이상형 추천기 구현

2011132021 김동은

2012191011 송민규

2011272097 장도영

(6조)

[plantarflex@naver.com](mailto:plantarflex@naver.com)

**Essential Recommending Platform: Seven44**

우리가 정한 기본적인 input 자료형은 1:3 relation으로, user instance 하나가 picture intstance 3개를 가리키는 꼴과 같다. 이 모든 것을 포함하고 있는 플랫폼이 바로 Seven44 코드이며, 이것을 바탕으로 User-based recommenmding algorithm과 item-based filtering을 구현할 수 있다. 먼저 User-based한 알고리즘은 다음과 같은 방법으로 유사성을 측정한다.

주어진 사진 P1을 기준으로 P1과 같이 묶일 수 있는 다른 사진들을 찾는다고 하자. 먼저 P1을 고른, 즉 P1이라는 picture instance를 가리키고 있는 user instance를 모두 찾는다. 여기서는 u1과 u2가 될 것이다. 이제 이 user instance들이 가리키는 다른 모든 picture instance들을 picture domain에서 검색하되, 여러 user instance들에게 지목을 받고 있는 picture instance에게는 높은 점수를 부여한다. 위 상황에서는 P2,P3,P5가 P1과 user-based한 유사성을 갖고 있는 셈이며 그 중 P2가 가장 높은 유사도를 갖는다고 이야기할 수 있겠다. 위와 같은 방법으로 가장 높은 유사도를 가지는 picture instance 5개가 본래 기준으로 둔 picture인 P2와 “친척” 이라고 할 수 있겠다. 첨부된 Seven44 플랫폼 알고리즘은 이렇게 “친척”이 되는 picture instance 의 index 값을 boolean array로 갖는 HerRelatives 배열로써 반환한다.

Seven44 플랫폼에 포함된 class 와 메소드는 다음과 같다.

① User class

위에서 설명했던 대로 이 클래스는 설문조사 결과마다 인스턴스화된다. 한 user instance는 서로다른 3개의 picture instance를 가리킨다. 지금부터 진행될 모든 이야기의 중심 뼈대가 될 instance들이다. MaxUserNum 의 길이만큼 인스턴스 어레이로 선언되어있다.

② Picture class

접근을 쉽게 하기 위해 index화되었으며, user instance와 마찬가지로 MaxPicNum의 길이만큼 인스턴스 어레이로 선언되어있다. 이후 item-based filtering을 할 시에 picture 하나하나에 대응되어야 할 벡터들도 이 class에서 초기화된다.

③ Greenlight class

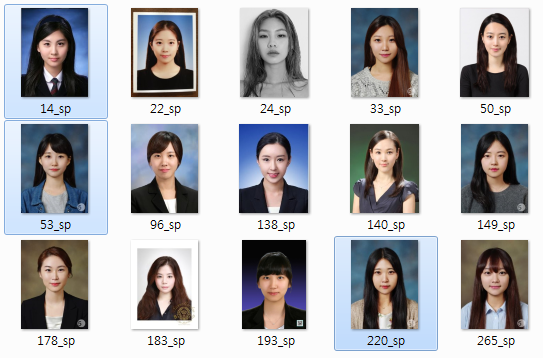
특정 picture instance인 Picture X를 지목한 user instance를 추적하기 위한 클래스이다. SearchUser 메소드의 인자로 그 picture을 대입할 시 picture을 지목한 user instance의 index가 boolean array 형태로 반환된다.

④ Relative class

Picture X 의 “친척” picture instance를 찾아주기 위한 클래스이다. 위 Greenlight class 에서 찾은 user instance index를 이용해 SearchRelatives 메소드는 모든 picture instance의 Picture X와의 유사도를 계산하여 Relativity 배열에 저장한다. 이 Relativity 배열에서 가장 높은 5개의 값을 갖는 picture들은 boolean array 인 HerRelatives 배열에 저장되게 된다.

무엇보다 Relative class에서 가장 중요한 기능은 item-based filtering 기능인데, 특정 picture가 우사용자가 선택한 picture의 item-based한 특징에서 멀리 벗어나면 Relativity array 값을 0으로 만들어준다는 것에서 의미를 가진다.

한정된 자료형으로는 위와 같은 user-based 유사성이 의미가 없기 때문에 더 많은 user와 더 많은 picture을 필요로 한다. 우리는 Google에서 298개의 여성 증명사진을 다운로드하여 150여명과의 설문조사를 통해 450여개의 자료형을 만들어냈다. 설문조사 과정은 다음과 같다.



인간의 인지 능력이 300여개의 사진을 모두 훑고 그 중 사진 3개를 뽑는 것은 이치에 맞지 않을 뿐더러, 시간상으로도 너무 많은 자원을 허비하게된다. 따라서 우리는 Survey 프로그램을 구현하여 우리가 준비한 300여개의 사진 중 임의로 15장의 사진을 외부 폴더로 추출하여 각각 1회의 설문조사를 실시하기로 하였다. 구현한 survey 프로그램은 appendix에 덧붙여 놓았다.

이제 추출된 15장의 사진 가운데 설문자에게 가장 마음에 드는 이성 3명을 고르라 한 다음, 이를 하나하나 user instance가 picture instance를 가리키는 상태로 초기화하도록 한다. 이 모든 과정은 Seven44에 하드코딩되어있다.

지금까지 만들어진 설문을 플랫폼에 장착한 후, 사용자가 원하는 사진의 번호를 main class에 입력하면 그 사진의 “친척” 5명의 사진 번호가 아웃풋 콘솔로 출력되며, 외부 폴더로 5명의 사진이 복사된다. 물론 새로운 사진 번호를 입력하게 되면 기존 사진은 삭제되고 새로운 친척 5명이 복사된다. 프로그램이 잘 동작할 시, 다음과 같은 콘솔 아웃풋이 나오게 된다.

**Platform assiatant: ImageManager**

프로젝트 성격상 이미지 파일을 입출력하는 일이 많으므로, 이러한 작업을 수행하는 method를 모두 모아서 하나의 class로 모았다. class 이름은 ImageManager이다.

ImageManager의 큰 method는 두 개가 있다.

① resizeJPGImage(int MaxWidth, int MaxHeight) : GUI 형태로 만들기 위해 이미지 크기를 전부 맞추는 것이 좋은데, 그 전처리 작업에 필요한 method이다.

② saveRecommendedItem() : Seven44에서 결정한 HerRelatives 배열을 받아 그에 해당하는 사진파일을 output한다.

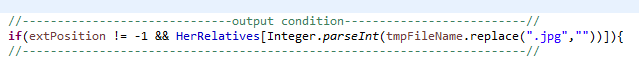
① resizeJPGImage(int MaxWidth, int MaxHeight)

디렉토리에서 모든 파일을 가져온 뒤, 이중에 jpg 파일만 골라 비율을 유지하며 축소하는 method이다. 축소하는 정도를 인수로 받는다(MaxWidth, MaxHeight).

② saveRecommendedItem()

resizeJPGImage method와 마찬가지로 디렉토리에 모든 파일을 가져온 뒤, 이중 추천대상으로 골라진 파일만 골라서 output 디렉토리에 복사한다.

추천대상 사진을 고르는 데는 Seven44에서 결정한 HerRelatives[]의 값이 기준이 된다. HerRelatives의 원소 중 true값에 해당하는 사진만 output 폴더에 복사하는 것이다.



위 부분의 코드가 필터링 역할을 하는 곳이다. extPosition은 해당 파일이 jpg 파일인지 나타내 주는 것이다. jpg파일이 아닐 경우 extPosition이 –1 값을 갖게 되므로 필터링된다.

tmpFileName은 파일의 이름인데, 여기서 “.jpg”를 “”로 replace하면 tmpFileName은 숫자로만 이루어진 string 이 된다. (미리 사진 파일명을 전부 숫자로 해놓았다.) 그것을 int로 변환한 것을 HerRelatives의 index로 입력해서, true값을 반환하면 그 사진을 output 폴더에 저장하는 것이다.

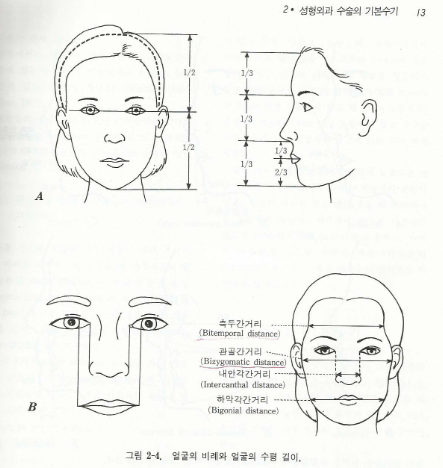
한편, 위와 같이 user-based한 recommendation으로는 이상형 추천과는 관계없는 사람이 “친척”으로 출력되어, 다음 선택을 하는 데에 방해를 할 수 있다. 즉, 우리의 설문조사 방식에는 설문자의 이상형과는 관계없이 선택될, 아웃라이어 급으로 아름다운 여성들이 있었다. 이를테면, 한예슬은 다른 어떤 두 명의 여성과도 함께 선택되어지는 경향성을 보였는데, 이것은 한예슬이 만인의 이상형이라기보다는 그저 절대적으로 아름답기 때문에 설문조사 결과에 반영된 것으로 보아야 한다. 우리는 이것을 극복하기 위해 item-based filtering을 적용시키기로 하였다. 즉, 비록 한예슬이 설문조사 당시 세 명의 picture instance에 포함되더라도, 눈코입과 다른 정량적인 변수들이 그 “친척” instance들과 유사하지 않을 경우엔 친척의 범위에 포함시키지 않는 방법을 만들어냈다. 이를 위해서 OpenCV를 활용한 얼굴 데이터 분석을 실시하였다.

**Facial Data analysis using OpenCV**

Opencv는 오픈 소스 컴퓨터 비전 C 라이브러리이다. 원래는 인텔이 개발하였다. 윈도, 리눅스 등의 여러 플랫폼에서 사용할 수 있다. 실시간 이미지 프로세싱에 중점을 둔 라이브러리이다.

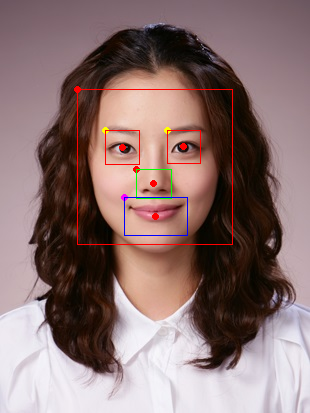
이번 프로젝트에서 opencv를 이용한 목적은 Item based 분석을 위한 얼굴 성분을 추출하기 위함 이다. 코드는 C++로 했으며 VISUAL Studio 2010을 사용했다. Opencv 버전은 2.4.10을 사용했다.

얼굴 인식 알고리즘은 HAAR Wavelet 을 이용하였다.



위의 정보는 성형외과학에 기초한 이상적인 얼굴의 비율 측정이다. 위쪽 사진은 이마 위에서부터 턱까지의 길이의 비율을 측정하고 있으며 이러한 비율을 이용하여 얼굴 성분을 추출 했다. 아랫쪽 사진은 얼굴의 가로성분을 이용한 비율 측정인데, 우리는 여기서 내안각간거리(미간거리)를 활용하였다.

Opencv에서 얼굴 인식이 되어 나온 정보를 가지고 사용할 수 있는 정보는 얼굴의 길이, 눈 위치, 코 위치, 입 위치가 가장 대표적이며 이것을 이용 하기 위해선 왼쪽의 비율을 100% 같게 못하기 때문에 4단계로 나누어 성분을 추출하여 상대적인 비율을 비교한 것을 이용하였다. 성공적으로 프로그램이 실행된다면 다음과 같은 성분들이 측정된다.



성분 1. (얼굴 위 ~ 눈 중심) / 얼굴 높이

성분 2. 눈 중심 ~ 코 중심 / 얼굴 높이

성분 3. 코 중심 ~ 입 중심 / 얼굴 높이

성분 4. 입 중심 ~ 턱 밑 / 얼굴 높이

성분 5. 코 가로 길이 / 눈 사이 간격

성분 6. 입 가로 길이 / 눈 사이 간격

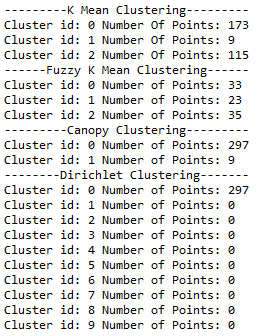
성분들의 값은 프로젝트 폴더 안에 저장 된다. 파일이 누적되어 저장되게 하였으므로 result.txt 파일을 삭제 하지 않는다면 결과 값이 추가로 저장 되게 된다. OpenCV 코드는 appendix에 첨부해 놓았다.

**Clustering with Mahout**

Open CV에서 얻은 6개의 데이터를 분석하기 위해 Mahout을 사용했다. 사진 하나당 6개의 데이터가 나오므로, 이것을 하나의 6차원 벡터로 변환하고, 298개의 사진을 Clustering 하는 방법이다.

Mahout 책에 소개된 Clustering 방법은 4개로, K-mean Clustering, Fuzzy K-mean Clustering, Canopy Clustering, Dirichlet Clustering이 있다. K-mean과 Fuzzy K-mean Clustering 방법은 사용자가 처음에 Cluster의 개수를 정해줘야 하는 특징이 있다. 따라서 정확한 Cluster의 개수를 예측하고 처음에 집어넣어 줘야 하는데, 주로 정확한 Cluster의 개수를 얻기 위해 Canopy Clustering 방법을 사용한다. 애초에 벡터가 6차원이기 때문에 시각화를 시키는 것이 불가능하기 때문에 Canopy Clustering에 의존할 수밖에 없다.

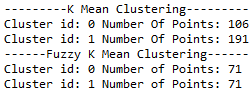
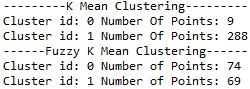
아래는 그 결과이다.



K-mean과 Fuzzy K-mean Clustering에서 처음에 Cluster 개수를 3개로 지정해준 결과이다. 이 결과 Canopy Clustering에서 Cluster가 2개로 잡아야 한다는 것을 알려주었고, Dirichlet Clustering은 애초에 하나의 Cluster가 있다고 말한다.

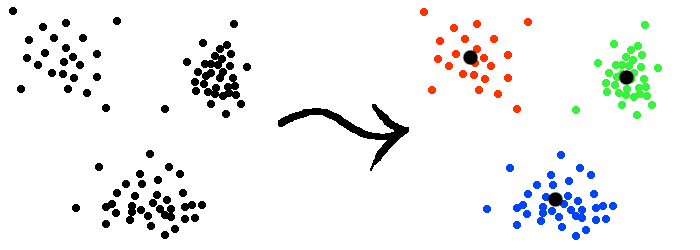
참고로 여기서 총 사진은 298개이지만, open CV 의 코드에서 오류가 나 297개의 데이터를 받았었다. 하지만 Canopy Clustering의 결과를 보면 첫 번째 Cluster와 두 번째 Cluster의 원소의 개수의 합이 304이다. 이는 Clustering이 정확하게 점들을 분할하지 못하고, 하나의 Cluster 안에 또 하나의 Cluster가 들어가 있는 형태라고 생각할 수 있을 것이다.

어찌 됐든 Canopy Clustering이 Cluster가 두 개라는 결론을 내려주므로, K-mean과 Fuzzy K-Mean Clustering을 Cluster 개수가 2개라고 설정하고 코드를 실행 해 보았다.

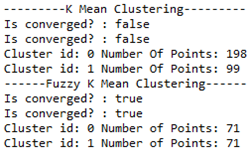
단 두 번만 실행했는데도, 위와 같이 전혀 일관되지 않은 결과가 나온다.

여기서 K-mean Clustering의 방법을 알 필요가 있다.



K-mean Clustering[[1]](#footnote-1)

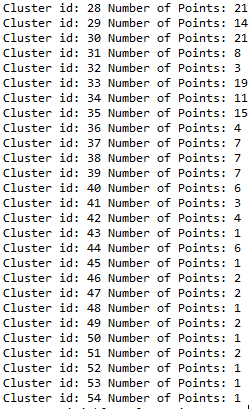
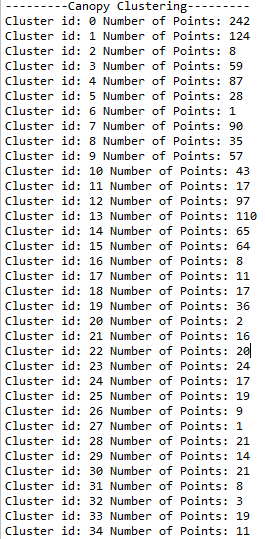
K-mean Clustering 방법은 실행될 때 Cluster의 개수를 인수로 받는다. 예를 들어 위 그림에서 3을 인수로 받았다면, 위와 같은 분포에 random으로 3개의 점을 잡는다. 그리고 각 각의 점 주위의 점들의 정보를 얻어 Cluster의 중심을 새로 잡고, 그 다음에 새로운 중심 근처의 점들을 정보를 모아 Cluster의 중심을 새로 잡는 식이다. 이러한 방식이기 때문에 처음에 인수로 받는 Cluster의 개수가 매우 중요하고, center 값이 수렴하는지의 여부도 매우 중요하다.



Clustering Class의 method 중 하나인 isConverged()을 통해 간단히 center가 수렴하는지 알아 볼 수 있다. 위와 같은 결과가 나오는데, K-mean Clustering은 Cluster를 2개 잡을 경우 몇 번을 수행해도 절대 수렴하지 않는다. Fuzzy k-mean의 경우 수렴하지만, 한정된 점들만을 Clustering하기 때문에 우리가 원하는 대로 적용하기는 힘들다.

지금까지의 방법은 점들 간의 거리를 Euclidean 방법(root mean square)으로 측정한 결과를 가지고 분석한 것이다. 실제로 거리를 재는 방법은 4가지가 더 있는데, Squared Euclidean 측정법, Manhattan 측정법, Tanimoto 측정법, Cosine 측정법이 그것이다. 이 중에서 cosine 측정법은 벡터 간 원점으로부터 각도 거리를 측정하는 것이기 때문에 이 측정법에는 의미가 없다고 판단했다. 나머지 세 개의 측정법을 모두 적용해서 해봐도 K-mean Clustering에서 일관된 값이 나오지 않았다.

지금까지는 점들에 가중치를 부여하지 않고 분석했다. 실제로 이런 것이 문제가 될까 해서, 각 차원마다 평균을 0으로, 표준편차를 1로 조정해서 분석을 해보았다.



Canopy Clustering 방법에 따르면 총 55개의 Cluster가 나온다. 하지만 이는 신뢰할 수 없는데, 왜냐하면 우리가 점들의 표준편차를 어느 정도로 조정하느냐에 따라 Cluster의 개수가 천차만별로 나오기 때문이다. 표준편차를 0.5로 조정할 경우 16개의 Cluster가 있다는 결과가 나오고, 심지어 표준편차를 10으로 조정할 경우 280개의 Cluster, 즉 대부분의 점이 그냥 하나의 Cluster가 되어버리는 결과가 나온다. 우리가 표준편차, 즉 가중치를 어느 정도로 주어야 하는지 전혀 알 수가 없기 때문에, 결국 지금까지 한 Clustering이 정말 의미가 있는지에 의문을 품게 되었다.

지금까지의 결과를 따져보면,

① Clustering 방법

② 점들 간의 거리 측정법

③ 가중치(표준편차) 조절

의 방법을 다르게 해도 만족할 만한 결과를 얻지 못하였다.

아예 Clustering이 불가능한 데이터 분포라고 추측할 수 있다.

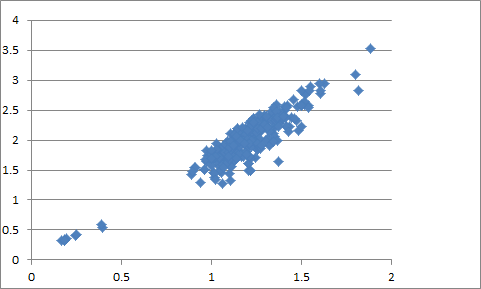
**Filtering by distance**

우리가 얻은 데이터는 6차원이기 때문에 시각화가 불가능하다. 시각화가 불가능하기 때문에 분포를 눈으로 확인할 수 없고, 그러기 때문에 이러한 난항을 겪는다고 말할 수 있다.

사실 open CV를 통해 6개의 데이터를 뽑아 낼 때를 생각해보면 6개의 데이터가 모두 동등한 가중치를 갖는 것이 아닌 것을 확인할 수 있다. 벡터가 Vector(x1, x2, x3, x4, x5, x6)으로 구성되어 있다면, 처음 x1부터 x4까지의 값은 얼굴의 세로 비율, x5와 x6은 얼굴의 가로 비율에 관한 정보이다. 상식적으로도 앞의 네 개와 뒤의 두 개의 가중치가 같을 수 없다. 우리가 각각의 요소에 어느 정도의 가중치를 주어야 할지 알 수 없지만, 앞의 네 개와 뒤의 두 개를 따로따로 분석해야 할 필요성은 있다고 할 수 있다.

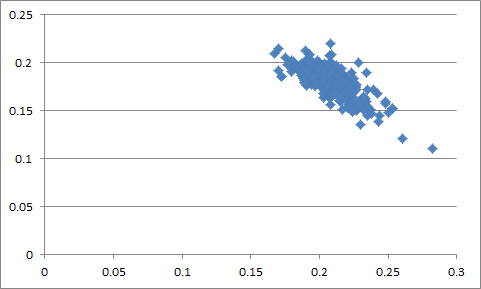
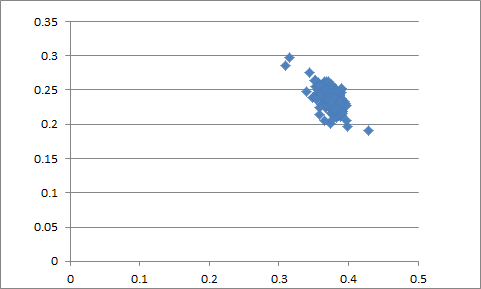
이 방법의 장점은 뒤의 두 개를 분석할 경우 2차원이기 때문에 시각화가 가능하다는 점이다.

일단은 뒤의 두 요소로만 2차원 점을 얻어서 분포를 그려보았다.

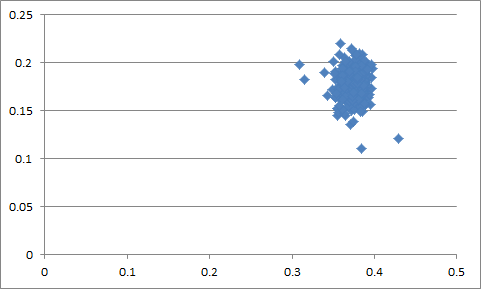
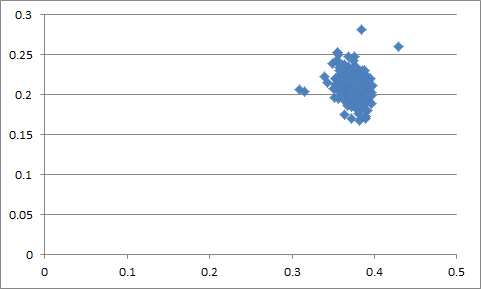


x5 vs x6 distribution

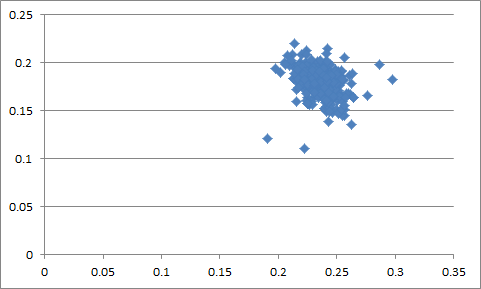
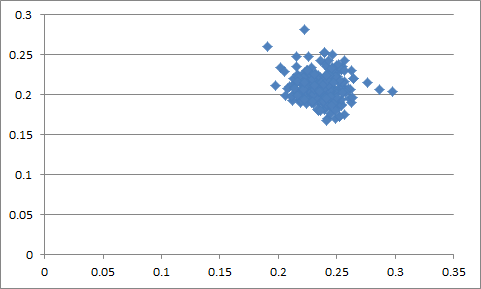
x1, x2, x3, x4는 4차원이지만 여기서 두 개씩만 골라 간접적으로나마 분포를 확인해보았다.



**x1 vs x2 x3 vs x4**

****

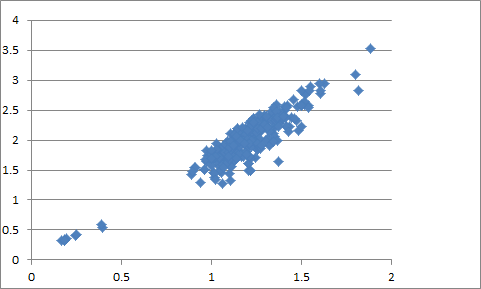
**x1 vs x3 x1 vs x4**

****

**x2 vs x3 x2 vs x4**

결국 이렇게 간접적으로 시각화를 해서 분포를 관찰한 결과, 데이터가 Clustering 될 여지가 없다는 것을 추측할 수 있다. 그렇기 때문에 앞의 Mahout을 통한 Clustering 분석도 잘 안됐을 수 있는 것이다.

Clustering을 할 수 없으므로 데이터의 직접적인 분류는 불가능하지만, 점들 간의 거리 분석의 통해 다른 방법을 모색할 수 있다.

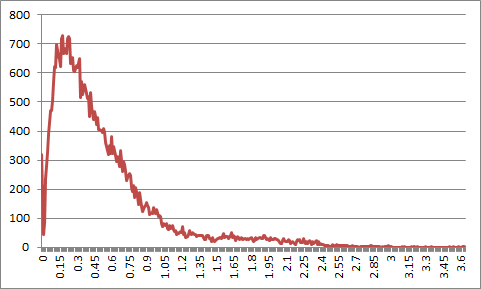


**x5 vs x6 distribution**

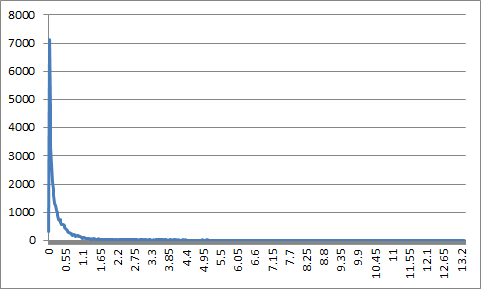
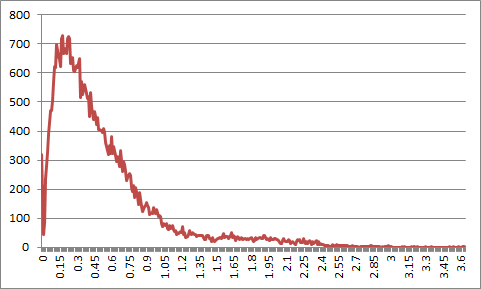
앞의 x5 vs x6 분포를 예로 들면, 사용자가 오른쪽 상단에 위치한 점을 마음에 든다고 선택했다면 왼쪽 하단에 위치한 점을 추천하는 일은 없어야 할 것이다. 이러한 논리를 확장해서 일정 거리 이상 떨어져 있는 점은 추천대상에서 제외하는 ‘Filtering’을 구현할 수 있다.

문제는 어느 정도까지 Filtering을 강하게 적용시켜야 하냐는 점이다. 예를 들어 전과 같이 사용자가 오른쪽 상단에 위치한 점을 선택했다면, 중간에 위치한 점은 제외 대상에 포함시켜야 할까? 어느 정도 멀리 떨어져 있는 점까지 Filtering해야 할까?

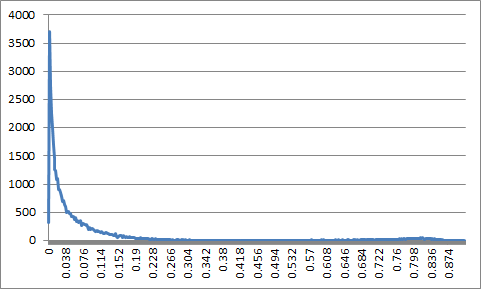
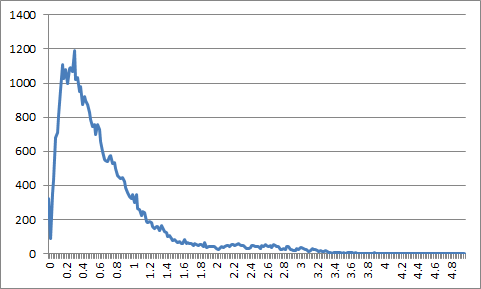
단순하게 생각하기로는 점들 간의 거리를 모두 잰 뒤, 그 평균 이상의 거리를 갖는 점들 간에 Filtering을 하는 것이다. 이를 확인하기 위에 직접 분포를 눈으로 확인해 보았다.



가로축은 점들 간의 거리, 세로축은 그 값에 해당하는 ‘거리의 개수’이다. 0.2정도 떨어져 있는 점들이 가장 많다는 것을 확인할 수 있다. 지금 이 분포는 Euclidean 측정법으로 얻은 거리 분포를 나타낸 것인데, 다른 방법 – squared Euclidean, Manhattan, Tanimoto 로도 거리를 재서 분포를 나타내 보았다.

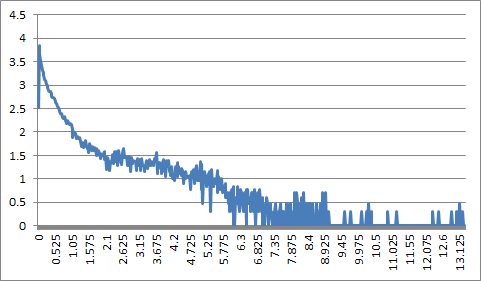
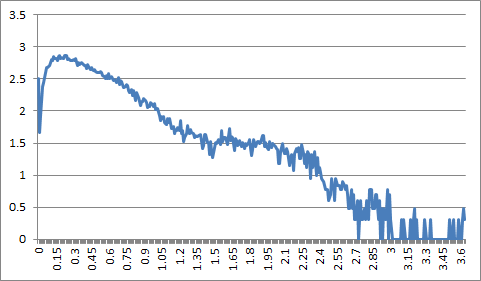


**Euclidean Squared Euclidean**

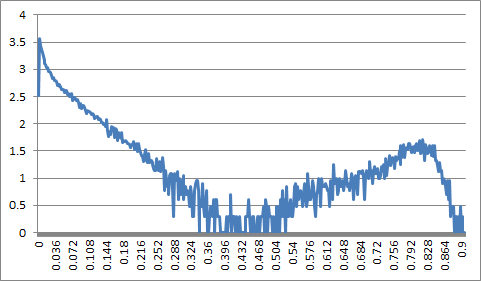
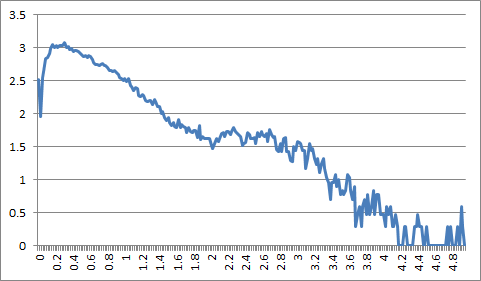
****

**Manhattan Tanimoto**

자연계에 나타나는 분포는 멱급수 분포를 제외하고는 대부분 지수함수적인 요소를 가지고 있다. 따라서 로그를 취하면 근사적으로 직선 형태가 되어 더욱 분포의 특성을 보기 쉬워진다. 이러한 생각으로 로그를 취해 보았더니 다음과 같은 결과가 나왔다.



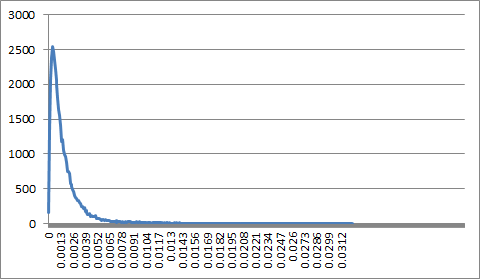
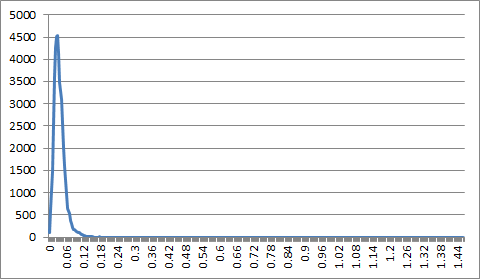
**Euclidean Squared Euclidean**

****

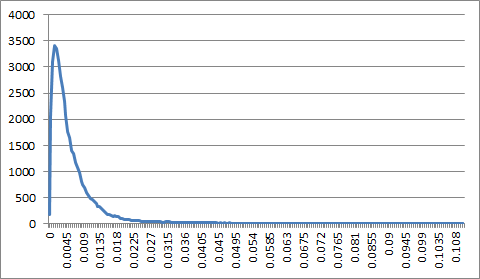
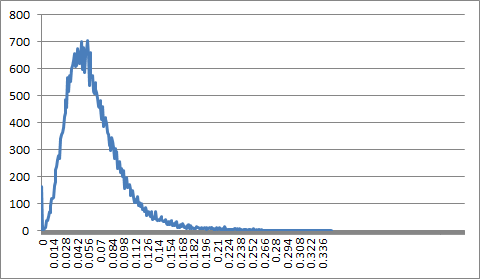
**Manhattan Tanimoto**

Tanimoto 측정법으로 거리를 잰 경우, 흥미롭게도 명확하게 두 구간이 나누어지는 것을 확인할 수 있었다. 따라서 두 점의 거리를 Tanimoto 측정법으로 잰 후, 저 중간에 0이 되는 값(0.448) 보다 그 거리가 작으면 ‘가깝다’, 거리가 0,448보다 크면 ‘멀다’고 생각할 수 있다. 우연찮게도 Filtering의 기준을 명확하게 구한 것이다.

다만 아쉬운 점은 4차원 점 (x1, x2, x3, x4)으로 거리를 재서 분포를 살펴볼 때는 일반적인 푸아송 분포가 나온다는 것이다. 그래프는 다음과 같다.



**Euclidean Squared Euclidean**

****

**Manhattan Tanimoto**

이 경우 로그를 취해도 별다른 특징이 나타나지 않았다. 그나마 가장 고른 분포가 나타나는 Manhattan 측정법을 선택하여, 이 평균값인 0.048을 Filtering의 기준으로 삼기로 하였다.

정리하자면 다음과 같다.

Open CV를 통해 얻은 6개의 데이터에서 앞의 4개와 뒤의 2개 데이터를 분리하여 4차원 점과 2차원 점을 각각 얻는다. 이와 같은 실행을 298개의 사진에 모두 적용하여 298개의 4차원 점들과 298개의 2차원 점들을 얻는다.

서비스 사용자가 어느 한 사진 A를 선택했다면, 일단 사용자 기반 알고리즘으로 사진들이 추천 될 것이다. 만약 추천된 사진들이

① A의 4차원 점에서 (Manhattan 측정법 기준으로) 0.048 보다 멀리 있거나,

② A의 2차원 점에서 (Tanimoto 측정법 기준으로) 0.448 보다 멀리 있으면,

추천 대상에서 제외한다.

이제 이렇게 설정된 filtering 방법을 Seven44 플랫폼의 Relatives 클래스에 옮겨놓기만 하면 된다.

1. 이미지 출처 : http://perception.csl.illinois.edu/gpca/introduction/ [↑](#footnote-ref-1)