Extremely Fast Decision Tree

C. Manapragada, G.I. Webb, M. Salehi

24.02.2018, Monash University, Victoria, Australia
Published at KDD'18, August 2018, London, United Kingdom

Problemstellung

Datenbanken wachsen um mehrere Millionen Einträge täglich. Data-Mining[1] wirft bei solchen Umfängen neue Herausforderungen auf:

- Wenn Daten schneller ankommen als sie verarbeitet werden können, wächst die Zahl der ungenutzten Daten unbeschränkt.
- Die Daten einfach zwischenzuspeichern und später zu nutzen wirft eigene Probleme des Verlustes und der Korruption auf.

[1] Data-Mining: Eingedeutschter Begriff nach Definition des Dudens

Das modernste und standardmäßig verwendete Verfahren ist der Hoeffding Tree[2].

Der Aspekt der Fehlerrate kann durch geringfügig erhöhten Berechnungsaufwand (bei den meisten der größten Klassifizierungs-Datensätzen des UCI Repositoriums) signifikant verbessert werden.

[2] P. Domingos, G. Hulten. 2000. *Mining High-Speed Data Streams.* KDD 2000, Boston, MA USA.

Relevanz

Es gibt viele denkbare Szenarien, in denen eine niedrigere Fehlerrate wünschenswerter ist als etwas schnellere Ergebnisse:

- Banken (z.B. für Transaktionen)
- Medizin (z.B. für Diagnosen)
- Börse (z.B. für Aktienkurse)
- ...

Hoeffding Tree → Very Fast Decision Tree

Der Hoeffding Tree konstruiert schrittweise einen Baum, wobei der Hoeffding Bound verwendet wird, um anhand des Informationszuwachses herauszufinden, welcher potentielle Split die bessere Wahl ist.

Der Split wird so lange herausgezögert, bis sicher ist, dass dies die beste Option ist.
Diese Entscheidung wird jedoch nie wieder überdacht und ist final.

Hoeffding Adaptive Tree

Diese Methode baut einen Baum auf, der wiederum alternative Teilbäume aufbaut, falls einer der Teilbäume für neuere Beispiele schlechtere Voraussagen trifft.

Eine dieser Alternativen wird eingewechselt, wenn sie eine bessere Genauigkeit als das Original hat. HAT basiert also auf den Ergebnissen der Vorhersage.

Concept-Adapting Very Fast Decision Tree

Der CVFDT hat ein sich bewegendes Fenster, welches bei den Knoten die Wertung der Beispiele, die aus dem Fenster herausfallen, vermindert.

Existierende Splits werden ersetzt, falls ihr Attribut nicht länger das gewinnende ist und ein Set von alternativen Teilbäume mit anderen Splits eine höhere Genauigkeit verbucht.

Der CVFDT ist dabei explizit für drifting scenarios[3] geeignet.

[3] drifting scenario: Ein Szenario, bei dem die eingegebenen Daten ihr Muster verändern

Hoeffding Anytime Desicion Tree: Ziele

- HATT konvergiert in seiner Wahrscheinlichkeit zum asymptotischen Batch Tree[4].
- EFDT hat bei den meisten Instanzen eine höhere Genauigkeit, egal ob sie geordnet sind oder eine zufällige Reihenfolge haben.

[4] Batch Desicion Tree: Baum, der durch batch learning generiert wird.

Hoeffding Anytime Tree

Der Hoeffding Anytime Tree ist ein schrittweise lernender Entscheidungsbaum-Algorithmus für stationary scenarios[5].

Anytime ist hier so zu verstehen, dass ein Ready-To-Use-Modell zu jeder Zeit nach dem Betrachten der ersten paar Beispiele bereit steht[2].

HATT zielt darauf ab, einen Split durchzuführen, sobald er nützlich erscheint, und überdenkt diese Entscheidung zu einem späteren Zeitpunkt noch einmal.

---

[5] stationary scenario: Ein Szenario, bei dem alle eingegebenen Daten das gleiche Schema haben

[2] P. Domingos, G. Hulten. 2000. *Mining High-Speed Data Streams.* KDD 2000, Boston, MA USA.

Hoeffding Anytime Tree:

Let HATT be a tree with a single leaf, the root
Let $X_1 = X \cup X_\varnothing$
Let $G_1(X_\varnothing)$ be the G obtained by predicting the most frequent class in S
for all classes y_k **do**
    for all values x_{ij} of each attribute $X_i \in X$ **do**
        Set counter $n_{ijk}(\text{root}) = 0$
for all examples $(\vec{x}, y)$ in S **do**
    Sort $(\vec{x}, y)$ into a leaf l using HATT
    for all nodes in path (root . . . l) **do**
        for all x_{ij} in $\vec{x}$ such that $X_i \in X_{\text{node}}$ **do**
            Increment $n_{ijk}(\text{node})$
            if node = l **then**
                AttemptToSplit(l)
            else
                ReEvaluateBestSplit(node)

Hoeffding Anytime Tree:

Let HATT be a tree with a single leaf, the root
Let $X_1 = X \cup X_\varnothing$
Let $G_1(X_\varnothing)$ be the G obtained by predicting the most frequent class in S
for all classes y_k **do**
    for all values x_{ij} of each attribute $X_i \in X$ **do**
        Set counter $n_{ijk}(\text{root}) = 0$
for all examples $(\vec{x}, y)$ in S **do**
    Sort $(\vec{x}, y)$ into a leaf l using HATT
    for all nodes in path (root ... l) **do**
        for all x_{ij} in $\vec{x}$ such that $X_i \in X_{\text{node}}$ **do**
            Increment $n_{ijk}(\text{node})$
            if node = l **then**
                AttemptToSplit(l)
            else
                ReEvaluateBestSplit(node)

AttemptToSplit (leafNode l):

Label l with the majority class at l
if all examples at l are not of the same class **then**
  Compute $\vec{G}_1(X_i)$ for each attribute $X_l - \{X_\varnothing\}$ using the counts $n_{ijk}(l)$
  Let X_a be the attribute with the highest $\vec{G}_1$
  Let $X_b = X_\varnothing$
  Compute ε using equation 1
  if $\vec{G}_l(X_a) - \vec{G}_l(X_b) > \varepsilon$ and $X_a \neq X_\varnothing$ **then**
    Replace l by an internal node that splits on X_a
    for all branches of the split **do**
      Add a new leaf l_m and let $X_m = X - X_a$
      Let $\vec{G}_m(X_\varnothing)$ be the G obtained by predicting the most frequent class at l_m
      for all classes y_k and all values x_{ij} of each attribute $X_i \in X_m - \{X_\varnothing\}$ **do**
        Let $n_{ijk}(l_m) = 0$

ReEvaluateBestSplit (internalNode int):

Compute $\vec{G}_{\text{int}}(X_i)$ for each attribute $X_{\text{int}} - \{X_\varnothing\}$ using the counts $n_{ijk}(\text{int})$
Let X_a be the attribute with the highest $\vec{G}_{\text{int}}$
Let X_{current} be the *current* split attribute
Compute ε using equation 1
if $\vec{G}_l(X_a) - \vec{G}_l(X_{\text{current}}) > \varepsilon$ **then**
    if $X_a = X_\varnothing$ **then**
        Replace internal node int with a leaf (kills subtree)
    if $X_a \neq X_{\text{current}}$ **then**
        Replace int with an internal node that splits on X_a
        for all branches of the split **do**
            Add a new leaf l_m and let $X_m = X - X_a$
            Let $\vec{G}_m(X_\varnothing)$ be the G obtained predicting the most frequent class at l_m
            for all classes y_k and all values x_{ij} of each attribute $X_i \in X_m - \{X_\varnothing\}$ **do**
                Let $n_{ijk}(l_m) = 0$

Gütevergleich: Speicher

- Bei Daten mit d Attributen, v Werten pro Attribut und c Klassen benötigen sowohl HT als auch HATT $\mathcal{O}(dvc)$ Speicher, um für jeden Knoten die Statistiken zu speichern.
- Da die Worst Case-Komplexität für HT in Abhängigkeit der aktuellen Anzahl an Blättern l als $\mathcal{O}(ldvc)$ gegeben ist, kann die Komplexität für HATT als $\mathcal{O}(ndvc)$ mit n als Knotenanzahl geschrieben werden.
- Durch $l \in \mathcal{O}(n)$ ist die Speicherkomplexität für HT und HATT äquivalent.

Gütevergleich: Zeit

- Für jedes Blatt eines HTs und jeden Knoten eines HATTs müssen nicht mehr als d Attribute betrachtet werden.
- Jede Untersuchung eines Attributs an einem Knoten benötigt die Berechnung von v Informationszuwächsen.
- Für jeden Informationszuwachs benötigt die Berechnung $\mathcal{O}(c)$, so benötigt jede Split-Reevaluation $\mathcal{O}(dvc)$ an jedem Knoten.
- Die Worst Case-Laufzeit für jede Split-Evaluation beläuft sich bei jedem Zeitschritt auf $\mathcal{O}(dvc)$ für HT und auf $\mathcal{O}(hdvc)$ für HATT, da entweder ein Blatt oder ein Pfad untersucht werden muss.

Daten

Es wurden 13 Vergleiche zwischen dem EFDT und dem VFDT gezogen, in denen alle UCI[21] Datensets mit über 200.000 Instanzen verwendet wurden, die keine fehlenden Werte behinhalten und kein text mining[6] benötigen.

Die zufällig angeordneten Daten wurden 10 Mal mit der Unix *shuf* Utility und der Hilfe von reproduzierbaren Zufallsbytes[7] gemischt, wobei diese 10 Ergebnisse anschließend gemittelt wurden.

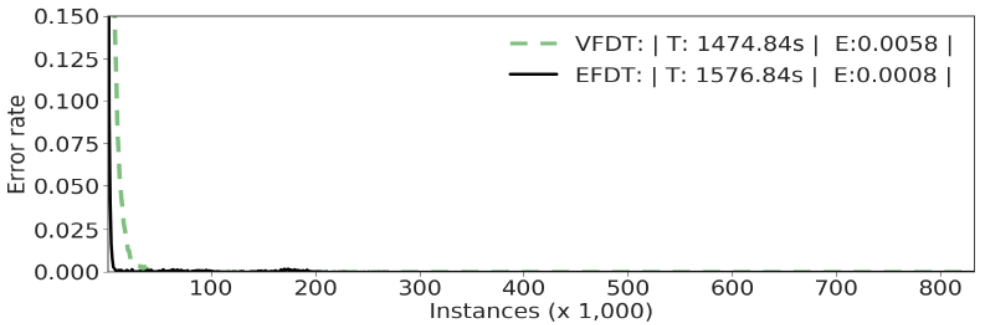
---

[21] Moshe Lichmann. 2013. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml

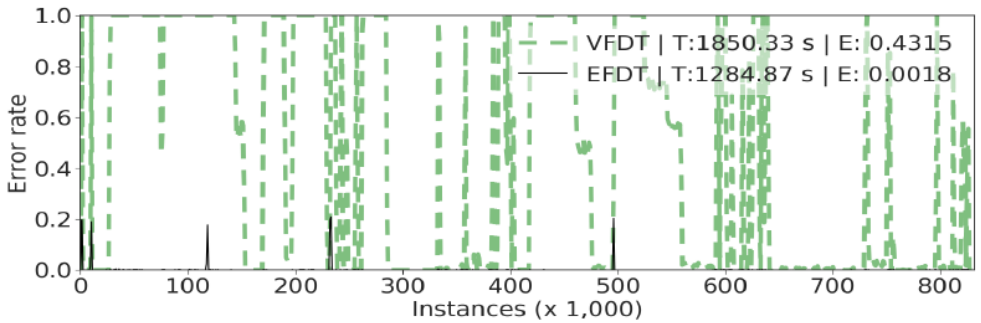
[6] Bündel von Analyse-Algorithmen zur Entdeckung von Bedeutungsstrukturen aus Textdaten.

[7] GNU Coreutils: Random sources. 2018. Retrieved Jan 05, 2018 from www.gnu.org

Gütevergleich: VFDT



(a) 10 stream shuffled average.



(b) Unshuffled.

Figure 5.3: Fonts dataset [21]

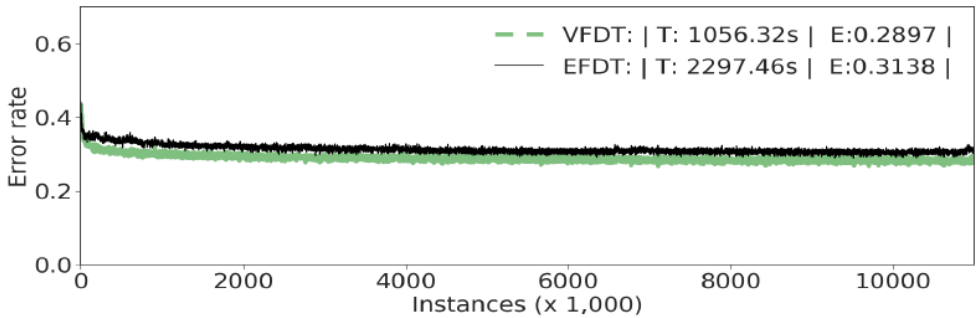
Bei sortierten Instanzen (z.B. hier alphabetisch nach Schriftart):

Zu jedem Beginn eines neuen Schriftsets muss der VFDT ein neues Konzept lernen

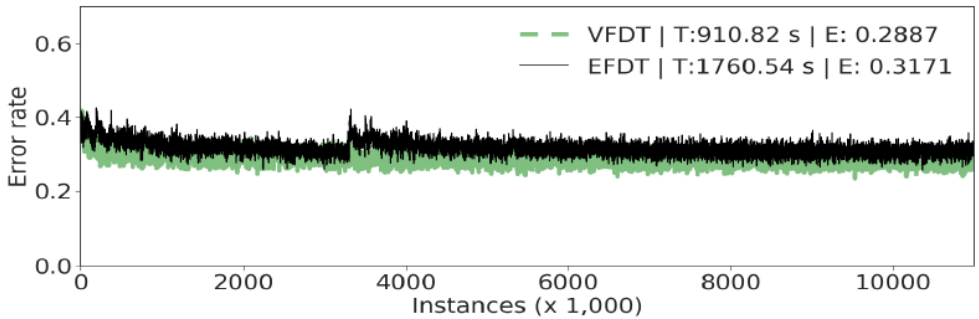
Der EFDT kann sein Modell anpassen und effektiv veraltete Splits überarbeiten.

[21] Moshe Lichmann. 2013. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml

Gütevergleich: VFDT



(a) 10 stream shuffled average.



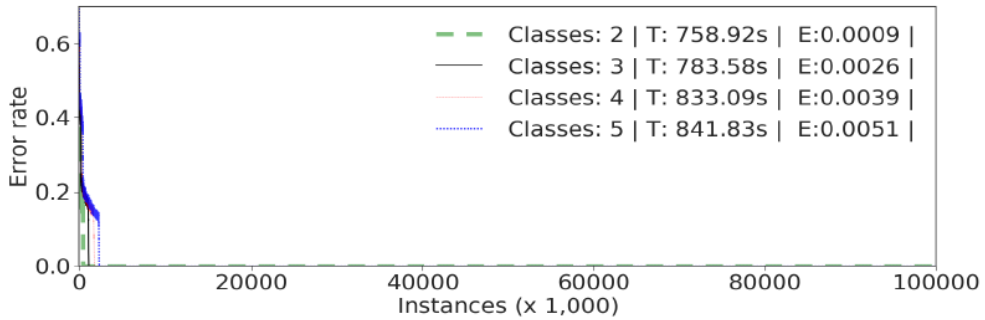
(b) Unshuffled.

Figure 5.10: Higgs dataset [1, 21]

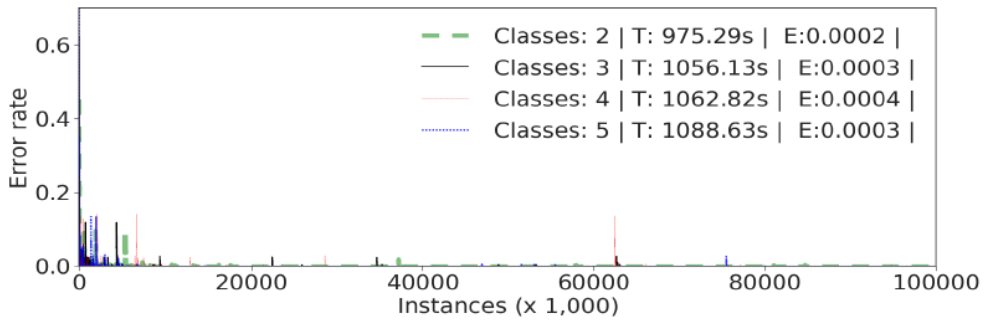
Falls der Informationszuwachs zwischen sowohl den Top-Attributen als auch der Zuwachs selbst niedrig sind, wählt HATT möglicherweise einen Split, welcher viele neue Beispiele zum Korrigieren benötigt.
Synthetische Daten aus Physik-Simulationen (Higgs, Hepmass, SUSY) führten z.B. dazu.

[1] Pierre Baldi. 2014. Searching for exotic particles in high-energy physics with deep learning.
[21] Moshe Lichmann. 2013. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml

Gütevergleich: VFDT



(a) VFDT



(b) EFDT

Figure 5.13: A longer term view of the experiments from Fig. 1.1 shows us that even 100 million examples in, EFDT maintains a commanding lead on prequential accuracy.

Über verschiedene Komplexitätsklassen mit jeweils 100 Millionen Beispielen sieht man gut, wie der EFDT eine deutlich niedrigere Fehlerrate gegenüber dem VFDT hält.

Konklusion

- In den 10/13 Testfällen, die nicht darauf ausgelegt sind das Worst Case-Szenario des HATTs zu testen, schneidet HATT im Aspekt der Fehlerrate immer mindestens genau so gut ab wie der VFDT, eher besser.
- Obwohl HATT nicht darauf abgezielt hat, scheint er eine Toleranz gegen über drifting scenarios zu haben.
- Falls ein Baum nicht zuende gelernt hat, bringt HATT gegenüber HT deutliche Vorteile.