => 0
$species$ = "Iris-virginica" => 1
TRUE => 2
```

Listing 11.1 Transformation der Daten mit »RuleEngine«

Die erste Zeile bedeutet: Ist der Inhalt der Variable `$species$` »Iris-setosa«, wird die Zahl 0 verwendet. Die zweite Zeile ist entsprechend für die nächste Unterart zu interpretieren. Auch die dritte Zeile könnte genauso für »Iris-versicolor« formuliert werden. Aber hier sehen Sie eine andere Möglichkeit: Bei allen anderen Variableninhalten werden die Werte durch die Zahl 2 ersetzt. Sie können noch einstellen, ob der Inhalt der Spalte *species* überschrieben oder eine neue Spalte mit den Zahlenwerten erstellt werden soll. Nach dem Ausführen des Bausteins ist im Node Monitor zu sehen, dass die Transformation erfolgreich durchgeführt wurde. Beachten Sie in der Abbildung 11.5 die Spalte *species*.

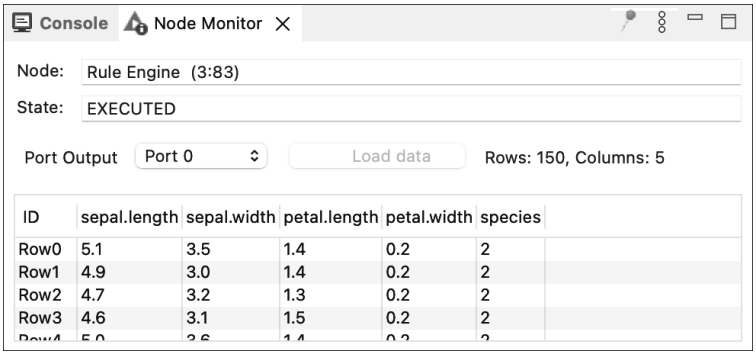


Abbildung 11.5 Ausgabe des Node-Monitors zu »Rule Engine«

Der Baustein für KNN erwartet als Ausgangsdaten (Zielspalte) Listen mit Zahlenwerten, wenn die Softmax-Funktion als Aktivierungsfunktion verwendet werden soll. Dies wird mit dem Baustein CREATE COLLECTION COLUMN realisiert, dessen Einstellungsfenster Abbildung 11.6 zeigt.

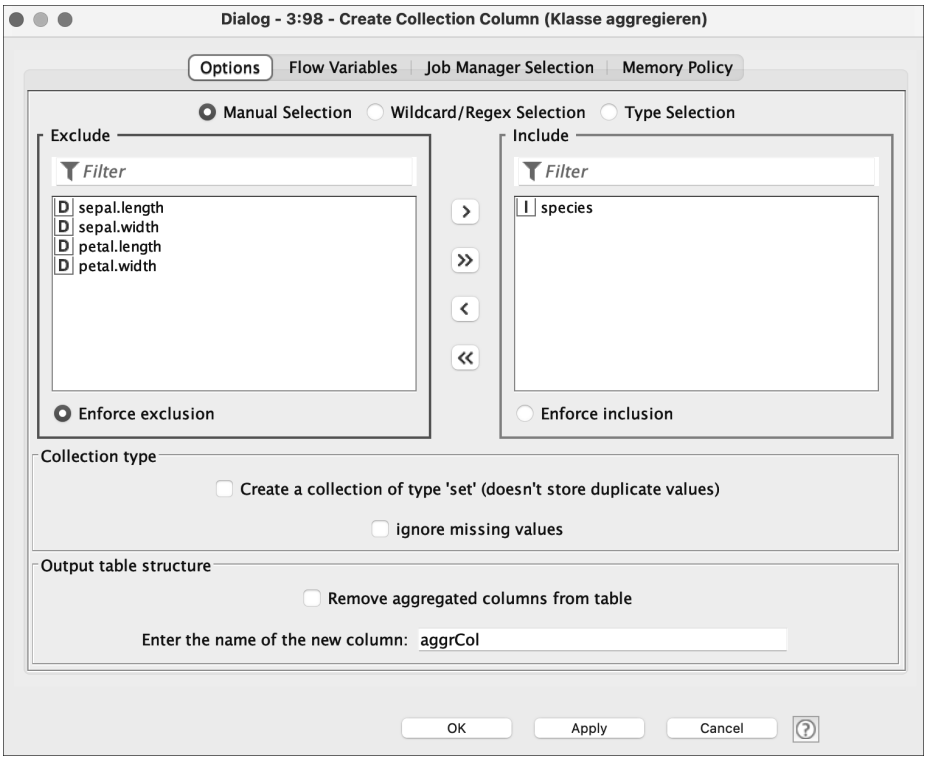


Abbildung 11.6 Das Einstellungsfenster von »Create Collection Column«

Mit diesem Baustein kann man mehrere Spalten in eine Liste zusammenführen. Wir müssen nur die einzelnen Elemente in eine Liste »packen«, damit wir später die Softmax-Funktion verwenden können. Dieser Schritt ist also nur notwendig, weil der Baustein für das KNN mit Softmax als Aktivierungsfunktion diesen Datentyp erwartet. Im Abschnitt INCLUDE werden die Spalten, welche zusammengeführt werden sollen, aufgelistet (hier nur *species*). Weiterhin ergänzen wir die Tabelle wie in Abbildung 11.7 zu sehen um die Spalte *aggrCol*, in welcher die Ergebnisse der Datentypumwandlung (Liste mit jeweils einem Element) gespeichert werden.

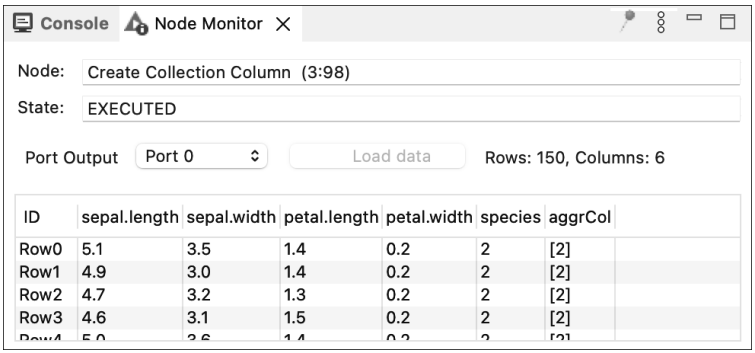


Abbildung 11.7 Ausgabe des Node-Monitors zu »Create Collection Column«

Wenn Sie den Baustein ausführen, sehen Sie im Node-Monitor, dass die Zahlenwerte dieser neuen Spalte in eckigen Klammern stehen (siehe Abbildung 11.7). Dabei wird z. B. aus der Zahl 0 die Liste [0].

Die Daten sind nun eingelesen und vorbereitet. Jetzt können die Daten mit den Bausteinen PARTITIONING in Traings-, Test- und Validierungsdaten aufgeteilt werden (siehe Abbildung 11.8). 80 % der gemischten Daten sollen am oberen Ausgabeknoten ausgegeben werden (Trainingsdaten), der Rest am unteren. Die Daten am unteren Ausgang werden nochmals in Test- und Validierungsdaten aufgeteilt (80 % bzw. 20 %). Wir stellen auch ein, dass die Daten reproduzierbar gemischt werden sollen (USE RANDOM SEED).

Bei Bausteinen mit mehreren Ausgängen können Sie im Node-Monitor einstellen, welche Daten angezeigt werden sollen (PORT OUTPUT).

Mit den Trainingsdaten wird das KNN trainiert, und somit werden die Gewichte aktualisiert. Mit den Testdaten wird die Korrektklassifizierungsrate nach jeder Epoche während der Trainingsphase ermittelt. Die Trainingsdaten dienen bei diesem Programm dazu zu erkennen, ob eine Stagnation auftritt und eine weitere Optimierung der Gewichte ausbleibt. In diesem Fall wird die Trainingsprozedur abgebrochen, um ein Overfitting zu verhindern. Am Ende wird mit den neuen, unbekannten Evaluierungsdaten die Korrektklassifizierungsrate ermittelt.

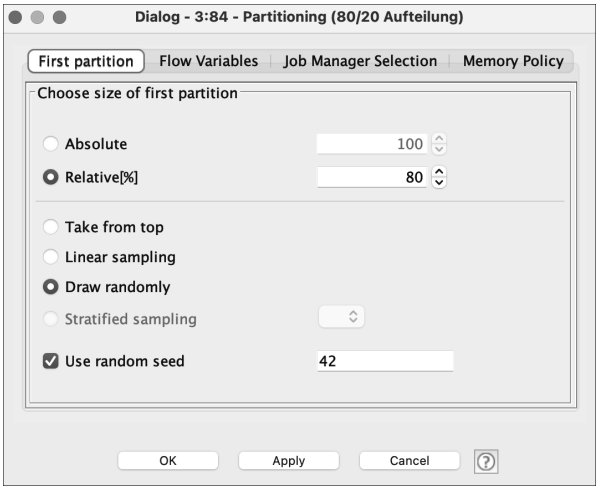


Abbildung 11.8 Das Einstellungsfenster von »Partitioning«

Wie Sie bereits aus vorherigen Kapiteln wissen, können KNN mit normierten Daten bessere Vorhersagen treffen, da sonst die verschiedenen Attribute durch sehr unterschiedliche Wertebereiche der Daten zu einer Verzerrung bei der Aktualisierung der Gewichte führen. Die Eingabedaten werden hier mittels Z-Score normiert (siehe Abbildung 11.9, entspricht dem Standardscaler von sklearn).

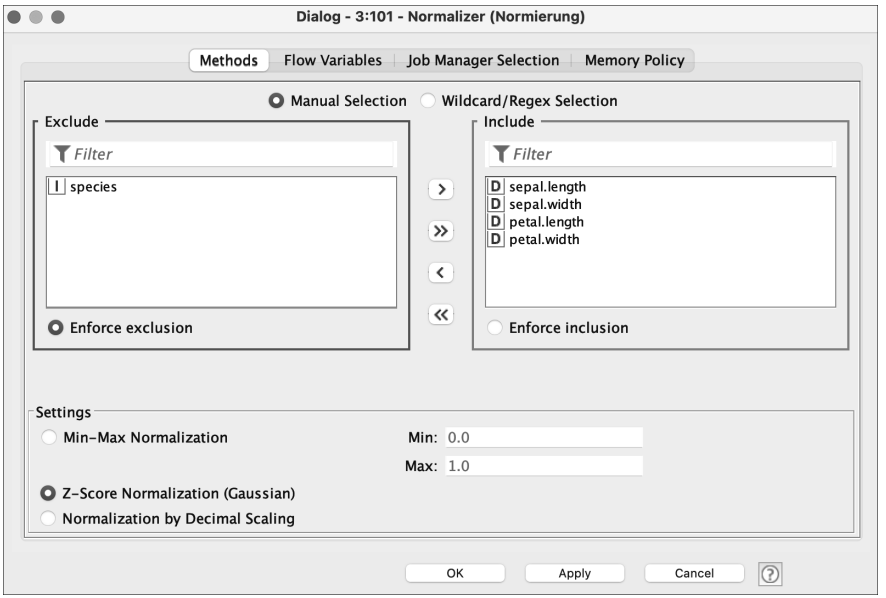
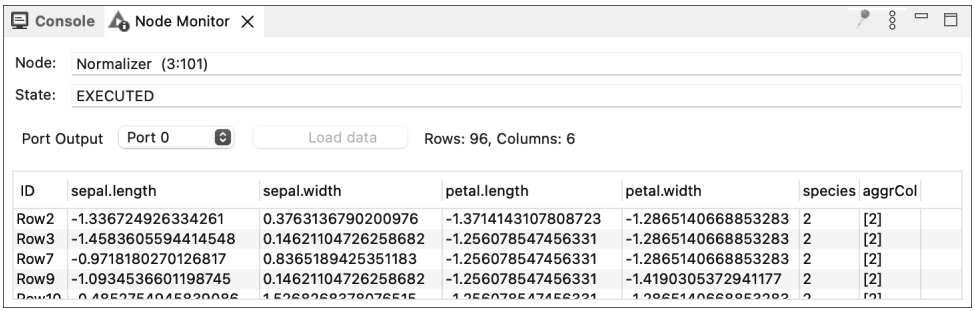


Abbildung 11.9 Das Einstellungsfenster von »Normalizer«

Der Baustein wird nur auf die Trainingsdaten angewandt. Im Node Monitor können Sie wieder die Ausgabedaten des Bausteins betrachten:



ID	sepal.length	sepal.width	petal.length	petal.width	species	aggrCol
Row2	-1.336724926334261	0.3763136790200976	-1.3714143107808723	-1.2865140668853283	2	[2]
Row3	-1.4583605594414548	0.14621104726258682	-1.256078547456331	-1.2865140668853283	2	[2]
Row7	-0.9718180270126817	0.8365189425351183	-1.256078547456331	-1.2865140668853283	2	[2]
Row9	-1.0934536601198745	0.14621104726258682	-1.256078547456331	-1.4190305372941177	2	[2]

Abbildung 11.10 Ausgabe des Node-Monitors zu »Normalizer«

Sie sehen, dass die Test- und Validierungsdaten bei diesem Programm jeweils mit dem Baustein Normalizer (Apply) normiert werden und dass dieser Baustein mit Normalizer verbunden ist. Der Mittelwert und die Standardabweichung werden nur mithilfe der Trainingsdaten ermittelt. Die Test- sowie Validierungsdaten sollen neue, unbekannte Daten simulieren, daher wird die Normierung bei diesen Daten nur angewendet. Dies ist die korrekte Vorgehensweise bei der Normierung von Datensätzen, wenn für praktische Anwendungen programmiert werden soll. Testdaten sollen ja neue, unbekannte Daten simulieren, daher ist es nicht ratsam, diese Werte für die Berechnung von Mittelwert und Standardabweichung zu verwenden.

Aufbau des KNN

Jetzt wird es Zeit, das KNN aufzubauen. Die Verbindungen der bisher verwendeten Bausteine zeigen den Datenfluss an, symbolisiert durch Pfeile. Die Verbindungen der Layer eines KNN sind einfache Linien, welche lediglich eine Verbindung ohne Datenfluss symbolisieren. Am Baustein Keras Network Learner werden wir einige zentrale Einstellungen für das ganze Netz vornehmen (Abbildung 11.11). Dieser Baustein muss vorher mit Keras Input Layer und drei in Reihe geschalteten Keras Dense Layer-Bausteinen verbunden werden. Der erste Layer hat vier Eingangsknoten. Danach folgen zwei Hidden Layer mit jeweils 32 bzw. 64 Knoten, als Aktivierungsfunktion wählen wir ReLu. Der Output Layer hat drei Knoten mit Softmax als Aktivierungsfunktion.

Das Einstellungsfenster von Keras Network Learner ist in folgenden Registern organisiert; die ersten vier sind:

- Input Data: Die vier Eingangsvariablen werden ausgewählt, unter Conversion wird From Number (double) eingestellt.

- Target Data: Die Spalte *col* wird als Eingabespalte ausgewählt, unter Conversion wird From Collection of Number (integer) to One-Hot Tensor eingestellt. Zusätzlich wird Categorical cross entropy als Verlustfunktion eingestellt.
- Options: Die Epochenzahl beträgt 40, Batch Size ist 10. Wählen Sie mit Shuffle training data before each epoche noch aus, dass reproduzierbar gemischt werden soll. Sie müssen hier eine Zahl für Use random seed eingeben, der Wert ist allerdings irrelevant.
- Advanced options: Hier können Sie Early Stopping (über Validation Loss) einstellen. Wenn Sie sich dafür entscheiden, können Sie unter Options die Epochenzahl z. B. auf 100 setzen. In diesem Fall wird der Trainingsprozess abgebrochen, wenn keine Verbesserung mehr auftritt.

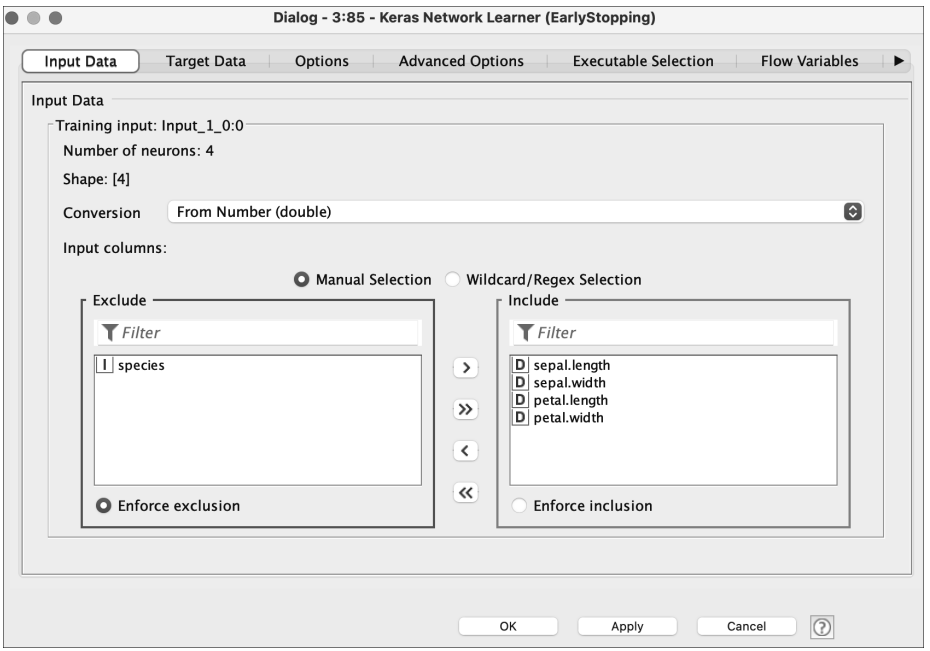


Abbildung 11.11 Das Einstellungsfenster von »Keras Network Learner«

Der Baustein Keras Network Learner wird mit Keras Network Executor verbunden. Auch an diesem Baustein sind einige Einstellungen vorzunehmen, alle unter dem Register Options (siehe Abbildung 11.12):

- Die vier Eingangsvariablen werden ausgewählt, unter Conversion wird From Number (double) eingestellt.

- Mit ADD OUTPUT können Sie den Ausgang hinzufügen. Es muss ausgewählt werden, welcher der Layer als Ausgang definiert werden soll (OUTPUT_1/SOFTMAX:0). Auch bei den Ausgangsdaten muss CONVERSION eingestellt werden, hier TO NUMBER (DOUBLE), wie unten im Bild zu sehen.

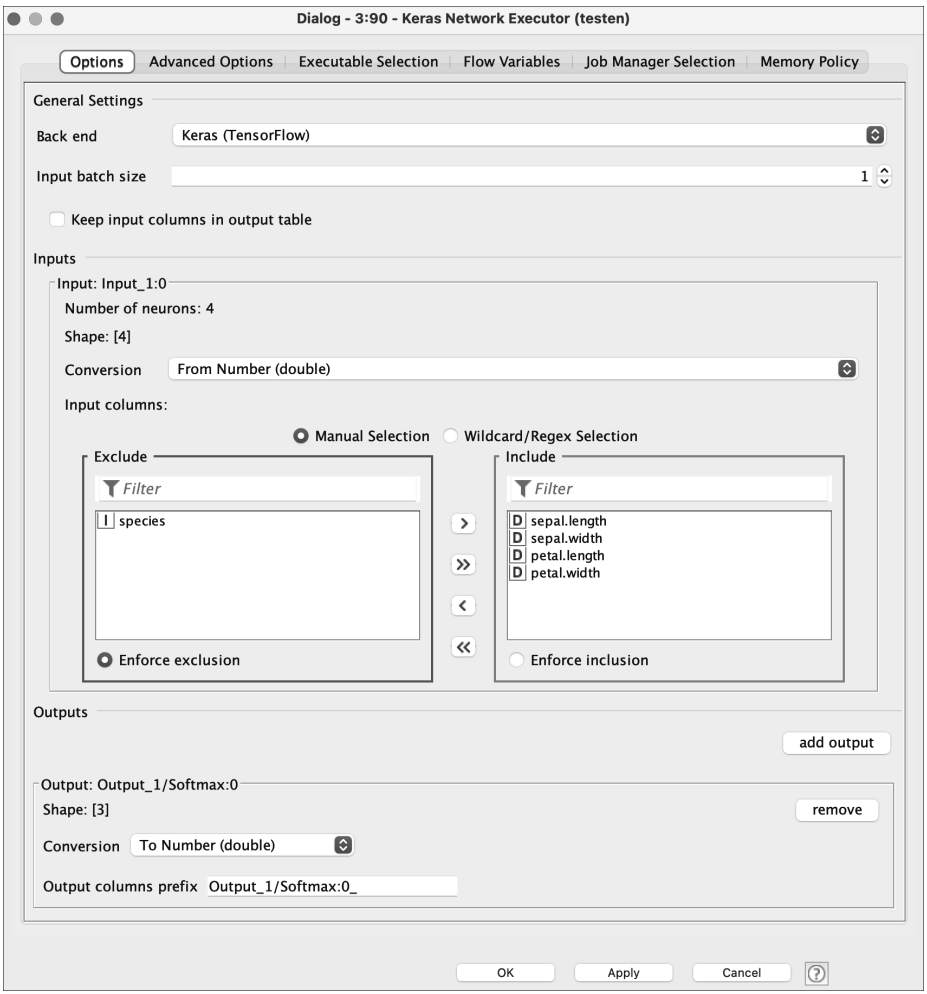


Abbildung 11.12 Das Einstellungsfenster von »Keras Network Executor«

Training, Test und Validierung

Wie läuft das Verfahren für Training, Test und Validierung ab? Die Trainings- und Testdaten werden dem Baustein KERAS NETWORK LEARNER zugeführt. Mit den Trainingsdaten wird trainiert (also die Gewichte aktualisiert), mit den Testdaten nach jeder Iteration

überprüft, wie gut die Vorhersage des KNN ist (die Gewichte werden dabei nicht aktualisiert). Wenn es zu keiner Verbesserung der Vorhersage bei den Testdaten kommt (Early Stopping), wird die Trainings- und Testprozedur abgebrochen. Anschließend werden dem Baustein Keras Network Executor die Validierungsdaten zugeführt. Im Feldeinsatz wäre hier Schluss, wir müssten den Vorhersagen des Bausteins vertrauen. Aber wir kennen ja zu den Validierungsdaten die Zielwerte, daher können wir die tatsächlichen Werte mit den Vorhersagen vergleichen und somit das KNN nochmals validieren.

Der Baustein Math Formula (siehe Abbildung 11.13) vergleicht die drei Werte der Softmax-Funktion des Bausteins Keras Network Executor und ermittelt den Index des Maximalwertes. Dadurch sind als Ergebnisse die Zahlen 0 bis 2 möglich, welche den codierten Unterarten entsprechen.

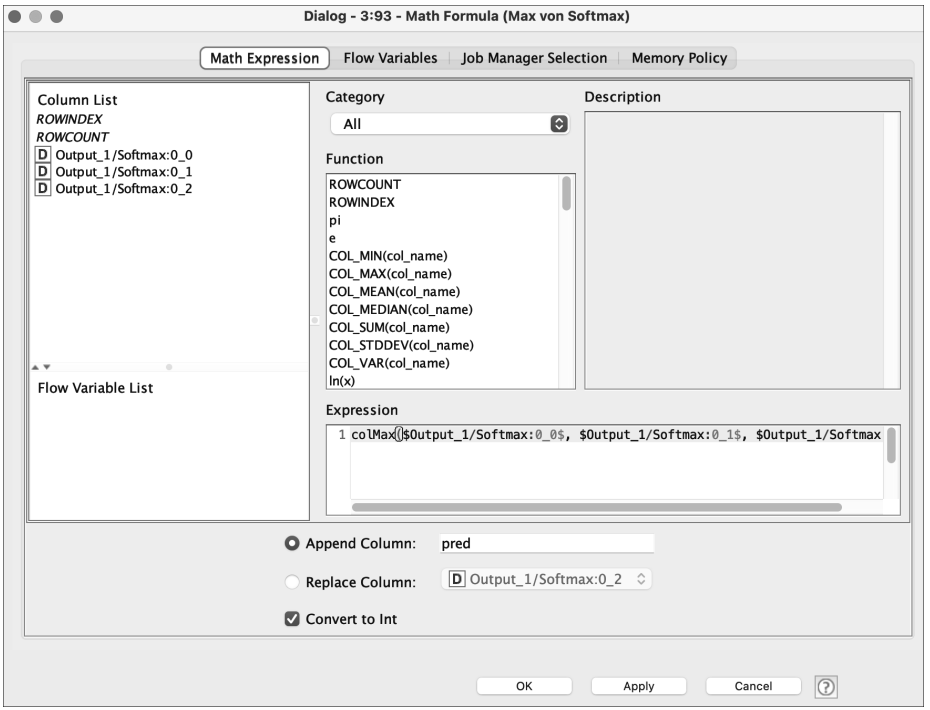
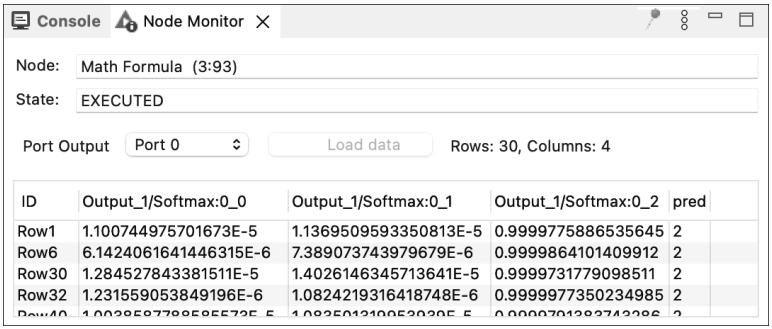


Abbildung 11.13 Das Einstellungsfenster von »Math Formula«

```
colMax($Output_1/Softmax:0_0$, $Output_1/Softmax:0_1$, $Output_1/Softmax:0_2$)
```

Listing 11.2 Ausdruck im Baustein »Math Formula«

Das Ergebnis wird in den Datentyp Integer umgewandelt und in der neuen Spalte *pred* gespeichert (siehe Abbildung 11.14).



ID	Output_1/Softmax:0_0	Output_1/Softmax:0_1	Output_1/Softmax:0_2	pred
Row1	1.100744975701673E-5	1.1369509593350813E-5	0.9999775886535645	2
Row6	6.1424061641446315E-6	7.389073743979679E-6	0.9999864101409912	2
Row30	1.284527843381511E-5	1.4026146345713641E-5	0.9999731779098511	2
Row32	1.2315590538491 96E-6	1.0824219316418748E-6	0.9999977350234985	2

Abbildung 11.14 Ausgabe des Node-Monitors zu »Math Formula«

Diese Spalte *pred* muss mit der Spalte *species* der Validierungsdaten verglichen werden. Mit dem Baustein JOINER wird eine neue Tabelle mit diesen beiden Spalten erstellt. Im Anschluss können wir die Werte dieser Tabelle verwenden, um Berechnungen durchzuführen, in diesem Fall, um die Korrektklassifizierungsrate zu berechnen (siehe Abbildung 11.15).

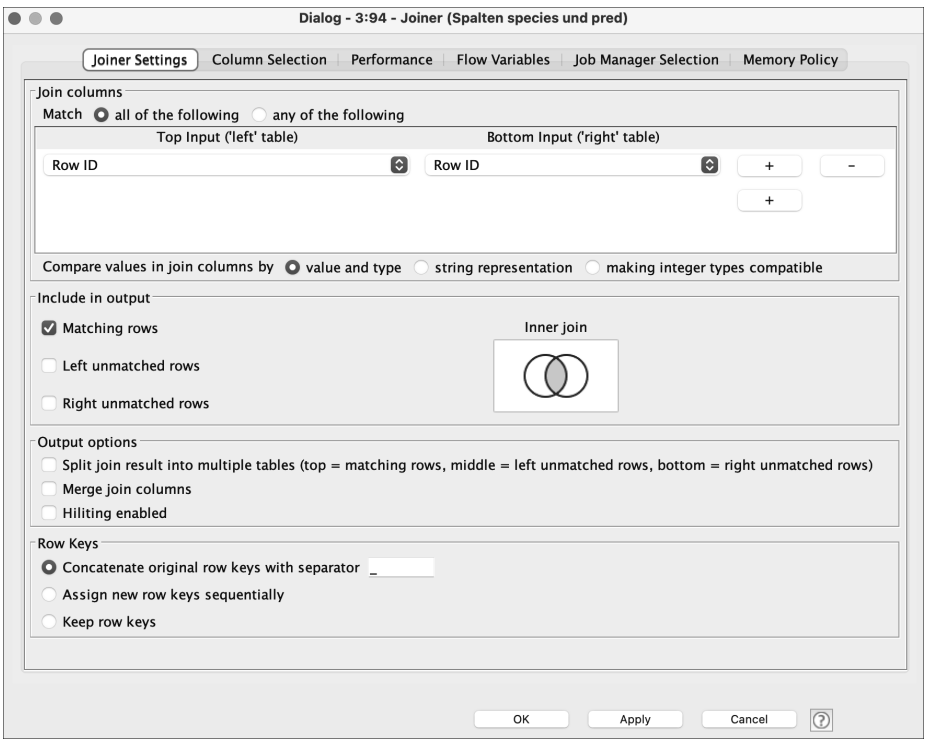


Abbildung 11.15 Das Einstellungsfenster von »Joiner«

Jede Zeile einer Tabelle hat eine eindeutige RowID. Da beide Spalten ursprünglich aus einer einzigen Tabelle stammen, haben die Zeilen der übereinstimmenden Spalten dieselbe RowID. Diese Information aus beiden Tabellen wird als Grundlage für die Zusammenführung verwendet. Weiterhin wird die Schnittmenge (*Inner Join*) der beiden Spalten gebildet. Nur RowIDs, die in beiden Tabellen enthalten sind, werden für die neue Tabelle verwendet. Hat eine Spalte mehr Einträge als die andere oder hat eine Spalte keinen »Gegenpart«, würde diese verworfen werden. Im Register Column Selection des Bausteins Joiner wählen wir *pred* der einen und *species* der anderen Tabelle aus.

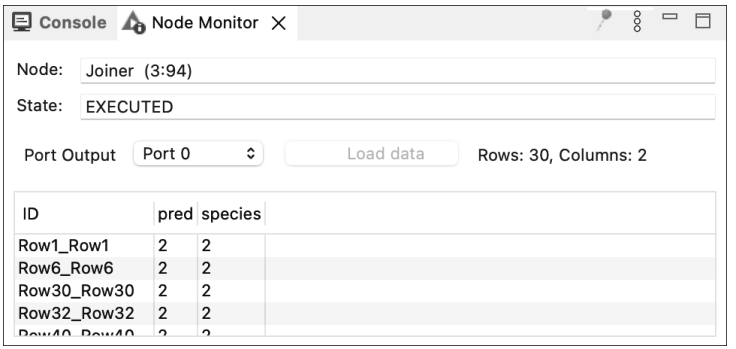


Abbildung 11.16 Ausgabe des Node-Monitors zu »Joiner«

Jetzt könnten wird diese neue Ergebnistabelle Zeile für Zeile durchgehen und uns anschauen, wo eventuell eine Fehlvorhersage generiert worden ist. Das ist aber sehr umständlich und nicht Sinn der Sache. Daher kommt als letzter Baustein Scorer zum Einsatz. Auch hier müssen *pred* und *species* ausgewählt werden. Selektieren Sie den Baustein, und lesen Sie die Beschreibung dafür im rechten Fenster (Node-Beschreibung) durch, um zu erfahren, welche Variable an welchen Eingang angeschlossen werden soll. Nach dem Ausführen erfolgt ein Rechtsklick und die Auswahl von View: Confusion Matrix (siehe Abbildung 11.17).

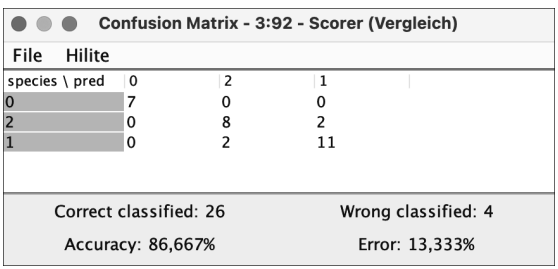


Abbildung 11.17 Confusion Matrix

Mit diesem ersten Entwurf wurde eine Korrektklassifizierungsrate von 87 % erreicht.

11.1.2 Klassifizierung mit Python Node

Sie können die Aufgabenstellung vom vorherigen Abschnitt auch mit Python-Bausteinen lösen (siehe Abbildung 11.18):

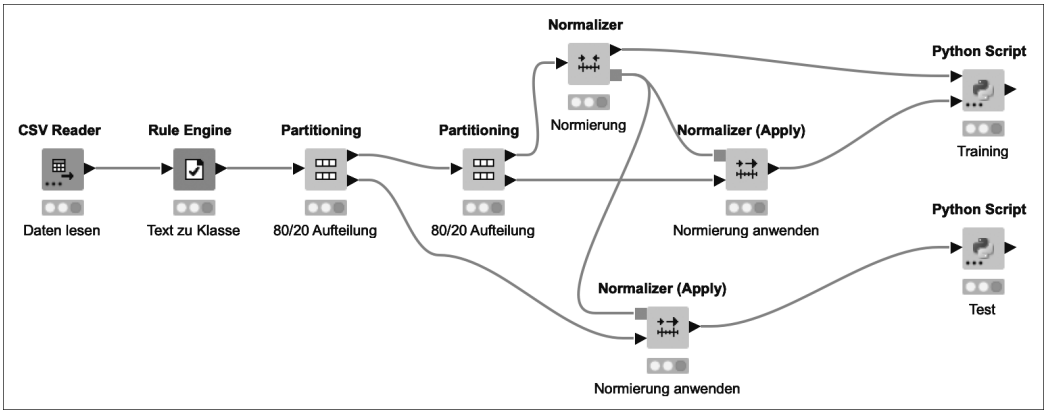


Abbildung 11.18 Das Programm »01-iris-py.knwf«

Der Aufbau des Programmes bleibt ähnlich, es fehlen lediglich die Keras-Bausteine. Dafür kommen zwei PYTHON SCRIPT-Bausteine für Training und Test zum Einsatz.

```
Import knime.scripting.io as knio
import tensorflow as tf
import pandas as pd

# Trainingsdaten
train_col = knio.input_tables[0][4].to_pandas().astype('category')
train_data = knio.input_tables[0][0:4].to_pandas()

# Testdaten
test_col = knio.input_tables[1][4].to_pandas().astype('category')
test_data = knio.input_tables[1][0:4].to_pandas()

# Aufbau Modell
model = tf.keras.Sequential([
    tf.keras.layers.Dense(32, activation=tf.nn.relu, input_dim=4),
    tf.keras.layers.Dense(64, activation=tf.nn.relu),
    tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
```

```
# Konfiguration, Training und Test
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
 metrics=['accuracy'])
cb_early = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=3)
history = model.fit(train_data, train_col, epochs=100,
 validation_data=(test_data, test_col),callbacks=[cb_early])

# Modell speichern
model.save("model1.keras")

# Ausgabe des Ergebnisses
out = [history.history['val_accuracy'][-1]]
df = pd.DataFrame(out, columns=['acc'])
knio.output_tables[0] = knio.Table.from_pandas(df)
```

Listing 11.3 Python-Script für das Training des Modells

Der Quellcode sollte leicht zu interpretieren sein, lediglich das Einlesen und Ausgeben der Daten kann ungewöhnlich erscheinen. Die Datentypen dieser Daten müssen jeweils umgewandelt werden, damit der Datenfluss zu anderen Bausteinen gewährleistet ist. Halten Sie sich nicht zu lange mit dieser Konvertierung auf, Sie werden noch weitere Beispiele sehen, die Sie für Ihre zukünftigen Programme kopieren und verwenden können. Bei Interesse finden Sie auf *http://r-wrk.de/9763-input-output* eine ausführliche Dokumentation zur Verarbeitung bzw. Vorbereitung der Ein- und Ausgabedaten. Noch ein Hinweis: Mit `meine_liste[-1]` lesen Sie das letzte Element einer Liste aus. In diesem Programm wird nur der letzte Wert der Evaluierung ausgegeben.

```
Import knime.scripting.io as knio
import tensorflow as tf
import pandas as pd

# Validierungsdaten laden
val_col = knio.input_tables[0][4].to_pandas().astype('category')
val_data = knio.input_tables[0][0:4].to_pandas()

# Modell laden
loaded_model = tf.keras.saving.load_model("model1.keras")

# Evaluation
val_loss, val_acc = loaded_model.evaluate(val_data, val_col)
```

```
# Ausgabe des Ergebnisses
df = pd.DataFrame([val_acc], columns=['acc'])
knio.output_tables[0] = knio.Table.from_pandas(df)
```

Listing 11.4 Python-Script für Test

Das gespeicherte Modell wird geladen und auf die Testdaten angewendet. Der Wert der Korrektklassifizierung wird ausgegeben, sodass Sie ihn im Node-Monitor sehen können.

11.1.3 Regression

Im nächsten Beispiel soll wieder die Länge des Kelchblattes (Sepalum) ermittelt werden. Bei der Vorhersage einer stetigen Zahl mittels KNN führt ebenfalls die Normierung der Daten wie bei der Klassifizierung oft zu besseren Ergebnissen. Wir werden von dem KNN den mittleren absoluten Fehler berechnen lassen. Das gibt uns Aufschluss darüber, wie weit die Vorhersage vom tatsächlichen Wert entfernt ist.

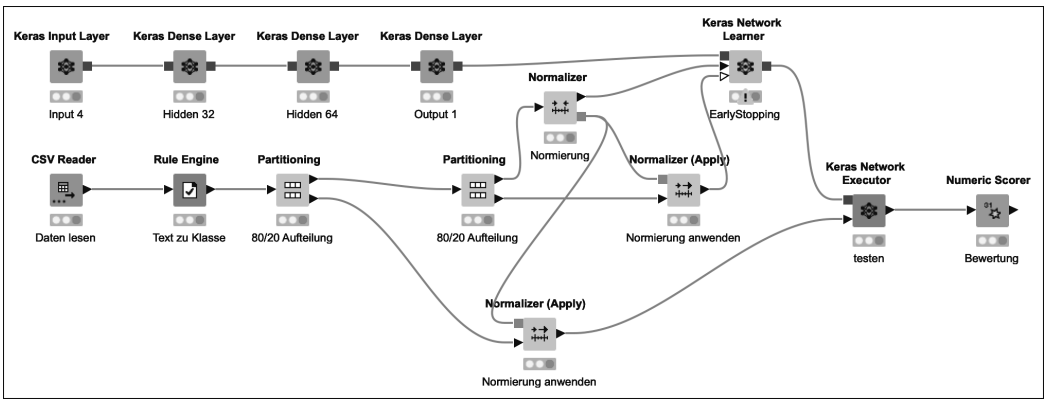


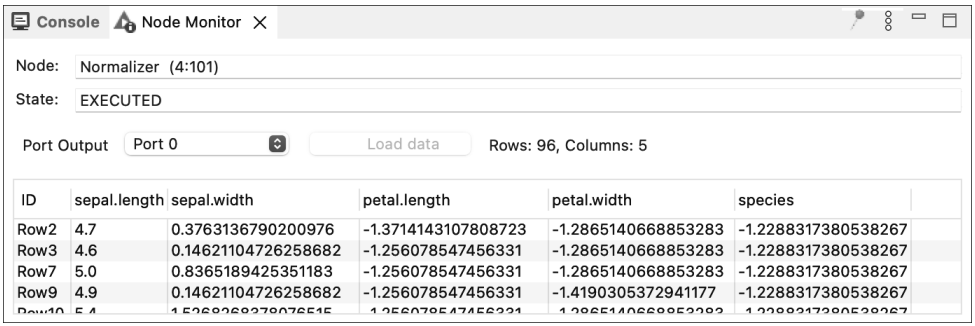
Abbildung 11.19 Das Programm »02-iris.knwf«

Um nicht den ganzen Workflow komplett neu zu erstellen, können Sie den vorherigen kopieren und umbenennen. Vieles bleibt gleich oder ähnlich. Schauen wir uns im Folgenden die Unterschiede an.

Die Zielspalte brauchen wir nicht zu aggregieren, da wir keine Softmax-Funktion benötigen. Unser Ziel ist nicht die Klassifizierung, sondern die Ermittlung des mittleren absoluten Fehlers.

Der Baustein NORMALIZER normiert die Trainingsdaten, die Spalte für die Länge des Kelchblattes (Zielspalte) ist von der Normierung ausgeschlossen. Sonst wären die ermit-

telten MAE-Werte des KNN auch normiert und müssten zurückskaliert werden, um die Ergebnisse interpretieren zu können (siehe Abbildung 11.20).



ID	sepal.length	sepal.width	petal.length	petal.width	species
Row2	4.7	0.3763136790200976	-1.3714143107808723	-1.2865140668853283	-1.2288317380538267
Row3	4.6	0.14621104726258682	-1.256078547456331	-1.2865140668853283	-1.2288317380538267
Row7	5.0	0.8365189425351183	-1.256078547456331	-1.2865140668853283	-1.2288317380538267
Row9	4.9	0.14621104726258682	-1.256078547456331	-1.4190305372941177	-1.2288317380538267

Abbildung 11.20 Ausgabe des Node-Monitors zu »Normalizer«

Der Output Layer des KNN hat jetzt auch nur einen Knoten mit einer linearen Aktivierungsfunktion, Ein- und Ausgabewerte dieser Aktivierungsfunktion sind identisch. Man könnte auch sagen, es wird gar keine Aktivierungsfunktion angewendet. Bei dem Baustein können wir die Aktivierungsfunktion nicht deaktivieren, aber mit der Auswahl dieser Funktion erreichen wir genau das.

Beim Baustein Keras Network Learner sind wieder einige Einstellungen bei folgenden Registern vorzunehmen:

- Input Data: Bei Conversion muss From Number (double) eingestellt werden. Die Länge des Kelchblattes wird vom Datensatz ausgeschlossen.
- Target Data: Bei Conversion muss wieder From Number (double) eingestellt werden. Vom Datensatz wird nur die Länge des Kelchblattes verwendet. Als Verlustfunktion wird Mean absolute error eingestellt.
- Options: Die Epochenzahl wird auf 100 und die Batch Size auf 10 gesetzt. Die Daten sollen reproduzierbar gemischt werden. Die Wahrscheinlichkeit für Overfitting ist bei dieser Epochenzahl hoch. Aber im nächsten Punkt wird das Problem gelöst.
- Adavnced Options: Early Stopping wird mithilfe von Training loss (total) ausgewählt. Somit wird der Trainingsprozess abgebrochen, wenn keine Verbesserung mehr auftritt.

Bei dem Baustein Keras Network Executor müssen ebenfalls die entsprechen Eingangsdaten selektiert werden. Im Abschnitt Output wird der Ausgang des letzten Output Layers ausgewählt. Selektieren Sie auch keep input columns in output table. Dann stehen am Ausgang des Bausteins die Ist- und die Sollwerte weiterhin zur Verfügung. Somit brauchen Sie keinen Joiner, um diese Spalten für die Berechnung im Anschluss aus anderen Tabellen zusammenzuführen.

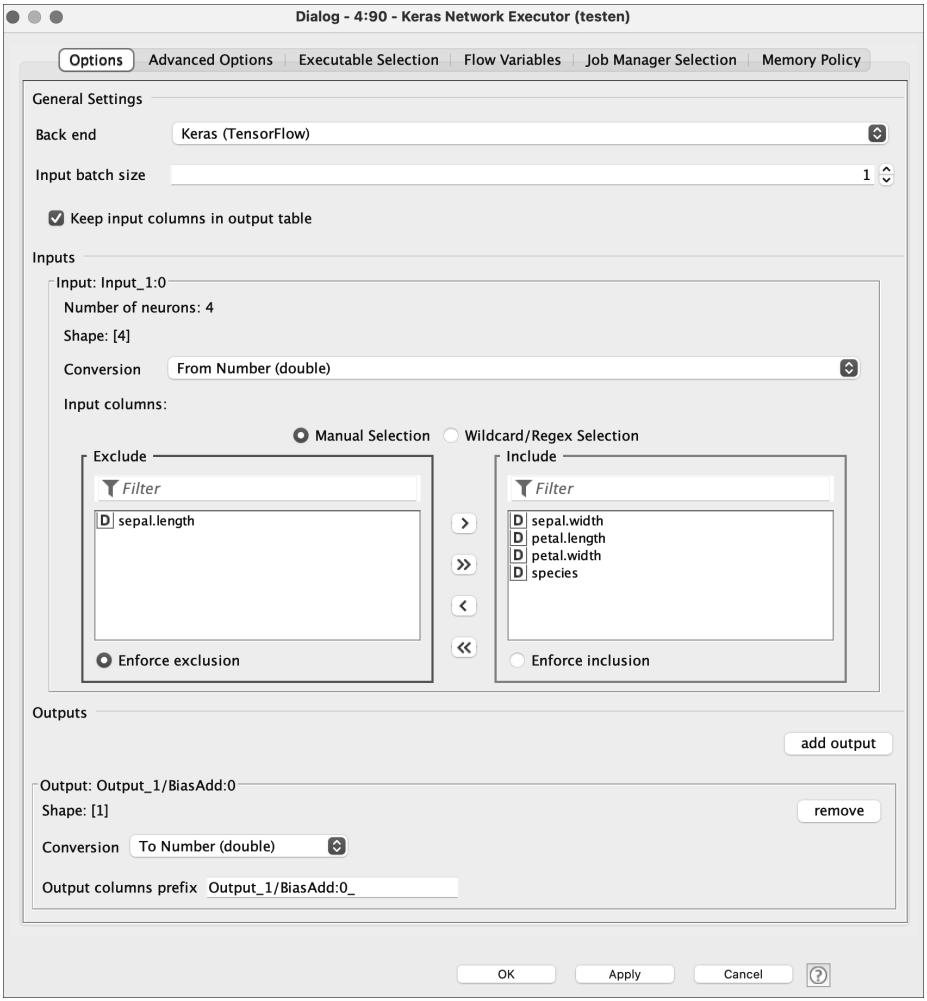


Abbildung 11.21 Das Einstellungsfenster von »Keras Network Executor«

Der Numeric Scorer berechnet unter anderem den MAE, dafür müssen Sie lediglich die zwei Spalten für Ist- und Sollwerte selektieren. Nach dem Ablauf des Programmes können Sie wieder mit Rechtsklick und View: Statistics unter anderem den mittleren absoluten Fehler einsehen (0,45).

Während des Ablaufs oder auch danach können Sie mit Rechtsklick auf Keras Network Learner und Auswahl von Learning Monitor den Trainingsverlauf grafisch analysieren (siehe Abbildung 11.22).

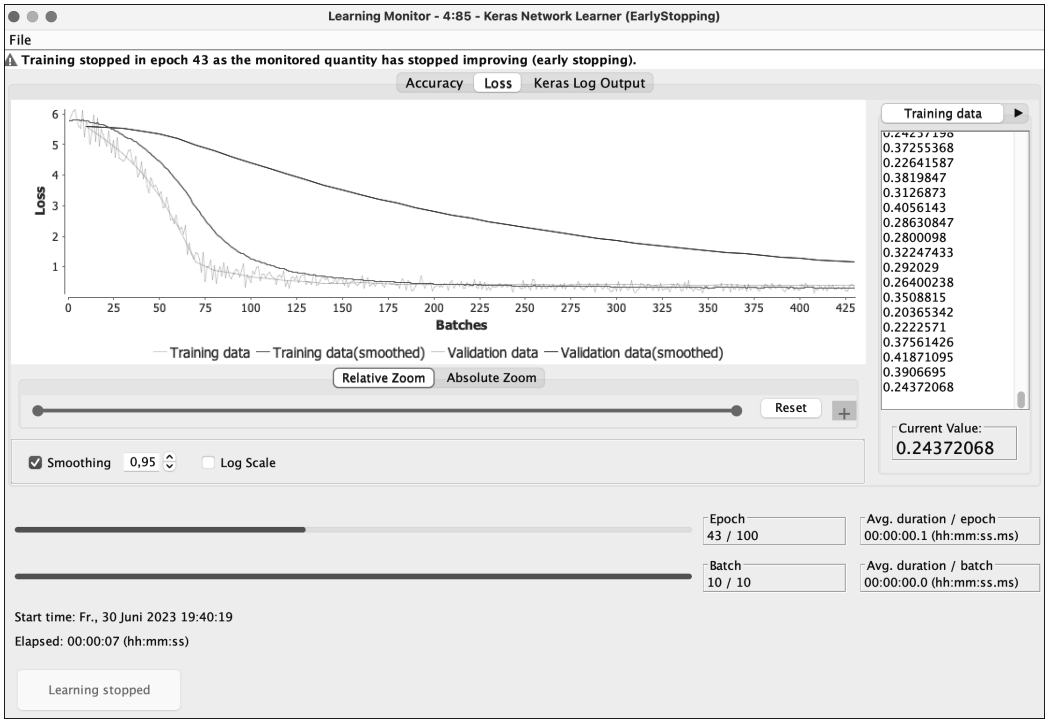


Abbildung 11.22 Learning Monitor

Sie können den Kurvenverlauf glätten und bestimmte Bereiche mit der Maus auswählen, um diese zu vergrößern. Wenn der Trainingsprozess zu lange dauert, können Sie das Training mit dem Button STOP LEARNING beenden. Das führt zu keinem Fehler, der Rest des Programmes kann weiter ausgeführt werden.

11.1.4 Regression mit Python Node

Wenn Sie das Programm mit Python-Bausteinen realisieren wollen, müssen Sie lediglich das KNN selbst programmieren (siehe Abbildung 11.23).

Betrachten wir zuerst den Code für das Training:

```
import knime.scripting.io as knio
import tensorflow as tf
import pandas as pd

# Trainingsdaten
train_col = knio.input_tables[0][0].to_pandas().astype('category')
```

```
train_data = knio.input_tables[0][1:5].to_pandas()

# Testdaten
test_col = knio.input_tables[1][0].to_pandas().astype('category')
test_data = knio.input_tables[1][1:5].to_pandas()

# Aufbau Modell
model = tf.keras.Sequential([
    tf.keras.layers.Dense(32, activation=tf.nn.relu, input_dim=4),
    tf.keras.layers.Dense(64, activation=tf.nn.relu),
    tf.keras.layers.Dense(1)
])

# Konfiguration, Training und Test
model.compile(optimizer='adam', loss='mae', metrics=['mae'])
cb_early = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=3)
history = model.fit(train_data, train_col, epochs=100,
 validation_data=(test_data, test_col),callbacks=[cb_early])

# Modell speichern
model.save("model2.keras")

# Ausgabe des Ergebnisses
out = [history.history['val_mae'][-1]]
df = pd.DataFrame(out, columns=['mae'])
knio.output_tables[0] = knio.Table.from_pandas(df)
```

Listing 11.5 Python-Script für Training

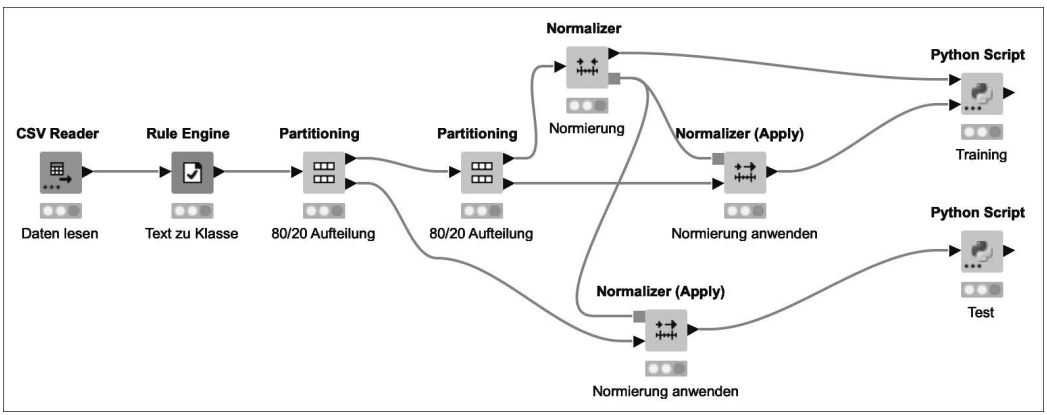


Abbildung 11.23 Das Programm »02-iris-py.knwf«

Auch dieses Modell wird trainiert und gespeichert. Fühlen Sie sich bitte frei, die Hyperparameter zu verändern.

```
import knime.scripting.io as knio
import tensorflow as tf
import pandas as pd

# Validierungsdaten laden
val_col = knio.input_tables[0][0].to_pandas().astype('category')
val_data = knio.input_tables[0][1:5].to_pandas()

# Modell laden
loaded_model = tf.keras.saving.load_model("model2.keras")

# Evaluation
val_loss, val_mae = loaded_model.evaluate(val_data, val_col)

# Ausgabe des Ergebnisses
df = pd.DataFrame([val_mae], columns=['mae'])
knio.output_tables[0] = knio.Table.from_pandas(df)
```

Listing 11.6 Python-Script für Test

Die Testdaten werden ausgewertet und ausgegeben. Im Node-Monitor kann das Ergebnis betrachtet werden.

11.2 XGBoost

In diesem Abschnitt werden wir Regression und Klassifizierung mit XGBoost realisieren. Sie erinnern sich noch: KI-Modelle, die auf XGBoost basieren, sind übersichtlicher, und die Daten müssen nicht aufwändig vorbereitet werden. Auch erreicht man mit diesen Modellen auf einfache Art und Weise sehr gute Ergebnisse. Zuvor muss diese Bibliothek installiert werden. Sie können allgemein Bibliotheken über Help • Install New Software... installieren. Suchen Sie nach XGBoost, und installieren Sie »KNIME XGBoost Integration«.

11.2.1 Klassifizierung

Mit dem Programm *03-iris* werden wieder Schwertlilien klassifiziert. Vergleichen Sie den Aufbau mit dem Programm *01-iris*. Sie sehen, wie einfach und übersichtlich das Programm mit XGBoost ist (siehe Abbildung 11.24).

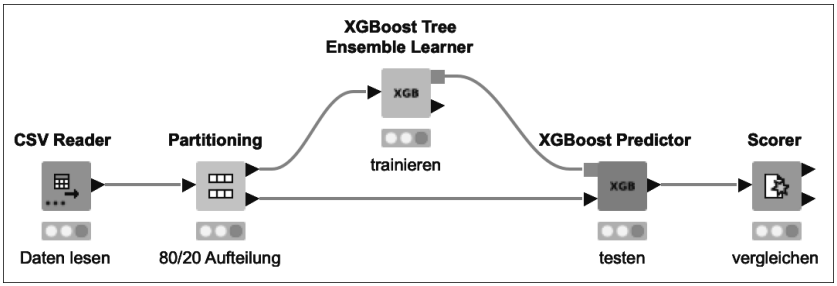


Abbildung 11.24 Das Programm »03-iris.knwf«

Bei dem Baustein XGBoost Tree Ensemble Learner müssen zentrale Einstellungen vorgenommen werden (siehe Abbildung 11.25).

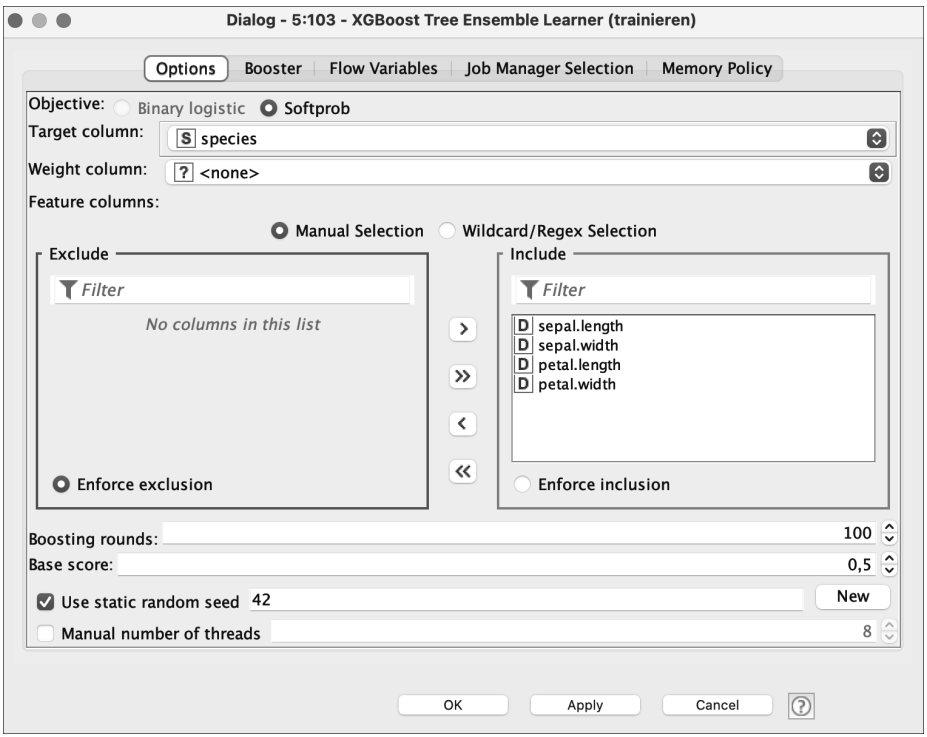
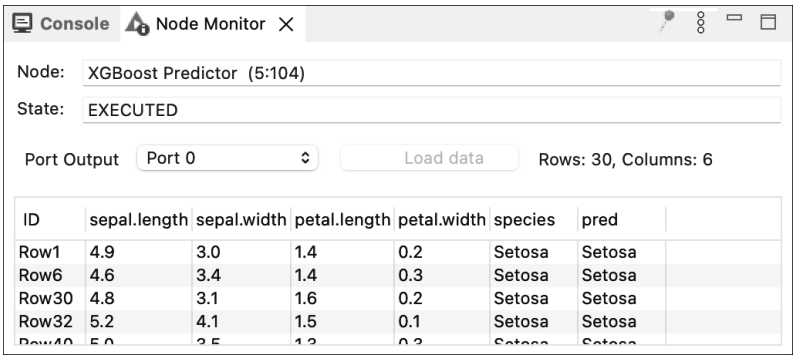


Abbildung 11.25 Das Einstellungsfenster von »XGBoost Tree Ensemble Learner«

Die wichtigsten Einstellungen sind die Auswahl von Eingabe- und Zielspalten. Mit Use static random seed werden die Bäume reproduzierbar aufgebaut, sodass ein wiederholter Aufruf auch die gleichen Ergebnisse liefert.

Bei dem Baustein XGBoost Predictor stellen wir ein, dass die Vorhersagen in der neuen Spalte *pred* gespeichert werden. Der Ausgang dieses Bausteins enthält alle Daten der ursprünglichen Tabelle, ergänzt um die Vorhersage:



ID	sepal.length	sepal.width	petal.length	petal.width	species	pred
Row1	4.9	3.0	1.4	0.2	Setosa	Setosa
Row6	4.6	3.4	1.4	0.3	Setosa	Setosa
Row30	4.8	3.1	1.6	0.2	Setosa	Setosa
Row32	5.2	4.1	1.5	0.1	Setosa	Setosa

Abbildung 11.26 Ausgabe des Node-Monitors zu »XGBoost Predictor«

Mit dem letzten Baustein Scorer werden die Spalten *pred* und *species* miteinander verglichen. Dieses Programm erreicht eine Korrektklassifizierungsrate von über 93 %.

11.2.2 Deployment

In diesem Abschnitt werden wir nicht nur eine KI trainieren und testen (Klassifizierung von Schwertlilien), sondern auch das Modell (mit Model Writer) und die Validierungsdaten (mit table Writer) speichern. Ein anderes Programm, das Sie in Abbildung 11.27 sehen, lädt diese Daten und erstellt Vorhersagen.

10 % der Gesamtdaten werden als Validierungsdaten im Binärformat gespeichert, ebenso das Modell. Bei diesen Speicherbausteinen müssen Sie den Pfad sowie Speicheroptionen einstellen. Achten Sie darauf, dass die Einstellung Write options: overwrite gesetzt ist. Sonst generiert ein erneuter Ablauf des Programmes eine Fehlermeldung mit dem Hinweis, dass die Datei bereits existiert, wie in Abbildung 11.28 zu sehen.

Was ist der große Unterschied zwischen Validierungsdaten und realen Daten beim produktiven Einsatz? Die Validierungsdaten enthalten die Zielspalte (also die Information, was vorhergesagt werden soll). Die KI ist eigentlich dafür da, diese Information zu ermitteln. Aber mit den Validierungsdaten können wir das Modell bewerten, indem wir die Vorhersage mit den tatsächlichen Werten vergleichen (siehe Abbildung 11.29).

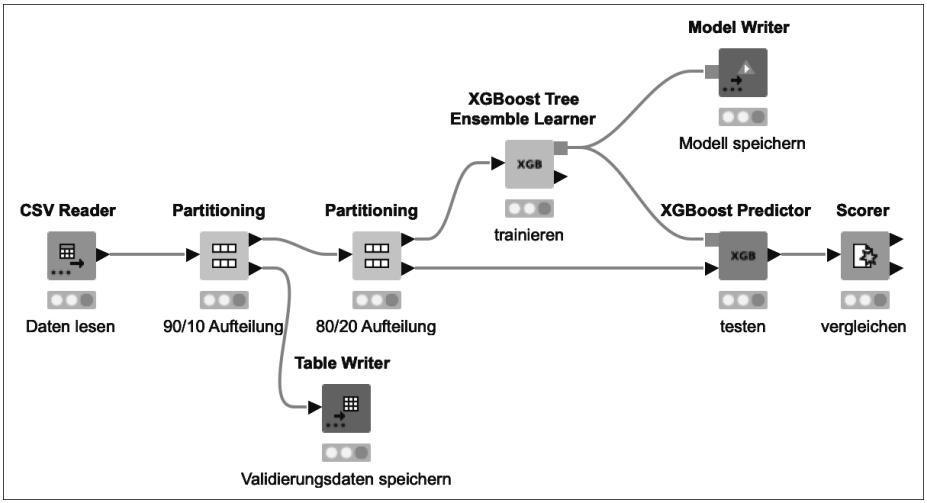


Abbildung 11.27 Das Programm »04a-iris.knwf«

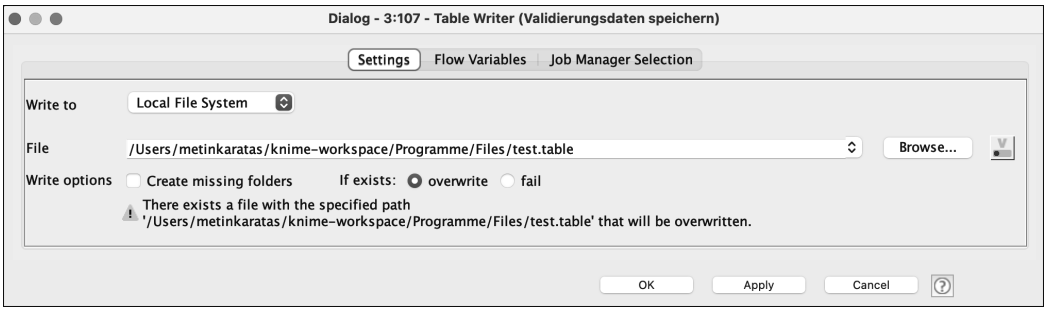


Abbildung 11.28 Das Einstellungsfenster von »Table Writer«

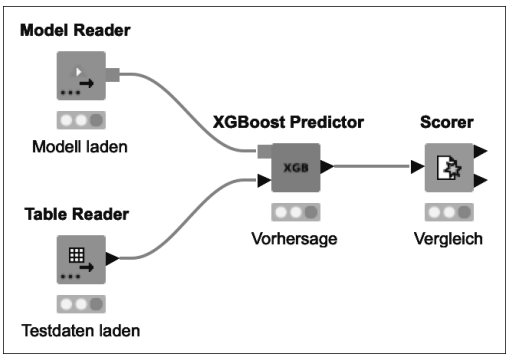


Abbildung 11.29 Das Programm »04b-iris.knwf«

Auch in diesem Programm legt der Baustein eine Spalte *pred* mit den Vorhersagen an, damit SCORER die Korrektklassifizierungsrate ermitteln kann.

Bei einer Vorhersage für reale Daten, in denen die Information der Unterart nicht enthalten ist, wäre der Baustein SCORER hinfällig. Sie müssten den Vorhersagen der KI vertrauen.

11.2.3 Regression

Die Vorhersage von stetigen Zahlenwerten sollte mit Ihrem jetzigen Wissen nicht allzu schwer umzusetzen sein. Es soll die Breite des Kronblattes ermittelt werden (siehe Abbildung 11.30).

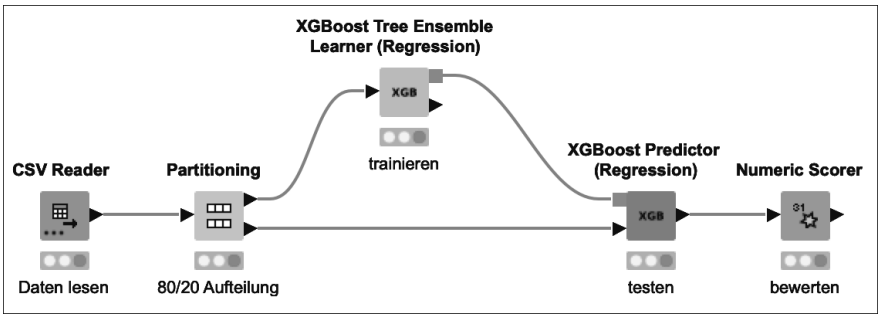


Abbildung 11.30 Das Programm »05-iris.knwf«

Wichtig ist, dass Sie immer einen Überblick über die Daten haben. Welche Spalten sind die Eingangsdaten und welche Spalte ist die Zielspalte? Darüber sollten Sie sich vor der Programmierung Gedanken machen. In diesem Programm wird die Breite des Kronblattes mit einer Genauigkeit von 0,17 cm (MAE) vorhergesagt.

11.3 Bildklassifizierung mit vortrainiertem Modell

Im nächsten Beispiel werden wir ein vortrainiertes Modell verwenden, um ein Objekt auf einem Bild zu klassifizieren (Teddybär). Es wird jeweils ein Programm mit Keras- und Python-Bausteinen.

11.3.1 Bildklassifizierung mit Keras Node

Diesmal verwenden wir nicht VGG19, sondern InceptionV3 (*https://arxiv.org/abs/1512.00567*) von Google. Dieses Modell ist schneller in der Ausführung als VGG19 (zumindest mit KNIME).