Hausarbeit:
Radiale Basisfunktionen Netze

Im Modul Maschinelles Lernen
am Campus Velbert/Heiligenhaus der Hochschule Bochum
Im Wintersemester 20/21

bei **Prof. Dr.-rer Jörg Frochte**

Dimitri Sonkwa Nguegoua und
Donald Wilfrank Yamegueu

Student in der Fachrichtung
Technische Informatik Grundstudium an der Hochschule Bochum CVH

14. April 2021

Inhaltsverzeichnis

1 Einführung

Mit dem schnellen Wachstum der Technologie, werden mehr und mehr neue Techniken für die Entwicklung der künstlichen Intelligenz und deren Automatisierung entwickelt. Dazu zählen wir das Maschinelle Lernen, welches ein Untersuchungsgebiet der Künstlichen Intelligenz ist, und welches auf mathematischen und statistischen Ansätzen beruht, um Computern aus Daten zu lernen", d.h. ihre Leistung bei der Lösung von Aufgaben zu verbessern. Im weiteren Sinne geht es um die Analyse, Optimierung, Entwicklung und Implementierung solcher Methode. In dieser Hausarbeit werden wir uns mit einem bestimmten Thema von maschinellen Lernen beschäftigen. Es handelt sich um die Netze Radialer Basisfunktionen. Die Netze Radialer Basisfunktionen (RBF-Netze) oder auch Radiale Netze genannt sind eine spezielle Form von künstlichen Neuronalen Netzen, welche einen oder mehrere numerische Eingabewerte (Inputs-Werte) annehmen kann und generiert einen oder mehrere Outputs-Werte. Basierend auf historische Daten sind die RBF-Netze ein gutes Computer-Lernen Techniken um Beispielsweise Daten zu klassifizieren und Vorhersagen zu machen. Es könnte zum Beispiel verwendet werden um die Gewinn oder die Produktionsmenge eines Unternehmen in nächsten Jahren vorherzusagen oder um ein Krankenhaus-Patienten Risiko von Diabetes zu klassifizieren,basierend auf den Werten der medizinische Testergebnisse und andere Faktoren wie Alter und Geschlecht.

Diese Hausarbeit stellt die Grundlagen der RBF-Netze dar und erklärt das Trainingsverfahren von RBF-Netzen. Im weiteren Sinne wird es erklärt wie Funktionen sich mit radialer Basisfunktionen approximieren lassen bzw. wie man mit radialer Basisfunktion Vohersagen machen kann. Abschließend stellt die Hausarbeit durch einige Beispiele den Unterschied zwischen radialen Netzen (RBF) und anderen Verfahren wie Dichte Neuronale Netze und Random Forest dar.

Die Beispiele, die wir während dieser Hausarbeit verwenden, beziehen sie sich auf die Klassifikation und die lineare Regression. Als Datensatz verwenden wir das von der Bibliothek Scikit-Learn angebotenes “Iris-Datasets” um unseren Algorithmus zu trainieren. Das Ganze wird in python implementiert. Im nächsten Abschnitt stellen wir zunächst ganz grob die künstlichen Neuronale Netze dar.

2 Künstliche Neuronale Netze (ANN)

Basierend auf dem biologischen neuronalen Netz wie dem Gehirn, ein künstliches neuronales Netzwerk versucht, den Rechenweg des biologischen neuronalen Netzwerk wiederherzustellen. Unser Gehirn enthält ungefähr 100 Milliarden Neuronen, die über elektrochemische Signale kommunizieren. Die Neuronen sind durch bezeichnete Übergänge verbunden. Jedes Neuron empfängt Tausende von Verbindungen mit anderen Neuronen und empfängt ständig eingehende Signale, um den Zellkörper zu erreichen. Wenn die resultierende Summe der Signale einen bestimmten Schwellenwert überschreitet, wird eine Antwort durch das Axon gesendet.
In der gleichen Logik bearbeitet ein künstliches neuronales Netz die Daten. Es besteht aus Neuronen auch Knoten genannt, die untereinander über die sogenannte Kanten (Gewichtungen) verbunden sind, und aus drei Schichten nämlich einem Input-Layer, einem Hidden-Layer (auch als versteckte Schicht bezeichnet) und einem Output-Layer (also die resultierende Summe des Produkts aller Input-Werte mit Gewichtungen der Neurone im Hidden-Layer) ,siehe **Abbildung 1**. Der Unterschied zwischen den drei Schichten ist, dass im Input-Layer werden Informationen in Form von numerischen Werten von der realen Welt aufgenommen. Dann werden die Input-Werte mit jeweils unterschiedlichen Gewichtungen der Neurone multipliziert und in Hidden-Layer weitergeleitet. Die resultierende Summe aller Produkte bildet das Ergebnis des Netzes, welches an die Außenwelt weitergegeben wird. Der Ausgang des Netzes ist eine lineare Funktion.

Weiterhin hängt die Aktivierung jeder versteckten Neuronen von dem Produkt der Input-Werte und der Gewichtungen ab. Das Neuron wird dann aktiviert wenn die Summe des Produkts oberhalb des Schwellenwerts liegt. Als Aktivierungsfunktion verwendet ein künstliches neuronales Netz nicht lokale Funktion, meisten wird die Signumfunktion.

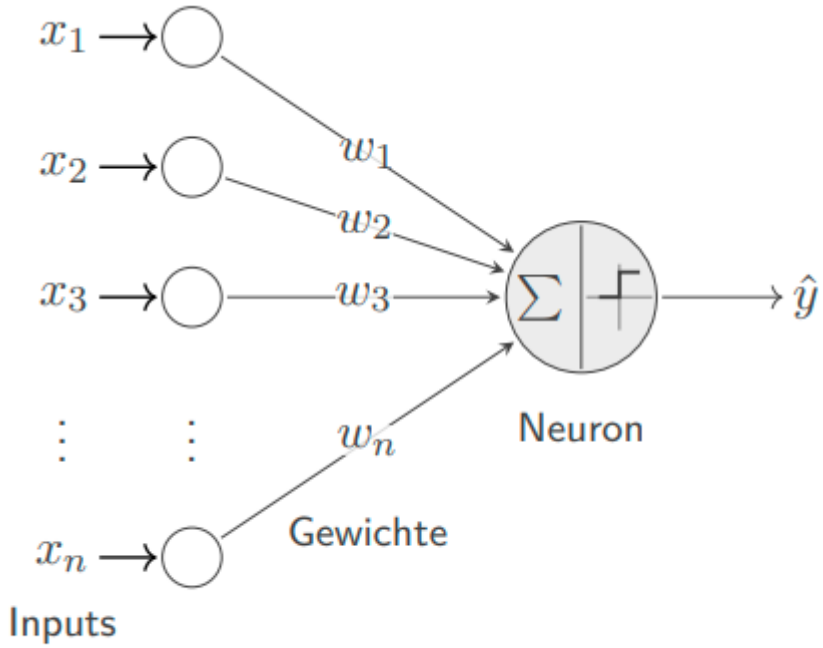


Abbildung 1: Einlagiges Perceptron eines KNN

3 Radiale Basisfunktion

3.1 Definition

Eine radiale Basisfunktion (RBF) kann allgemein als eine reelle Wertfunktion definiert werden, deren Wert nur vom Abstand zwischen der Eingabe und einem festen Punkt abhängt, wobei letzterer der Ursprung ist, so dass:

$$(x) = (||x||),$$

oder ein anderer fester Punkt, der Mittelpunkt genannt wird, so dass:

$$(x) = (||x - c||).$$

Jede Funktion , die die folgende Eigenschaft erfüllt:

$$(x) = (||x||),$$

wird als radial bezeichnet. Der besagte Abstand ist in den meisten Fällen der euklidische Abstand, obwohl es auch andere Maße gibt, die zur Berechnung des Abstands verwendet werden.

3.2 Mathematische Darstellung

Eine Radialfunktion ist eine Funktion

$$g : [0, \infty) \to \mathbb{R}.$$

Gepaart mit einer Metrik auf einem Vektorraum

$$|| \cdot || : V \to [0, \infty),$$

Eine Funktion $g_c = g(||x - c||)$ soll ein radialer Kern sein, der bei $\mathbf{c}$ zentriert ist. Eine Radialfunktion und die zugehörigen Radial-Kerne werden als radiale Basisfunktionen bezeichnet, wenn für eine beliebige Menge von Knoten

$$(\mathbf{x}_k)_{k=1}^n.$$

$*$ Die Kernel $g_{x1}, g_{x2}, ..., g_{xn}$ sind linear abhängig
zum Beispiel: $g(r) = r^2 \in V = \mathbb{R}$ ist keine radiale Basisfunktion.)
$*$ Die Kernel $g_{x1}, g_{x2}, ..., g_{xn}$
bildet eine Basis für einen Haarraum, was bedeutet, dass die Interpolationsmatrix

$$\begin{pmatrix} g(||x_1 - x_1||) & g(||x_2 - x_1||) & \cdots & g(||x_n - x_1||) \\ g(||x_1 - x_2||) & g(||x_2 - x_2||) & \cdots & g(||x_n - x_2||) \\ \vdots & \vdots & \ddots & \vdots \\ g(||x_1 - x_n||) & g(||x_2 - x_n||) & \cdots & g(||x_n - x_n||) \end{pmatrix}$$

nicht singulär ist.

3.3 Die Arten Radialer Basisfunktion

Dies gehört zu besonderen Funktionen,Ihre Ansprechverhalten steigt oder fällt monoton mit dem Abstand zu einem zentralen Punkt. Der Mittelpunkt,der Abstand und die Form der radialen Basisfunktion sind die Parameter des Modells, das linear ist,wenn sie festgelegt ist.
Eine typische radiale Basisfunktion ist von der Form:

$$g(x) = exp(-\frac{(x-c)^2}{r^2}).$$

Diese Parameter sind der Mittelpunkt cünd der Radius r".Eine Gaußsche Radiale-Basisfunktion nimmt mit zunehmendem Abstand vom Zentrum ab. (siehe Abbildungen)
Im Gegensatz dazu nimmt eine multiquadratische radiale Basisfunktion mit zunehmendem Abstand vom Zentrum zu. Sie hat die folgende Form:

$$g(x) = \frac{\sqrt{r^2+(x-c)^2}}{r^2}$$

Zum Schluss merken wir ,dass Gaußsche RBFs werden am häufigsten verwendet, weil sie biologisch plausibler sind und eine endliche Antwort haben, was praktischer ist.
Die Abbildung da zeigt die Skizzen von Baisifuntionsarten im 2D bzw im 1D. Die drei Funktionen lassen sich gut differenzieren und sie sind jeweils Symmetrisch durch ihren Zentrum.

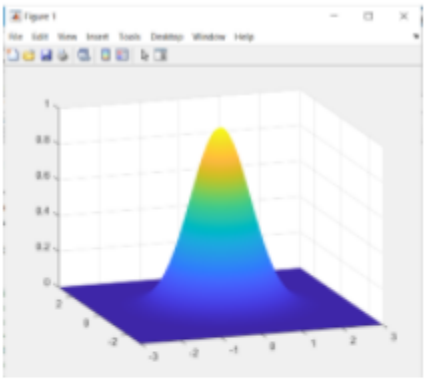


Figure 2: Gaußian_plot_3d

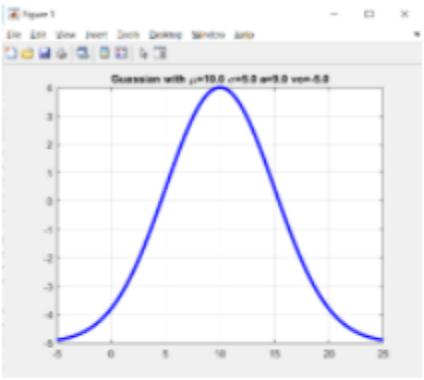


Figure 3:Gaußian_plot_1d

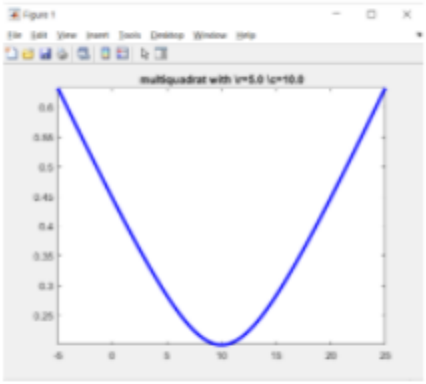


Figure 4: multiquadratic_1d

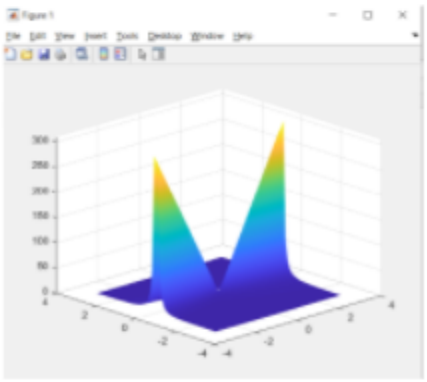


Figure 5: multiquadratic_3d

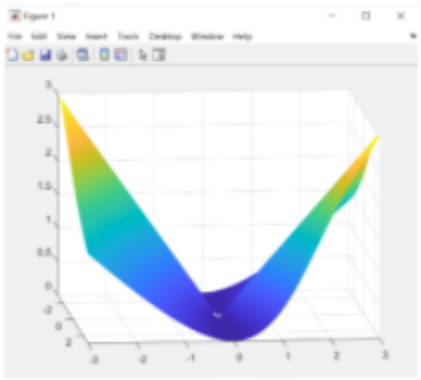


Figure6:inv.multiquadratic_1d

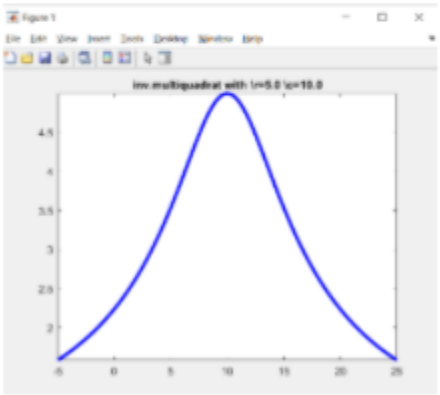


Figure7:inv.multiquadratic_3d

Abbildung 2: Arten von Radiale Funktion

- Gauß :

$$\varphi(r) = e^{-(\varepsilon r)^2}$$

- Multiquadric :

$$\varphi(r) = \sqrt{1 + (\varepsilon r)^2}$$

- Inverses Quadrat :

$$\varphi(r) = \frac{1}{1 + (\varepsilon r)^2}$$

- Inverse Multiquadric :

$$\varphi(r) = \frac{1}{\sqrt{1 + (\varepsilon r)^2}}$$

Abbildung 3: Mathematische Beschreibung

4 Radiale Basis Funktionsnetze

4.1 Problembeschreibung

Im gegensatz von künstlichen neuronalen Netzen wird bei Netzen Radialen Basisfunktionen lokale Neurone betrachtet d.h.jedes Neuron ist eine radiale Basisfunktion, das nur auf Eingaben in einem bestimmten Teil des Eingaberaums reagiert. Die Idee ist, dass bei ähnlichen Inputwerten,sollten auch ähnliche Antworten auf diese Inputwerte ergeben, und daher sollte das gleiche Neuron reagieren, solange bis es ein bestimmtes Grad erreicht hat. Wenn das der Fall ist, dann wird das Neuron aktiviert.
Um diesen Prozess modellieren zu können verwendet wir ein mathematisches Modell, welches sehr gut differenzieren lässt und das symmetrisch (in alle Richtungen oder Radial) von einem Maximum auf Null abfällt. Mehrere Funktionen mit dieser Eigenschaft (**Abschnitt 3.3**) können benutzt werden. Wir beschränken uns aber in dieser Hausarbeit auf die häufigste, die auch sehr oft in der Statistik verwendet wird, also die Gauss-Funktion.
Die Funktion wird als Aktivierungsfunktion verwendet und beschreibt den Abstand zwischen den Eingaben und dem Zentrum jedes Neurons. Die Funktion lässt sich durch folgende Gleichung schreiben:

$$\varphi(x, \mu, \sigma) = \exp(\tfrac{-\|x - X_k\|^2}{2\sigma^2})$$

In dieser Gleichung ist die Wahl von σ sehr relevant, weil es die Breite des Gauß angibt bzw. es bestimmt, wie weit ein Neuron ein signal erhalten kann. Daher ist die Gauss-Funktion wie oben dargestellt abhängig von dem Parameter σ. Dieses Parameter ist für jedes Zentrum der Radialfunktion unterschiedlich auszuwählen. Ein Grund ist, dass das Neuron auf jede Eingabe reagieren wird, wenn wir σ zum Beispiel groß machen und letztendlich wird dieses Neuron nur auf ein einziges Input reagieren, wenn wir dieses Parameter immer kleiner machen.

Für die Modellierung brauchen wir daher die Gauß-Glieder um visualisieren zu können, wie die Neurone im Hidden-layer reagieren wenn der Eingang bei Ihnen steht,oder wenn der Eingang weiter entfernt ist oder wenn der Abstand zwischen dem Eingang und dem Neuron immer noch größer wird. Bevor wir darüber diskutieren, stellen wir zunächst die Architektur eines RBF-Netzes im folgenden Abschnitt dar.

4.2 Aufbau und Architektur eines RBF-Netzes

Die **Abbildung 4** zeigt die Architektur eines RBF-Netzes. Es besteht aus einer Inputschicht mit N Eingaben (Eingangsvektor) und aus einem einzigen Hidden-Layer mit K Neuronen, deren jedes Neuron eine radiale Basisfunktion (Nichtlineare Funktion) ist. Jede Schicht ist vollständig mit der nächsten verbunden und es gibt keine Verbindungen innerhalb einer Schicht.Also, in der Inputschicht werden die Werte jeder Eingaben an alle Neuronen des Hidden-Layer verteilt und im Hidden-Layer wird in jedem Neuron der Abstand zwischen der Eingabe und dem Zentrum X_k mit Hilfe einer Norm gebildet:

$\|x - X_k\|$. x und X_k müssen den gleichen Raum aufweisen.

Danach wird dieser Abstand durch eine Gauss-Kennlinie zugeschickt:

$$\varphi(x, X_k, \sigma) = \exp(\tfrac{-\|x-X_k\|^2}{2\sigma_k^2}).$$

Wobei x die Trainingsdaten ist, X_k ist das Zentrum jedes Neurons und σ_k ist ein Steuerungsparameter. Hier ist es wichtig, dass die Dimension des Eingabevektors x mit der Dimension von Zentren X_k übereinstimmt.
Das Output-Layer des Netzes mit M Neuronen ist eine lineare Kombination der radialen Basisfunktionen von Input-Werten und Gewichte (gewichte Summe). Der Ausgang eines RBF-Netzes ist gegeben durch folgende Gleichung:

$$y_k = f(x; X_k, \sigma, \omega) = \omega_0 + \sum_{i=1}^{k} \omega_k \exp(\frac{-\|x - X_k\|^2}{2\sigma^2})$$

$$\implies y_k = f(x; X_k, \sigma, \omega) = \omega_0 + \sum_{i=1}^{k} \omega_k \varphi_k(x; X_k, \sigma, \omega)$$

mit $k \in \{1, \ldots, n\}$
$Wobei, f: \ R^d \longrightarrow R$, eine stetige Funktion ist,

$X = [X_1, \ldots, X_k]$ ist eine Matrix, welche die Spalten die Zentren von Hidden-Neuronen sind,

$\sigma = [\sigma_1, \ldots, \sigma_k]^T$ ist der Vektor der Steuerungsparameter und ,

$\omega = [\omega_1, \ldots, \omega_k]^T$ der Gewichtungsvektor ist.
$\omega_\mathbf{0}$ wird als Bias bezeichnet. Es dient dazu unsere lineare gerade aus dem Ursprung zu verschoben.
Diese oben dargestellte Parameter sind für den Aufbau bzw das Training unseres Netzes sehr wichtig. Sie müssen daher geregelt werden. Wird ein der Parameter geändert, ändert sich auch das Verhalten des Netzes. Daher werden für das Training unseres Netzes folgende Parameter zum Teil des Algorithmus gehören:

- Die Anzahl der RBF-Neuronen in der einzelnen versteckten Schicht oder die Anzahl der Gauß'schen (**N**),
- Die Position der Gauß'schen Zentren eines jeden Neurons ($_k$),
- Die Breite dieser Gauß-Linien (σ),
- Das Gewicht der Verbindungen zwischen den RBF-Neuronen und dem/den Ausgangsneuronen (ω). enditemize

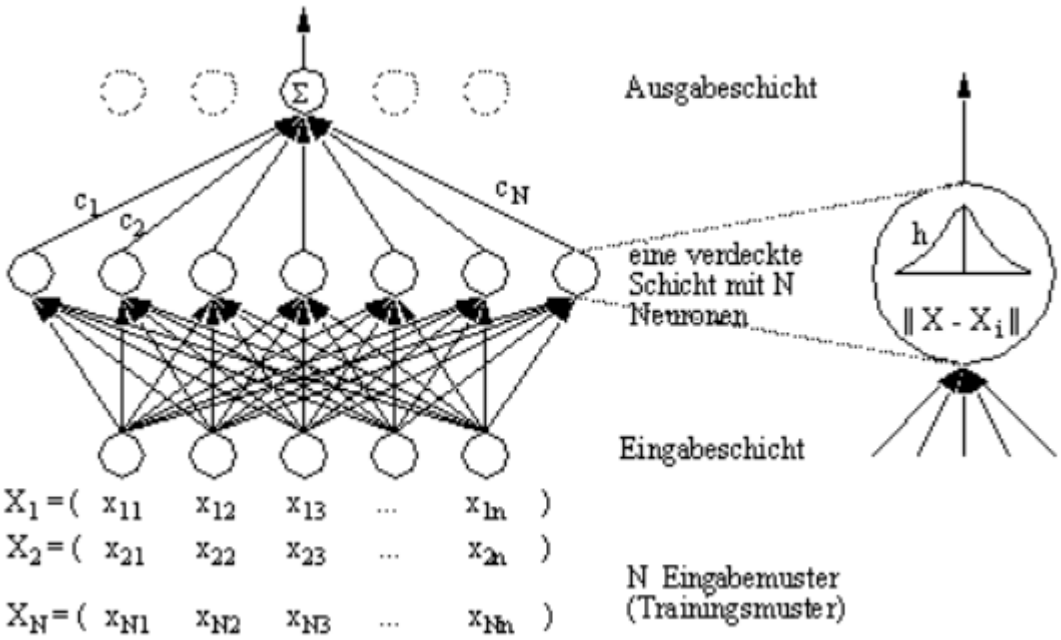


Abbildung 4: Architektur von RBF-Netzen

4.3 Training von RBF-Netze

Das RBf-Netz-Algorithmus gehört zu den wichtigsten überwachten Lernalgorithmen des Maschinen Lernens, welches am häufigsten für die Bilderkennung oder für die Klassifikation verwendet wird. Dieser Algorithmus nimmt wie alle überwachten Lernalgorithmen die Trainingsdaten und die dazugehörige Ausgabe, d.h. es nimmt die Antworten mit jeder Trainingsdaten während des Lernprozesses auf. Das Hauptziel besteht darin, eine Assoziation zwischen Trainingsdaten und den entsprechenden Ausgaben zu lernen.Zum Beispiel: Um eine folgende Zuordnungsfunktion zu lernen :

$$\mathbf{Y} = \mathbf{f}(\mathbf{X}),$$

hätten wir $\mathbf{X}$ Trainingsdaten als Eingangsdaten und $\mathbf{Y}$Ausgaben, wenn wir die Daten an unseren Algorithmen anwenden würden.

Zusätzlich müssen wir die Zuordnungsfunktion so gut approximieren,dass wir selbst bei neuen Eingangsdaten ($\mathbf{X}$) die Ausgabevariable ($\mathbf{Y}$) für diese neuen leicht vorhersagen können. Das Training des RBF-netzes wird sequentiell und in zwei Teilen durchgeführt, weil sowohl das Hidden- als auch das Output-Layer verschiedene Funktionen anbieten und diese daher nicht zusammen trainiert werden müssen. Der Zweck der RBF-Knoten in der versteckten Schicht besteht darin, eine nichtlineare Darstellung der Eingaben zu finden, während der Zweck des Output-Layer darin besteht, eine lineare Kombination dieser versteckten knoten zu finden, die die Klassifizierung durchführen .

Wir werden zunächst mittels eines Algorithmus (K-Means Algorithmus), welches zum unüberwachten Lernen gehört, die Zentroiden jedes Neuron im Hidden-Layer bzw. die Breite des Gauß festlegen bzw. bestimmen . Und danach verwenden wir im zweiten Teil unseres Verfahrens ein überwachtes Algorithmus, der Für das Training der linearen Ausgabe die Aktivierung dieser versteckten Knoten verwendet. Der ganze Prozess wird als Hybride-Algorithmus bezeichnet, weil er sowohl ein überwachtes als auch ein unüberwachtes Lernen kombiniert.

4.3.1 Trainingsprozess

1) Die Position der Gauß'schen Zentren der RBF-Knoten,

   – Anwendung des K-Means Algorithmus Zur Initialisierung der Positionen von RBF Zentren oder,

   – Festlegen von RBF Zentren als zufällig ausgewählte Datenpunkte

2) Berechnen mit Hilfe der Gauß Gleichung die Aktivierung der RBF versteckten Schichten,

3) Trainieren der Gewichtsmatrix entweder :

– Nutzung der Perzeptron bei Dichte Neuronalen Netzen oder

– Berechnung der Pseudo-Inversen der Aktivierungen der RBf Zentren . endenumerate

4.4 Mathematische Beschreibung

Wir nehmen an:
Sei X eine Trainingsdaten (Eingabematrix) mit n Beobachtungen und d Merkmale,

$$\mathrm{X}_{n,d} = \begin{pmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,d} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,d} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n,1} & x_{n,2} & \cdots & x_{n,d} \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \in \mathbb{R}^{n \times d}$$

Mit Hilfe der Radialen Basisfunktion Netze möchten wir durch die Trainingsdaten X unseren Algorithmus lernen, um die Ausgabedaten vorhersagen zu können.
Dafür müssen wir für den Lernprozess die Zentren der Radiale Basisfunktionen festlegen.

Wir haben die Möglichkeit entweder in der Trainingsdaten zufällige Datenpunkte auszuwählen, die die Zentroide entsprechen würden oder mit Hilfe des K-Means Algorithmus die Zentroiden zu bestimmen.

- seien folgende Datenpunkte, die die Zentroiden darstellen: $x^{(1)}, \ldots, x^{(b)}$

- Nun wird die Steuerungsparameter σ (Bandbreite) bestimmt:

  $$\sigma = \frac{d_{Max}}{\sqrt{2M}}$$

  Wobei M die Anzahl der RBF-Knoten in der versteckten Schicht ist und d_{Max} der maximale Abstand zwischen zwei Zentroiden. d_{Max} wird auch als euklidischer Abstand genannt. Es lässt sich durch folgende Formel berechnen:

$d_{Max} = \|x - X_n\| = \sqrt{(x_1 - X_1)^2 + (x_2 - X_2)^2 + \ldots + (x_n - X_n)^2}, mit$
$x = [x_1, x_2, \ldots, x_n] und\, X = [X_1, X_2, \ldots, X_n]$

Nachdem die beiden Parameter gefunden worden sind, können wir für jede Neurone in der Versteckten Schicht eine radiale Basisfunktion definieren, in dem wir folgende Formel anwenden:

$\varphi^{(i)}(x) = \exp(\frac{-\|x-x^{(i)}\|^2}{2\sigma^2}) \quad \forall i \in \{1, \ldots, b\} \quad und \quad x \in \mathbb{R}^d$

Daher erhalten wir folgende transformierte Datenmatrix:

$$\varphi = \begin{pmatrix} \varphi^{(1)}(x_1) & \varphi^{(2)}(x_1) & \cdots & \varphi^{(b)}(x_1) \\ \varphi^{(1)}(x_2) & \varphi^{(2)}(x_2) & \cdots & \varphi^{(b)}(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ \varphi^{(1)}(x_n) & \varphi^{(2)}(x_n) & \cdots & \varphi^{(b)}(x_n) \end{pmatrix} = \begin{pmatrix} \varphi_1 \\ \varphi_2 \\ \vdots \\ \varphi_n \end{pmatrix} \in \mathbb{R}^{n \times (b)}$$

Anders formuliert, wir berechnen für jeden Eingabevektor die Aktivierung jedes Neurons der versteckten Schicht und parken das in eine Matrix φ rein .

- Die Ausgaben unseres Netzes lässt sich dann wie folgt berechnen:

  $\mathbf{Y} = \varphi\omega$
  In der Gleichung ist uns noch ω unbekannt. Um es zu bestimmen, können wir die Inverse der Matrix φ berechnen,

  Es gilt: $\omega = \mathbf{Y}\varphi^{-\mathbf{1}}$.

  Zur Erinnerung: um eine Matrix investieren zu können, muss die Matrix die Dimension $\mathbf{n} \times \mathbf{n}$ haben, also die Matrix muss quadratisch sein. Wir wissen aber nicht genau ob φ eine $\mathbf{n} \times \mathbf{n}$ Matrix ist, da die Anzahl der versteckten Neurone nicht die Anzahl der Eingabedaten übereinstimmen könnte.

- Aus Diesem Grund verwenden wir die Pseudoinverse.
  Die Pseudoinverse einer Matrix φ lässt sich wie folgende Gleichung beschreiben: $\varphi^{+\mathbf{1}} = (\varphi^{\mathbf{T}}\varphi)^{-}\mathbf{1}\varphi^{\mathbf{T}}$

$$\Longrightarrow \omega = (\varphi^{\mathbf{T}}\varphi)^{-}\mathbf{1}\varphi^{\mathbf{T}}\mathbf{Y}$$

- Nachdem wir die Gewichtungen approximiert haben, suchen wir nach einer mögliche Lösung von ω, die die Differenz zwischen dem erwarteten Ausgang bzw. dem beobachten Ausgang optimiert. Man erhält dann ein folgendes lineares Gleichungssystem:

$$\mathbf{Y} = \varphi\omega + \mathbf{e},$$

  Wobei Y das erwartetes Ergebnis ist und $\varphi\omega$ das beobachtete Ergebnis. Die Matrix φ von Dimension $d \times N$ gibt die Antwort von N Zentren und d Beobachtung.

$$\Longrightarrow \mathbf{e} = \mathbf{Y} - \varphi\omega$$

  Mit dieser dargestellten Gleichung sehen wir wie gut unseren Algorithmus ist.

5 Implementierung des RBF-Netzes in Python

Wir verwenden als Datensatz "Iris", der schon von der Bibliothek Scikit-Learn zu unserer verfügung gestellt wird.

5.1 Dataset Iris

Das Iris-Dataset wird am meisten für die Visualisierung und das überwachte Lernen benutzt. Es besteht aus drei Variationen der Irisblume. Es enthält 150 Beobachtungen (Zeile des Datensatzes). Jede Beobachtung besteht aus vier Attributen (Merkmalen) zur Beschreibung einer Irisblume. Der Datensatz ist nach dem Blumentyp gekennzeichnet. Für vier Attribute, die eine Irisblume beschreiben, wissen wir also, um welche Variante es sich handelt.Schließlich enthält der Datensatz keine fehlenden Werte. Dadurch müssen wir sie nicht mehr verarbeiten.

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

150 rows × 5 columns

Abbildung 5: Iris-Datasets

Die Letzte Spalte des Datensatzes entspricht das erwartete Ergebnis. Es handelt sich um ein Vektor Y der Dimension 150×1.
Die Umsetzung des Verfahrens wird sequential in zwei Teilen durchgeführt. Zu einem verwenden wir das unüberwachte Lernen um die Zentren der versteckten Neurone festzulegen und die Standardabweichung (Steuerungsparameter) zu bestimmen. Mit der Pseudoinverse lässt sich zu anderem die Gewichtung des Netzes approximieren. Es ist zusätzlich auch Mögliche die Gewichtung mit dem Gradientenabstieg zu trainieren, welches sich auf ein überwachtes Lernen bezieht.

5.2 Unüberwachtes Lernen

Im Gegensatz zum überwachten Algorithmus wo der Algorithmus sowohl die Trainingsdaten (Eingabedaten $\mathbf{X}$) als auch die dazugehörigen Ausgabedaten ($\mathbf{Y}$) zum Training des Modells bekommt, erhält ein unüberwachter Algorithmus nur die Trainingsdaten.Der Trainingsalgorithmus versucht in diesem Fall, allein die Ähnlichkeiten und Unterscheidungen innerhalb dieser Daten zu finden und diejenigen zu gruppieren, die gemeinsame Merkmale aufweisen. In unserem Beispiel verwendet wir der K-Means Algorithmus um die Zentroiden unseres Modells zu bestimmen.

5.2.1 K-Means Algorithmus

In diesem Abschnitt zeigen wir wie die Zentroiden mit Hilfe des K-Means bestimmt werden können. Um die Zentroiden festlegen zu können, leitet der K-Means Algorithmus alle Daten zum nächsten Zentrum, der am nähesten der Eingabe liegt. Für jeden neuen Eingangsvektor $\mathbf{X} = [\mathbf{X_1}, \mathbf{X_2}, \mathbf{X_3}, \ldots, \mathbf{X_n}]$ berechnet der Algorithmus den Abstand zwischen dem Eingangsvektor und dem Referenzvektor (Zentrum). Mit der Anzahl der angegebenen Zentren führt der Algorithmus diese Berechnung zu jedem Zeitpunkt für den Eingabevektor des Netzwerks durch und weist die Zentren nacheinander den Knoten zu. Für den Algorithmus wird die Anzahl der Cluster bei der Initialisierung festgelegt. Dort wird auch jedem Cluster ein Referenzvektor zugewiesen. Nun wird ein Vektor aus der Datensatz ausgewählt und der ähnlichste Cluster bestimmt. Der Referenzvektor dieses Clusters wird in Richtung des Datenvektors gezogen. Das Verfahren terminiert, wenn alle Referenzvektoren stabil bleiben. Es lässt sich Im Python wie folgt umsetzen.

- Bibliothek importieren:
  Aus der Bibliothek Scikit-Learn wird KMeans importiert

```
8 import seaborn as sns
9 import numpy as np
10 import matplotlib.pyplot as plt
11 from sklearn.cluster import KMeans
```

Abbildung 6: Bibliothek Importieren

- Datensatz aufladen:
  Die Bibliothek Scikit-learn bietet unterschiedliche Methoden um einen Datensatz aufzuladen. Diese Methoden befinden sich in der Klasse **datasets**.
  Um unseren Iris-Datensatz aufzuladen, benutzen wir die Methode **load_dataset()**.

```
13 data= sns.load_dataset("iris")
14 data['species'].replace(to_replace=['setosa'], value= 1, inplace=True)
15 data['species'].replace(to_replace=['virginica'], value= 3, inplace= True)
16 data['species'].replace(to_replace=['versicolor'], value= 2, inplace= True)
```

Abbildung 7: Datasets aufladen

Die letzten drei Zahlen ermöglichen die Letzte Spalte unseres Datasets von **"String"** nach **"Integer"** umzurechnen.

Die **Abbildung 6** zeigt, die neue Darstellung des Datasets ohne die String-Namen **"Setosa", "Versicolor", oder "Virginica"**. Wir stellen fest, dass die Irisblume **"Setosa"** durch **"1"**, **"Versicolor"** durch **"2"** und **"Virginica"** durch **"3"** ersetzt wurde.

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	1
1	4.9	3.0	1.4	0.2	1
2	4.7	3.2	1.3	0.2	1
3	4.6	3.1	1.5	0.2	1
4	5.0	3.6	1.4	0.2	1
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	3
146	6.3	2.5	5.0	1.9	3
147	6.5	3.0	5.2	2.0	3
148	6.2	3.4	5.4	2.3	3
149	5.9	3.0	5.1	1.8	3

150 rows × 5 columns

Abbildung 8: Datasets: Die letzte Spalte wurde durch ganze Zahl ersetzt

Die Attributes **Sepal_length, sepal_widht, petal_length, petal_width** beschreiben die Merkmale jeder variation der Irisblume. Ein Teil davon wird als Trainingsmatrix benutzt und wird in X gespeichert. Bei **species** handelt es sich um die erwarteten Ergebnisse, Sie werden dann in Y gespeichert.

- KMeans aufbauen:

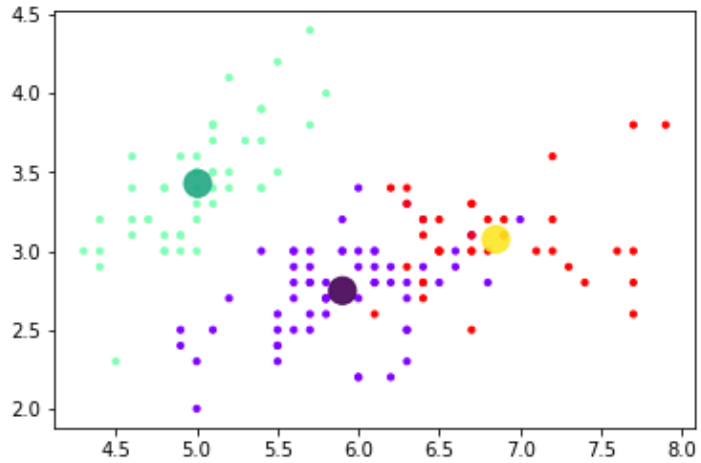
  Mit die Bibliothek Sklearn geht das ganz einfach. Man benötigt nur die notwendige Bibliothek bzw. Methoden aufzurufen, außerdem muss man die Anzahl der Cluster bzw. Zentren fixieren und dann die Methode **fit()** aufrufen. Danz ganze wird dann automatisch ausgeführt.

```python
# Bestimmung der Zentren des RBF-Netzes mit Kmeans algorithmus
def kmeans(train_x, k):
    kmeans = KMeans ( n_clusters=k)
    kmeans.fit(train_x)
    y_kmeans = kmeans.predict (train_x)
    centers= kmeans.cluster_centers_

    plt.scatter(train_x [:, 0], train_x[:, 1], c = y_kmeans, s=10, cmap = 'rainbow')
    plt.scatter( centers [:, 0], centers [:, 1], c = (30,150,230), s = 200, alpha = 0.9
    plt.show()
    return centers
```

Abbildung 9: Kmeans-Algorithmus aus Sklearn Bibliothek

Die Methode ist ganz schnell und einfacher, da man nicht so viel Codezeilen schreiben muss. Wenn wir dieses Code in Spyder plotten, erhalten wir im 2D die Zentren des RBF-Netzes. Bei $K = 3$ (Anzahl der Clusters) bekommt man diese folgende Abbildung

```
In [1]: runfile('C:/Users/User/Anaconda3/kmeans_rbf.py', wdir='C:/Users/User/Anaconda3')

In [2]:  a = kmeans(train_x,3)
```




```
In [3]: a
Out[3]:
array([[5.9016129 , 2.7483871 , 4.39354839, 1.43387097],
       [5.006     , 3.428     , 1.462     , 0.246     ],
       [6.85      , 3.07368421, 5.74210526, 2.07105263]])
```

Wir fangen hier mit den zwei ersten Spalten unserer Trainingsdaten an. Aus dem Bild sind die festgelegten Zentren zu sehen (Hier K = 3, Anzahl der Zentroiden) .Die Zentralwerte bilden eine 4×3 Matrix, wobei jede Zeile den Zentrum einzelner RBF-Knoten beschreibt.

Mit dem K-Means-Algorithmus haben wir die Zentren der RBF-knoten berechnet und festgelegt.

Eine zweite Alternative wäre den Algorithmus ohne die Nutzung der sklearn Bibliothek zu implementieren. Bei dieser Alternative implementieren wir den K-Means-Algorithmus, in dem wir eine Methode Kmeans()(**Abbildung 8**) erzeugen, die drei Parameter aufnimmt und wiederum die Zentren bzw. die Standardabweichung zurückgibt (**Abbildung 8**). wir entnehmen zuerst aus der Trainingsdaten zufällige Zentren. Wir berechnen den Abstand mittels dem euklidischen Norm zwischen Eingabevektor und jeweiligen ausgewählten Zentren. Das Ergebnis wird in einer variable Clusterliste gespeichert. Danach machen wir eine Kopie davon. Nun laufen wir unsere Liste durch und wir berechnen den Mittelwert von jedem **j** Element in der Liste. Das Ergebnis wird dann in einer Variable Centroids hinzugefügt. Abschließend berechnen wir die Differenz zwischen der aus Trainingsdaten ausgewählten Zentroiden und der berechneten. Falls die Differenz Null wäre dann haben den richtigen Zentrum herausgefunden.
Die Methode liefert uns auch die Standardabweichungen von jedem einzelnen RBF-Neuron.

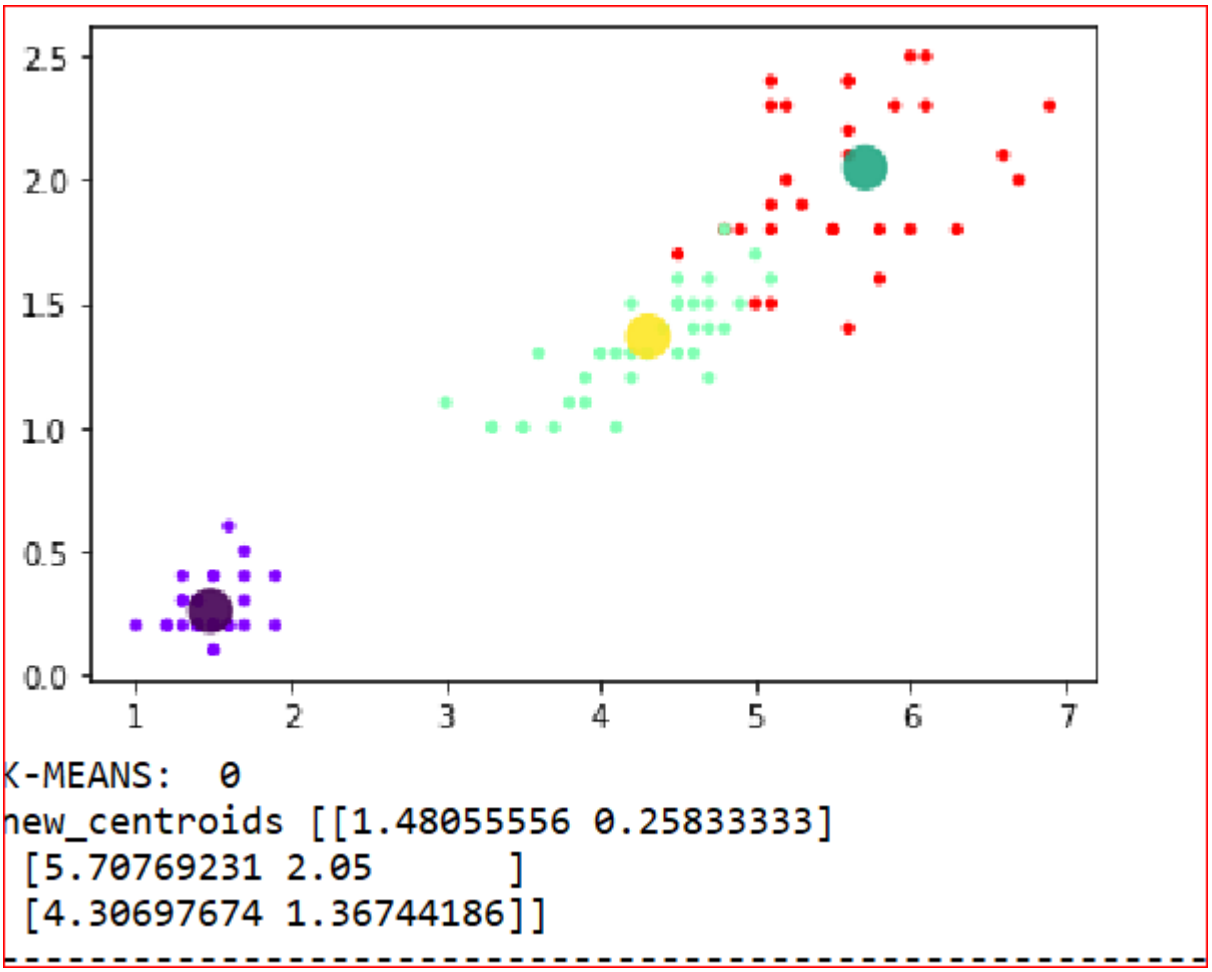
```
38 def kmeans(X, k, max_iters):
39     """Performs k-means clustering for 1D input
40 
41         Arguments:
42             X {ndarray} -- A Mx1 array of inputs
43             k {int} -- Number of clusters
44 
45         Returns:
46             ndarray -- A kx1 array of final cluster centers
47         """
```

Abbildung 10: Kmeans(...) Methode

```
In [51]:

In [51]: bandbreite_rbf
Out[51]: [1.7241956223725157, 1.843716287827387, 1.9783645473095404]

In [52]: centers_rbf
Out[52]:
array([[5.88360656, 2.74098361, 4.38852459, 1.43442623],
       [5.006     , 3.428     , 1.462     , 0.246     ],
       [6.85384615, 3.07692308, 5.71538462, 2.05384615]])

In [53]:
```

Abbildung 11: Standardabweichung und Zentren bei K=3

Wenn wir Nun Zwei anderen Spalten unserer Trainingsdaten betrachten, stellt man, die festgelegten Zentren anders aussieht.
Die Abbildung unten Zeigt wie die Position der Zentren mit anderen Eingabevektoren gehändert wurde.



5.3 Training der Gewichtsmatrix

Sind die Zentroiden und die Steuerungsparameter festgelegt, muss das Netz noch auf die entsprechende Ausgabe hin trainiert werden. Dazu hat man mehrere Möglichkeiten um die Gewichtsmatrix W zu trainieren. Man kann entweder mit dem Gradientenabstiegsverfahren oder mit der Pseudoinverse die Gewichtsmatrix trainieren. In dieser Hausarbeit beschränken wir uns auf die Pseudoinverse . Dazu wird zunächst die Aktivierung der Neuronen in der Hidden-Schicht berechnet.Dieses wird dann später wie die Eingabe des einstufigen Netzes behandelt.
Wie im vorherigen Abschnitt gesehen, die Aktivierung des Netzes ist eine $k \times N$ Matrix, wobei K die Anzahl der Cluster (Anzahl der Zentren) und N die Beobachtungen ist. Die Aktivierung-Matrix wird durch folgende

Matrix :

$$\varphi = \begin{pmatrix} \varphi^{(1)}(x_1) & \varphi^{(2)}(x_1) & \cdots & \varphi^{(b)}(x_1) \\ \varphi^{(1)}(x_2) & \varphi^{(2)}(x_2) & \cdots & \varphi^{(b)}(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ \varphi^{(1)}(x_n) & \varphi^{(2)}(x_n) & \cdots & \varphi^{(b)}(x_n) \end{pmatrix} = \begin{pmatrix} \varphi_1 \\ \varphi_2 \\ \vdots \\ \varphi_n \end{pmatrix} \in \mathbb{R}^{n \times (b+1)}$$

beschrieben.
Man berechnet danach die Pseudoinverse der Aktivierung-Matrix φ, um die Gewichtsmatrix berechnen zu können. Durch die folgenden Gleichungen beschreiben:

$$\varphi^{+\mathbf{1}} = (\varphi^{\mathbf{T}}\varphi)^{-}\mathbf{1}\varphi^{\mathbf{T}}$$

.

$$\Longrightarrow \omega = (\varphi^{\mathbf{T}}\varphi)^{-}\mathbf{1}\varphi^{\mathbf{T}}\mathbf{Y}$$

Wenn man auch die obige dargestellte Gleichung sieht, merkt man, dass es sich um ein lineares Gleichungssystem geht. Dieses lässt sich ganz normal lösen.
Wenn die Gewichtsmatrix schon bekannt ist können wir nun den Ausgang vorhersagen. Das machen wir im folgenden Abschnitt, indem wir den Algorithmus durch einen Klassifikationsproblem verwenden.

6 Anwendung auf die Klassifikation

RBF-Klassifizierer verwenden im Wesentlichen eine Teilmenge von Trainingsdaten, sodass das Ergebnis sehr wenig Speicher benötigt. Wir verwenden daher für den ganzen Prozess nur die zwei ersten Spalten. Und speichern 30
Für K=3 (Anzahl der Zentren):

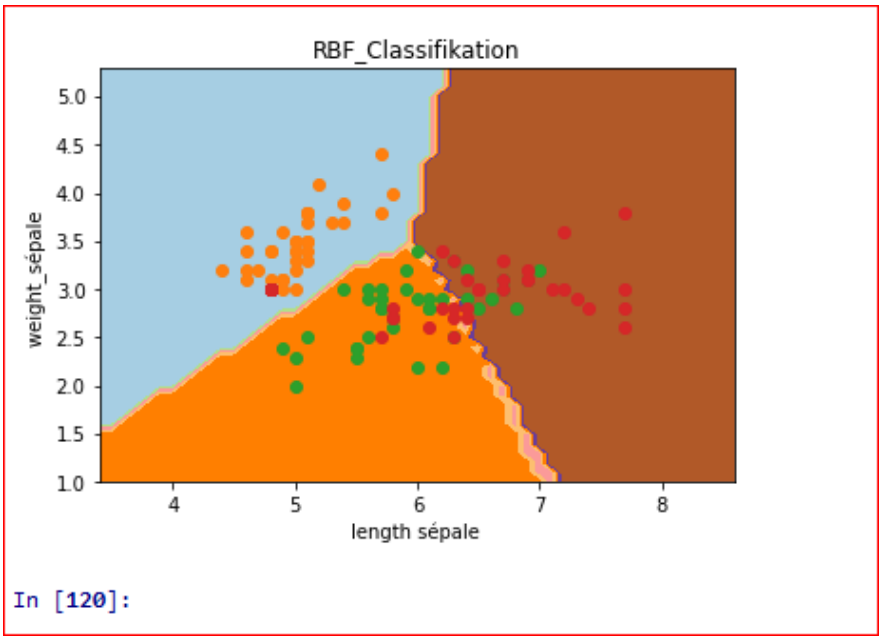


Abbildung 12: RBF Klassifikator mit K = 3

Die Abbildung zeigt wie mit 3 Zentren unser Algorithmus Daten klassifizieren kann. Aus der sind unsere drei Klassen gut klassifiziert. Es gibt zwar schlechte Datenpunkte, die Klassifiziert wurden, aber man kann sehr gut unsere drei Klassen bzw. die drei Irisblumen Variante erkennen.

Nun verändern wir die Anzahl der Zentren, also K =5 , dann wird unser Algorithmus schon schlechter und die Daten werden mit einer geringen Genauigkeit klassifiziert. Aus der Abbildung beträgt die Genauigkeit 35

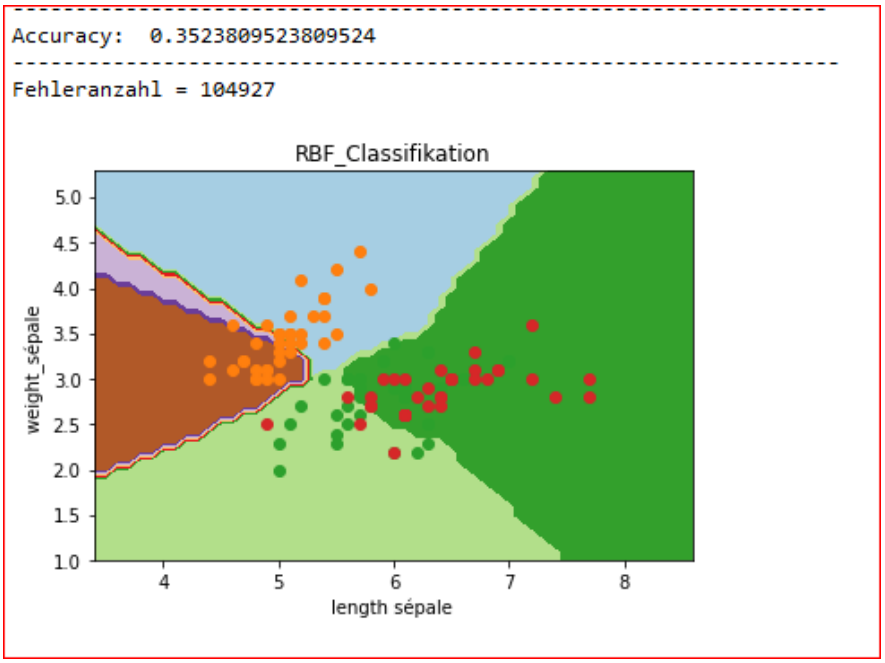


Abbildung 13: RBF Klassifikator mit K=5

wir stellen fest, wie schlecht der Algorithmus unsere Daten klassifiziert wenn die Anzahl der Zentren sich vergrößert.Daher können wir Sagen, dass je größer die Anzahl der Zentren wird, desto schlechter wird der Algorithmus. Um ein besseres Ergebnis zu haben, muss der Algorithmus mit einer Anzahl der Zentren von $K = 3$ trainiert werden.

Mit Zwei anderen Eingabevektoren aus der Trainingsdaten sieht die Klassifikation besser aus: die Abbildung darunter zeigt wie das aussieht. Hier beträgt die Anzahl der Zentren k =3

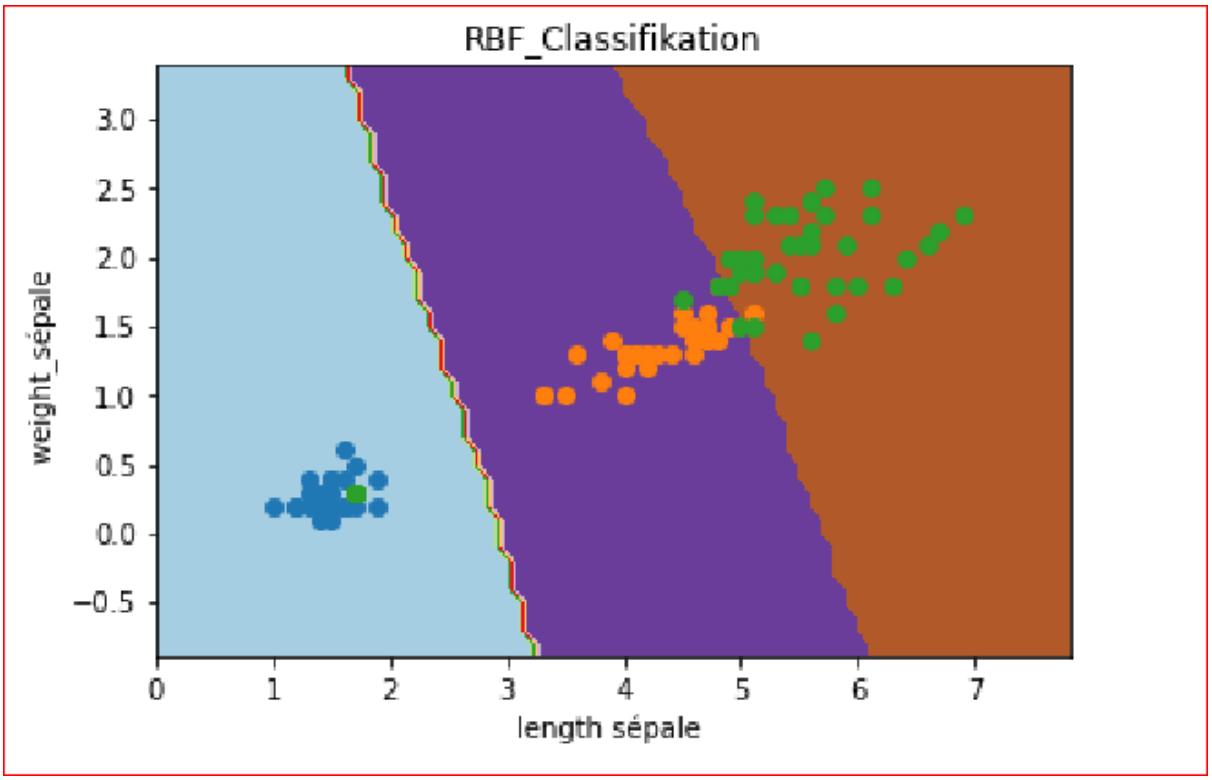


Abbildung 14: Klassifikation mit anderen Eingabevektoren

7 Unterschied mit dem Random Forest

Im Vergleich zum Random-Forest Verfahren, erhält man bessere Ergebnisse. Die Genauigkeiten bei den drei Trainingsdaten sind auch besser und sehr stabiler als die der RBF-Netze .

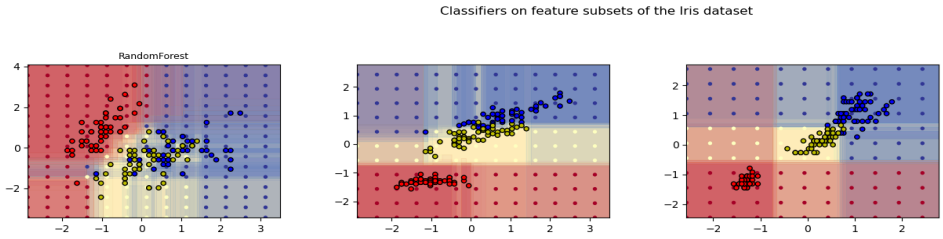


Abbildung 15: Random-Forest-Plot

```
RandomForest with 30 estimators with features [0, 1] has a score of 0.9266666666666666
RandomForest with 30 estimators with features [0, 2] has a score of 0.9933333333333333
RandomForest with 30 estimators with features [2, 3] has a score of 0.9933333333333333
```

Neuronale Radiale Basisfunktionen gehören zum Netzwerk im Allgemeinen der beste Algorithmen. Das liegt an seinen Hyperparameter-Einstellungen wie die Standardabweichung (Sigma) , und die Anzahl der Cluster K oder die Anzahl der Zentren.

8 Fazit

Für eine Approximation einer Funktion mit Hilfe des RBF-Verfahrens muss allerdings kein iteratives Lernverfahren angewendet werden. Da die Ausgabe der versteckten Neuronen bei einer einer bestimmten Eingabe berechnet werden kann, muss die Gewichtsmatrix noch, so berechnet werden, dass das Netz die gewünschte Ausgabe liefert.

Die RBFN-Netz ist eines der leistungsstarken Modelle für Klassifizierungs-Aufgaben. RBF-Netze können lernen, die zugrunde liegenden Muster mithilfe vieler RBF-Kurven zu approximieren. Die Praxis der statistischen Gleichung für den Optimierungsprozess macht den Algorithmus im Vergleich zu strukturierten MLP-Netzwerken förderlicher und schneller. Die Feinabstimmung von Hyper-Parametern wie K - Anzahl der Cluster und σ erfordert jedoch Arbeit, Zeit und Übung.

Zusammenfassen kann man folgende Methode verwenden um den RBF-Algorithmus zu trainieren:
Zentren werden durch zufällige, gleichmäßige Breiten bestimmt.
Die Gewichtungen der Ausgangsschichten werden durch ein pseudoinverses Verfahren bestimmt. Klassisches neuronales RBF-Netzwerk:

* Wir geben die Anzahl der versteckten Einheiten als einzigen Parameter an.
* Die Zentren der Gaußschen werden mit zufälligen Werten der Räume initialisiert. Die Breiten sind gleichmäßig und gegeben durch:

$$\sigma = \frac{d_{Max}}{\sqrt{2M}}$$

, wobei dMax den euklidischen Abstand ist und M die Anzahl der RBF-Knoten.Dann wird die Gewichtungen der Ausgangsschicht durch die Pseudoinverse optimiert.

  - Eine andere Alternative wäre: Bestimmung von Zentren durch k-Mittel-Segmentierung, Bestimmung von Breiten durch eine heuristische Methode von P nächsten Nachbarn und Bestimmung der Gewichte nach der Methode der kleinsten Quadrate. Das Lernen der Parameter ist hybride, da die verborgene Schicht durch Selbstorganisation optimiert wird. Während die Ausgabeschicht von einem Supervisor optimiert wird. Die Anzahl N der versteckten Einheiten muss definiert werden. Der k-

Mittelwert-Segmentierungs-Algorithmus wird verwendet, um die Zentren der verborgenen Schichten zu bestimmen. Die Breiten von RBF werden durch eine heuristische Methode von P nächsten Nachbarn bestimmt. Die Gewichte der Ausgabeschicht werden durch eine Methode der kleinsten Quadrate optimiert.

Abbildungsverzeichnis

9 Literaturverzeichnis

Machine Learning (lagout.org)

http://www.ra.cs.uni-tuebingen.de/lehre/ss06/pro$_l earning/proseminar_B R.pdf$

https://en.wikipedia.org/wiki/Radial$_b asis_f unction$

http://morere.eu/IMG/pdf/rbf.pdf (Erste Kapitel)

https://hal.archives-ouvertes.fr/hal-00085092/document

http://documents.irevues.inist.fr/bitstream/handle/2042/2240/Seghouane.pdf?