Kurze Zusammenfassung meines Aufenthaltes

29. Januar 2023

Inhaltsverzeichnis

1 Gaussprozessesregression

Die grundlegende Idee der **Gaussprozessesregression** (siehe Rasmussen und Williams [1]) ist es zu gegebenen Datenpunkten $D = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ eine Funktion $g : \mathbb{R}^d \to \mathbb{R}$ abzuschätzen unter der Annahme, dass

$$y_i = g(x_i) + \epsilon, \qquad \forall i = 1, \ldots, n. \tag{1}$$

erfüllt ist, mit $\epsilon \sim \mathcal{N}(0, \sigma^2)$. Dabei ist der Gaussprozess bestimmt durch die Mittelwertfunktion $m(x) = E\{g(x)\}$ und die Kovarianzfunktion $k(x, x') = \mathrm{cov}(g(x), g(x'))$ für $x, x' \in \mathbb{R}^d$. Die vorgegebenen Eingabevektoren $X_D = \{x_1, \ldots, x_n\}$ bestimmen dabei die Matrix

$$K := \begin{bmatrix} k(x_1, x_1) & \cdots & k(x_1, x_n) \\ \vdots & & \vdots \\ k(x_n, x_1) & \cdots & k(x_n, x_n) \end{bmatrix} \tag{2}$$

Die Funktionswerte bestimmen den Vektor $y = (y_1, \ldots, y_n)^T$. Zu einem Eingabevektor $x \in \mathbb{R}^d$ wird mithilfe des Vektors $k_x := (k(x_1, x), \ldots, k(x_n, x))$ der Erwartungswert und die Varianz des Funktionswertes $g(x)$ durch folgende Regeln abgeschätzt:

$$E\{g(x)\} = m(x) + k_x K^{-1} y \tag{3}$$
$$\mathrm{var}(g(x)) = k(x, x) + k_x K^{-1} k_x^T \tag{4}$$

Die Kernelfunktion ist ein zu bestimmender Hyperparameter. Meistens wird

$$k(x, x') = \alpha \mathbf{e}^{-(x-x')\Lambda(x-x')^T} \tag{5}$$

angenommen, wobei $\alpha > 0$ (Varianz) und die Diagonalmatrix Λ (Lengthscales) mit positiven Einträgen die Hyperparameter bilden.

2 Untersuchte Ansätze

2.1 recursiveGP

Der **recursive Gaussian Process** von Marco Huber (siehe [2]) benutzt sogenannte "Basis"-Vektoren $X_D = \{x_1, \ldots, x_n\}$ zu denen während des Trainings die dazugehörigen Funktionswerte $m(x_i) = g(x_i)$ abgeschätzt werden und diese dann zur Vorhersage in (3) und (4) anstatt der unbekannten Funktionswerte y benutzt werden. Die Variante bei der die Hyperparameter gelernt werden konnte ich nicht zum Laufen bringen. Verwendet man dagegen schon optimierte Hyperparameter, dann sind die Ergebnisse für die Datensätze $3, 5, 6, 7$ recht gut.

2.2 localGPR

Die **local Gaussian Process** Regression von Duy Nguyen-Tuong et al. (siehe [3]) basiert auf der Idee, dass man die Regression durch kleinere lokale Gaussprozess approximiert um somit dem Problem entgegen zu wirken, dass die Kovarianzmatrix K zu groß und damit die Inversenbildung K^{-1} zu kostenintensiv wird. Jeder lokale Gaussprozess deckt ein lokales Gebiet im Eingaberaum ab und wird durch ein Zentroid (=Schwerpunkt) bestimmt. Die lokalen Gebiete werden während des Trainings erzeugt: die Abstände ein neues Trainingsdatum (x, y) zu den Zentroiden der lokalen Gebiete wird ermittelt und das am nächsten gelegene Zentroid ausgewählt, wenn der Abstand klein genug ist. Dann wird das neue Testdatum in das entsprechende lokale Gebiet aufgenommen, die Inverse der neugebildeten lokale Kovarianzmatrix berechnet und ein neues Zentroid ermittelt. Ist das neue Testdatum (x, y) an keinem Zentroiden nah genug, dann wird ein neues lokales Gebiet mit Zentroid (x, y) angelegt. Die *Nähe* wird durch die Kernelfunktion k ermittelt. Ein Hyperparameter w_gen bestimmt die Schwelle, ab wann ein neues Zentroid angelegt wird. Leider ist nicht klar, wie man w_gen wählen sollte und es können leicht sehr viele oder zu wenige lokale Gebiete erzeugt werden. Der Speicherplatzverbrauch und das Berechnen der Abstände bzw. das wiederholte Berechnen von (kleineren) Inversen der lokalen Kovarianzmatrizen stellt sich als Nachteil heraus. Die Datensätze $1, 2, 3$ mit 10.000 Daten terminierten bei mir nicht.

2.3 GPEnsKF

Eine Gaussprozessregression durch einen **Ensemble Kalman Filter** wird von Kuzin et al. in [4] eingeführt. Dabei werden ähnlich wie bei Huber Basisvektoren gewählt, die hier *grid points* genannt werden. Das Positive am GPEnsKF ist, dass die Hyperparameter und die Trainingsdaten trainiert werden und zwar beim *dualGPEnsKF* nach einander. Ein sogenanntes *Ensemble* von Vektoren führt dabei eine Mittelwertberechnung der approximierten Hyperparameter und Trainingsdaten durch. Wie bei Huber's recursiveGP waren die Ergebnisse bei den Datensätzen 1,2,4 unbrauchbar. Bei den anderen Datensätzen lagen die mittleren Fehler etwas über denen von Huber's recursiveGP, aber der Vorteil war der, dass die Hyperparameter nicht durch einen vollen Gaussprozess ermittelt werden mussten.

3 Vergleich der drei Modelle anhand der Datensätze

Abbildung 1 zeigt den Mean Square Error der xyz-Koordinaten am Ausgang in mm. Hierbei ist zu beachten, dass das volle *gpflow*-Modell für die Datensätze 1-3 nur mit 2000 antrainiert worden ist. Während *dualGPEnKF*die Hyperparameter selber bestimmt, wurden für die *recursiveGP*-Modelle die Hyperparameter der antrainierten *gpflow*-Modelle verwendet. Die Hyperparameter der *localGPR*-Modelle habe ich nicht weiter angepasst. Bei größeren Datenmengen versagen diese, bzw. werden sehr langsam. Wie man an Abbildung 1 erkennt, sind die Werte für die Datensätze 1,2 und 4 unbrauchbar, obwohl man für Datensatz 3 noch recht gut Werte erhält.

datasets	*gpflow*	*dualGPEnKF*	*recursiveGP*	*localGPR*
1	0.2258	5794	15966	-
2	0.7349	127305	4314	-
3	0.0222	**0.6787**	**0.317**	-
4	0.00639	104	618	73045
5	0.00297	**0.00924**	**0.00436**	0.43205
6	0.01093	**0.02760**	**0.02007**	0.84221
7	0.01048	**0.02487**	**0.15196**	0.60805

Abbildung 1: MSE von xyz in mm

4 Bedienung der Pythonprogramme

Auf GitHub unter dem Verzeichnis code finden sich die Programme *testsuite.py* und *testGPEnKF.py* sowie die Verzeichnisse *RGP*, *localGPR* und *gpenkf*, die die Implementierung der oben beschrieben Modelle enthalten. Um zum Beispiel das *recursiveGP*-Modell auf den Datensatz 7 zu testen startet man von der Kommandozeile

python testsuite.py -m recursiveGP -d 7

Möchte man das dualGPEnKF Modell testen startet man

python testGPEnKF.py -d 7

Benutzte Packages: *tensorflow*, *gpflow*, *scikit-learn*, *scipy*, *numpy*, *matplotlib*, *tqdm*, *argparse*.

Literatur

[1] C. E. Rasmussen and C. K. I. Williams, *Gaussian processes for machine learning*. Adaptive computation and machine learning, MIT Press, 2006.

[2] M. F. Huber, "Recursive gaussian process: On-line regression and learning," *Pattern Recognition Letters*, vol. 45, pp. 85–91, 2014.

[3] D. Nguyen-Tuong, M. W. Seeger, and J. Peters, "Local gaussian process regression for real time online model learning," in *Advances in Neural Information Processing Systems 21, Proceedings of the Twenty-Second Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 8-11, 2008* (D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, eds.), pp. 1193–1200, Curran Associates, Inc., 2008.

[4] D. Kuzin, L. Yang, O. Isupova, and L. Mihaylova, "Ensemble kalman filtering for online gaussian process regression and learning," *CoRR*, vol. abs/1807.03369, 2018.