Machine Learning Übungsblatt 6

Ramon Leiser Tobias Hahn

February 5, 2017

1 Begriffsdefinitionen

1.1 VC-Dimension eines Intervalls auf dem Zahlenstrahl

Intervalle auf dem reellen Zahlenstrahl sind definierbar durch zwei Punkte, dem Anfangs und dem Endpunkt. Somit muss ein Klassifikator in der Lage sein diese Menge und alle ihre Untermengen zu zerschmettern. Die VC-Dimension beträgt also 2.

1.2 Realisierbarkeitsannahme

Die Realisierbarkeitsannahme ist eine Annahme über die Verteilung D und einem dazugehörigen Sample S. Man geht i.d.R. davon aus, dass die Samples mit D über den Zustandsraum $\mathcal{X}$ verteilt sind und ihr Lables f einem Konzept $c*$ folgen so dass $f = c*(x)$. Die annahme würde nicht zutreffen wenn die Lables selbst zufällig und nicht von $\mathcal{X}$ abhängig sind, so dass $S = \{(x_n, y_n)\}$ aus der Verteilung D über $\mathcal{X} \times \mathcal{Y}$ Stammt.

1.3 Vervollständigen sie den Satz ...

Eine Hypothesenklasse $\mathcal{H}$ ist PAC-lernbar, falls es ein Lernverfahren L und eine Funktion m_H abhängig von ϵ und δ gibt, so dass für alle Verteilungen D über $\mathcal{X}$ und jede Klassifizierungsfunktion $f : \mathcal{X} \to \{0, 1\}$ unter der Realisierbarkeitsannahme und $m \geq m_H(\epsilon, \delta)$ unter D identisch und unabhängig verteilten und mit f beschrifteten Beispielen gilt,
dass die VC-Dimension von D endlich ist.

1.4 Fundamentalsatz der Lerntheorie

Der Fundermentalsatz der Lerntheorie beschreibt den Zusammenhang zwischen PAC-Lernbarkeit und der VC-Dimension. Nach Paul Fischer genügt es einen Konsistenten Hypothesen-Finder zu haben, welcher Stichproben aus C auf $\mathcal{H}$ abbildet, so dass die Hypothesen konsistent sind mit beiden Konzepten und zu zeigen, dass die VC-Dimension von $\mathcal{H}$ endlich ist um zu wissen dass nach dem PAC-Modell $\text{VcDim}(\mathcal{H})/\epsilon$ Stichproben genügen um die PAC-Bedingung zu erfüllen. [1]

1.5 No-Free Lunch

No-Free Lunch geht ursprünglich auf den Autor **Robert A. Heinlein** und den Science Fiction Roman *The Moon Is a Harsh Mistress* zurück und entwickelte sich zu einer Bezeichnung in der Mathematik und Informatik für einen gewissen Zusammenhang bei Optimierungsproblemen und somit auch im Machine-Learning. Das **No-free-Lunch-Theorem** besagt dass beim Abstrahieren von Datensätzen kein einzelner Lösungsansatz auf der Menge aller möglichen Probleme besser abschneidet als ein anderer. Jeder Lösungsansatz wäre bei einem Test auf allen möglichen Problemen nur genau so gut wie der Lösungsansatz alle Ergebnisse zu Raten. Da die meisten Probleme in der Realität z.B. den Naturgesetzen genügen müssen sich Lösungsansätze jedoch nie auf allen möglichen Problemen beweisen.

1.6 Sokoban

Bei Sokoban verschiebt man Kästen in einem rasterisierten Labyrinth. Man kann Kästen nur vor der Spielfigur her schieben und kann nur eine Kiste gleichzeitig bewegen. Es gibt eine Anzahl von vorgegebenen Zielfeldern auf die Jeweils ein beliebiger Kasten gestellt werden muss um ein Level ab zu schließen. Die Anzahl der gebrauchten Züge ergeben die Punktzahl für das Level (desto kleiner desto besser).
Dadurch dass man die Kästen nur schieben und nicht ziehen kann, ergeben sich schnell Zustände aus denen man die Kästen nicht mehr heraus bekommt, z.B. wen ein Kasten in einer Ecke im Labyrinth steht. Diese Gebilde kann man direkt mit den Wänden des Labyrinths gleich setzten, da man dadurch den Suchgraph zum lösen des Spiels einschränken würde und die Blöcke ohnehin nicht mehr bewegbar sind. Wenn es in dem Spiel dadurch jedoch weniger bewegbare Blöcke als Zielfelder gibt ist das Level sofort nicht mehr lösbar und der Suchpfad muss auch nicht weiter propagiert werden.
Um diese Teilstellungen sofort zu erkennen müsste man am Anfang des Spiels global und nach jedem Zug, der eine Kiste bewegt lokal die unmittelbare Umgebung des Spielers auf solche Zwickmühlen überprüfen.

[1] Paul Fischer *Algorithmisches Lernen* 1999 unter: https://books.google.de/books?id=34HzBQAAQBAJ&pg=PA37&lpg=PA37

1.7 PCA auf Iris Datensatz

Wir haben versucht Korrelation im Iris Datensatz auf zu decken um so eventuell die Dimensisionalität des Problems zu senken. Der Iris-Datensatz besteht aus Daten zu drei Schwertlilienarten, welche anhand der aufgeführten Merkmale fast unterscheidbar sind. Zu jeder art gibt es 50 Einträge also insgesamt 150 Datenzeilen. Die Merkmale lauten:

1. Kelchblütenblatt-Länge.
2. Kelchblütenblatt-Breite.
3. Blütenblatt-Länge.
4. Blütenblatt-Breite.

Die PCA in Weka zeigte die folgenden Eigenwerte mit zugehörigem Anteil an der gesamten Varianz an:

```
eigenvalue   proportion   cumulative
  2.91082      0.7277       0.7277      -0.581petL-0.566petW-0.522sepL+0.263sepW
  0.92122      0.23031      0.95801    0.926sepW+0.372sepL+0.065petW+0.021petL
  0.14735      0.03684      0.99485     -0.721sepL+0.634petW+0.242sepW+0.141petL
```

Der letzte Eigenwert (und somit auch der zugehörige Eigenvektor) tragen nur 3% zur Varianz der Daten bei und könnte deshalb außen vor gelassen werden.
Besser mehr Information enthalten die Eigenvektoren sind die Eigenvektoren:

```
Eigenvectors
 V1         V2         V3
-0.5224     0.3723    -0.721     sepL
 0.2634     0.9256     0.242     sepW
-0.5813     0.0211     0.1409    petL
-0.5656     0.0654     0.6338    petW
```

Der erste Eigenvektor wird relativ stark von allen Variablen beeinflusst. Man könnte ihn als die Pflanzengröße in Relation zur Kelchblüten-Breite betiteln, da die Kelchblüten-Breite in einem inversen Verhältnis zur Kelchblüten-Länge, sowie zu den anderen Variablen steht.
Der zweite Eigenvektor wird vor allem von den maßen der Kelchblüte bestimmt. Hier stehen die beiden Werte jedoch in positiver Korrelation. Man könnte diesen Eigenvektor als die Kelchblütengröße Bezeichnen.
Der dritte Eigenvektor beschreibt vor allem einen negativen Zusammenhang zwischen der Kelchblütenblatt-Breite und der Blütenblatt-Länge und ist dem ersten Eigenvektor ähnlich da er auch genau ein andersartiges Vorzeichen besitzt. Er trägt allerdings nur noch zu einem sehr kleinen Anteil der Varianz bei.
Der Letzte Eigenvektor kann nun bei der Basistransformation weggelassen werden, was die Dimensionalität des Datensatzes von vier auf drei Dimensionen senkt.
Trainiert man nun einen Naive-Bayes Klassifikator auf dem normalen und dem reduzierten Datensatz und vergleicht die Ergebnisse stellt man fest, dass der Reduzierte Datensatz wie zu erwarten nur leicht schlechter ist als die Klassifikation auf dem ganzen Datensatz. Zum Test wurde ein Cross-validation Verfahren benutzt mit 10 Faltungen. Hier ein Auszug der Ergebnisse:

- Voller Datensatz

```
=== Confusion Matrix ===

   a   b   c   <-- classified as
  50   0   0 |   a = Iris-setosa
   0  48   2 |   b = Iris-versicolor
   0   4  46 |   c = Iris-virginica

Mean absolute error                       0.0342
```

- Reduzierter Datensatz

```
=== Confusion Matrix ===

   a   b   c   <-- classified as
```

```
 50   0   0 |   a = Iris-setosa
  0  44   6 |   b = Iris-versicolor
  0   5  45 |   c = Iris-virginica

Mean absolute error                       0.0717
```

Der Mittlere absolute Fehler steigt also von 3% auf 7%. Daran erkennt man dass im Iris-Datensatz kaum redundante Korrelation der Daten steckt.

2 Apriori Algorithmus

2.1 Entdeckung von Assoziazionsregeln

Wir sind fälschlicherweise davon ausgegangen, dass der Algorithmus implementiert werden soll. Die folgende Berechnung ist der Output einer selbst geschriebenen Implementation des Algorithmus,welche mit abgegeben wird. Da das Hauptaugenmerk dieser Aufgabe nicht die Implementation ist wurde der Quellcode nur minimal dokumentiert. Die Ausgabe wurde aber mit einem Resultat einer Implementierung im Weka-Toolkit verglichen und ist korrekt.

```
Transactions :
Transaction size: 1[Wasser]
Transaction size: 2[Brot, Bier]
Transaction size: 2[Cola, Bier]
Transaction size: 2[Wasser, Saft]
Transaction size: 3[Schokolade, Chips, Cola]
Transaction size: 3[Saft, Cola, Wein]
Transaction size: 3[Saft, Cola, Bier]
Transaction size: 3[Schokolade, Schinken, Brot]
Transaction size: 4[Saft, Cola, Bier, Wein]
```

Der erste Teil des Algorithmus besteht darin große, mehrmals vorkommende Itemsets in der Eingabe zu finden.

```
-----------------------------------
|   Apriori     Part       One    |
|   Generate Large Item Sets       |
-----------------------------------

---------------- k - Equals : 0 ----------------
[       aprioriGen input :[Schokolade][Wasser][Saft][Brot][Bier][Cola][Wein]]
[      after aprioriGen step join :[Bier, Cola][Wasser, Saft][Schokolade, Cola][Schokolade,
    Bier][Wasser, Brot][Schokolade, Wein][Schokolade, Brot][Wasser, Bier][Cola, Wein][
    Schokolade, Wasser][Brot, Wein][Brot, Cola][Saft, Cola][Schokolade, Saft][Saft, Brot][
    Saft, Bier][Wasser, Wein][Brot, Bier][Wasser, Cola][Saft, Wein][Bier, Wein]]
[     after aprioriGen step prune :[Bier, Cola][Wasser, Saft][Schokolade, Cola][Schokolade,
    Bier][Wasser, Brot][Schokolade, Wein][Schokolade, Brot][Wasser, Bier][Cola, Wein][
    Schokolade, Wasser][Brot, Wein][Brot, Cola][Saft, Cola][Schokolade, Saft][Saft, Brot][
    Saft, Bier][Wasser, Wein][Brot, Bier][Wasser, Cola][Saft, Wein][Bier, Wein]]
```

Hier wurden die neuen Hypothesen durch die Apriori-Gen Methode erzeugt. Es handelt sich im ersten Schritt um zweielementige Mengen.

```
new apriori-gen:[Bier, Cola][Wasser, Saft][Schokolade, Cola][Schokolade, Bier][Wasser, Brot
    ][Schokolade, Wein][Schokolade, Brot][Wasser, Bier][Cola, Wein][Schokolade, Wasser][Brot
    , Wein][Brot, Cola][Saft, Cola][Schokolade, Saft][Saft, Brot][Saft, Bier][Wasser, Wein][
    Brot, Bier][Wasser, Cola][Saft, Wein][Bier, Wein]
Support[Bier, Cola]       is 0.33333334
Support[Wasser, Saft]       is 0.11111111
Support[Schokolade, Cola]       is 0.11111111
Support[Schokolade, Bier]       is 0.0
Support[Wasser, Brot]       is 0.0
Support[Schokolade, Wein]       is 0.0
Support[Schokolade, Brot]       is 0.11111111
Support[Wasser, Bier]       is 0.0
Support[Cola, Wein]       is 0.22222222
Support[Schokolade, Wasser]       is 0.0
Support[Brot, Wein]       is 0.0
```

```
Support[Brot, Cola]       is 0.0
Support[Saft, Cola]       is 0.33333334
Support[Schokolade, Saft]       is 0.0
Support[Saft, Brot]       is 0.0
Support[Saft, Bier]       is 0.22222222
Support[Wasser, Wein]       is 0.0
Support[Brot, Bier]       is 0.11111111
Support[Wasser, Cola]       is 0.0
Support[Saft, Wein]       is 0.22222222
Support[Bier, Wein]       is 0.11111111
survivors: [Bier, Cola][Saft, Cola][Saft, Bier][Saft, Wein][Cola, Wein]
```

Alle Mengen, die nicht dem minimalen Support entsprechen wurden entfernt. Basierend auf den noch vorhandenen Mengen werden nun dreielementige Mengen mit Apriori-Gen erzeugt.

```
---------------- k - Equals : 1 ----------------
[       aprioriGen input :[Bier, Cola][Saft, Cola][Saft, Bier][Saft, Wein][Cola, Wein]]
[      after aprioriGen step join :[Saft, Bier, Wein][Saft, Bier, Cola][Bier, Cola, Wein][
    Saft, Cola, Wein]]
[     after aprioriGen step prune :[Saft, Bier, Wein][Saft, Bier, Cola][Bier, Cola, Wein][
    Saft, Cola, Wein]]

new apriori-gen:[Saft, Bier, Wein][Saft, Bier, Cola][Bier, Cola, Wein][Saft, Cola, Wein]
Support[Saft, Bier, Wein]       is 0.11111111
Support[Saft, Bier, Cola]       is 0.22222222
Support[Bier, Cola, Wein]       is 0.11111111
Support[Saft, Cola, Wein]       is 0.22222222
survivors: [Saft, Bier, Cola][Saft, Cola, Wein]

---------------- k - Equals : 2 ----------------
[       aprioriGen input :[Saft, Bier, Cola][Saft, Cola, Wein]]
[      after aprioriGen step join :[Saft, Bier, Cola, Wein]]
[     after aprioriGen step prune :[Saft, Bier, Cola, Wein]]

new apriori-gen:[Saft, Bier, Cola, Wein]
Support[Saft, Bier, Cola, Wein]       is 0.11111111
survivors:
None! Iteration Over.!
```

Es sind nun keine Mengen mehr vorhanden, die noch den minimalen Support erfüllen. Größere Mengen hätten einen noch kleineren Support und werden deshalb nicht verfolgt.

```
Large Itemsets:
<Item Set:[Bier, Cola] has support0.33333334
<Item Set:[Wasser] has support0.22222222
<Item Set:[Saft, Bier, Cola] has support0.22222222
<Item Set:[Saft] has support0.44444445
<Item Set:[Cola] has support0.5555556
<Item Set:[Saft, Cola, Wein] has support0.22222222
<Item Set:[Cola, Wein] has support0.22222222
<Item Set:[Schokolade] has support0.22222222
<Item Set:[Saft, Cola] has support0.33333334
<Item Set:[Brot] has support0.22222222
<Item Set:[Saft, Bier] has support0.22222222
<Item Set:[Bier] has support0.44444445
<Item Set:[Wein] has support0.22222222
<Item Set:[Saft, Wein] has support0.22222222
```

Im zweiten Teil des Algorithmus werden nun Regeln erzeugt. Zunächst wird den großen Itemsets jeweils ein Element entnommen, auf das von den restlichen Einträgen aus geschlossen werden soll.

```
-------------------------------
|  Apriori   Part        Two  |
|  Generate  Association Rules |
-------------------------------
Confidence[Cola]-->[Bier]       is 0.59999996
Confidence[Bier]-->[Cola]       is 0.75
Confidence[Bier, Cola]-->[Saft]       is 0.6666666
Confidence[Saft, Cola]-->[Bier]       is 0.6666666
Confidence[Saft, Bier]-->[Cola]       is 1.0
```

```
Confidence[Cola, Wein]-->[Saft]       is 1.0
Confidence[Saft, Wein]-->[Cola]       is 1.0
Confidence[Saft, Cola]-->[Wein]       is 0.6666666
Confidence[Wein]-->[Cola]       is 1.0
Confidence[Cola]-->[Wein]       is 0.39999998
Confidence[Cola]-->[Saft]       is 0.59999996
Confidence[Saft]-->[Cola]       is 0.75
Confidence[Bier]-->[Saft]       is 0.5
Confidence[Saft]-->[Bier]       is 0.5
Confidence[Wein]-->[Saft]       is 1.0
Confidence[Saft]-->[Wein]       is 0.5
```

Alle Hypothesen unter der minimalen Confidence werden entfernt. Es bleiben lediglich Hypothesen die von den folgenden Mengen ausgehen. Da all diese Hypothesen auf `Cola` oder `Saft` Schließen werden diese beiden Element für den Apriori-Gen verwendet.

```
first hypotheses: [Saft][Saft, Bier][Bier][Wein][Saft, Wein][Cola, Wein]

[      aprioriGen input :[Saft][Cola]]
[     after aprioriGen step join :[Saft, Cola]]
[    after aprioriGen step prune :[Saft, Cola]]

new apriori-gen:[Saft, Cola]
Confidence[Bier]-->[Saft, Cola]       is 0.5
0.5
Confidence[Wein]-->[Saft, Cola]       is 1.0
1.0
survivors:
None! You Done!
```

Damit ist der Algorithmus beendet und es folgen die Ergebnisse.

```
Saft  --->  [[Cola]]
Saft  Bier  --->  [[Cola]]
Bier  --->  [[Cola]]
Wein  --->  [[Saft], [Saft, Cola], [Cola]]
Saft  Wein  --->  [[Cola]]
Cola  Wein  --->  [[Saft]]
```

2.2 Verbesserung des grundlegenden Algorithmus

Eine Variante vom Apriori-Algorithmus ist der Max-Miner. Er Arbeitet ähnlich wie der Apriori, generiert jedoch nicht alle häufigen Teilmengen aus den Transaktionen sondern direkt die maximal großen. Er arbeite auf einer Baumstruktur.
Außerdem fanden wir den M-Apriori Algorithmus, welcher beim generieren der Itemsets speichert welches der Items im Set den kleinsten Support hatte. Außerdem Besteht zu jedem Item eine Referenz in welchen der Transaktionen es auftaucht. Nun kann zum bestimmen des Supports darauf zurückgegriffen werden und nur diese Transaktionen nach dem Itemset durchsucht werden.
Durch ihre Verbesserungen sind die beiden Algorithmen schneller als der normale Apriori-Algorithmus ohne sein Ergebniss zu verändern, nur durch das einsparen von unnötigen Lookups und Erzeugung von Sets die dem minimalen Support nicht standhalten.

3 GVGAI Controller

3.1 General idea

Die generelle Idee ist es, den OLMCTS Controller zu verwenden, und einige Modifikationen im Heurismus anzubringen. Der bisherige Heurismus ist ein ganz simpler Algorithmus, der nur den Game Score in Betracht zieht und ob gewonnen oder verloren wurde. Dieser Heurismus wurde um drei Modifikationen erweitert:

1. Strafe für Opposite Actions

Um den Controller beizubringen, nicht dauernd Aktionen zu machen die im Endeffekt keinen Effekt haben, wurde eine Strafe für Opposite Actions eingeführt. Hierbei bekommt ein Zug eine Strafe, wenn er die Effekte vom letzten Zug rückgängig macht - beispielsweise nach hinten geht, wenn vorher nach vorne gegangen wurde. Diese Züge bekommen nur eine Strafe und werden nicht komplett entfernt, weil sie unter Umständen Sinn machen können - wenn man sich bspw. nicht bewegen will.

2. Strafe für blocking Movements

   Auch hier eine Strafe, jedoch für Züge die blocking sind, d.h. beispielsweise in eine Mauer hineinbewegen. Dies kann auch unter Umständen Sinn machen, ist meistens jedoch sinnlos und wird daher bestraft.

3. Pheromonsystem

   Um dem Controller beizubringen das Level zu erkunden, und nicht nur auf einer Stelle zu bleiben, wurde ein Pheromonsystem eingebracht. Bei diesem stößt der Avatar Pheromone aus, an der Stelle an der er sich befindet. Diese Pheromone diffundieren und verblassen über Zeit, und werden vom Score abgezogen, was heißt dass der Player eher angehalten ist global zu exploren statt an einer Stelle zu bleiben.

3.2 Werte

Die Werte für die jeweiligen Strafen bzw. Berechnungen wurden dem Paper Open Loop Search for General Video Game Playing entnommen. Folgende Werte sind dabei von Bedeutung: Strafe für Opposite Actions: 0.1 Strafe für Blocking Movements: 100 Parameter für Pheromondiffundierung: 0.4 Parameter für Pheromonverblassung: 0.99

3.3 Formeln

Im Grunde wurden zwei Formeln implementiert: Die Heuristik-Regel sowie die Pheromonberechnung auf welcher diese aufbaut. Die Formeln sind: Value = Game Score + WIN if WIN - LOSE if LOSE - Strafe für Opposite Action if Opposite Action - Strafe für Blocking Movements if Blocking Movement - Pheromon an der Stelle

Für die Pheromonberechnung werden die benachbarten Zellen in Betracht gezogen, also links, rechts, oben und unten von der Zelle die betrachtet wird. p_sigma ist dabei der Durchschnittspheromonwert dieser Zellen. p_df ist der Diffundierungsparameter, p_dc der Verblassungsparameter. $p(zelle) = p_df * p_sigma + (1 - p_df) * p_dc * p(zelle)$

3.4 Code

Heuristik

```
public double value(StateObservation a_gameState, ACTIONS last, ACTIONS secondToLast) {
    boolean gameOver = a_gameState.isGameOver();
    Types.WINNER win = a_gameState.getGameWinner();
    double rawScore = a_gameState.getGameScore();

    if(gameOver && win == Types.WINNER.PLAYER_LOSES)
        rawScore += HUGE_NEGATIVE;

    if(gameOver && win == Types.WINNER.PLAYER_WINS)
        rawScore += HUGE_POSITIVE;

    //penalty for taking reverse action
    if (ACTIONS.reverseACTION(last) == secondToLast) {
        rawScore -= PEN_OA;
    }

    //penalty for using a blocked movement
    if (!a_gameState.getEventsHistory().isEmpty()) {
        Event lastEvent = a_gameState.getEventsHistory().last();
        if (ACTIONS.isMoving(last) && lastEvent.gameStep == a_gameState.getGameTick()) {
            rawScore -= PEN_BM;
        }
```

```
    }

    rawScore -= a_gameState.getPheromones();

    return rawScore;
}
```

Pheromonupdate

```
private void updatePheromones() {
    float sigma = 0;
    int neighbours = 0;

    for (int row = 0; row < this.pheromones.length; row++) {
        for (int col = 0; col < this.pheromones[row].length; col++) {
            //calc sigma
            sigma = 0;
            neighbours = 0;
            if (row < this.pheromones.length - 1) {
                sigma += this.pheromones[row+1][col];
                neighbours++;
            }
            if (row > 0) {
                sigma += this.pheromones[row-1][col];
                neighbours++;
            }
            if (col < this.pheromones[row].length - 1) {
                sigma += this.pheromones[row][col+1];
                neighbours++;
            }
            if (col > 0) {
                sigma += this.pheromones[row][col-1];
                neighbours++;
            }
            sigma /= neighbours;

            //calc this value
            float old = this.pheromones[row][col];
            this.pheromones[row][col] = PDF * sigma + (1 - PDF) * PDC * old;
        }
    }
}
```