Data Mining Tutorial

Frequent Itemset Mining

Erich Schubert, Dr. Arthur Zimek

Ludwig-Maximilians-Universität München

2013-07-12 — KDD Übung

NetFlix

NetFlix:

- Online-Videothek
- Nutzer können Filme bewerten
- Film-Empfehlung basierend auf Bewertungen?

NetFlix

NetFlix:

- Online-Videothek
- Nutzer können Filme bewerten
- Film-Empfehlung basierend auf Bewertungen?

Netflix Prize

- 1 Million USD für den ersten, der die Qualität der Empfehlungen um 10% verbessert.
- 100 Millionen Bewertungen von 480 Tausend Nutzern für 17770 Filme.
- Skala: 1-5 Sterne
- Begonnen: 2006

NetFlix

NetFlix:

- Online-Videothek
- Nutzer können Filme bewerten
- Film-Empfehlung basierend auf Bewertungen?

Netflix Prize

- 1 Million USD für den ersten, der die Qualität der Empfehlungen um 10% verbessert.
- 100 Millionen Bewertungen von 480 Tausend Nutzern für 17770 Filme.
- Skala: 1-5 Sterne
- Begonnen: 2006 – Gewonnen: 2009.

NetFlix

NetFlix:

- Online-Videothek
- Nutzer können Filme bewerten
- Film-Empfehlung basierend auf Bewertungen?

Netflix Prize

- 1 Million USD für den ersten, der die Qualität der Empfehlungen um 10% verbessert.
- 100 Millionen Bewertungen von 480 Tausend Nutzern für 17770 Filme.
- Skala: 1-5 Sterne
- Begonnen: 2006 – Gewonnen: 2009.
- Lösung nie “in Produktion” eingesetzt: zu viel Aufwand, das real einzusetzen

NetFlix und Frequent Itemsets

Assoziationsregeln aus NetFlix-Daten:

- Sehr naiver Ansatz, ignoriert die echte Bewertung!

NetFlix und Frequent Itemsets

Assoziationsregeln aus NetFlix-Daten:

- Sehr naiver Ansatz, ignoriert die echte Bewertung!
- Aber gute Übung: $\approx$ 2 GB Daten "bestrafen" Fehler

NetFlix und Frequent Itemsets

Assoziationsregeln aus NetFlix-Daten:

- Sehr naiver Ansatz, ignoriert die echte Bewertung!
- Aber gute Übung: $\approx$ 2 GB Daten "bestrafen" Fehler
- Vereinfachung: nur 5 Sterne ("Favoriten")
  Ca. 23 Mio Items in 480189 Transaktionen

NetFlix und Frequent Itemsets

Assoziationsregeln aus NetFlix-Daten:

- Sehr naiver Ansatz, ignoriert die echte Bewertung!
- Aber gute Übung: $\approx$ 2 GB Daten "bestrafen" Fehler
- Vereinfachung: nur 5 Sterne ("Favoriten")
  Ca. 23 Mio Items in 480189 Transaktionen
- Erwartet: "Erkenntnisse" wie:
  Wer Star Wars I und II angeschaut hat,
  schaut auch Star Wars III an.

Planning Apriori

Weka skaliert leider sehr schlecht auf große Datenmengen.
Des weiteren müssten wir sie zuerst in das `.arff` Format konvertieren, das von Weka verwendet wird.

Warum nicht Apriori mal kurz selbst implementieren?

Parameter (minsupp) unklar – können wir sie schätzen?

23 millionen integer IDs – passt in den Hauptspeicher.

Vorverarbeitung der Daten

Preprocessing: 1 Datei pro *Film* $\rightarrow$ 1 Gesamtdatei
Nur Filme mit mindestens 10000 5-Sterne Wertungen
(Ohne UserID, Bewertungen, 1-4 Sterne, Datum!)

```
import gzip, glob, collections

favorites = collections.defaultdict(lambda: set())
for filename in glob.glob("training_set/mv_*.txt.gz"):
    instream = gzip.open(filename)
    filmid = instream.readline().split(":")[0]
    users = set()
    for line in instream:
        userid, rating, date = line.strip().split(",")
        if int(rating) >= 5: users.add(userid)
    instream.close()
    if len(users) >= 1000:
        for userid in users:
            favorites[userid].add(filmid)

for userid, favs in favorites.iteritems():
    print " ".join(favs)
```

Vorverarbeitung der Daten

Ergebnis Vorverarbeitung:

```
Laufzeit: ca. 10 Minuten
 22.258.632 Bytes komprimiert
 71.265.709 Bytes dekomprimiert
    450.342 Zeilen ("Transaktionen")
 13.129.796 Wörter ("Items")
```

(statt 688 MB komprimiert, 2 GB dekomprimiert!)

(Wieder-) Einlesen der Daten

Einlesen der kondensierten Gesamtdatei.

```
#!/usr/bin/python
import gzip

names=dict()
for line in open("movie_titles.txt"):
    filmid, year, title = line.strip().split(",", 2)
    names[int(filmid)] = title

db=[]
for line in gzip.open("preprocessed-apriori.txt.gz"):
    db.append(set(map(int, line.split())))

print "Database size:", len(db)
```

Output:

```
Database size: 450342
```

Die Klasse Itemset

Klasse für ein Itemset:

```
class itemset(tuple):
  def __new__(typ, items, support=0):
    self = tuple.__new__(typ, items)
    self.support = support
    return self

  def __str__(self):
    return "; ".join(map(names.get, self)) +
      ": "+str(self.support)
```

Die Klasse Itemsets

Klasse für die Menge der Frequent Itemsets:

```
class itemsetsClass():
  def __init__(self, oneitems, size):
    alli = itemset([], support=size)
    self.itemsets = [dict({alli:alli}), {i:i for i in oneitems}]

  def __getitem__(self, item):
    if isinstance(item, int): return self.itemsets[item].keys()
    return self.itemsets[len(item)][item]

  def __contains__(self, item):
    return item in self.itemsets[len(item)]

  def get(self, item, default=None):
    if isinstance(item, int): return self.itemsets[item].keys()
    return self.itemsets[len(item)].get(item, default)

  def add(self, item):
    while len(item) >= len(self.itemsets):
      self.itemsets.append(dict())
    self.itemsets[len(item)][item] = item

  def maxsize(self):
    return len(self.itemsets) - 1
```

1-Itemsets

```
oneitems = dict()
for rec in db:
  for tag in rec:
    item = itemset([tag])
    item = oneitems.setdefault(item, item)
    item.support += 1
oneitems = list(oneitems.keys())

# Inspect:
oneitems.sort(lambda a,b: cmp(b.support, a.support))
print len(oneitems), map(str, oneitems[:10])
print str(oneitems[100]), str(oneitems[200])
```

```
565 ['Lord of the Rings: The Two Towers: 96535',
'Forrest Gump: 95532',
'The Shawshank Redemption: Special Edition: 95150',
'Lord of the Rings: The Fellowship of the Ring: 94655',
'The Green Mile: 92863',
'Lord of the Rings: The Return of the King: 92805',
'Pirates of the Caribbean: The Curse of the Black Pearl: 82549',
'Finding Nemo (Widescreen): 79447', 'The Sixth Sense: 74790',
'Indiana Jones and the Last Crusade: 74553']
John Q: 32859 Blade Runner: 22964
```

Computing the 1-Itemsets

Probieren wir also mal minsupport$= 10000$:

```
minsupport = 10000
oneitems = filter(
  lambda x: x.support >= minsupport,
  oneitems)
itemsets = itemsetsClass(oneitems, len(db))

print len(oneitems)
```

```
565
Candidates of size 2: 159330
```

Kandidatenset der Größe 2: $\binom{565}{2} = 159330$ – und noch kein pruning möglich (erst ab Größe 3, interessant ab Größe 4!)

Apriori-Gen

Kandidaten erzeugen mit Apriori-Gen:

```
def apriorigen(curitems):
  curitems = sorted(curitems)
  for i in range(0, len(curitems) - 1):
    itema = curitems[i]
    for j in range(i + 1, len(curitems)):
      itemb = curitems[j]
      # Prefix test:
      if not itema[:-1] == itemb[:-1]: break
      cand = itema + itemb[-1:] # Extend with last
      # Pruning test:
      ok = True
      for i in range(len(cand) - 2):
        t = cand[:i] + cand[i+1:] # without i
        if not itemset(t) in curitems:
          ok = False
          break
      if ok: yield itemset(cand) # generate itemset
```

Apriori FIM

Hauptschleife für Apriori Frequent Itemsets:

```
while True:
  size = itemsets.maxsize() + 1
  cand = dict()
  for c in apriorigen(itemsets[size - 1]): cand[c] = c
  if len(cand) == 0: break
  print "Candidates of size %d: %d" % (size, len(cand))
  for rec in db:
    for subset in itertools.combinations(rec, size):
      subset = cand.get(itemset(subset))
      if subset: subset.support += 1
  for iset in cand:
    if iset.support >= minsupport: itemsets.add(iset)
  print map(str, sorted(itemsets.itemsets[-1],
    lambda a,b: cmp(b.support, a.support))[:20])
```

Output:

```
['Lord of the Rings: The Two Towers+Lord of the Rings: The
Return of the King: 60637', 'Lord of the Rings: The
Fellowship of the Ring+Lord of the Rings: The Return of the
King: 50940', 'Forrest Gump+The Green Mile: 42350', 'Forrest
Gump+The Shawshank Redemption: Special Edition: 41778', ...
```

Frequent 3-itemsets

Output:

```
Candidates of size 3: 61892
'Lord of the Rings: The Two Towers
+Lord of the Rings: The Return of the King
+Lord of the Rings: The Return of the King: Extended Edition: 24525',
'Forrest Gump
+The Shawshank Redemption: Special Edition
+The Green Mile: 23181',
'Lord of the Rings: The Two Towers: Extended Edition
+Lord of the Rings: The Return of the King
+Lord of the Rings: The Return of the King: Extended Edition: 22148',
'Lord of the Rings: The Two Towers
+Lord of the Rings: The Return of the King
+The Green Mile: 20763']
```

Könnte spannender sein... Verbesserungsmöglichkeit im preprocessing: "extended editions" mit den normalen editionen zusammenlegen!

Assoziationsregeln:

Assoziationsregeln erzeugen (mit "Lift" als Maß):

```
minconfidence, minterest = .5, 2.
def findRules(iset, body, head):
  for i in range(len(body)):
    if len(head) > 0 and head[0] <= body[i]: break
    nhead = itemsets[(body[i],) + head]
    nbody = itemsets[body[:i] + body[i+1:]]
    confidence = iset.support / float(nbody.support)
    if confidence < minconfidence: continue
    interest = confidence * (len(db) / float(nhead.support))
    if interest > minterest:
      print interest, confidence, iset.support,
      print nbody, "->", nhead
    findRules(iset, nbody, nhead)

for lenk in range(itemsets.maxsize(), 0, -1):
  for iset in itemsets[lenk]:
    findRules(iset, iset, ())
```

Interessanteste Assoziationsregeln:

(Nach Sortierung) interessante Assoziationsregeln:

```
34.5886566462 0.885565217391 5092
Friends: Season 4; The Best of Friends: Vol. 1: 5750
-> The Best of Friends: Season 2: 11530
... friends, friends, friends, friends, ...
18.8493812263 0.520308250227 5739
Chappelle's Show: Season 2: 11030
-> Chappelle's Show: Season 1: 12431
17.5474057799 0.832596041909 7152
Sex and the City: Season 5; Sex and the City: Season 1: 8590
-> Sex and the City: Season 4: 21368
17.1448021092 0.607112281857 9219
Family Guy: Vol. 2: Season 3: 15185
-> Family Guy: Vol. 1: Seasons 1-2: 15947
... sex and the city, family guy, the sopranos ...
16.2833046702 0.772616487455 5389
Lord of the Rings: The Fellowship of the Ring;
Sex and the City: Season 3: 6975
-> Sex and the City: Season 4: 21368
... mehr solche Kombinationen ...
```

Interessanteste Assoziationsregeln:

(Nach Sortierung) interessante Assoziationsregeln:

```
13.787836446 0.675366464069 5667
Lock, Stock and Two Smoking Barrels;
Pulp Fiction: 8391 -> Snatch: 22059
...
12.6205046533 0.618187315746 8178
Lock, Stock and Two Smoking Barrels: 13229
-> Snatch: 22059
... viel mit X-Men und X-Men United ...
9.37842707241 0.612612612613 5032
You've Got Mail; The Green Mile: 8214
-> Sleepless in Seattle: 29417
...
9.10929521819 0.595032525133 5031
Pretty Woman; You've Got Mail: 8455
-> Sleepless in Seattle: 29417
...
8.99123187537 0.853897695702 5225
The Lord of the Rings: The Fellowship of the Ring: Extended Edition;
Harry Potter and the Chamber of Secrets: 6119
-> Harry Potter and the Sorcerer's Stone: 42769
...
8.52899912383 0.512165694752 5094
The Godfather; Blazing Saddles: 9946
-> National Lampoon's Animal House: 27043
...
8.20439333138 0.621766879409 5553
Pulp Fiction; Trainspotting: Collector's Edition: 8931
-> Reservoir Dogs: 34129
```

Fazit

- Eine gute Übung, das mal selbst zu implementieren
- Laufzeit in Python: ≈ 6 Stunden!
- Ergebnisse: sinnvoll, aber i.d.R. nicht überraschend (Der "Durchschnittsnutzer" sollte aber auch nicht überraschend sein)
- Bessere Vorverarbeitung nötig: Staffeln / Seasons, Editions, Best Of, Serien, ...
- Korrelation von Blockbustern im selben Jahr!
- Data Mining *garantiert* leider nicht neue Erkenntnisse