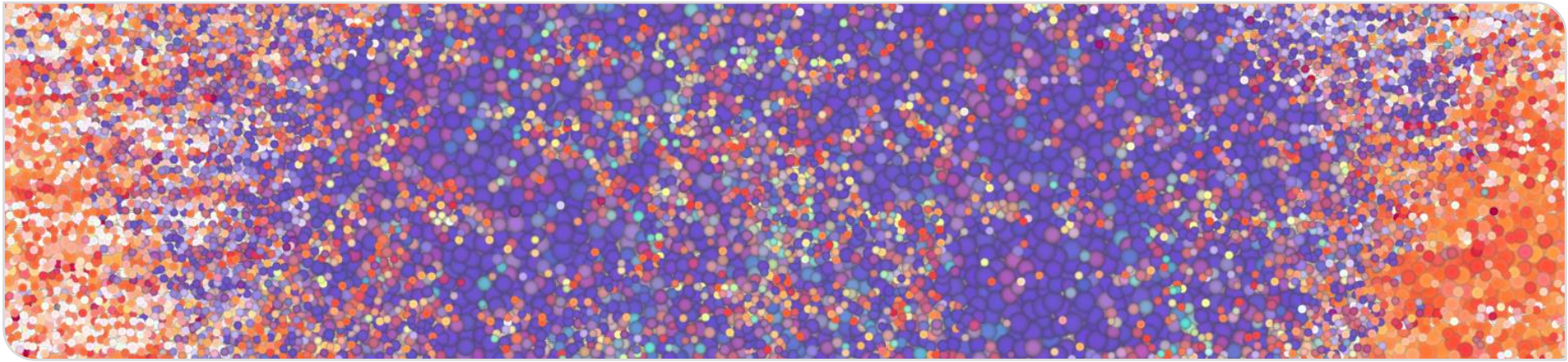
Karlsruher Institut für Technologie

AGD – WS 2020/21 – Übungssitzung 3

Jakob Bach (jakob.bach@kit.edu)



Zeitplan

Woche	Datum*	Übung
1	03.11.2020	
2	10.11.2020	
3	17.11.2020	Abgabe Blatt 1
4	24.11.2020	
5	01.12.2020	Abgabe Blatt 2
6	08.12.2020	Sitzung 1
7	15.12.2020	Abgabe Blatt 3

Woche	Datum*	Übung
8	22.12.2020	
9	12.01.2021	Abgabe Blatt 4
10	19.01.2021	Sitzung 2
11	26.01.2021	Keine Abgabe
12	02.02.2021	Sitzung 3
13	09.02.2021	Abgabe Blatt 5
14	16.02.2021	Sitzung 4

* regulärer Vorlesungstermin am Dienstag; Abgabetermin der Übungsblätter kann ein anderer Tag sein

Agenda

Inhalt: Übungsblätter 4 und 5, Vorlesungskapitel 6-9

- Umfrage: Vorbereitung auf Sitzung
- Eure Fragen: Code-Aufgaben, Online-Tests, Sonstiges
- Theorie-Aufgaben
- Kurze Feedback-“Runde“

Ankündigungen:

- Qualifikationsaufgabe für das Praktikum (Übungsblatt 6) veröffentlicht.

Fragen (1)

- Wo finde ich die Online-Tests im ILIAS?
  - Übungsordner, und darin Unterordner für jedes Übungsblatt
  - Pro Übungsblatt jeweils ein Online-Test (abgesehen von Übungsblatt 6)

- Warum ist die Abgabefrist der Qualifikationsaufgabe für das Praktikum nicht später, z. B. Mitte März?
  - aus Tradition ;-)
  - Haben in Übungssitzung Umfrage gemacht, wird auf Mitte März verschoben
  - Jakob muss Abgaben vor Praktikumsbeginn auch noch bewerten ;-)

Fragen (2)

- Wie viel Zeit benötigt man für die Qualifikationsaufgabe?
  - Hängt von Programmierkenntnissen ab
  - Erwartungswert bei entsprechenden Vorkenntnissen (z. B. wie in Programmieraufgaben der AGD-Übung vermittelt): nicht mehr als ein paar Stunden
  - Aufgabenstellung beachten: es geht nicht um perfekte Vorhersage, sondern darum, gewisse Programmierkenntnisse und Verständnis von Data Science zu zeigen

Fragen (3)

- Ist die Abgabe der Qualifikationsaufgabe eine Bewerbung für das Praktikum oder kann man auch nur das Blatt abgeben?
  - Blatt kann natürlich auch als Übung gelöst werden
  - Allerdings keine Musterlösung und kein Feedback (außer Information, ob man zum Praktikum zugelassen wird)
  - Abgabe des Blattes verpflichtet nicht zur Teilnahme an Praktikum, ist nur Voraussetzung für Zulassung (reguläre Prüfungsanmeldung findet in der ersten Woche des Praktikums statt)

Fragen (4)

- Was sind die Anforderungen für ein „einfaches“ / „kompliziertes“ Modell in der Qualifikationsaufgabe?
  - siehe Präsentation und Video zum Data-Science-Prozess
  - “einfach” ist subjektiv; es geht im Wesentlichen darum, zwei verschiedene Vorhersagemodelle (+ eine Baseline) einzusetzen und zu vergleichen

Theorieaufgaben – Apriori Candidate Generation

- Warum muss man nicht alle Teilmengen für das Pruning verwenden?
  - Beim Support Counting fliegen die Kandidaten-Itemsets sowieso raus, wenn sie nicht frequent sind
  - Pruning ist für Korrektheit des Algorithmus somit nicht notwendig
  - Pruning verringert aber Aufwand für Support Counting

Theorieaufgaben – Apriori-Trick

- Wie nutzt der Apriori-Algorithmus die Tatsache aus, dass jede Teilmenge eines Frequent Itemsets ebenfalls frequent ist?
  - Eigenschaft heißt Anti-Monotonizität
  - Wenn eine Teilmenge nicht frequent ist, kann die Obermenge nicht frequent sein (ist die logische Umkehrung der Aussage in der Frage)
  - Solche Obermengen (Frequent-Itemset-Kandidaten) können beim Pruning-Schritt im Apriori-Algorithmus entfernt werden

Theorieaufgaben – Apriori vs. FP-Trees

- Wann ist Apriori teuer? Wozu nutze ich hierbei die Anti-Monotonizität aus? Wie helfen FP-Trees gegen diese Herausforderungen?
  - Apriori ist teuer, wenn Frequent Itemsets groß (für jede Itemset-Größe Datenbankscan zwecks Support Counting, Generate&Test-Paradigma)
  - Apriori ist teuer, wenn viele Frequent Itemsets (wenn dagegen nur wenige Itemsets wirklich frequent, gibt es wenige Kandidaten und Pruning ist effektiv)
  - Anti-Monotonizität hilft beim Pruning der Itemsets
  - Bei FP-Tree Datenbank nur zweimal gescannt, unabhängig von Größe der Frequent Itemsets, danach alle Operationen auf dem FP-Tree selbst
  - FP-Tree befindet sich im Hauptspeicher, was Zugriffe auf Transaktionen billiger macht (falls immer noch zu groß, dann Partitionierung des Trees)
  - Transaktionen sind in FP-Trees bereits zusammengefasst (ähnliche Transaktionen teilen sich Pfade im Baum, Häufigkeiten gezählt)

Theorieaufgaben – Erweiterungen von Apriori

- Welche Verfeinerungen gibt es für den Apriori-Algorithmus und welche Beobachtungen nutzen sie jeweils aus?
  - Apriori-B: nicht alle Itemset-Größen k prüfen, sondern k zunächst in größeren Schritten erhöhen, und wenn nicht frequent, Werte dazwischen untersuchen
  - Hash-Filter: beim Support Counting für Größe k auch bereits Support Counting für Hashes von Itemsets der Größe k+1; insbesondere hilfreich für zwei-elementige Frequent Itemsets, da dort Pruning nicht funktioniert
  - Sampling: Frequent Itemsets auf Sample des Datenbestandes bestimmen, Konfidenzintervalle für die Grenze zwischen frequent / nicht frequent (Negative Border) bestimmen, für Itemsets innerhalb dieses Bereichs dann Support Counting auf Gesamtdaten
  - Constraints für Constraint-basiertes Pruning (je nach Anwendungsfall)

Theorieaufgaben – Succintness (1)

- Erklären Sie Succinctness anhand eines Gegenbeispiels!
  - Vorlesungsdefinition: „wenn man alle Itemsets, die das Constraint erfüllen, explizit ‚in kurzer Art und Weise‘ hinschreiben kann“
  - In Literatur formale Definitionen, z. B. in Ng et al. (1998): "Exploratory mining and pruning optimizations of constrained associations rules.” (Definition 2)
    - Itemset ist succint set, wenn für alle Items darin festgestellt werden kann (mittels einer Abfrage in der Datenbank), ob sie Constraint erfüllen
    - Constraint ist succint, wenn alle gültigen (sie erfüllenden) Itemsets aus den Potenzmengen einer endlichen Zahl von succinct sets zusammengesetzt werden können, und zwar allein mit den Mengenoperationen „Vereinigung“ und „Differenz“
  - Vereinfacht: Wenn wir für jedes Item wissen, ob es Constraint erfüllt, dann können wir davon alle gültigen Itemsets ableiten (ohne noch einmal in Datenbank zu gehen und exakte Attributwerte / Transaktionen zu betrachten)

Theorieaufgaben – Succintness (2)

- Sei S eine Menge an Attributwerten in einem Itemset, z. B. die Preise der Items, und sei v ein fester (aber beliebiger) Wert
- Beispiele für succint: $\min(S) \leq v$, $\min(S) \geq v$, $\min(S) = v$
  - Wenn ich für zwei Items / Itemsets weiß, ob Constraint erfüllt, dann weiß ich es auch für Vereinigung, ohne mir die Attributwerte noch einmal anzuschauen
  - Wenn in einem Itemset $\min(S_1) \leq 4$ (Constraint erfüllt) und in anderem $\min(S_2) \leq 4$ (Constraint erfüllt), dann $\min(S_1 \cup S_2) \leq 4$ (Constraint erfüllt)
  - Wenn in einem Itemset $\min(S_1) \leq 4$ (Constraint erfüllt) und in anderem $\min(S_2) \not\leq 4$ (Constraint verletzt), dann $\min(S_1 \cup S_2) \leq 4$ (Constraint erfüllt)
  - Wenn in einem Itemset $\min(S_1) \not\leq 4$ (Constraint verletzt) und in anderem $\min(S_2) \not\leq 4$ (Constraint verletzt), dann $\min(S_1 \cup S_2) \not\leq 4$ (Constraint verletzt)
  - In allen Fällen kann also hergeleitet werden, ob Constraint erfüllt ist

Theorieaufgaben – Succintness (3)

- Gegenbeispiel für succinct: $\text{sum}(S) \leq v$, $\text{sum}(S) \geq v$, $\text{sum}(S) = v$
  - Wenn in einem Itemset $\text{sum}(S_1) \leq 4$ (Constraint erfüllt) und in anderem $\text{sum}(S_2) \leq 4$ (Constraint erfüllt), dann weiß ich nicht, ob Constraint für Vereinigung erfüllt (hängt von konkreten Attributwerten ab)
  - Wenn dagegen in einem der beiden Itemsets Constraint verletzt, dann auch immer in Vereinigung
  - Insgesamt also nicht immer (ohne Rückgriff auf Attributwerte) Aussage möglich, ob Vereinigung der Itemsets die Constraint erfüllt → nicht succint
- Weitere Beispiele im Paper von Ng et al. (1998) sowie im Buch „Data Mining" von Han et al. (2012)

Theorieaufgaben – Support-/Constraint-Pruning

- Grenzen Sie Support-basiertes von Constraint-basiertem Pruning ab anhand eines Beispiels!
  - Support-basiert: Entferne Kandidat der Größe k, wenn (k-1)-elementige Teilmenge, die Constraints erfüllt, nicht frequent ist
  - Constraint-basiert: Entferne Kandidat, wenn Constraints nicht erfüllt
  - Beide Arten des Prunings können in Konflikt treten (wenn z. B. durch Constraints weniger Support-basiertes Pruning möglich)
  - Beispiel für Pruning: Filzstift im Supermarkt
    - Item „Filzstift“ vermutlich nur in wenigen Transaktionen, insofern geringer Support
    - Constraint „Type=Non-Food“ ist von {Filzstift} erfüllt, aber vermutlich auch von anderen Item(set)s (mit womöglich größeren Support; insofern unklar, wie stark Pruning durch die Constraint allein ist)
    - Support-basiertes Pruning unter Berücksichtigung der Constraint: untersuche nur Itemsets bestehend aus Non-Food-Items und prune, wenn Item „Filzstift“ enthalten