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

**Homework 4**

Due April 23

**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.**

**제출**

* **요구한대로 작성한 보고서와 주피터 노트북(주석 필수)**
* **최종 점수는 비율적으로 수정해서 들어갈 예정**

**와인 데이터란** 와인의 등급을 분류하기 위한 목적으로 만들어진 데이터 집합. 해당 과제에 올려진 주피터 노트북의 데이터 로드 방식을 따라 과제를 진행.

**scikit-learn 라이브러리를 바탕으로 과제를 진행한다**. scikit-learn 라이브러리에 대한 매뉴얼 홈페이지는 다음과 같다.

<https://scikit-learn.org/stable/>

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

해당 두 데이터를 바탕으로 수업시간에 배운 분류 모델을 적용하고 최적의 파라미터를 찾아본다. 주피터 노트북으로 해당 모델들을 적용시킨다. 노트북 제출시 주석을 달아 설명한다.

훈련과 검증을 위한 데이터 샘플 split 방식은 다음과 같다.

* 총 주어진 샘플 중 80%를 training 데이터로 구성하여 훈련시킨다.
* 10%를 validation 데이터로 구성하여 하이퍼 파라미터를 최적화한다.
* 최종적으로 남은 10%를 test 데이터로 구성하여 성능을 보고서와 주피터 노트북에 표시한다.

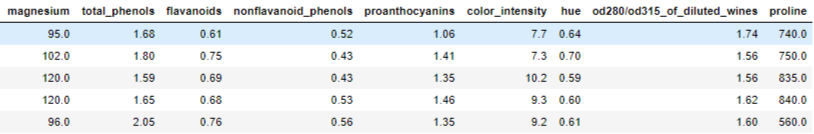
보고서 작성 요령

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

적용해야 할 알고리즘과 최소한의 하이퍼 파라미터 탐색의 종류

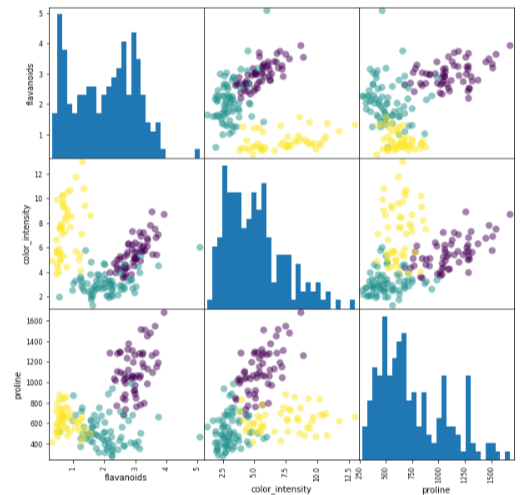
1. **Knn**
   1. **수업시간에 배운 metric 들 중 최소 두 개 비교**
   2. **이웃의 개수 k를 최소 3개 비교**

먼저 wine 데이터의 feature마다 value들이 어떻게 형성되어 있는지 확인하기 위해 data frame을 생성해 확인했다.



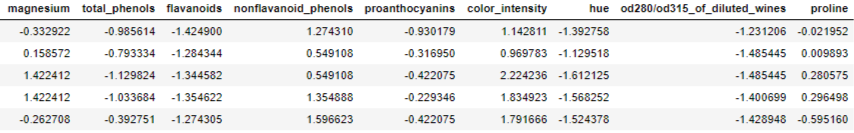
**그림1. Scaling 작업 전 wine data의 일부**

위의 그림1에는 wine data의 모든 feature들에 해당하는 값들이 나타난 것은 아니지만, proline, magnesium과 같은 feature들은 다른 feature에 비해 값이 상당히 큰 것을 확인할 수 있다. KNN에서 이러한 불균등한 feature size는 정확한 측정을 하기에 적합하지 않기 때문에 wine data의 각 feature들의 값들을 기본적인 scaling방법인 standard scaler를 통해 표준화 작업을 수행했고, 각 feature들이 0을 기준으로 표준분포를 갖는 형성을 띄게 되었다. 학습시킬 데이터와 성능평가에 사용할 데이터들은 모두 Scaling된 data를 분할하여 사용할 것이다.



**그림2. 와인 데이터의 feature간의 관계(사이즈가 크므로 전체 13개 feature 중 3개 feature 추출)**

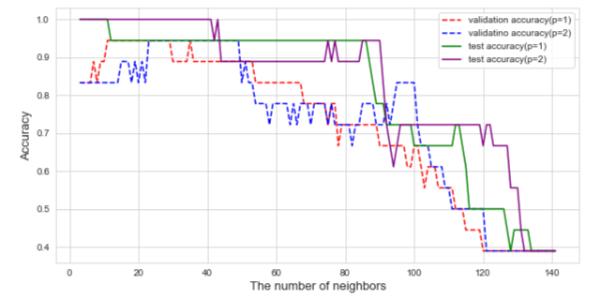
위 그림2는 분류 모델을 생성하기 전 와인데이터의 전체적인 구조를 파악하기 위해 나타낸 자료이며, 자료를 살펴보면 flavonoids값이 작을수록 노란색(2번 class)으로, flavonoids값과 proline값이 커질수록 보라색(class 0번 클래스)로 분류되는 것을 확인할 수 있으며, 이처럼 feature들 간의 관계에 따라 어떻게 분류되는지 알 수 있다.



**그림2. Scaling 작업 후 wine data의 일부**

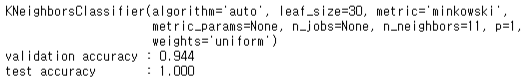
data set을 크게 training set과 test set으로 나눈 뒤, 다시 training set을 최종 모델을 선정하기 위한 성능을 평가할 validation set과 sub train set으로 나누었다. Validation set으로 sub training set으로 학습한 모델의 성능이 나쁘지 않다면, 해당 모델에 전체 training set을 이용해 성능을 측정해본 뒤, test set으로 선정된 모델의 최종 성능을 평가할 것이다. 최적인 hyper parameter를 찾기 위해 K의 값을 3부터 141까지 증가시키면서 sub train set으로 학습시켰고, validation set에서의 성능을 측정한 후, 모델을 다시 전체 training set으로 학습시키고, test set으로 성능을 평가했을 때의 측정값들을 살펴보았다.

사용한 모델은 두 가지로, 첫번째 모델은 Manhattan 거리 공식(p=1)을 이용하며, 다른 모델은 Euclidean 거리 공식(p=2)을 이용했다. 분석 결과 k의 값이 증가할수록 성능이 나빠지는 것을 확인할 수 있었으며, 전반적으로 유클리드 거리공식(p=2)을 사용하고, k값이 50 미만일때 성능이 두 모델 모두 높게 측정되었다(p=1일 때와 p=2일 때의 성능 차이는 거의 없었으며, 유클리드 거리 공식을 사용한 모델의 성능이 조금 더 높게 측정되었다). 성능면에서는 거의 차이가 없다고 볼 수 있지만, 유클리드 거리공식의 경우 제곱 연산이 들어가는데 반면, Manhattan 공식의 경우 그저 절대값만 취해주기 때문에, 계산 속도 측면에서는 Manhattan 거리 공식을 사용하는 모델이 더 효율성 있다고 말할 수 있다.



**그림3. K값 증가에 따른 성능 측정**

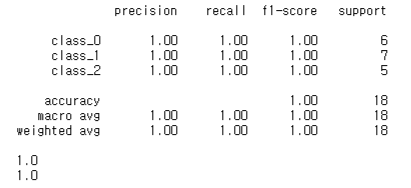
k값마다의 accuracy를 전부 출력했을 때, k=11, p=1일 때 성능이 가장 높게 나왔고, 이를 최적인 hyper parameter로 사용하여 최종 모델을 선정하였다. 최종 모델을 sub train set으로 학습시켰을 때, validation set에 대한 accuracy는 0.944로 높게 측정되었고, 다시 해당 모델을 전체 train set으로 학습시킨 후 Test set에 대해 완벽하게 분류하고 있는 모델임을 알 수 있다.



**그림4. 최종 모델의 성능 평가**

생성된 모델이 test data에 대해 얼마나 잘 분류했는지를 확인하기 위해 와인의 각 class 별로 precision, recall 그리고 f1-score를 분석했다. Precision은 정밀도로 해석되며, 모델이 true라고 분류한 것 중에서 실제로 true인 것의 비율을 나타내고, recall은 실제 true인 것 중에서 모델이 true라고 예측한 것의 비율이다. 어떻게 보면 precision과 recall이 같다고 생각할 수 있는데, 두 metric 모두 실제 true인 정답을 모델이 true라고 예측했을 경우에 관심이 있다는 점에서 공통성을 띄지만, 바라보는 관점이 다르다고 할 수 있다. 결과적으로 두 지표 모두 높을수록 좋은 모델이라고 할 수 있다.

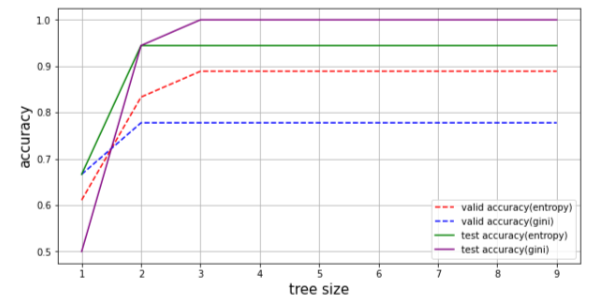
최종 모델의 test set분류 결과, test set의 모든 sample들을 제대로 분류하고 있는 것을 알 수 있다. 따라서 해당 모델은 최적의 hyper parameter로 설정된 좋은 모델이라고 평가될 수 있다.



**그림5. 최종 모델로 분류했을 때의 precision, recall, f1-score 값**

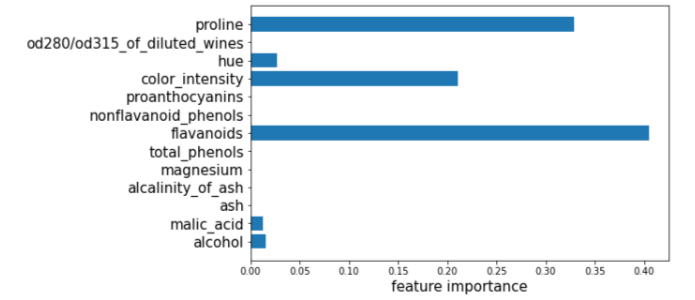
1. **Decision Tree**
   1. **Gini 와 entropy criterion의 비교**

Decision tree에서 설정할 hyper parameter는 tree의 depth이다. 따라서 depth값을 1부터 9까지 사용했을 때 해당 tree모델의 validation set으로의 성능을 확인해보았다. 하나의 tree는 criterion 인자의 값을 entropy로, 다른 하나의 tree는 Gini로 설정하였다. 그 결과 max depth가 3, 4정도일 때, validation set으로의 성능이 잘 나왔기에 depth를 3으로 설정한 모델을 최종 모델로 선정하였다. Entropy와 Gini계수를 이용하는 두 트리 모두 정확도 0.944와 1.000으로 높게 측정되었음을 아래 그래프를 통해 알 수 있다.

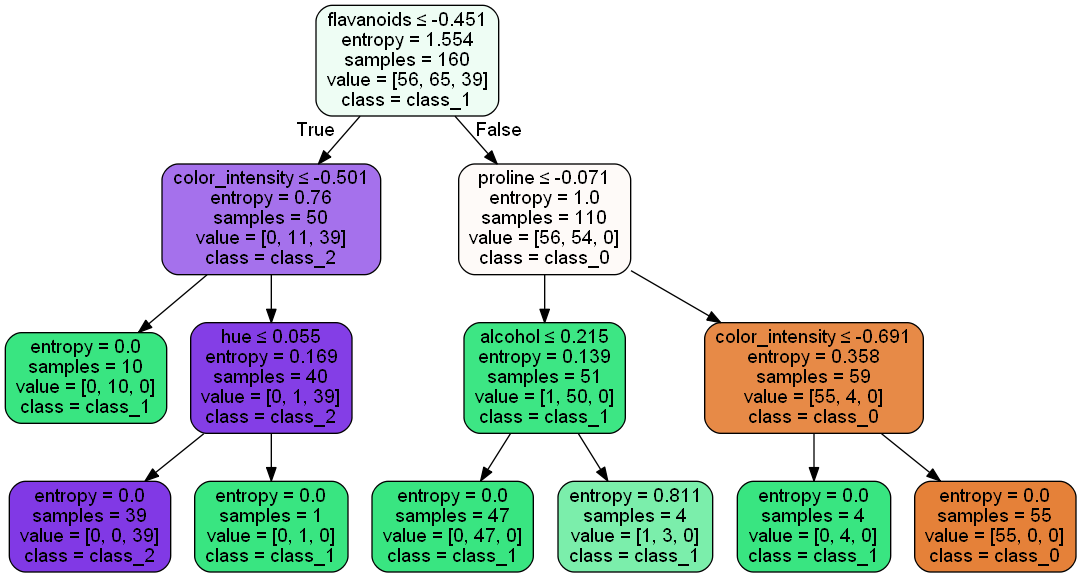


**그림1. depth증가에 따른 성능 변화.**

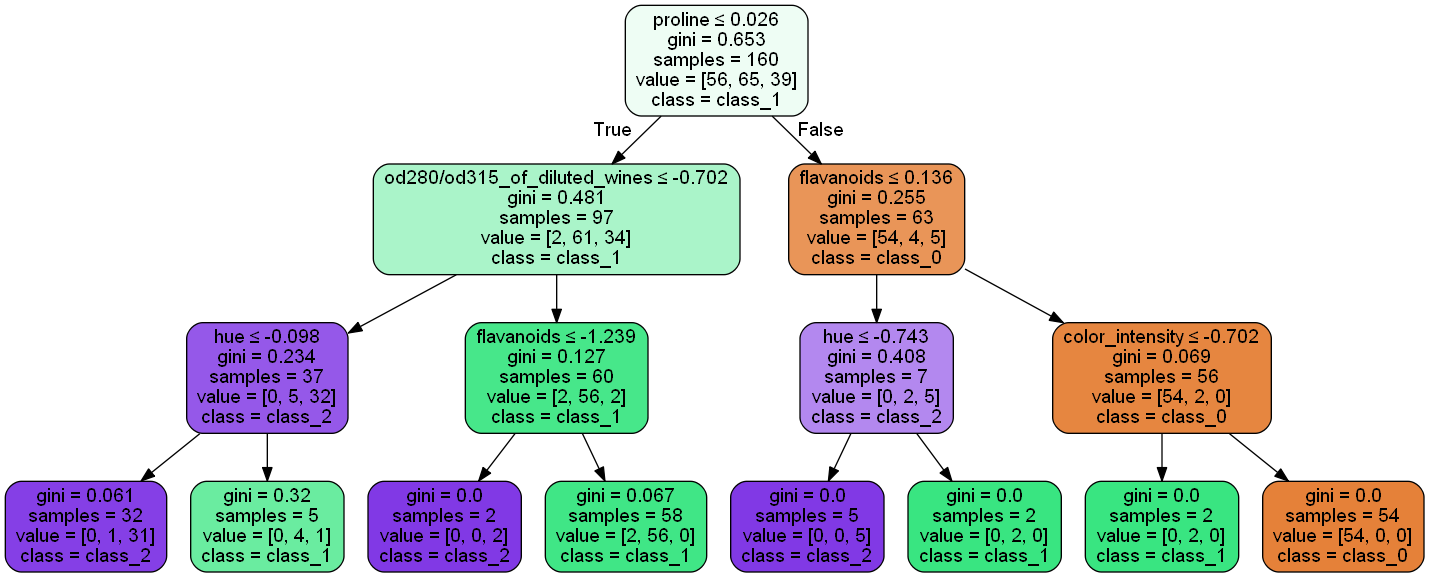
decision tree에서 node가 분리되는 것은 impurity 즉, 불확실성에 의해서 수행된다. 불확실성이란 각 node에서 label의 동질성을 측정한 것인데, 간단하게 말하자면 해당 node의 split조건에 의해 얼마나 잘 분리되지 않았는지에 대한 지표이다. 바로 이 불확실성을 측정하는 지표로 Gini계수와 entropy가 존재한다. 여기서 information gain이라는 개념이 등장하는데, 이는 분리 전과 후의 entropy값의 차이를 의미한다. 즉, information gain이 크다는 것은 entropy의 값이 많이 줄었다는 것을 의미하고 이는 split 조건에 의해 잘 분리되었음을 의미한다. 반면 Gini계수는 target이 되는 attribute 값의 확률분포의 차이를 측정하고 가장 적은 값을 갖도록 node를 분리한다. 간단하게 말하자면 node를 분리할 때 어떤 값의 차이를 기준으로 분리하는지가 바로 Gini계수와 entropy의 차이라고 말할 수 있다. 분석가들은 entropy를 사용했을 때의 성능이 조금 높은 경향이 있다고는 하지만 어떤 기준을 사용하든 decision tree의 성능에는 큰 차이를 주지 않는다고 한다. Entropy의 경우 로그 계산을 수행하기 때문에 Gini 계수의 계산속도보다 약간 느릴 수 있으므로, 계산속도를 향상시키고자 한다면 Gini계수를 기준으로 트리 모델을 선정하면 될 것이다.



**그림2. 각 feature의 중요도**



**그림3. Entropy를 기준으로 분리**



**그림4. Gini계수를 기준으로 분리**

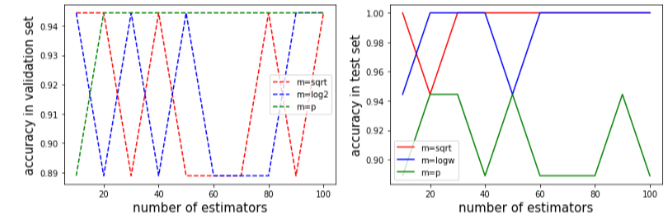
그림3과 4는 각각 entropy를 사용했을 때와 Gini계수를 사용했을 때 생성된 트리 모델을 그림으로 나타낸 모습이다. root부터 필요한 feature들을 기준으로 split하는 것을 볼 수 있다. 이 때, Root 노드부터 높은 중요도의 feature의 값을 기준으로 split하는데, 각 특성의 중요도를 측정한 그림2를 보면, 중요도가 높은 특성들(proline, color intensity, flavonoids)이 트리의 높은 계층에서 분리의 기준으로 사용되는 것을 확인할 수 있다. 엔트로피 분류모델의 level-3에 엔트로피가 0.8로 높은 leaf node가 있기는 하지만, 4개의 sample들 중에 한 하나를 제외한 나머지 샘플들은 같은 클래스이므로 성능저하에 크게 영향을 주는 수치로 보이지는 않는다. Gini 계수를 사용한 트리에서도 마지막 레벨에 0.32로 비교적 높은 수치를 가진 leaf node가 있지만 마찬가지로 성능저하에 큰 영향을 주는 수치로 보이지는 않는다. Depth가 좀 더 깊었더라면 완벽하게 분류했겠지만, 이는 training data set을 학습시킨 결과이기 때문에 depth가 커지면 overfit의 우려가 있다.

\* 추가적으로 Decision tree 모델에서는 scaling된 data와 원래 data 모두 분류해봤는데, 생성된 트리 모델도 거의 일치했는데, 이를 통해 wine data는 그렇게 massive한 data가 아니라는 것을 알 수 있었다.

1. **Ensemble (decision tree-based)**
   1. **bagging과 random forest간의 비교**

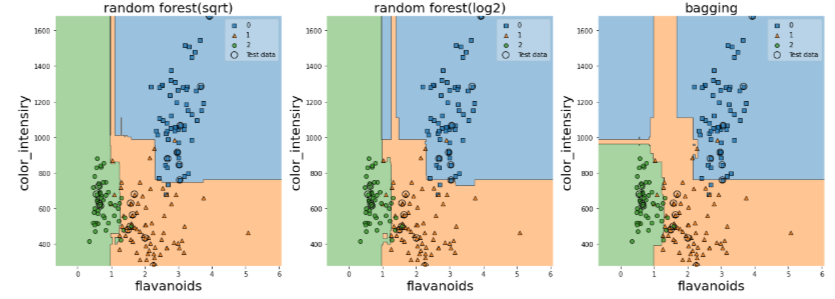
먼저 bagging과 random forest 모두 training set으로부터 여러 개의 subset을 랜덤하게 뽑아 여러 개의 decision tree를 이용해 예측한 뒤 과반수 혹은 확률을 이용해 최종 예측 값을 결정하는 ensemble approach의 일부이다. 다만 다른 것은 bagging에서는 각 tree모델을 만들 때 dataset의 전체 feature(p개)를 이용한다는 것이고, random forest는 각 tree를 구성하는 feature를 전부 사용하는 것이 아니라 무작위로 p개의 feature들 중에 m개를 뽑아 구성하는 것의 차이이다. m개의 feature를 뽑아 사용하는 이유는 영향력이 강한 feature가 존재하는 경우, 모든 bagging된 tree들이 모두 비슷하게 생성되고, tree들로부터 생성된 예측 값들은 다양하지 않은 결과를 나타내게 된다. 다양하지 않은 예측 값들의 평균을 구해서 최종 예측 값을 결정하는 것으로 variance를 많이 낮출 수 없게 된다.

Random forest에서 tuning이 필요한 hyper parameter로 트리의 개수, feature의 개수, 트리의 깊이, 분할하기 위한 최소한의 샘플 수, leaf node가 되기 위한 최소한의 샘플 수 등이 있다. Bagging과 random forest를 비교하기 위해 feature의 개수(m값만 다르게 설정하고 나머지 parameter들은 동일하다는 전제하에 모델을 생성하고 성능을 측정했다.



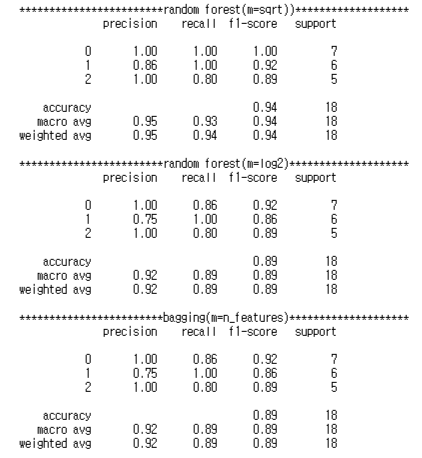
**그림1. Estimators의 변화에 따른 성능 측정**

3개의 모델을 만들고 각각 max depth=3으로 고정시키고 Feature의 개수를 전체 feature를 모두 사용하는 경우(bagging)와, 전체의 제곱근, 그리고 log2(전체)로 설정하여, estimator의 개수를 증가시키면서 성능을 확인해보았다. 전반적으로 전체 feature를 모두 사용하는 것보다 몇 개를 뽑아 사용하는 경우에 조금 더 성능이 잘 나왔다. Bagging 기법도 나쁘지 않은 성능이지만, variance를 낮추기 위해서는 안정적으로 random forest 방식으로 분류하는 것이 선호된다.



**그림2. Wine data의 color intensity와 flavonoids특성만 고려해 분류**

위의 그림2는 wine data에서 두개의 특성만 추출한 뒤, feature의 개수를 다르게 한 모델들에 각각 학습 데이터로 학습시킨 뒤, 테스트 샘플을 각 모델로 분류했을 때의 모습을 보여준다(O친 부분이 test 샘플들이다). 전반적으로 3개의 모델 모두 학습 데이터를 잘 분류한 것처럼 보이고, 테스트 샘플에 대해서도 잘 분류한 것으로 보인다.



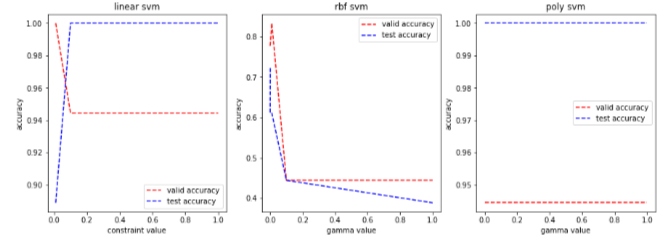
**그림3. 각 모델의 metric 분석**

각 모델에서 test 샘플들을 얼마나 잘 분류했는지 알아보기 위해 precision, recall 점수를 측정해보았더니, feature를 전체 feature수의 제곱근만큼 뽑아 사용한 모델에서 test sample들을 각 클래스 별로 나쁘지 않게 잘 분류했다. 반면 모든 feature를 사용한 bagging 모델에서는 random forest 모델만큼 잘 분류하지는 못하였다. bagging과 random forest의 가장 큰 차이점은 dataset의 각 특성들의 값들의 영향력이 얼마나 차이가 큰지에 따라 다른 것 같다. feature들의 영향력이 비슷비슷하다면 bagging 방식이나, random forest방식이나 둘 다 성능이 좋으므로, 계산속도나 계산량과 같은 다른 요소에 근거해 분류 모델을 선택하면 될 것 같다.

1. **SVM**
   1. **linear SVM과 kernel trick이 적용된 SVM간의 비교 (kernel의 종류는 상관없음)**

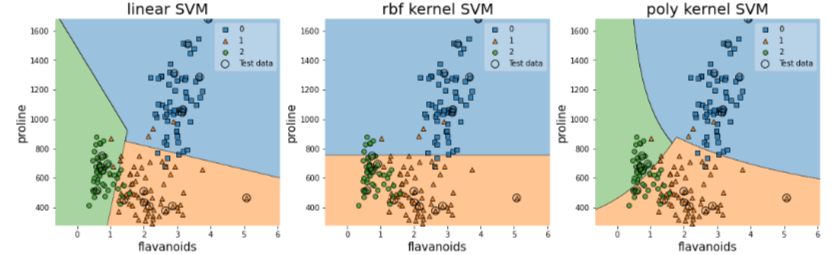
Linear SVM 모델은 데이터를 선형을 분류할 수 있을 때 사용하며, 분류선의 마진을 최대로 하는 모델이다. 반면 선형으로 분류되지 않는 데이터의 경우 차원을 높여서 선형으로 분리되도록 해야 하는데, 이 때 고차원에서의 계산은 직관적으로도 힘들고 계산량도 많기 때문에 kernel trick이라는 유용한 함수를 이용해 고차원에서의 계산을 대체할 수 있다. 사용자는 kernel 함수의 내부를 알 필요 없으며, 몇 개의 유명한 여러 kernel 함수들을 적용시켜 성능이 좋은 모델을 선택하면 된다. 실제 data에는 선형으로 분류되지 않는 data들이 많기 때문에 이러한 kernel trick이 적용된 모델은 유용하게 사용된다.

Linear SVM에서 Tuning이 필요한 hyper parameter로는 C값이다. C값은 margin을 빡빡하게 할 것인지 아니면 soft하게 할 것인지를 결정해주는 요소로, C값이 작아질수록 약간의 오차는 허용하는 것을 의미한다. RBF Kernel trick을 이용하는 SVM에서는 C값과 추가로 gamma 값을 결정해야한다. SCIKIT LEARN 페이지에서 정의하고 있는 Gamma값은 단일 training set이 영향을 미치는 정도라고 되어있으며, 감마 값이 낮으면 멀고, 높으면 가깝다는 것을 의미한다고 한다.



**그림1. Linear SVM과 RBF, poly SVM의 hyper parameter 값에 따른 성능**

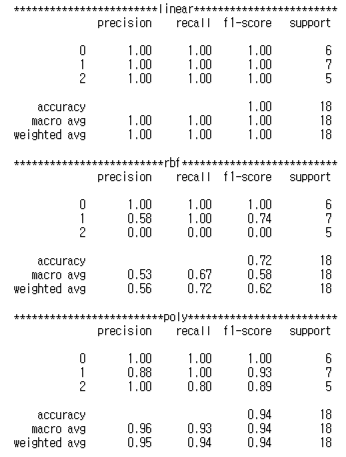
Linear SVM에서는 c의 값을 0.01, 0.01, 1로 바꿔가며 측정했으며, RBF와 poly SVM의 경우 gamma값을 0.0001, 0.001, 0.01, 0.1, 1로 설정해서 측정해봤다. 대체적으로 linear SVM과 poly kernel trick SVM으로 분류했을 때는 validation set에 대한 성능도 괜찮고 test set에서의 최종 성능도 잘 나오는 반면, RBF 커널을 사용했을 경우 성능이 좋게 나오지는 않았다. 이를 통해 wine data를 굳이 커널 트릭을 이용한 SVM으로 분류할 필요가 없으며, kernel trick을 사용하고자 한다면, 적절한 trick을 골라야 한다는 것을 알 수 있다.



**그림2. 2개의 특성으로 train set을 분류한 3가지 모델**

위의 그림2는 각각 wind data에서 중요도가 높다고 평가되었던 feature 2개(proline, flavonoids – decision tree 모델 참고)에 해당하는 value들만을 training set과 test set으로 나누어, 각 모델을 training set으로 학습시켰을 때, 학습의 정확도와, 각 모델로 test set을 분류했을 때를 보여준다. 두 feature 관점에서 보면 두 특성 값이 모두 클수록 0번 class로, 둘 다 작을수록 2번 class로, 그리고 나머지 경우 1번 class로 분류되는 것을 확인할 수 있다.

분류기의 성능면에서 linear SVM과 Poly kernel trick을 사용한 SVM에서는 학습도 잘 되었고, test set도 거의 모두 정확하게 부류하고 있음을 알 수 있다(각 plot에는 178개의 샘플들이 전부 나타났으며, O친 부분이 test data에 해당되는 샘플들이고, 나머지들은 training data에 해당된다).



**그림3. 각 model의 metric 평가표**

위의 그림3 각 모델에서의 예측 정확도를 알아보기 위해 precision, recall값을 살펴보았다. RBF SVM은 0번 클래스에 대해서는 나쁘지 않게 분류했지만, 전체적으로는 낮은 성능을 보이며, poly SVM도 완벽하지는 않지만 나쁘지 않게 각 class들을 분류하고 있다. Linear SVM의 경우 test set의 샘플들을 각 class 별로 정확하게 분류했음을 알 수 있다.

**[2] Cross-validation을 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 하이퍼 파라미터와 모델을 찾아낸다.

최적화할 모델은 [1]을 기반으로 하며, 가장 좋은 성능을 가진 모델과 하이퍼 파라미터를 기반으로 보고서를 작성한다.

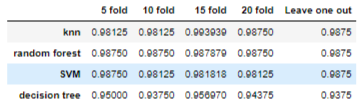
최종적인 성능은 [1]과 동일한 10%의 test set에서 측정하고, 나머지 90%를 cross-validation을 이용하여 모델 최적화를 진행한다.

보고서 작성 요령

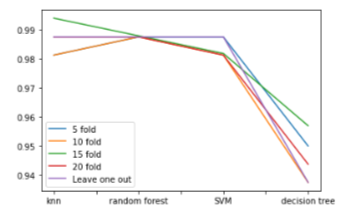
cross-validation을 적용하여 정확도가 가장 높은 하이퍼 파라미터와 모델을 찾는 과정에 대해 설명하고 해당 결과에 대해 EDA한 시각화 자료와 통계량 등을 이용하여 해석한다.

먼저 위의 문제 1과는 다르게 모든 모델에 대해 정규화 된 데이터를 사용했으며, 각 모델에 대해 fold값을 5, 10, 15, 20, 그리고 leave one out을 적용시키고, 아래와 같이 각 모델 별로 사용해볼 hyper parameter들을 선언한 다음, grid search cv를 이용해 최적의 모델을 선정해보았다.

* KNN: neighbors (1~30), p (1,2)
* Decision tree: criterion (entropy, Gini), max depth (1~10)
* SVM: C (0.01, 0.1, 1, 10, 100, 1000), gamma (1, 0.1, 0.01, 0.001, 0.0001), kernel (RBF, poly, Sigmoid, linear)
* Random forest: estimators (1, 5, 10, 15, 20), criterion (entropy, Gini), max depth (2, 3, 4, 5)



**그림1. Fold값 설정에 따른 각 모델에서의 best score**



**그림2. 그림1을 plot으로 표현**

위의 그림1을 통해 fold 값을 각각 다르게 설정했을 때, 각 모델에서 grid search를 수행했을 때, optimal한 hyper parameter들이 적용된 모델들의 best score를 data frame으로 확인했으며, 이를 plot에 나타냈더니 그림2와 같이 표현되었다. 매번 실행할 때마다 측정값은 다르게 나왔지만, 전반적으로 decision tree의 경우 어떤 K fold cross validation으로 모델을 평가해도 가장 낮은 성능이 나왔으며, KNN, random forest 모델은 가장 좋은 성능의 모델로 측정되었다. 수업시간에 KNN모델이 optimal에 가깝다고 배웠는데, 이는 각 모델을 비교하면서 알 수 있었다.



**그림3. 각 최적 모델의 설정 값**



**그림4. Test set으로 최종 모델 성능 평가**

Cross validation 결과 가장 높게 평가되었던 모델은 KNN이었지만 이는 training set에서 validation set에 대한 성능이므로, 최종적으로 모델을 평가할 때는 test set을 사용해야한다. Test set으로 최종 평가 결과, Linear SVM 모델이 완벽하게 샘플을 분리했으므로, 그림3에서 보이는 것처럼 설정된 SVM모델을 최종 모델로 선정하게 되었다. Train set과 test set을 나눌 때, random 하게 나누기 때문에 매번 결과는 다르게 나올 수 있지만, 전반적으로 SVM, KNN, Random forest 모델 모두 비슷한 성능을 보이기 때문에, 최종 모델은 계산속도나, 메모리 등, 사용자가 기호에 맞게 선택하면 될 것 같다.

**참고 사이트**

* <http://rasbt.github.io/mlxtend/user_guide/plotting/plot_decision_regions/>
* <https://woolulu.tistory.com/3?category=778360>
* <https://www.analyticsvidhya.com/blog/2020/03/beginners-guide-random-forest-hyperparameter-tuning/>
* <https://medium.com/@mohtedibf/indepth-parameter-tuning-for-decision-tree-6753118a03c3>
* <https://scikit-learn.org/stable/modules/grid_search.html>