**Machine Learning & Data Mining, Spring 2020**

**Homework 4**

사이버보안학과

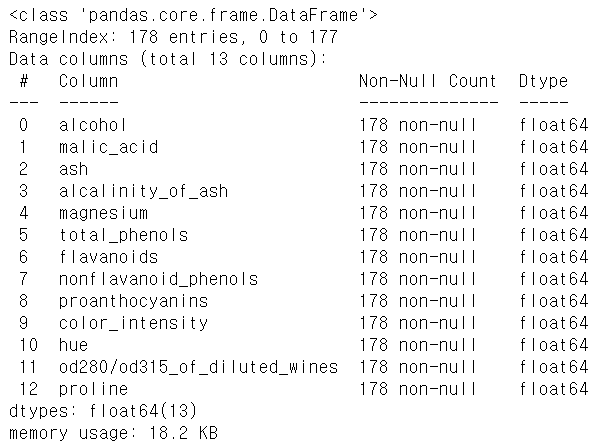
201620641

유 상 정

**과제목표**

* **수업시간에 배운 classification 모델들을 실제 적용시켜본다.**
* **데이터셋에 따라서 다양한 모델을 적용해야 한다는 것을 실제로 확인한다.**
* **다양한 모델과 파라미터 선택을 통해서 성능을 높이고 이에 대해 이해한다.**

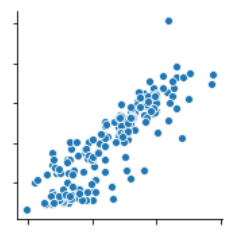
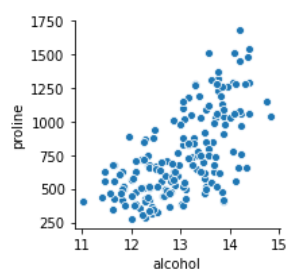
**[0] EDA**



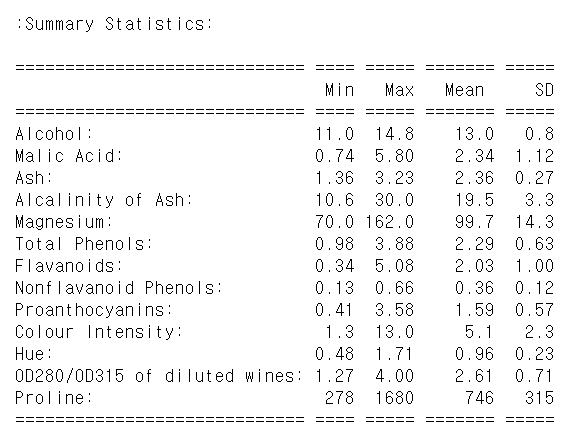
각 feature가 실수형 data이며 NaN값이 없음을 확인한다.



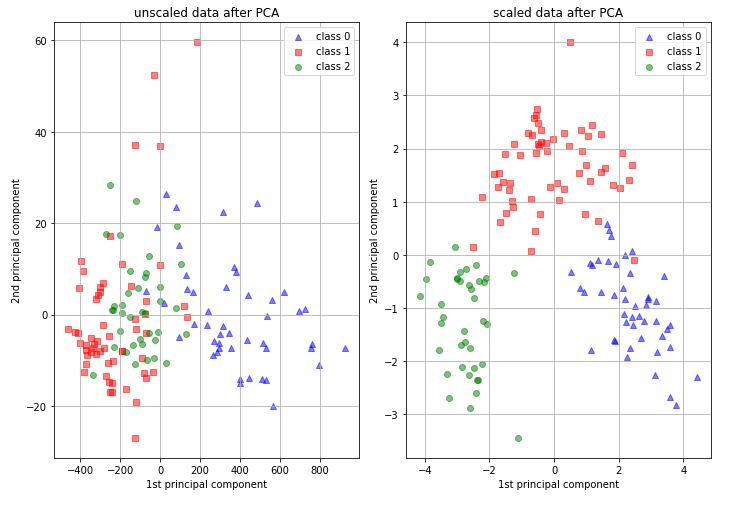
다음으로 모든 feature가 실수형이므로 pair plot으로 각 열의 조합에 따라 scatter plot을 나타낸다.

(x: Total Phenols, y: flavanolds)

각 pair를 확인해보니, proline - alcohol, flavanolds - Total Phenols 가 상관 관계를 보였다. 또한 다른 feature 들의 그래프에서는 크게 무슨 관계를 나타낸다고 확인할 수 없었다. 하지만, 많은 feature들을 사용하지 않고 4개만의 feature를 사용한다면, 잘못된 data가 나타날 수 있다고 판단하여, [1]과정에서는 모든 feature를 사용한다.



각 Feature에 대한 summary를 확인하였다. 우선 직관적으로 판단하기에 모든 feature가 수치형 변수이며 총 13개의 feature가 존재한다. 하지만, 눈으로만 판단해서는 가장 큰 영향을 주는 feature는 proline으로 보인다. 하지만 다른 feature들과의 값 차이가 너무 크므로 scaling을 해야 한다고 본다. 그래서 본인은 가장 큰 영향을 주는 feature를 n개 뽑아서 data의 class 분포를 확인해주는 PCA 함수를 사용하였다. 여기서 scaled data의 분포와 unscaled data분포의 차이를 본다. 위에서 상관관계가 나타난 feature가 4개를 뽑아서 확인한다. 즉, 차원 축소를 하여 주성분 분석을 한다.



Scaled train data와 unscaled train data를 PCA사용으로 얻은 것은 scaling data를 차원 축소를 진행할 때, 대부분의 data가 분류가 된다는 것이다. 즉, data 분석에 있어서 scaling 단계는 매우 중요함을 알 수 있다. 3개의 주요 feature를 뽑았다고 해도, 각 feature간의 data범위의 크기가 크다면 차원 축소를 하여도 잘못된 분류가 이루어 질 것이다. 물론, scaling은 거리기반으로 판단하는 알고리즘에서만 필요하고 다른 경우에는 크게 영향을 받지 않는다고 한다. 그리고 scaled data의 data분포를 통해, class 1인 data가 가장 많고, 다음으로 class 0 다음으로 class1 이 많음을 알 수 있다.



Wine data의 describe에서도 확인할 수 있다.

지금까지, Data의 분포나 각 feature의 관계를 보았다. 하지만, 판단하기에 버릴 만한 feature와 sample은 없다고 생각했다. 왜냐하면, NaN값도 존재하지 않았고, 크게 영향을 줄 만한 outlier도 없다고 판단했다. 다음으로 각 알고리즘에 대해서 train, valid, test를 진행한다.

**[1] 수업시간에 배운 알고리즘들의 파라미터를 최적화**

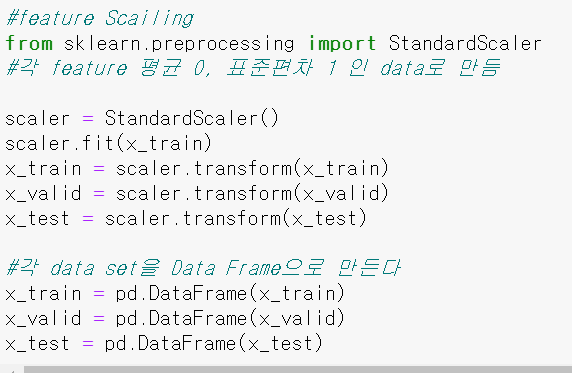
공통 사항

- train : valid : test = 80 : 10 : 10 비율로 data를 나눴다.

-train\_test\_split 함수의 random\_state값을 지정하여 data가 동일하게 뽑히도록 설정

**(1) KNN 알고리즘**

1-1. Feature Scaling



(feature scaling code)

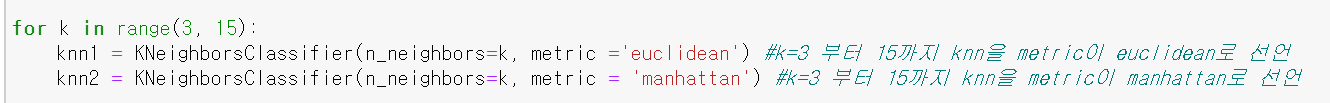
KNN알고리즘은 data scaling이 필수적이다. 왜냐하면 데이터의 크기 분포가 너무 들쑥날쑥하면 각 data의 거리를 구할 때, 상대적으로 data크기가 큰 feature에 영향을 크게 받게 되기 때문이다. 그래서 각 feature를 standard Scaler를 사용하여 평균 0, 표준펴차 1인 feature로 만들었다.

1-2. hyperparameter 구하기



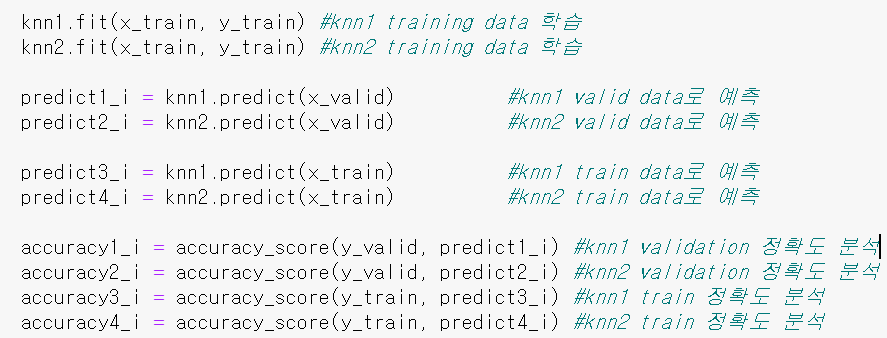
우선 KNN을 import해준다.

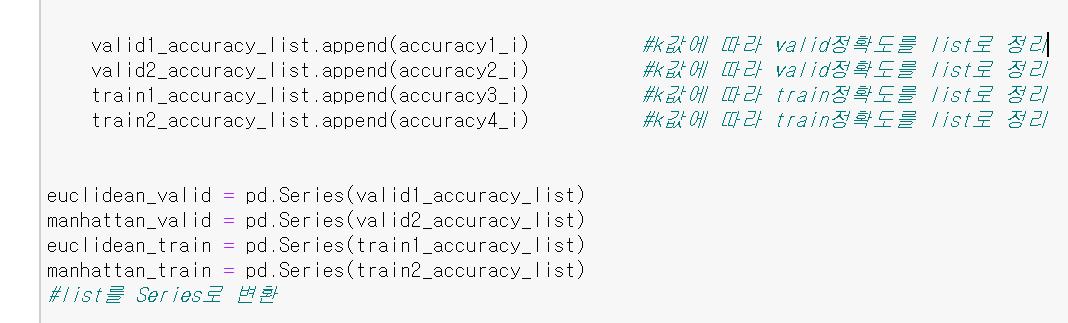
KNN은 k값과 거리계산방식은 metric이 hyperparameter이다.



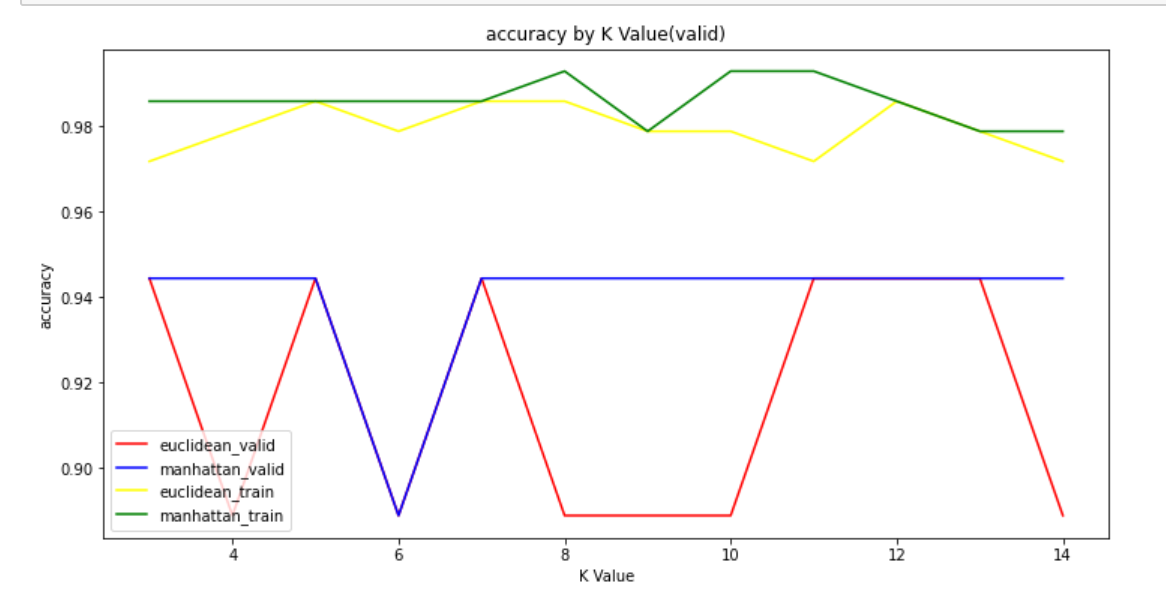
위와 같이 K값을 3부터 15까지 증가시키면서 metric을 두방식으로 나눠서 KNN을 선언한다.

또한 아래 사진을 보면 우선 training data로 학습을 시키고, train 정확도와 valid 정확도를 분석한다. 그리고 각각 list로 정리해둔 뒤, 최종적으로 pandas.Series로 만든다.





위의 코드를 통해서 metric에 따른 k값변화시의 train 정확도와 valid 정확도를 함께 나타냈다.

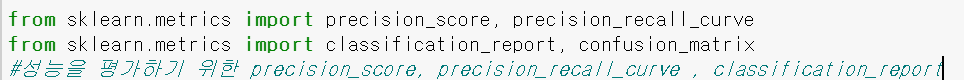


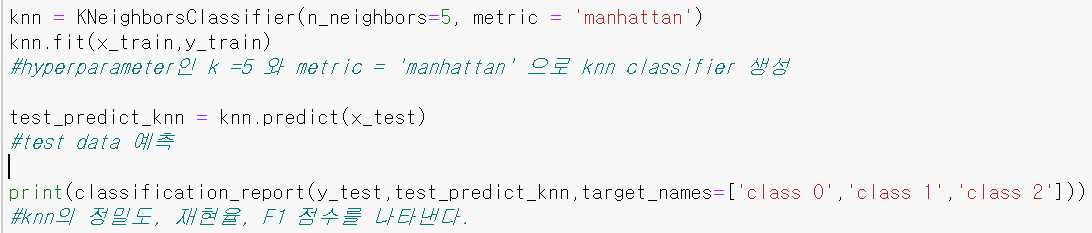
(k값에 따른 validation & train 정확도 그래프)

K값에 따른 accuracy를 plot으로 나타낸다. EDA 그리는 과정에서 판단했을 때의 예상으로는 metric 방식이 무엇이든지 상관없을 것이라고 생각했다. 하지만 그래프의 추이를 보면 전체적으로 train 뿐만 아니라 validation에서 manhattan의 정확도가 더 좋다. 그래서 metric은 manhattan으로 hyperparameter로 이용하는 것이 좋다고 평가된다. 또한 k값은 6일때를 제외하면 모두 정확도가 높으며, k값이 짝수일때, 잘못된 값을 선택될 수 있으므로 적당한 k값인 5를 hyperparameter 로 선택하는 것이 좋다고 평가했다. 이 과정으로 검증을 함으로서 overfitting을 방지한다. 본인이 k값을 5정도로 선택한 것은 EDA를 그리는 과정에서 scaling 되었을 때의 train data 분포를 보면 일분 두개정도의 data를 제외하면 class 별로 뭉쳐 있는 경향이 나타난다. 그래서 k=5 정도가 적당하다고 생각했다.

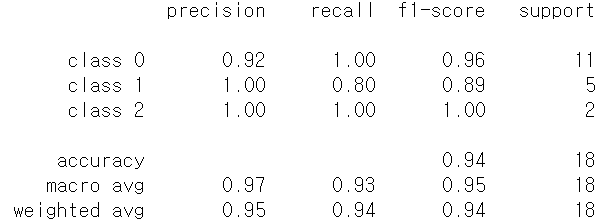
1-3. test 단계

우선 test 단계의 성능을 평가하기위해서 필요한 것들을 import 해준다.





Import 후 hyperparameter 로 지정한 k 값 5 와 metric값 mahattan으로 KNN classifier 생성해준 후, test data를 predict한다. 그리고 classification\_report 함수를 사용해서 knn의 정밀도, 재현 율, F1점수를 나타냈다.



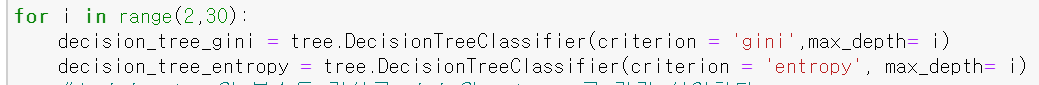
위의 table을 보면, class 0의 precision, recall, f1-score은 0.92, 1, 0.96이고, class 1은 1, 0.8, 0.89 이며, class 2 는 1 , 1, 1 이다. Class 2은 완벽한데 비해서, class 1은 FN의 비율이 0.2, class 1는 FP의 비율이 0.08이다. 그리고 전체적인 정확도는 0.94 로서 굉장히 괜찮은 성능을 보인다고 할 수 있다. 위의 표를 통해서 hyperparameter가 잘 선택되었음을 알 수 있다.

**(2) Decision Tree**

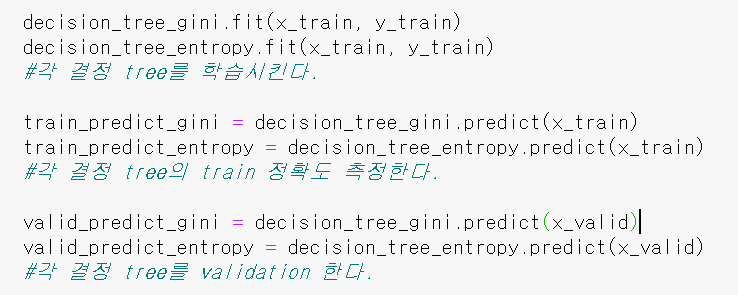


Decision Tree의 사용을 위해서 import 해준다.

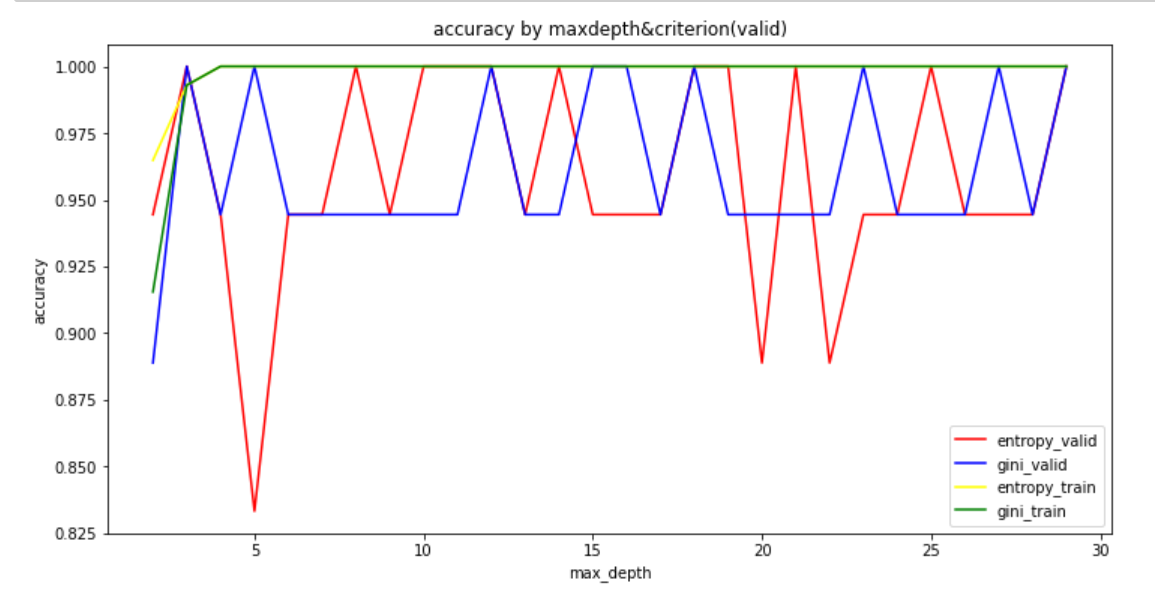
2-1. hyperparameter 구하기



Decision tree의 decision boundary를 나누는 기준이 되는 불순도 검사 방식인 gini와 entropy 방식을 사용하는 decision tree 분류기를 각각 선언한다.



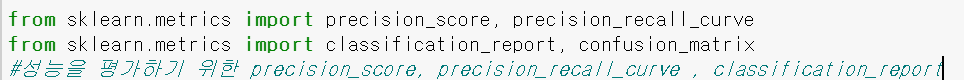
그리고 max\_depth에 따라서 각각 학습시키고 train 정확도와 valid 정확도를 측정한다.



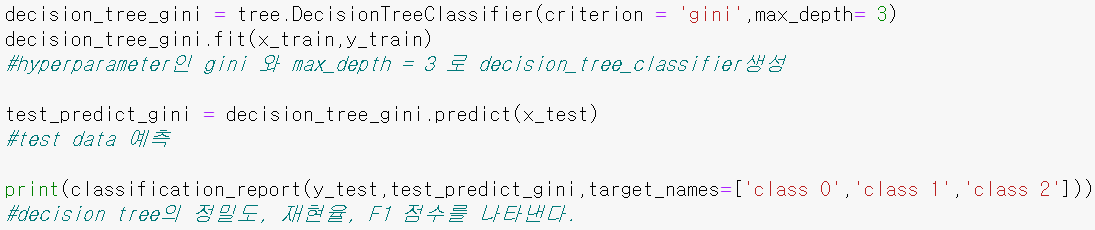
criterion와 max\_depth에 따른 valid정확도와 train 정확도를 그래프로 나타냈다. 우선 train 그래프는 데이터양이 많지 않아서 인지 거의 100%정확하게 맞춘다. 그래서 validation만 판단했다. Decision tree의 max depth에 대한 그래프를 그리기 전에는 gini가 연산속도가 빠르지만 편향적인 data를 만드는 경향이 있고, entropy가 조금 느린 대신, 균형 잡힌 tree를 만든다고 알고있어서 entropy가 더 좋은 성능을 낼 것으로 예측하였다. 그러나 그래프를 보면 criterion이 entropy일때보다 gini일때가 평균적으로 더 높은 평균 그래프를 보인다. 왜냐하면, entropy는 정확도가 아래로 튀는 경우도 보이기 때문이다. 그래서 본인은 hyperparameter로 criterion = gini 로 선택하였다. 왜냐하면 valid 단계에서 평탄한 그래프가 test 단계에서도 준수한 정확도를 보이는 경우가 많기 때문이다. 결정 트리의 깊이를 제한하지 않으면 트리는 무한정 깊어지고 복잡해질 수 있다. 본인은 그래프를 통해서 max\_depth를 3으로 놓는 것이 적합하다고 생각한다. max\_depth깊어지면 잘못된 결과가 나타날 수 있기 때문에 3으로 하였다.

2-2. test 단계

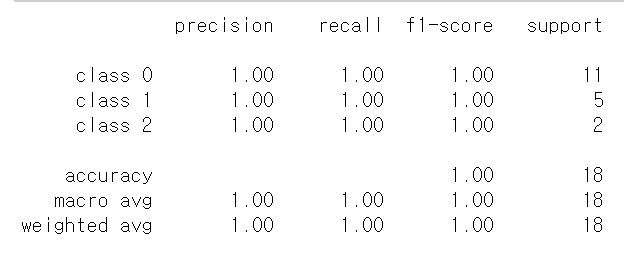
다음으로 test 단계를 진행한다.



필요한 것들을 import 해준다.



그리고 hyperparameter로 지정한 criterion = gini 와 max\_depth = 3으로 decision tree classifier를 생성하고 학습시킨다. 그리고 test data 예측한 후 classification\_report로 precision 과 recall, 그리고 f1-score를 나타낸다.



위의 table을 보면 최상의 test 정확도가 나타났다. Validation 단계와 마찬가지로 정확도가 1이 나왔다. 이를 통해서 decision tree의 hyperparameter가 잘 선택되었음을 평가할 수 있다.

**(3) Ensemble (decision tree-based)**

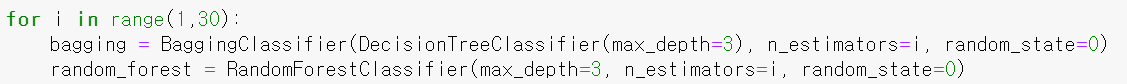


필요한 모듈을 import 해준다.

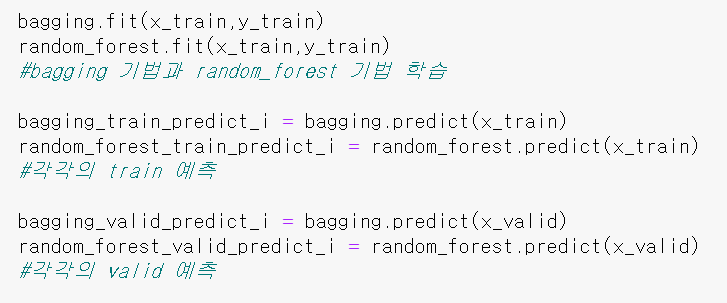
3-1. random forest 와 bagging 차이 비교

이 부분에서 random forest와 Bagging의 차이를 보이기 위해서 먼저 둘이 비교를 하고, 그 후에 random forest의 hyperparameter를 좀 더 자세히 구하겠다.

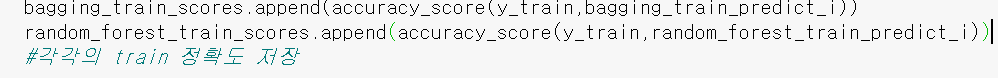
기본적으로 bagging 방식은 Bootstrapping과 voting or Averaging을 사용한다. 그리고 random forest도 같은 방식을 사용하는데, random forest는 bootstrapping을 할 때, 전체 feature의 일부 feature를 뽑아서 resampling 한다.



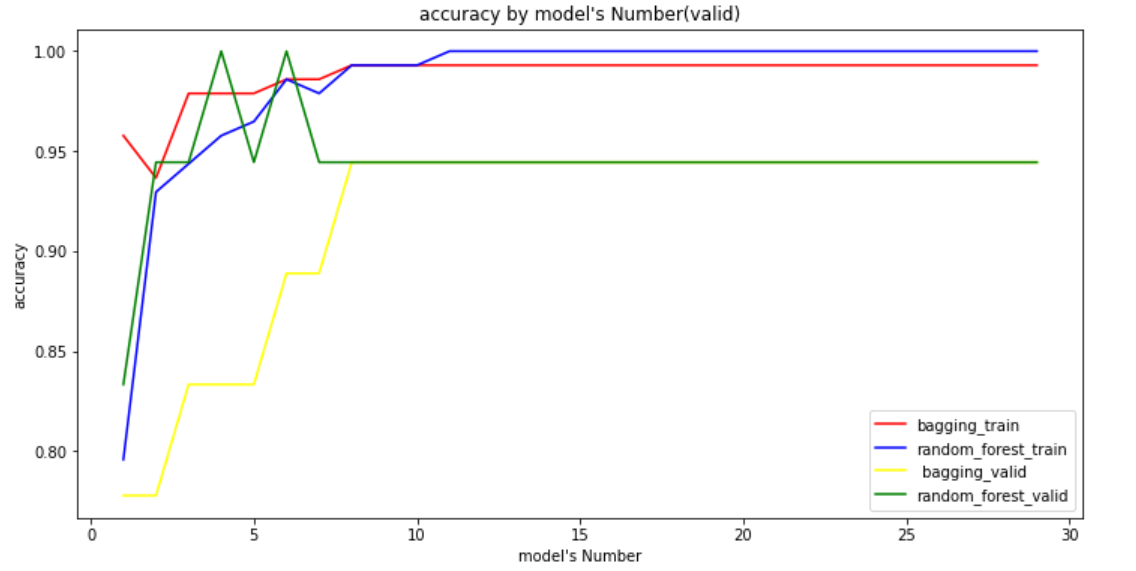
기본적으로 둘 다 decision tree를 기반으로 하기 때문에, decision tree에서 hyperparameter로정한 max\_depth는 3으로 지정하겠다. 둘의 비교는 subset(model) 개수를 늘려가면서 valid 한다. 이때, bootstrapping 둘 다 사용한다. random\_forest의 feature 참조 수는 default값인 10으로 사용한다.



우선 bagging 기법과 random forest기법을 train data로 학습시키고, 각각을 예측한다.



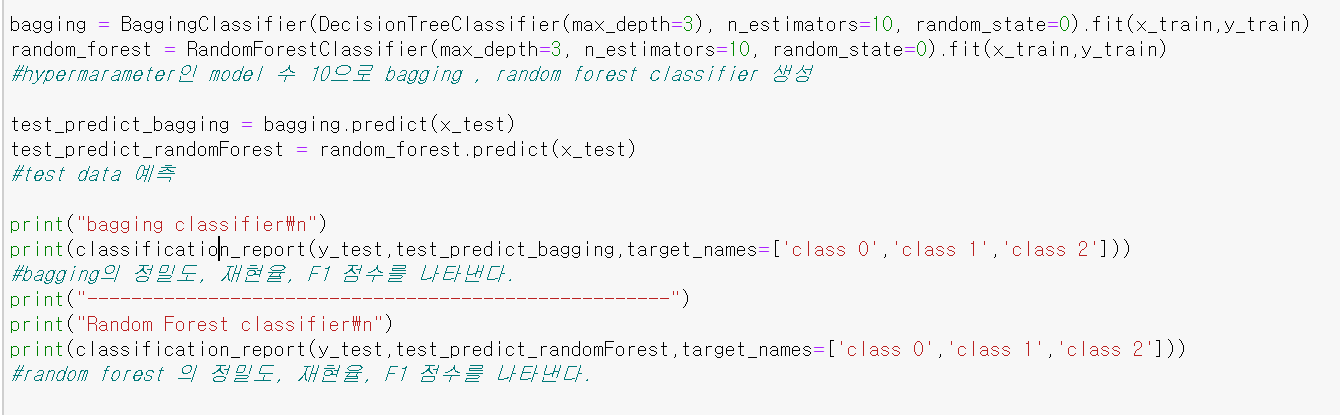
그리고 정확도를 subset의 개수에 따라서 list에 저장한 후 Series로 변환한다.



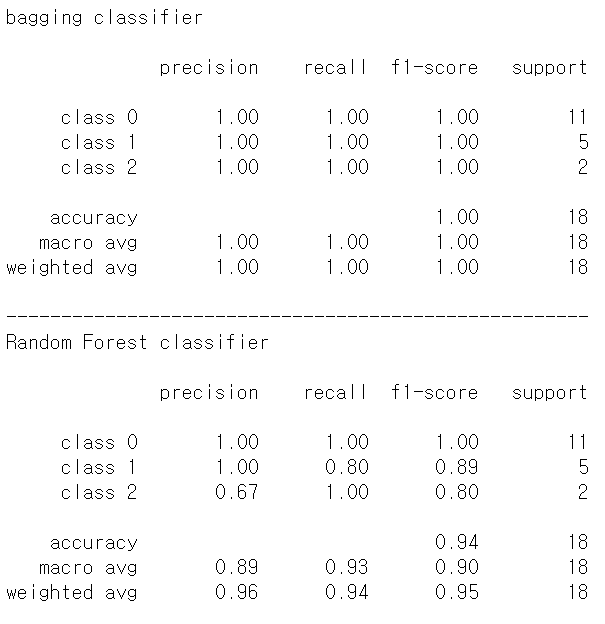
전체 data의 subset(model) 수에 따라서 train과 valid의 정확도를 그래프로 나타냈다. 일반적으로 model의 수가 증가하면 정확도는 증가한다. 하지만 모델 수가 적을 때는 그래프에서 볼 수 있듯이 아닌 경우도 존재한다. 위 그래프를 통해서 hyperparameter인 model의 수는 10정도 이후에 대체로 일정한 정확도를 보이므로, 10으로 정하였다.

bagging과 random Forest의 차이는 feature를 전부 가져오는 것과 아닌 것이다. 이때 feature의 일부만 가져오게 되면 feature의 의존도에 따라 잘못된 값이 예측되는 경우를 막을 수 있다. 그래서 bagging 보다 random forest의 성능이 더 좋을 것으로 예측하였다. 실제 validation과정을 거치면서, 그래프에서 보이듯이 bagging만 사용한 경우보다 random Forest를 사용한 경우가 train, valid에서 모두 정확도가 우세하다.

3-2. bagging와 random forest의 test 단계비교



Hyperparameter로 정한 model의 수 10으로 Bagging, random forest classifier를 생성한다. 그리고 각각 train data 학습 후, 각각의 test data에 대한 classification report를 출력한다.

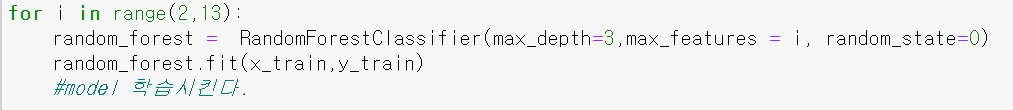


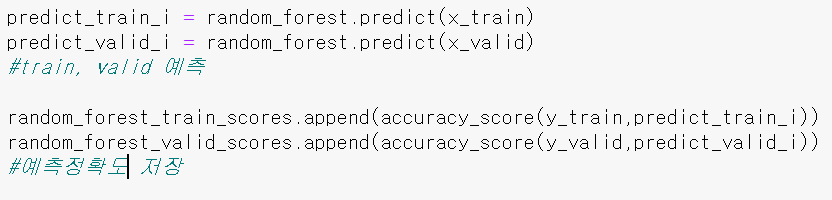
위 표는 bagging과 randomForest의 classification report이다. 그런데 전혀 예상하지 못한 결과가 발생하였다. 분명 validation 단계에서 random Forest의 정확도가 높았는데, bagging이 더 좋은 정확도를 보이고 있다. 예상하지 못한 결과여서 많이 당황하였다. 하지만, 특정 test data에서 이런 결과를 보인다고 하여 bagging이 더 좋다고 할 수는 없다. random\_forest의 hyperparameter들을 정확히 잡아주지 않아서 혹시 overfitting이 일어난 것이 아닌가 싶기도 하다. 하지만 가장 의심되는 것은 전체 데이터 양이 적어서 일 것이다. 또한 특정 test집단만이 이런 경우 일 수도 있다. 하지만 전체적으로 random forest가 훨씬 우수한 성능을 보인다. 따라서 좀 더 test 집단이 필요하다고 평가된다.

다음 그래프에선 random forest의 중요한 특징인 참조 feature 수를 구한다.

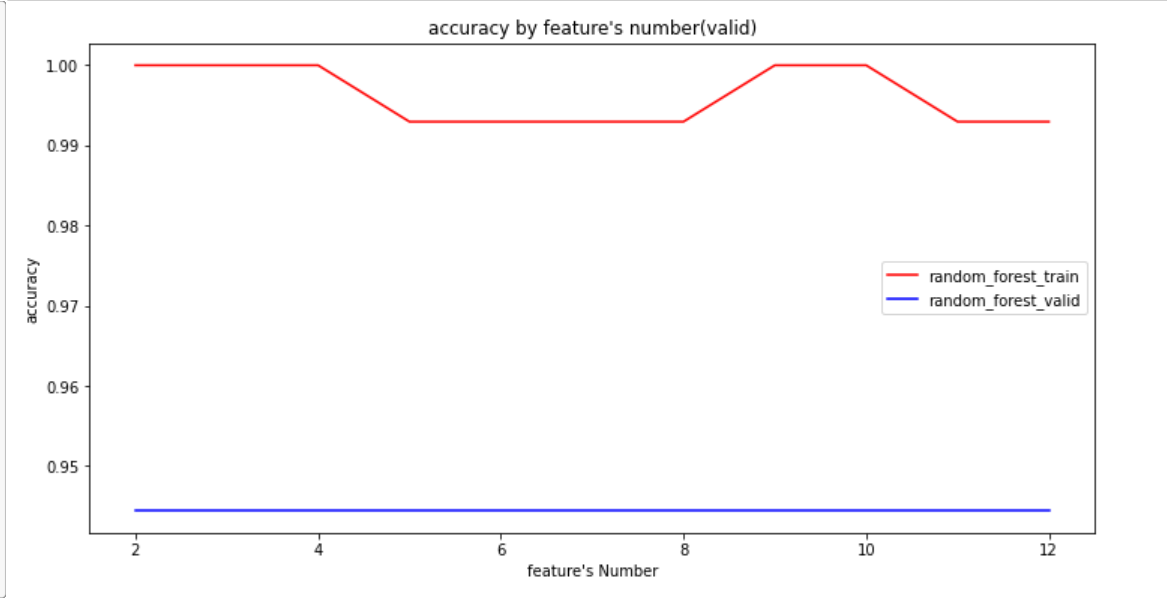
3-2. random forest의 max feature 수 결정

우선 random forest의 hyperparameter는 decision tree의 개수, data feature를 참조할 비율(개수), max\_depth, 리프 노드가 되기 위한 최소한의 sample data의 수, 노드를 분열하기 위한 최소한의 데이터 수 가 있다. 하지만 과제의 data set은 많은 편이 아니므로 가장 중요한 feature 참조 수를 결정하였다. 또한 max depth는 decision tree에서 구한 3으로 고정하고, 위에서 bagging과 random forest를 비교할 때, decision tree (model)의 수가 10 정도 되면 일정한 정확도를 보였기에 default인 10으로 고정한다.

 Random forest classifier를 위와 같은 조건으로 참조 feature 수의 변화를 주면서 선언하고, train data로 학습시킨다.

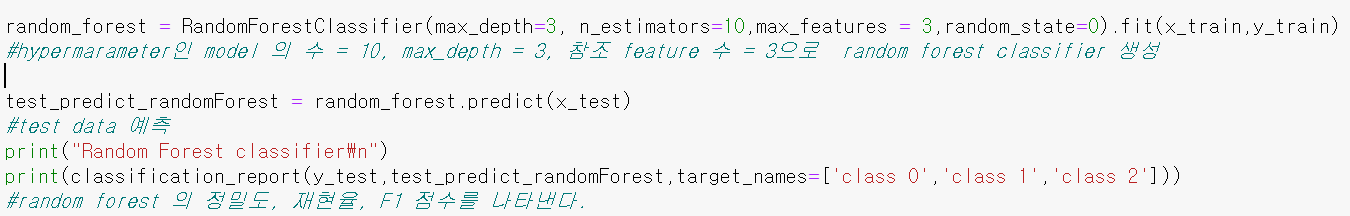


그 후 train data와 valid data를 예측하고, 참조 feature 수에 따라 예측 정확도를 저장한다.

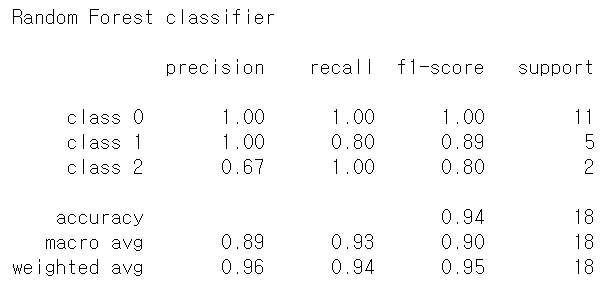


그래프를 보면 valid에서 feature수의 변화에 따라 변화가 없다. train에서도 크게 변화가 없다. 따라서 일반적으로 feature의 수는 root(p)이기 때문에, feature 수는 root(13) = 3.xxx이므로 3으로 하겠다. 최종적으로 결정한 hyperparameter는 참조 feature 수는 3개, max\_depth는 3, model 수 10개이다. 위와 같이 큰 변화가 없는 이유는 전체data set이 적기 때문이라고 생각한다.

3-3. test 단계



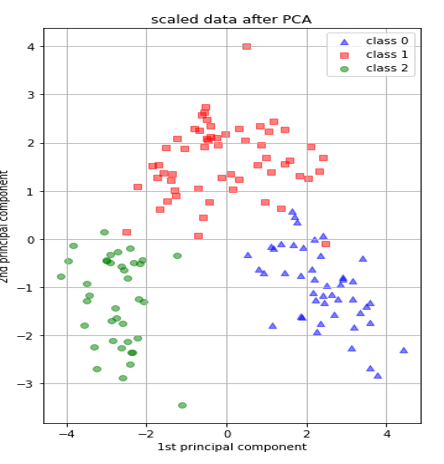
Test 단계에서 validation 단계에서 구한 hyperparameter로 random forest classifier 생성, 학습시킨 후 test data에 대한 classification report를 출력한다.



위의 표를 보면 class 0는 완벽하게 구하고, class 1에서는 recall: 0.8, class 2에서는 precision: 0.67이라는 조금 좋지 않은 결과가 나타났다. 하지만, 전체 정확도 0.94 라는 꽤 괜찮은 정확도가 나타난다. Test 단계에서 이 정도의 정확도라면 꽤 괜찮은 정확도라고 할 수 있다. 따라서, random forest의 hyperparameter를 잘 지정했다고 할 수 있다.

**(4) SVM**

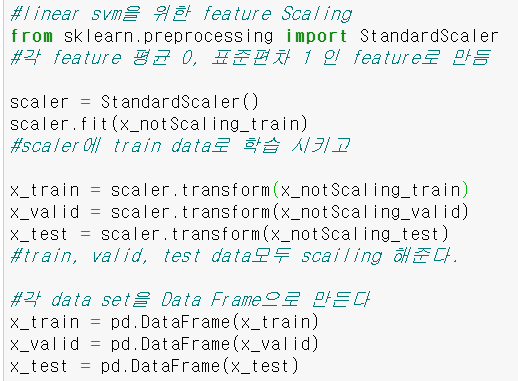
SVM 알고리즘은 linear-svm 알고리즘과 kernel-trick을 사용한 svm 알고리즘이 있다. 이번 단계에서 linear-svm 알고리즘과 kernel-trick 사용한 svm 알고리즘을 비교한다.



우선 맨 처음 EDA 그릴 때, data scaling을 하면 각 class 별로 분류가 된다. 이 그래프를 통해서 이용된 dataset이 linear한 형식으로 분포되어 있어서 linear SVM에 적합하다는 것을 판단하였다. 그래서 linear SVM이 kernel SVM보다 좋은 성능을 보일 것으로 판단했다.

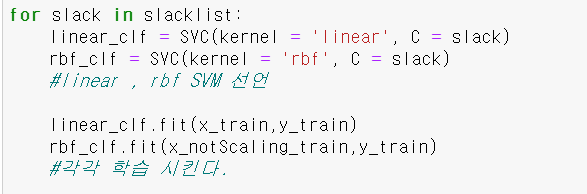


SVM 알고리즘을 사용하기위해서 import 해준다.

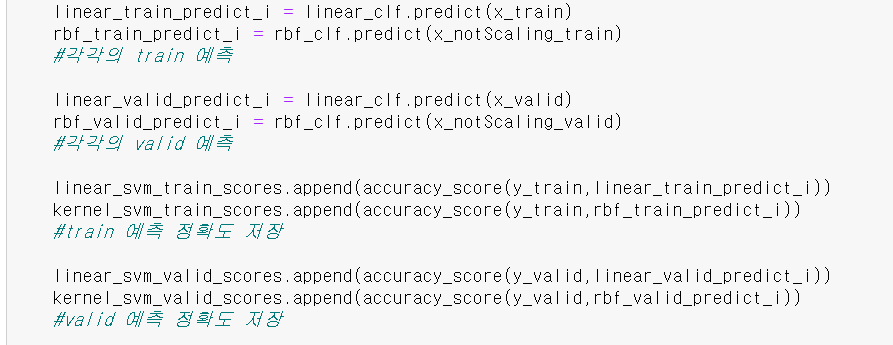


Linear SVM을 위해서 feature scaling 해준다.

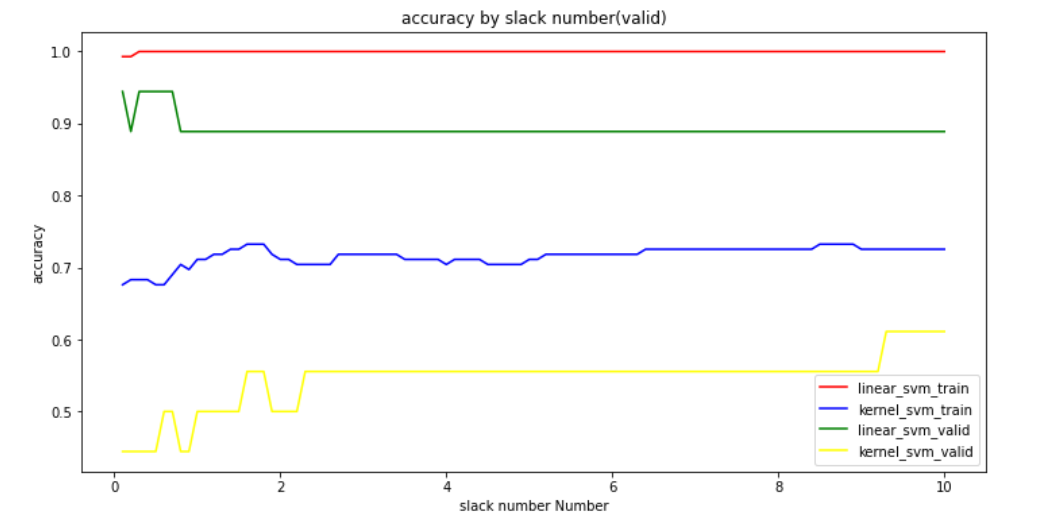
4-1. hyperparameter 구하기



SVM알고리즘의 hyperparameter는 slack 변수 가중치 C이다. 따라서 C의 변화를 주면서 hyperparameter를 tuning한다. 또한 kernel – trick을 사용하는 SVM의 kernel 함수는 RBF를 사용한다. 그래서 slack변수를 0.1씩 증가시키면서 linear SVM과 Kernel SVM알고리즘을 사용한다.

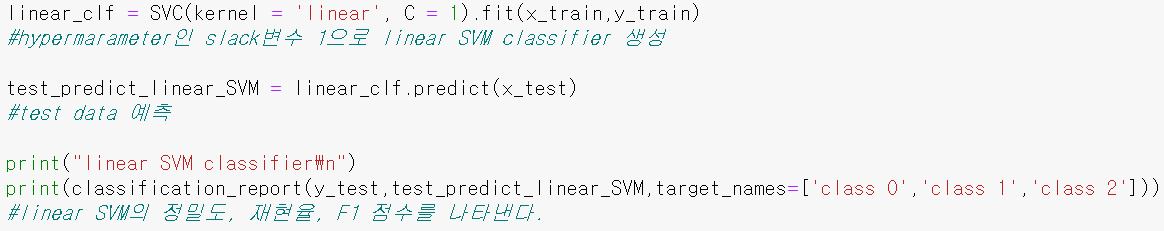


다음으로 각각의 train 정확도와 validation 정확도를 저장한다.

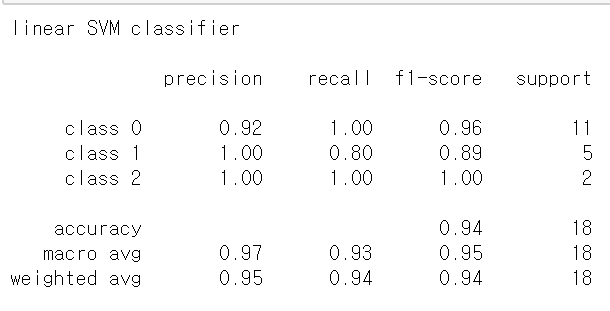


Slack 변수가중치의 값에 따라 linear SVM과 kernel SVM의 정확도를 그래프로 나타냈다. 그래프를 보면 train과 valid 단계의 정확도가 linear SVM이 kernel SVM보다 월등하게 높다. 이 것은 EDA 그리는 단계에서 linear SVM이 더 좋은 성능을 보일 것으로 예상한 것과 일치했다. 또한 linear SVM 은 slack variable의 영향을 거의 받지 않지만, kernel SVM은 c값에 따라 조금씩 변동이 보인다. 이 그래프를 통해서 이용된 dataset이 linear한 형식으로 분포되어 있어서 linear SVM에 적합하다는 것을 증명했다. 그리고 hyperparameter 인 slack 변수는 1이상부터는 linear에서 일정한 정확도를 보이므로 C값은 1로 하겠다. 그리고 SVM 방식은 linear로 선택한다.

4-2. test 단계



우선 hyperparameter로 지정한 slack 변수 가중치 1로 linear SVM을 생성, 학습한다. 그리고 test data의 classification report를 출력한다.



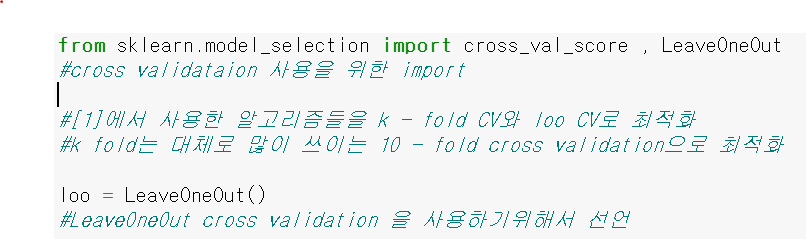
위 표를 보면 class 2는 모두 1 이고 class 0은 0.92, 1, 0.96 이다. 또한 class 1은 1, 0.8 ,0.89 로서 굉장히 괜찮은 성능을 보이고 있다. 또한 예측 정확도는 0.94로 꽤 괜찮은 성능을 보이고 있음을 알 수 있다. 이를 통해서 SVM 방식을 linear로 하고 Slack 변수 가중치를 1로 설정한 hyperparameter는 잘 설정하였음을 평가할 수 있다.

**[2] Cross-validation을 이용한 최적화된 알고리즘 탐색**

**(0) 공통사항**



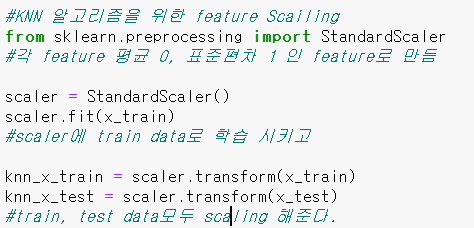
Data를 [1]와 마찬가지로 train : valid : test = 80 : 10 : 10 으로 나눠서 [1]과 똑 같은 data로 만든다. 그 후 pandas의 concat을 이용하여 train와 valid data를 결합해준다. 즉, data를 train : test = 90 : 10 으로 만든다.



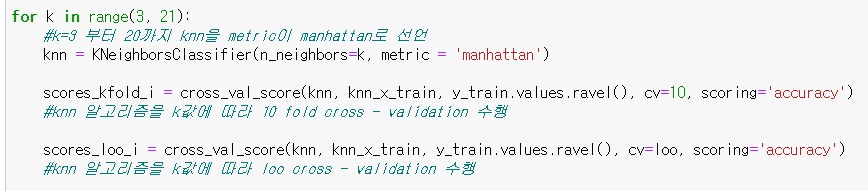
주석으로 설명을 써 놓은 대로, cross validation을 사용하기위해서 필요한 것을 import해준다.

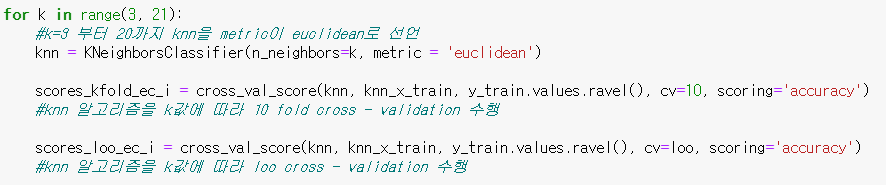
**(1) 각 알고리즘 최적화**

1-1. KNN 알고리즘

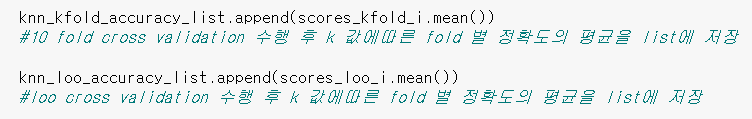


우선 KNN 알고리즘을 cross validation 하기전에 feature scaling 해주었다.



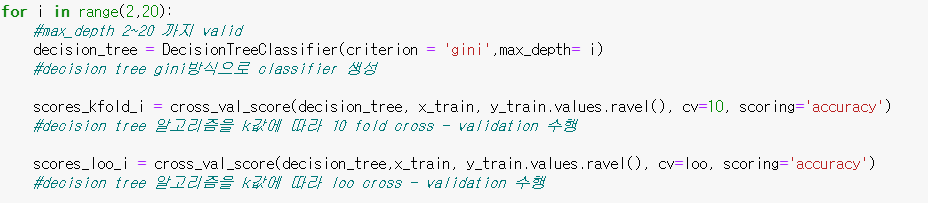


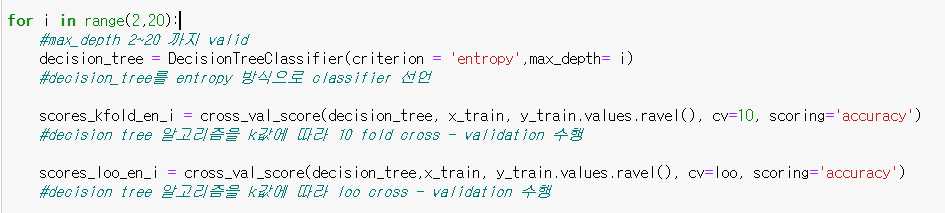
Metric 방식이 manhattan과 euclidean방식으로 선언한 KNN 알고리즘을 k 값의 변화에 따라서 cross validation을 한다. 이때, K fold CV 와 loo CV를 동시에 진행했다.



그리고 각 k값에 따라 나오는 valid의 score에 평균을 저장해 놓는다.

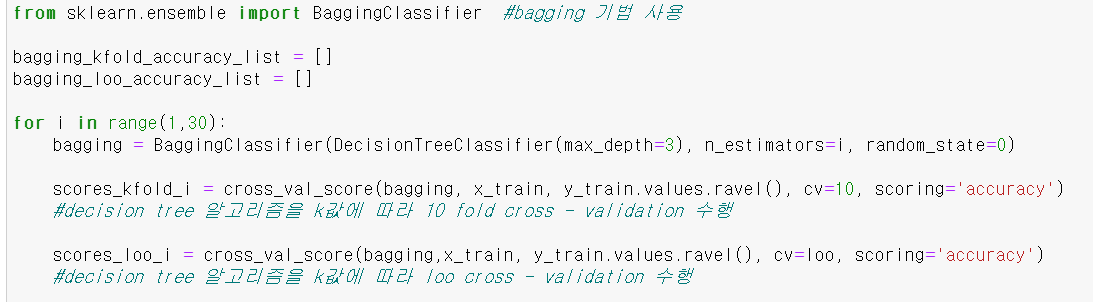
1-2. Decision Tree





Decision Tree도 gini과 entropy방식으로 각각max\_depth에 변화를 줘가면서 cross validation을 진행했다.

1-4 Bagging

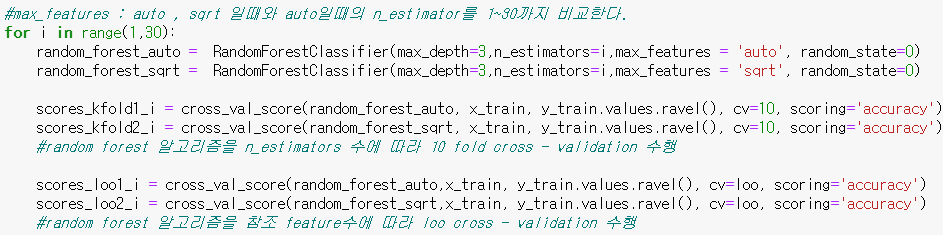


Bagging을 n\_estimator의 수에 변화를 주며 10 fold CV 와 loo CV를 수행한다.

1-3. random forest



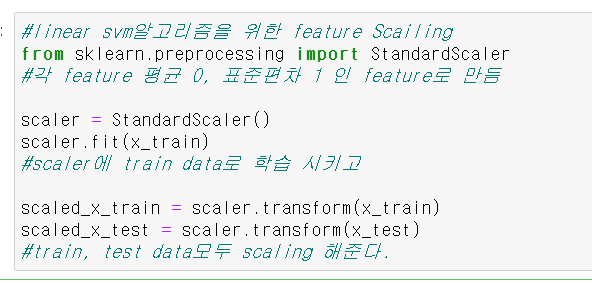
Random forest import해준다.



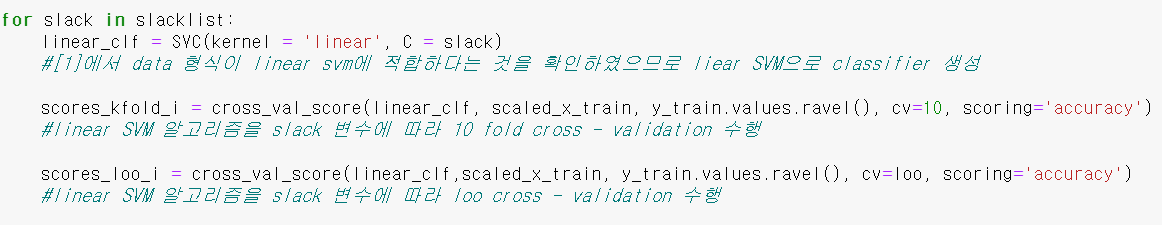
Max\_Feature를 ‘auto’인 경우와 ‘sqrt’인 경우로 나누고 n\_estimators의 수에 변화를 주면서 classifier를 생성한다. 그리고 각각 10 fold CV 와 loo CV를 수행한다.

.

1-4. SVM 알고리즘

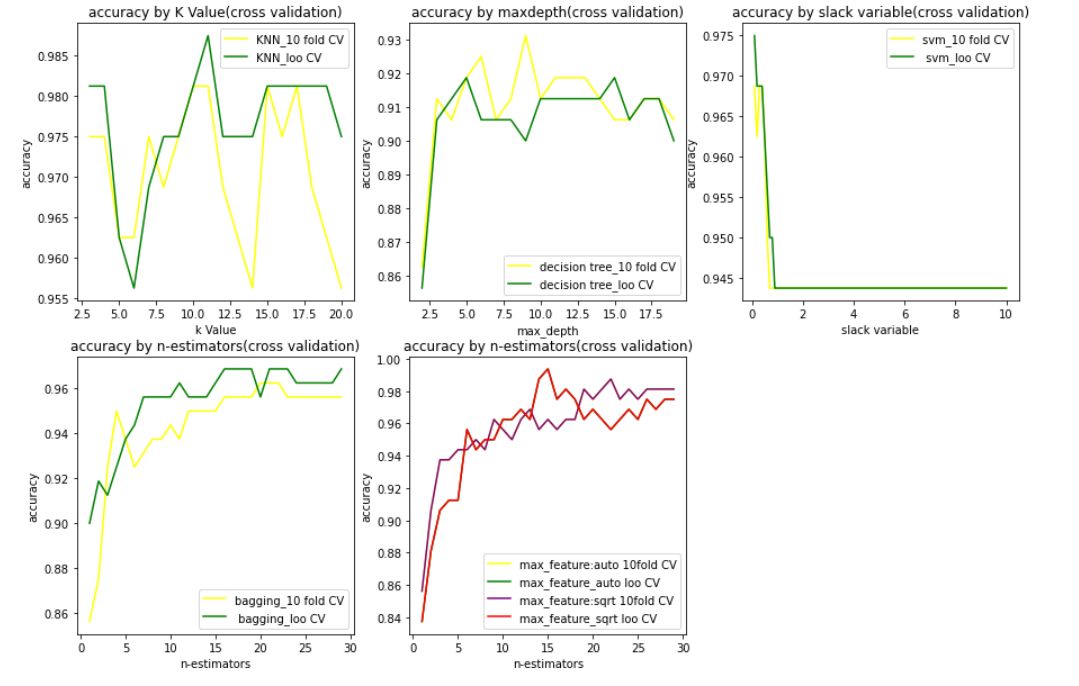


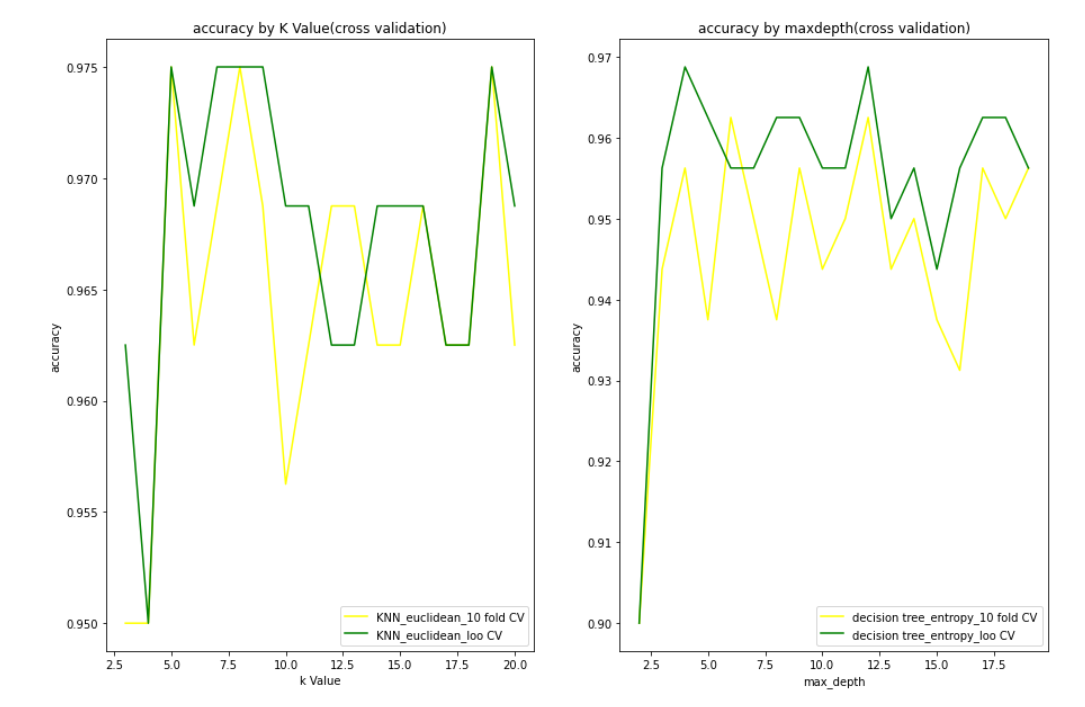
Feature scaling 해준다.



SVM 알고리즘은 slack 변수의 가중치를 0.1 부터 10까지 0.1씩 변화를 주면서 평가하였다.

**(2) 각 알고리즘의 cross validation을 통한 accuracy 그래프**





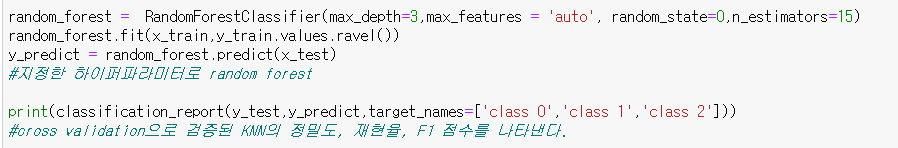
각 알고리즘의 hyperparameter에 따른 정확도를 subplot을 이용한 그래프로 나타냈다. 예상과 달리 KNN과 decision tree의 entropy방식을 제외하면 10 fold CV와 비교했을 때, loo CV의 정확도가 높은 편이 아니다. 예상대로라면 모든 data를 validation에 사용하는 loo CV가 더 성능이 좋았어야 한다. 하지만 오히려 성능이 떨어지는 경우가 있는 것으로 볼 때, loo CV 방식이 항상 좋은 것은 아니라고 볼 수 있다.

과제 조건에 의하면 가장 좋은 모델을 위주로 보고서를 작성하라고 하였다. 위 그래프들을 통해서 본인은 random forest가 가장 좋은 model이라고 판단했다. 그래프를 보면 전체적으로 loo CV와 10 fold CV의 평균 정확도가 비슷하게 나타났다. 이 것에 대한 이유는 말했듯이, loo CV가 항상 좋은 성능을 보이는 것은 아니라고 판단된다.

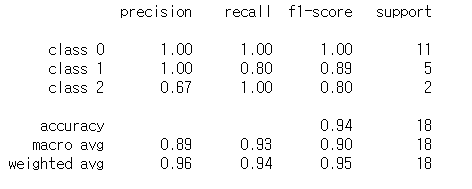
Max feature를 살펴보면, max feature가 ‘auto’ 인지 ‘sqrt’인지에 대해서는 영향을 받지 않았다. 그래서 max feature의 값은 임의로 auto로 지정한다. 그리고 그래프를 보면 n\_estimators의 값이 15일 때, 가장 높은 정확도인 0.99를 보인다. 따라서 cross validation로 찾은 hyperparameter는 max feature = ‘auto’, n\_estimators = 15이다.

**(3) Cross validation으로 검증한 random forest test 단계**

Cross validation으로 평가해서 가장 좋은 성능을 가진 모델로 random forest를 선택했다.



그래서 이 최적의 hyperparameter로 random forest classifier를 생성 후 model을 평가하기로 판단하였다. Random forest의 평가를 위해서 [1]에서 test단계에서 사용한 classification report를 사용하였다.

' 

Cross validation으로 최적화된 random forest 모델을 이용하니 test단계에서 0.94의 정확도가 나왔다. Class 2의 전체 Positive로 판단 data중 FP 비율이 0.33 이고 class 1의 전체 positive 중 FN 비율이 0.2이지만, 전체적으로 나쁘지 않은 정확도라고 생각한다. 다. 이를 통해서 Cross validation을 통한 hyperparameter를 아주 잘 설정했음을 알 수 있다. 또한 [1]에서 valid data를 이용한 validation 단계보다 cross validation이 시간은 더 걸려도, valid를 더욱 많은 data로 이용하므로 더 좋은 결과를 나타낸다. 물론 이 Wine data에서는 큰 차이를 보이지는 않는데, 이는 data set이 적고, hyperparameter를 [1], [2] 모두 잘 설정했다고 할 수 있다.

**결론**

Wine data에 대한 EDA를 통한 data 조사 후, 각 알고리즘으로 train, valid, test과정을 하였다. 모든 알고리즘이 hyperparameter를 잡아주었을 때, 준수한 성능을 보였다. 그 중, 가장 큰 성능을 보인 것은 random forest였다. 그리고 DATA의 EDA를 판단하는 과정에서, 모든 feature와 sample을 버리지 않은 판단은 괜찮은 판단임을 [1],[2] 과정을 거치면서 잘 판단했다고 할 수 있다. 또한 cross validation과 valid set을 이용한 validation의 hyperparameter의 차이를 느낄 수 있었다. 하지만 너무 높은 정확도에 의문이 생기기도 한다. 처음에 생각을 할 때, data가 적으면 test단계의 error 율이 높지 않을까 생각을 했다. 하지만, 내 생각과는 다르게, 분류기들이 준수한 성능을 보여줬다. 그래서 느끼기에, hyperparameter를 잘 설정한 것 같다. 하지만 다른 test set으로 test를 해야 할 필요가 있음을 판단했다.