**Homework 4 Report**

**학번: 2015020908**

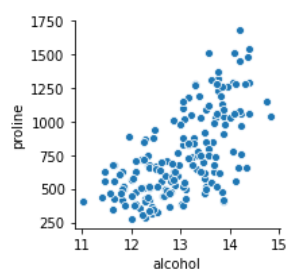
**이름: 유성민**

**학년: 3**

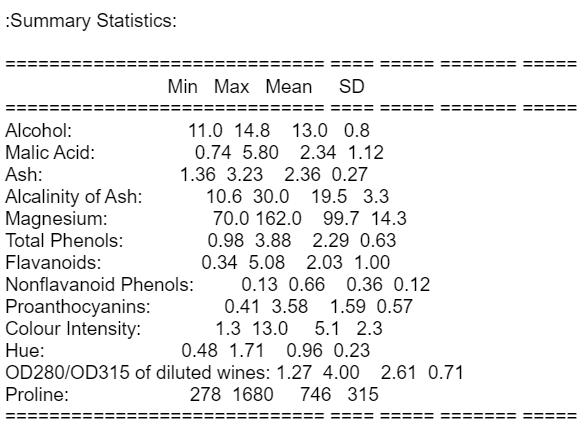
**학과: 소프트웨어**

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

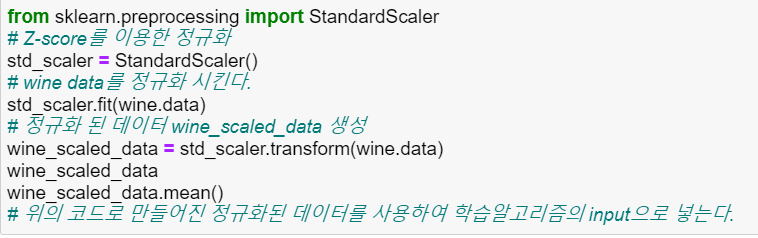
**텍스트이(가) 표시된 사진

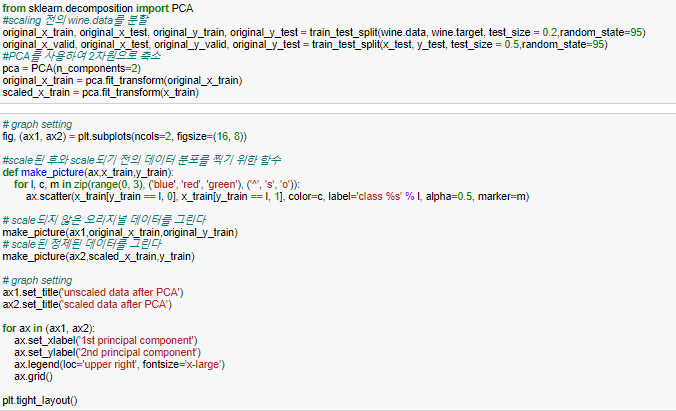
자동 생성된 설명1. EDA for recognizing Data Distribution**

**각 feature들의 상관관계를 봤는데 몇몇 쌍이 눈에 띄는 상관관계가 있었습니다. 대표적으로 proline – alcohol, flavanolds – total phenois 이러한 상관관계는 학습에 영향을 미칠 수 있습니다. 왜냐면 데이터가 상관관계가 없을 경우 선형 알고리즘들의 성능이 급격히 떨어질 수 있기때문입니다. 해석시에도 유용하게 쓰일 수 있고, 또한 여러 개의 feature가 있으면 좋은 점은 multiple feature와 single feature가 각각 최종적으로 p-value같은 지표를 보고 해석이 서로 다를 수 있기 때문입니다. multiple에서 slope의 의미는 다른 변수들이 고정돼 있을 때, 특정변수가 증가할 정도를 나타내는데, single일 경우 다른 변수들을 고려할 수 필요가 없기 때문에 서로 상이한 결과가 나올 수 있어 더 다양한 해석이 가능한 multiple의 경우가 더 좋을 수 있습니다. 하지만 이것은 단점으로도 작용할 수 있습니다. 왜냐하면 차원의 저주문제가 있기 때문입니다. 차원이 증가할수록 필요한 데이터는 기하급수적으로 증가할 뿐더러 모델도 더 복잡해지기 마련입니다. 현재 데이터의 분포를 보면 상관관계가 있는 변수들의 쌍은 선형모델을 쓰면 좋은 성능을 보일 것이고, decision tree나 ensemble approach에서는 piecewise한 축에 평행인 decision boundary를 만들며 나눠 가기 때문에 이 알고리즘들 에서는 나누기가 어려울 수 있습니다. svm의 경우 linear svm인 경우 잘 나눌 수 있을 것이지만 non-linear의 경우 kernel function을 어떤 것을 쓰느냐에 따라 차이가 날 수 있습니다. 이러한 이유 때문에 이번과제에서는 4개의 분류기를 모두 시험하기에 모든 feature를 반영하여 학습을 진행하는 것이 타당하다고 생각했습니다.**

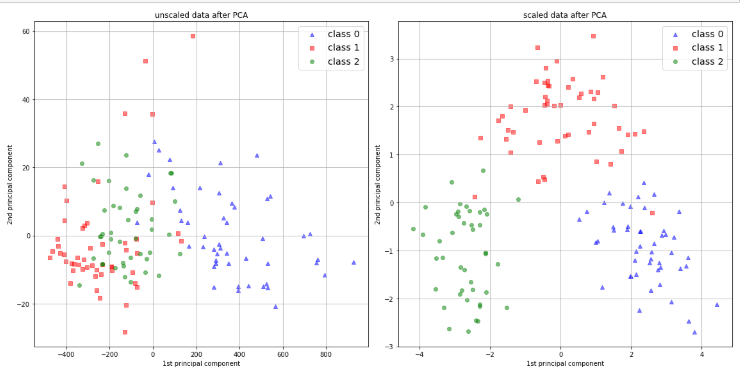
**2. Data standardization**

**Data의 통계수치를 보면 데이터의 분포를 대략적으로 추론해볼 수 있습니다.**

**KNN알고리즘의 경우 데이터의 크기 분포가 너무 차이 날 경우 상대적으로 Scale이 큰 데이터에 영향을 지대하게 받기에 잘못될 가능성이 있습니다. 때문에 표준화를 통하여 데이터를 정제할 필요가 있습니다. 현재 보면, 그래프의 크기가 작아서 현재 proline – alcohol pair의 경우도 데이터 스케일이 차이가 크게 나는 것을 볼 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. 이제 정규화의 효과를 한 번 보겠습니다.**

**차원을 축소하여 데이터의 분포를 보는 PCA기법을 사용하여 데이터의 분포를 보도록 하겠습니다. 위에서 각 Pair를 비교하니 솔직히 해석이 어려워서 설명하기가 나빴습니다. 때문에 보다 직관적으로 2차원으로 축소하여 데이터 분포의 운동을 가장 잘 표현하는 축(주성분)을 기준으로 정사영 시켜보면 전체데이터의 분포를 대략적으로 파악할 수 있습니다. 2차원으로 내려서 진행한 결과입니다.**

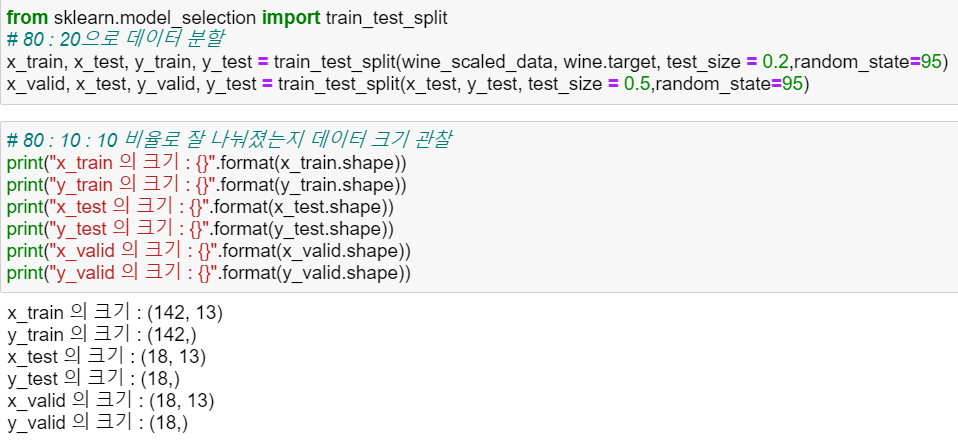
**왼쪽은 스케일하기 전의 데이터분포를 PCA한 것, 오른쪽은 스케일 후의 데이터 분포를 PCA한 것입니다. 엄청난 차이를 볼 수 있습니다. 데이터가 상대적으로 잘 나눠지는 것을 볼 수 있는데 더이상의 차원축소 등의 전처리없이 곧바로 진행해도 될 것 같은 결과가 나온 것을 볼 수 있습니다. 또한 왼쪽의 경우 overfit확률이 굉장히 높습니다. 데이터가 중구난방이기 때문입니다. 오른쪽은 overfit확률이 거의 없어 보입니다. 클래스 간의 군집이 서로 멀찍이 떨어져 있고, 특이한 데이터도 몇 개 안보이기 때문입니다. 데이터 분포와 통계치를 보면 knn에서 좋은 성능이 나올 것 같습니다. 이유는 데이터의 군집화가 잘 됐고, 테스트 데이터 또한 outlier의 가능성이 적기 때문에 training data와 고만고만 할 것으로 판단했습니다. 또한 SVM에서도 좋은 성능을 보일 것으로 판단되는데, class1의 경우 몇 개의 데이터가 다른 클래스의 군집을 침범하는 경우도 있는데 slack variable을 두어 약간의 여유를 허용하기 때문입니다. 정리하면, 현재 분포로 보았을 때, knn의 이웃 수는 많을수록 좋을 것 같고, decision tree나 ensemble approach에서는 작은 깊이(3-4정도), svm에서는 slack variable을 좀 크게 하되 적당하게 다른 군집에 너무 들어가지않게 허용해야 할 것으로 판단됩니다.**

**EDA결과 모든 Feature를 사용하는 것이 좋아 보이는 것이 여러 개 보여서 저는 모든 feature를 사용하기로 했습니다.**

**3. 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 80:10으로 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.**

**4. 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**

**4-1. KNN algorithm**

**1)개요**

**metric: Manhattan distance, Euclidean distance**

**K: [1,10] 내의 정수**

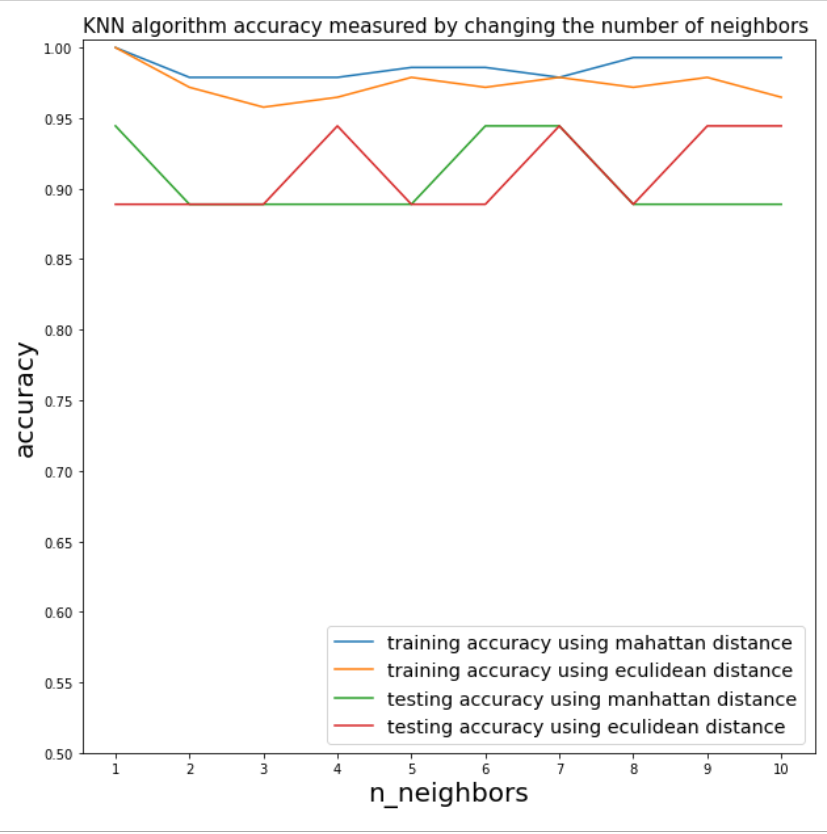
**성능측정 지표: 예측 값과 실제 값의 일치 율 (y\_pred == y\_test).mean()**

**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.**

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

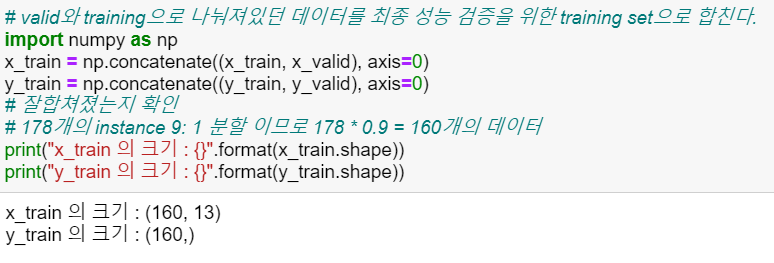
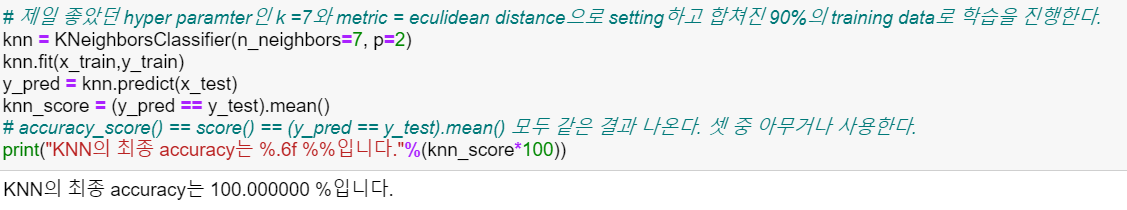
**<EDA한 시각화 자료 및 통계량>**

**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.**

**4-2. Decision Tree algorithm**

**Decision Tree**

* 1. **gini 와 entropy criterion의 비교**

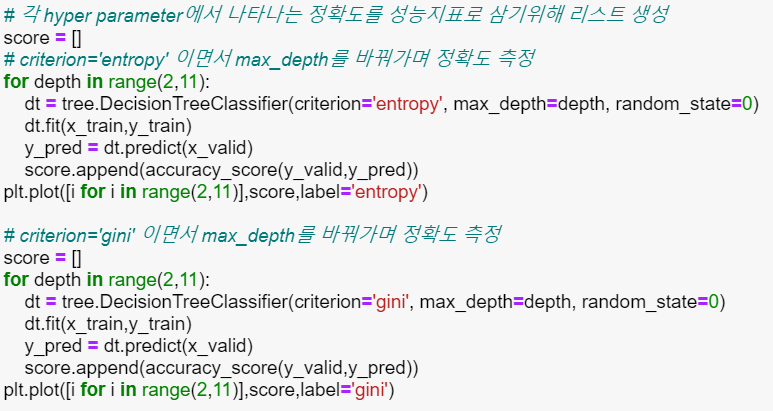
**1) 개요**

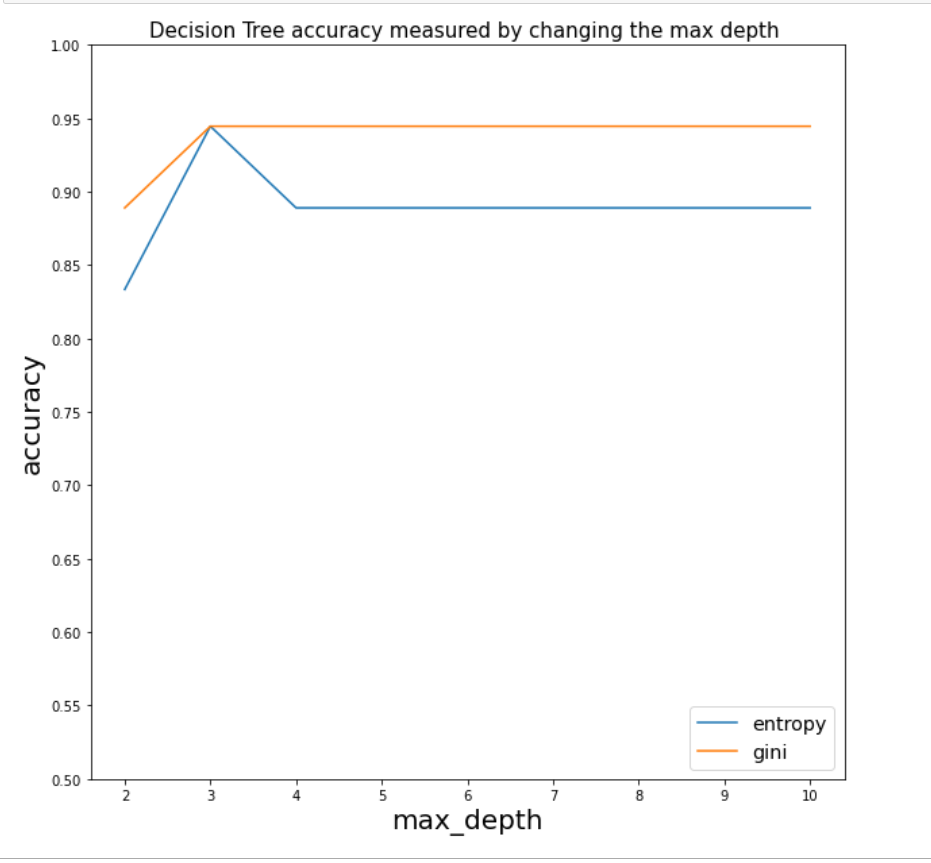
**max depth: [2,10]사이의 정수 (overfit 방지를 위해서 어느 정도가 적당한지 보기위해서 추가했습니다.)**

**criterion: gini, entropy**

**성능측정 지표: 성능측정 지표: 예측 값과 실제 값의 일치 율**

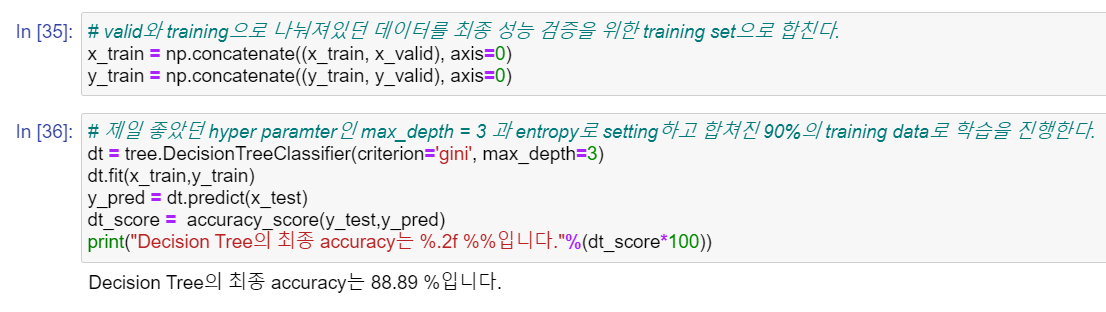
**2) 알고리즘 작성 코드**

**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.**

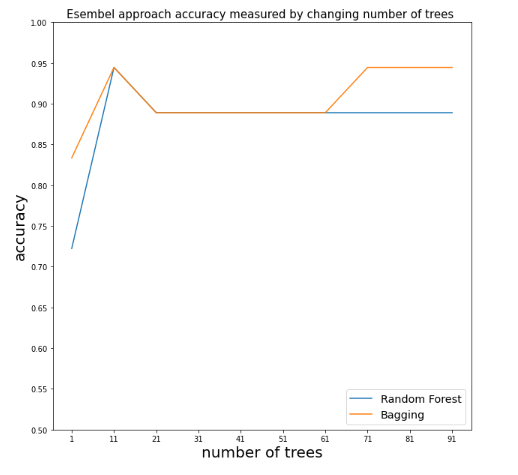
**3-3 Ensemble Approach – Bagging, Random Forest**

**Ensemble (decision tree-based)**

**bagging과 random forest간의 비교**

**1) 개요**

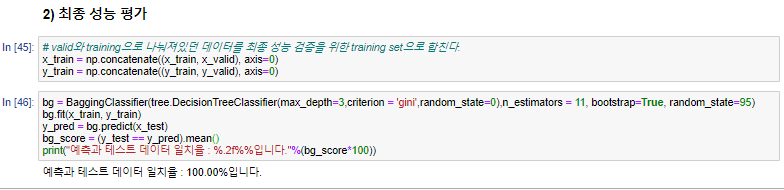
**random forest는 전체 p개의 feature중 m개(보통 sqrt(p))를 뽑아 서로 다른 모양의 트리가 나오게 하여 보다 일반적인 결과를 얻는 알고리즘이고, bagging은 p개의 feature를 모두 사용하는 알고리즘입니다. 두 개의 공통점으론 ensemble approach에서 서로 다른 알고리즘 n개로 학습시키는데 이 서로 다른 n개를 decision tree로 고정하고 bootstrapping을 통한 training data를 re-sampling하여 학습한다는 것입니다. 그래서 위에서 진행했던 Decision Tree에서 제일 좋은 결과를 보였던 max depth 3과 criterion gini를 활용합니다. 그리고 나무의 수인 n\_esitimator의 수를 바꿔가면서 성능을 측정해보겠습니다.**

**스크린샷이(가) 표시된 사진

자동 생성된 설명2) 알고리즘 작성 코드**

**각각의 알고리즘에서 사용되는 decision tree는 위에서 도출된 max\_depth 3을 사용하였고, 나무의 수를 1부터 100까지 늘려가면서 수행했습니다. 굉장히 그래프가 특이한데, 나무의 수가 11에서 최고성능을 기록했고, 구역 분할 기준인 criterion을 보자면 두 성능지표 gini의 경우 불순도를 나타내는 데 entropy보다는 해당 구역에서 클래스들이 차지하는 비율에 덜 민감하게 반응합니다. 즉, 어느정도 섞여도 entropy는 과민반응을 하고 gini는 반응은 하지만 미지근합니다. eda한 데이터 분포에서 클래스가 잘 분류 되었지만, 가끔 다른 클래스의 구역을 침범하는 샘플들이 있었습니다. 때문에 entropy가 더욱 민감하게 반응하여 overfit할 확률이 높습니다. 나온 지표를 보면 bagging이 random forest보다 안정적인 성능을 보였습니다. 이런 이유에 대한 제 견해는**

**일반적으로는 random forest가 bagging보다는 다양성 있는 데이터에 잘 대처가능 하지만 wine데이터는 이미 정규화 되었고, 그 말은 scale을 줄였기에 m개의 feature를 선택하는 것의 의의가 감소했음을 의미하고, 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) 최종 성능 평가**

**hyper parameter인 bagging 기법과 max\_depth = 3, n\_estimators = 11로 최종 학습을 진행하였더니 결과가 100%의 정확도를 보였습니다. 상당히 잘 분류된 것을 볼 수 있습니다. 이유는 스케일 하여 트리형성에 지대한 영향을 미치는 feature를 줄였고, 그 결과로 다양한 나무들이 형성되어서 이러한 결과가 나왔다고 생각합니다.**

**4-4. SVM**

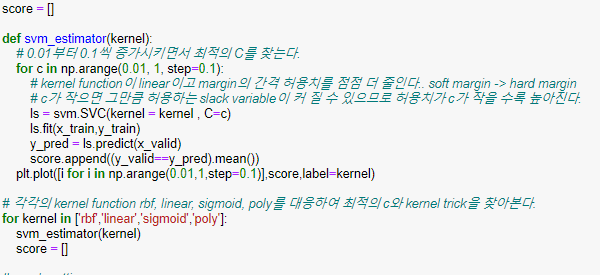
**SVM**

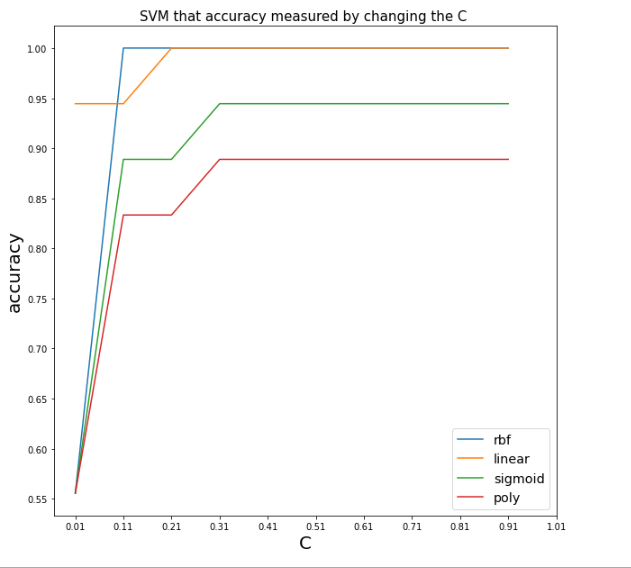
**linear SVM과 kernel trick이 적용된 SVM간의 비교 (kernel 의 종류는 상관 없음)**

**1) 개요**

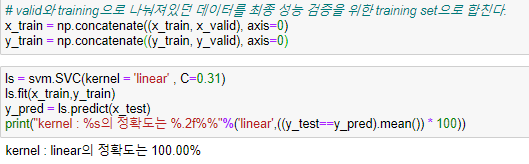
**SVM은 margin을 최대화하는 것에 목적을 두는 알고리즘이고, 이를 좀 더 간단하게 구하기 위해 역수를 취해 margin의 역수 값을 최소화하는 과정을 통해 진행되고, 직선에 의해 완전히 분리되지 않는 경우 slack variable이라는 것을 두어 어느 정도의 오류는 허용하는 방식인데, 이 때 사용되는 hyper parameter로 C라는 것이 존재하고, 이 값이 크면 각 slack variable의 값들의 합이 작아져야 하므로 margin 허용범위가 줄어들고 이를 hard margin이라고도 한다. 반대의 경우 C가 작으면 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.**

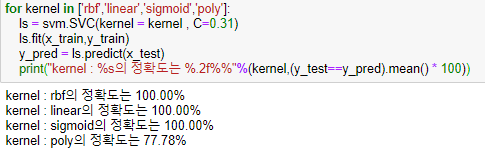
**2) 알고리즘 작성 코드**

**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.**

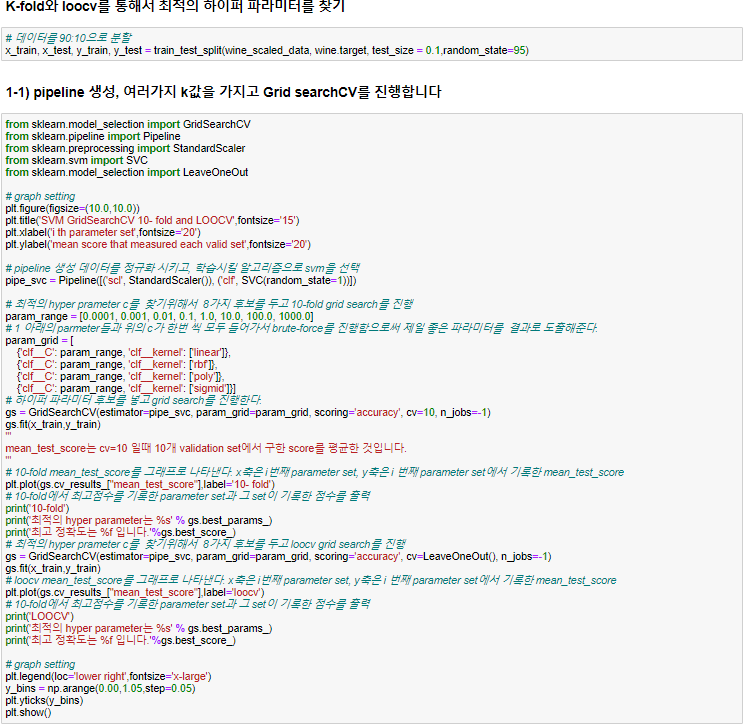
**최종적인 성능은 [1]과 동일한 10%의 test set에서 측정하고, 나머지 90%를 cross-validation을 이용하여 모델 최적화를 진행한다.**

**2-1. SVM**

**1) 개요**

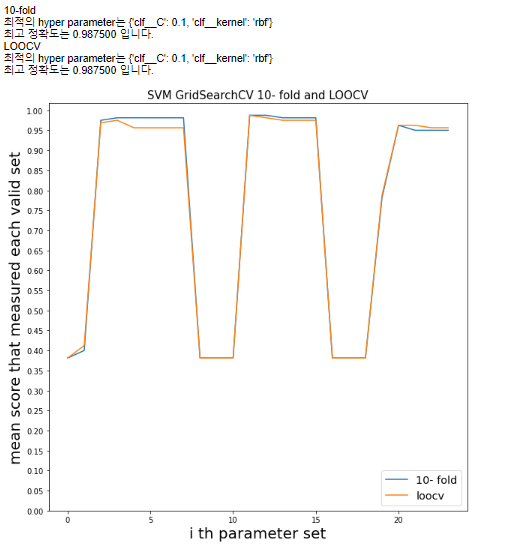
**10-fold, loocv와 GridSearchCV, pipeline이용**

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

**2) 코드**

**일단 데이터를 90:10으로 분할 합니다. Scale된 데이터와, svm알고리즘이 파이프라인의 인자로 들어가고 param\_rage는 slack variable의 가중을 나타내는 지표로 [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.**

**grid search는 10-fold로 한번, loocv로 한 번 총 두번 진행하였고, 조사한 바로는 최고의 성능을 낸 하이퍼파라미터 조합과 그때의 성능수치, 그리고 각 k개의 validation에서 나왔던 성능 평균을 나타낸 지표인 mean\_test\_score를 시각화 하였습니다. 즉 이 그래프의 x축은 i번째 하이퍼파라미터 조합 set, y축은 그 하이퍼 파라미터 셋으로 모델을 학습시켰을 때, 각각의 k개의 fold를 valid set으로 두고 성능을 측정한 뒤, 총 k개의 결과가 나오면 그 결과를 평균한 값입니다. 또한gridsearch는 훌륭한 지표를 나타내는 속성을 갖고있는데 best\_params와 bese\_score\_입니다. 모든 하이퍼 파라미터 통틀어서 제일 성능이 좋았던 하이퍼파라미터 집합과 그 것을 기반으로 성능을 측정한 점수를 의미합니다. 이로 인해 사실 시각화가 필요없으나 시각화 한 것을 보며 어떤 파라미터 셋에서 왜 이런결과가 나왔는지를 분석해보려 합니다.**

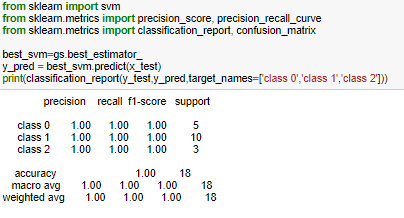
**3)결과 분석**

**svm의 경우 굉장히 들쭉날쭉한 성능을 보이는데 이는 10-fold와 loocv모두 마찬가지이고 사실상 같은 그래프로 보여집니다. 10분할과 178분할의 차이인데 10분할은 100% training set에 근접시키기에 둘이 굉장히 비슷한 결과가 나올 수 밖에 없습니다. 두 검증메소드 결과가 하나를 가리킵니다. c =0.1, kernel = rbf 이고 이 때의 성능의 정확도는 0.987500입니다 정말 높은 정확도를 보였는데 위에서 eda한 결과가 어느정도 맞아들어가는 것 같습니다. 들쭉날쭉한 이유는 다음 장의 그림은 I 번째 파라미터 셋을 나타냅니다. 7,8,9,10번째와 16,17,18,19번째에서 최악의 성능인데**

**텍스트이(가) 표시된 사진

자동 생성된 설명 공통점이라고 하면 c가 작을 때 즉, soft margin일 때 낮은 성능이었고 이는 slack variable의 가중치를 줄이고, margin 역수의 최솟값을 구하는데 집중한다는 얘기인데, 여유를 늘린다는 얘기입니다. 여유가 늘어나니, 오히려 성능이 떨어지는 것을 볼 수있었는데 특히 rbf와 poly에서 그렇다. 다른 kernel 함수들은 그렇지 않았습니다. rbf와 poly를 사용하는 경우 여유를 좁게 설정해야 좋은 성능을 보였습니다. 정리하자면 여유를 늘릴것이면, linear나 sigmoid를 사용, 줄일 거면 rbf나 poly를 사용해야 합니다.**

**4)최종 성능 검증**

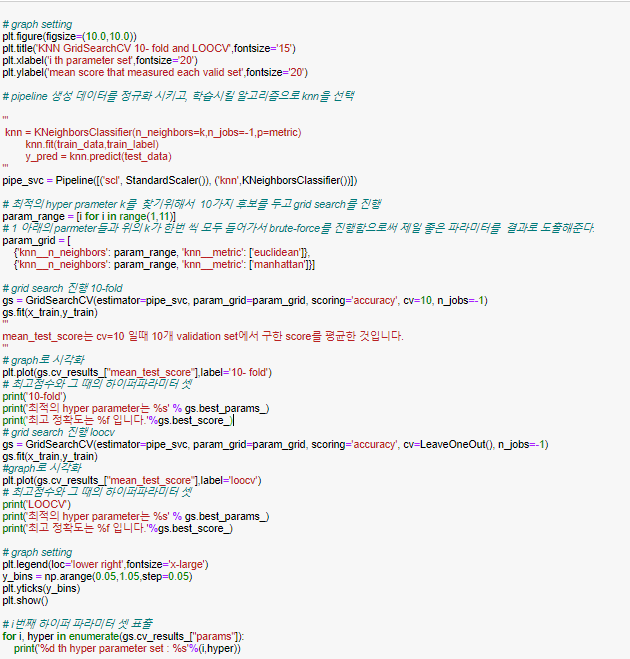
**grid search는 최고의 파라미터를 기반으로 모델을 만드는데 이것이 best\_esimator\_이고 이것을 기반으로 학습을 시킨 결과 모두 1.00이 나오는 굉장한 결과를 보였습니다. 앞에서 svm이나 knn에서 성능이 제일 좋을 것 같다고 했는데, 그것이 어느 정도 들어맞는 것 같습니다.**

**2-2) KNN**

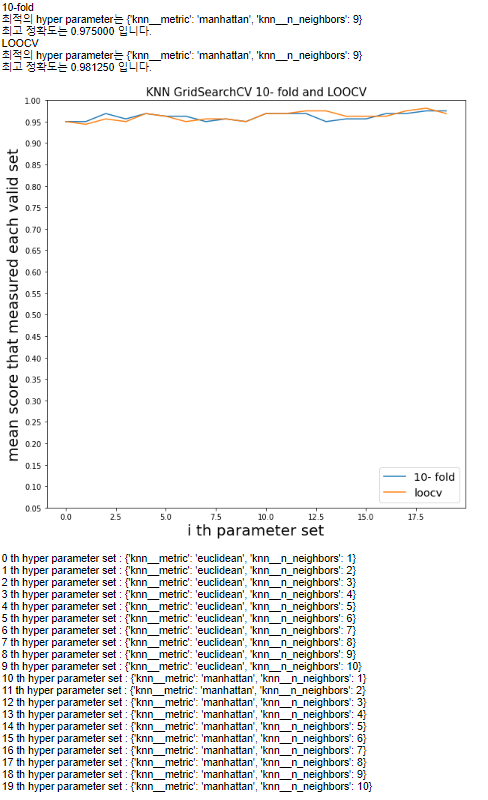
**1) 개요**

**10-fold, loocv와 GridSearchCV, pipeline이용**

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

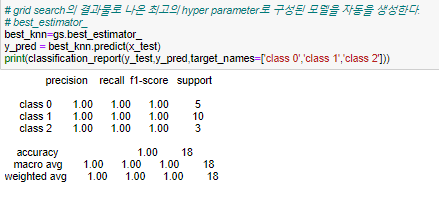
**2) 코드**

**사실 상 파이프라인에 들어가는 알고리즘과 평가할 파라미터 후보들만 바뀐 것일 뿐 SVM코드와 동일합니다. 평가할 Parameter set으로는 k는 [1 ,10]사이의 정수, metric으로는 유클리드와 맨하탄 거리가 있습니다.**

**3) 결과 해석**

**거의 모든 하이퍼 파라미터 셋에서 좋은 성능을 보였고, 이는 데이터 군집화가 잘 되어있고, 악영향을 끼치는 outlier가 거의 없기 때문에 그런 것으로 생각됩니다. 그리드 서치의 결과 k=9일 때, 메트릭은 맨하탄 거리일때가 나왔는데 10-fold와 loocv가 이번에도 같은 하이퍼 파라미터 셋을 가리킵니다. 앞서 예측했듯이 knn과 svm에서 좋은 성능을 낼 것이라고 예측 했는데 들어맞았습니다.**

**4) 최종 성능평가**

**knn역시 압도적으로 좋은 성능을 보였습니다. 표준화된 데이터가 knn에 적합하다고 했는데, 스케일이 큰 데이터가 거리에 영향을 미치는 것을 줄였기에 이에 해당하는 결과가 나온 것으로 판단됩니다. metric또한 별 차이 없었는데 y변화량 + x변화량, 점과 점사이 거리가 군집화가 되어있어 별로 차이가 안 나기에 두 메트릭이 비슷한 결과를 냈다고 생각됩니다.**

**2-3) Decision Tree**

**1) 개요**

**10-fold, loocv와 GridSearchCV, pipeline이용**

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

**2) 코드**

**코드는 svm, knn과 거의 동일하기에 더 이상의 코드 설명은 중복된 내용이어서 빼겠습니다. 바뀐 점은 파이프라인에 들어가는 알고리즘과 평가할 하이퍼 파라미터 후보입니다.**

**하이퍼 파라미터 후보로는 나무의 깊이([1,10]사이의 정수), 구역을 나누는 기준인 criterion은 gini와 entropy를 선정하였습니다.**

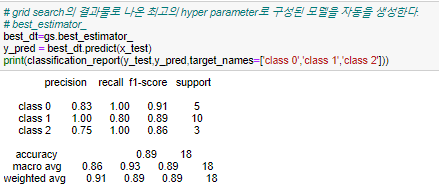
**3) 결과해석**

**거의 모든 하이퍼 파라미터 셋이 동일한 성능을 보였지만 특이한 조합이 있었습니다. 11번째 조합과 0번째 조합인데 공통점은 깊이가 1-2사이입니다. 각각 나누는 기준은 엔트로피와 지니로 다릅니다. 이 말은 나무 깊이가 작을 때는 분류기준이 적어 잘 분류가 안되는 것을 뜻하고 구역을 더 나눠야 한다는 뜻입니다. 1번에서는 gini가 entropy보다는 좋은 성능을 보였지만 여기서는 다소 특이한 결과가 나왔는데 loocv와 10-fold의 결과가 사뭇 다릅니다. loocv는 가끔 100%에 가까울 것 같지만 이상한 결과를 내놓기도 합니다. 둘 중 어떤 검증방식이 더 정확한 것인지는 데이터의 분포를 조금 볼 필요가 있습니다. 이 데이터에는 eda에서도 봤듯이 다른 구역을 침범하는 좀 특이한 데이터가 있었습니다. 그 데이터들이 validation set으로 들어가면 당연히 성능이 안 좋아 질 수 밖에 없고, 10 - fold에서는 다른 데이터들과 함께 들어가기 때문에 그 단점을 무마시킵니다. 때문에 저는 10 – fold가 좀더 정확한 것이라고 판단되었고 이를 기반으로 학습을 진행 시켰습니다.**

**10- fold의 경우 depth 7, gini, loocv의 경우 depth 7, entropy가 최고의 성능을 기록한 하이퍼 파라미터 셋이 되었습니다.**

**4) 최종 성능 평가**

**스크린샷이(가) 표시된 사진

자동 생성된 설명** **왼쪽은 loocv기반으로 생성된 것이고 오른쪽은 10-fold기반으로 생성된 것입니다. 제가 했던 예측이 틀렸는데 loocv의 결과가 더 좋았음을 볼 수 있었습니다. 이는 아마도 테스트 데이터에는 다른 클래스를 침범하는 특이한 데이터가 없었거나 더 다양한 데이터로 검증한 다시 말해 이론적으로 100%에 더 가까운 training data로 학습한 모델이 더 좋았음을 방증하는 것입니다. 또한 1번에서 했던 것과는 달리 loocv에서 최고의 하이퍼 파라미터로 entropy가 나왔는데 이는 좀 더 많은 데이터로 좀 더 민감하게 반응하여 구역을 보다 정확하게 분할해서 그렇다고 생각됩니다.**

**2-4) random forest**

**1) 개요**

**10-fold, loocv와 GridSearchCV, pipeline이용**

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

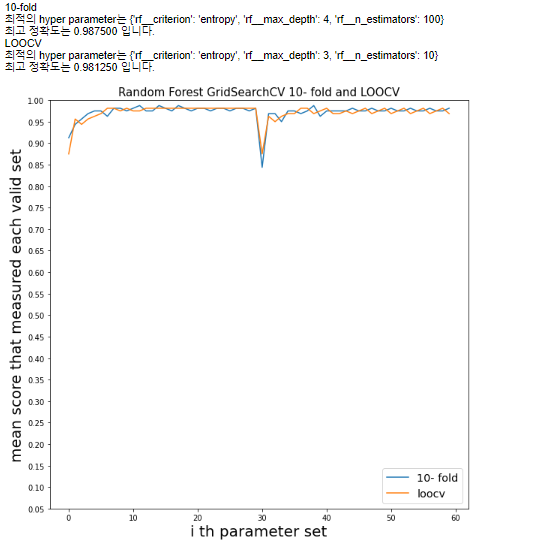
**2) 코드**

**평가할 파라미터 셋으로는 max\_depth와 criterion 그리고 나무의 수 입니다.**

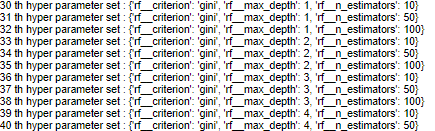
**각각 [1,10] 사이의 정수, [entropy, gini], [10, 50, 100]입니다.**

**나무의 수를 많이 만들고 싶었지만 굉장히 시간이 걸려서 줄였습니다.**

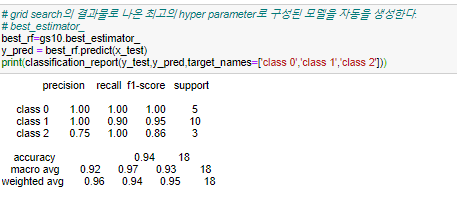
**3)결과 해석**



**결과를 보면 대체적으로 상당히 정확도가 높습니다.다만 30 40 사이에 한번 안 좋은 성능은 보이는 하이퍼 파라미터 셋이 있는데 이것을 분석해보고자 합니다.**

**30번대 초반의 하이퍼 파라미터 셋을 보니 나무가 많아도 max\_depth가 작으니 성능이 잠시 줄었습니다. 이는 decision tree에서도 봤듯이 분류기준이 너무 없어서 클래스 분류가 잘 안 됐음을 의미합니다. 나무의 수가 암만 많아도, 많은 데이터로 테스트해도 성능이 안 좋았습니다. eda할 때 보았듯이 decision boundary가 적어도 2개는 되야 클래스를 잘 분류할 수 있음을 보았습니다.**

**10- fold의 경우 엔트로피, 깊이 4, 나무의 수 100개일 때 최고점수를 기록했고, loocv의 경우 엔트로피, 깊이 3, 나무의 수 10일 때 최고의 점수를 기록했습니다. 데이터 뭉텅이로 테스트 할 때는 더 많은 나무가 필요했고, 데이터 하나를 테스트할 때는 나무가 10개여도 충분했음을 방증합니다. 깊이 또한 더 높은 위치에서 해도 잘 분류가 됨을 알 수 있었습니다. 하지만 10-fold에서 더 높은 점수를 기록하여 이를 기준으로 최종 성능을 평가하고자 합니다.**

**4) 최종 성능 평가**

**94%의 정확도를 보여주었고 대체적으로 클래스 2를 제외하고는 잘 분류되는 것같습니다. 아마 구분 기준이된 feature가 class2를 다른 클래스에 비해 비교적 잘 분류를 못하는 것으로 선정되었을 것으로 생각됩니다.**

**2-5) Bagging**

**1) 개요**

**10-fold, loocv와 GridSearchCV, pipeline이용**

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

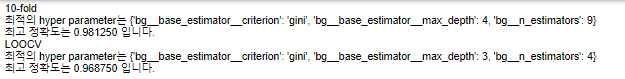
**2) 코드**

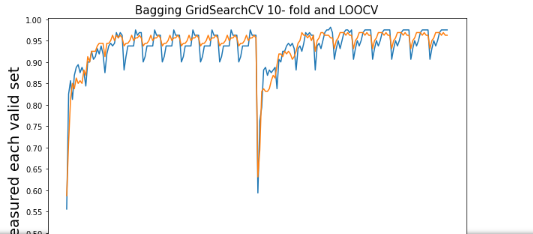
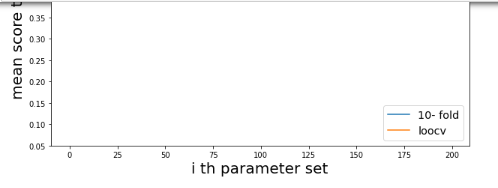
**평가할 하이퍼 파라미터 셋으로는**

**들어가는 트리의 깊이, 구분 기준, 나무의 수가 있고 random forest와 동일한 범위로 설정하였습니다. 각각 [1,10] 사이의 정수, [entropy, gini], [10, 50, 100]입니다.**

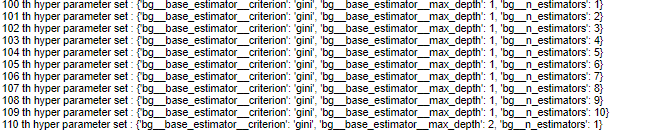


**3)결과 분석**

**결과가** **너무 길이서 그래프 잘린 점 양해 부탁드립니다.**

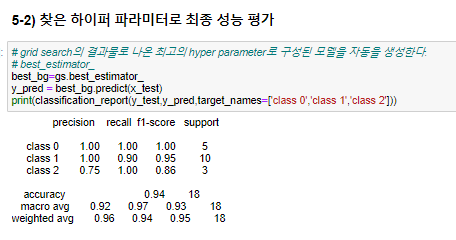


**100번째 파라미터 셋에서 성능이 낮아졌는데 이를 토대로 분석해보겠습니다.**

**random forest와 마찬가지로 이번에도 문제는 나무의 깊이였습니다. decision boundary가 최소 2개이상은 되야 분류가 잘 되는 데이터 분포를 이루고 있기 때문입니다.**

**최고점수는 loocv가 기록한 파라미터 셋 criterion : gini, depth: 3, 나무의 수 :4였습니다. 이 결과에서 주목할 점은 criterion이 random forest와 달려졌는데, 이는 랜덤 포레스트는 feature의 수를 m개로 설정해 데이터를 학습시키기 때문임으로 간주됩니다. 둘의 차이가 난이유는 랜덤포레스트가 m개를 선별하는 과정에서 구역을 나누는데 비교적 지대한 영향을 끼치는 feature가 끼어있어 더 민감하게 반응하는 entropy가 더 구역을 잘 분할하였던 것 같습니다. 하지만 모든 feature가 들어간 bagging의 경우는 물론 지대한 영향을 끼치는 feature가 여기서도 트리구축에 영향을 끼쳤을 확률이 높지만, 그 feature와 비교적 비슷한 scale을 가지는 feature들도 들어가서 그 영향을 random forest만큼 못 끼쳤기 때문에 덜 민감하게 구역을 나눠도 잘 분류되었을 것으로 판단됩니다. 만약 구역 분할에 지대한 영향을 끼치는 feature가 계속적을 선정되면 나무 모양도 비슷하고, 일반적인 성능을 기대하기 어렵기 때문입니다. 해당 feature만 지속적을 선택될 확률이 높았을 것이고, 해당 축을 기준으로 계속적인 분할이 이루어졌을 것이므로, 더 민감하게 분할하는 엔트로피에서 성능이 높았던 것 같습니다.**

**4) 최종 성능 평가**

**loocv로 최종 성능 평가를 진행하였고 random forest와 똑같은 성능을 냈습니다. 데이터가 scale되면 bagging과 random forest의 알고리즘은 별 성능 차이가 안 나는 것을 이번과제를 통해 여러 번 확인 할 수 있었습니다.**

**감사합니다.**