８章　回帰：レコメンドの改良

CAO　RUI

推薦システムを改良する

- 提案：複数のアプローチを組合せで、パフォーマンスを上回させる

- 使ったデータ：映画レコメンド用のデータセット

  一つの軸がユーザID　もう一つの軸は映画ID

  要素：各ユーザの各映画に対してレーティング

  行列：行列は疎行列です

二値行列を用いたレコメンド

- ユーザ：点数を付けた映画を１、点数を付けていない映画を０とします→　映画のスコアを予測することができる

- 手順：
- 各ユーザごとに、他のユーザを似ている順にランク付けします。ユーザの「似ている度合い」を計測するために、二値行列から相関係数を計算する
- その特定の映画に点数をつけている類似ユーザが存在すれば、そのスコアを報告します。

二値行列を用いたレコメンド

```
corr_between_user1_and_user2 = np.corrcoef(user1, user2)[0,1]
```

```
import numpy as np
def all_correlations(bait, target):
    '''
    corrs = all_correlations(bait, target)
    corrs[i]はbaitとtarget[i]の相関係数です
    '''
    return np.array(
            [np.corrcoef(bait, c)[0,1] for c in target])
```

各ユーザに対して最近傍法のユーザを選び出すことができる

二値行列を用いたレコメンド

```
def estimate(user, rest):
    '''
    'rest'に基づいて、'user'の映画スコアを予測する
    '''
```

```
# userのレーティングを二値化する
bu = user > 0
# restのレーティングを二値化する
br = rest > 0
ws = all_correlations(bu,br)
# 相関係数の大きい順に100個インデックスを選ぶ
selected = ws.argsort()[-100:]
# 平均値を基に予測する
estimates = rest[selected].mean(0)
# ある映画は他の映画より多くのユーザから点数を付けられている
# それを考慮して、予測結果を修正する必要がある
estimates /= (.1+br[selected].mean(0))
```

RMSEを２０％減らすことができる

映画の点数を付ける回数が多ければ多いほど、予測の精度はよくなります

類似映画について考える

- ユーザUの映画Mに対するレーティングを予測する場合、映画Mに最も似ている映画と同じレーティングを結果とすることができる

手順：

→ 映画の類似行列を計算する

→ 類似行列を基に、ユーザと映画の各組み合わせについて予測する

類似映画について考える

```
for i in range(nmovies):
    movie_likeness[i] = all_correlations(reviews[:,i], reviews.T)
    movie_likeness[i,i] = -1
```

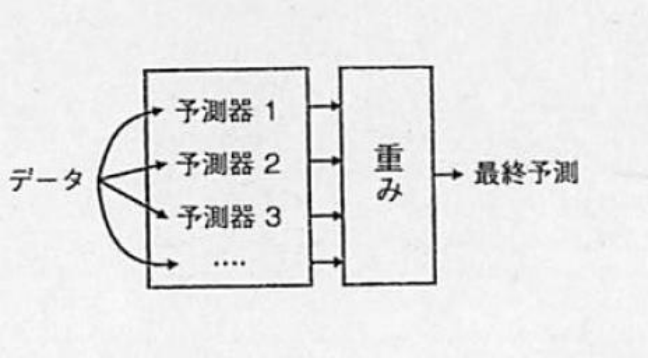
この行列に基づいて、映画のレーティングを予測する関数を定義する

```
def nn_movie(movie_likeness, reviews, uid, mid):
    # uidはユーザID、midは映画ID
    likes = movie_likeness[mid].argsort()
```

```
# 最も類似した映画が先頭に来るようにするため、配列の順番を反転させる
likes = likes[::-1]
# 類似映画から、uidのユーザが点数付けした映画のスコアを返す
for ell in likes:
    if reviews[uid,ell] > 0:
        return reviews[uid,ell]
```

RMSEはわずか0.85です。

複数の手法を組み合わせる



加重平均を用いて、各予測が予測したレーティングと決められた重みの積をすべて足し合わせた値を最終の予測レーティングとする

```
# 先ほどの例で用いたコードをインポートする
import similar_movie
import corrneighbors
import usermodel
from sklearn.linear_model import LinearRegression

# esには、複数の手法の予測結果が格納される。
# 各手法のestimate_all()は二次元配列を返し、
# その配列の要素は、対応するユーザと映画のレーティング予測である
es = [
  usermodel.estimate_all(), #第7章で用いた手法
  corrneighbors.estimate_all(), #類似ユーザから予測する手法
  similar_movie.estimate_all(), #類似映画から予測する手法
]

coefficients = []
#「一個抜き交差検定」のループを実行する
for u in xrange(reviews.shape[0]): # 全てのユーザに対して
    es0 = np.delete(es,u,1) # u番目のユーザを除く
    r0 = np.delete(reviews, u, 0)
    P0,P1 = np.where(r0 > 0) # 点数が付けられている要素だけ対象とする
    X = es[:,P0,P1]
    y = r0[r0 > 0]
    reg.fit(X.T,y)
    coefficients.append(reg.coef_)
    prediction = reg.predict(es[:,u,reviews[u] > 0].T)
    # 前と同様に誤差を計測する
```

、[0.25164062, 0.01258986, 0.60827019]になります。

バスケット分析

- 分析で扱うデータは、とのアイテムが一緒に購入されているか（気に入ったかどうかという情報は必要ありません）

スーパの買い物かごを分析する

- データの中身：数字の集合として構成される
- 統計データを算出する：
- ほんの数回しか購入されない商品がたくさん存在する

購入回数	該当する商品の数
1回	2224
2～3回	2438
4～7回	2508
8～15回	2251
16～31回	2182
32～63回	1940
64～127回	1523
128～511回	1225
512回以上	179

- アプリオリ・アルゴリズム：ある集合を多数集めたものを入力として受け取り、結果として、よく起こる組み合わせの集合（閾値より大きい支持度を持つ組み合わせなるアイテム集合）を返します。

  最小の支持度：アイテムが同時に購入された回数

  → 閾値として設定する必要がある

  目標：支持度の大きい組み合わせからなるアイテム集合を見つけること

アソシエーション・ルール・マイニング

Xであるならば**Y**である

→**X**を買った人は、通常よりも**Y**を買う傾向が高い

頻度集合から**X**と**Y**の可能な組み合わせを試行することで → たくさんのルールが作れる

ルールの有用性を評価する指標が必要

リフト値：通常の場合の**Y**が買われる確率とルールを適用した場合の**Y**が買われる確率の比率で表されます。

$$\mathrm{lift}(X \to Y) = \frac{P(Y|X)}{P(Y)}$$

P(Y): **Y**が含まれるトランザクションが全トランザクションに占める割合

P(Y|X): **Y**と**X**両方を含むトランザクションが**X**を含むトランザクションに占める割合

ベストセラーの商品だけをレコメンドするという問題を防ぐことができる

アソシエーション・ルール・マイニング

```
def rules_from_itemset(itemset, dataset):
  itemset = frozenset(itemset)
  nr_transactions = float(len(dataset))
  for item in itemset:
    consequent = frozenset([item])
    antecendent = itemset - consequent

    base = 0.0
    acount = 0.0 #「条件 (antecedent)」の回数
    ccount = 0.0 #「結論 (consequent)」の回数

    for d in dataset:
      if item in d: base += 1
      if d.issuperset(antecedent): account += 1
      if d.issuperset(itemset): ccount += 1

    base /= nr_transactions
    p_y_given_x = ccount/acount
    lift = p_y_given_x / base
    print('Rule {0} ->  {1} has lift {2}'
          .format(antecedent, consequent,lift))
```

回数：トランザクションの数
結果の回数：結果だけに含まれる商品のトランザクション
条件の回数：条件だけに含まれる商品のトランザクション
条件と結果の回数：条件と結果に含まれる商品の
トランザクションの数

1378/1379/1380の商品を含むトランザクション
-> 80
1269の商品が含むトランザクション
-> 57

条件確率：57/80 -> 71%

全体の0.3のトランザクションにしか1269の商品が含まれない

条件	結果	結果の回数	条件の回数	条件と結果の回数	リフト値
1378, 1379, 1380	1269	279 (0.3%)	80	57	225
48, 41, 976	117	1026 (1.1%)	122	51	35
48, 41, 16011	16010	1316 (1.5%)	165	159	64

進んだバスケット分析

- 買い物の順番を考慮に入れた手法：

- パーティーをやりたい人 → ゴミ袋を買う
- ゴミ袋を買う → パーティー用品（X）
- → 買い物の順番を推薦結果が影響を与える