8장 회귀: 향상된 추천

7장 내용

추천 엔진 만들기

회귀를 이용한 평점 예측

8장 내용

좀 더 발전된 평점 추정기 만들기

개념들 결합은 회귀를 이용

장바구니 분석 ‘X을 구매한 분들은 Y도 구매했습니다’ cf. 수치적 평점

**추천 향상**

추천 시스템 better than 무착위 추측

영화 추천 데이터셋 <= 사용자 x 영화 ; 희소 매트릭스 (평점 매기기)

**추천 이진 매트릭스 사용**

영화 평점을 예측하기 위해서 이진 매트릭스를 사용할 것임.

1 = 평점을 준 영화

0 = 평점을 주지 않은 영화

Obvious-in-hindsight ideas평점을 준 영화만 알아도 사용자에 대해 알 수 있다. 왜냐하면 아무 영화가 아닌 본인이 좋아했던 영화와 관련된 영화를 고르기 때문이다.

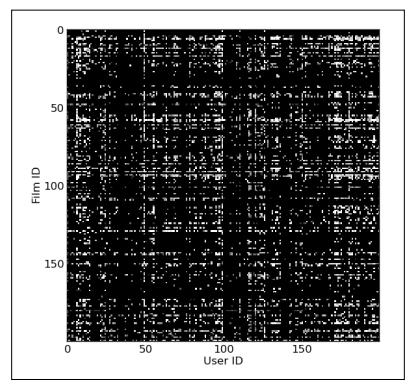
희소 매트릭스로 표현을 하면, 각 평점을 작은 사각형으로 표현한 이미지로 매트릭스의 값을 시각화 가능. (회색= 평점 값, 검정= 평점 없음)

ex. 다른 사람보다 많은 영화에 평점 준 사용자,, 다른 영화보다 많은 평점을 받은 영화

데이터를 시각화하는 코드

**from** matplotlib **import** pyplot **as** plt  
imagedata = reviews[:200, :200].todense()  
plt.imshow(imagedata, interpolation=**'nearest'**)

코드 결과 제시



영화 평점을 예측하기 위한 일반적인 알고리즘.

1. 각 사용자에 대해 가까움의 관점으로 다른 사용자의 순위를 정한다. 이 단계에서 이진 매트릭스를 사용하고 가까움의 척도로서 상관 관계를 사용한다.
2. 사용자-영화 짝에 대해 평점을 추정할 필요가 있을 때, 사용자의 이웃을 연속적으로 본다. 관련된 영화에 대한 평점을 찾을 때, 평점을 보고한다.

코드 구현

*#np.corrcoeff 상관 관계 계산  
#두 사용자의 상관 관계 계산*corr\_between\_user1\_and\_user2 = np.corrcoef(user1, user2)[0,1]  
*#all\_correlations 함수: 사용자와 다른 모든 사용자 사이의 상관 관계 계산위한 함수***import** numpy **as** np  
**def** all\_correlations(bait, target):  
 *'''  
 corrs = all\_correlations(bait, target)  
 corrs[i]는 bait와 target[i]의 상관 관계다.  
 '''* **return** np.array([np.corrcoef(bait, c)[0,1] **for** c **in** target])  
*#사용자의 최근접 이웃을 선택하는 상관 관계***def** estimate(user, rest):  
 *'''  
 'rest’를 기반으로 ‘user’를 위한 영화 평점을 추정한다.  
 '''  
 #사용자 평점의 이진 버전* bu = user > 0  
 *#나머지 평점의 이진 버전* br = rest > 0  
 ws = all\_correlations(bu,br)  
 *#상위 100위 값까지 선택* selected = ws.argsort()[-100:]  
 *#평균을 기반으로 추정* estimates = rest[selected].mean(0)  
 *#일부 영화는 여타 영화보다 평점을 좀 더 갖는 다는 사실을 기반으로 정확한 추정 필요* estimates /= (.1+br[selected].mean(0))

RMSE = 0.2

Root Mean Square Error

가장 직관적이면서도 의미 있는 근사모델의 정확도 평가 방법

임의의 실험점 셋에서의 ‘실제모델과 근사모델의 차’의 평균 제곱합으로써 사용한 실험점의 수가 많으면 많을 수록 더 정확히 근사모델의 정확도를 평가 함

**영화의 이웃 보기**

이번에는 가장 유사한 영화 찾기.

영화에 대한 최근접 이웃을 기반으로 추천.

방법: 사용자 U에 대한 영화 M의 평점을 예측할 때, 시스템은 U가 M과 가장 유사한 영화에게 준 평점을 예측

1. 유사 매트릭스 계산(어떤 영화가 유사한지)
2. 각 사용자-영화 짝에 대한 추정치 계산

코드 작성

movie\_likeness = np.zeros((nmovies,nmovies))

#instead of

#movie\_likeness = np.zeros((movies, nmovies))

#allms = np.ones(nmovies, bool)

#cs = np.zeros(nmovies)

*#모든 영화에 대해 반복***for** i **in** range(nmoives):  
 moive\_likeness[i] = all\_correlations(reviews[:,i], reviews.T)  
 movie\_likeness[i,i] = -1  
*#대각선은 -1로 설정  
#이 매트릭스를 기반으로 평점 측정하는 함수 작성 (최근접 이웃 분류)***def** nn\_movie(movie\_likeness, reviews, uid, mid):  
 likes = movie\_likeness[mid].argsort()  
 *#가장 유사한 영화를 시작으로 하기 위해 역순으로 만듦* likes = likes[::-1]  
 *#가능한 가장 유사한 영화의 평점을 반환함* **for** ell **in** likes:  
 **if** reviews[uid,ell] > 0:  
 **return** reviews[uid,ell]

instead of

def nn\_movie(movie\_likeness, reviews, uid, mid):

likes = movie\_likeness[mid].argsort()

#가장 유사한 영화를 시작으로 하기 위해 역순으로 만듬

likes = likes[::-1]

#가능한 가장 유사한 영화의 평점을 반환함

for ell in likes:

if reviews[u,ell] > 0:

return reviews[u,ell]

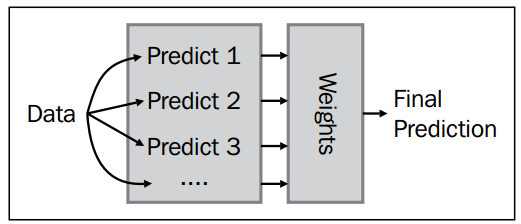
RMSE = 0.85

**여러 기법의 결합**

앙상블 학습(ensemble learning)

데이터 -> [예측기] -> [가중치] -> 최종 예측

예측기의 앙상블, 즉 집합을 학습한 후, 이들을 결합하는 방법. 각 예측을 새로운 속성으로 볼 수 있고, 줄곧 작업하던 훈련 데이터를 기반으로 속성들을 결합함. 여기에서는 예측기를 학습시킨 훈련 데이터를 다시 사용함.



어떤 아이디어가 좋고 나쁜 지를 가중치를 부과하면 더 나은 결과를 얻을 수 있다.

*#이전 예제에 사용했던 코드 import***import** similar\_movie  
**import** corrneighbors  
**import** usermodel  
**from** sklearn.linear\_model **import** LinearRegression  
reg = LinearRegression()  
  
es = [  
 usermodel.estimate\_all()  
 corrneighbors.estimate\_all(),  
 similar\_movie.estimate\_all()  
]  
coefficients = []  
*#단일 잔류(leave-1-out) 교차 검증 반복 시행***for** u **in** xrange(reviews.shape[0]): *#모든 사용자의 ids* es0 = np.delete(es,u,1) *#사용자 u를 제외한 모든 사용자* r0 = np.delete(reviews, u, 0)  
 P0,P1 = np.where(r0 > 0) *#실제 예측에 관심* 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) *#이전처럼 오차 구함*

RMSE = 1

#예측기 작동 여부를 알기 위해 coefficients 변수 분석

**print** coefficients.mean(0)   
*#모든 사용자에 대한 평균값  
#결과 [0.25164062, 0.01258986, 0.60827019]  
#similar\_movie에 대한 가중치 가장 높음  
#최종 결과 영향x, correlation-based 기법 제거 가능*

응용

하나의 (최대 유사 영화가 좋은) 예측기를 5개의 최대 유사 영화를 학습 과정에 사용해보자.

K번째의 최대 유사 영화를 생성하기 위해 이전 코드 채택 가능

-> 계층Stacked 학습기 사용

es = [  
 usermodel.estimate\_all()  
 similar\_movie.estimate\_all(k=1),  
 similar\_movie.estimate\_all(k=2),  
 similar\_movie.estimate\_all(k=3),  
 similar\_movie.estimate\_all(k=4),  
 similar\_movie.estimate\_all(k=5)  
]  
*#나머지는 이전 코드와 같다는 데 이건가….*coefficients = []  
**for** u **in** xrange(reviews.shape[0]): *#모든 사용자의 ids* es0 = np.delete(es,u,1) *#사용자 u를 제외한 모든 사용자* r0 = np.delete(reviews, u, 0)  
 P0,P1 = np.where(r0 > 0) *#실제 예측에 관심* 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)  
**print** coefficients.mean(0)

응용의 결과 -> 최종 결과를 보면 더 나아지지 않았다고 한다.

**장바구니 분석**

지금까지의 기법: 사용자가 제품을 얼마만큼 좋아하는지를 나타내는 수치적 평점 필요

장바구니 기법: 추천을 학습하는 대안.

ex. 제품을 구매한 소비자가 다른 어떠한 제품을 구매하였는가

ex. 이메일을 쓸 때 사용자에게 추가적인 수신자 추천

ex. 브라우징 히스토리를 기반으로 방문할 웹 페이지 추천하는 어플

ex. 맥주와 기저귀(믿음직한 정보는 없다함;;)



**유용한 예측 얻기**

장바구니 기법의 경우, 자주 언급되는 물품을 말해주고 개인화가 되어있지 않고 그저 인기 있는 것을 추천하므로 실제로도 예측과 같은 결과를 얻을 수 없다.

ex. 세제를 구입한 사람 중의 50%가 빵을 구입할 수 있지만 빵은 다른 물품이랑도 자주 함께 구매되기에 꼭 세제를 구매하면 빵도 구입한다고 말하기 힘듦.

**슈퍼마켓 쇼핑 장바구니 분석**

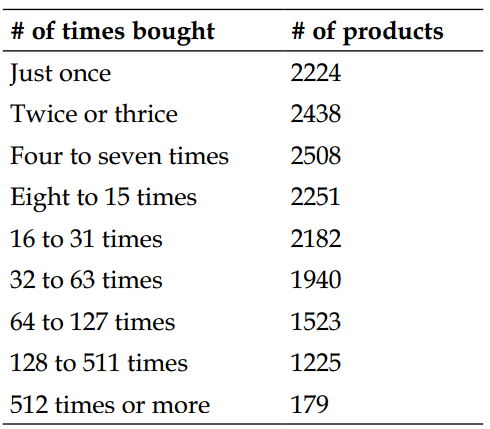
데이터 셋: retail.dat “벨기에 슈퍼마켓의 거래내용: 제품 번호 & 장바구니 집합 번호”

**from** collections **import** defaultdict  
**from** itertools **import** chain  
dataset = [[int(tok) **for** tok **in** line.strip().split()] **for** line **in** open(**'retail.dat'**)]  
counts = defaultdict(int)  
**for** elem **in** chain(\*dataset):  
 counts[elem] += 1

#이거 대신 쓰라는데 오류;;

#dataset = [[int(tok) for tok in line.strip().split()] for line in GzipFile('retail.dat.gz')]

히스토그램 결과



장바구니 분석 알고리즘 => 아프리오리Apriori 알고리즘

방법 => 입력(집합 모음; 장바구니), 반환값(매우 빈도수가 높은 집합;많이 포함된 물품)

상향식 접근법에 따라 작동.

가장 작은 후보로 시작하여, 한번에 한 원소가 추가되기에 최소 support의 정의 필요

support는 함께 구매되는 물품 집합의 횟수로, 최소 support보다 큰 집합의 원소로 하나의 집합이 구성이 됨.

minsupport = 80

valid = set(k **for** k,v **in** counts.items() **if** (v >= minsupport))

itemsets = [frozenset([v]) **for** v **in** valid]  
new\_itemsets = []  
**for** iset **in** itemsets:  
 **for** v **in** valid:  
 **if** v **not in** iset:  
 *#추가된 v와 이전과 같은 새로운 가능한 집합을 만듬* newset = = (iset | set([v]))  
 *#새로운 집합이 나타나는 횟수를 세기위해 데이터집합에 반복  
 #이 단계는 느리고 적당한 구현이 아니라고 함* c\_newset = 0  
 **for** d **in** dataset:  
 **if** d.issuperset(newset):  
 c\_newset += 1  
 **if** c\_newset > minsupport:  
 new\_newsets.append(newset)

문제점:

정확하지만 매우 느림.

모든 데이터셋에 대해 반복하는 일을 피할 수 있는 하부 구조가 필요.

**연관 룰 마이닝**

빈도 아이템 집합은 그닥 유용하지 않기에 연관 룰association rule을 만드는 작업 필요하게 됨.

연관 룰은 ‘만약 X라면 Y이다’ 형태의 표현으로 반드시 그렇다는 의미는 아니고 확률적으로 그럴 가능성이 높음을 의미함(X을 샀다고 반드시 Y를 항상 구매하지 않음).

빈도 집합에서 X의 모든 가능한 조합은 Y를 시사하는 룰을 얻을 수 있다.

lift = 룰을 적용해 얻은 확률과 기준선 사이의 비율

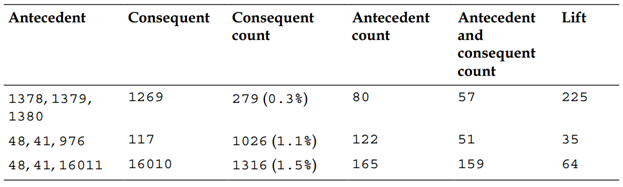
P(Y) = Y를 포함한 모든 트랜잭션의 일부

P(Y|X) = Y와 X 모두를 포함한 트랜잭션의 일부

ex. 베스트 셀러의 경우 lift 값이 1에 가까워짐. 그닥 좋지 않다고 생각된다고 함.

**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  
 *#선례의 카운트* account = 0.0  
 *#결과의 카운트* ccount = 0.0  
  
 **for** d **in** dataset:  
 **if** item **in** d: base += 1  
 **if** d.issuperset(itemset): ccount += 1  
 **if** d.issuperset(antecendent): account += 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))

일부 결과



카운트 = 트랜잭션의 수

결과(제품을 구매한 기준 비율), 선례의 모든 물품, 선례와 결과의 모든 물품

80개의 트랜잭션에서 1378, 1379, 1380이 함께 구매. 57개에는 1269 포함. 그래서 추정된 조건 확률은 아래와 같음.

1269를 포함한 트랜잭션이 0.3%임을 비교해 보면 리프트는 225가 됨.

71/.3=236.7…..????

**좀 더 발전된 장바구니 분석**

pymining. 아프리오리보다 빠른 장바구니 분석 알고리즘.

<https://github.com/bartdag/pymining>