Kódverifikáció gépi tanulással

Szoftver verifikáció és validáció kiselőadás

Hidasi Balázs
2013. 12. 12.

Áttekintés

- Gépi tanuló módszerek áttekintése
- Kódverifikáció
- Motiváció
- Néhány megközelítés
- Fault Invariant Classifier módszer ismertetése

Gépi tanuló módszerek

Tanuló módszerek

Felügyelt tanulás

Megerősítéses tanulás

Felügyelet nélküli tanulás

Félig felügyelt tanulás

Regresszió

Osztályozás

Klaszterezés

Dimenzió-csökkentés

Jellemző kinyerés

…

Módszerek:
- Lineáris regresszió
- Logisztikus regresszió
- Döntési fák
- Szupport vektor gépek
- Neurális hálók
- Ensemble módszerek
- …

Módszerek:
- Hierarchikus klaszterezők
- Partícionáló klaszterezők (pl. kNN)
- Sűrűség alapú klaszterezők (pl. DBSCAN)
- …

Kódverifikáció

- Rejtett implementációs hibák felfedezése
- Szintaktikailag helyes kód ≠ hibamentes
- Többféle hibatípus
  - Rejtett (látens) hiba
    - Nem minden esetben fut hibára a program, de időről időre hiba történik a lefutásokban
    - Pl. kicímzések tömbökből; párhuzamosított program hibás konkurencia kezelése
  - Szemantikai hiba
    - Nem azt csinálja a program, amit szeretnénk (nem a feladatot oldja meg)
- Vizsgált módszerek: látens hibák felderítése

Motiváció (1/2)

- Gépi tanulás előnyei
  - Általános modellek
    - Nem kell külön teszteseteket generálni
  - Taníthatóság
    - Kód fejlődésével a tesztek is automatikusan fejlődnek
- Követelmény: tanítókészlet
  - Alkalmazás feltétele
  - Pl. korábbi projektek kódjai; aktuális projekt hibás és javított verziói; közösség által összegyűjtött kódhalmaz

Motiváció (2/2)

- Nem tökéletes
  - Tökéletes eredményt adó tanuló módszer → hibás
  - Hamis pozitív és hamis negatív hibák
  - Eredmény átnézése szükséges
    - De az átnézendő esetek száma jelentősen csökken
  - Eljárás = szűrés
- Nagy mennyiségű adat (gyors/olcsó) kezelése
  - Költséges eljárások megspórolhatóak
  - Futtatásuk elegendő a tanuló módszer által gyanúsnak ítélt részeken

Néhány megközelítés (1/4)

- Mit akarunk jelezni?
  - Rejtett hiba van a forráskódban
  - Ez egy rejtett hiba a forráskódban
- Hogyan reprezentáljuk a kódot?
- Milyen módszert használunk?
  - Felügyelt tanulás
    - Hibás nem hibás példák előállítása
    - Hogyan generáljuk le a tanítóhalmazt?
  - Felügyelet nélküli tanítás
    - Hogyan értelmezzük az eredményt?

Néhány megközelítés (2/4)

- 1. megközelítés
  - Statikus jellemzők definiálása
    - Pl.: kódduplikáció, nem használt kód, stb.
  - Valószínűségi modell építése
    - Jellemzők jelenléte – hibás/nem hibás viszonyra
  - File szintű vizsgálat
    - Hibás-e a file?

Néhány megközelítés (3/4)

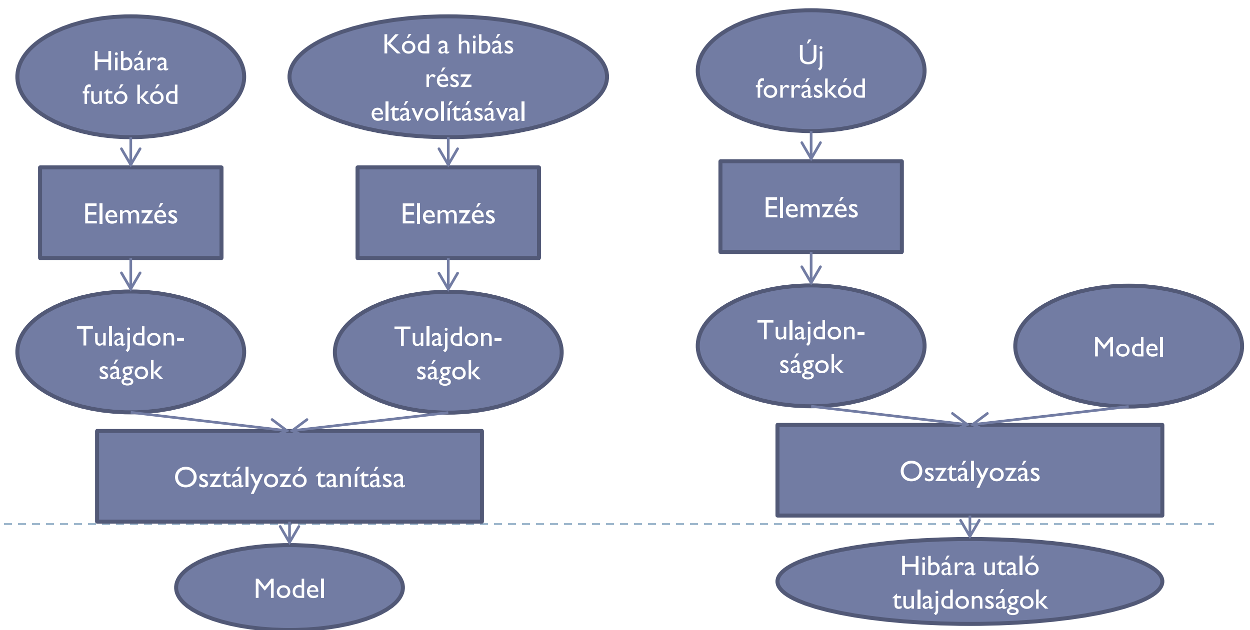
- 2. megközelítés
  - Kódrészletek többszöri futtatása
    - Feltevés: futtatás olcsó, verifikáció költséges
  - Futtatások klaszterezése
  - Feltevés: hibás futtatások kisebb klasztereket képeznek
    - Kevesebb hibás futtatás, mint nem hibás
    - Hibák okai eltérőek: más klaszterekbe kerülnek
  - Mintavételezés
    - Minden klaszterből egy eset választása verifikálásra
      - Kisebb klaszterekből és így hibás esetekből több kerül kiválasztásra
    - Ha a verifikáció során hibát találunk, a klaszter más elemeit is ellenőrizzük

Néhány megközelítés (4/4)

- 3. megközelítés
  - Hibás és nem hibás forráskódok vagy futtatások osztályozása
    - Címke: hibás volt-e?
  - Az egyes entitások jellemzése fontos kérdés

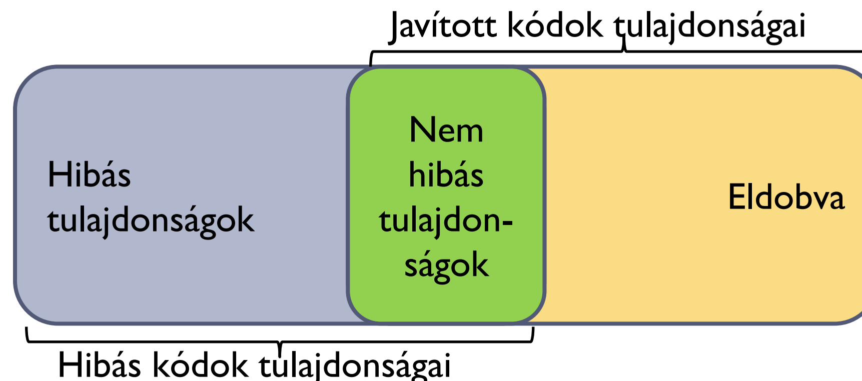
Fault Invariant Classifier (1/6)

- Áttekintés
  - Hibára utaló tulajdonságok felfedezése forráskódban
    - Konkrét hibára utaló kódrészletek megtalálása
  - Automatikus tulajdonság generálás
  - Felügyelt tanulás a gyanús és nem gyanús tulajdonságok megkülönböztetésére
    - Osztályozás



Fault Invariant Classifier (2/6)

- Tanítóhalmaz generálás
  - Eltérés ismerten hibás és nem hibás kódváltozatok között
  - Egy kód két verziója
    - P: Legalább egy hiba van benne
    - P': Legalább egy hiba ki lett javítva P-hez képest
      - Nem szükséges, hogy hibátlan legyen
  - Csak hibás kódokban lévő tulajdonságok → hibás címke
  - Hibás és javított kódban is meglévő tulajdonságok → nem hibás címke
  - Csak javított kódban lévő tulajdonságok → eldobva

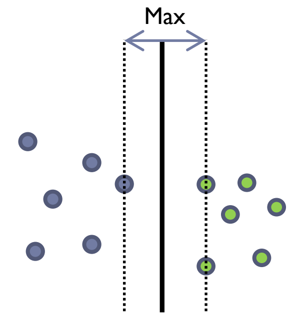


Fault Invariant Classifier (3/6)

- Tulajdonság generálás
  - Szemantikus jellemzők
  - Dinamikus invariáns detektálás
    - Futási idejű tulajdonságok
    - Futtatások során érvényes logikai kifejezések a változókra
  - Példák:
    - Skalárok milyen értéktartományba esnek bizonyos programpontokon
    - Skalárok közötti relációk
    - Listák legkisebb és legnagyobb eleme
    - Listák viszonya egymással
    - Implikáció
    - Stb.

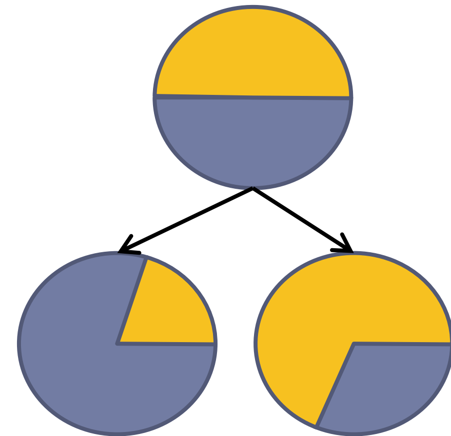
Fault Invariant Classifier (4/6)

- Tulajdonságok → karakterisztikus vektor
  - Tanuló eljárások bemenete (általában) egy fix hosszúságú vektor
  - Konvertálás szükséges
  - Ezen a vektoron tanítunk
    - Ne legyen program specifikus
    - Pl. x=y helyett (értékadás, kétváltozós, skalár)
- Tanuló eljárások
  - Szupport vektor gépek (SVM)
    - Bináris osztályozó
    - Osztályok szétválasztása egy hipersíkkal úgy, hogy a távolsága hozzá legközelebb eső, vele párhuzamos, az osztályok egy pontját tartalmazó hipersíkkal maximális
      - Maximal margin separation
      - Lineáris szeparálás → kernelek alkalmazásával nemlineárissá tehető
      - A legközelebbi párhuzamos hipersíkokra eső tanítópontok a szupport vektorok; a modell velük kifejezhető (többi pont eldobható)



Fault Invariant Classifier (5/6)

- Tanuló eljárások (folyt.)
  - Döntési fák
    - Partícionáló eljárások
    - Egy gyökér csomópontból indulunk, ami minden tanítópontot tartalmaz
    - A csomópontban a legjobban vágó feltétel választása
      - Feltétel: attribútum + logikai kifejezés
        - Pl.: Kor > 20, Nem=Nő, stb.
      - Vágás jó
        - Osztályok eloszlása a vágás után eltér az eredetitől
        - Csomópontok mérete közel egyenlő (ne 1-1 elemet vágjunk le)
        - Célfüggvény: pl. információ nyereség, Gini-index, stb.
    - Tanítópontok szétdobása a feltétel alapján a gyermek csomópontokba
    - Ha a leállási feltétel nem teljesül, akkor ki nem fejtett csomópontok kifejtése a fentiek szerint
    - Könnyen értelmezhető modellt állít elő



Fault Invariant Classifier (6/6)

- Kiértékelés
  - Leave-one-out teszt
    - Egy program kivételével az összesen tanítanak, az egyen tesztelnek
    - Minden programon tesztelnek egyszer
  - Hibásnak ítélt- és a teljes halmazban a hibásak arányának arányát nézik
  - Nem az összes hibát okozó tulajdonság megkeresése a cél
    - Rangsorolás: az elől lévők nagy valószínűséggel legyenek hibát okozóak
  - Jelentős javulás rangsorolás esetén a korábbi módszerekhez képest (~50-szeres a hibás tulajdonságok aránya)
    - Mérsékelt javulás címkézés esetén (~5-szörös)
  - (Hiba a cikkben: döntési fa nem használható sorrendezésre → de használható)
  - Használhatósági teszt: egy hiba okának megtalálásához átlagosan 3 gyanús tulajdonságot kell átnézni

Források

- Brun, Yuriy, and Michael D. Ernst. "Finding latent code errors via machine learning over program executions." *Proceedings of the 26th International Conference on Software Engineering*. IEEE Computer Society, 2004.
- Dickinson, William, David Leon, and Andy Podgurski. "Finding failures by cluster analysis of execution profiles." *Proceedings of the 23rd international conference on Software engineering*. IEEE Computer Society, 2001.
- Xie, Yichen, and Dawson Engler. "Using redundancies to find errors." *ACM SIGSOFT Software Engineering Notes* 27.6 (2002): 51-60.
- Hangal, Sudheendra, and Monica S. Lam. "Tracking down software bugs using automatic anomaly detection." *Proceedings of the 24th international conference on Software engineering*. ACM, 2002.