붙임3

연구논문/작품 최종보고서

2018 학년도 제 2학기

제목 : OpenCV를 활용한 k-means clustering 기반의 포스터 색감 분석 기법 및 추천 시스템

김태홍(2012310132)

2018년 11월 6일

지도교수: 김 응 모 서명

계획(10)	주제(20)	개념(20)	상세(30)	보고서(20)	총점(100)
10	20	19	30	18	97

■ 요약

본 연구는 k-means clustering 기반의 영화 포스터 색감 분석 기법을 제안하고 이를 활용하여 포스터의 대표색 간의 색상 차를 구하는 방법을 제안한다. 또한 해당 색상 차를 이용하여, 한 영화와 이와 유사한 대표색을 가지는 영화를 추천해주는 시스템을 제안한다. 본 연구는 다음과 같은 가정을 기반으로 한다. 첫 번째, 포스터는 한 장으로 영화를 가장 잘 나타내는 이미지이다. 두 번째, 유사한 색상을 가지는 영화는 유사한 분위기를 가지는 영화이다. 이를 가정하여 3단계로 나누어 연구를 진행한다. 우선 k-means clustering 기법을 통하여 데이터를 전 처리하여 영화별 대표색을 선정한다. 다음은 선정한 대표 색들 중 유사율이 높은 영화를 같은 장르로 묶을 수 있음을 확인한다. 마지막으로 선정한 영화와 유사율이 높은 영화를 추천한다. 본 연구 내용이 영화를 추천하는 과정에서 반영된다면 추천 시스템의 정확도와 사용자 만족도 향상에 기여할 것으로 기대된다.

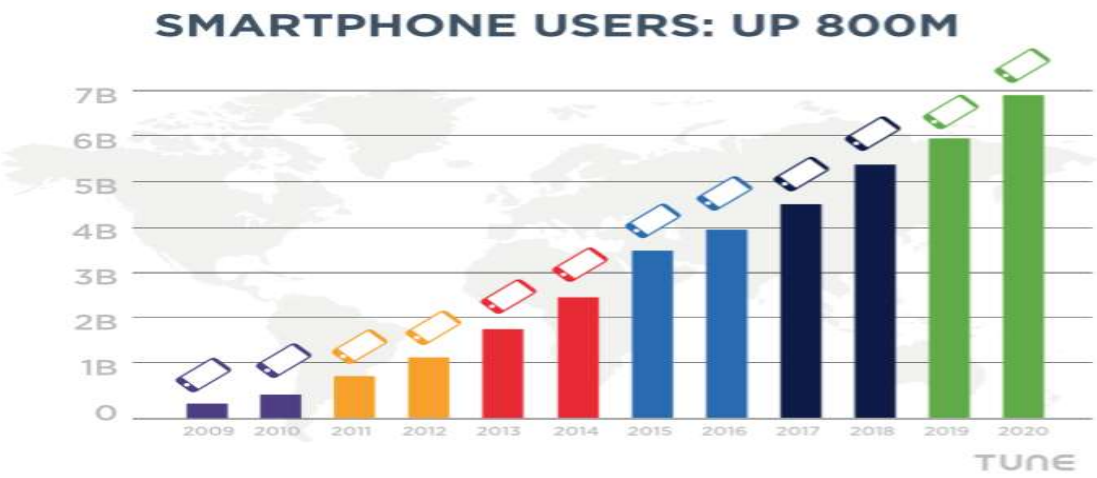
1. 서론

1.1. 제안배경 및 필요성

불과 수십 년 전만에도 우리는 데이터 수집에 많은 시간과 비용이 들었다. 하지만 현재에는 기술의 발전으로 이제는 방대한량의 데이터가 수집되고, 이를 빅 데이터라 표현하고 있다. 말 그대로 기존의 데이터베이스의 관리 능력을 넘은 엄청난 량의 데이터이다. 이러한 상황에서 우리는 이제 수집뿐만 아니라 이를 분석 및 변환하는 방법의 기술 또한 매우 발전하여 다양한 형태의 데이터도 이용할 수 있게 되었으며 그 처리 또한 매우 빠르게 가능하다.

그리하여 최근에는 빠르게 발전해온 영상처리 기술을 이용한 데이터처리에 관심이 증가하고 있다. 최초의 영상 정보라 할 수 있는, 사진이 등장한 이후 120여 년 동안은 아무도 영상에서 정보를 추출하려는 노력을 하지 않았다. 하지만 디지털 컴퓨터의 등장과 함께 상황은 급변했다. 사람들은 영상에서 정보를 얻으려는 노력을 꾸준히 진행해 왔으며 이에 영상 처리 기술이 발전하였고, 최근에는 사람 시각에 필적하는 기계장치를 만드는 컴퓨터 비전이라 일컫는 기술 분야 또한 생겨났다. 이처럼 영상처리 기술의 발달을 통하여 응용분야 또한 오락, 교통 보안 등 그 적용 분야는 나날이 증가하고 있다. 이 때 영상처리 기술이란, 이미지로 입출력되는 모든 형태의 영상을 처리하는 기술을 의미하는데, 이 기술이 발전하면서 우리는 단순한 시각적 영상에서 중요하거나 혹은 우리가 필요로 하는 정보만을 가공하여 사용할 수 있게 되었다. 이는 데이터분야에서 각각의 영상 또한 우리가 다룰 수 있는 데이터화 될 수가 있고 더 나아가 그를 통한 분석 또한 가능하다는 것이다.

그리고 현대에는 스마트폰 보급이 매우 대중화된 시대이다.



GLOBAL MOBILE USERS WILL REACH ALMOST 7B IN 2020. DATA FROM ERICSSON AND TUNE FORECASTS.
그림 1. 전세계 스마트폰 보급률 및 이후 추이

[그림 1]에서 확인할 수 있듯이, 현재에도 그 이용자가 매우 많으며, 앞으로도 계속

하여 증가할 것으로 전망하고 있다. 이러한 현상에서 우리가 주목할 것은, 스마트폰을 통하여 우리는 누구나 쉽게 영상을 촬영할 수 있고, 연결된 인터넷을 이용하여 쉽게 공유가 가능하다는 점이다. 이처럼 현재에는 영상 정보라 할 수 있는 것들이 넘쳐나고 있다. 이러한 방대한 영상 정보를 데이터라 볼 수 있고 영상 처리 기술을 이용하여 이를 적절히 분석하고 활용한다면 많은 곳에서 이용이 가능할 것이다.

본 연구에서는 이러한 영상처리 기술을 이용하여 영상에서 유의미한 데이터를 생성 및 수집하고, 이를 머신러닝 기법을 활용하여 분석하여 처리된 데이터를 적절히 활용하고자 한다.

1.2. 연구목표

영상 처리기술이 많은 발전을 이루었다고 하지만, 그 처리와 패턴인식 등의 대한 이론적인 학습량이 엄청난 시간투자를 요구한다. 그리고 이러한 이론내용을 바탕으로 적용과 응용 과정이라 할 수 있는 관련 문제 해결은 해당 이론에 대한 알고리즘 구현을 통한 프로그래밍 과정으로 이루어진다. 그런데 영상처리 알고리즘은 그 복잡도가 매우 높아 비교적 간단한 알고리즘을 직접 구현하는 것도 쉬운 일이 아니다. 다행히 최근에 영상처리에 사용할 수 있는 다양한 알고리즘을 오픈소스 라이브러리 형태로 제공을 하고 있고 그 중 가장 대표적이라 할 수 있는 것이 OpenCV이다.

본 연구에서는 OpenCV library가 기대할만한 수준의 이미지처리가 가능하기에 이를 영화에 적용해보고자 한다. 영화는 현재 우리나라에서 굉장히 대중적인 문화생활로 자리 잡았다. 과거에 비해 여가 시간이 늘고 취미 생활을 즐기는 사람이 많아졌다고는 하지만 그 절대적인 시간과 충분한 여가 활동이 부족하다 보니 영화 관람이 적절한 시간과 비용을 투자하여 즐길 수 있다는 것에 그 이유가 있다. 이러한 상황 속에 영화는 현재 어떤 문화 콘텐츠보다도 강력한 영향력을 가지게 되었다. 하지만 매 년 천 편이 넘는 영화가 개봉을 하고, 이들을 모두 본다는 것은 불가능하기에 우리는 영화 관람 시 영화를 선택해야하는 상황에 직면한다. 이 선택에 기준이 될 수 있는 것은 각 줄거리, 장르와 함께 영화의 흥행 여부, 및 주변 사람들의 관람 평 또한 매우 중요하게 작용하여 최근에는 영화마다 평점 알바까지 성행하는 상황이다. 하지만, 이러한 주변사람들의 평가 혹은 평점은 개인의 견해와 성향에 따라 차이를 보일 수 있고, 심지어 평점에 경우 평점 알바라 하는 조작의 여지도 있기에 이를 객관적이고 신뢰할만한 기준이라 보기는 어렵다. 그렇다고 장르만으로는 같은 장르의 다른 영화에도 호불호가 나뉘는 경우가 허다하다. 그래서 이번 연구에서는 선택의 도구를 영화 포스터를 중점으로 한다. 영화 포스터는 영화 매체를 이용 시 처음 관심을 증폭시키는 것에 최소의 시공간으로 접할 수 있는 수단이다. 한 컷으로 예상 관객들로 하여금 핵심내용을 상상하는 수단이 되며, 감독의 전달 의도를 최대한 잘 반영한 이미지라 할 수 있다[1, 2].

각 영화 포스터들을 가지고 영상 처리 기술을 이용하여, 영화들에 대한 전반적인 색감에 대한 정보를 얻을 수 있을 것이다. 이러한 포스터의 전반적인 색감들에 대한 데이터 분석을 이용하여 영화 선택에 있어 중요한 기준으로 사용해보고자 하는 것이 이번 연구의 목적이다.

1.3. overview

본 연구에선 영화 포스터를 데이터 기반으로 하여 대표색을 추출하고 이를 통해 색상 차를 구하는 방법을 제안하며 또한, 구해진 색상 차를 활용하여 유사한 분위기의 영화를 추천하는 시스템은 제안한다. 1장에서는 연구배경과 목표에 대하여 설명하고 2장에선 본 연구에 앞서 이용된 선행 연구 및 기술 현황에 대하여 소개한다. 3장에선 본 연구의 가정과 데이터 수집 및 전처리 그리고 Similarity를 구하는 과정에 대해 소개하고 있다. 4장에서는 실제 구현과 분석된 데이터를 통한 결과를 제공한다. 마지막 5장에선 본 연구의 결론과 소감에 대하여 언급한다.

2. 관련연구 및 기술현황

2.1. 관련 연구

연구 목적절에서 언급했듯이 영화 포스터는 영화의 핵심내용부터 전하고자 하는 바 등 많은 정보를 내포하도록 만들어진 이미지이다. 이번 연구에서 이러한 포스터에서 주목하고자 하는 것은 색채이다. 색체는 빛이 물체와 만날 때 물리적 현상들을 통하여 색이 감각기관인 눈을 통해서 지각되거나 그와 같은 지각현상과 마찬가지의 경험효과를 가리키는 현상을 의미ㅇ다. 이러한 색체들이 사람의 심리에 영향을 줄 수 있고 이에 대하여 연구하는 색체심리학 분야 또한 존재한다.

[3, 4]를 통해 영화의 지배적인 색채 비율이 영화 분위기에 지대한 영향을 끼치고 있는 것을 확인할 수 있다. 또한 영화와 유사하게 동영상을 이용한 대중 매체인 애니메이션 장르에서도 시각 이미지로서의 색채가 의미전달의 도구로서 상징적 의미표현에 효과적인 활용이 가능하다는 것을 알 수 있다[5]. 그러므로 영화의 핵심내용과 전달하는 바가 잘 나타내도록 만들어진 포스터를 영상처리 기법을 활용하여 추출한 색감에서도 우리는 영화전반의 분위기와 정보들을 얻을 수 있을 것 이다.

앞서 포스터에서의 색채 추출은 영상처리 기법을 이용한다 하였다. 그리고 이러한 기법은 OpenCV 라이브러리를 활용할 것이다.

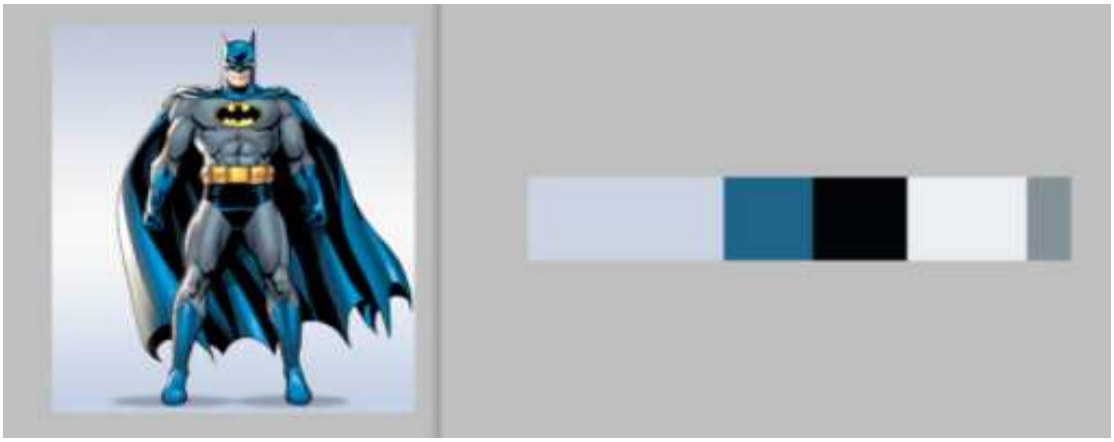


그림 2. OpenCV를 이용한 지배적인 색채 추출 예제

위 [그림 2]와 같이 영상처리 분야에 전반적인 이론적 지식을 모두 가지고 있지 않은 상태에서도 라이브러리를 이용하여 충분히 기법들을 적용해 볼 수 있다. 물론 이를 나의 연구에 맞추어 수정 및 보완해야할 부분들에 대해서는 더욱 학습이 필요하겠지만 상당수준의 알고리즘을 이용하기 쉽게 제공해 준다는 것에 라이브러리를 활용할 가치는 충분하다는 것을 알 수 잇다.

2.2. 기술 현황

2.2.1. 영상 처리

영상 처리 기술은 최근 딥러닝 기술이 적용됨에 따라 더욱이 발전하였다. 여기서 딥러닝 기술은 글로벌 시장 컨설팅 업체 Frost & Sullivan사에서는 기계가 대량의 데이터로부터 고도의 추상화를 해결하는 알고리즘을 이해하고 사용하기 위한 기계학습의 한 형태라고 정의하고 있다. 영상 분석 대상으로부터 복잡한 특징들을 추출하여 처리하고 그에 따른 결과를 도출하는 영상인식 분야에 이러한 특징이 직접적으로 응용될 수 있다.

영상 처리는 크게 4가지 분야에 적용되고 있다. 첫 번째로 시각인지 분야이다. 컴퓨터 스스로가 객체 및 행동 인식 혹은 상황 등을 이해에 활용되는 기술이다. 영상처리를 말할 때 빠지지 않고 등장하는 컴퓨터 비전이라는 용어는 이 분야에 해당한다. 두 번째는 공간인지 분야이다. 이는 대게 3차원 영상에서 거리나 깊이 분석, 혹은 공간상의 사물을 이해하는 기술이다. 이는 자율주행자동차에서 매우 핵심적인 분야라 할 수 있다. 세 번째는 스토리 압축/창작분야로 영상으로부터 스토리를 이해하거나 의미 있는 내용을 요약하는 등의 기술이다. 이는 주로 자동 영상 편집 기능에 적용되고 있다. 그리고 끝으로 기계학습 분야인데, 영상 내 사물에 대한 통계적 처리나 사물 간 클러스터링 등에 활용되는 기술이다. 이번 연구에서는 이 기계학습 분야 그 중에서도 클러스터링 기법을 중점으로 영상 처리를 이용할 것이다.

2.2.2. OpenCV

OpenCV는 Open Source Computer Vision Library의 줄인 말로 BSD 라이센스하에 배포돼 학술 및 상업적 용도로 무료로 이용가능하다. C++, python 및 Java로 이용이 가능하고 Windows, Linux, Mac OS, iOS 및 Android까지의 현재 활용되는 대부분 OS에서 사용이 가능하다는 장점을 가지고 있다. OpenCV는 계산 효율성과 실시간 응용 프로그램 제작에 중점을 두고 설계되었고 멀티 코어 프로세싱을 활용할 수 있다. 전 세계적으로 4만 7천명 이상의 사용자 커뮤니티와 1400만이 넘는 다운로드 수로 현재 영상 처리 분야에서 가장 대중적인 library이다. 현재는 간단한 아키텍쳐와 함께 사용하는 경우[6]부터 광산 검사, 웹상의 지도 스티칭부터, 다른 로봇 공학에 까지 다양한 분야에 이용되고 있다. 이전에는 멀게만 느껴졌던 영상처리 분야를 웬만한 처리 결과물들을 원리와 분석에 대한 지식 없이도 단순한 코드 몇 줄로 구현할 수 있어 영상처리 분야를 대중화 시킨 주역이라 할 수 있다.

지금도 꾸준히 버전 업데이트가 이루어지고 있다. 3.4.1 버전이 최근에 업데이트 됐다. 기존 2014년 까지는 2.4.x 형태의 버전으로 진행을 해오다. 이후 3.x버전으로 업데이트하기 시작했다. 대부분 기능들이 3.x버전에서 호환되지만 몇 가지 큰 차이가 있기에 기존 이용 시스템을 위하여 이후에도 꾸준히 2가지 버전이 모두 업데이트되고 있다. 본 연구에서는 기존 시스템을 사용하는 것이 아니기에 최신 버전인 OpenCV 3.4.1을 사용할 것이다.

2.2.3. k-means

k-means Clustering에 대하여 보기에 전 우선 머신 러닝에 대하여 알아야 한다. 머신러닝은 어떤 task가 주어지고 이를 풀기위한 일반적인 알고리즘 개발의 한 형태로 볼 수 있지만, 이 때 성능 측면에서 프로그램 자체가 학습을 하여 개선이 되도록 하는 알고리즘의 개발을 목적으로 하는 것에 차이가 있다.

우리가 머신 러닝에 주목하는 이유는 세상의 갖갖이 현상들에 대하여 우리가 준 입력과 그에 따른 출력만으로 이를 단순한 함수로 정의할 수 있는 경우는 거의 없다. 그러니 찾고자 하는 이 함수가 매우 복잡하고 어려운 경우가 대부분이고 이 때 머신 러닝이 유용하게 이용될 수 있다. 우리가 특정 문제를 해결할 방법을 결정하는 것이 아니라, 우리는 해결 가이드라인만 제시해 주면, 컴퓨터가 스스로 풀 수 있도록 하는 것이다. 또한, 머신 러닝이 급부상하게 된 계기 중에는 빅데이터와 연관이 있다. 그 데이터의 양이 너무도 많아져, 사람이 직접 분석 처리하기에는 어려움이 많다. 이에 머신 러닝을 이용하여 기계가 방대한 데이터를 input data로 이용하여 더욱 좋은 성능의 결과를 만들어 낼 수 있는 것이다.

Clustering은 머신 러닝의 일종으로, 특정 category가 주어지지 않은 데이터들을 가장 잘 설명할 수 있는 cluster를 찾는 것이다. 그러면 이러한 cluster를 정의하는 방법은 여러 가지가 있을 것이다. k-means는 같은 cluster 내부의 데이터는 가깝다고 가정한다. 이 때, 각각의 클러스터마다 중심이 존재하고, 클러스터 내의 데이터

가 중심과 얼마나 가까운가가 cost가 되어 cost를 최소화 하는 cluster를 찾는 알고리즘이다. 각 클러스터를 찾을 때 optimal한 중심을 찾기 위해 cluster의 mean을 계산하고, 총 k개의 cluster를 찾는다 하여 k-means라 부르는 것이다.

k-means clustering은 비교적 구현이 간편하여 이미지 또는 비디오 시퀀스의 픽셀 그룹화에 종종 사용된다. 그리고 이러한 k-means clustering을 이용에서 성능을 개선하려는 노력은 계속 진행해 왔다[7, 8].

3. 제안 논문 소개

3.1. 가정

본 연구에서는 결과 도출을 위한 과정에 2가지 가정을 기반으로 한다.

- 포스터는 영화를 대표하는 요소이다. 서론의 연구목표에서 언급했듯이, 포스터는 영화에 대한 최초 관심 증폭에 이용하는 수단이며 이를 위하여 영화의 핵심내용과 감독의 전달 의도를 잘 반영하고 있는 이미지라 할 수 있다[1, 2]. 그러므로 대표 이미지로서의 포스터를 분석하여 얻은 색감은 영화의 전반적인 색감으로 판단할 수 있을 것이다.
- 포스터 색감의 유사성을 이용하여 영화의 분위기를 판단 가능하다는 것이다. 영화는 각 영화별로 내용은 각기 다르지만, 이들을 크게 장르로 구분하여 묶일 수 있고 이 때 같은 장르의 영화들은 다른 장르의 영화에 비하여 전반적인 분위기가 유사하다. 본 연구에서는 이를 이용하기 위하여 색감의 유사성으로 분위기 및 장르를 구분할 수 있을 것을 가정으로 한다. 이는 4.2절의 similarity 분석에서 본 가정이 입증된다.

3.2. 데이터 수집 및 전처리

3.2.1. 데이터 수집

본 연구에서는 장르별로 전반적인 분위기에 차이가 있는 것을 이용하기 위하여, 장르를 기준으로 데이터를 수집하였다. 하지만 하나의 영화에 다양한 장르를 포함하는 경우도 많고, 다른 장르라도 느껴지는 분위기가 비슷한 경우가 있다. 예를 들어 스릴러와 공포는 확실히 다른 장르로 구분하지만 전반적인 분위기는 유사성을 가지고 있다. 그러므로 이번 연구에서 데이터 수집할 장르를 드라마와 공포 두 가지로 한정한다. 이들은 한 영화에서 공존할 수 없는 장르이고 전반적인 분위기도 큰 차이를 보인다. 드라마 장르에서 12가지 영화에 대한 포스터 이미지를, 공포 장르에서 10가지 영화에 대한 포스터 이미지를 데이터로 수집하여 사용한다.

3.2.2. 데이터 전처리

수집한 데이터를 활용하기에 앞서 전처리 과정이 필요하다. 전처리를 진행하면서 이슈가 되었던 것은 크게 두 가지이다.

- 흰 배경의 비중이 큰 경우 대표색의 처리에 문제가 있다. 흰 배경의 경우 색상 자체에 의미가 있기 보다는 여백으로서 의미를 가진다. 그러므로 이 비중이 충분히 크다면 clustering 결과에 흰색이 나타날 것이고 이는 색을 이용한 비교에서는 적절하지 않은 데이터로 작용할 것이다. 그러므로 이를 RGBα 색상체계로 옮겨 투명여부를 결정하는 α값으로 흰색배경을 투명으로 처리하는 것이 가능하다. 그러나 clustering 함수는 α값을 사용하지 않아, 흰색배경이 그대로 들어가는 문제가 발생하므로, 다른 해결방안을 고려해

야만 한다.

- clustering의 속도를 개선해야할 필요가 있다. clustering을 할 때 입력 이미지에 따라도 그 속도가 매우 상이했고, 전처리 결과에 얻을 색상의 수를 결정하는 k값에 따라서도 속도의 차이가 있었다. 이것이 문제가 되는 것은 한 이미지에서 k값에 따라 결과를 얻는데 5분이 넘는 시간을 소요돼 전체 전처리과정의 시간이 너무 길어진다. 그러므로 처리 속도는 높이지만 그 결과 값에는 차이가 없게 개선을 해야만 한다.

위 이슈들을 해결은 4장의 구현 부분에서 다룰 것이다.

그림 3. (a) 영화 "써니" 포스터 (b) clustering 결과

[그림 3]은 영화 "써니"를 전 처리한 결과를 png파일의 형태로 저장한 것이다. 각 색상의 값은 대표색별로 RGB값을 [0,255]값과 각 색상별 차지 비중을 전체 100%를 기준으로 txt파일로 저장하고 있다.

3.3. Similarity

본 연구에서는 전처리한 데이터를 이용하여 포스터 대표 색감사이에 유사도를 통하여 분위기를 예측하려 한다. 유사도는 Lab색상체계에서 구할 수 있는 색상 차이 값 $\delta(A_i, B_j)$를 이용한다.

$$Sim(A,B) = \sum_{i=1}^{k}\sum_{j=1}^{k}\delta(A_i,B_j)*w_{Ai}*w_{Bj} \quad \textbf{(1)}$$

수식 (1)은 두 이미지 *A, B* 데이터 사이의 색상 유사도 $Sim(A,B)$를 얻는 식이다. 이 때, A_i는 이미지 A에서 i번 째 clustering 데이터를 의미한다. 또한, w_{Ai}는 그 비중을 나타내며, 그 값은 폐구간 [0,1]을 가진다.

증명. $Sim(A,B)$의 최솟값과 최댓값을 구하기 위해, 다음과 같이 clustering 파라미터 k값을 1로 가정한다. 우선 이미지의 색이 동일한 상황, 즉, $Sim(A,B)$가 최솟값을 가지는 상황을 가정한다. 두 이미지가 동일하기에 $\delta(A_1, B_1) = 0$이다. 따라서 S는 최솟값으로 0을 가진다. 다음으로는 두 이미지의 색상이 완전히 상반되는 경우, 즉 A_1는 RGB(0,0,0), B_1는 RGB(255,255,255)를 가정한다. $\delta(A_1, B_1) = 1, w_{A1} = w_{B1} = 1$ 따라서 $Sim(A,B)$는 최댓값으로 1을 가진다. 이를 통해, 수식 (1)의 결과인 $Sim(A,B)$는 폐구간 [0,1]임을 알 수 있다.
다음의 증명에서는 k 값을 1로 가정하였지만, 실제 k값은 1보다 큰 값을 사용할 수 있는데 이 때, 동일한 이미지를 비교하여도 $Sim(A,B)$는 0의 값을 가질 수 없다. 최댓값 또한 1의 값을 가질 수 없다.

4. 구현 및 결과분석

4.1. 데이터 전처리

본 절에서는 3.3.2에서 언급한 이슈사항들을 해결하기 위하여 다음과 같이 구현하였다.

4.1.1. 흰색 배경 처리

흰색 배경의 비중이 큰 경우에는 clustering 결과에 흰색 혹은 흰색에 아주 근접한 결과가 나타날 것이다. 그러므로 원하는 k값 대신 k+1을 파라미터로 사용하여 clustering을 한다. 그 결과에서 흰색을 나타내는 값은 제거하고 남은 색들을 이용하여 비중을 새로 구하는 것으로 데이터를 처리한다. 이 때 흰색 배경이 없는 경우에도 같은 k+1의 파라미터로 k개의 결과를 얻기 위하여, 가장 비중이 낮은 색을 제거하는 것으로 한다.

4.1.2. 속도 개선

각 이미지 별로 속도 차이는 이미지를 구성하는 pixel수의 차이에서 발생한다. 따라서 이를 해결하기 위해 pixel수 자체를 줄이는 방법을 선택하였고, python 라이브러리 cv2에서 제공하는 resize 메소드를 사용하여 x, y축을 기준으로 pixel수를 반으로 감소시킨다.

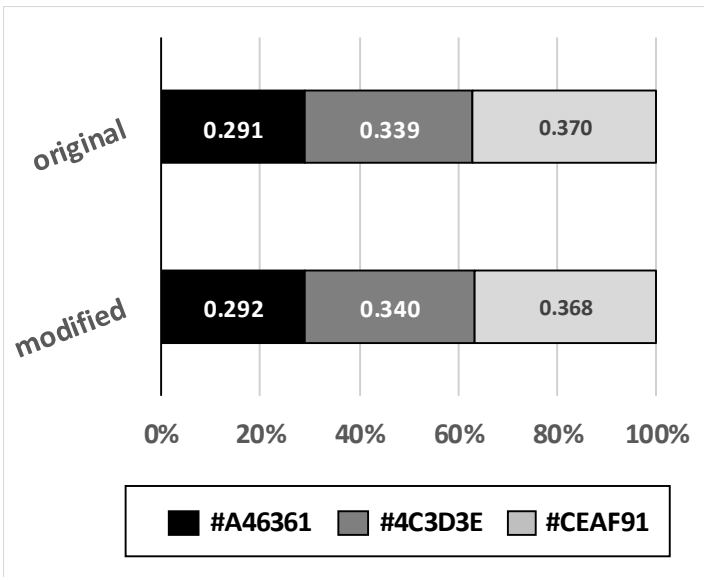


그림 4. 영화 "써니"를 k값 3개로 clustering한 결과

[그림 4]는 원본 파일과 축소된 파일의 clustering 결과를 시각화한 그래프이다. 두 결과를 비교하였을 때, 선택된 색상의 RGB값의 차이는 거의 없었으며, 그 비중 또한 매우 유사한 것을 확인할 수 있었다. 따라서 본 연구에서는 이미지 자체의

pixel 수를 감소하여 속도 개선을 해결한다.

4.2. Similarity 분석

본 절에서는 3.3절에서의 두 색의 Similarity를 구하고 본 연구의 두 번째 가정을 뒷받침하고자 한다.

164.03369584116115 99.35438256315925 97.27555771022239
76.1729989815839 60.53790841287966 62.369208140456784
205.55858532366403 175.4398968197173 144.67350030718717
0.3398874539357923 0.2918292055376647 0.3682833405265429
#a46361 #4c3d3e #ceaf91

그림 5. 영화 "써니"를 전 처리한 output data파일

[그림 5]는 전처리된 데이터 파일의 예시를 보여주고 있다. 총 3개의 clustering 결과를 도출하는 것이므로, 세 번째 줄까지는 대표색 3개의 RGB값을 폐구간 [0,255] 3개로 저장하고 있다. 4번째 줄은 순서대로 대표색의 비중을 나타내고 있고 마지막 줄을 각 대표색들의 RGB값을 hexacode 형태로 저장한 것이다. RGB값을 Lab 체계로 옮기는 것은 python colormath 라이브러리를 이용한다. 이 때 변환에 이용할 RGB 값을 255로 나누어 폐구간 [0,1]의 값으로 변형하고 sRGBColor 메소드의 파라미터로 활용하여 sRGB 색상체계로 변환한 뒤 convert_color 메소드를 이용하여 sRGB에서 Lab 체계 상의 색상으로 변형한다.

Lab 색상체계는 두 색상간의 차이를 delta_e_cie2000 메소드를 활용하여 폐구간 [0,1]값으로 반환할 수 있다. 이 차이값을 3.3절 similarity 절에서의 수식 (1)의 $\delta(A_1, B_1)$로 사용하고 나머지 수식은 값의 곱 및 합 연산이므로 쉽게 구현할 수 있다.

표 1. 영화들 간의 similarity 결과

	A	B	C	D	E	F	G	H	I	J	K	L
A	0	0.02	0.05	0.08	0.09	0.25	0.91	0.9	0.88	0.75	0.8	0.81
B	0.02	0	0.06	0.08	0.1	0.25	0.89	0.88	0.87	0.75	0.8	0.81
C	0.05	0.06	0	0.11	0.06	0.2	0.8	0.8	0.79	0.72	0.76	0.76
D	0.08	0.08	0.11	0	0.13	0.28	0.87	0.86	0.85	0.74	0.79	0.8
E	0.09	0.1	0.06	0.13	0	0.2	0.8	0.79	0.78	0.72	0.75	0.75
F	0.25	0.25	0.2	0.28	0.2	0	0.49	0.5	0.48	0.44	0.45	0.45
G	0.91	0.89	0.8	0.87	0.8	0.49	0	0.06	0.02	0.1	0.07	0.06
H	0.9	0.88	0.8	0.86	0.79	0.5	0.06	0	0.06	0.08	0.05	0.05
I	0.88	0.87	0.79	0.85	0.78	0.48	0.02	0.06	0	0.08	0.06	0.05
J	0.75	0.75	0.72	0.74	0.72	0.44	0.1	0.08	0.08	0	0.03	0.04
K	0.8	0.8	0.76	0.79	0.75	0.45	0.07	0.05	0.06	0.03	0	0.02
L	0.81	0.81	0.76	0.8	0.75	0.45	0.06	0.05	0.05	0.04	0.02	0

Brief	Movie
A	cat
B	closeknit
C	life
D	ride
E	wonder
F	timetraveler
G	annabelle
H	counjuring2
I	jigsaw
J	gonziam
K	hereditary
L	getout

[표 1]은 각 장르별로 6가지씩 영화를 선정하여, Similarity 분석까지의 과정을 마친 결과를 나타내고 있다. 이 때, 표의 가독성을 위하여 각 이름은 알파벳으로 표현하고 각 알파벳이 나타내는 영화 제목은 우측의 표에 별도로 표기하였다.

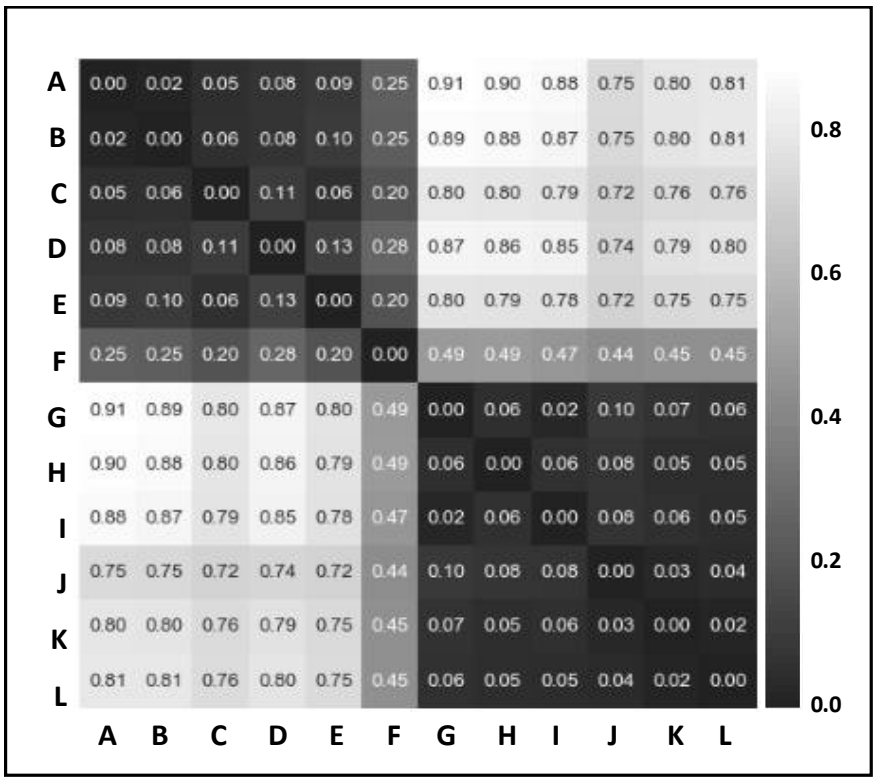


그림 6. similarity값의 히트맵

[그림 6]은 [표 2]를 이용하여 $Sim(A,B)$을 시각화할 히트맵을 나타낸 것이다. [그림 6]을 통해 같은 장르의 영화에서는 낮은 $Sim(A,B)$, 즉 높은 유사율을 가짐을 확인할 수 있다. 이는 본 연구의 기반인 가정 2를 뒷받침하는 증거가 된다.

4.3. 결과 분석

본 연구에서는 기존 데이터에 분석할 영화 포스터 데이터를 추가하여 해당 영화와 기존 영화간의 $Sim(A,B)$를 구한다. 그리고 이들 중 가장 낮은 값, 즉 가장 유사율이 높은 영화 K개를 추천한다. 본 절에서는 K를 3으로 선정하였다.

표 2. 영화별 추천 결과

test movie			1	2	3
1	barefoot	movie	wonder	life	cat
		value	0.077	0.078	0.131
2	vanishing	movie	sunny	timetraveler	dangsin
		value	0.142	0.175	0.193
3	postman	movie	ride	cat	closeknit
		value	0.108	0.178	0.181
4	haunted	movie	jigsaw	annabelle	getout
		value	0.019	0.025	0.038
5	demon	movie	counjuring2	annabelle	jigsaw
		value	0.029	0.032	0.045

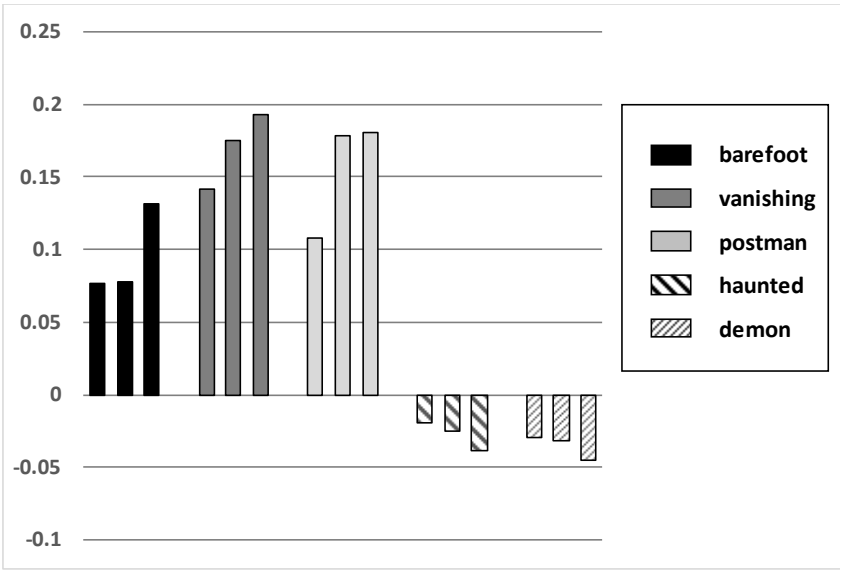


그림 7. 영화별 추천 결과의 그래프

[표 2]는 새롭게 추가된 5가지 영화에 대하여 결과를 정리한 것이다. 각 영화별 유사율 순위로 영화를 나타내고 그 때의 $Sim(A,B)$값을 value로 나타낸 것이다. [그림 7]은 장르의 구분을 위해 드라마 장르의 값은 양수로 공포 장르의 값은 음수로 [표 2]의 value값을 변경하여 그래프로 나타낸 것이다. [표 2]와 [그림 7]을 통해 각 영화별로 같은 장르에서 3가지 씩 영화를 추천하고 있는 것을 확인할 수 있다. 이는 가정에 부합하는 적절한 결과가 도출된 것이다.

5. 결론 및 소감

본 연구는 영화 포스터를 활용하여 영화를 분석하여 영화 추천에 활용하고자 한다. OpenCV를 활용한 k-means clustering 기법을 이용해 포스터에서 대표색을 추출하고, 이를 이용해 색상 차를 구하고 영화간의 유사도를 수치화한다. 마지막으로 이 유사율을 이용하여 유사한 분위기를 지닌 영화 K개를 추천한다. 분석할 영화와 선정된 영화가 같은 장르로 묶이는 것으로 보아 충분히 활용 가능한 결과를 얻은 것을 확인했다.

본 연구는 장르를 2가지로 한정하였기에, 다양한 장르에 대하여 적용할 때에는 좀 더 추가적인 검증을 요구한다. 또한, 데이터로 활용한 영화의 수가 더욱 많을수록 다양한 풀에서 비교가 가능할 것으로 예상된다. 하지만, 포스터에서 추출한 색감을 통해 충분히 분위기를 구분할 수 있는 것을 확인했기에 추천 시스템에 하나의 척도로 활용할 수 있을 것으로 예상되므로, 향후 연구 가치를 가진다.

이번 연구에서 처음으로 python 언어를 사용해 보았다. python의 우수함에 대해서는 익히 들어왔지만, 사용해본 결과 사용자 편의에 맞게 매우 잘 구성되어 있는 언어라는 것을 알 수 있었다. 또한 매우 많은 기능들이 라이브러리로 이미 구현 되어 있기에 다양한 기능들을 쉽게 접근하여 사용할 수 있는 것도 큰 장점이었다. python을 이용하였기에 연구의 진행도 더욱 수월하였고, 다양한 예시가 있어 오류를 수정해 나가는 것에서도 많은 도움을 받을 수 있었다. 또한 기존에 존재하는 데이터를 이용하여 전 처리 과정과 분석을 거쳐 유의미한 결과를 도출하는 과정을 진행해봄으로서, 데이터 분석을 경험해보는 좋은 기회였다.

6. 참고문헌

[1] 김형석, 김성훈, "국내 다양성영화 포스터의 시각적 상징성에 관한 연구" 한국디자인문화학회, 3월 2015.

[2] 조성근, 김종근, "계획행동 이론을 적용한 영화관람 의도의 결정요인에 관한 연구: 영화포스터 표현형식의 조절역할을 중심으로" 한국벤처창업학회, 12월 2015.

[3] Cheng-Yu Wei, Nevenka Dimitrova, Shih-Fu Chang, "Color-Mood Analysis of Films Based on Syntactic and Psychological Models" IEEE International Conference on Multimedia and Expo, 6월 2004.

[4] 임누리, 오인영, "회화에 나타난 색채상징성 및 색채심리", Joural of the Korean Society of Costume, 6월 2010.

[5] 진정식, "애니메이션에 적용된 색채의 심리적 의미 전달", 한국콘텐츠학회, 11월 2006.

[6] Dr. S. Syed Ameer Abbas, Dr. P. Oliver Jayaprakash, M. Anitha, X. Vinitha Jaini, "Crowd Detection and Management using Cascade classifier on ARMv8 and OpenCV-Python", 2017 International Conference on Innovations in Information, Embedded and Communication Systems, 3월 2017.

[7] Tapas Kanungo, David M. Mount, Nathan S. Netanyahu, Christine D. Piatko, Ruth Silverman, Angela Y. Wu, "An Efficient k-Means Clustering Algorithm:

Analysis and Implementation", IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, 7월 2002.

[8] Tse-Wei Chen, Yi-Ling Chen, Shao-Yi chein, "Fast Image Segmentation Based on K-Means Clustering with Histograms in HSV Color Space", Multimedia Signal Processing, 10월 2008.

[9] 김광용, 조기성 "딥러닝 기반 영상처리 응용 기술개발 및 서비스 동향", 정보통신기술진흥센터, 5월 2016.

[10]OpenCV, https://opencv.org/

[11] 스마트폰 이용 추이,

https://www.tune.com/blog/global-mobile-why-2016-is-the-global-tipping-point-for-the-mobile-economy/

Appendix A. clustering code

```
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import numpy as np
import argparse
import utils
import cv2
from matplotlib.colors import rgb2hex

ap = argparse.ArgumentParser()
ap.add_argument("-i", "--image", required = True, help = "Path to the image")
ap.add_argument("-c", "--clusters", required = True, type = int, help = "# of clusters0")
args = vars(ap.parse_args())

image = cv2.imread(args["image"], cv2.IMREAD_UNCHANGED)
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
image = cv2.resize(image, None, fx=0.5, fy=0.5, interpolation=cv2.INTER_NEAREST)

r,g,b = cv2.split(image)

plt.figure()
plt.axis("off")
plt.imshow(image)

image2 = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
H, S, V = cv2.split(image2)

meanH,meanS,meanV,_ = np.uint8(cv2.mean(image2))

print(meanH,meanS,meanV)

image = image.reshape((image.shape[0] * image.shape[1], 3))
for x in range(1):
    args["clusters"] = args["clusters"] - 1
    clt = KMeans(n_clusters = args["clusters"])
```

```
clt.fit(image)

hist = utils.centroid_histogram(clt)

print(hist)
print(clt.cluster_centers_)
new_hist = np.zeros(args["clusters"] - 1)
new_centers = np.zeros((args["clusters"] - 1,3))
k = 0

for i in range(args["clusters"]):
    mean = np.mean(clt.cluster_centers_[i])
    if mean > 240:
        for j in range(args["clusters"]):
            if j == i:
                continue
            new_hist[k] = hist[j]
            new_centers[k] = clt.cluster_centers_[j]
            k += 1
        hist = new_hist
        clt.cluster_centers_ = new_centers
        sum_hist = np.sum(hist)

        for j in range(args["clusters"]-1):
            hist[j] = hist[j] / sum_hist
        break

if k == 0:
    m = np.argmin(hist)
    for i in range(args["clusters"]):
        if i == m:
            continue
        new_hist[k] = hist[i]
        new_centers[k] = clt.cluster_centers_[i]
        k += 1
    hist = new_hist
    clt.cluster_centers_ = new_centers
```

```
        sum_hist = np.sum(hist)
        for i in range(args["clusters"]-1):
            hist[i] = hist[i] / sum_hist


    new_centers = new_centers/255
    tmp=[]
    for i in range(args["clusters"] - 1):
        tmp.append([rgb2hex(new_centers[i])])

    hex_centers = np.array(tmp)


    print(hex_centers)


    print(hist)
    print(clt.cluster_centers_)

    bar = utils.plot_colors(hist, clt.cluster_centers_)

    cv2.imwrite(str(args["clusters"]-1) + "_" + args["image"][0:-4] + "_"
+"cluster.png", cv2.cvtColor(bar, cv2.COLOR_BGR2RGB))


    out = open(str(args["clusters"]-1) +'_data_' + args["image"][0:-4] + '.txt' , 'w')


    for i in range(args["clusters"] - 1):
        for j in range(3):
            print(clt.cluster_centers_[i][j], file = out ,end =" ")
        out.write("₩n")
    for i in range(args["clusters"] - 1):
        print(hist[i], file = out ,end =" ")
    out.write("₩n")
    for i in range(args["clusters"] - 1):
        out.write(hex_centers[i][0])
```

```
        out.write("  ")

out.close()
```

B. clustering utils code

```
import numpy as np
import cv2

def centroid_histogram(clt):
	# grab the number of different clusters and create a histogram
	# based on the number of pixels assigned to each cluster
	numLabels = np.arange(0, len(np.unique(clt.labels_)) + 1)
	(hist, _) = np.histogram(clt.labels_, bins = numLabels)

	# normalize the histogram, such that it sums to one
	hist = hist.astype("float")
	hist /= hist.sum()

	# return the histogram
	return hist

def plot_colors(hist, centroids):
	# initialize the bar chart representing the relative frequency
	# of each of the colors
	bar = np.zeros((50, 300, 3), dtype = "uint8")
	startX = 0

	# loop over the percentage of each cluster and the color of
	# each cluster
	for (percent, color) in zip(hist, centroids):
		# plot the relative percentage of each cluster
		endX = startX + (percent * 300)
		cv2.rectangle(bar, (int(startX), 0), (int(endX), 50),
			color.astype("uint8").tolist(), -1)
		startX = endX

	# return the bar chart
	return bar
```

C. get color similarity between genres code

```
from colormath.color_objects import sRGBColor, LabColor
from colormath.color_conversions import convert_color
from colormath.color_diff import delta_e_cie2000
import argparse
ap = argparse.ArgumentParser()
ap.add_argument("-c", "--clusters", required = True, type = int, help = "# of clusters0")
ap.add_argument("-g1", "--group1", required = True, help = "# of clusters0")
ap.add_argument("-g2", "--group2", required = True, help = "# of clusters0")

args = vars(ap.parse_args())

movie = [['cat', 'closeknit', 'life', 'ride', 'wonder', 'timetraveler', 'summer', 'sunny', 'still', 'dangsin', 'ROADS', 'ssesibong'],
         ['quiet', 'iden', 'lodgers', 'para', 'annabelle', 'counjuring2', 'jigsaw', 'gonziam', 'hereditary', 'getout']]
#movie = [['cat', 'closeknit', 'life', 'ride', 'wonder', 'timetraveler'],['annabelle', 'counjuring2', 'jigsaw', 'gonziam', 'hereditary', 'getout']]

genre =['drama', 'horror']
group1=genre.index(args["group1"])
group2=genre.index(args["group2"])
f2 = open('new_'+genre[group1] + '_' + genre[group2] + '_data.txt', 'w')
k = args['clusters']

for x in range(1):
    k = k-1
    print(k,file = f2)
    for y in range(len(movie[group1])):
        for z in range(len(movie[group2])):
            r1=[]
            g1=[]
            b1=[]
            colors1=[]
            r2=[]
```

```
            g2=[]
            b2=[]
            colors2=[]
            delta=[]
            f= open(str(k) + "_data_" + movie[group1][y] + ".txt", "r")
            for i in range(k):
                temp=f.readline().split(sep = ' ')
                r1.append(float(temp[0])/255)
                g1.append(float(temp[1])/255)
                b1.append(float(temp[2])/255)
            grav1 = f.readline().split(sep = ' ')
            f.close()

            f= open(str(k) + "_data_" + movie[group2][z] + ".txt", "r")
            for i in range(k):
                temp=f.readline().split(sep = ' ')
                r2.append(float(temp[0])/255)
                g2.append(float(temp[1])/255)
                b2.append(float(temp[2])/255)
            grav2 = f.readline().split(sep = ' ')
            f.close()

            for i in range(k):
                colors1.append(convert_color(sRGBColor(r1[i], g1[i], b1[i]),
LabColor))
                colors2.append(convert_color(sRGBColor(r2[i], g2[i], b2[i]),
LabColor))

            for i in range(k):
                for j in range(k):
                    delta.append([i,j,delta_e_cie2000(colors1[i],
colors2[j])*float(grav1[i])*float(grav2[j])])


            tmp=[i[2]/100 for i in delta]
            print(str(round(sum(tmp),3)), file = f2, end = ",")
        print(file = f2)
```

D. get color similarity between movies code

```
from colormath.color_objects import sRGBColor, LabColor
from colormath.color_conversions import convert_color
from colormath.color_diff import delta_e_cie2000
import argparse
ap = argparse.ArgumentParser()
ap.add_argument("-i", "--image", required = True, help = "Path to the image")
ap.add_argument("-c", "--clusters", required = True, type = int, help = "# of clusters0")
ap.add_argument("-g1", "--group1", required = True, help = "# of clusters0")
ap.add_argument("-g2", "--group2", required = True, help = "# of clusters0")

args = vars(ap.parse_args())

movie = [['cat', 'closeknit', 'life', 'ride', 'wonder', 'timetraveler', 'summer', 'sunny', 'still', 'dangsin', 'ROADS', 'ssesibong'], ['quiet', 'iden', 'lodgers', 'para', 'annabelle', 'counjuring2', 'jigsaw', 'gonziam', 'hereditary', 'getout']]

#movie = [['cat', 'closeknit', 'life', 'ride', 'wonder', 'timetraveler'],['annabelle', 'counjuring2', 'jigsaw', 'gonziam', 'hereditary', 'getout']]
genre =['drama', 'horror']
group1=genre.index(args["group1"])
group2=genre.index(args["group2"])

movie[0].append(args["image"][0:-4])
movie[1].append(args["image"][0:-4])


f2 = open(args["image"][0:-4]+genre[group1] + '_' + genre[group2] + '_data.txt', 'w')
k = args['clusters']
# Red Color

for x in range(1):
    k = k-1
```

```
        print(k,file = f2)
        for y in range(len(movie[group1])):
            for z in range(len(movie[group2])):
                r1=[]
                g1=[]
                b1=[]
                colors1=[]
                r2=[]
                g2=[]
                b2=[]
                colors2=[]
                delta=[]
                f= open(str(k) + "_data_" + movie[group1][y] + ".txt", "r")
                for i in range(k):
                    temp=f.readline().split(sep = ' ')
                    r1.append(float(temp[0])/255)
                    g1.append(float(temp[1])/255)
                    b1.append(float(temp[2])/255)
                grav1 = f.readline().split(sep = ' ')
                f.close()

                f= open(str(k) + "_data_" + movie[group2][z] + ".txt", "r")
                for i in range(k):
                    temp=f.readline().split(sep = ' ')
                    r2.append(float(temp[0])/255)
                    g2.append(float(temp[1])/255)
                    b2.append(float(temp[2])/255)
                grav2 = f.readline().split(sep = ' ')
                f.close()



                for i in range(k):
                    colors1.append(convert_color(sRGBColor(r1[i], g1[i], b1[i]),
LabColor))
                    colors2.append(convert_color(sRGBColor(r2[i], g2[i], b2[i]),
LabColor))
```

```
            for i in range(k):
                for j in range(k):
                    delta.append([i,j,delta_e_cie2000(colors1[i],
colors2[j])*float(grav1[i])*float(grav2[j])])

            #print(delta)

            tmp=[i[2]/100 for i in delta]
            #print(round(sum(tmp),3))
            print(str(round(sum(tmp),3)), file = f2, end = ",")
        print(file = f2)
```