フィルタリングとよばれる一連の技術は発展した。
調フィルタリングのアルゴリズムは、普通、大規模な人々の集団を検索し、あなたに好みが似た
の小集団を発見することで動作する。協調フィルタリングでは、彼らが好きな他のものも見て、そ
を組み合わせ、順序付けをした推薦のリストを作成する。類似している人々を選び出す方法や、
リストを作る際に人々の過去の選択を反映させる方法にはさまざまなやり方がある。本章ではその
つかについて説明する。

協調フィルタリングという用語は1992年にXerox PARCのDavid Goldbergが書いた"Using collaborative filtering to weave an information tapestry"という論文で使用された。彼はTapestryというシステムを設計した。Tapestryでは、人々に文書を面白い／面白くないで評価をさせ、この評価情報をドキュメントをフィルタする際に利用していた。現在、世の中には協調フィルタリングを利用したサイトがあふれている。映画、書籍、異性、ショッピング、さらにはポッドキャスト、記事やジョークに対するサイトまで存在する。

## 2　嗜好の収集

ずは、さまざまな人々と彼らの嗜好の情報をどうにかして表現する方法が必要である。Pythonで
を行うにはディクショナリをネストして使うのがシンプルなやり方である。このセクションの例題
行するには、以下のコードが書かれたrecommendations.pyという名前のファイルを用意し、デー
ットを作成するために次のコードを書き込もう。

```
映画の評者といくつかの映画に対する彼らの評点のディクショナリ
itics={'Lisa Rose': {'Lady in the Water': 2.5, 'Snakes on a Plane': 3.5,
'Just My Luck': 3.0, 'Superman Returns': 3.5, 'You, Me and Dupree': 2.5,
'The Night Listener': 3.0},
Gene Seymour': {'Lady in the Water': 3.0, 'Snakes on a Plane': 3.5,
'Just My Luck': 1.5, 'Superman Returns': 5.0, 'The Night Listener': 3.0,
'You, Me and Dupree': 3.5},
Michael Phillips': {'Lady in the Water': 2.5, 'Snakes on a Plane': 3.0,
'Superman Returns': 3.5, 'The Night Listener': 4.0},
Claudia Puig': {'Snakes on a Plane': 3.5, 'Just My Luck': 3.0,
'The Night Listener': 4.5, 'Superman Returns': 4.0,
'You, Me and Dupree': 2.5},
'Mick LaSalle': {'Lady in the Water': 3.0, 'Snakes on a Plane': 4.0,
'Just My Luck': 2.0, 'Superman Returns': 3.0, 'The Night Listener': 3.0,
'You, Me and Dupree': 2.0},
'Jack Matthews': {'Lady in the Water': 3.0, 'Snakes on a Plane': 4.0,
'The Night Listener': 3.0, 'Superman Returns': 5.0, 'You, Me and Dupree': 3.5},
'Toby': {'Snakes on a Plane':4.5,'You, Me and Dupree':1.0,'Superman Returns':4.0}}
```



本章ではPythonで対話的に実行しながら進めて行くので、recommendations.pyをPythonの対話型インタプリタが探し出せる場所に保存するとよい。保存場所はpython/Libでもよいのだが、もっとも楽な方法は保存したディレクトリと同じ場所でPythonのインタプリタを動作させるやり方だ。

このディクショナリは与えられた映画それぞれに対するみんな（と私）の評点を、1から5の数字で表している。嗜好がどのように表現されるにせよ、それらに数値を割り当てる必要がある。もしあなたがショッピングサイトを構築する場合であれば、誰かがアイテムを買った場合は1を割り当て、買っていない場合には0を割り当てるとよい。また、ニュースに対する投票のサイトを構築する場合、表2-1のように-1、0、1の数字を使い、それぞれに"好き"、"投票していない"、"嫌い"を割り当てるとよい。

表2-1　利用者の行動に対する数値割り当ての例

| コンサートチケット | | オンラインショッピング | | サイトの推薦 | |
|---|---|---|---|---|---|
| 買った | 1 | 買った | 2 | 好き | 1 |
| 買わなかった | 0 | 閲覧した | 1 | 投票せず | 0 |
| | | 買わなかった | 0 | 嫌い | −1 |

ディクショナリを使うと、アルゴリズムを試したり分かりやすく表示する際に便利である。検索や変更も簡単に行える。Pythonのインタプリタを起動させ、いくつかのコマンドを試してみよう。

```
c:\code\collective\chapter2> python
Python 2.4.1 (#65, Mar 30 2005, 09:13:57) [MSC v.1310 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
>> from recommendations import critics
>> critics['Lisa Rose']['Lady in the Water']
2.5
>> critics['Toby']['Snakes on a Plane']=4.5
>> critics['Toby']
{'Snakes on a Plane':4.5,,'Superman Returns':4.0,'You, Me and Dupree':1.0}
```

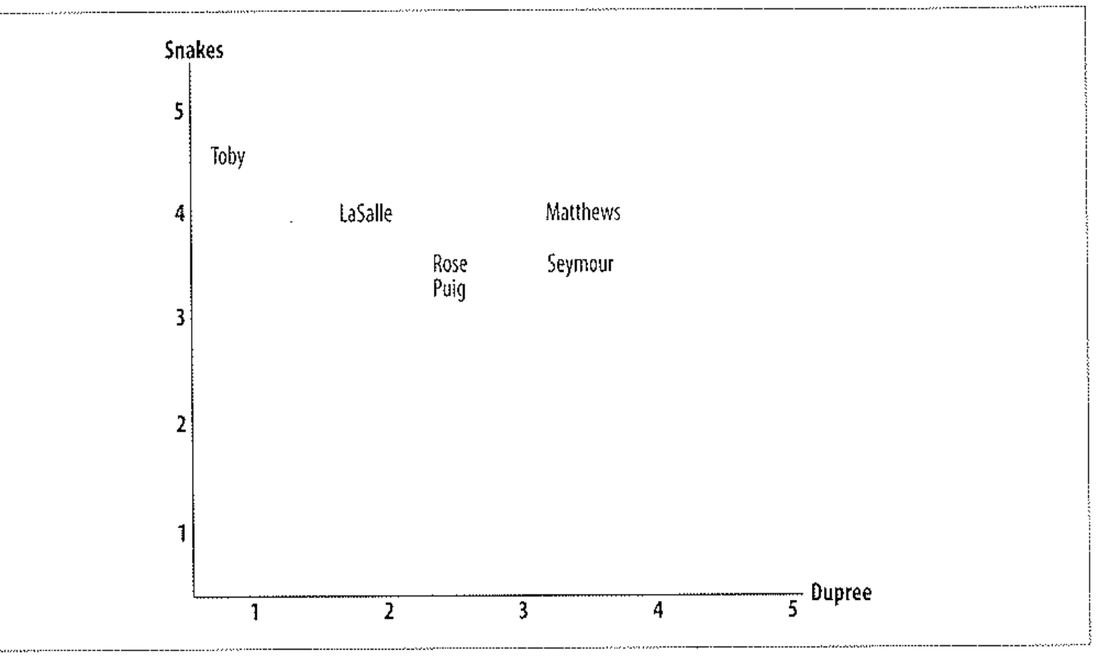
メモリ上のディクショナリには膨大な数の嗜好を収めることができるが、巨大なデータセットの場合、嗜好情報をデータベースに入れてもよい。

## 2.3　似ているユーザを探し出す

人々の嗜好についてのデータを集めた後は、人々が好みの上でどの程度似ているかを決める方法が必要になる。これは、それぞれの人を（当人以外の）すべての人と比較して、どの程度似ているかという事を表す**類似性スコア**を算出するとよい。このスコアを計算するにはいくつかの方法があるが、ここでは**ユークリッド距離**と**ピアソン相関**について説明する。

## .1　ユークリッド距離によるスコア

似性を表す数値を算出するための単純な方法の一つにユークリッド距離というものがある。これ々が評価したアイテムを軸にとってグラフに表示する。グラフ上に人々を配置し、それぞれがどれい近いかを見ることができる（図2-1）。




1　嗜好空間上の人々

の図は人々が**嗜好空間**に図示されている様子を表している。TobyはSnake軸の4.5、そしてree軸では1.0に位置している。この嗜好空間での距離が近ければ近いほど、嗜好は近いということなる。このグラフは2次元なため、一度に二つの映画での比較しか見ることができないが、もっとの映画に対してでも、原則は同じである。

ラフ上でのTobyとLaSalleの距離を測るには、それぞれの軸上での差を取り、それぞれを乗じて合わせる。そして、その合計の平方根を利用する。Pythonでは関数pow(n,2)で2乗の値を計算ことができ、関数sqrtを使って平方根を算出することができる。

```
 from math import sqrt
 psqrt(pow(4.5-4,2)+pow(1-1,2))
1622776601683795
```

の計算式で人々がどの程度似通っているかを表す距離を算出することができる。この距離が小さ々ほど似ているということになる。しかし、必要なのは似ていれば似ているほど高い数値を返すよ関数である。これはこの関数に1を加え（0で除算してエラーになってしまうのを防ぐため）逆数をことで実現できる。



```
>> 1/(1+sqrt(pow(5-4,2)+pow(4-1,2)))
0.2402530733520421
```

この新たな数式は常に0から1の間の値を返す。値が1の場合は二人の好みが寸分たがわず同じであるということを表している。類似性を算出するための関数を作るため、これらをまとめてしまうとよい。以下のコードをrecommendations.pyに付け加えよう。

```
from math import sqrt

# person1とperson2の距離を基にした類似性スコアを返す
def sim_distance(prefs,person1,person2):
  # 二人とも評価しているアイテムのリストを得る
  si={}
  for item in prefs[person1]:
    if item in prefs[person2]:
      si[item]=1

  # 両者共に評価しているものが一つもなければ0を返す
  if len(si)==0: return 0

  # すべての差の平方を足し合わせる
  sum_of_squares=sum([pow(prefs[person1][item]-prefs[person2][item],2)
                      for item in prefs[person1] if item in prefs[person2]])

  return 1/(1+sum_of_squares)
```

この関数を二人の名前と一緒に呼び出すことで、類似性スコアを得ることができる。Pythonインタプリタで次のようにして動作させよう。

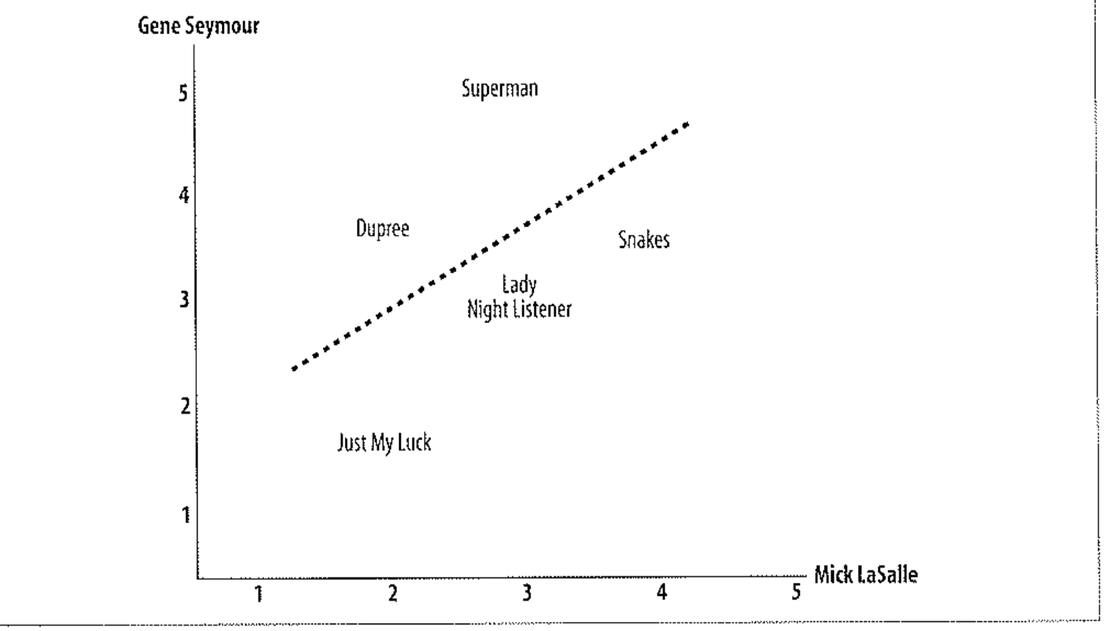
```
>>> reload(recommendations)
>>> recommendations.sim_distance(recommendations.critics,
... 'Lisa Rose','Gene Seymour')
0.148148148148
```

この場合、Lisa RoseとGene Seymourの類似性スコアが得られる。もっと似ている人や、逆に似ていない人を探せるかどうか、他の人々の名前でも試してみるとよい。

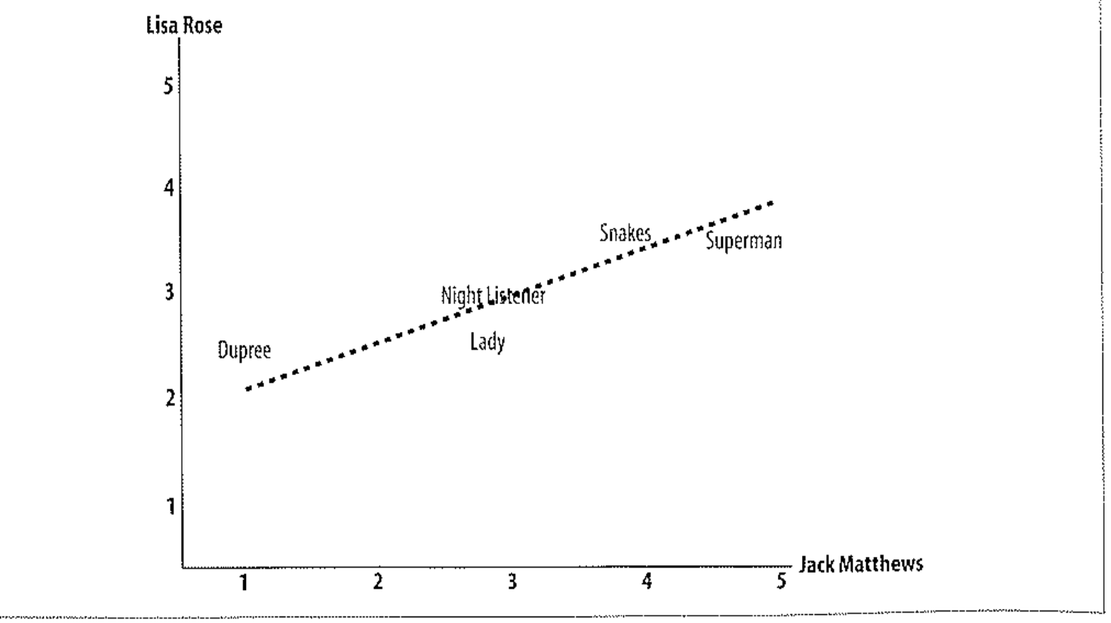
### 2.3.2　ピアソン相関によるスコア

人々の興味の類似度を決めるもう少し洗練された手法として、ピアソン相関係数を用いる方法がある。この相関係数は二つのデータセットがある直線にどの程度沿っているかを示す。この数式はユークリッド距離を算出するためのものよりも複雑であるが、データが正規化されていないような状況ではユークリッド距離を用いるより、よい結果を得られることが多い。正規化されていない状態の例として

たとえば映画の評価が平均より厳しい場合などが挙げられる。
の手法を視覚化するために、図2-2のように二つの評価をグラフに書いてみる。Supermanは
k LaSalleによって3点が付けられており、Gene Seymourによれば5点が付けられている。そのた
グラフ上では(3,5)に位置している。
ラフ上に見える直線は、グラフ上のすべてのアイテムにできるだけ近くなるように引かれた直線で
直線と呼ばれる。二人がすべての映画に対してまったく同じ評点をつけている場合、この直線は




-2　二人の評者の散布図上での比較




-3　高い相関スコアの二人の評者



グラフ上のアイテムがすべて乗るような斜めの直線になる。そして、スコアは完全に相関しているため1になる。ここで描いた図のような場合、いくつかの映画の評価で意見が合っていないため、相関スコアは0.4付近になる。もっと高い相関のものを図2-3に示す。この場合相関スコアは0.75程度になる。

ピアソン相関係数を利用する上で興味深いことの一つとして、よい成績の大判ぶるまいによる誤差を修正してくれるという点が挙げられる。この図ではJack MatthewsはLisa Roseより甘く点を付けている傾向がある。しかし、それでも直線は比較的彼らの嗜好は近いことを示すところに引かれている。もしある評者が他の評者と比較して高いスコアを付ける傾向にあったとしても、その二人の間のスコアの差が一貫していれば完全な相関が現れる。先ほど紹介したユークリッド距離によるスコアでは、一方の評者が一貫して厳しい点の付け方をしていると、例え好みが似ていたとしても、似ていないとみなされてしまう。このような挙動を問題ないとみなすかどうかは、あなたがどのようなアプリケーションを作りたいのかによって異なる。

ピアソン相関のスコアのためのコードは、まず両方の評者が得点を付けているアイテムを探し出す。そして、それぞれの評者の評点の合計と、評点の平方の合計を算出する。次に評価を掛け合わせた値の合計を算出する。最後に以下のコードの太字の部分でピアソン相関係数を算出する。距離を測定基準に利用していたときと違い、この数式は直感的ではないが、変数それぞれの変化の程度の積で割ることで、変数たちがどの程度一緒に変化していくかを教えてくれる。

この数式を利用するために、sim_distance関数と同じシグネチャの新たな関数をrecommendations.pyの中に作ろう。

```
# p1とp2のピアソン相関係数を返す
def sim_pearson(prefs,p1,p2):
  # 両者が互いに評価しているアイテムのリストを取得
  si={}
  for item in prefs[p1]:
    if item in prefs[p2]: si[item]=1

  # 要素の数を調べる
  n=len(si)

  #共に評価しているアイテムがなければ0を返す
  if n==0: return 0

  # すべての嗜好を合計する
  sum1=sum([prefs[p1][it] for it in si])
  sum2=sum([prefs[p2][it] for it in si])

  # 平方を合計する
  sum1Sq=sum([pow(prefs[p1][it],2) for it in si])
  sum2Sq=sum([pow(prefs[p2][it],2) for it in si])
```

```
# 積を合計する
pSum=sum([prefs[p1][it]*prefs[p2][it] for it in si])

# ピアソンによるスコアを計算する
num=pSum-(sum1*sum2/n)
den=sqrt((sum1Sq-pow(sum1,2)/n)*(sum2Sq-pow(sum2,2)/n))
if den==0: return 0

r=num/den

return r
```

この関数は-1から1の間の値を返す。値が1であれば、両者はすべてのアイテムに対してまったく同じ評価をしているということになる。距離を利用する方法と異なり、数値のスケールを変える必要はない。図2-3の相関スコアを得るためには次のようにするとよい。

```
>>> reload(recommendations)
>>> print recommendations.sim_pearson(recommendations.critics,
... 'Lisa Rose','Gene Seymour')
0.396059017191
```

### 2.3.3　どちらの類似性尺度を利用すべきなのか？

私はここまで異なる二つの尺度の関数を紹介してきたが、二つのデータセットの類似性を計るための方法は実はもっとたくさん存在する。どの方法がベストであるかは、あなたが作るアプリケーションによって異なる。ピアソンやユークリッド距離、またはその他の方法のどれがあなたにとって望ましい結果を返してくれるのかを試してみるとよい。

本章の残りの部分で紹介する関数はsimilarityというオプションのパラメータを持っている。このパラメータにsim_pearsonやsim_distanceと指定することで、使用する類似性関数を切り替えることができる。これを利用して試してみるとよい。類似性を計算するための方法としては、他にもJaccard係数やマンハッタン距離など、数多く存在する。これまで紹介した類似性関数と同じシグネチャを受け取り、高い数値が高い類似性を表すようなfloat値を返すという約束を守る限り、自分で類似性関数を作って試すことができる。

アイテムを比較するためのほかの尺度についてhttp://en.wikipedia.org/wiki/Metric_%28mathematics%29#Exampleで読むことができる。

### 2.3.4　評者をランキングする

ここまで二人の人々を比較する関数を作ってきた。あなたは与えられたある人に対するすべての人々のスコアを算出し、もっとも近い組を探す関数を作ることもできる。ここでは映画を見るときに誰のアドバイスを受ければよいか判断するため、私の映画の好みにもっとも似ている映画の評者を探し出すことをやってみたい。人々を特定の人の好みに近い順に並べたリストを得るための次の関数をrecommendations.pyに付け加えてみよう。



```
# ディクショナリprefsからpersonにもっともマッチするものたちを返す
# 結果の数と類似性関数はオプションのパラメータ
def topMatches(prefs,person,n=5,similarity=sim_pearson):
  scores=[(similarity(prefs,person,other),other)
          for other in prefs if other!=person]
  # 高スコアがリストの最初に来るように並び替える
  scores.sort()
  scores.reverse()
  return scores[0:n]
```

この関数はPythonのリスト内包を使い、私と私以外のすべてのユーザを先ほど定義した尺度を利用して比較している。そして、ソートされた結果の中から最初のn個の結果を返している。

この関数を私の名前と一緒に呼び出せば、評者たちとその類似性スコアを得ることができる。

```
>> reload(recommendations)
>> recommendations.topMatches(recommendations.critics,'Toby',n=3)
[(0.99124070716192991, 'Lisa Rose'), (0.92447345164190486, 'Mick LaSalle'),
 (0.89340514744156474, 'Claudia Puig')]
```

この結果によると私にもっとも似ているのはLisa Roseであるため、彼女のレビューを読むべきだということがわかる。もしあなたがこれらの映画のどれかを見たことがあるなら、このディクショナリにあなたの名前とあなたの評点を付け加えてみて、あなたのお気に入りの評者が誰になるのか見てみるとよい。

## 2.4　アイテムを推薦する

よい評者を探し出すというのは素晴らしいことではあるが、私が本当にやりたいのは映画を今すぐ推薦してもらうことである。私に似た嗜好を持つ人間を探し出し、その人が好きな映画の中から私がまだ見ていないものを探すことはできるが、それでは回りくどい。このようなやり方では、いくつかの私が好きであるはずの映画を見ていない評者を選んでしまうことで、漏れが生じる可能性がある。また、ある映画に対して、他のすべての評者はよい評価をしていないにも関わらず、topMatchesによって選ばれた評者だけが、奇妙なことにその映画に対してよい評価をしてしまっている場合に出くわす可能性もある。

これらの問題を解決するためには、評者に順位付けをした重み付きスコアを算出することで、アイテムにスコアを付ける必要がある。私以外全員の評者の評点を集め、それぞれの映画への点数を私との類似性に掛け合わせる。表2-2でこの実際の様子を示す。

2-2　筆者のための推薦の作成

| 評者 | 類似性 | Night | S.xNight | Lady | S.xLady | Luck | S.xLuck |
|---|---|---|---|---|---|---|---|
| Rose | 0.99 | 3.0 | 2.97 | 2.5 | 2.48 | 3.0 | 2.97 |
| Seymour | 0.38 | 3.0 | 1.14 | 3.0 | 1.14 | 1.5 | 0.57 |
| Puig | 0.89 | 4.5 | 4.02 | | | 3.0 | 2.68 |
| LaSalle | 0.92 | 3.0 | 2.77 | 3.0 | 2.77 | 2.0 | 1.85 |
| Matthews | 0.66 | 3.0 | 1.99 | 3.0 | 1.99 | | |
| 合計 | | | 12.89 | | 8.38 | | 8.07 |
| Sim. Sum | | | 3.84 | | 2.95 | | 3.18 |
| 合計/Sim.Sum | | | 3.35 | | 2.83 | | 2.53 |

この表はそれぞれの評者の相関値と、私がまだ見ていない3つの映画（The Night Listener、Lady n the Water、Just My Luck）への評点が示されている。S.xから始まる列は評点を掛け合わせた類似 の列である。そのため、私により似ている人の評点の方が、似ていない人の評点より、全体の点数 に大きく影響を与えるようになっている。合計の行はこれらの数字の合計を表している。

ランキングを算出するために合計値だけを利用することもできるが、それでは多くの人に見られて いる映画の点数が上がってしまう。これを正すため、その映画を見た評者の類似度の合計（表のSim. umの行）で割る必要がある。The Night Listenerは全員に見られているため、すべての類似度の合 計で除算する。Lady in the WaterはPuigは見ていないので、Puig以外の人々の類似度の合計で除算 する。最後の行がこの割り算を行った結果を示している。

次のコードは非常に理解しやすく、また、ユークリッド距離でもピアソン相関スコアでも動作する。 ecommendations.pyに以下を加えるとよい。

```
# person以外の全ユーザの評点の重み付き平均を使い、personへの推薦を算出する
def getRecommendations(prefs,person,similarity=sim_pearson):
  totals={}
  simSums={}
  for other in prefs:
    # 自分自身とは比較しない
    if other==person: continue
    sim=similarity(prefs,person,other)

    # 0以下のスコアは無視する
    if sim<=0: continue

    for item in prefs[other]:
      # まだ見ていない映画の得点のみを算出
      if item not in prefs[person] or prefs[person][item]==0:
        # 類似度　*　スコア
        totals.setdefault(item,0)
```



```
        totals[item]+=prefs[other][item]*sim
        # 類似度を合計
        simSums.setdefault(item,0)
        simSums[item]+=sim

  # 正規化したリストを作る
  rankings=[(total/simSums[item],item) for item,total in totals.items()]

  # ソート済みのリストを返す
  rankings.sort()
  rankings.reverse()
  return rankings
```

このコードはprefsディクショナリの中のすべての人々に対してループされ、それぞれの人と特定のpersonとの類似性を算出する。そして、彼らが点数をつけたアイテムをすべてループする。太字の行ではアイテムの最終的なスコアを算出している――それぞれのアイテムには類似度が掛け合わされ、その積は合計されている。そして最後にそのスコアを類似度の合計で割ることで正規化し、ソート済みの結果が返される。

これであなたは次に私が見るべき映画を知ることができる。

```
>>> reload(recommendations)
>>> recommendations.getRecommendations(recommendations.critics,'Toby')
[(3.3477895267131013, 'The Night Listener'), (2.8325499182641614, 'Lady in the
Water'), (2.5309807037655645, 'Just My Luck')]
>>> recommendations.getRecommendations(recommendations.critics,'Toby',
... similarity=recommendations.sim_distance)
[(3.5002478401415877, 'The Night Listener'), (2.7561242939959363, 'Lady in the
Water'), (2.4619884860743739, 'Just My Luck')]
```

順序づけられた映画のリストを得ることができるだけでなく、それぞれの映画に対する私の評点の予想を得ることができている。これを参考に、映画を見たいか、それともまったく別のことをやりたいのかを決めることができる。あなたが作りたいアプリケーションにもよるが、与えられたユーザの基準に合うものがない場合、何も推薦をしないという風に作ることもできる。また、この結果はどの類似性尺度を選んでもあまり大差はない。

ここまででであなたはどのような商品やリンクにでも適用することができる完璧な推薦システムを作り上げたことになる。人々とアイテム、そしてスコアのディクショナリを用意さえすれば、これを利用して誰に対してでも推薦を行うことができる。本章の後半では、del.icio.usのAPIを通じて、人々にWebサイトを推薦するための実際のデータを取得する方法について説明する。

## 5　似ている製品

現在、あなたは似ている人々を探し出し、特定の人のために製品を推薦する方法について理解してる。しかし、製品同士、似ているものを知りたい場合にはどのようにすればよいだろうか？　あなたこのような例にショッピングサイトで出くわしたことがあるはずだ。特に、サイトがあなたの情報をだ十分に集めていない場合に見かけたかもしれない。図2-4はAmazonのProgramming Pythonのめのページである。

Customers who bought this item also bought

Learning Python, Second Edition by Mark Lutz
Python Cookbook by Alex Martelli
Python in a Nutshell by Alex Martelli
Python Essential Reference (2nd Edition) by David Beazley
Foundations of Python Network Programming (Foundations) by John Goerzen

▸ **Explore similar items :** Books (42)

2-4　AmazonによるProgramming Pythonと似ている商品

この場合、類似度を決めるには、特定のアイテムを誰が好きなのかを調べ、彼らが好きな他のアイムを探すとよい。これは本質的には先ほど行った人々の間の類似度を決めるやり方と同じである。々とアイテムを入れ替えるだけで実現できる。つまり、ディクショナリを変更すれば先ほどあなたがいた方法を利用することができる。

```
{'Lisa Rose': {'Lady in the Water': 2.5, 'Snakes on a Plane': 3.5},
'Gene Seymour': {'Lady in the Water': 3.0, 'Snakes on a Plane': 3.5}}
```

これを次のように変換する。

```
{'Lady in the Water':{'Lisa Rose':2.5,'Gene Seymour':3.0},
'Snakes on a Plane':{'Lisa Rose':3.5,'Gene Seymour':3.5}} etc..
```

この変換を行うための関数をrecommendations.pyに付け加える。

```
def transformPrefs(prefs):
  result={}
  for person in prefs:
    for item in prefs[person]:
      result.setdefault(item,{})
```



```
      # itemとpersonを入れ替える
      result[item][person]=prefs[person][item]
  return result
```

これで、先ほど使用した関数topMatchesを呼び出してSuperman Returnsに似ている映画たちを探し出すことができる。

```
>> reload(recommendations)
>> movies=recommendations.transformPrefs(recommendations.critics)
>> recommendations.topMatches(movies,'Superman Returns')
[(0.657, 'You, Me and Dupree'), (0.487, 'Lady in the Water'), (0.111, 'Snakes on a
Plane'), (-0.179, 'The Night Listener'), (-0.422, 'Just My Luck')]
```

この例の場合、マイナスの相関スコアもあることに注意してほしい。これはSuperman Returnsを好きな人はJust My Luckのことを好まない傾向にあるということを示している（図2-5）。

さらに、映画の評者を推薦してもらうこともできる。これはあなたがプレミアに誘う人を決めたい場合に役立つだろう。

```
>> recommendations.getRecommendations(movies,'Just My Luck')
[(4.0, 'Michael Phillips'), (3.0, 'Jack Matthews')]
```

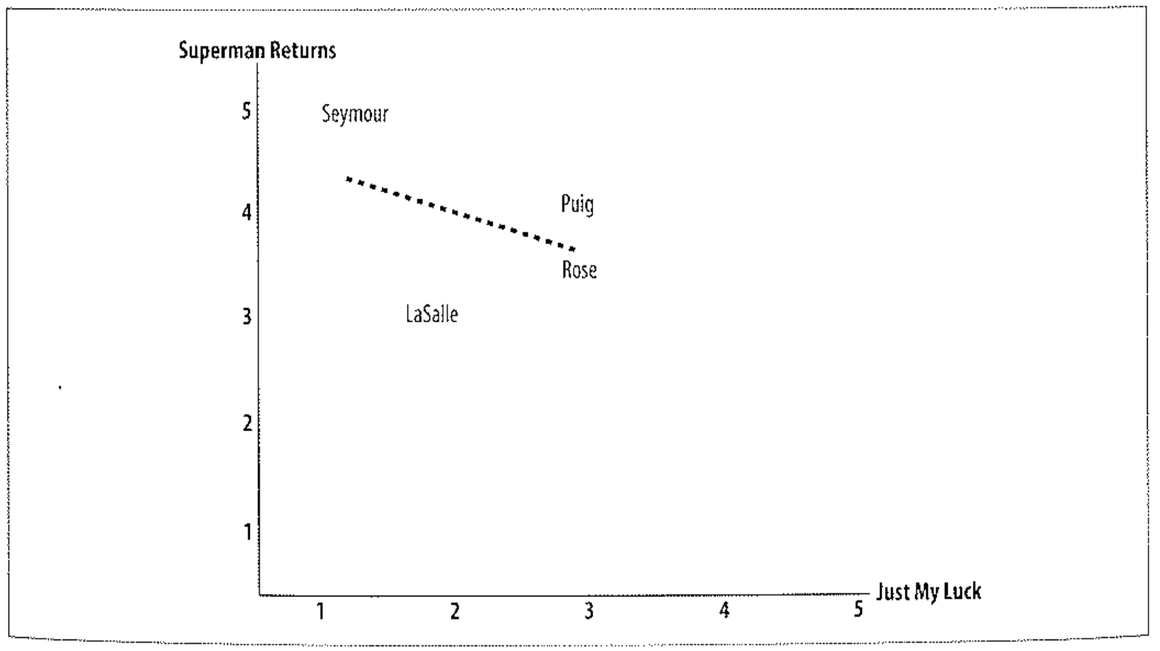



図2-5　Superman ReturnsとJust My Luckの負の相関

人々とアイテムを入れ替えることで役立つ結果を得ることができる、とはっきり言い切ることはできないが、多くの場合、比較してみると面白いと感じるような結果を得ることができるだろう。オンライ

## 5　似ている製品

現在、あなたは似ている人々を探し出し、特定の人のために製品を推薦する方法について理解してる。しかし、製品同士、似ているものを知りたい場合にはどのようにすればよいだろうか？　あなたこのような例にショッピングサイトで出くわしたことがあるはずだ。特に、サイトがあなたの情報をだ十分に集めていない場合に見かけたかもしれない。図2-4はAmazonのProgramming Pythonのめのページである。

Customers who bought this item also bought

Learning Python, Second Edition by Mark Lutz

Python Cookbook by Alex Martelli

Python in a Nutshell by Alex Martelli

Python Essential Reference (2nd Edition) by David Beazley

Foundations of Python Network Programming (Foundations) by John Goerzen

▸ **Explore similar items :** Books (42)

2-4　AmazonによるProgramming Pythonと似ている商品

この場合、類似度を決めるには、特定のアイテムを誰が好きなのかを調べ、彼らが好きな他のアイムを探すとよい。これは本質的には先ほど行った人々の間の類似度を決めるやり方と同じである。々とアイテムを入れ替えるだけで実現できる。つまり、ディクショナリを変更すれば先ほどあなたがいた方法を利用することができる。

```
{'Lisa Rose': {'Lady in the Water': 2.5, 'Snakes on a Plane': 3.5},
'Gene Seymour': {'Lady in the Water': 3.0, 'Snakes on a Plane': 3.5}}
```

これを次のように変換する。

```
{'Lady in the Water':{'Lisa Rose':2.5,'Gene Seymour':3.0},
'Snakes on a Plane':{'Lisa Rose':3.5,'Gene Seymour':3.5}} etc..
```

この変換を行うための関数をrecommendations.pyに付け加える。

```
def transformPrefs(prefs):
  result={}
  for person in prefs:
    for item in prefs[person]:
      result.setdefault(item,{})
```



```
      # itemとpersonを入れ替える
      result[item][person]=prefs[person][item]
  return result
```

これで、先ほど使用した関数topMatchesを呼び出してSuperman Returnsに似ている映画たちを探し出すことができる。

```
>> reload(recommendations)
>> movies=recommendations.transformPrefs(recommendations.critics)
>> recommendations.topMatches(movies,'Superman Returns')
[(0.657, 'You, Me and Dupree'), (0.487, 'Lady in the Water'), (0.111, 'Snakes on a
Plane'), (-0.179, 'The Night Listener'), (-0.422, 'Just My Luck')]
```

この例の場合、マイナスの相関スコアもあることに注意してほしい。これはSuperman Returnsを好きな人はJust My Luckのことを好まない傾向にあるということを示している（図2-5）。

さらに、映画の評者を推薦してもらうこともできる。これはあなたがプレミアに誘う人を決めたい場合に役立つだろう。

```
>> recommendations.getRecommendations(movies,'Just My Luck')
[(4.0, 'Michael Phillips'), (3.0, 'Jack Matthews')]
```

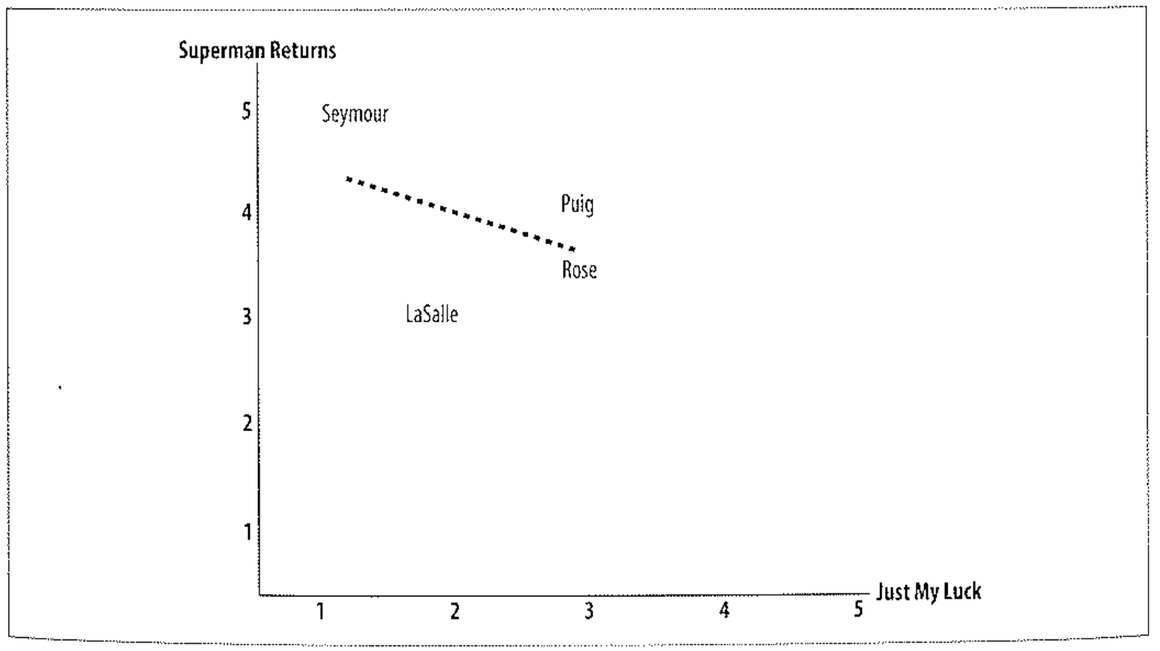


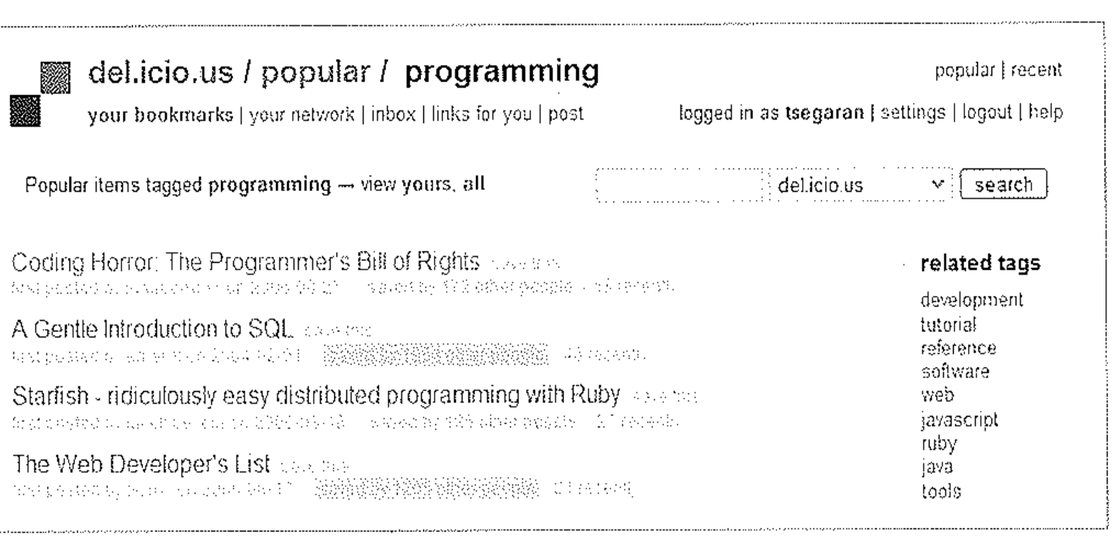

図2-5　Superman ReturnsとJust My Luckの負の相関

人々とアイテムを入れ替えることで役立つ結果を得ることができる、とはっきり言い切ることはできないが、多くの場合、比較してみると面白いと感じるような結果を得ることができるだろう。オンライ

ョップの業者は個々の利用者へ製品を推薦するために購買履歴情報を集めている。あなたが行っ
うに、製品と人々を反転させる行為は、彼らにとっては特定の製品を買いそうな人を検索すること
できるようになることを意味している。これは、特定の商品の在庫セールの売り込み方を計画する際
非常に役に立つ。別の可能性としては、リンク推薦サイトの新たなリンクを、そのリンクをもっとも
しめそうな人々に対して提示する際に役に立つかもしれない。

## 6　del.icio.usのリンクを推薦するシステムを作る

このセクションではオンラインブックマークのサイトとしてもっとも人気のあるサイトの一つである
.icio.usからデータを取得する方法について説明し、次にそのデータを使って似ているユーザを探
し、彼らがまだ見ていないリンクを推薦する方法について説明していく。http://del.icio.usでアク
スできるこのサイトでアカウントを作れば、ユーザはリンクを投稿して後で参照することができるよ
になる。このサイトでは他のユーザが投稿したリンクを閲覧することもできる。そして多くの人々に
って投稿された"popular"なリンクを見ることができる。実際のdel.icio.usのページの例を示す。




2-6　del.icio.usのprogrammingに関するpopularページ

他のソーシャルブックマークサイトと異なり、del.icio.usは似たような人々を探したり、ユーザが好
そうなリンクを推薦するような機能は持っていない（執筆時現在）。ありがたいことに本章で説明す
テクニックを使えば、この機能を自分で付け加えることができる。

### 6.1　del.icio.usのAPI

del.icio.usのデータはAPIを通じて利用することができる。データはXMLで返される。Pythonの
PIも存在するので、楽をするために次のURLからダウンロードしておこう。(http://code.google.com/
ydelicious/source またはhttp://oreilly.com/catalog/9780596529321)



このセクションでの例に取り組むためには、このライブラリの最新バージョンをダウンロードして
Python Libraryパスに置いておく必要がある（このライブラリのインストールに関しての詳細は付録A
を参照）。

このライブラリは人々が投稿したリンクを取得するためのシンプルな関数をいくつか持っている。た
とえばprogrammingに関して最新の"popular"な投稿リストを取得するにはget_popularを呼び出せ
ばよい：

```
>> import pydelicious
>> pydelicious.get_popular(tag='programming')
[{'count': '', 'extended': '', 'hash': '', 'description': u'How To Write
Unmaintainable Code', 'tags': '', 'href': u'http://thc.segfault.net/root/phun/
unmaintain.html', 'user': u'dorsia', 'dt': u'2006-08-19T09:48:56Z'}, {'count': '',
'extended': '', 'hash': '', 'description': u'Threading in C#', 'tags': '', 'href':
u'http://www.albahari.com/threading/', 'user': u'mmihale', 'dt': u'2006-05-17T18:09:
24Z'},
...etc...
```

この関数はディクショナリのリストを返していることがわかるだろう。それぞれのディクショナリは
URL、詳細、そして誰が投稿したのかという情報を持っている。ここでは実際のデータを使って試し
ているので、あなたが試す場合の結果はここで示している結果とは異なるだろう。これから使う関数は
他に二つある。get_urlpostsとget_userpostsだ。get_urlpostsは与えられたURLのすべての投稿
を返す。get_userpostは与えられたユーザのすべての投稿を返す。これらの関数のデータもここで示
した例と同様に返される。

### 2.6.2　データセットを作る

del.icio.usからすべてのユーザの投稿をダウンロードするのは無理がある。したがって、一部を選び
出す必要がある。どうやって選んでもいいが、面白い結果を得るためには、投稿頻度が高く、似たよ
うな内容の投稿を繰り返しているような人を探し出すというのもよいだろう。

これを行う方法として、特定のタグでの"popular"なリンクを最近投稿した人々のリストを得るやり
方がある。deliciousrec.pyという名前のファイルを作成して、以下のコードを入力しよう。

```
from pydelicious import get_popular,get_userposts,get_urlposts
import time

def initializeUserDict(tag,count=5):
  user_dict={}

  # popularな投稿をcount番目まで取得
  for p1 in get_popular(tag=tag)[0:count]:
    # このリンクを投稿したすべてのユーザを取得
    for p2 in get_urlposts(p1['href']):
```

```
    user=p2['user']
    user_dict[user]={}
  return user_dict
```

このコードはユーザの名前がキーになっているディクショナリを作り出す。ディクショナリのそれぞれの要素は、これからリンクを入れるための空のディクショナリを参照している。このAPIはそのリンクを投稿した最新の30名の投稿者しか返さない。この関数では最初の5つのリンクを投稿したユーザを利用して大きなデータセットを作る。

映画評者のデータセットとは異なり、この場合の評価は二つの値が考えられる。つまり、このリンクを投稿していないユーザは0で、もし投稿している場合は1にする。このAPIを使うことで、すべてのユーザの評点で埋めるための関数を作ることができる。次のコードをdeliciousrec.pyに付け加えよう。

```
def fillItems(user_dict):
  all_items={}
  # すべてのユーザによって投稿されたリンクを取得
  for user in user_dict:
    for i in range(3):
      try:
        posts=get_userposts(user)
        break
      except:
        print "Failed user "+user+", retrying"
        time.sleep(4)
    for post in posts:
      url=post['href']
      user_dict[user][url]=1.0
      all_items[url]=1

  # 空のアイテムを0で埋める
  for ratings in user_dict.values():
    for item in all_items:
      if item not in ratings:
        ratings[item]=0.0
```

このコードで、本章の最初の部分で作った評者のディクショナリと似たようなデータセットを作ることができる。

```
>> from deliciousrec import *
>> delusers=initializeUserDict('programming')
>> delusers ['tsegaran']={} # あなたがdeliciousのユーザであれば、あなた自身を付け加えよう
>> fillItems(delusers)
```



3行目ではtsegaran（筆者）をリストに加えている。もしあなたがdel.icio.usのユーザであれば、tsegaranを自分のユーザネームに置き換えるとよい。

fillItemsの実行には数分かかるかもしれない。これはサイトに対して数百回のリクエストを行っているためである。このAPIは短い時間に多くのリクエストが繰り返されるとブロックすることがある。そのような場合、このコードはリクエストを一旦休み、3回を上限としてリトライを行う。

### 2.6.3 ご近所さんとリンクの推薦

これであなたはデータセットを作り上げたことになる。このデータセットには、映画評者のデータセットに対して使った関数と同じ関数を適用することができる。試しに適当なユーザを選んで、そのユーザに似た嗜好を持ったユーザを探し出してみる。次のコードを動かしてみよう。

```
>> import random
>> user=delusers.keys()[random.randint(0,len(delusers)-1)]
>> user
u'veza'
>> recommendations.topMatches(delusers,user)
[(0.083, u'kuzz99'), (0.083, u'arturoochoa'), (0.083, u'NickSmith'), (0.083,
u'MichaelDahl'), (0.050, u'zinggoat')]
```

getRecommendationsを呼び出して、このユーザが好みそうなリンクを推薦することもできる。getRecommendationsはすべてのアイテムをソートして返すため、最初の10個程度に制限しておいたほうがよい。

```
>> recommendations.getRecommendations(delusers,user)[0:10]
[(0.278, u'http://www.devlisting.com/'),
(0.276, u'http://www.howtoforge.com/linux_ldap_authentication'),
(0.191, u'http://yarivsblog.com/articles/2006/08/09/secret-weapons-for-startups'),
(0.191, u'http://www.dadgum.com/james/performance.html'),
(0.191, u'http://www.codinghorror.com/blog/archives/000666.html')]
```

もちろん、先ほど紹介したように、この嗜好リストも入れ替えることができる。これによって人々ではなく、リンクという観点から検索することができる。あなたが特に気に入った特定のリンクに似ているリンクを探すためには次のようなコードを試してみるとよい。

```
>> url=recommendations.getRecommendations(delusers,user)[0][1]
>> recommendations.topMatches(recommendations.transformPrefs(delusers),url)
[(0.312, u'http://www.fonttester.com/'),
(0.312, u'http://www.cssremix.com/'),
(0.266, u'http://www.logoorange.com/color/color-codes-chart.php'),
(0.254, u'http://yotophoto.com/'),
(0.254, u'http://www.wpdfd.com/editorial/basics/index.html')]
```

これでおしまいだ！ これだけであなたはdel.icio.usに対して推薦エンジンを付け加えることに成功したことになる。他にもやれることはいろいろある。たとえば、del.icio.usはタグで検索することもできるため、お互いに似ているタグを探し出すこともできるし、あるページを"popular"にするために同じリンクを複数のアカウントで繰り返し投稿しているユーザを探し出すこともできる。

## 2.7 アイテムベースのフィルタリング

今まで実装してきた推薦エンジンは、データセットを作成するためには、すべてのユーザからのランキングが必要だった。これは人々やアイテムの数が数千件程度であればうまく動作するだろう。しかし、Amazonのような巨大なサイトでは客や製品の数は数百万件存在する。このようなサイトでは、あるユーザを他のすべてのユーザと比べたり、ユーザが評価したそれぞれの製品を比較すると、ものすごい時間がかかる。また、数百万点の製品を売るようなサイトでは、人々の重なりあう部分が少ないため、誰と誰が似ているのかを決めることが難しい。

これまで使用してきた技術はユーザベースの協調フィルタリングと呼ばれている。これに代わるものとしてアイテムベースの協調フィルタリングというものがある。巨大なデータセットが対象の場合、アイテムベースの協調フィルタリングのほうがよい結果を生み出してくれる。また、計算の大部分は事前に実行しておくことが可能なので、推薦が必要なユーザは、より時間をかけずに結果を得ることができる。

アイテムベースの協調フィルタリングの手順はすでに解説してきた内容に大部分が含まれている。大まかにいうと、それぞれのアイテムに似ているアイテムたちを前もって計算しておくということである。そして、あるユーザに推薦をしたくなった時にはそのユーザが高く評価しているアイテムたちを参照し、それらに対して似ている順に重み付けされたアイテムたちのリストを作り出す。ここでの重要な違いは、最初のステップではすべてのデータを調べる必要があるが、アイテム間の関係というのは、人間同士の関係ほど頻繁には変わらないという点である。これはそれぞれのアイテムに似ているアイテムを見つけるために計算し続ける必要はないということを意味している。このような計算はトラフィックの軽い時間帯やメインのアプリケーションとは別のコンピュータでやればよい。

### 2.7.1 アイテム間の類似度のデータセットを作る

アイテムを比較するためにまずやらなければならないのは、似ているアイテムたちの完全なデータセットを作るための関数を書くことだ。先程も述べたように、これは推薦を行うときに毎回行う必要があるわけではない。一度データセットを作りさえすれば、必要な時に再利用することができる。

このデータセットを作り出すために、次の関数をrecommendations.pyに付け加えよう。

```
def calculateSimilarItems(prefs,n=10):
  # アイテムをキーとして持ち、それぞれのアイテムに似ている
  # アイテムのリストを値として持つディクショナリを作る。
```



```
  result={}

  # 嗜好の行列をアイテム中心な形に反転させる
  itemPrefs=transformPrefs(prefs)
  c=0
  for item in itemPrefs:
    # 巨大なデータセット用にステータスを表示
    c+=1
    if c%100==0: print "%d / %d" % (c,len(itemPrefs))
    # このアイテムにもっとも似ているアイテムたちを探す
    scores=topMatches(itemPrefs,item,n=n,similarity=sim_distance)
    result[item]=scores
  return result
```

この関数はまず、先ほど定義したtransformPrefsを使ってスコアのディクショナリを反転させて、それぞれのユーザにアイテムがどのように評価されているかというリストを作っている。そしてすべてのアイテムをループして、この反転させたディクショナリをtopMatchesに渡し、似ているアイテムたちをその類似性スコアと共に取得している。最後にアイテムをキーとし、それぞれのアイテムにもっとも似ているアイテムたちのリストを持ったディクショナリを返している。

あなたのPythonの対話型セッションで、このアイテムの類似度のデータセットを作り、どんなものか見てみよう。

```
>>> reload(recommendations)
>>> itemsim=recommendations.calculateSimilarItems(recommendations.critics)
>>> itemsim
{'Lady in the Water': [(0.40000000000000002, 'You, Me and Dupree'),
                       (0.2857142857142857, 'The Night Listener'),...
 'Snakes on a Plane': [(0.22222222222222221, 'Lady in the Water'),
                       (0.18181818181818182, 'The Night Listener'),...
etc.
```

この関数をアイテムの類似度が最新な状態に保たれる程度の頻度で走らせるとよい。ユーザと、評価されたアイテムの数が少ない最初のころにはこれをより多く走らせる必要があるが、ユーザの数が増えればこのアイテム同士の類似性スコアは安定してくるので、この関数を呼び出す必要は少なくなる。

### 2.7.2 推薦を行う

ここまででですべてのデータセットを調べることなしにアイテムの類似度のディクショナリを使って推薦を行うための準備ができた。これから特定のユーザが評価したすべてのアイテムを取得し、似ているアイテムを探し、類似度を利用して重みをつける。アイテムのディクショナリを使えばこの類似度は簡単に得ることができる。

表2-3はアイテムベースのアプローチで推薦を行う過程を示している。表2-2とは異なり、他の評者

たちはまったく表には現れていない。その代わりに私が評価した映画と、まだ評価していない映画の表になっている。

表2-3 筆者のためのアイテムベースの推薦

| 映画 | 評価 | Night | R.xNight | Lady | R.xLady | Luck | R.xLuck |
|---|---|---|---|---|---|---|---|
| Snakes | 4.5 | 0.182 | 0.818 | 0.222 | 0.999 | 0.105 | 0.474 |
| Superman | 4.0 | 0.103 | 0.412 | 0.091 | 0.363 | 0.065 | 0.258 |
| Dupree | 1.0 | 0.148 | 0.148 | 0.4 | 0.4 | 0.182 | 0.182 |
| 合計 | | 0.433 | 1.378 | 0.713 | 1.764 | 0.352 | 0.914 |
| 正規化 | | | 3.183 | | 2.473 | | 2.598 |

それぞれの行は私がすでに見た映画の名前と、その映画に対する私の個人的な評価を保持している。そしてそれぞれの行の映画に対して、私がまだ見ていない映画がどれくらい似ているかということを示している列がある。例えば、SupermanとNight Listenerの類似度は0.103であると表示されている。R.xで始まる列は類似度と私の評点を掛け合わせたものを表している。私はSupermanには4.0を付けているため、Supermanの行のNightの隣にあるR.xNightは4.0 × 0.103=0.412となっている。

合計の行ではそれぞれの映画の類似度の合計とR.xの合計が入っている。それぞれの映画に対する私の評点を予測するには、R.xの列の合計を類似度の列の合計で割ればよい。つまり、Night Listenerに対する私の評点の予想は1.378/0.433 = 3.183となる。

これを実際に行うためには次の最後の関数をrecommendations.pyに付け加えるとよい。

```
def getRecommendedItems(prefs,itemMatch,user):
  userRatings=prefs[user]
  scores={}
  totalSim={}

  # このユーザに評価されたアイテムをループする
  for (item,rating) in userRatings.items():

    # このアイテムに似ているアイテムたちをループする
    for (similarity,item2) in itemMatch[item]:

      # このアイテムに対してユーザがすでに評価を行っていれば無視する
      if item2 in userRatings: continue

      # 評点と類似度を掛け合わせたものの合計で重みづけする
      scores.setdefault(item2,0)
      scores[item2]+=similarity*rating

      # すべての類似度の合計
      totalSim.setdefault(item2,0)
      totalSim[item2]+=similarity
```



```
  # 正規化のため、それぞれの重み付けしたスコアを類似度の合計で割る
  rankings=[(score/totalSim[item],item) for item,score in scores.items()]

  # 降順に並べたランキングを返す
  rankings.sort()
  rankings.reverse()
  return rankings
```

この関数を先ほど作った類似性データセットに適用することで、Tobyに対する新たな推薦を行うことができる。

```
>> reload(recommendations)
>> recommendations.getRecommendedItems(recommendations.critics,itemsim,'Toby')
[(3.182, 'The Night Listener'),
 (2.598, 'Just My Luck'),
 (2.473, 'Lady in the Water')]
```

Night Listenerはまだ余裕を持ってトップにいる。Just My LuckとLady in the Waterは互いに近いところに位置はしているが、順位が変わっている。ここで着目すべきは、アイテムの類似性データセットは事前に計算されているため、getRecommendedItemsが呼び出された際には、他の評者たち全員の類似性スコアを計算する必要がないという点である。

## 2.8 MovieLensのデータセットを使う

最後のサンプルとして、MovieLensという実際の映画の評価のデータセットについて見てみよう。MovieLensはGroupLens Projectによってミネソタ大学で作られた。このデータセットはhttp://www.grouplens.org/node/73#attachmentsからダウンロードすることができる。2種類のデータセットがあるが、100,000件のデータセットの方をダウンロードしよう。tar.gzとzip形式で用意されているので、プラットフォームに合わせて好きな方を選ぶとよい。

このアーカイブはいくつかのファイルを含んでいる。この中でも面白いのは映画のIDとタイトルのリストを含んでいるu.itemと、次のようなフォーマットで映画の実際の評価が記録されているu.dataである。

```
196 242 3 881250949
186 302 3 891717742
22 377 1 878887116
244 51 2 880606923
166 346 1 886397596
298 474 4 884182806
```

それぞれの行はユーザID、映画のID、ユーザによるその映画の評価、そしてタイムスタンプから構成されている。映画のタイトルを得ることはできるが、ユーザのデータは匿名になっている。そこでここではユーザIDを対象に取り組んでいく。このデータセットは1682本の映画に対する943名のユーザの評価から構成されている。各ユーザは最低でも20本の映画について評価している。

このデータセットを読み込むためにloadMovieLensというメソッドをrecommendations.pyに付け加えよう。

```
def loadMovieLens(path='/data/movielens'):

  # 映画のタイトルを得る
  movies={}
  for line in open(path+'/u.item'):
    (id,title)=line.split('|')[0:2]
    movies[id]=title

  # データの読み込み
  prefs={}
  for line in open(path+'/u.data'):
    (user,movieid,rating,ts)=line.split('\t')
    prefs.setdefault(user,{})
    prefs[user][movies[movieid]]=float(rating)
  return prefs
```

実際にPythonのセッションでデータを読み込んで、任意のユーザの評価を見てみよう。

```
>>> reload(recommendations)
>>> prefs=recommendations.loadMovieLens()
>>> prefs['87']
{'Birdcage, The (1996)': 4.0, 'E.T. the Extra-Terrestrial (1982)': 3.0,
 'Bananas (1971)': 5.0, 'Sting, The (1973)': 5.0, 'Bad Boys (1995)': 4.0,
 'In the Line of Fire (1993)': 5.0, 'Star Trek: The Wrath of Khan (1982)': 5.0,
 'Speechless (1994)': 4.0, etc...
```

これでユーザベースの推薦を行うことができる。

```
>>> recommendations.getRecommendations(prefs,'87')[0:30]
[(5.0, 'They Made Me a Criminal (1939)'), (5.0, 'Star Kid (1997)'),
 (5.0, 'Santa with Muscles (1996)'), (5.0, 'Saint of Fort Washington (1993)'),
 etc...]
```

あなたのコンピュータのスピードにもよるが、この方法で推薦を行う場合、ちょっと時間がかかるということに気づくだろう。これは大きなデータを対象にしていることが原因である。ユーザベースの推薦は、ユーザ数が増えれば増えるほど時間がかかる。次にアイテムベースの推薦を試してみよう。



```
>>> itemsim=recommendations.calculateSimilarItems(prefs,n=50)
100 / 1664
200 / 1664
etc...
>>> recommendations.getRecommendedItems(prefs,itemsim,'87')[0:30]
[(5.0, "What's Eating Gilbert Grape (1993)"), (5.0, 'Vertigo (1958)'),
 (5.0, 'Usual Suspects, The (1995)'), (5.0, 'Toy Story (1995)'),etc...]
```

アイテムの類似度のディクショナリを作るのに時間はかかるが、いったん作ってしまえば推薦は一瞬で行うことができる。さらに、ユーザ数が増えても推薦にかかる時間は増えない。

このデータセットは、スコアリングの方法が変わることによって、出力がどのように変わるかを試してみる際に非常に役立つ。また、アイテムベースとユーザベースのフィルタリングのパフォーマンスの違いを理解するためにも役立つ。GroupLensのウェブサイトには、他にも書籍、ジョーク、さらなる映画のデータセットなど、試してみると面白いデータがある。

## 2.9 ユーザベースVSアイテムベース

大きなデータセットから、たくさんの推薦を得ようとする際には、アイテムベースのフィルタリングはユーザベースと比較すると非常に高速である。しかし、アイテム類似度テーブルをメンテナンスする必要があるというオーバーヘッドも持っている。また、データセットがどの程度「疎」であるかによって正確性が異なってくる。映画の例では評者たちはほとんどすべての映画に対して評価をしていた。したがってこのデータセットは「密」(疎ではない)であるといえる。一方、del.icio.usではまったく同じデータセットを持った二人の人がいる、ということはありそうもない——多くのブックマークは少人数のグループによって記録されているため、データセットは「疎」なものになっている。アイテムベースのフィルタリングは、疎なデータセットに対しては、一般的にユーザベースのフィルタリングより性能が優れている。しかし、データセットが密であればこの二つの性能は大体同等になる。

これらのアルゴリズムのパフォーマンスの違いについてさらに学びたければ、Sarwar et alによる"Item-based Collaborative Filtering Recommendation Algorithms"という論文をチェックするとよい(http://citeseer.ist.psu.edu/sarwar01itembased.html)。

これまで述べたように、ユーザベースのフィルタリングはよりシンプルで、余分なステップを必要としない。そのため、メモリに収まるサイズで、変更が頻繁に行われるようなデータセットに対しては、こちらの方が適している場合がある。最終的には、ユーザの嗜好が独自の値を持っているようなサイト——ショッピングサイトではなく、リンクを共有するサイトや音楽を推薦するサイト——に向いているだろう。