Homework 4 Report

학번: 2015020908

이름: 유성민

학년: 3

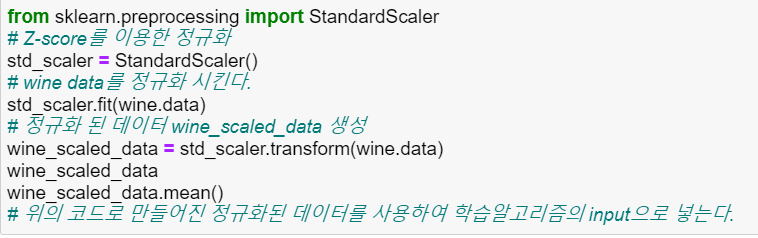
학과: 소프트웨어

**[1] 수업시간에 배운 알고리즘들의 파라미터를 최적화[10점]**

스크린샷이(가) 표시된 사진

자동 생성된 설명1. Data standardization

Data의 통계수치를 보면 데이터의 분포를 대략적으로 추론해볼 수 있습니다.

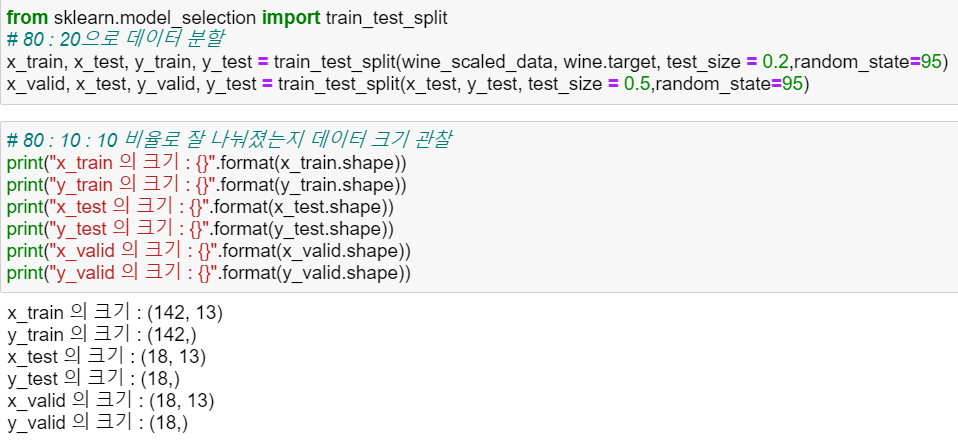
대부분의 feature들은 대개 0.5 ~ 15정도의 값을 가지지만 proline과 magnesium의 경우 min과 max를 보면 다른 feature들의 데이터 분포와 상당히 차이나는 것을 볼 수 있습니다. 따라서 데이터들을 같은 시점에서 비교하기 위해, 데이터 분포 scale 차이를 줄이기 위해 z-score를 이용하는 standardization이 필요하다고 판단되었습니다. 이를 통해 얻을 수 있는 효과는 대부분의 알고리즘들이 정규분포를 이루는 데이터임을 가정하기에 다양한 알고리즘에서 더 좋은 성능을 낼 수 있고, 이상한 결과를 초래하는 것을 방지할 수 있습니다.

위의 코드를 통해 기존의 wine데이터를 정규화 시키고 이 데이터를 통해 알고리즘들을 학습시켰습니다. 확실히 하기 전보다 성능이 훨씬 좋아졌습니다.

2. Data Split

훈련과 검증을 위한 데이터 샘플 split 방식은 다음과 같다.

* 총 주어진 샘플 중 80% 를 training 데이터로 구성하여 훈련시킨다.
* 10% 를 validation 데이터로 구성하여 하이퍼파라미터를 최적화 시킨다.
* 최종적으로 남은 10%를 test 데이터로 구성하여 성능을 보고서와 주피터 노트북에 표시한다.

hold-out 방식은 수집한 데이터를 90 : 10의 비로 분할 하고, 90의 데이터를 다시 90:10으로 분할하여 총 80 : 10 : 10의 비를 가지는 데이터로 나누는데 80은 training set, 10은 validation set, 그리고 마지막 10은 test set으로 사용됩니다.

wine 데이터의 경우 DESCR에 178개의 sample이 있다고 되어있습니다. 그래서 데이터 분할 후 제대로 나뉘었는지 개수를 관찰하기 위해 shape를 사용하여 각각의 크기를 보니 178 \* 0.8 = 142.8, 178 \* 10 = 17.8 잘 나뉜 것을 볼 수 있었습니다.

3. Find Optimal Hyper parameter

각각의 알고리즘에 대해서 정확도가 가장 높은 하이퍼파라미터를 찾는 과정에 대해 설명하고 해당 결과에 대해 EDA한 시각화 자료와 통계량 등을 이용하여 해석한다.

1. Knn
   1. 수업시간에 배운 metric 들 중 최소 두 개 비교
   2. 이웃의 개수 k 를 최소 3개 비교

3-1. KNN algorithm

1)개요

metric: Manhattan distance, Euclidean distance

K: [1,10] 내의 정수

성능측정 지표: 예측 값과 실제 값의 일치 율 (y\_pred == y\_test).mean()

2)알고리즘 작성코드

K를 1부터 10까지 1씩 증가시키면서 metric =1(Manhattan distance), metric = 2(Euclidean distance)

을 비교했습니다. 그리고 각각의 k와 metric의 조합에서 예측 값 실제 값을 비교하여 얼마나 일치했는지를 확률로 나타내어 점수로 매기고 score리스트에 추가시켰습니다.

test\_estimator는 training data으로 학습시키고, validation data를 이용하여 성능을 측정하여 어떤 파라미터에서 제일 높은 일치율을 보였는지 찾는 함수이고, training\_estimator는 training data을 이용해서 학습과 성능을 측정하는 함수입니다.

스크린샷이(가) 표시된 사진

자동 생성된 설명3) 해당 결과에 대해 EDA한 시각화 자료와 통계량 등을 이용하여 해석

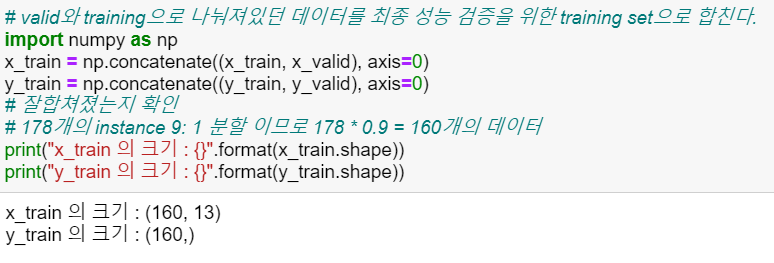
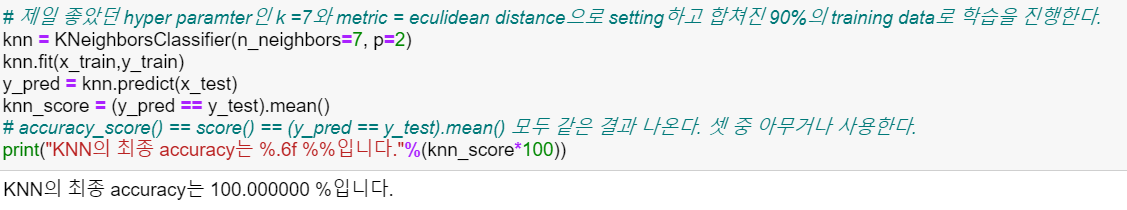
<EDA한 시각화 자료 및 통계량>

트레이닝 에러의 경우 K=1일 때 0인데, 실제로 그렇게 나오는 것을 볼 수 있었고, K가 커질수록 정확도의 증감을 반복하지만 대체적으로 감소하는 경향을 보였습니다. Manhattan distance를 사용한 경우가 정확도가 높았지만, validation set으로 실제 평가를 한 결과 두 metric이 굉장히 비슷한 그래프를 가지고 있습니다. 두 정확도 95%밑 구간에서 최고 정확도를 기록한 hyper parameter로는 [k=1, metric = 1], [k=4, metric=2], [k=6, metric=1], [k=7, metric =1,2], [k=9,10, metric = 2]가 있었습니다. 하나만 나올 줄 알았는데 데이터의 개수가 적고 표준화를 거친 후 여서 그런지 선정할 hyper-parameter의 후보가 많았습니다. 그 중에서 k=7일 때 두 metric이 최고 정확도를 기록했고, metric =1이 3번, metric = 2가 4번 최고 정확도를 기록했으므로 저는 k는 7로 선택하였고, metric은 2로 선택하여 최종적인 성능평가를 진행했습니다.

4) 최종 성능 평가

본래 wine data의 90%로 학습시키고, 분리시켰던 10%를 test set으로 사용하여 측정하기 위해 나눴던 validation set과 training set을 합치는 작업이 필요합니다.

original data가 178개 였고 그 중 90%는 160개이므로 정확히 합쳐진 것을 볼 수 있었습니다.



찾은 최적의 hyper parameter를 이용하여 test set으로 최종적 성능평가를 해보니 실제 레이블과 예측 레이블이 100%일치하는 굉장한 성능을 보여줬습니다. 확률적으로 1이 나오기 굉장히 힘든데, 이는 아마도 데이터가 10%만 사용하여 18개 정도밖에 되지 않아 그런 것으로 추측됩니다.

특이한 점은 data split시에 랜덤 시드에 따라 hyper parameter와 정확도가 굉장히 차이가 났습니다. 수업시간에 다 수의 알고리즘들이 데이터 분포를 가정하고 작성되었다고 말씀해 주셨는데, 실제로 training data의 분포, validation data의 분포 test data의 분포가 랜덤 시드에 따라 달라지기 때문에 부분적으로도(80:10:10 부분집합) 가정된 데이터 분포가 변화될 것이기에 이러한 현상이 발생되는 것 같습니다.

3-2. Decision Tree algorithm

Decision Tree

* 1. gini 와 entropy criterion의 비교

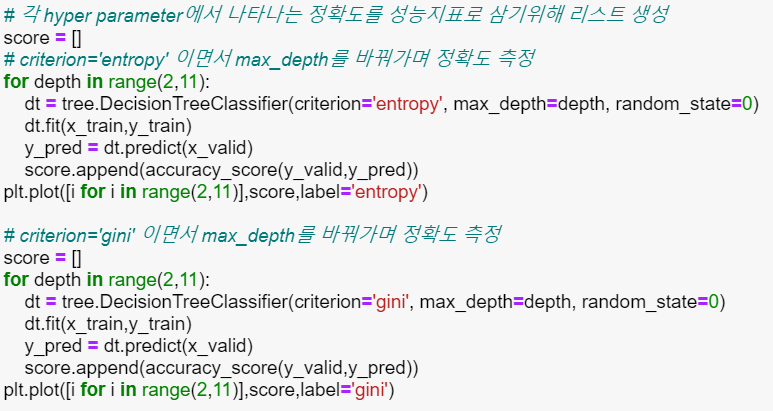
1) 개요

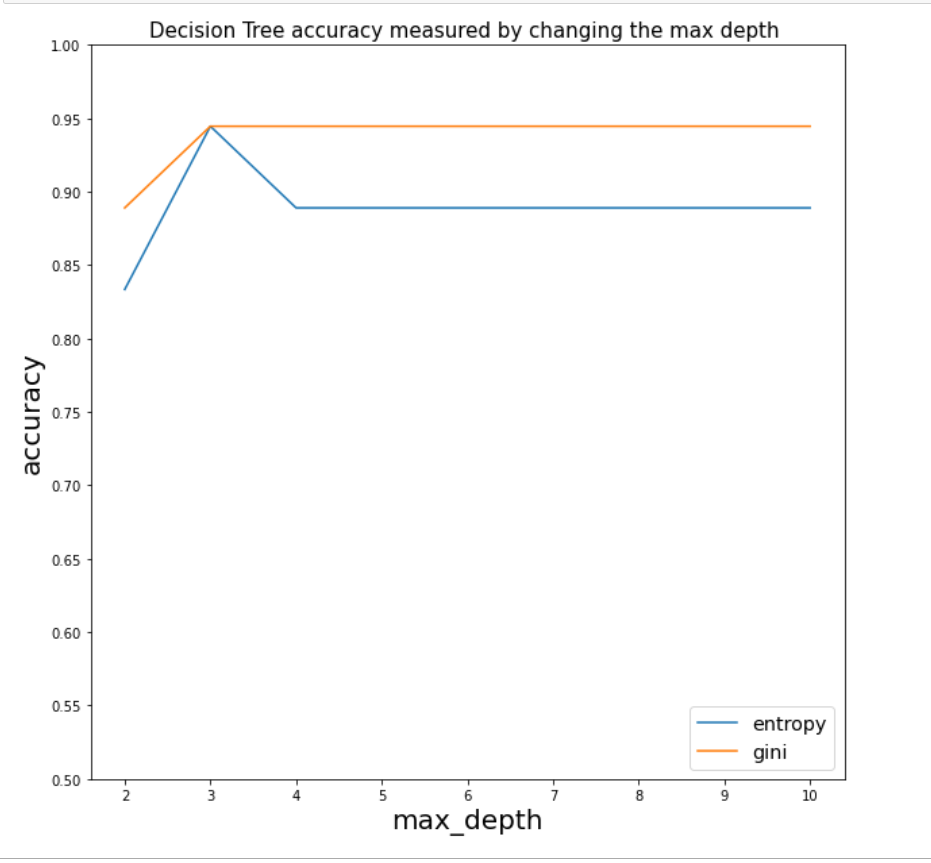
max depth: [2,10]사이의 정수 (overfit 방지를 위해서 어느 정도가 적당한지 보기위해서 추가했습니다.)

criterion: gini, entropy

성능측정 지표: 성능측정 지표: 예측 값과 실제 값의 일치 율

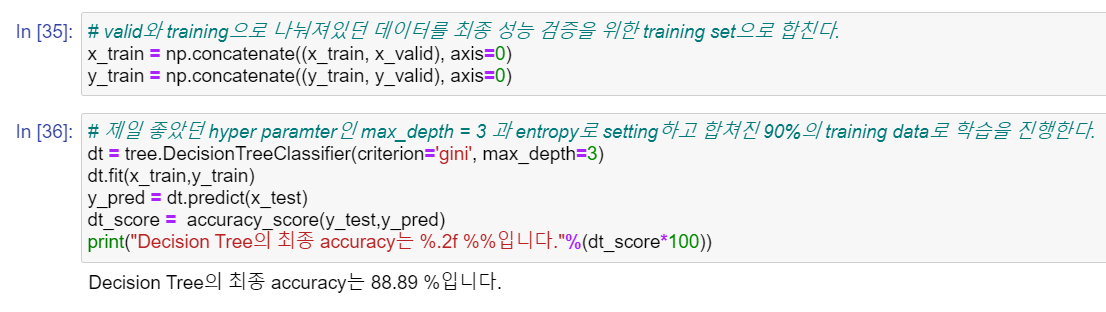
2) 알고리즘 작성 코드

max depth를 2부터 10까지 변화시키면서 criterion이 entropy일 때와 gini일 때의 각각의 예측 레이블과 실제 레이블의 일치율을 score 리스트에 추가하며 진행하였습니다.

3) 해당 결과에 대해 EDA한 시각화 자료와 통계량 등을 이용하여 해석

criterion으로 gini를 사용한 결과는 대체적으로 상당히 정확도가 높았고, 반면에 criterion이 entropy인 경우 max depth가 3일 때, 최고의 성능을 보이고 3이상으로는 정확도가 하락하였는데, decision boundary가 3개 일 때 가장 잘 분류되었고, 이후로 boundary를 추가하여 더 세분화 할수록 성능이 감소되었는데, 이는 4번째 boundary부터 class 0, 1, 2가 옹기종기 모여 있는 구간을 잘라서 서로서로 개수가 비슷하여 이러한 현상이 나타났다고 보여집니다. entropy는 해당 구역에 어떤 클래스가 될지가 명확할수록 엔트로피가 작아지고, 반대의 경우 엔트로피가 커지는데 구간을 자르기 전보다 자른 후가 오히려 엔트로피가 커진 것을 의미하기 때문입니다. 그래서 3번 분할 후 멈추는 것이 최적인 것으로 생각됩니다. 그리하여 최적의 hyper parameter로는 max depth가 3이고 대체적으로 상당히 높은 정확도를 보인 gini를 hyper parameter로 선택하였습니다.

4) 최종 성능 평가

찾은 hyper parameter를 사용하여 모델을 학습시키고 10%의 test set을 사용하여 최종적으로 성능을 평가하였더니 정확도가 88.89%가 나왔고, hyper parameter를 찾는 과정에서 보였던 정확도 보다는 낮지만 그래도 정확도가 높은 축에 속하는 것을 볼 수 있었습니다.

3-3 Ensemble Approach – Bagging, Random Forest

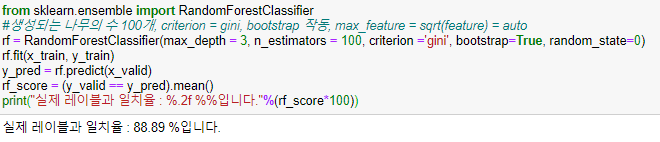
Ensemble (decision tree-based)

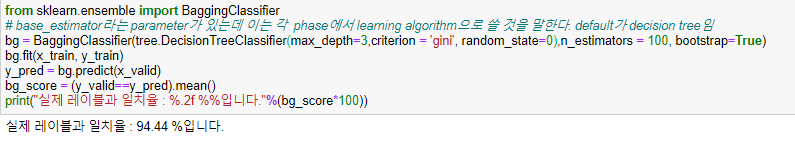
bagging과 random forest간의 비교

1) 개요

random forest는 전체 p개의 feature중 서로 다른 모양의 트리가 나오게 하여 보다 일반적인 결과를 얻는 알고리즘이고, bagging은 p개의 feature를 모두 사용하는 알고리즘입니다. 두 개의 공통점으론 ensemble approach에서 서로 다른 알고리즘 n개로 학습시키는데 이 서로 다른 n개를 decision tree로 고정하고 bootstrapping을 통한 training data를 re-sampling하여 학습한다는 것입니다. 그래서 위에서 진행했던 Decision Tree에서 제일 좋은 결과를 보였던 max depth 3과 criterion gini를 활용합니다.

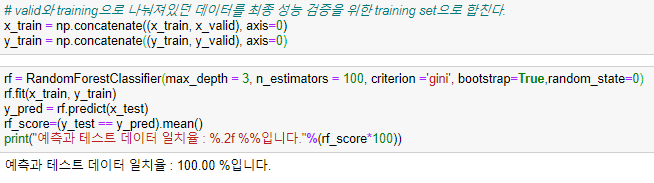
2) 알고리즘 작성 코드

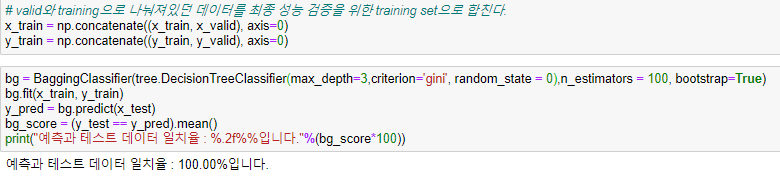
Random Forest

Bagging

두 알고리즘 모두 생성되는 decision tree의 개수를 100개로 설정하고 max depth를 3 그리고 criterion을 gini로 설정하고 학습을 진행하였습니다. validation set을 활용하여 성능을 측정하였더니 random forest는 88.89% bagging은 94.44%가 나왔습니다. bagging의 경우 feature의 scale 차이가 심할 경우 커다란 영향을 끼치는 feature가 decision tree 생성에 지대한 영향을 끼쳐 모두 비슷한 모양이 나오게 되어 일반적인 성능을 잘 내지 못하고 반면에 Random Forest의 경우 그 단점을 보완하기 위해 전체 p개의 feature 중 대략 m개()를 사용하여 bootstrapping을 진행하여 서로 다른 feature들이 반영된 다양한 모양의 decision tree를 생성해 내어 보다 다양한 데이터에 잘 반응하는 알고리즘이 된다고 수업시간에 배웠는데 학습 진행 전 표준화를 통해서 feature들의 scale을 감소시켜서 feature들 간의 scale차이를 감소시켰고, 지금 보면 random forest가 성능이 더 낮은데, wine데이터를 분류 하는데 영향을 많이 끼치는 feature가 오히려 m개 선택과정 중 빠지게 되어서 더 낮은 성능이 나오게 되었다고 생각됩니다. 이론과는 좀 다른 결과였습니다.

3) 최종 성능 평가

Random Forest

bagging

남은 10%의 test data를 활용하여 두 알고리즘의 최종 성능을 평가했더니 이번에는 random forest, bagging 모두 100%의 정확도를 보였습니다. 일반적으로는 random forest가 bagging보다는 다양성 있는 데이터에 잘 대처가능 하지만 wine데이터는 이미 정규화 되었고, 데이터 따라 m개를 선택한다는 장점이 오히려 단점이 될 수 있다는 것을 다시한번 깨우쳤습니다. 수업시간에 배운 알고리즘 선택과 데이터 조작은 data에 따라 application에 따라 맞게 선택해야 한다는 정보를 다시한번 일깨워 주는 계기였습니다. 제 결과에 한해서 hyper parameter로는 좀 더 반응을 잘한 bagging을 선택하는 게 맞습니다.

3-4. SVM

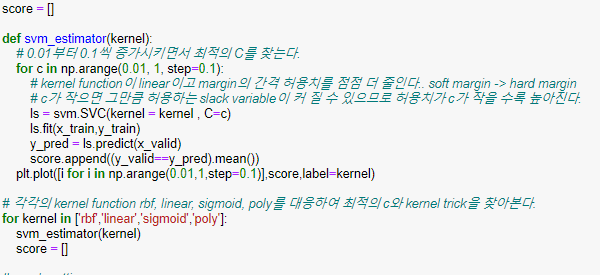
SVM

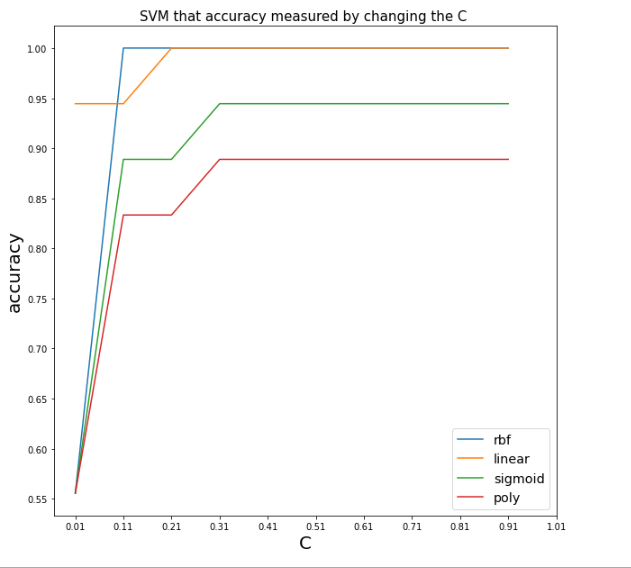
linear SVM과 kernel trick이 적용된 SVM간의 비교 (kernel 의 종류는 상관 없음)

1) 개요

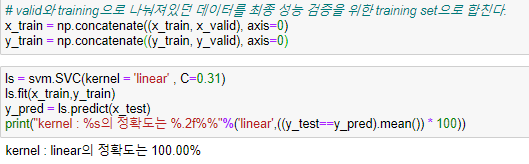
SVM은 margin을 최대화하는 것에 목적을 두는 알고리즘이고, 이를 좀 더 간단하게 구하기 위해 역수를 취해 margin의 역수 값을 최소화하는 과정을 통해 진행되고, 직선에 의해 완전히 분리되지 않는 경우 slack variable이라는 것을 두어 어느 정도의 오류는 허용하는 방식인데, 이 때 사용되는 hyper parameter로 C라는 것이 존재하고, 이 값이 크면 각 slack variable의 값들의 합이 작아져야 하므로 margin 허용범위가 줄어들고 이를 hard margin이라고도 한다. 반대의 경우 C가 작으면 그만큼 slack variable이 커도 되므로 margin 허용 범위가 넓어진다고 볼 수 있습니다. 이 경우 의 값을 최소화하는데 집중한다고 볼 수 있습니다. 왜냐하면 margin이 커질수록 slack variable로 그만큼 줄어들게 되어 최소화 하고자 하는 값으로 다가갈 수 있기 때문입니다. 이런 이유로 C값을 변화시켜가며 최적의 C를 찾고, kernel trick을 적용해보며 최적의 kernel trick을 찾아봤습니다.

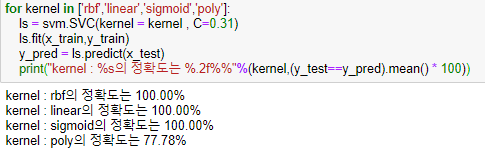
2) 알고리즘 작성 코드

함수 svm\_estimator는 서로 각기 다른 c를 [0.01,1]구간에서 0.1씩 증가시키면서 점점 margin을 줄였습니다. 이에 따라 각각의 kernel function들의 정확도를 잽니다.

3) 해당 결과에 대해 EDA한 시각화 자료와 통계량 등을 이용하여 해석

wine데이터는 margin의 허용범위가 작을수록 높은 정확도를 보입니다. 데이터가 넓게 펴져 있지 않고, 분류기에 가까이 붙어있음을 의미합니다. 옹기종기 모여 있는 것입니다. 이는 다른 알고리즘에서도 확인한 바이므로 다른 내용을 보면, 최적의 kernel function으로는 linear가 제일 좋은 정확도를 보였습니다. 정확도가 거의 바뀌지 않고 거의 1.00에서 일직선을 이루는데 상당히 높습니다. c는 0.31 이상부터는 모든 kernel function이 정확도가 rbf, linear는 1.00, sigmoid 0.95, poly 0.90으로 동일합니다. 최적의 hyper parameter로는 linear와 c >= 0.31을 고를 수 있겠습니다.

4) 최종 성능 평가

c는 0.31로 설정하고 kernel함수는 linear를 선택하여 10% test set으로 최종 성능을 평가한 결과 100%의 일치율을 보였습니다.

나머지 kernel function도 테스트에 활용했는데 경이할 정도의 정확도를 보여주었습니다. 수업시간에 배운 바로는 Neural Network 이전에는 가장 좋은 성능을 냈던 분류기라고 배웠는데 정말 그 사실에 방증 같은 결과가 나왔습니다.

이로써 4개의 분류알고리즘을 모두 살펴보았는데, 모두 성능이 너무 좋게 나와서 무엇을 선택해야 할지 조금 난감했습니다. 그래서 data split에서 random\_state인자를 지우고 10번 정도 돌려보고 제일 높은 성능을 보였던 것을 선택하기로 했습니다. 이런 이유는 데이터 어떻게 나뉘던 간에 잘 나눠주는 알고리즘이 최고기 때문입니다. 결과로는 SVM이 제일 안정적인 정확도를 보여주었습니다. 그래서 2번에 사용할 알고리즘으로는 SVM을 선택하게 되었습니다.

**[2] Cross-validation 을 이용한 최적화된 알고리즘 탐색 [10]**

[1] 에서 정의한 validation set을 이용하여 모델 선택을 하는 것이 아닌, 수업시간에 배운 두가지 cross-validation 방법인 k-fold cv와 loocv를 이용하여 최적화된 하이퍼파라미터와 모델을 찾아낸다.

최적화 시킬 모델은 [1]을 기반으로 하며, 가장 좋은 성능을 가진 모델과 하이퍼파라미터를 기반으로 보고서를 작성한다.

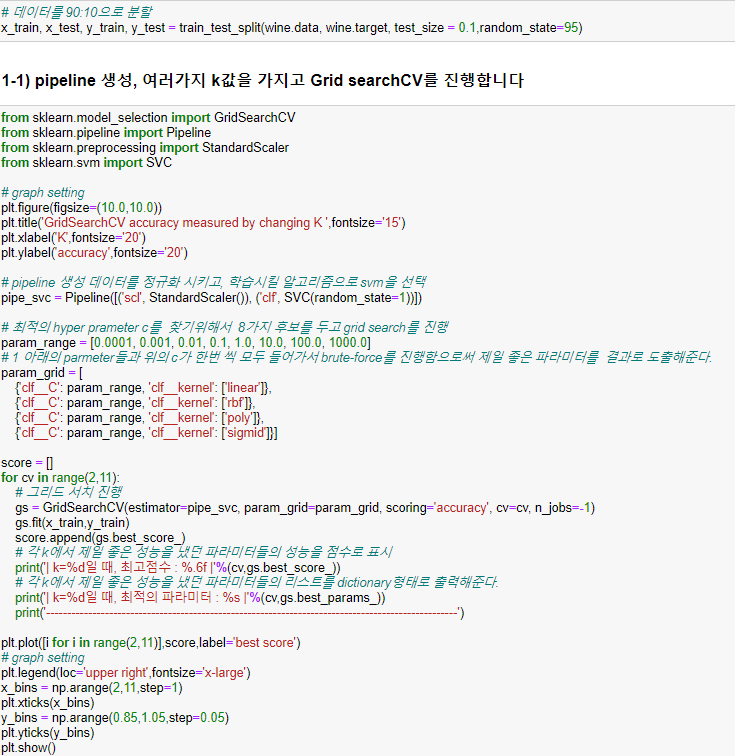
최종적인 성능은 [1]과 동일한 10%의 test set에서 측정하고, 나머지 90%를 cross-validation을 이용하여 모델 최적화를 진행한다.

1) 개요

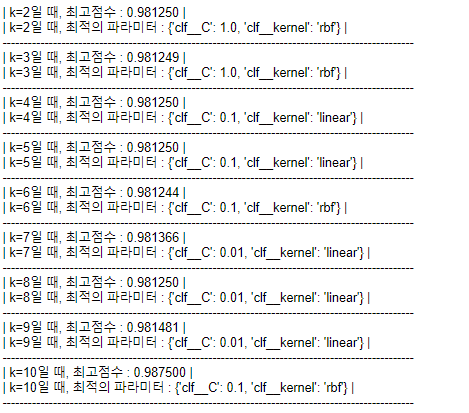
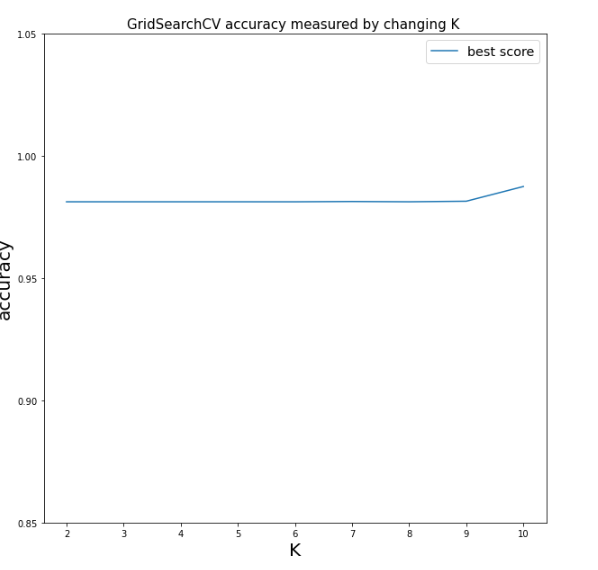
일반적으로 K=10으로 K-fold를 진행하지만, 시간이 없으면, k를 줄여가면서 평가하는데 k를 줄임으로써 발생하는 현상을 보기위해서 k를 2부터 10까지 증가시키며 cv를 진행하였습니다.

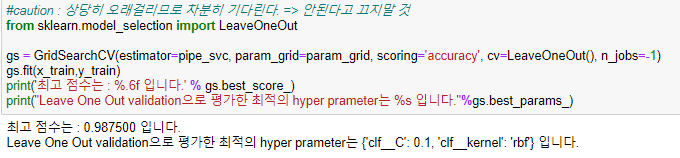
모든 알고리즘이 거치는 과정을 합쳐서 파이프라인이라고 하는데, 파이프 라인을 생성하고, grid search를 통하여 최적의 hyper parameter를 찾아보았습니다.

마지막으로 loocv를 통해서도 grid search를 통해서 hyper parameter를 찾아보았습니다.

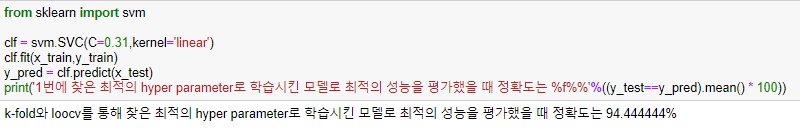
2) 코드

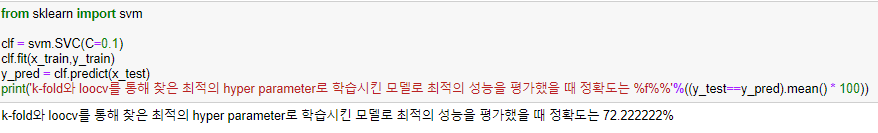
일단 데이터를 90:10으로 분할 합니다. 그리고는 데이터를 z-score로 표준화 시키고 학습알고리즘으로는 SVM을 선택하는 파이프라인을 생성합니다. 평가해볼 C로는 [0.0001, 0.001, 0.01, 0.1, 1.0, 10.0, 100.0, 1000.0] 8개와 Kernel function은 linear, rbf, poly, sigmoid를 해봄으로써 어떤 C와 kernel function의 조합이 최적의 성능을 내는지 봤습니다.

3) 해당 결과에 대해 EDA한 시각화 자료와 통계량 등을 이용하여 해석

rbf와 linear가 번갈아 가며 좋은 성능을 내고 이는 앞서 봤던 1번에서 SVM을 측정할 때와 비슷한 결과를 냅니다. 1번의 결과도 c >=0.31 에서 둘 다 정확도 1.0을 가졌기 때문입니다. k가 클수록 100% training set에 근접하므로 보다 더 정확한 측정을 한 것이라고 할 수 있습니다. 가장 높은 점수를 보인 것은 kernel rbf와 C = 0.1입니다. 1번에서 제가 선택했던 hyper parameter와는 사뭇 다릅니다. SVM은 training data가 적은 경우에도 높은 성능을 보입니다. k=2인 경우 90%의 절반을 training으로 쓰는데 이는 80개입니다. training data가 적은 경우 model이 학습을 아예 못할 수도 있는데(선형회귀의 경우) 이에 반해 적은 경우에도 SVM은 높은 성능을 보입니다. 다음으로 LOOCV를 사용하여 training 100%에 극단적으로 근사 시켜보면,

10-fold와 같은 결과가 나옵니다. 이로써 wine data를 사용할 때 SVM알고리즘으로 학습시키는 경우, 최적의 hyper parameter로는 kernel = rbf, C = 0.1로 하면 가장 좋은 일반적인 성능을 기대할 수 있습니다.

4) 최종 성능 평가

 1번에서 찾은 hyper parameter가 오히려 더 높은 성능을 보였는데, 이는 평가 시 사용한 C가 달려져서 그런 것으로 생각됩니다. 이로써 알 수 있는 점은 무조건 K-ford와 Grid Search를 하는 것이 아니라 데이터를 보고 어느 정도의 hyper parameter가 적당한지 체크한 후에, 찾은 구간에서 best를 찾기위해서 진행하는 것이라는 것을 다시한 번 복습하게 되는 계기였습니다. 2번의 경우 데이터를 평가하지 않고 임의로 10배씩 증가시켜가면서 만든 c들의 집합이고, 1번은 어느정도 오밀조밀 모여있다는 것을 알고있었기에, 일부러 C를 작게 설정하고 건너뛰는 구간도 작게 설정하여 그 세밀한 구간 중에서 best를 찾았습니다. 그랬더니 더 좋은 성능을 보이는 것을 볼 수 있었습니다.

-긴 보고서 읽어주셔서 감사합니다-