Technische Universität München

Lehrstuhl für Wissenschaftliches Rechnen (SCCS)

Abschlussbericht Forschungspraktikum

Einbindung der Rosenblatt-Transformation und erweiterter Dateisupport für die SG++-Datamining-Pipeline

Abgabedatum: 24. März 2018

Autor
Lars Wolfsteller
Matrikelnr.: 03647749

Betreuer
Prof. Dr. Hans-Joachim Bungartz
Kilian Röhner, M. Sc.

1 Einführung

Das Forschungspraktikum wurde am Lehrstuhl für Wissenschaftliches Rechnen ("Scientific Computing in Computer Science") absolviert und dauerte von Dezember 2017 bis März 2018. Der Lehrstuhl entwickelt unter anderem eine Datamining-Pipeline auf Basis von dünnen Gittern, die *General Sparse Grid Toolbox SG++*[1]. Sie implementiert adaptive Dünngittermethoden und bietet verschiedene Datamining-Verfahren wie Clustering oder Regression. Der Inhalt des Forschungspraktikums war dabei, den Lehrstuhl bei der Entwicklung der SG++-Toolbox zu unterstützen.

Die Arbeit wurde in zwei verschiedene Aufgabenbereiche unterteilt. Zum einen sollte, auch um den Einstieg in die Toolbox und die Implementierung der Pipeline zu erleichtern, die Unterstützung eines weiteren Datenformats (neben ARFF-Dateien) für den Dateninput implementiert werden. Zum anderen sollte ein Verfahren gefunden werden, um mehrdimensionale Daten(punkte), die unter Umständen eine ungleichmäßige Dichteverteilung aufweisen, gleichmäßig über den Raum zu verteilen. Dies stellte den Hauptteil des Forschungspraktikums dar.

2 Unterstützung des CSV-Formats

Die SG++-Toolbox bot bisher nur die Unterstützung des ARFF-Formats (Attribute-Relation File Format) an. Geplant ist jedoch, dass die Pipeline möglichst viele Dateiformate wie CSV, XML, einfache Textdateien oder SQL-Formate unterstützt. Im Rahmen des Forschungspraktikums wurde dabei die Unterstützung von CSV-Dateien implementiert. Hierbei wurden folgende Änderungen vorgenommen:

- Geänderte Klassen:
  - **DataSourceBuilder**: In der Methode assemble() Abfrage für Dateityp um CSV erweitert und als SampleProvider einen CSVFileSampleProvider zurückgeben.
  - **DataSourceConfig**: Verfügbare Datentypen in DataSourceFileType um CSV erweitert.
  - **DataSourceFileTypeParser**: Abfrage für Dateiendung um "csv" erweitert und DataSourceFileType CSV zurückgeben; durch FileTypeMap_t DataSourceFileType CSV mit String "CSV" verknüpfen.
- Neue Klassen:
  - **CSVFileSampleProvider**: FileSampleProvider der verschiedene Methoden wie getNextSamples(), getAllSamples(), getDim(), readFile(fileName), ... für CSV anbietet. Grundgerüst anderer FileSampleProvider übernommen und die jeweiligen Methoden angepasst.
  - **CSVTools**: Klasse um die CSV-Datei mittels Methoden wie readCSV(..), readCSVSize(..) einzulesen. Da CSV-Format ähnlich zum ARFF-Data-Format und die Datenfelder statt in einem eigenen Feld meist nur in der ersten Zeile definiert werden, Anpassung auf Format und Berücksichtigung ob die erste Zeile übersprungen werden soll.

Die oben genannte Änderungen lassen sich problemlos auch auf andere Datenformate übertragen. Während das meiste lediglich Anpassungen auf den jeweiligen Dateinamen sind, müssen in der Klasse *Dateityp*Tools die "eigentlichen" Änderungen entsprechend dem jeweiligen Dateiformat vorgenommen werden.

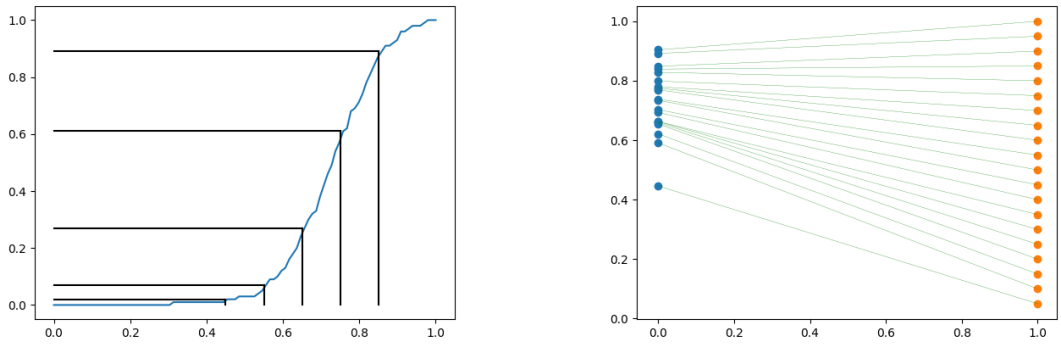
3 Rosenblatt-Transformation

Der Hauptteil des Forschungspraktikums bestand wie oben beschrieben darin, ein Verfahren zu finden, um Daten von einer ungleichmäßigen Verteilung zu einer annähernden Gleichverteilung zu transformieren. Der Nachteil einer sehr konzentrierten Verteilung (mit vielen Datenpunkten in einem oder wenigen kleinen Gebieten in $[0,1]^d$) ist dabei, dass sich das adaptive Gitter sehr stark verfeinern muss, um diese Datenpunkte gut abzubilden und dadurch viele andere Gitterpunkte unnötig verwendet werden. Bei einer

[1] http://sgpp.sparsegrids.org/

Gleichverteilung wird das Gitter somit gleichmäßiger und insgesamt besser genutzt. Nach anfänglicher Recherche zeigte sich, dass die Wahrscheinlichkeitsintegral-Transformation hierfür geeignet ist. Sie ist auch bekannt unter dem Namen Rosenblatt-Transformation, benannt nach Murray Rosenblatt, der sie 1952 das erste Mal beschrieb.

Für eine beliebige Zufallsvariable X (1D) mit kontinuierlicher Verteilung ist die Verteilungsfunktion (cumulative distribution function (CDF)) gegeben durch F_X. Die Zufallsvariable $Y = F_X(X)$ ist dann gleichverteilt auf dem Intervall $[0, 1]$.



(a) Beispielhafte Verteilungsfunktion

(b) Wahrscheinlichkeitsintegral-Transformation

Abbildung 1

Nimmt man beispielsweise die ersten 100 Werte der ersten Dimension des DR5-Trainingsdatensatzes (ein Beispieldatensatz der SG++-Toolbox), erhält man die Verteilungsfunktion in Bild 1a. Man sieht, dass insbesondere viele Werte zwischen 0.6 und 0.9 vorkommen (größere Steigung). Die Wahrscheinlichkeitsintegral-Transformation "löst" diesen Bereich dann besser "auf", erkennbar an schwarzen Markierungen in Bild 1a. Diese haben auf der x-Achse alle den gleichen Abstand 0.1, sind auf der y-Achse jedoch unterschiedlich weit voneinander entfernt sind.

Bild 1b zeigt anhand der ersten 20 Werte der ersten Dimension des DR5-Trainingsdatensatzes nochmals die Häufung der Punkte im Intervall $[0.6, 0.9]$. Durch die Wahrscheinlichkeitsintegral-Transformation sind diese dann gleichverteilt im Intervall $[0, 1]$.

Formeller ausgedrückt und für mehr Dimensionen ergibt sich folgende Definition.

3.1 Mathematischer Hintergrund

Die Rosenblatt-Transformation bildet einen Zufallsvektor mit absolut stetiger, beliebiger Verteilungsfunktion auf einen Zufallsvektor ab, dessen Zufallsvariablen gleichmäßig auf dem k-dimensionalen Hypereinheitswürfel verteilt sind. Sei $\boldsymbol{X} = (X_1, ..., X_n)$ dieser Zufallsvektor und $F_{\boldsymbol{X}}(X_1, ..., X_k)$ seine Verteilungsfunktion. Sei $z = (z_1, ..., z_k) = Tx = T(x_1, ..., x_k)$, wobei die Transformation gegeben ist durch:

$$\begin{aligned} z_1 &= P\{X_1 \leq x_1\} = F_1(x_1) \\ z_2 &= P\{X_2 \leq x_2 | X_1 = x_1\} = F_2(x_2|x_1) \\ &\vdots \\ z_k &= P\{X_k \leq x_k | X_{k-1} = x_{k-1}, \cdots, X_1 = x_1\} = F_k(x_k|x_{k-1}, \cdots, x_1) \end{aligned} \tag{1}$$

Der dadurch entstehende Zufallsvektor $Z = TX$ bzw. die durch ihn gegebenen Zufallsvariablen $Z_1, ..., Z_k$ sind gleichverteilt auf dem Einheitsintervall und voneinander unabhängig. Für den zugehörigen Beweis siehe beispielsweise die originale Veröffentlichung [6] oder [4].

4 Implementierung

Die Rosenblatt-Transformation als solche war zufälligerweise bereits im Sourcecode der SG++-Toolbox vorhanden. Daher galt es im Rahmen des Forschungspraktikums, sie an geeigneter Stelle in der Datamining-Pipeline einzubauen. Es wird zunächst kurz die vorhandene Implementierung der eigentlichen Transformation, anschließend die Einbindung dieser in die Datamining-Pipeline beschrieben.

4.1 Implementierung in der SG++-Toolbox (C++)

Im Sourcecode der SG++-Toolbox finden sich zum einen Implementierungen der Rosenblatt-Transformation für lineare Grids für eine sowie mehrere Dimensionen, zum anderen auch die Rosenblatt-Transformation mithilfe eines Kerndichteschätzers (KDE). Da dieser in der späteren Implementierung nicht genutzt wird, wird hier nur auf die "lineare" Transformation eingegangen.

Die vorhandene Implementierung teilt die Transformation in verschiedene Methoden für eine Dimension und mehrere Dimensionen auf:

- **doTransformation1D(base::Grid*** *grid*1*d*, **base::DataVector*** *alpha*1*d*, **double** *coord*1*d*) nimmt das Grid und dessen hierarchische Überschüsse *alpha*1*d* sowie die zu transformierende Koordinate als Variablen. Mithilfe des Grids und Alpha berechnet die Methode zunächst die Dichte bzw. Probability Density Function (PDF). Anschließend summiert es die Fläche unterhalb dieser Funktion auf und berechnet damit die Verteilungsfunktion bzw. Cumulative Distribution Function (CDF). Ist dies geschehen, sucht die Methode das passende Intervall für den gegeben Wert *coord*1*d* innerhalb der x-Koordinaten der CDF und interpoliert den entsprechenden y-Wert gegebenenfalls. Als Ergebnis gibt sie dann diesen berechneten y-Wert zurück.

- **doTransformation(base::DataVector*** *alpha*, **base::DataMatrix*** *points*, **base::DataMatrix*** *pointscdf*, **size_t** *dim_start*) berechnet zunächst die Randverteilung (siehe 3.1) für *dim_start* und ruft mithilfe des berechneten Grids und Alpha doTransformation1D(...) für diese Dimension auf. Anschließend ruft sie rekursiv doTransformation_in_next_dim(...) auf und transformiert die Koordinaten für restlichen Dimensionen

- **doTransformation(base::DataVector*** *alpha*, **base::DataMatrix*** *points*, **base::DataMatrix*** *pointscdf*) berechnet die Rosenblatt-Transformation für den Fall, dass keine Start-Dimension gegeben ist. Sie berechnet zunächst alle möglichen Randverteilungen und anschließend die jeweilige Start-Dimension für jedes Sample (alle paar Samples wird die Start-Dimension geändert, um so den Error der Interpolation gleichmäßig zu verteilen). Für jedes Sample berechnet ruft sie dann die entsprechende doTransformation1D(...)-Funktion auf

4.2 Einbindung in die Datamining-Pipeline

Zu verarbeitende Daten werden zunächst von der DataSource-Klasse gemäß der übergebenen Config mit einem FileSampleProvider eingelesen, eine CSV-Datei beispielsweise mit dem oben beschriebenen CSVFileSampleProvider und CSVTools. Die DataSource-Klasse gibt dann ein vom vorigen Dateiformat unabhängiges Dataset bestehend aus Samples zurück. Falls eine Transformation der Daten gewünscht ist, so besteht hier die Möglichkeit, sie durchzuführen, bevor das Dataset vom Fitter oder Scorer weiterverarbeitet wird. In die Klasse DataSource wurde daher eine entsprechende Abfrage eingefügt.

Die Rosenblatt-Transformation soll in der SG++-Toolbox später nur eine von mehreren möglichen Transformationen sein. Aus diesem Grund wurde zum einen, neben der eigentlichen Wrapper-Klasse, die die bereits implementierte Rosenblatt-Transformation aufruft, eine abstrakte DataTransformation-Klasse geschaffen, die die Struktur für die verschiedenen Datentransformationen vorgibt. Zum anderen wurde eine Factory-Klasse DataTransformationBuilder geschaffen, die die in der Config spezifierte jeweilige DataTransformation instanziiert. Auf der zurückgegebenen DataTransformation wird dann die eigentliche Transformation des Datasets ausgeführt.

Darüber hinaus wurde die Konfiguration über json-Dateien erweitert: einmal um eine allgemeine DataTransformation-Struktur die den Typ der Datentransformation bestimmt, zusätzlich noch eine für die Rosenblatt-Transformation spezifische Konfigurationsstruktur mit verschiedenen Parametern für den

Learner, der die PDF des Grids annähert. Insgesamt wurden somit folgende Änderungen für die Implementierung der Rosenblatt-Transformation vorgenommen:

- Geänderte Klassen:
  - **DataMiningConfigParser**: Neue Methode hasDataTransformationConfig() ob DataTransformationConfig vorhanden sowie parseDataTransformationConfig() und parseRosenblattTransformationConfig() für die Abfrage der Parameter für die Berechnung der Probability Density Function (PDF) (darunter die Anzahl *numSamples* die verwendet werden sollen oder Optionen für den Solver); Methode getDataSourceConfig(...) um Abfrage für DataTransformationConfig und RosenblattTransformationConfig erweitert.
  - **DataSource**: Erweiterung um Abfrage ob Transformation gewünscht, falls ja Aufrufen des DataTransformationBuilders und anschließendes Ausführen der Transformation.
  - **DataSourceConfig**: Um neue Konfiguration dataTransformationConfig erweitert (in der weitere Optionen festgelegt sind).
- Neue Klassen:
  - **DataTransformation**: Abstrakte Klasse die die Struktur für die jeweiligen Datentransformation wie z.B. Rosenblatt-Transformation definiert (mit Methoden doTransformation(dataset) und doInverseTransformation(dataset)).
  - **DataTransformationBuilder**: Factory-Klasse um mittels buildTransformation(config, dataset) entsprechend der gegebenen Config die gewünschte DataTransformation für ebenfalls übergebenes Dataset zu initialisieren.
  - **DataTransformationConfig**: Konfigurationsdatei für den Typ der DataTransformation und Transformations-spezifische Einstellungen wie RosenblattTransformationConfig. Später um die Konfigurationen anderer Datentransformationen erweiterbar.
  - **RosenblattTransformationConfig**: Konfigurationsdatei für Parameter der Rosenblatt-Transformation wie *numSamples*, *gridLevel* oder Einstellungen für den Solver.
  - **DataTransformationTypeParser**: Hilfsklasse für den DataSourceConfigParser um String-Repräsentation der DataTransformation in eigentliche DataTransformationType und zurück zu verwandeln.
  - **RosenblattTransformation**: Wrapper-Klasse für die eigentliche Rosenblatt-Transformation. Bei der Initialisierung wird mit in Config festgelegten *numSamples* Samples des Datensets und weiteren Parametern eine Dichtefunktion angenähert und dadurch Grid und die hierarchischen Überschüsse *alpha* berechnet. Diese werden dann für die Rosenblatt-Transformation bei doTransformation(dataset) bzw. doInverseTransformation(dataset) verwendet.
- Tests:
  - **JSON-Dateien**: dataMiningConfig.json um DataTransformationConfig und RosenblattTransformationConfig erweitert; neue Konfigurationsdatei test_RosenblattTransformation_in_Pipeline.json für die beiden neuen Tests.
  - **DataMiningConfigParserTest** (geändert): Erweiterung des bestehenden Tests um Abfragen für die implementierte DataTransformationConfig und RosenblattTransformationConfig.
  - **test_RosenblattTransformation_in_Pipeline** (neu): Boost-Tests um Implementierung zu überprüfen:
    - **testRosenblattWrapper**: Test ob die Wrapper-Klasse funktioniert. Dabei wird mittels Config und DataTransformationBuilder eine DataTransformation des Typs Rosenblatt initialisiert und nur über diese doTransformation() und mit dem erhaltenen Dataset doInverseTransformation() aufgerufen und schließlich das ursprüngliche und das zweifach transformierte Dataset verglichen.

* **testDataTransformationParser**: Test um Abfrage in DataSource.cpp zu überprüfen. In der beschriebenen Konfigurationsdatei wird der DataTransformationType Rosenblatt sowie weitere Optionen spezifiert und im Test die Rosenblatt-Transformation mit dem gleichen Dataset einmal über DataSource und einmal über den DataTransformationBuilder aufgerufen und die transformierten Datasets miteinander verglichen.

4.3 Test der Implementierung

Wie oben beschrieben wurden zwei Boost-Tests für die Einbindung der Rosenblatt-Transformation in die Datamining-Pipeline implementiert, eine um die Wrapper-Klasse, die andere um den DataTransformationParser bzw. die Abfrage im DataSource-Modul zu überprüfen. Dabei wurde beim ersten Test jeweils ein ursprüngliches Sample mit dem hin- und rücktransformierten Sample, beim zweiten Samples des "automatisch" über das DataSource-Modul transformierten Datasets mit solchen des "manuell" im Wrapper (also nur über die Wrapper-Klasse) transformierten verglichen. Wenn der Unterschied pro Samplevergleich eine Toleranz (je nach Test 10^{-10} bis 10^{-14}) nicht überschritt, war der Test erfolgreich.

Beim bereits erwähnten DR5-Datensatz mit über 370000 Samples gab es dabei einige Fehler (im zweistelligen Bereich). Ein paar davon waren numerisch (sie überschritten die vorgegebene Toleranz), was bei so vielen Datenpunkten jedoch vernachlässigbar ist. Die anderen (30) waren NaN-Fehler (Not a Number), sie waren auf fünf besondere Samples zurückzuführen. Diese hatten eine Koordinate $x = 0.0$ oder $x = 1.0$, lagen also in einer Dimension genau auf dem Rand des Hypereinheitswürfels. Wie es in der bestehenden Implementierung der Rosenblatt-Transformation zu diesem Fehler kommen kann, muss noch untersucht werden. Bei kleineren Datensets verliefen die Tests ohne Probleme.

5 Zusammenfassung und Ausblick

Im Rahmen des Forschungspraktikum wurden primär zwei Änderungen an der SG++-Toolbox des Lehrstuhls für Wissenschaftliches Rechnen vorgenommen. Zum einen wurde als Einstieg in das Projekt der Dateisupport um das Format CSV erweitert. Dies lässt sich leicht auch auf andere Datenformate übertragen. Zum anderen, was auch den Hauptteil der Arbeit darstellte, wurde ein Verfahren gefunden, um beliebig verteilte Datenpunkte bzw. Datensets zu einer annähernden Gleichverteilung zu transformieren: die Wahrscheinlichkeitsintegral- oder Rosenblatt-Transformation. Die bereits vorhandene reine Implementierung dieser wurde in die Datamining-Pipeline eingebunden. Dabei wurde softwaretechnisch ein Grundgerüst geschaffen, welches es erlaubt, weitere Datentransformationen sowie deren Konfiguration schnell und einfach zur bestehenden Implementierung hinzuzufügen und somit dem Anwender weitere Verarbeitungsmöglichkeiten in der Pipeline anzubieten.

Literatur

[1] Probability integral transform. https://en.wikipedia.org/w/index.php?title=Probability_integral_transform&oldid=81866372, 2018.

[2] Rosenblatt-transformation in Python. http://nbviewer.jupyter.org/github/mlocs/ipython-nb/blob/master/Rosenblatt%20Transformation.ipynb, 2018.

[3] Fabrizio Durante. The multivariate probability integral transform.

[4] Benjamin-Philip Lamers. Transformation archimedischer Copulas. page 72.

[5] Régis Lebrun and Anne Dutfoy. A generalization of the Nataf transformation to distributions with elliptical copula. 24(2):172–178.

[6] Murray Rosenblatt. Remarks on a Multivariate Transformation. 23(3):470–472.