# Term Project Report

2022-17611 김은형

Problem1. DNA Image Classification

DNA Image Classification의 경우, svm을 이용한 모델 1개와, image augmentation을 이용하여 학습 데이터셋을 다양화시킨 후, svm을 이용한 모델을 1개 더 만들어, 두 모델끼리 직접 투표를 시키는 Voting Classifier를 직접 class로 구현하여 classification을 진행하였다.

첫 번째 모델의 경우, scikit-learn에서 제공되는 SVC를 이용하였다. 사용한 코드와 파라미터들은 제출한 ipynb 파일에서 확인할 수 있다. 사용된 파라미터는 scikit-learn의 GridSearchCV를 이용해 찾았으며, 해당 코드는 ipynb의 Appendix에 주석처리하여 첨부하였다. 위의 모델로 얻은 csv 파일('svm2.csv')을 제출한 결과, 0.96055의 public score를 얻을 수 있었다. 가독성을 위하여, 동일한 파일을 'svm.csv'이라는 이름으로 첨부하였다.

csv 파일과 test dataset의 사진들을 직접 비교해 본 결과, 위 모델은 2번 class를 1번 class로 오분류하거나, 1번 class를 3번 class로 오분류하는 경우가 오류의 대부분을 차지함을 알 수 있었다. 이는 2번 class와 3번 class의 데이터셋 자료가 부족한 것이 이유라고 판단하여, opencv의 함수들을 이용해 직접 augmentation을 구현하여 데이터셋의 수를 늘리는 방식을 시도하였다. 1번 class의 경우, 특별한 경향성이 없는 경우가 대부분이라고 판단했기에, augmentation 없이 동일한 이미지들을 한 번, 좌우반전한 이미지들을 한 번 추가했다. 2번 class의 경우, 이미지들을 상하반전하여 두 번 추가하였다. 3번 class의 경우, 3번 class를 명확하게 인식하는 것이 우선이라 판단했기에 augmentation 없이 동일한 이미지들을 두 번 추가하였다. 학습 데이터를 조정한 이후, scikit-learn에서 기본적으로 제공되는 SVC를 이용하였다. 사용된 파라미터는 scikit-learn의 GridSearchCV를 이용해 찾았으며, 해당 코드는 ipynb의 Appendix에 주석처리하여 첨부하였다.

Augmentation의 결과로 얻은 csv 파일을 기존의 svm 모델로 얻은 파일과 비교해 본 결과, Augmentation을 거친 모델은 2번 class를 1번 class로 오분류하는 경우가 많이 줄어들었으나, 3번 class를 오분류하는 새로운 오류가 생겼음을 확인할 수 있었다. 두 모델의 단점을 상호보완하기 위하여, VotingClassifier라는 이름의 클래스를 직접 정의하였다. 앞서 확인할 수 있었듯, svm 모델은 3번 class를 인식하는 데에 비교우위를 가졌고, augmentation을 거친 모델은 전반적으로 2번 class를 인식하는 데에 비교우위를 가짐을 확인할 수 있었다. 두 csv파일의 차이를 알아내는 함수와, 서로 다르게 판단한 이미지들을 시각화한 코드는 ipynb 파일의 Appendix에 첨부하였다.

이에 따라, svm 모델이 3번이라고 인식한 사진은 3번으로, svm 모델이 1번이라고 인식했으나 augmentation 모델이 2번으로 인식한 사진은 2번으로, 나머지 사진은 augmentation 모델의 판단을 따르도록 VotingClassifier를 정의하였다. 이 결과 얻은 csv 파일이 가장 높은 score를 얻은 'voting3_debug.csv' 파일이다. 가독성을 위하여, 동일한 파일을 'problem1_2022_17611.csv'이라는 이름으로 첨부하였다.

프로젝트 진행 결과, 가장 아쉬움이 남았던 부분은 시간상의 이유로 더 다양한 augmentation 기법을 적용해 보지 못했던 점이다. 첨부한 ipynb 파일에서는 랜덤성을 배제하기 위하여 실제 2번 class에 사용된 상하반전 두 번으로 코드를 작성했으나, 실제 프로젝트 진행 과정에서는 random 모듈을 불러와서, 상하반전, 좌우반전, 30도 회전, -30도 회전의 4가지 옵션 중에서 2번 랜덤하게 선택하여 augmentation을 진행하였다. 해당 코드는 ipynb 파일의 Appendix에 주석처리하여 첨부하였다.Random 모듈을 사용하면 사람이 직접 선택하는 것보다는 다양하고 적절한 augmentation이 적용될 것이라고 예상하여 진행한 것인데, 프로젝트 종료 후, 16번의 경우의 수를 모두 적용해 보며 검토한 결과, 실제 사용된 augmentation은 상하반전 두 번이라는 것을 알 수 있었다. 반전과 회전을 모두 사용하는 방식으로 augmentation을 진행했다면, 더 정확한 모델을 얻을 수 있었을 것 같아서 아쉬움이 남는다. 추후에 더 다양한 방식의 augmentation을 적용해 보며 가장 정확한 모델을 찾아보고 싶다.

또한, 진행 결과 성능이 부족하여 끝까지 채택되지는 못했지만, 이미지의 특성을 보다 잘 추출하기 위하여 수업 시간에 배운 Sobel filter를 이용하여 학습 데이터셋을 조절해 보기도 하였다.

Sobel filter를 적용하여 얻은 feature들을 이용하여 학습을 시켜 본 결과, 각 class의 특성을 보다 잘 추출하여 성능이 증가할 것이라는 예상과 달리, 일반 svm보다 떨어지는 성능을 보여주어 실제로 사용하지는 못했다. 추후, filter를 이용하여 동일한 프로젝트를 사용하여, 예상과 달리 성능이 떨어졌던 이유를 다시 한 번 찾아내 보고 싶다.

Problem2. Mechanical MNIST Regression

Mechanical MNIST regression 문제의 경우, scikit-learn에서 제공되는 SVR(Supportive Vector Regression)과 NuSVR(파라미터 Nu를 이용해서 사용되는 supportive vector의 수를 설정할 수 있는 supportive vector regression)을 이용하여, SVR와 NuSVR 모두에 대하여 scikit-learn에서 제공되는 GridSearchCV를 이용한 하이퍼파라미터 튜닝으로 최적의 모델을 찾는 과정을 병렬적으로 진행하였다. GridSearchCV의 경우, 파라미터 값들의 범위를 조정하며 최종적으로 10번 이상의 많은 튜닝 과정을 거쳤기에, 제출한 ipynb 파일에서는 직접 튜닝이 실행되지 않도록 모두 주석 처리를 하여 첨부하였다. 또한, 프로젝트 진행 초반의 튜닝 과정에서 일부 유실된 코드들이 있어, 기록을 남기기 시작했던 NuSVR의 후반 13번과 SVR의 후반 5번의 코드들만 첨부하였다.

튜닝 과정에서 score의 기준은 negative mean squared error로 설정하였다.

파라미터 튜닝 결과, NuSVR에 사용되는 최적의 파라미터들은 다음과 같았다.

```
Fitting 5 folds for each of 27 candidates, totalling 135 fits
best hyperparameter: {'svr__C': 289, 'svr__gamma': 1e-07, 'svr__kernel': 'rbf', 'svr__nu': 0.725} best score:  15.154150389467603
3.747896031913582
0.9407363896451317
```

파라미터 튜닝 결과, SVR에 사용되는 최적의 파라미터들은 다음과 같았다.

```
Fitting 5 folds for each of 27 candidates, totalling 135 fits
best hyperparameter: {'svr__C': 121, 'svr__epsilon': 0.0025, 'svr__gamma': 1.2e-07, 'svr__kernel': 'rbf'} best score:  15.333030559169057
3.7391352219809257
0.9410131264611675
```

NuSVR의 최적의 파라미터들을 사용하한 파일이 kaggle에 제출한 ‘nusvr_tuned_final.csv’파일이며, SVR의 최적의 파라미터들을 사용한 파일이 kaggle에 제출한 ‘svr_tuned_final.csv’파일이다. 각각 3.47132의 private score와 3.72974의 public score, 그리고 3.45220의 private score와 3.71972 의 public score를 얻었다.

프로젝트 진행 결과, 가장 아쉬움이 남았던 부분은 validation 과정을 생략한 상태로 최종 csv 파일을 제출하지 못했다는 점이다. test_train_split을 이용해서 validation set을 만드는 과정을 생략하고, 주어진 test data를 전부 사용한 결과, 동일한 파라미터들을 사용하더라도 비교적 더 낮은 RMSE가 나오는 것을 확인할 수 있었다. 이는 학습할 수 있는 데이터의 수가 vallidation을 만든 경우보다 더 많았기 때문으로 추측되며, 당연한 결과라고 볼 수 있다. 그러나, 제출용 최종 파일을 만드는 과정에서, 마감 직전까지 계속해서 튜닝을 진행하느라 이 과정을 깜빡 잊고 파일을 제출하였다. 이 과정에서 보다 부정확한 모델이 생긴 것 같아 아쉬움이 남는다.

또한, 진행 결과 성능이 부족하여 끝까지 채택되지는 못했지만, SVR에 사용된 kernel을 다양하게 바꿔 보지 않고 rbf 하나로 고정한 채로 처음부터 끝까지 튜닝을 진행했던 점에서도 약간의 아쉬움이 남는다. 수업 시간에 배운 kernel trick의 내용을 참고해 보았을 때, Gaussian Kernel을 사용하는 Radial Basis Function의 경우, 이론적으로 무한대의 nonlinear feature map을 사용하는 것과 같은 효과를 낸다는 점을 참고하여, rbf가 가장 좋은 성능을 내는 kernel function일 것이라고 믿고 다른 함수는 사용해 보지 않았다. 그러나, 다양한 여러 가지의 kernel 함수들을 사용해 튜닝을 진행해 보았으면 더 적합한 모델을 찾을 수 있었을까 아쉬움이 남는다. 추후에 기회가 있다면, 다른 kernel 함수들 또한 사용하여 최적화를 진행해 보고 싶다.