Gaussian Process Report

Budjan, Haas, Klumpp, Reitze

19 Februar 2019

Einleitung

Für unser Paket zu Gausschen Prozessen hatten wir zusammengefasst zwei Aufgaben die wir meistern mussten. Erstens das Schreiben zweier "regression functions" zur Vorhersage des Prozesses an anderen Punkten und zweitens die Bestimmung der richtigen Kovarianzfunktion für den Prozess.

Generelle Paketstruktur

Wir haben uns dazu entschieden, die Regressionen in R6 Klassen zu schreiben. Grund hierfür war die leichte Erstellung von schreibgeschützen Bereichen. Dies erschien sinnvoll, da die Algorithmen einige rechenaufwendige Schritte beinhalten, die nur einmal ausgeführt werden sollten (z.B. Cholesky Zerlegung) und somit beim Erstellen der Klasse ausgeführt werden müssen, wir aber den Nutzer davor schützen wollten beim Hinzufügen von Daten eine neue Berechnung der implizit genutzten Variablen in den Algorithmen zu vergessen.

Bei der Bestimmung der optimalen Kovarianzfunktion war dies hingegen nicht nötig. Hier ist unser Algorithmus als einfache Funktion implementiert. Hier haben wir Metaprogrammierung genutzt um Kovarianzfunktionen mit unterschiedlich vielen Parametern zu optimieren. Außerdem Error Handler um die Fehleranfälligkeit der Choleskyzerlegung und beim Invertieren abzufangen.

Um die Qualität der Ausgaben zu überprüfen und dem Nutzer eine Visualisierung der Ergebnisse zu ermöglichen, haben wir für die beiden Klassen Methoden zum Plotten der Resultate geschrieben. Dabei wird für einen Eingabevektor die Vorhersage gezeichnet und zusätzlich werden die ursprünglichen Trainingspunkte dargestellt.

Regression

Wie bereits erwähnt haben wir uns in diesem Teil für die Implementierung mit R6 Klassen entschieden. Uns erschien es als sinnvoll, dass der Nutzer einerseits einige Möglichkeiten für häufig genutzte Kovarianzfunktionen zur Verfügung hat, bei welchen er nurnoch die von ihm erwarteten Parameter eingeben muss, aber andererseits auch die Möglichkeit hat, der Klasse eine beliebige eigene Kovarianzfunktion übergeben kann. Hier muss allerdings erwähnt werden, dass für die Funktionsfähigkeit des Algorithmus eine positiv definite Kovarianzmatrix nötig ist, sodass in Wahrheit nur geeignete Funktionen eine Regression liefern können.

Als dritte Option haben wir die Möglichkeit implementiert, der Klasse eine Liste von Kovarianzfunktionen zu übergeben, sodass vor der Vorhersage durch Optimierung der Hyperparameter automatisch die am besten geeignete Kovarianzfunktion auswählt wird.

Optimierung der Hyperparameter

Die Funktion zur Optimierung der Hyperparameter - im Folgenden als fit-Funktion bezeichnet - beruht auf den Abschnitten zur Bayessian Model Selection durch die Marginal Likelihood. Wir haben uns für diese Methode entschieden, da in der Quelle Cross-validation als das numerisch aufwendigere Verfahren beschrieben wurde. Insbesondere ist bei letzterem die Invertierung der Ableitungsmatrix komplizierter.

Allerdings mussten wir das Verfahren an einigen Stellen leicht anpassen um, wie bereits erwähnt, einige Probleme des Algorithmus abzufangen. Dazu war es notwendig, dass wir uns mit den Eigenheiten der einzelnen Kovarianzfunktionen befassen. Bei konstanter und linearer Kovarianz ist die Matrix der Ableitungen

im Allgemeinen nicht invertierbar. Daher mussten wir hier die Optimierung anpassen und konnten nicht die Newton Methode mithilfe der Ableitung anwenden. Außerdem mussten wir sicherstellen, dass bei polynomiellen Kovarianzfunktionen die Exponenten aus der Menge der natürlichen Zahlen gewählt werden und somit über diskrete und stetige Parmeter optimieren. Dies haben wir gelöst, indem wir uns für den Exponenten auf die Menge $\{1, \ldots, 10\}$ beschränken und jeweils den besten Wert für den zweiten Parameter sigma bestimmen. Dann wird die beste Kombination von Werten zurückgegeben. Außerdem beinhaltet der Algorithmus wie bereits erwähnt das mehrfache Ausführen der Cholesky-Zerlegung innerhalb der Optimierung, welche leider nicht immer ausgeführt werden kann. Um dieses Problem zumindest einzugrenzen haben wir uns entschieden ein Error Handling einzubauen, das bei einem Fehler in einem Iterationsschritt die besten bisher berechneten Parameter zurückgibt zusammen mit einem Hinweis, dass eine weitere Berechnung nicht möglich ist.

Klassifizierung

Bei Klassifizierung sind wir ähnlich vorgegangen wie im Buch und wurden vor keine größeren Probleme gestellt. Jedoch war die im Buch vorgestellte Version durch das Lösen von Gleichungssystemen sehr Zeitaufwendig. Hier war dies ein deutlich größeres Problem als bei Regression, da wir mehr Punkte vorhersagen mussten. Daher haben wir uns entschieden, es zu vektorisieren und konnten so die Rechendauer um ca. 80% reduzieren. Dies haben wir dann nachträglich auch in Regression eingefügt.

Plots

Für die Plot-Methoden der Klassen haben wir das Paket `ggplot2` verwendet, aufgrund seiner einfachen Struktur mehrere Plots zu überlagern.

Shiny App

Die Shiny App stellt die wesentlichen Features des Pakets anhand von Plots dar. Wir haben hierbei die Möglichkeiten von `shiny` genutzt, um sowohl für Regression als auch für Klassifikation eine Visualisierung anzuzeigen. Außerdem haben wir durch eine dynamische Änderung des Interface die Möglichkeit erstellt für verschiedene Kovarianzfunktionen verschiedene Parameter anzupassen. Um die Ergebnisse der fit-Funktion darzustellen besitzt die App auch die Option die berechneten optimalen Parameterwerte zu verwenden.
Der Plot zur Klassifikation lässt sich durch das manuelle Hinzufügen von Punkten verändern, um die Verschiebung der Entscheidungsregionen zu beobachten. Hierbei haben wir uns nur auf 2 dimensionale Probleme beschränkt, da deren Darstellung am anschaulichsten ist.

Eigenanteil

Abschließende Bemerkungen

Was zur Teamarbeit Insgesamt war es ein spannendes Projekt, auch wenn R manchmal das Maschinentourette in einem neu erwecken kann.