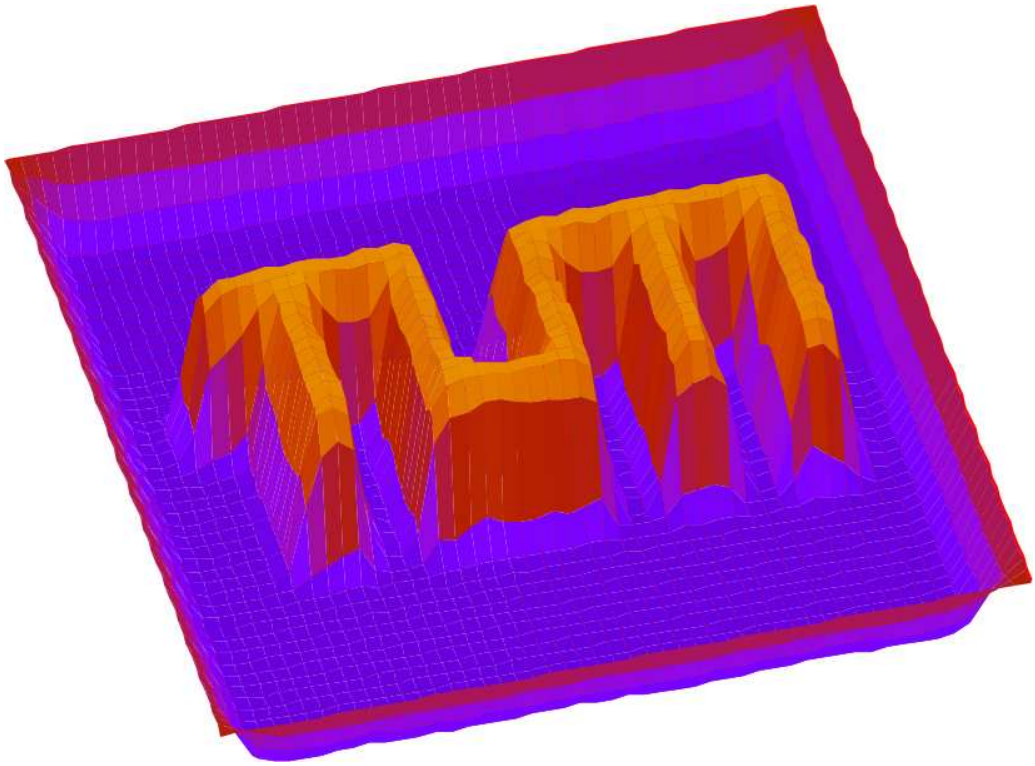
Klassifikation auf dünnen Gittern

SEP: Eine performante Implementierung in C++ eines Klassifizierers als Interpolationsproblem auf dünnen Gittern

Jörg Blank, Richard Röttger
{blankj, roettger}@in.tum.de



Inhaltsverzeichnis

1 Einleitung

1.1 Das Problem

Wir fassen das Klassifizierungsproblem als Interpolationsproblem auf, in dem wir annehmen, dass die uns zur Verfügung stehenden Trainingsdaten

$$S = \{(\mathbf{x}_i, y_i) \in \mathbb{R}^d \times \mathbb{R}\}_{i=1}^{M} \tag{1}$$

Auswertungen einer unbekannten Funktion f sind. Anhand dieser Trainingsdaten rekonstruieren wir die gesuchte Funktion, aber um eine eindeutige Lösung zu erhalten, stellen wir noch bestimmte Glattheitsbedingungen an den Interpolanten. Somit beschränkt sich das Problem auf folgende Formel mit gegebenen Trainingsdatensatz:

$$R(f) = \frac{1}{M} \sum_{i=1}^{M} \Psi(f(\mathbf{x}_i), y_i) + \lambda \Phi(f) \tag{2}$$

Hier misst Ψ den Interpolationsfehler. Für Ψ benutzen wir in unserem Programm:

$$\Psi(x, y) = (x - y)^2 \tag{3}$$

Die in (2) in Ψ benutzte Funktion f ist unser Interpolant

$$f = \sum_{j=1}^{N} \alpha_j \phi_j(\mathbf{x}) \tag{4}$$

wobei N die Anzahl der vorhandenen Gitterpunkte bestimmt. Als letztes ist noch Φ, der „Glattheitsoperator“ zu erwähnen. Wir benutzen:

$$\Phi(f) = \|\Delta f\|_2^2 \tag{5}$$

Nun suchen wir das Minimum von R, also das beste Verhältnis zwischen minimalern Interpolationsfehler und maximaler Glattheit. Dieses Minimum erreichen wir durch die Nullierung der ersten Ableitung von $R(f)$. Eine genauere Beschreibung dieses Prozesses und dessen Korrekheit wurde in [1] gezeigt. Wir geben hier nur das Ergebnis an:

$$\sum_{j=1}^{N} \alpha_j \left[M\lambda(\Delta\varphi_j, \Delta\varphi_k)_{L_2} + \sum_{i=1}^{M} \varphi_j(\mathbf{x}_i) \cdot \varphi_k(\mathbf{x}_i) \right] = \sum_{i=1}^{M} y_i \varphi_k(\mathbf{x}_i) \tag{6}$$

Hier stehen die φ_i für die hierachischen Basisfunktionen des dünnen Gitters. In Matrixschreibweise erhält man nun:

$$(\lambda C + B \cdot B^T)\alpha = By \tag{7}$$

1.2 Die effiziente Lösung

Für die Lösung des linearen Gleichungssystems (7) bietet sich die Verwendung des Verfahrens der konjugierten Gradienten an. Das die Matrizen die nötigen Bedingungen erfüllen wurde in [1] gezeigt. Die Anwendung von B bzw. B^T auf einen Vektor ist sehr effizient und kanonisch implementiert, da die Matrizen nur dünn besetzt sind und zur Berechnung nicht aufgestellt werden müssen.
Schwierigkeiten bereitet hingegen die Anwendung von C, eine $N \times N$-Matrix, wobei N die Anzahl der verwendeten Gitterpunkte ist. Zur Berechnung haben wir den UpDown-Algorithmus [3] verwendet. Aber selbst dieser effiziente Algorithmus benötigt mit seiner Laufzeit von $\mathcal{O}(2^d * N)$ (N bezeichent die Anzahl der Gitterpunkte, d die Dinmension des Gitters) einen erheblichen Anteil der Rechenzeit, und verlangsamt gerade bei hochaufgelösten Gittern mit vielen Basisfunktionen die Berechnung spürbar.

2 Alternative Glattheitsoperatoren

2.1 Neue Methoden

Unsere Aufagabe war neben der Implementierung des „korrekten“ Lösers auch das finden alternativer Glattheitsbedingungen mit einer deutlich besser berechenbaren Matrix C. Das Prinzip der neuen Glattheitsbedingung besteht nun darin die einzelnen Ausschläge verschiedener Basisfunktionen zu „bestrafen“. Auf diese Weise erhält man verschiedenste Diagonalmatrizen, die allesamt sehr leicht zu berechnen sind. Wir haben verschiedenste Methoden implementiert und deren Wirksamkeit untersucht. Die Methoden waren:

- **C** - klassischer Laplace Operator
- **I** - Einheitsmatrix
  Als C wird einfach nur die Einheitsmatrix benutzt, d.h. alle Basisfunktionen werden unabhängig von Ihrem Level oder Grundfläche gleich gewichtet.
- **R** - quadratische Basisfunktionen werden bevorzugt
  Die Diagonalelemente werden mit
  $$c_{ii} = \frac{1}{2} \log \left(1 + \left(\frac{l_{max}}{l_{min}} \right) \cdot d \right)$$
  berechnet. Hier ist l_{max} das größte, im Gridindex i vorkommende Level, l_{min} das ensprechende Minimum. Leicht zu erkennen ist, dass die Funktion für $l_{max} = l_{min}$ den kleinsten Wert liefert, und somit die quadratischen Basisfunktionen bevorzugt.
- **E** - Basisfunktionen entsprechend Energienorm-Gitter bevorzugt
  Im Gegensatz zu R werden nicht die quadratischen Funktionen bevorzugt, sondern die Basisfunktionen mit einer möglichst schmalen Grundfläche. Hier ist eine Analogie zur Verwendung der Energienorm zu beachten, die eine ähnliche Basisfunktionsstruktur besitzt. Die Berechnungsvorschrift ist:
  $$c_{ii} = \frac{1}{l_{max} - l_{min} + 1} \cdot d$$
  Hier ist der Wert am kleinsten, wenn der Unterschied zwischen größtem Level und dem Kleinstem möglichst groß ist. Abbildung 1 visualisiert die verschiedenen Bevorzugungen.

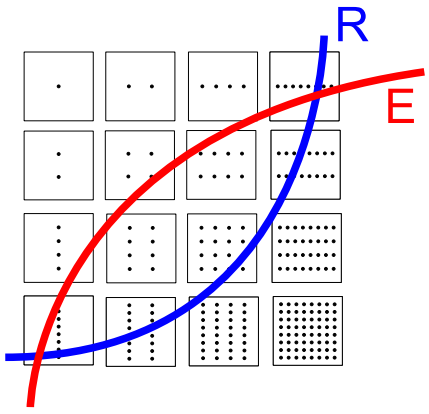


Abbildung 1: Bevorzugte Basisfunktionen der Methoden R und E

- **L** - Summe aller Level einer Basisfunktion
  Bei dieser Methode wird von einem Gridindex die Summe aller Level gebildet, also $|\,\mathbf{l}\,| = \sum_{i=1}^{d} l_i$ und durch das Level des Gitters normiert:

$$c_{ii} = \frac{|\,\mathbf{l}\,|}{l_{Gitter}}$$

- **F** - Summe der Zeilen der klassischen C-Matrix als Eintraege einer Diagonalmatrix
  Bei der letzten Methode wird ein Vektor mit den Zeilensummen der klassischen Laplace C-Matrix gebildet. Das heißt die Diagonale der neuen Matrix C ist der Ergebnisvektor von Vektor $v = (1, 1, \ldots, 1)$ angewandt auf das echte C.

2.2 Vergleich der Methoden

Um die Funktionsweise der verschiedenen Methoden zu vergleichen haben wir diverse Standard-Datensätze durchgerechnet. Für alle Datensätze wurden verschiedene Werte von λ benutzt um so ein optimales Ergebnis für die einzelnen Methoden zu erzielen. Verwendet wurden:

- **Bupa-Liver**
  6 dimensionaler Datensatz mit 345 Punkten. Auf diesen Datensatz wurde die 10-fold cross-validation angewandt. Die einzelnen Klassen wurden zufällig generiert, allerdings stratifiziert, d.h. das Verhältnis der beiden Klassifizierungsklassen ist in allen Mengen nahezu gleich. Die Berechnung wurde auf einem dünnen Gitter mit Level 3 durchgeführt. Siehe Abbildung 2 auf Seite 5.

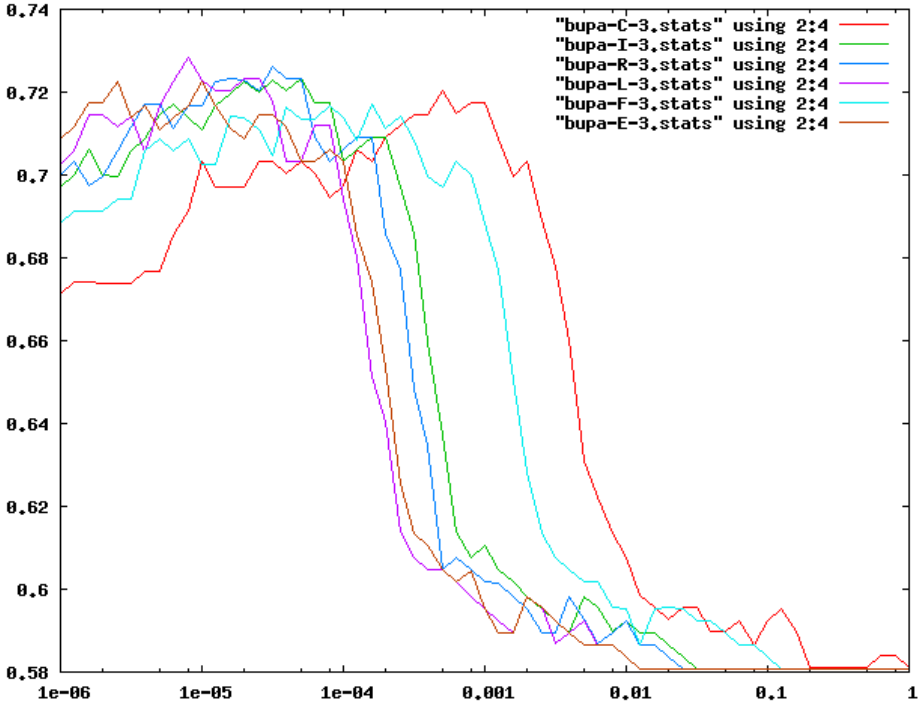


Abbildung 2: Verschiedene Methoden für C im Vergleich mit Bupa-Liver-Datensatz, dünnes Gitter Level 3

- **Ripley**
  2 dimensionaler Datensatz bestehend aus 250 Trainingsdaten und 1000 Testdaten. Beide Klassen enthalten die identische Zahl von Punkten, allerdings mit einer Fehlerrate von 8%. Aus diesem

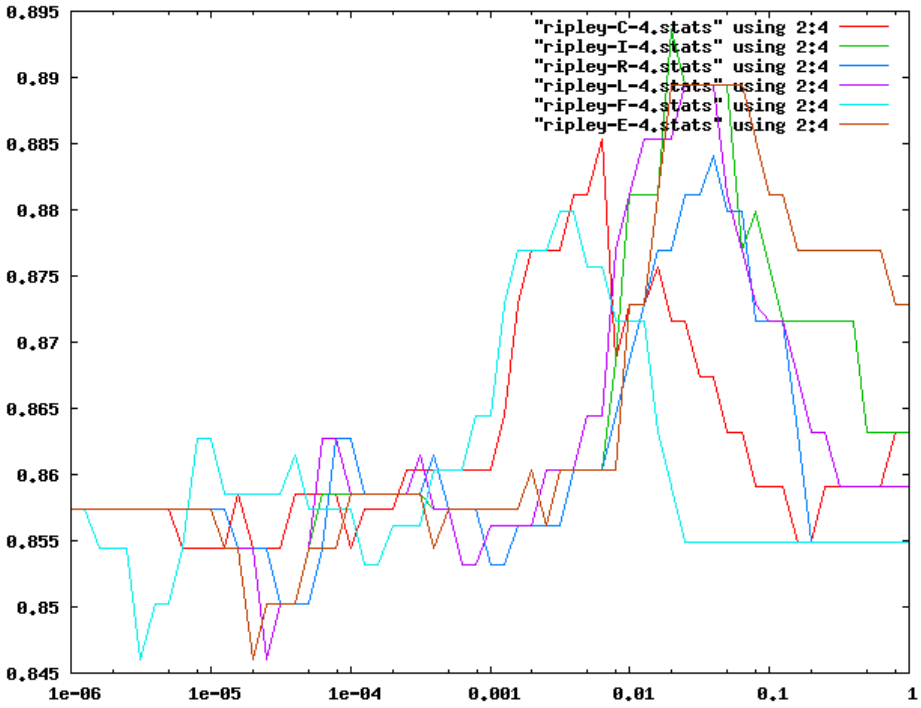


Abbildung 3: Verschiedene Methoden für C im Vergleich mit Ripley-Datensatz, dünnes Gitter Level 4

Grund kann eine Klassifizierungrate jenseits der 92% nicht erwartet werden. Dieser Datensatz wurde auf einem Gitter mit dem Level 4 klassifiziert. Siehe Abbildung 3 auf Seite 6.

- **Schachbrett-Datensatz**
  Dieser Datensatz besteht aus 1000 zufällig verteilten Punkten, die so in die Klassen eingeteilt wurden, dass ein Schachbrettmuster mit 4 Feldern pro Achse ensteht. Da hier kein Gleichgewicht zwischen den beiden Klassen herrscht, wurde für die Messung wieder die stratifiziere 10-fold cross-validation angewandt. Aufgrund der vielen einzelnen Feldern, die jeweils einer anderen Klasse zugeordnet sind, wurde ein Gitter mit einer Auflösung von 6 benutzt. Siehe Abbildung 4 auf Seite 7.

- **synthetischer, ripley-ähnlicher Datensatz**
  Ein sysnthetisch erzeugter 2 dimensionaler Datensatz mit 5000 Punkten. Klassifiziert wurde auf auf einem Gitter mit Level 4. Auf disem Datensatz wurde ebenfalls eine 10-fold cross-validation durchgeführt. Siehe Abbildung 5 auf Seite 7.

	C	I	R	E	L	F
Ripley	0.886	0.893	0.884	0.889	0.889	0.880
Bupa-Liver	0.720	0.723	0.726	0.723	0.728	0.717
Checkerboard	0.943	0.942	0.944	0.932	0.940	0.945
Eigener	0.949	0.949	0.949	0.949	0.949	0.949

Tabelle 1: Testgenauigkeit der verschiedenen Methoden auf einzelne Datensätze

Bei allen Berechnungen wurde der selbe Seed benutzt, um Verschlechterungen oder Verbesserungen auf Grund von günstig gewählten Klassen zu vermeiden. Wie man auf den Abbildungen 2 bis 5 erkennt, bewegen sich alle Methoden auf ungefähr einem Level. Die Kurven sind entlang der λ-Achse lediglich

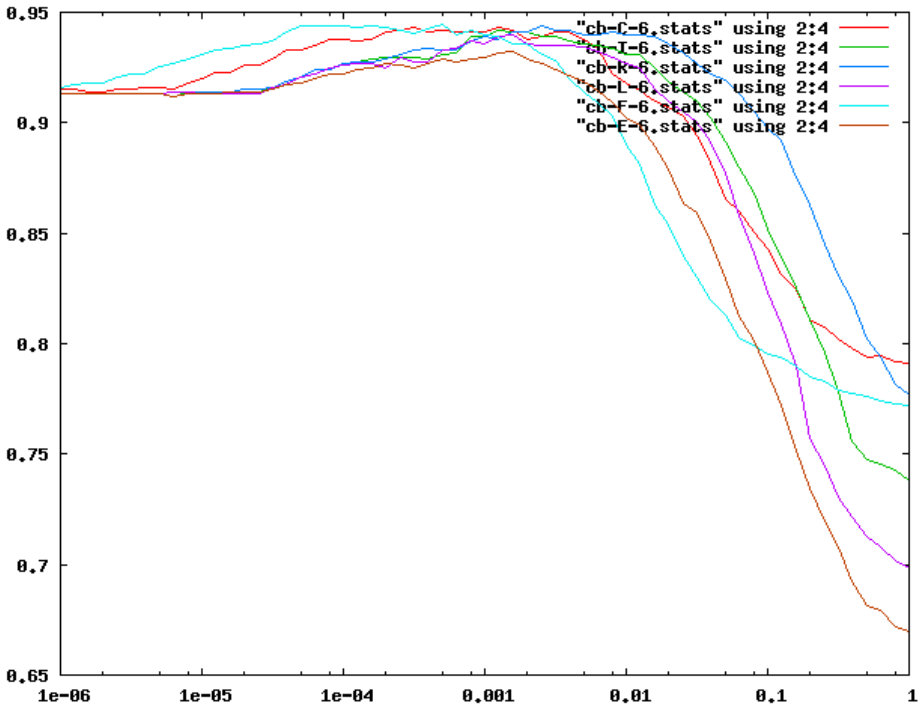


Abbildung 4: Verschiedene Methoden für C im Vergleich mit Schachbrett-Datensatz, dünnes Gitter Level 6

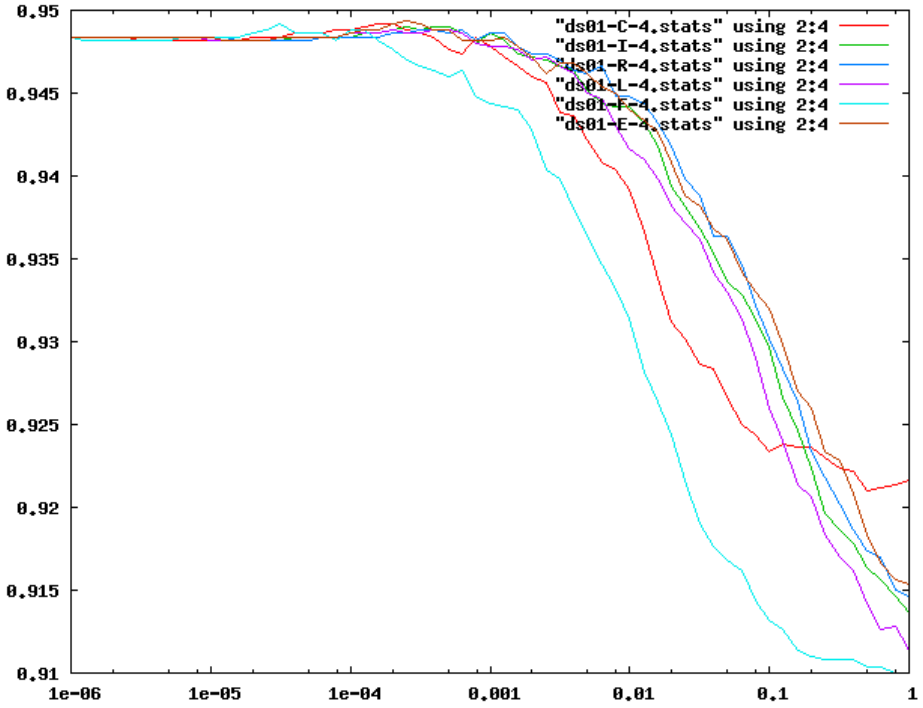


Abbildung 5: Verschiedene Methoden für C im Vergleich mit einem ripley-ähnlichen Datensatz, dünnes Gitter Level 4

verschoben, qualitativ ist kaum ein Unterschied auszumachen. Bei der Rechenzeit sind allerdings erhebliche Verbesserungen bemerkbar, da die Auswertung einer Diagonalmatrix ungleich einfacher ist, als die ursprünglich Laplace-Matrix auf einem dünnen Gitter. Die Tabelle 1 zeigt jeweils die höchste erreichte Testgenauigkeit für die einzelnen Methoden der C-Berechnung.
Abbildung 6 auf Seite 8 zeigt die Unterschiede der verschiedenen gelernten Funktionen. Die Bilder sind alle auf einem dünnen Gitter mit Level 7 gerechnet, allerdings mit einem großen λ von $0,01$ (Eine Außnahme bildet hier die Methode F - sie wurde mit $\lambda = 0,001$ gerechnet), so dass die Auswirkungen der verschiedenen Glättungsmethoden sichtbar werden. Besonders augenfällig ist die Auswirkung der Energienorm, man erkennt besonders in den 4 Ecken, dass die quadratischen Basisfunktionen am meisten „bestraft“ werden, deshalb ist deren Ausschlag sehr gering, was in diesem Fall zu sichtbaren Erhebungen führt. Des Weitern sieht man, dass nur die Laplacebedingung eine wirklich glatte Funktion liefert. Allerdings ist auch deutlich, dass die „unglatteren“ Methoden gleich gut klassifizieren, da keine Vertiefung ein ändern der Klassifikationklasse verursacht. Lediglich wenn man die Güte der Klassifizierung bewerten wollte, d.h. mit welcher Wahrscheinlichkeit ein Punkt in einer Klasse liegt, würden diese „Dellen“ eine Rolle spielen.

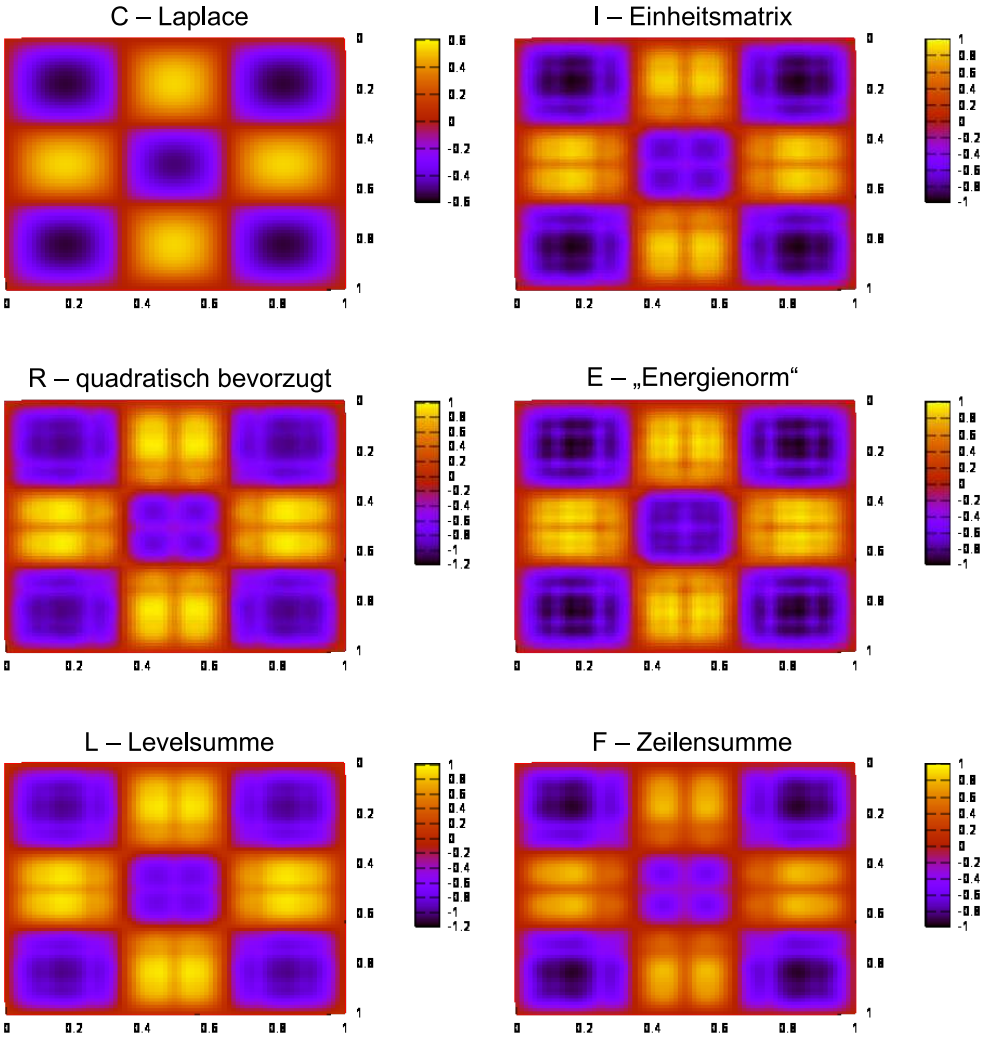


Abbildung 6: Direkte Unterschiede zwischen den einzelnen Methoden mit großem λ auf einem dünnen Gitter der Auflösung 7

2.3 Adaptiv

Nachdem wir festgestellt haben, dass die neuen Methoden sehr zuverlässig sehr gute Werte liefern, haben wir uns entschlossen die Einheitsmatrix noch auf adaptiv verfeinerten dünnen Gitter mit C zu vergleichen. Als Kriterium für eine Verfeinerung nehmen wir den Punkt des Gitters mit dem größten hierachischen Überschuss, der noch nicht sämtliche Kinder besitzt. Auch hier lieferte die Einheitsmatrix hervorragende Ergebnisse, wie man im Vergleich der maximal erreichten Klassifizierungskorrektheit sieht (Tabelle 2). Des Weiteren erkennt man eine deutliche Verbesserung bei der Klassifizierung des Bupa-Liver-Datensatzes im Gegenstatz zu einem „vollen“ dünnen Gitter um über 2%.

	Id training	Id testing	Laplace training	Laplace testing
Ripley	0.890	0.881	0.906	0.894
Bupa-Liver	0.835	0.746	0.879	0.723

Tabelle 2: Identität im Vergleich mit der Laplace-Matrix auf adaptiv verfeinerten Gittern.

Die Verbesserungen, die man mit adaptiven Verfeinerungen erziehlt, sieht man in Abbildung 7. Hierfür wurde bei einem dünnen Gitter mit Level 2 begonnen und daraufhin bis zu 20 mal verfeinert. Man kann deutlich erkennen, dass bereits nach 5 bis 10 Verfeinerungen sehr gute Ergebnisse geliefert werden. Danach verbessern die Verfeinerungen den Klassifizierungserfolg nicht mehr.

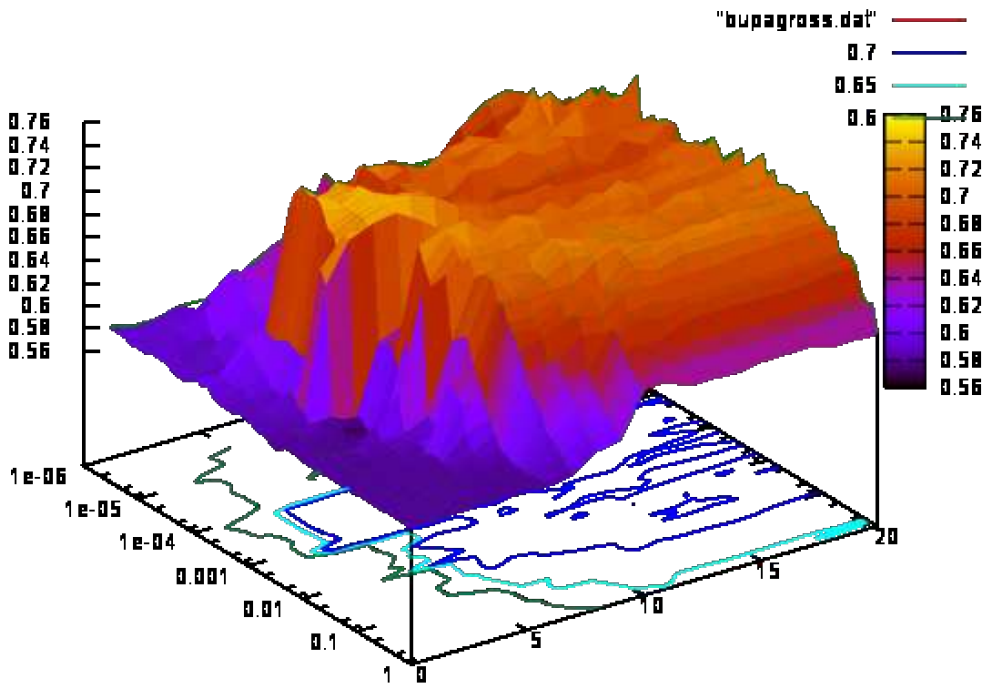


Abbildung 7: Klassifizierungskorrektheit auf Bupa-Liver-Datensatz in Abhängigkeit von λ (Linke Achse) und der Anzahl der Verfeinerungen (Rechte Achse).

3 Die Hilfsprogramme

3.1 Beschreibung

Die Hilfsprogramme bestehen aus zwei Pythonscripts, `classifier.py` und `converter.py`. Ersteres ist das Hauptprogramm mit dessen Hilfe Datensätze klassifiziert werden können. Das Programm erwartet die die Daten im `ARFF`-Format[1], allerdings bereits normiert auf das Interval $[0, 1]$ und im Falle von Trainingsdaten bereits in die Klassen $\{-1, 1\}$ eingeteilt. Die Normierung, richtige Einteilung und Ausgabe im `ARFF`-Format geschieht mit Hilfe von `classifier.py` und erwartet als Eingabedaten entweder eine Datei im `ARFF`-Format oder in einer Datei mit der schlichten Form: `x1 x2 ... xN class`.

3.2 Kommandozeilenparameter

3.2.1 converter.py

```
python converter.py  -i|--infile infiles
                     [-t|--types types]
                     [-o|--outfile outfiles]
                     [-m|--merge]
                     [-b|--border data-border]
                     [-c|--class classification-border]
                     [-C|--noclasses]
```

- `-i | --infile` *infiles*
  Diese Option bestimmt eine Eingabedatei. Für mehrere Dateien wird diese Option wiederholt. Die Normierung erfolgt nicht für jede Datei getrennt, sondern alle Dateien werden mit dem selben Faktor normiert. Wenn mehrere, unabhängige Dateien normiert werden sollen, muss das Programm mehrmals aufgerufen werden. *infiles* können Dateien im `ARFF` -Format sein, oder das eingangs erwähnte einfache Format. Die Ausgabe erfolgt in Dateien mit angehängten `{.arff}`, außer es werden Ausgabedateinamen angegeben.

- `-t | --types` *types*
  Bestimmt den Typ der ersten Eingabedatei. Bei Wiederholung dieser Option wird der Typ der zweiten, dritten usw. Eingabedatei bestimmt. Werden weniger Typen als Eingabedateien angegeben, so wird für die restlichen Dateien der Typ erraten, genauso wie wenn die Typenangabe weggelassen wird.

- `-o | --outfile` *outfiles*
  Bestimmt den Namen der Ausgabedatei für die erste Eingabedatei. Wird diese Option wiederholt, bestimmt sie den Namen der zweiten, dritten, usw. Ausgabedatei für die zugehörige Eingabedatei. Sollten mehr Eingabedateien als Ausgabedateien vorhanden sein, genauso wie wenn die Option weggelassen wird, dann werden die übrigen Eingaben mit angehängtem `{.arff}` benamt.

- `-m | --merge`
  Normiert mehrere Einagbedateien und gibt Sie zusammen als eine einzige Datei aus.

- `-b | --border` *data-border*
  Bestimmt den Rand des normierten Datensatzes, d.h. jede Dimension wird auf

  $$[0 + border, 1 - border]$$

  skaliert. Defaultwert ist $0,01$

[1] Attribute-Relation File Format, ASCII-Dateien, die eine Liste von Werten mit den gleichen Attributen enthalten. Spezifikation und genaue Beschreibung unter `http://www.cs.waikato.ac.nz/~ml/weka/arff.html`. ARFF ist im Zusammenhang mit der *Weka machine learning software* entstanden (`http://www.cs.waikato.ac.nz/~ml/weka/`).

- `-c | --class` *classification-border*
  Bestimmt die Trennlinie zwischen den Klassifizierungsdaten, d.h $x < border$ wird -1 zugeordnet, dem Rest 1. Der Defaultwert ist 0.5.

- `-C | --noclasses`
  Gibt an, das die Inputfiles keine Klassendaten besitzen, z.B. um Testdaten zu normieren.

3.2.2 classifier.py

```
python classifier.py  -m|--mode  mode
                      -l|--level level
                      -D|--dim dim
                      -a|--adaptive num
                      -C|--zeh c-methode
                      -f|--foldlevel level
                      -L|--lambda lambda
                      -i|--imax max
                      -r|--accuracy accuracy
                      -d|--data datafile
                      -t|--test testdata
                      -a|--alpha alphafile
                      -o|--outfile outfile
                      -g|--gnuplot gnuplot
                      -R|--resolution gnuplot-resolution
                      -s|--stats statisticfile
                      --seed random-seed
                      -v|--verbose
```

- `-m|--mode` *mode*
  Bestimmt die Aktion die der Classifier ausführen soll. Erlaubte Werte für *mode* sind:

  - `normal`
    In diesem Modus generiert der Classifier aus der Eingabedatei (`--data`) ein Klassifizierungsfunktion, und schreibt diese in die angegebene Alphadatei (`--alpha`). Optional kann bei 2-dimensionalen Daten auch eine gnuplot-Datei ausgegeben werden. (`--gnuplot`).

  - `test`
    In diesem Modus generiert der Classifier aus der Eingabedatei (`--data`) eine Klassifizierungsfunktion, klassifiziert daraufhin die Testdaten (`--testdaten`). Optional können die statistischen Auswertungen in eine Statistikdatei geschrieben werden (`--stats`).

  - `foldX`
    Diese modes führen einen n-fold-crossvalidation-Check durch. Die verschiedenen Foldarten unterscheiden sich lediglich in der Art und Weise wie die Eingabedaten in die einzelnen Klassen eingeteilt werden. Die Anzahl der Klassen kann mit `--foldlevel` bestimmt werden. Optional können die statistischen Auswertungen in eine Statistikdatei geschrieben werden (`--stats`).

    * `fold`
      In diesem Modus wird die Eingabedatei in gleichgroße Klassen eingeteilt. Die Daten werden dabei zufällig den einzelnen Klassen zugeordnet.
    * `folds` (sequence)
      In diesem Modus wird die Eingabedatei in gleichgroße Klassen eingeteilt. Diesmal werden die Daten blockweise zugeordnet, d.h. die ersten $\frac{gesamtdaten}{foldlevel}$ Daten werden der ersten Klasse zugeordnet, die Zweiten der Zweiten Klasse usw.

    * `foldr` (relation)
      In diesem Modus wird die Eingabedatei in gleichgroße Teile zerlegt. Die Daten werden zufällig verteilt, allerdings wird beachtet, das in allen Klassen das gleiche Verhältniss der Klassifizierungsklassen herrscht.
    * `foldf` (file)
      Es werden durch wiederholtes `--data` mehrere Eingabedateien gelesen. Jede einzelne Eingabedatei steht dabei für jeweils eine eigene Klasse. Damit wird das Foldlevel automatisch durch die Anzahl der Eingabedateien bestimmt.
  - `apply`
    Es wird eine, in einem vorangegangenen Lauf generierte Klassifizierungsfunktion eingelesen (`--alpha`) und die übergebenen Daten (`--data`) anhand dieser Funktion klassifiziert. Die Ausgabe (`--output`) enthält die Eingabedaten mit einer Zusätzlichen Klassifizierungsspalte. Diese Methode benötigt überdies die Angabe der Gitterdimension (`--dim`), da diese aus den α-Werten nicht mehr extrahiert werden können.

- `-l | --level` *level*
  Bestimmt das Level des zu benutzenden dünnen Gitters. Die Dimension ergibt sich aus den Eingabedateien.

- `-D | --dim` *dimension*
  Bestimmt die Dimension des zu verwendenden Gitters. Diese Angabe ist nur notwendig, wenn eine Alpha-Datei geladen wird, da nicht mehr herausgefunden werden kann welche Dimension und Auflösung benutzt wurde.

- `-a | --adaptive` *num*
  Bestimmt die Anzahl der adaptiven Verfeinerungen. 0 ist der normale Modus, d.h. es wird lediglich das Gitter zum angegebenen Level benutzt.

- `-C | --zeh` *c-methode*
  Dies Schlüsselwörter für die in dem Kapitel 2.1 beschriebenen C-Berechnungsmethoden lauten wie folgt:

  - `laplace`
  - `identity`
  - `ratio`
  - `levelsum`
  - `energy`
  - `copy`
  - `pseudounit`
  - `laplaceadaptive`

- `-f|--foldlevel` *level*
  Bestimmt für die Folds die Anzahl der Klassen, in die die Eingabedatei aufgeteilt wird. Dieser Wert wird nur von `fold | folds | foldr` verwendet.

- `-L|--lambda` *lambda*
  Bestimmt den Wert der Glattheitsbewertung λ. Wenn nicht angegeben, wird $0,0001$ verwendet.

- `-i|--imax` *imax*
  Bestimmt die maximale Anzahl an Durchläufen des CG-Verfahrens. Default ist 400.

- `-r|--accuracy` *accuracy*
  Definiert die Genauigkeit, bei der CG abbrechen soll. Defaultwert ist 0.0001

- `-d|--data` *datafile*
  Definiert in den verschiedenen Modi die Eingabedateien mir den Lerndaten, im Modus `apply` sind es die zu klassifizierenden Daten.

- `-t|--test` *testdata*
  Im Modus `test` werden die Testdaten angegeben.

- `-a|--alpha` *alphafile*
  Im Modus `apply` wird durch diese Option die schon berechnete Funktion bestimmt.

- `-o|--outfile` *outfile*
  Bestimmt im Modus `apply` die Datei in der die Klassifizierten Daten geschrieben werden, sonst ist es die Ausgabe für die neu berechnete Klassifizierungsfunktion.

- `-g|--gnuplot` *gnuplot*
  Definiert die Ausgabedatei für die berechneten Gnuplotdaten. Diese Funktion steht nur im 2D-Fall im Modus `normal` zu Verfügung.

- `-R|--resolution` *gnuplot-resolution*
  Optional für `--gnuplot`. Bestimmt die Anzahl der Funktionsauswertungen pro Dimension, der Defaultwert ist 50.

- `-s|--stats` *statisticfile*
  Definiert die Datei in die die ermittelte Treffergenauigkeit der Klassifizierungsfunktion geschrieben wird.

- `--seed` *random-seed*
  Definiert den zu benutzden Randomseed, um z.B. im Modus `fold` eine zwar zufällige, aber wiederholbare Einteilund der Daten erhält.

- `-v|--verbose`
  Zusätzliche Ausgabe von Informationen während der Berechnung.

Literatur

[1] J. Garcke, M. Griebel and M. Thess: *Data mining with sparse grids* Computing 67(3), p. 225-253, 2001

[2] J. Garcke: *Maschinelles Lernen durch Funktionsrekonstruktion mit verallgemeinerten dünnen Gittern* Dissertation, Mathematisch-Naturwissenschaftlichen Fakultät, Rheinische Friedrich-Wilhelms-Universität Bonn, 2004

[3] D. Pflüger: *Data Mining mit Dünnen Gittern* Diplomarbeit Nr. 2264, Institut für Parallele und Verteilte Systeme, Universität Stuttgart, 2005