**컴퓨터응용확률 HW2**

12131495 김준수

**Part A. Preliminaries**

1. Cross-validation(k-fold). 기본적으로, Machine Learning(이하 ML)에서는 overfitting문제가 지금까지 해결되지 않고 있다. overfitting문제란, 어떤 model이 너무 많은 train을 받았을 때 흔히 일어나는데, train set에 model이 ‘갇히기’ 시작하면서 train set에 대한 성능만 좋아지고 실생활이나 다른 dataset에는 적합하지 않게 되는 문제를 이야기한다.

K-fold cross-validation은 이러한 overfitting을 예방하기 위해 고안된 간단하고 대표적인 사례이다. Data를 k개로 나눈(접은) 뒤, 각각의 조각 1~k번을 하나씩 test set으로 놓고, 나머지를 train set으로 놓으면서 성능을 번갈아가면서 측정하는 방식이다. 이 방법은 안전하고 overfitting에 대해 상당히 강한 저항력을 보이나, 다른 알고리즘에 비해 약 (k-1)배의 복잡도를 가져서 dataset이 크거나 속도가 중요시되는 경우에 사용하기는 어렵다.

1. Precision-Recall은 상대적으로 간단하다. TP, FP, TN, FN을 먼저 이해해야 하는데, 이 2글자 중 앞의 T/F는 실제 데이터와 비교했을 때 ‘모델이 맞췄는지’, 뒤의 P/N은 ‘모델의 대답’을 나타낸다. 즉, TP는 ‘현실과 일치하고’ ‘맞다고 예측한’ 결과이고, FN은 ‘현실과 불일치하고’, ‘아니라고 예측한’ 경우이다. 이 중 Precision은 ‘모델의 긍정적인 예측’ 중 ‘정답인 경우’의 비율을 나타낸다. 대수적으로 TP / (TP + FP)로 나타낸다. Recall은 ‘실제로 긍정’ 중 ‘모델이 예측한 긍정’의 비율이다. 대수적으로 TP/(TP + FN)으로 표현한다.
2. 생략

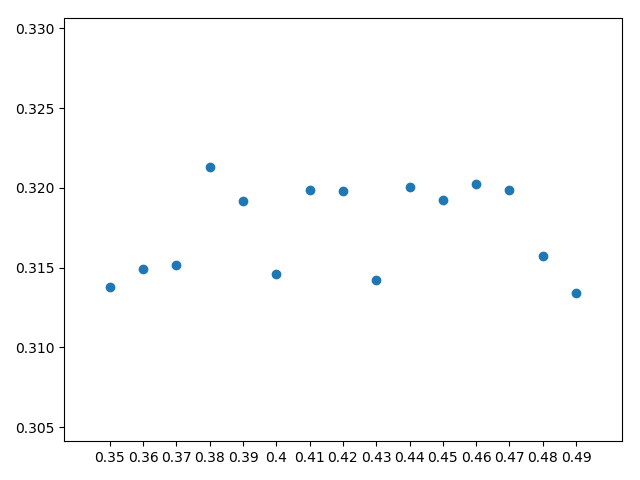
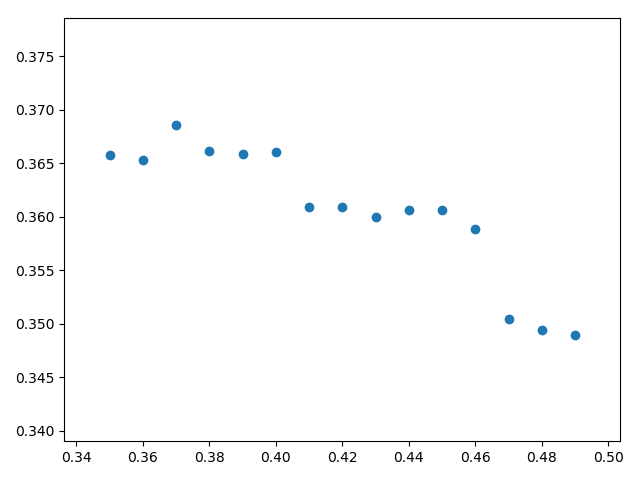
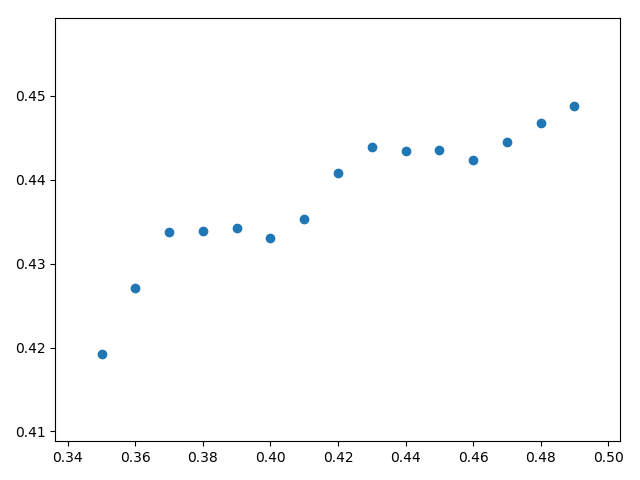
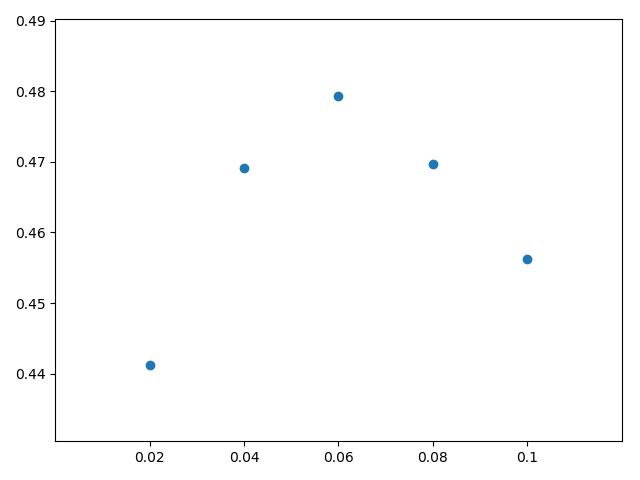
* 모든 코드에 대한 시행 결과는 main.py에서 얻을 수 있습니다. 자세한 사용법은 문항별로 따로 적었습니다.
* 모든 Evaluation은 F1 Score로 진행하였습니다. F1 score는 Precision, Recall의 Harmonic mean을 factor로 사용합니다..
* 모든 코드는 같은 폴더 안의 ‘originPics’ 폴더에 이미지 파일이, ‘FDDB-folds’에 텍스트 파일이 홈페이지에서 주어지는 대로 들어 있다는 가정하에 수행됩니다(사진 파일은 용량이 너무 커서 생략했습니다.).

**Part B. Skin detection using text data**

Skin\_nonskin.txt를 가져와서 처리하는 부분은 part\_B.py에 구현했습니다. 모든 클래스는 scikit-learn 라이브러리처럼 간단하게 fit, predict, evaluate 등의 함수를 이용한 손쉬운 사용을 최선으로 고려하여 구현하였습니다. 기본적인 인터페이스는 Classifier.py에 있습니다.

1. Part B – 1에 관한 모든 코드는 BayessianClassifier.py에 구현하였습니다. main.py의 아래쪽에 있는 b1a()와 b1b()에 대한 주석을 해제하신 뒤 main.py를 실행하시면 cross-validation 함수를 호출하고, 해당 문제의 조건에서 얻은 가장 높은 결과를 보여줍니다.
2. Part B – 2에 관한 모든 코드는 GaussianClassifier.py에 구현하였습니다. RGB의 경우에는 multivariate distribution을 사용했습니다. 기본적으로 Gaussian distribution의 pdf를 사용했는데, 주어진 input에서의 density를 판단의 근거로 삼았습니다. 물론 density와 mass는 다르지만, Gaussian distribution의 pdf가 mean에서 가장 큰 값을 가지고 멀어질수록 점점 작아지는 성질을 응용해, ‘값이 크’면 ‘평균과 가깝’다는 점을 근거로 density로 판단하여도 큰 문제가 없을 것이라 판단했습니다.

+) RGB의 경우, 각 feature의 독립을 가정하는 Bayessian을 사용할 수 없다고 판단되어, hyperparameter를 하나 골라서(pdf의 coefficient) prior처럼 사용했습니다.

1. b3() 함수에 구현되어 있습니다. hyperparameter\_tuning이라는 거창한 함수를 호출하지만, 하나의 parameter만 조정해서 실제 hyperparameter tuning처럼 많은 자원을 소모하지는 않습니다. 각 함수의 구현부에 있는 default value들은 이 함수들로 이미 구한 값을 적어 놨습니다.

* 각각 b-1-a. b-1-b, b-2-a, b-2-b의 prior(or coefficient)에 따른 F1-score

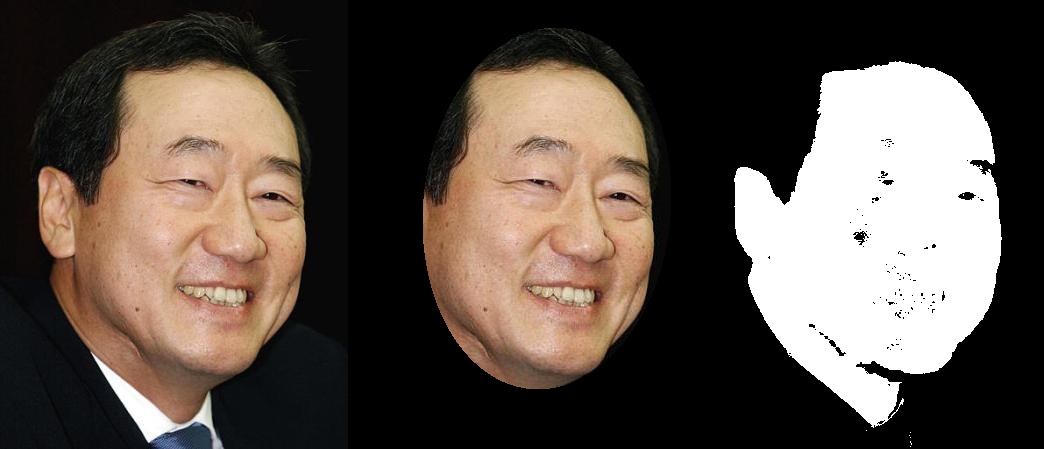
**Part C. Skin detection using real data**

* **주의:** 이 부분은 수행 시간과 메모리가 매우 많이 소모됩니다. 코드에서는 사진을 10장 정도만 처리하는데, 사진의 장수를 늘리면 메모리를 끝없이 계속 잡아먹어서 서버 컴퓨터를 사용하실 때는 out of memory에 주의해 주세요. (8GB RAM, Win10 64 bit, i7-8750 커피레이크 환경에서 100장 정도 돌리면 죽습니다.)
* 수행 여부를 빠르게 알아보실 수 있게, 미리 처리된 PicData.txt파일을 첨부했습니다. 텍스트 파일로 처리하시려면 main.py의 line 110의 주석을 해제하고 line 111을 주석처리하시면 됩니다. 다만 텍스트로 불러오실 때는 C2함수를 사용하거나 사진을 보실 수 없습니다. 빠른 체크를 위해, 미리 parse해둔 사진과 최종 결과를 마지막 example에 같이 붙여 두었습니다. 모든 parsing과 image process 부분은 part\_C.py에 구현되어 있습니다.
* 실제로 parse를 하는지, 제대로 detection한 결과가 잘 나오는지는 main.py의 가장 아래쪽 Part C 부분을 주석 해제하신 뒤 실행하시면 어느 정도 시간이 지난 후 results 폴더에서 확인하실 수 있습니다.

**Part D. Discuss about your approach and analyses.**

* 생각보다 결과가 잘 나와서 놀랐습니다. 처음에는 답이 없어 보여서 ‘그냥 Luminance나 greyscale로 단변수로 만들고 처리할까’ 싶었는데 생각보다 잘 처리됐습니다.
* Part B – 1 – b, Bayessian Classifier using RGB part에서 인상적인 부분이 있었습니다. 점수가 정말 높게 나옵니다. Precision, Recall이 default prior로 했을 때 거의 .95에 가까운 수치로 나옵니다. 하지만 예측을 해 보니, ‘밝은’ 부분만 잡은 경향이 강했습니다. overfitting이 있었던 것으로 예상됩니다.
* 제출한 프로젝트에서는 기존 이미지, parse된 이미지, 예측 이미지를 붙이는 과정에서 구현의 편의를 위해 C-1에서 처리된 이미지들만 예측하게 했습니다.
* 전체적으로 피부색이 다른 걸 고려하지 못한 게 아쉬웠습니다. 제가 GMM을 사용할 줄 알았다면 더 좋지 않았을까 싶습니다. 또는 labeling에서 아시아계, 아프리카계, 미국계 등으로 피부색에 대한 표시가 되어 있다면 성능이 월등히 올라가지 않을까 싶습니다.
* 또, annotation이 완벽하지 않아서 noise가 낀다는 점이나, 색깔로 분류하다 보니 비슷한 색의 물체와 구분하는 데 한계가 뚜렷하게 보였습니다.

**Good predict :**



**Bad predict :**

