Appendix A. 머신러닝 주요 개념

머신러닝을 수행하기 위해서는 분류, 회귀, 그리고 성능 평가에 대한 명확한 이해가 필요합니다. 분류 항목에서는 분류 개념, 이진 및 다중 분류, 불균형 클래스, 분류 평가 지표를 소개합니다. 회귀 항목에서는 회귀 개념과 회귀 평가 지표를 소개합니다. 이어서 강력한 성능 평가 툴인 교차 검증과 그리드 서치 기능을 소개합니다.

A.1 이진 및 다중 분류

분류는 주어진 입력 변수를 어떠한 기준에 의해서 나누는 것을 의미합니다. 이렇게 나눈 결과로서의 레이블(즉, 분류 결과명)을 출력 변수에 저장하게 되는데, 그 레이블을 **클래스**라고 지칭합니다.

예를 들어 자동차에는 제조사, 국적, 종류, 색상, 연비 등 여러 특성이 있습니다. 특정 기준을 근거로 자동차를 다양하게 분류할 수 있습니다. 자동차 제조사의 국적으로 구분하면 수십 개 국가명이 분류의 클래스가 됩니다. 이렇게 분류의 클래스가 여러 개, 정확하게는 3개 이상 있으면 **다중 분류**라고 합니다. 그런데 제조사의 국적을 좀 더 간추려서 국산 차와 외제 차, 두 가지 항목으로 분류할 수도 있습니다. 이처럼 분류의 클래스가 2개면 **이진 분류**라고 합니다.

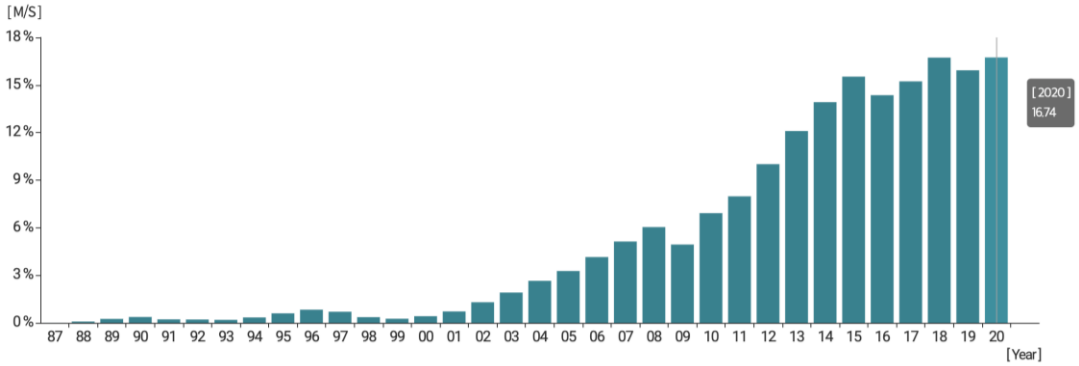
실제 현업 머신러닝 분석에서는 이진 분류를 더 많이 쓰고, 다중 분류는 필요시에만 사용합니다. 그 이유는 두 가지입니다. 첫째, 이진 분류는 출력 변수의 클래스를 0과 1로 치환하여 설정하는데 분석에서 관심 있는 대상을 1로 두면 분석 결과를 매우 쉽게 이해할 수 있습니다. 둘째, 다중 분류는 이진 분류의 응용이기에 이진 분류를 철저히 공부해두면 다중 분류도 쉽게 실시할 수 있습니다.

수업 에피소드!
졸업 학기에 팀 프로젝트로 주택 시장 가격 분석을 실시했던 당시, 우리 팀은 프로젝트 초반에 가격 분석을 분류 모델로 하자고 합의를 했습니다. 그리고 누군가가 이렇게 말을 덧붙였습니다. "우리가 이번에 다중 분류 기반의 모델을 수립해야 되는 것 아닌가?" 팀 내에서 이런 말이 나오자 나머지는 바로 꿀 먹은 벙어리가 되고 말았습니다.
왜냐하면 수업 시간에 이진 분류를 기반으로 수십 개의 모델을 돌리는 훈련을 받아서 이진 분류는 도사급이 되었는데, 다중 분류는 자신이 없었습니다. 해 본 적도 없고 어떻게 돌리는지도 몰랐습니다. 왜 MSBA에서는 다중 분류에 기반한 머신러닝 모델을 가르쳐주지 않았을까요?
그것은 우리가 SAS Enterprise Miner라는 상용 통계 패키지를 사용해서 머신러닝 기법을 배웠기 때문입니다. 다중 분류가 중요하지 않다는 뜻이 아니라 당시 졸업 학기까지 교과목에 다중 분류에 기반한 모델을 분석하는 방법이 나와 있지 않아서 학생들은 배울 기회가 없었습니다.
힘겹게 졸업 프로젝트를 마치고 나서야 알게 된 것은 파이썬 머신러닝 모델과 텐서플로 케라스 딥러닝 모델은 이진 분류 기반의 모델에서 조금만 응용하면 쉽게 다중 분류 기반의 모델을 작성할 수 있다는 점이었습니다.

A.2 불균형 클래스

클래스는 분류의 결괏값입니다. 이진 분류에서는 클래스의 레이블로 0과 1을 부여하는데, 0과 1의 비율이 5:5에 근접하면 균형 클래스라고 합니다. 5:5에서 한참 벗어나면 불균형 클래스라고 합니다. 클래스의 값이 n개가 있는 다진 분류에서는 각 클래스의 값의 분포 비율이 1/n에 가까워야 균형 클래스이고, 이 비율에서 너무 벗어나면 불균형 클래스가 됩니다. 예를 들어 클래스의 값이 4개가 있다면 이들 클래스의 값의 분포 비율은 1/4, 즉 0.25에 근접해야 균형 클래스가 됩니다.

특정 장소에서 한 시간 동안 지나다닌 자동차를 모두 기록했다고 가정해보겠습니다. 국산 차와 외제 차의 비율은 얼마나 될까요? 다음은 한국수입자동차협회 홈페이지에서 캡처한 자료입니다.



[그림 A-1] 국내 시장 외제 차 연도별 점유율

위 자료에 의하면 2020년 외제 차의 국내 시장 점유율은 16.74%입니다. 이를 반올림하여 약 17%를 외제 차의 국내 시장 점유율로 본다면, 인근 도로를 지나는 차들은 평균적으로 17%는 외제 차이고, 나머지 83%는 국산 차일 것입니다. 이처럼 국산 차와 외제 차의 비율이 5:5에서 너무 많이 차이 나는 것을 전형적인 불균형 클래스로 볼 수 있습니다.

불균형 클래스가 되면 분류 모델의 성능 평가를 하는 데 왜곡이 발생할 가능성이 큽니다. 인근 도로를 지나는 차량을 분류하는 머신러닝 모델을 만드는데, 모델의 분류 결과를 입력 변수와 상관없이 모두 국산 차로 판별하도록 만들었다고 생각해봅시다. 사실 이런 분류 결과를 만들려면 머신러닝 모델까지도 필요 없고, 그냥 출력 변수의 모든 레이블을 국산 차로 지정해주면 됩니다.

그런데 이렇게 단순한 막무가내 예측을 해도 지나가는 차가 국산 차인지 맞출 확률이 83%나 됩니다. 83%는 결코 낮지 않은 성능입니다. 이러한 왜곡된 성능 평가를 방지하기 위해서는 모델이 학습하고 예측하는 입력 변수의 클래스의 비율이 50%(혹은 1/n)에 접근하도록 만들어줘야 합니다.

우리 책의 3.7.2절을 참고하면 불균형 클래스를 갖고 있는 데이터세트를 보완하기 위한 조치를 확인할 수 있습니다.

수업 에피소드!
앞에서 언급한 졸업 학기 팀 프로젝트에서 이진 분류의 출력 변수(타깃 변수)를 만들기 위해서 우리 팀은 가격이 높은 주택과 낮은 주택을 구분해야 했습니다. 통상적으로는 주택 가격 평균 이상의 주택에 1, 평균 미만의 주택에 0으로 설정하면 됩니다. 그런데 부동산에 대해서

조금이라도 알고 있는 중년 아저씨로서 저는 다른 학생들에게 이렇게 설득을 했습니다. "사람들은 집값 인상률이 높은 주택을 사려고 하지, 값이 덜 오르는 집은 관심이 없을 거야. 우리 데이터에서 집값이 상당히 높은 집, 즉 상위 1/3에 드는 집과 그렇지 않은 집으로 타깃 변숫값을 삼자." 그래서 최초의 우리 팀 모델의 타깃 변수는 집값이 상위 1/3에 속하면 1, 그렇지 않으면 0으로 설정하여 중간 보고를 하였습니다.
그런데 이게 사달이 났습니다. 교수님께서는 집값의 범위를 상위 1/3과 나머지 2/3로 나눈 학문적 근거를 요구하셨고, 저희 팀은 답변을 할 수 없었습니다. 저는 팀원들에게 사과를 하고 나서 서둘러 주택 평균 가격 이상과 미만으로 타깃 변수를 재설정해야 했습니다. 그런데 지금 생각해 보니 만약 중간 보고 안대로 집값 상위 1/3과 하위 2/3로 타깃 변수를 설정했다면 불균형 클래스 문제까지 발생할 뻔했습니다. 그래서 이진 분류에서는 연속 변수(예: 집값, 자동차 가격 등)를 이진값 타깃 변수로 만들 때 평균(혹은 중위수)을 기준으로 사용하는 사례가 많습니다.
수업 중에 프로젝트 발표를 하면서 이렇게 엉뚱한 실수를 종종 했지만, 실수를 저지르고 이를 바로잡는 에피소드는 잊혀지지 않고 머릿속에 각인됩니다. 저는 복잡한 머신러닝 모델링 과정을 이런 수많은 실수를 통해서 배웠습니다. 당시에는 힘들었지만 그 과정 때문에 실력을 기를 수 있었습니다.

A.3 분류 평가 지표

머신러닝 모델을 실행하고 나면 성능 평가를 수행해서 더 나은 모델을 찾거나 만들기 위해 노력합니다. 분류를 기반으로 한 머신러닝 모델도 예외는 아닙니다. 실무에서는 정확도(Accuracy) 혹은 ROC AUC값을 기반으로 대부분의 평가를 내립니다. 그러나 예외적인 경우에 다른 평가 지표를 써야 할 때도 있습니다. 이를 위해 전반적인 분류 평가 지표를 살펴보겠습니다.

A.3.1 분류 결과표(혼동 행렬)

머신러닝 모델은 입력 데이터를 넣으면 예측값을 산출해냅니다. 예를 들어 가상의 코로나 감염 판별기가 있다고 가정하겠습니다. 이 판별기에 채취한 샘플 분석 데이터를 입력 데이터에 넣으면 내재된 머신러닝 모델에 의거하여 코로나 감염 여부를 출력합니다. 사람들이 실제로 코로나에 감염됐는지는 관찰값(Observed value)으로 표기합니다. 감염되었을 경우가 양성(Positive), 감염되지 않았을 경우는 음성(Negative)입니다.

관측값(Observed value)	
양성(Positive)	실제로 코로나에 감염된 경우
음성(Negative)	실제는 코로나에 감염되지 않은 경우

머신러닝 모델이 내재된 판별기에서 나온 출력값이 예측값(Predicted value)입니다.

예측값(Predictive value)	
양성(Positive)	코로나에 감염된 것으로 예측된 경우
음성(Negative)	코로나에 감염되지 않은 것으로 예측된 경우

관측값과 예측값의 조합을 하나의 표로 종합하면 다음과 같으며, 이를 **분류 결과표**(혼동 행렬, confusion matrix)라고 부릅니다.

		관측값(Observed)	
		양성(Positive)	음성(Negative)
예측값 (Predicted)	양성 (Positive)	TP (진단이 맞고 나쁜 소식) 감염됐다고 예측되었고 실제로도 감염됨	FP (덜 위험한 틀린 진단) 감염됐다고 예측되었으나 실제는 건강함
	음성 (Negative)	FN (매우 위험한 틀린 진단) 감염되지 않았다고 예측되었으나 실제로는 감염됨	TN (진단이 맞고 기쁜 소식) 감염되지 않았다고 예측되었고 실제로도 건강함

*__TP__: True positive, __FP__: False positive, __FN__: False negative, __TN__: True negative

A.3.2 정확도, 정밀도, 재현율, F1 스코어

이 절에서는 분류의 성능 평가지표로 정확도, 정밀도, 재현율, 그리고 F1 스코어를 설명합니다. 예제로 정확도를 계산하기 위해 다음과 같은 결과 분류표의 항목별 건수를 설정합니다.

		관측값(Observed)		
		양성(Positive)	음성(Negative)	합(Sum)
예측값 (Predicted)	양성 (Positive)	TP 3건 감염됐다고 예측되었고 실제로도 감염됨	FP 2건 감염됐다고 예측되었으나 실제는 건강함	TP+FP 5건
	음성 (Negative)	FN 1건 감염되지 않았다고 예측되었으나 실제로는 감염됨	TN 4건 감염되지 않았다고 예측되었고 실제로도 건강함	FN+TN 5건
	합(Sum)	TP+FN 4건	FP+TN 6건	총합 10건

*__TP__: True positive, __FP__: False positive, __FN__: False negative, __TN__: True negative

정확도(Accuracy)는 감염된 사람을 감염됐다고 예측하고, 건강한 사람은 건강하다고 예측한 비율입니다. 즉, 총 관측 개수의 합 중에서 예측이 올바른 경우의 비율입니다. 예측이 올바른 경우는 TP와 TN의 경우입니다. 위의 숫자 예에서 정확도는 아래와 같이 0.7로 계산됩니다.

정확도 = 예측이 올바른 경우 / 총 관측 개수

= (TP+TN) / 총 관측 개수 = (3+4) / 10 = 6 / 10 = 0.7

정확도는 양성 클래스(감염)의 관측값 개수가 상당히 적을 경우에 분류 결과의 성능 평가를 왜곡하여 보고하는 경향이 있습니다. 만약 우리가 100명을 관측했는데 그중에서 실은 단 1명만 코로나에 감염된 경우에 분류 머신러닝 모델에서 모든 입력 데이터에 대해 음성 클래스(감염되지 않음)를 출력하게 하면 정확도는 무려 0.99(=99/100)가 됩니다. 0.99라는 믿을 수 없이 높은 정확도

를 나타내지만, 이 경우의 분류 머신러닝의 분류 성능은 믿을 수 없게 됩니다. 따라서 우리는 다른 성능 평가 지표도 살펴봐야 합니다.

두 번째 평가지표는 **정밀도(Precision)**이며, 코로나에 감염됐다고 예측한 건 중에 실제 코로나에 감염된 비율을 의미합니다.

정밀도 = 양성(감염) 예측이 올바른 경우 / 예측값이 양성(감염)으로 나온 모든 경우

= TP / (TP+FP) = 3 / (3+2) = 3 / 5 = 0.6

정밀도는 코로나 진단 키트의 핵심 기능을 진단한다고 할 수 있습니다. 왜냐하면 이 비율이 높아야만 제대로 된 코로나 진단 키트라고 말할 수 있기 때문입니다. 위의 예에서는 정밀도가 0.6으로 나와 조금 아쉬운 평가 지표가 나왔습니다. 정밀도가 60%만 나온 것을 보면 이제 이 진단 키트가 신뢰할 만한 것인가 하고 의심이 가기 시작합니다. 그래서 평가 지표는 여러 개를 살펴보는 것이 좋습니다.

세 번째 평가지표는 **재현율(Recall)** 혹은 민감도(Sensitivity)로 불리며, 실제로 코로나에 감염된 사람 중에 이 진단 키트가 얼마나 올바르게 감염을 예측하는지의 비율입니다.

재현율 = 양성(감염) 예측이 올바른 경우 / 실제 양성인(감염된) 모든 경우

= TP / (TP+FN) = 3 / (3+1) = 3 / 4 = 0.75

정밀도는 코로나 진단 키트의 진단 결과가 내포한 위험성을 나타낸다고도 할 수 있습니다. 위의 결과를 해석하면 이 진단 키트는 실제로 감염된 사람이 검사를 받아도 75%만 감염됐다고 예측한다는 의미입니다. 보건 당국이 우리 사회를 지키기에 이 진단 키트는 재현율이 너무 낮은 수치인 것 같습니다.

네 번째 평가 지표인 **특이도(Specificity)**는 실제 건강한 사람 수 중에서 예측도 건강한 사람이라고 나온 경우입니다. 특이도는 재현율과 대비하여 비교할 만합니다. 당연히 둘 다 높은 수치를 보일수록 좋습니다.

특이도 = 음성(건강하다는) 예측이 올바른 경우/ 실제 음성인(건강한) 모든 경우의 수

= TN / (FP+TN) = 4 / (2+4) = 4 / 6 = 0.67

마지막으로 **F1 스코어**는 정밀도와 재현율을 둘 다 반영하는 지표로서 두 지표의 조화 평균(harmonic mean)입니다. 정밀도와 재현율 모두 상대적으로 비슷한 수치를 나타낼 때 F1 스코어 값이 커집니다. 즉, F1 스코어는 정밀도와 재현율이 비슷하게 좋게 나오는 경우가 중요할 때 사용할 수 있습니다. 계산 결과 F1 스코어는 정밀도 0.6과 재현율 0.75의 중간쯤에 위치한 0.67로 나타납니다.

F1 스코어 = 2 / ((1/재현율) + (1/정밀도)) = 2 * (정밀도*재현율) / (정밀도+재현율)

= 2 * (0.6*0.75) / (0.6+0.75) = 2 * 0.45 / 1.35 = 0.67

A.3.3 ROC 및 ROC AUC값

ROC 곡선(receiver operating characteristic curve)은 X축을 1-특이도, Y축은 재현율(민감도)로 설정하고 그린 그래프입니다. 1-특이도는 다음과 같이 계산할 수 있습니다.

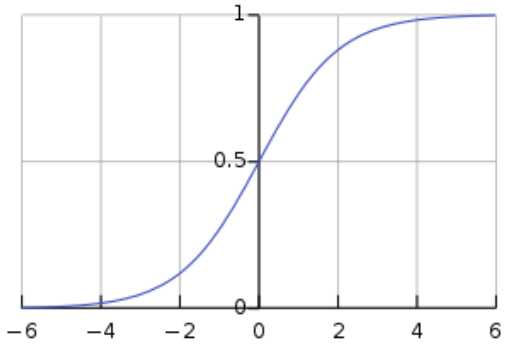
1 – 특이도 = 1 – (TN/ (FP+TN)) = (FP+TN–TN) / (FP+TN) = FP / (FP+TN)

위의 수식에 의하면 1-특이도는 실제 건강한 사람들을 대상으로 감염됐다고 잘못 예측하는 비율로 해석할 수 있습니다. 이는 FP 비율을 가리킵니다. 한편 재현율(민감도)은 정의상 TP 비율을 의미합니다.

종합해보면, ROC 곡선은 머신러닝 예측 모델의 TP 비율(실제 감염자를 제대로 감염자로 예측하는 비율)과 FP 비율(건강한데도 감염됐다고 예측하는 비율) 간의 트레이드-오프 상황을 그린 곡선입니다. 우리의 예에서는 X축에 FP 비율 0.33(=1–0.67), Y축에 TP 비율 0.7로 ROC 곡선상에 점을 찍을 수 있습니다.

ROC 곡선이 이름 그대로 곡선을 이루기 위해서는 임곗값의 개념에 대한 이해가 필요합니다. 예를 들어 로지스틱 함수에서는 원 데이터(변수 x)를 입력하면 0과 1 사이의 값으로 반환합니다. 로지스틱 함수의 정의는 다음과 같습니다.

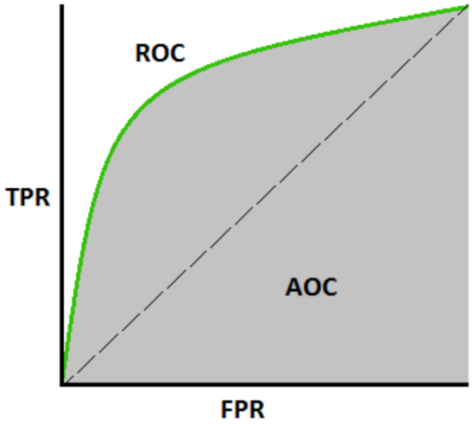
$$f(x) = \frac{1}{1 + e^{-x}}$$



[그림 A-2] 로지스틱 함수 그래프

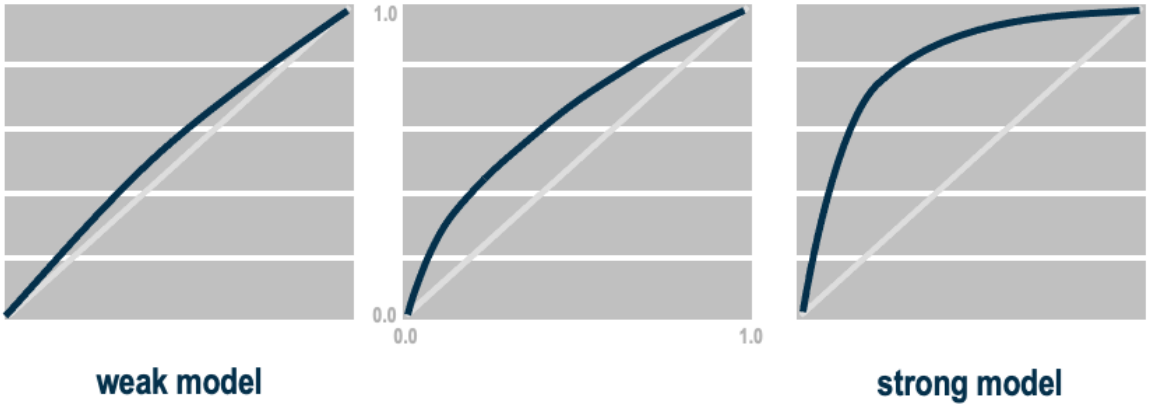
아울러 로지스틱 회귀 모델에서는 로지스틱 함수의 값이 0.5 이상이면 1로 최종 출력하고, 0.5 미만이면 0이라고 최종 출력합니다. 이때 기준인 0.5를 분류 확률 결정 임곗값(threshold)이라고 부릅니다. 즉, 임곗값은 양성(positive) 예측값(1)을 결정하는 확률 기준점을 의미합니다. 영어의 threshold라는 의미를 음미하면 더욱 이해가 잘 될 것입니다. 예를 들어서 임곗값을 0.75라고 재설정하면 로지스틱 함수가 변환하는 값이 0.75 이상이면 1, 0.75 미만이면 0으로 타깃 변수 예측값을 출력합니다.

이 임곗값을 바꿀 때마다 모델의 TP 비율과 FP 비율을 나타내는 점의 위치가 달라집니다. 이 점들의 궤적을 X축과 Y축의 2차원 평면에 그린 것이 ROC 곡선입니다. 여기서는 로지스틱 모델을 들어 ROC 곡선을 설명했지만, ROC 곡선은 모든 이진 분류 모델에서 작성할 수 있습니다.



[그림 A-3] ROC 곡선(출처: www.towardsdatascience.com)

주어진 임곗값마다 FP 비율 대비 TP 비율이 높을수록 더 좋은 모델이므로 ROC 곡선을 그리면 주어진 X축의 값에 대응하는 Y축의 값이 클수록(즉, 그래프의 높이가 높을수록) 머신러닝 모델의 분류 성능이 좋아지게 됩니다. 이는 ROC 곡선의 Y값들의 궤적이 최대한 1에 가까우면 좋다는 의미입니다(ROC 곡선의 X축의 총 길이가 1이고, Y축의 총 길이가 1이기 때문에 1*1=1이 최대 면적). 따라서 ROC AUC값이 클수록, 즉 1에 가까울수록 좋은 지표입니다.



[그림 A-4] Weak model과 Strong model(출처: Applied Analytics SAS EM Course Notes, 2017)

타깃 변수의 균형 분포 시 정확도와 ROC AUC값을 모두 보고, 가급적 두 지표가 동시에 높은 모델을 최적 모델로 선정해야 합니다. 두 지표 값이 상당히 차이가 나면 원인을 파악하고 대처하는 게 좋습니다. 두 지표가 크게 차이 나면 두 지표가 동시에 상위권에 있는 모델을 선택할지, 아니면 한쪽이 월등한 지표 결과에 따라 모델을 선택할지는 모델 및 데이터를 다시 들여다보고 판단해야 합니다. 일단, 정확도와 ROC AUC값 차이가 크면 좋은 현상이 아닙니다.

머신러닝 팁!

다양한 분류 평가 지표를 어느 경우에 쓰는 것이 적절한지 요약하면 다음과 같습니다.

- 정확도: 타깃 변수 클래스가 균형 잡힌 분포를 보이고, FN과 FP 비율이 유사할 때 적용
- F1 스코어: 타깃 변수 클래스가 불균형 분포를 보이고, FN과 FP 비율이 차이가 날 때 적용
- 재현율(=민감도): 당뇨병이나 암 진단처럼 환자를 병이 없다고 잘못 진단하면(FN) 큰 피해가

예상될 때 적용
  - 특이도: 마약 테스트(drug test)에서처럼 마약 중독자만 감옥에 보내기 위해 모든 TN을 제대로 식별하고, FP가 발생하는 것을 용납할 수 없는 경우에 적용
  - 정밀도: 스팸 메일을 걸러내는 것처럼 TP 결과를 더 확실하게 받아들여야 할 때, 즉 내용을 확인하지 않아도 해당 메일을 삭제할 수 있을 정도의 확실성이 필요할 경우에 적용

A.4 회귀

수업 에피소드!
머신러닝 모델을 배우다가 회귀 모델을 배울 때는 머릿속에서 '내가 통계 수업을 듣고 있는 것이 아닌가?'라는 의문이 떠나지 않았습니다. 사실 머신러닝 회귀 모형에서는 이론적으로는 통계학 수업과 동일한 내용을 배웠습니다. 그렇다면 머신러닝과 통계학의 관계는 무엇일까요?
통계학에 컴퓨팅 성능과 알고리즘을 도입한 것이 머신러닝입니다. 머신러닝의 기초에는 통계학이 자리 잡고 있습니다. 때문에 머신러닝은 요즘 시대에 컴퓨터의 도움을 받는 통계 기법이라고 볼 수도 있습니다. 통계학은 머신러닝 모델의 이론적 기초를 맡고 있는 주요 엔진입니다.

분류 모델에서는 머신러닝 모델의 출력값이 범주를 나타내는 레이블로 나옵니다. 이에 반해 회귀 모델은 머신러닝 모델의 출력값이 연속적인 숫자값을 갖는다는 것이 가장 큰 차이점입니다. 때문에 분류 평가 지표와 회귀 평가 지표도 달라집니다.

예를 들어서 상품 가격, 주식 가격 등을 추정하고 싶은 경우는 기본적으로 회귀 모델이 됩니다. 반면에 가격을 높은 가격대 범주와 낮은 가격대 범주로 나누고, 이 범주를 추정하는 경우는 분류 모델이 됩니다.

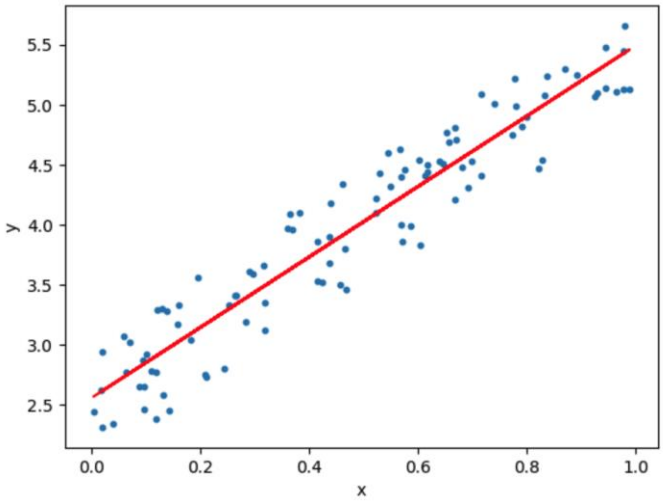
회귀 모델(Regression model)은 모델명 그대로 연속적인 값을 추정하는 회귀 작업을 수행하지만, 로지스틱 회귀 모델(Rogistic Regression Model)은 출력 결과가 범주(0 혹은 1)를 나타내므로 분류 모델이 됩니다. 회귀 모형은 다음과 같이 나타낼 수 있습니다.

$$Y = a*X + b$$

여기서 X는 입력 변수, Y는 출력 변수입니다. a는 계수, b는 절편입니다. 가장 쉽게 X 입력 변수가 하나인 경우를 가정해서 설명하겠습니다. X를 자동차 고유의 최고 속도라 하고, Y를 자동차 가격이라고 설정합니다. 회귀 모형의 목표는 다음과 같습니다.

- 1단계: X, Y 변수가 흩뿌려진 공간에 이들을 대표해서 가로지르는 선을 구합니다.
- 2단계: 추가로 입력 변수를 주면, 1단계에서 구한 선의 수식(즉, 모델)을 이용해서 출력 변수를 예측합니다.

이를 그래프로 설명해보겠습니다.



[그림 A-5] 회귀 모형(출처: www.towardsdatascience.com)

원래 파란색 점들이 주어져 있다고 해보겠습니다. 파란색 점은 입력 변수 X 좌표와 출력 변수 Y 좌표로 이루어집니다. 출력 변수의 값이 주어졌다는 의미에서 이 학습은 지도 학습입니다. 즉, 회귀 모델은 지도 학습입니다. 회귀 모형의 1단계 목표는 파란색 점(점 하나마다 특정 자동차의 최고 속도와 가격을 나타냄)들을 대표하는 직선을 구하는 것입니다. 참고로, 우리의 예에서는 1차 함수만 다루기 때문에 파란색 점들을 대표하는 궤적이 직선이 되지만, 회귀 모형이 고차 함수이면 대표 궤적은 곡선이 됩니다.

그렇다면 여기서 의문이 생깁니다. 어떤 기준에 의해서 파란색 점들을 대표하는 직선을 구할 수 있을까요? 그 기준은 바로 최소 제곱법입니다. 각 점에서 빨간색 선분까지 수직선을 내린 거리가 오차인데, 이 오차를 구하고 제곱한 후 모두 합합니다. 이를 오차 제곱합(Sum of Squared errors)이라고 합니다.

이 오차 제곱합이 작아지도록 (가상의) 빨간색 선분의 위치를 조정합니다. 오차 제곱합이 더 이상 작아지지 않는 빨간색 선분의 위치가 최종적으로 회귀 모형식이 됩니다. 빨간색 선분의 위치가 정해지면 계수인 a와 절편인 b를 구할 수 있습니다. 예를 들어 자동차 가격 Y와 최고 속도 X의 회귀 모형식이 다음과 같이 나왔다고 가정하겠습니다. 참고로 가격 Y는 1단위를 만 원으로 설정합니다.

$$Y = 10*X + 1{,}000$$

머신러닝 모델에서 학습이 일어나 회귀식 모델을 위와 같이 확정하면, 이제부터는 아직 시장에 나오지 않은 미래의 신차 가격을 예측할 수 있습니다. 신차의 최고 시속을 모델에 입력하기만 하면 됩니다. 신차의 최고 시속이 150(km/h)인 차량은 차량 가격 Y가 2,500(만 원)이고, 최고 시속이 200(km/h)인 차량의 가격은 3,000(만 원)이 됩니다.

A.5 회귀 평가 지표

앞서 설명한 분류 평가 지표는 실제 관측값과 모델의 예측값을 비교하여 작성한 바 있습니다. 마

찬가지로 회귀 평가 지표 또한 실제 관측값과 모델의 예측값을 비교하여 작성합니다. 분류 모델은 출력 결과가 범주로 나오기 때문에 분류 결과표(혼동 행렬)를 만들 수 있지만, 회귀 모델은 출력 결과가 연속적인 숫자로 나오기 때문에 실제 관측값과 모델 예측값의 차이(difference)를 직접 구할 수 있습니다. 이러한 차이를 기반으로 회귀 평가 지표를 다음과 같이 구할 수 있습니다.

첫째, **MSE(Mean Squared Error)** 지표입니다. 실제값과 예측값의 차이(difference)를 제곱하고, 이를 평균한 값입니다. 이 지표는 작을수록 좋습니다. 그런데 어째서인지 회귀 모형식을 구할 때 쓴 최소 제곱법의 향기가 납니다. 실은 두 개념이 같은 것이 아닌가 혼동되기도 합니다. 명확하게 다른 것은 최소 제곱법은 주어진 실제값 데이터를 그대로 쓰는 것이므로 예측값은 전혀 쓰지 않습니다. 반면에 회귀 평가 지표의 한 종류인 MSE는 실제값과 예측값을 모두 활용해야 값을 구할 수 있습니다. MSE 수식은 다음과 같습니다.

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

여기서 Y는 실제값, $\hat{Y}$은 예측값, n은 데이터 수입니다.

둘째, **RMSE(Root Mean Squared Error**) 지표가 있습니다. MSE는 매우 훌륭한 지표입니다만, 실제값과 예측값의 차이가 매우 큰 값이 존재하면 이 값의 제곱값은 훨씬 더 커져서 MSE에 왜곡이 발생합니다. 이를 감쇄하기 위해 MSE에 제곱근(root)을 씌운 것이 RMSE입니다. 이 지표 역시 작을수록 좋습니다.

셋째, **MAE(Mean Absolute Error)** 지표도 있습니다. 이 역시 이상값으로 인한 왜곡 현상을 방지하기 위해 고안되었으며, MSE의 제곱항을 절댓값으로 대체한 수식입니다. 이렇게 조치하면 이상값으로 인한 왜곡 현상이 제곱으로 인해 커질 위험이 사라집니다.

$$\mathrm{MAE} = \frac{\sum_{i=1}^{n}|y_i - x_i|}{n}$$

마지막으로 **결정계수 R^2**(R Squared)를 들 수 있습니다. R^2의 정의는 다음과 같습니다.

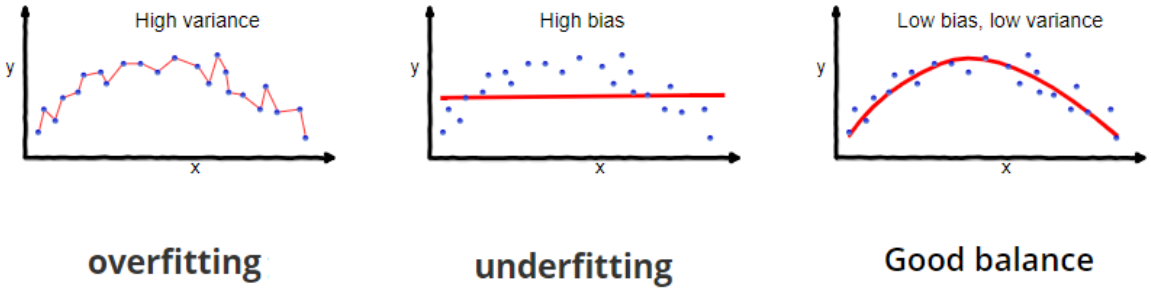
R^2 = 모델에 의해서 설명된 분산 / 전체 분산

즉, R^2는 모델이 설명하는 타깃 변수 Y의 분산 비율입니다.

A.6 교차 검증

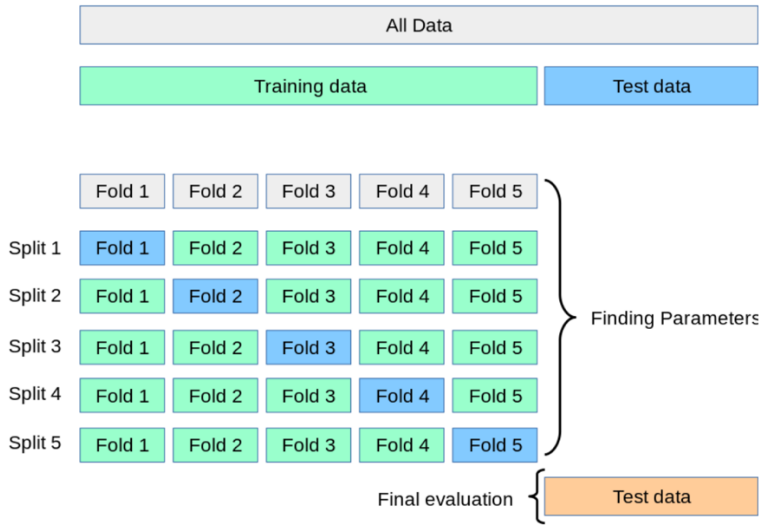
머신러닝 모델에서는 원본 데이터세트를 나눠서 학습 데이터세트와 테스트 데이터세트로 나눕니다. 학습 데이터에서 모델을 학습하며 테스트 데이터세트를 이용하여 예측을 합니다. 다만 머신러닝 모델이 학습 데이터세트에 과도하게 학습한 나머지, 현실에서 미지의 데이터세트 대용으로 투입하는 테스트 데이터세트에서 형편없는 예측을 할 위험성이 있습니다.

이렇게 학습 데이터세트를 과도하게 학습하여 모형을 필요 이상으로 정교하게 만드는 것을 **과적합**(Overfitting)이라고 부릅니다. 과적합을 방치하면 결국 머신러닝 모델이 현실에서는 쓸모없는 모델이 되므로 문제가 됩니다. 반면에 모형이 학습 데이터세트에 대한 학습이 형편없으면 **과소적합**(Underfitting)이라 하며, 이 역시 문제가 됩니다.



[그림 A-6] 과적합과 과소적합(출처: www.towardsdatascience.com)

과적합을 막기 위해서 교차 검증(Cross validation)을 도입해야 합니다. 교차 검증 과정은 다음과 같습니다. 학습 데이터세트를 특정 개수(예: 5개)로 나눕니다. 아래 그림에서 교차 검증이 학습 데이터세트에서만 수행되는 것에 유의하기 바랍니다. 테스트 데이터세트의 최종 평가(final evaluation)는 교차 검증 이후의 과정입니다.



[그림 A-7] 교차 검증(출처: www.scikit-learn.org)

첫 번째 과정은 검증(validation) 데이터세트로 첫 번째 조각의 학습 데이터세트를 지정하고(위의 그림에서 Split 1의 Fold 1), 나머지 4개 조각 학습 데이터세트에서 머신러닝 모델을 학습시킨 후 첫 번째 조각 검증 데이터세트에서 모델의 성능을 측정합니다.

두 번째부터 다섯 번째 과정은 첫 번째와 유사하게 하되 검증 데이터세트를 순차적으로 두 번째, 세 번째, 네 번째, 다섯 번째 조각으로 지정하고, 나머지 4개 조각 데이터세트를 학습 데이터세트로 쓰는 것만 다릅니다.

이처럼 설정한 개수만큼(예: 5개)의 교차 검증 과정을 거치면 모델의 성능이 5회 측정되는데, 이를 평균해서 학습 데이터세트에서의 모델 성능으로 삼습니다. 교차 검증 작업은 다음 절에서 교차 검증이 그리드 서치 기능 안에서 이루어지는 과정을 통해 살펴보겠습니다.

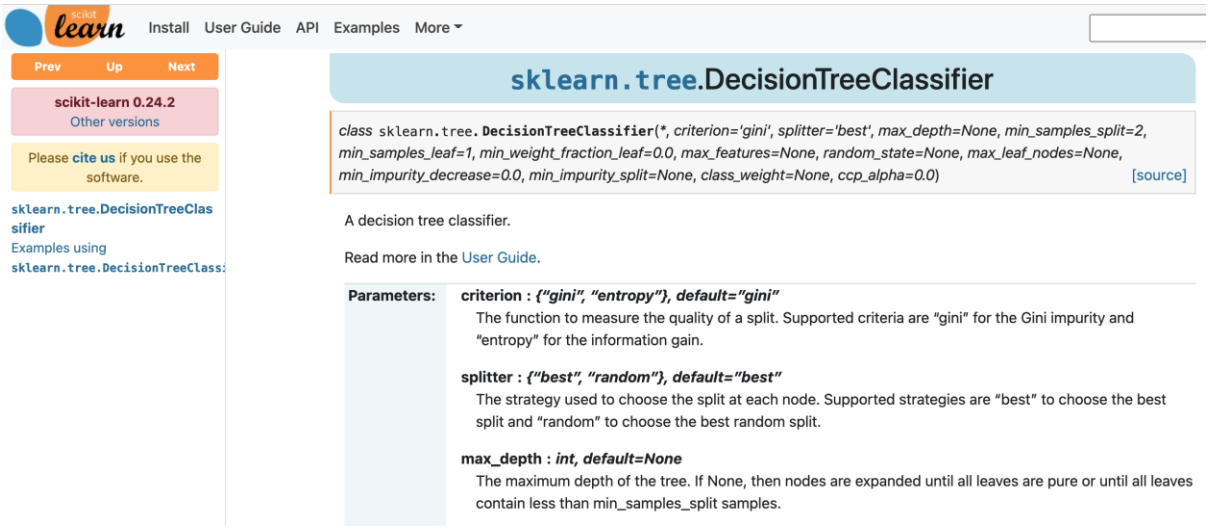
수업 에피소드!
교차 검증에서 핵심은 이 검증이 테스트 데이터세트가 아니라 학습 데이터세트를 이용한다는 데 있습니다. 머신러닝 모델의 최종 목표는 학습 데이터세트에서의 성능이 아니라 테스트 데이터세트에서의 성능을 최고로 올리는 것입니다. 교차 검증은 학습 데이터세트에서의 성능이 우연히 과도하게 높게 나오거나 낮게 나오는 것을 방지하고자 여러 번에 걸쳐 학습 데이터세트를 분할해서 머신러닝 모델을 학습시킵니다.
저를 포함한 MSBA 학생들은 교차 검증에 대해 사실 무지했었습니다. 왜냐하면 상용 통계 패키지인 SAS Enterprise Miner의 모델은 교차 검증이 기본적으로 포함되어 있어서 모델 내에서 자동으로 교차 검증 과정을 수행했기 때문입니다. 그래서 파이썬으로 머신러닝 모델을 구현할 때 교차 검증 개념을 알기가 어려웠습니다.
이쯤 되면 "수업 시간에 도대체 뭘 배운 거야?"라고 질문을 던질 만합니다. 워낙 자동화된 프로세스로만 배웠으니 이런 단점들이 존재했습니다. 그러나 GUI 기반으로 머신러닝 모델을 실행하는 과정을 먼저 배우고 나서 파이썬으로 머신러닝을 구현하는 방식으로 프로젝트를 수행했기에, 2년 동안의 학습 과정이 처음부터 흥미진진했고 끝까지 길을 잃지 않을 수 있었습니다. 각각의 방식은 장단점이 모두 있으므로 각 방식의 장점을 모두 활용하는 것이 바람직합니다.

A.7 그리드 서치

머신러닝 모델을 수립할 때 모델 설정에서 직접 값을 설정해야 하는 옵션값이 있습니다. 이를 파라미터(parameter)라고 합니다. 예를 들어서 결정 트리(Decision Tree) 모델에서는 나뭇가지 모양으로 뻗어 내려가는 결정 트리가 최대로 뻗어내려 갈 수 있는 max_depth를 일일이 지정해 줘야 합니다.

만약 max_depth를 지정하지 않으면 어떻게 될까요? 컴퓨터는 결정 트리 모델의 디폴트 설정에 따라 머신러닝 모델을 돌리게 됩니다. 디폴트 설정으로 머신러닝 모델을 돌려서 최적 결과가 나오면 좋은데, 대부분은 과적합이나 과소적합이 나오기 십상입니다.

결정 트리 모델의 max_depth처럼 직접 값을 설정해야 하는 파라미터는 몇 개나 있을까요? 이를 알아보기 위해 사이킷런 공식 사이트(www.scikit-learn.org)의 검색창에 DecisionTreeClassifier를 입력해봅시다. DecisionTreeClassifier는 사이킷런 결정 트리의 estimator를 의미합니다. 검색 결과, 맨 위에 있는 검색 결과를 클릭하면 다음과 같은 화면이 나타납니다.



위에 보이는 화면에는 DecisionTreeClassifier의 파라미터가 3개 나열돼 있습니다만, 실제 화면에서는 총 12개의 파라미터가 있습니다. max_depth는 그중 하나입니다. max_depth의 기본값은 none으로 설정돼 있어서, 가능한 한 끝까지 결정 트리 가지를 뻗습니다. 그러면 명백하게 과적합이 일어납니다.

머신러닝 모델 수십 개를 설정하고 실행하는 것은 어느 정도 경험이 쌓이면 그렇게 어려운 일이 아니지만, 하나의 머신러닝 프로젝트를 수행할 때는 수십 개의 머신러닝 모델을 돌려서 그중 최선의 모델을 선택해야 합니다. 그런데 수십 개의 모델마다 파라미터값의 조합을 설정해서 결과를 비교하는 것은 시간이 엄청나게 필요합니다. 제 경험상 개별 모델마다 2~5개 정도의 파라미터값 설정이 필요한데, 10개 모델만 돌려도 20~50개의 파라미터값을 설정하고 실행해야 하는 것입니다.

이를 방지해 주는 것이 그리드 서치(GridSearchCV) 기능입니다. 아울러 이 그리드 서치는 앞 절에서 언급한 교차 검증까지 동시에 수행합니다. 그리드(Grid)는 격자무늬를 의미합니다. 예를 들어 기본 모델(예: 결정 트리)에서 A 파라미터값을 3개 주고, B 파라미터값을 4개 주면 총 12(=3*4)개의 파라미터 조합이 생깁니다. A 파라미터값을 X축에 그리고, B 파라미터값을 Y축에 그리면 12개의 사각형으로 이루어진 격자가 그려질 것입니다. 이 모든 그리드에 담긴 기본 모델의 파라미터를 그리드 서치의 하이퍼파라미터라고 부릅니다.

그리드 서치는 이러한 하이퍼파라미터 조합을 기본 모델에 순차적으로 넣고 실행한 후, 최적 하이퍼파라미터 조합을 찾아냅니다. 거기에 교차 검증(CV)까지 추가로 하기 때문에 그리드 서치의 명령어가 GridSearchCV인 것입니다. 그리드 서치 예제를 간략하게 소개하기 위해 책 3.7.3절에서 쓰인 코드와 데이터세트를 빌려오겠습니다. 코드 및 파라미터에 대한 상세한 설명은 책을 참고하기 바랍니다.

그리드 서치를 실행하기 위해서는 사전에 기본 머신러닝 모델을 구축해야 합니다. 그리드 서치는 기본 머신러닝 모델의 파라미터 조합을 탐색하는 알고리즘인데, 어떤 머신러닝 모델을 기본 모델로 쓸 것인지 지정해주지 않으면 아예 작동하지 않습니다. 때문에 다음과 같이 기본 머신러닝 모델로 결정 트리 모델을 파이썬 사이킷런 라이브러리를 통해 설정해줍니다.

우선 사이킷런 결정 트리 estimator인 DecisionTreeClassifier를 사이킷런 라이브러리인 sklearn.tree

에서 불러옵니다. 그다음 DecisionTreeClassifier의 세 파라미터 criterion, random_state, max_depth 값을 아래와 같이 지정해주고, 이 결정 트리 분류기 모델을 tree라는 변수에 저장합니다.

```
# Decision Tree 모델
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
tree = DecisionTreeClassifier(criterion="gini", random_state=0, max_depth=5)
```

파라미터값을 지정하지 않으면 DecisionTreeClassifier는 기본 설정값으로 세팅됩니다. 결정 트리 모델에서 max_depth는 5로 설정하였습니다. 이제 이러한 기본 모델을 가지고 그리드 서치를 구현합니다.

```
from sklearn.model_selection import GridSearchCV
params = {'criterion':['gini','entropy'],'max_depth': range(1,21)}

grid_tree = GridSearchCV(tree, param_grid=params, scoring='accuracy',
                         cv=5, n_jobs=-1, verbose=1)
grid_tree.fit(X_train, y_train)
print("GridSearchCV max accuracy:{:.5f}".format(grid_tree.best_score_))
print("GridSearchCV best parameter:", (grid_tree.best_params_))
```

우선 코드의 처음 두 줄을 살펴보겠습니다.

```
from sklearn.model_selection import GridSearchCV
params = {'criterion':['gini','entropy'],'max_depth': range(1,21)}
```

처음에는 sklearn.model_selection에서 그리드 서치를 불러옵니다. 그리고 기본 모형인 결정 트리 모델에서 사용할 파라미터값을 params라는 변수에 지정합니다. 결정 트리 모델의 criterion값으로는 gini와 entropy를 주었습니다. 그리고 max_depth값으로 1, 2, 3, ..., 20까지의 값을 주었습니다.

range(1, 21) 함수는 1부터 시작해서 20까지 1 간격만큼 숫자를 더하라는 의미입니다. 두 번째 줄에 쓰인 params라는 변수는 그리드 서치 파라미터 param_grid의 값을 담은 변수입니다.

```
grid_tree = GridSearchCV(tree, param_grid=params, scoring='accuracy',
                         cv=5, n_jobs=-1, verbose=1)
```

이 코드가 바로 GridSearchCV 구문이 쓰인 부분으로, 괄호 안의 param_grid, scoring, cv, n_jobs, verbose가 그리드 서치의 파라미터들입니다. 참고로 괄호 안의 tree는 그리드 서치에 쓰이는 기본 모델을 담은 변수입니다.

머신러닝 팁! 파라미터 vs 하이퍼파라미터
파라미터는 머신러닝 모델에서 설정해야 하는 변수를 의미합니다. 예를 들어, DecisionTreeClassifier(criterion="gini", random_state=0, max_depth=26)이라는 코드에서 criterion, random_state, max_depth가 DecisionTreeClassifier의 파라미터입니다.

한편 하이퍼파라미터는 그리드 서치가 다루는 기본 머신러닝 모델(estimator)의 파라미터 말합니다. 더 쉽게 설명하자면 머신러닝 모델의 최적화를 위해서 일일이 값을 지정해줘야 하는 기본 모델 파라미터를 그리드 서치 입장에서 부르는 명칭이 하이퍼파라미터인 것입니다.

그리드 서치는 작게는 몇 개, 많게는 수십 개의 파라미터 조합을 통해 머신러닝 모델을 실행하고 이들의 성능을 평가합니다. 그 성능 평가 기준을 결정하는 파라미터가 scoring입니다. 위의 예에서는 정확도(Accuracy)를 성능 평가 기준으로 썼습니다.

CV 파라미터는 매우 중요한 파라미터입니다. 이 파라미터 설정을 통해서 교차 검증이 그리드 서치 내에서 자동으로 이루어집니다. 여기서는 CV값으로 5를 설정했으므로 하나의 파라미터 조합마다 학습 과정이 5번 일어나고, 검증 데이터세트에서의 성능 평가 결괏값도 5번 발생합니다. 이들 결괏값의 평균을 산출해서 최종적인 검증 세트의 성능 평가 지표로 활용합니다.

이제 다음과 같이 fit 명령어를 실행하면 학습 데이터세트 X_train과 y_train을 통한 그리드 서치 학습이 일어납니다.

```
grid_tree.fit(X_train, y_train)
```

학습 결과를 다음은 학습 결과를 출력하는 코드입니다.

```
print("GridSearchCV max accuracy:{:.5f}".format(grid_tree.best_score_))
print("GridSearchCV best parameter:", (grid_tree.best_params_))
```

그러면 다음과 같이 그리드 서치의 작업 개요와 학습 결과가 출력됩니다.

```
Fitting 5 folds for each of 40 candidates, totalling 200 fits
GridSearchCV max accuracy:0.79534
GridSearchCV best parameter: {'criterion': 'gini', 'max_depth': 4}
```

첫 번째 줄은 그리드 서치의 작업 개요입니다. 40 candidates는 파라미터 criterion값으로 2개, max_depth값으로 20개를 설정하였으니 40개(=2*20)의 경우의 수가 나오는 것을 의미합니다. 그리고 cv 파라미터에 의해 학습 데이터세트를 5개로 쪼개서 교차 검증을 실시해서 도합 200번(=40*5)의 머신러닝 모델 학습(fits)이 이루어졌습니다.

교차 검증 시 가장 높은 평균 정확도는 0.79534입니다. 이때 최적 결정 트리 모델의 criterion은 gini, max_depth는 4로 나왔습니다. 이렇듯 그리드 서치를 통해 얻을 수 있는 최적 모델은 교차 검증을 통해 검증 데이터세트(학습 데이터세트의 일부)에서 가장 높은 평균 정확도를 낸 모델입니다. 이러한 최적 모델은 best_estimator_라는 특성(Attribute)에 저장되고, 이를 사용해서 최적 모델을 통한 테스트세트의 성능을 평가할 수 있습니다.

참고로 타깃 변수의 클래스 분포가 지나치게 한쪽으로 치우쳐 있으면 stratified K 폴드를 써야 하고, 그렇지 않으면 일반적으로 K 폴드를 사용해도 무방합니다. 아울러 GridSearchCV의 교차 검증은 타깃 변수가 이진값이나 다중 클래스(multi class)값을 가질 때 디폴트로 stratified K 폴드를 사용합니다. stratified K 폴드는 타깃 변수의 클래스(값) 분포 비율을 신경 쓰지 않아도 되는 안전장치 역할을 합니다.

A.8 차원 축소

입력 변수가 수십, 수백 개에 이르면 정보량이 지극히 적은 입력 변수들은 입력 변수와 출력(타깃) 변수의 관계를 의미 있게 파악하는 것을 방해합니다. 따라서 컴퓨터의 도움을 받지 못했던 과거 통계학의 대가들은 입력 변수 자체의 수를 줄일 수 있는 방법을 가장 고민하였습니다. 머신러닝과 딥러닝 시대에 들어와서도 차원 축소 방법은 여전히 중요합니다. 이 장에서는 이러한 차원 축소에 대하여 소개합니다.

A.8.1 차원 축소 개요

통계학 기초 과정 교재의 예제에서는 데이터에서 다루는 입력 변수를 하나 혹은 두 개 정도로 설정합니다. 예를 들어서 회귀 모형을 $Y = aX + b$로 두거나 혹은 $Y = a_1X1 + a_2X2 + b$로 두어서 간단하게 설명합니다. 입력 변수와 출력 변수가 하나씩이면 평면에 궤적을 그래프로 그릴 수 있고, 입력 변수가 두 개이고 출력 변수가 하나이면 3차원 평면에 그래프를 그릴 수 있습니다. 이를 통해 직관적인 시각화가 가능합니다.

한편, 입력 변수의 개수가 늘어나면 다음과 같은 문제를 일으킵니다.

- 상관관계
- 과적합
- 차원의 저주

입력 변수의 개수가 많은 경우 높은 상관관계를 보이는 입력 변수들이 존재할 가능성이 큽니다. 입력 변수 간의 높은 상관관계는 회귀 기반 모델 결과에 심각한 왜곡을 불러옵니다. 또한, 입력 변수의 개수가 많은 경우 머신러닝 모델의 과적합을 초래합니다. 이런 경우 입력 변수를 줄이는 것이 과적합을 방지합니다.

차원의 저주는 자료 분포의 희소성(sparseness) 때문에 발생합니다. 변수가 많아지고 변수마다 값들이 드문드문 존재할 경우, 즉 소수의 값을 제외한 나머지 값들이 대부분 0으로 구성된 경우 머신러닝 모델의 백그라운드에서 실행되는 수학 연산에 부하가 걸립니다.

이러한 세 가지 문제 때문에 입력 변수의 개수를 줄이는 작업을 고려해야 합니다. 가장 간단한 방법은 문제가 되는 변수, 즉 상관관계가 높거나 과적합을 일으키거나 자료값이 드문 변수를 제거하는 것입니다. 출력 변수에 거의 영향을 미치지 않는 변수를 제거하는 것도 이에 포함됩니다.

또 다른 방법은 기존의 모든 변수에 가중치를 곱하고 이를 합한 조합(combination)을 몇 개 만드는 것입니다. 이때 기존 변수의 수보다 조합의 수가 적어야 의미가 있습니다. 이렇게 기존의 변수를 가지고 새로운 조합을 만들어내는 과정을 차원 축소(Dimension reduction)라고 합니다.

새로 생성된 조합은 서로 상관관계도 크지 않고, 기존 변수들보다 개수도 적기 때문에 머신러닝

모델의 과적합 위험을 최소화합니다. 아울러 여러 변수의 조합으로 하나의 값을 만들기에, 0값이 많이 포함된 변수들의 조합이라도 조합의 개별 결괏값은 대부분 0이 아닌 값을 가집니다. 때문에 자료의 희소성까지 방지합니다.

A.8.2 주성분 분석(PCA)

주성분 분석은 차원 축소의 대표적인 기법입니다. 주성분 분석은 출력 변수와 관계없이 입력 변수만 가지고 작업하기 때문에 비지도 학습의 일종입니다. 주성분 분석을 통한 차원 축소는 머신러닝과는 별개의 통계 기법인 구조 방정식 모델(Structural Equation Model)에서 필수적으로 쓰이는 기법입니다. 다음은 주성분 분석을 쉽게 설명하는 비유를 인용한 것입니다.

"(콜라)캔을 구겨서 2차원으로 만드는 것처럼 3차원을 2차원으로 감소시키려고 한다면 밟는 행동이 바로 차원 축소 행동이 된다. ... 어떻게 밟아야 좀 더 많은 데이터를 보존할 수 있을까? ... 캔을 눕혀서 밟는다면 캔에 적혀 있는 상표나 색깔을 볼 수가 있다. 다시 말해, 더 많은 정보를 남길 수 있는 것이다" (통계의 아름다움, 리찌엔·하이언 저)

위의 인용문에 대한 추가 설명을 덧붙이겠습니다. 여러분이 다음과 같은 질문을 받았다고 생각해보세요.

빈 콜라 캔을 발로 밟아 부피를 최소화하려는데
어느 방향으로 밟아 찌그러뜨려야 하는가?

이 질문에 대한 직관적인 답은 콜라 캔을 똑바로 세우고 위에서 아래로 밟으면 됩니다. 그런데 이 문제에는 다음과 같은 제약 조건이 따릅니다.

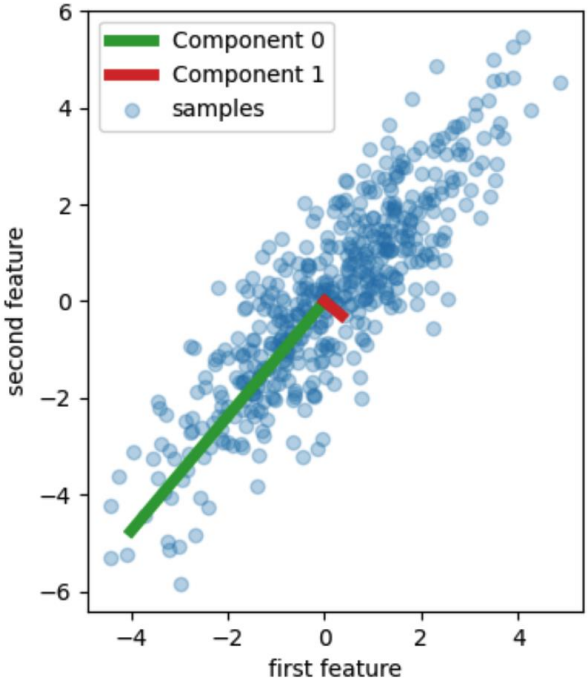
단, 콜라 캔의 표면에 있는 정보를 최대한 보전하면서 찌그러뜨려야 한다.

'으응? 그럼 어떻게 밟아야하지?' 하고 다시 생각해보면, 콜라 옆면에는 브랜드 로고, 상호, 디자인 요소, 제품 정보 등이 있습니다. 콜라 캔의 위아래 면에도 몇몇 정보가 음각돼 있기는 하지만, 그 정보량은 매우 적습니다. 따라서 올바른 대답은 '콜라 캔을 옆으로 뉜 다음, 위에서 밑으로 밟아 찌그러뜨린다'입니다. 그러면 납작해진 옆면에서 비교적 많은 정보를 눈으로 확인할 수 있습니다.

[그림 A-8] 콜라 캔 표면의 정보를 최대한 보전하면서 밟기

콜라 캔의 부피를 최소화하는 것은 차원 축소를 의미합니다. 그리고 캔 표면의 정보를 최대한 보전하라는 조건은 차원 축소 시 정보의 최대 보전을 의미합니다. 이러한 캔 찌그러뜨리기 비유를 염두에 두고 차원 축소 이론을 이해해봅시다.

입력 변수가 두 개 있는 샘플 데이터를 가정합니다. 예를 들어 입력 변수가 X_1, X_2 이렇게 있다고 생각하면 쉽습니다. X_1을 첫 번째 특성, X_2를 두 번째 특성이라고 호칭할 수도 있습니다. X_1과 X_2를 좌표에 나타내면 다음 그림의 파란색 원들이 됩니다. 원들이 겹치면, 겹친 부분은 짙은 파란색으로 표시됩니다.



[그림 A-9] 주성분 추출(출처: www.scikit-learn.org)

위 그림에서 X_1과 X_2의 분포는 양의 비례 관계를 보입니다. X_1이 커짐에 따라서 X_2도 대략적으로 커지는 분포입니다. 이런 경우 상당히 큰 상관관계가 도출됩니다. 입력 변수끼리 큰 상관관계(예: 0.7 이상)를 보이면 다중공선성 문제가 나타나서 회귀 모델 결과가 왜곡됩니다. 따라서 상관관계를 줄이기 위해 차원 축소를 시도합니다.

우선 X_1과 X_2의 분포에서 분산이 어느 방향으로 가장 큰지를 계산합니다. 럭비공처럼 생긴 위 분포에서 럭비공의 장축을 통과하는 직선 방향으로 분산이 가장 큽니다. 다시 말하면 이 방향이 X_1과 X_2의 분산을 가장 많이 설명합니다. 그래서 럭비공의 중심에서 녹색 선분 방향으로 첫 번째 주성분(principal component)을 뽑아냅니다. 간단한 수식으로 첫 번째 주성분을 표기하면 다음과 같습니다.

$$\text{주성분1} = a_1 * X_1 + b_1 * X_2$$

위 콜라 캔 비유를 여기에 적용해보겠습니다. X_1과 X_2의 샘플 데이터가 흩뿌려져 넓게 퍼진 럭비공 같은 형태가 콜라 캔의 원래 정보량입니다. 콜라 캔을 옆면으로 찌그러뜨려야 콜란 캔에 원래

쓰인 정보를 최대한 들여다볼 수 있듯이, 첫 번째 주성분은 럭비공 모양의 장축 방향으로 뽑아내야 합니다.

첫 번째 주성분을 뽑아내고 나면, 두 번째 주성분은 첫 번째 주성분이 설명하지 못하는 X_1과 X_2의 분산 정보를 뽑아낼 수 있는 방향으로 그리면 됩니다. 그것은 첫 번째 주성분의 방향과 수직 방향입니다. 위 그림에서는 빨간색 방향의 선분입니다. 이 역시 수식으로 표현하면 다음과 같습니다.

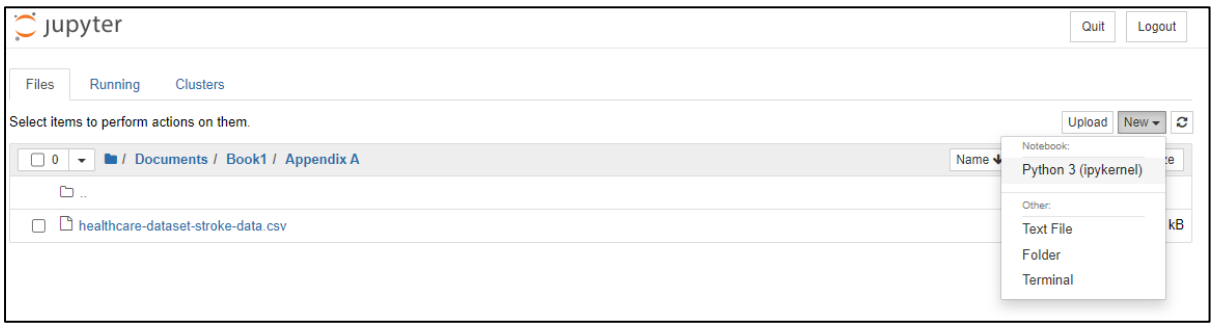
$$주성분2 = a_2*X_1 + b_2*X_2$$

여기서는 두 가지 입력 변수만 가지고 설정하였지만, 만약에 입력 변수가 3개 혹은 N개까지 있다고 해봅시다. 3차원 입력 데이터인 경우에는 세 번째 주성분은 두 번째 주성분과 수직(직교)이 되게 뽑아내면 됩니다. 즉, N차원 입력 데이터는 N-1차원까지의 주성분과 수직이 되게 주성분을 뽑으면 됩니다. 이런 식으로 N개의 입력 변수가 있을 때 최대 N개의 주성분을 도출할 수 있습니다.

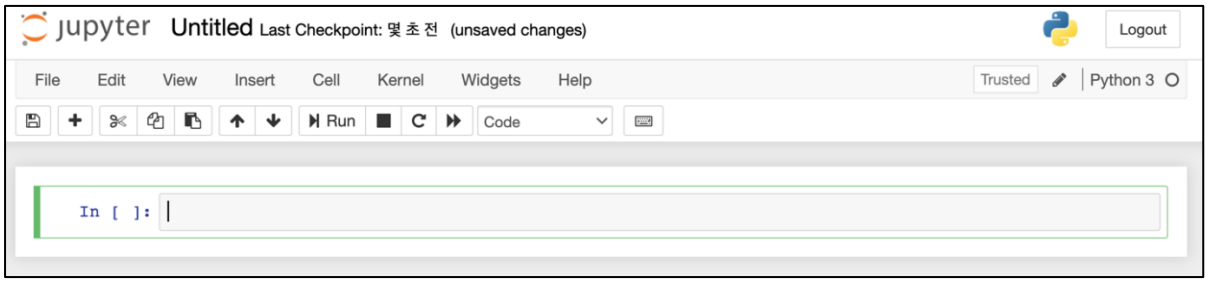
다음은 주성분 분석을 코드로 살펴보겠습니다. 책의 3장 초급 프로젝트에서 사용한 데이터를 불러옵니다. 캐글 사이트에서 다운받을 원본 데이터의 파일명은 healthcare-dataset-stroke-data.csv입니다. 그리고 아나콘다 내비게이터로 주피터 노트북을 엽니다. 주피터 노트북을 기준으로 코드를 설명하지만, 여기서의 모든 코드는 구글 코랩에서도 정상적으로 작동합니다.

컴퓨터의 Documents 폴더 안에 Book1 폴더를 만들고, 그 하위 폴더로 Appendix A 폴더를 만듭니다. 주피터 노트북에서 다음과 같이 Appendix A 폴더를 찾아간 다음, 다운받은 원본 데이터 파일을 Appendix A 폴더에 저장합니다.

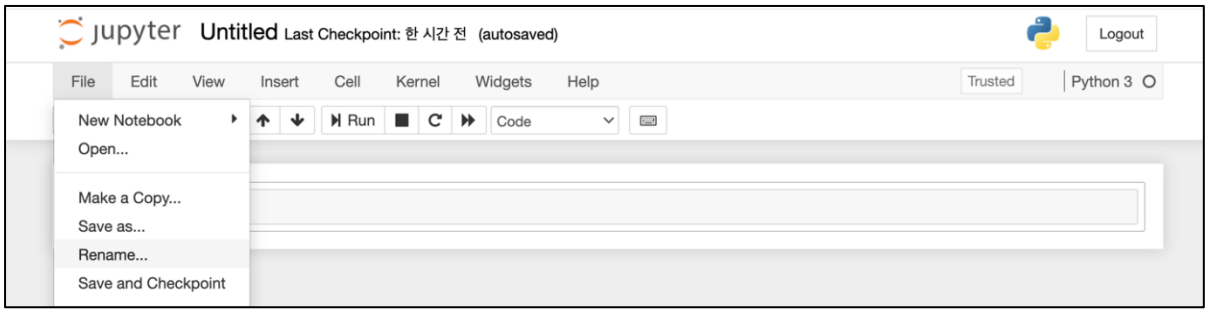
그다음 화면 상단 오른쪽의 <New> 버튼을 클릭한 후 Python3를 선택합니다.

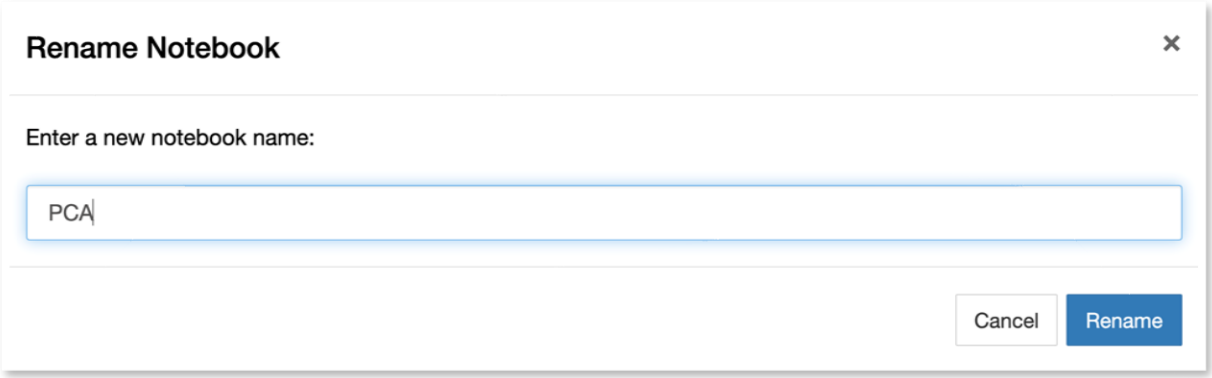


그러면 파이썬을 연습할 수 있는 화면이 열립니다.



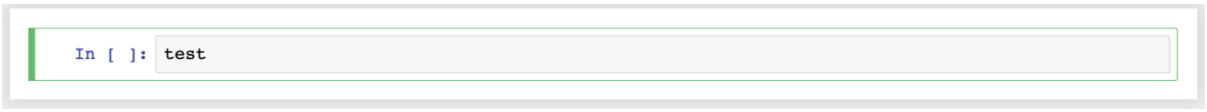
상단 메뉴 바에서 [File] → [Rename] 메뉴를 클릭한 후, PCA라고 입력합니다. 그리고 나타나는 팝업창의 오른쪽 하단에 있는 <Rename> 버튼을 클릭합니다.



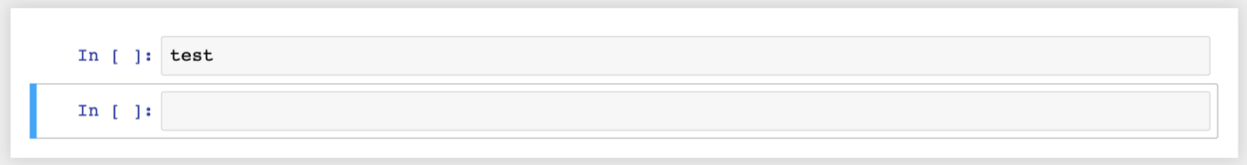


그러면 노트북의 파일명이 PCA로 바뀝니다. 노트북 파일의 확장자는 ipynb로, 구글 코랩 노트북의 확장자도 이와 같습니다.

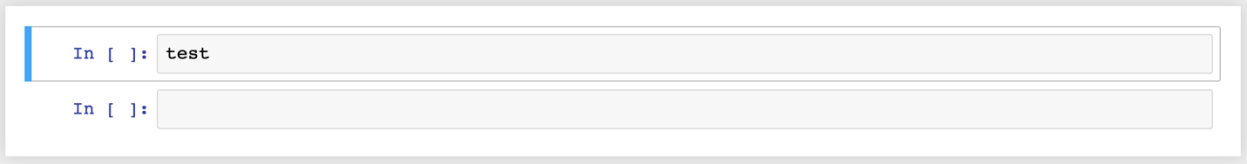
노트북 화면에는 코드를 입력할 수 있는 코드 셀이 있습니다. 첫 번째 줄의 입력 칸을 클릭하면 커서가 깜박이고, 코드 셀 전체가 초록색으로 표시됩니다. 이는 코드가 입력이 가능한 상태라는 것을 의미합니다. 여기에 test라는 단어를 입력해봅시다.



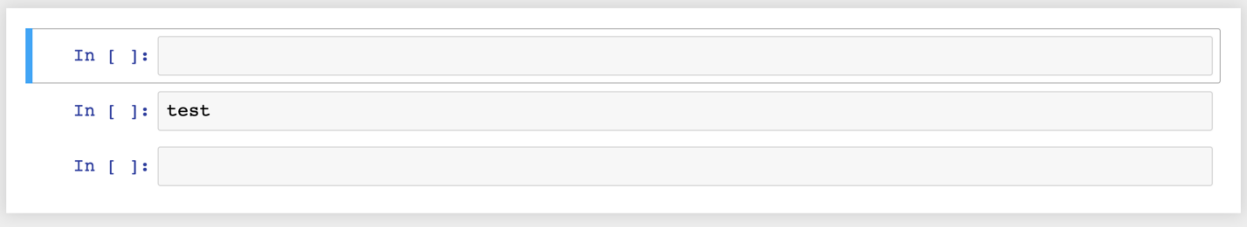
그리고 마우스를 텍스트 입력 칸의 왼쪽으로 옮겨서 In [ ]이 입력된 흰색 부분을 클릭하면 초록색이었던 코드 셀이 파란색으로 바뀝니다. 이때 키보드에서 <b>를 누르면 코드 셀 아래에 새로운 코드 셀이 추가로 생성됩니다.



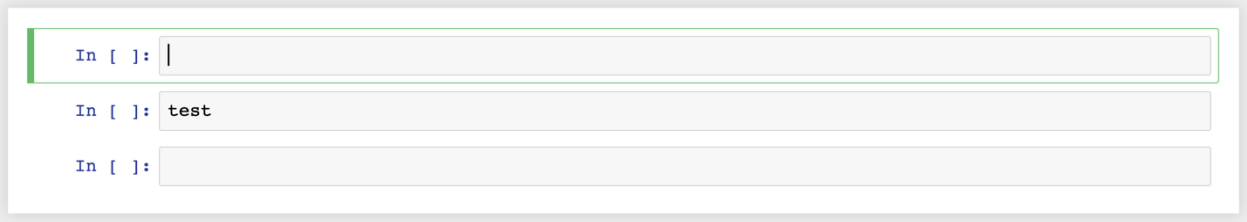
이번에는 test를 입력한 코드 셀의 위쪽에 새로운 코드 셀을 생성해보겠습니다. 다시 test를 입력한 코드 셀의 흰색 부분을 마우스로 클릭합니다.



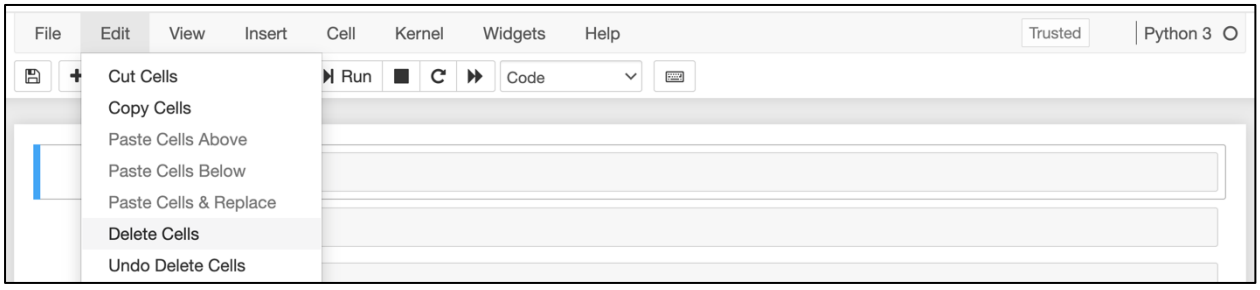
그리고 키보드에서 영문자 <a>를 누릅니다. 반드시 코드 셀이 파란색일 때 영문자를 눌러야 합니다. 그러면 test 위에 새로운 코드 셀이 생깁니다.



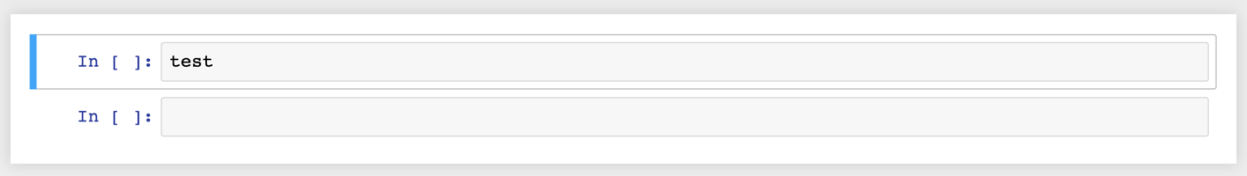
새로 생긴 맨 윗줄에 코드를 입력하려면 회색 부분을 클릭해야 합니다. 그러면 다시 초록색으로 바뀌면서 입력 모드로 바뀌고, 원하는 코드를 입력할 수 있습니다.



코드 셀 자체를 지우고 싶으면 지우려는 코드 셀을 클릭하고 [Edit] → [Delete Cells]를 누르면 됩니다.



이제 첫 번째 줄이 지워지고 나머지 두 개의 코드 셀만 남습니다.



test라고 입력한 코드 셀도 마저 지우고, 다음과 같은 코드를 입력하여 원본 데이터 파일인 healthcare-dataset-stroke-data.csv를 불러오겠습니다. 코드를 실행하려면 Shift 키와 Enter 키를 동시에 누르면 됩니다.

```
import pandas as pd   # pandas 라이브러리를 불러오고 pd 라고 명명
import numpy as np    # numpy 라이브러리를 불러오고 np라고 명명
df = pd.read_csv('healthcare-dataset-stroke-data.csv')
                  # csv 데이터 파일을 읽어와 df에 저장
df.shape              # 데이터프레임 df의 행과 열 개수 알아보기
```

```
(5110, 12)
```

원본 데이터 파일을 파이썬 판다스 라이브러리의 자료 저장 형식인 데이터프레임 df에 저장했습니다. 자료의 행과 열(컬럼) 개수는 각기 5,110줄과 12개입니다. 이 데이터의 처음 3줄을 표시해보겠습니다.

```
df.head(3)    # 데이터프레임 df 의 처음 3 줄까지의 자료를 표시
```

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	9046	Male	67.0	0	1	Yes	Private	Urban	228.69	36.6	formerly smoked	1
1	51676	Female	61.0	0	0	Yes	Self-employed	Rural	202.21	NaN	never smoked	1
2	31112	Male	80.0	0	1	Yes	Private	Rural	105.92	32.5	never smoked	1

열의 맨 위에는 변수명이 적혀 있습니다. id부터 stroke까지 12개의 변수가 있습니다.

머신러닝 팁!
주성분 분석(PCA)은 군집 분석과 더불어 대표적인 비지도 학습 기법 중 하나입니다. 비지도 학습은 출력 변수의 값(레이블)을 고려하지 않고, 입력 변수의 정보만 가지고 학습합니다. 때문에 주성분 분석 및 군집 분석에서는 출력 변수를 사용하지 않습니다.
아울러 주성분 분석은 입력 변수 중에서 범주형 변수가 아닌 구간 변수(Interval variable)에만 사용합니다. 참고로 구간 변수는 숫자로 이루어져 있고, 한 지점에서 다음 지점까지 측정 단위 간격이 일정한 변수입니다. 일반적으로는 실수나 정수 등으로 측정되는 변수라고 생각해도 무방합니다. 예를 들면 온도계나 속도계는 온도와 속도의 측정 단위가 연속적인 숫자이기 때문에 구간 변수에 속합니다.

머신러닝 팁에서 소개한 대로 주성분 분석은 입력 변수 중에서 구간 변수만 다룹니다. 따라서 위에서 불러온 데이터세트의 입력 변수 중에 구간 변수만 뽑아서 분석하겠습니다. 이 데이터세트에서 구간 변수는 나이(age), 평균 혈당치(avg_glucose_level), 그리고 체질량 지수(bmi)로 세 가지입니다.

변수명	데이터 정의	사용 구분	측정 수준
age	나이	Input	Interval
avg_glucose_level	평균 혈당치	Input	Interval
bmi	체질량 지수	Input	Interval

12개의 변수 중에서 이들 세 가지 변수만 모아 별도의 데이터프레임 df1을 생성합니다.

```
# 원본 데이터프레임 df 에서 3 개의 연속 변수만 별도로 데이터프레임 df1 에 저장
cols = ['age', 'avg_glucose_level', 'bmi']
df1 = df[cols]
df1.shape
```

```
(5110, 3)
```

데이터프레임 df1을 확인해보면 세 가지의 구간 변수만 저장되었습니다.

```
df1.head(3)
```

	age	avg_glucose_level	bmi
0	67.0	228.69	36.6
1	61.0	202.21	NaN
2	80.0	105.92	32.5

이들 구간 변수에 대한 기초 통계량을 확인합니다.

```
df1.describe()    # 데이터프레임에 포함된 변수들의 기초 통계량 확인
```

	age	avg_glucose_level	bmi
count	5110.000000	5110.000000	4909.000000
mean	43.226614	106.147677	28.893237
std	22.612647	45.283560	7.854067
min	0.080000	55.120000	10.300000
25%	25.000000	77.245000	23.500000
50%	45.000000	91.885000	28.100000
75%	61.000000	114.090000	33.100000
max	82.000000	271.740000	97.600000

결과 화면의 count 행을 살펴보면 나머지 두 변수는 데이터가 5,110개 있는데, 체질량 지수(bmi)만 4,909개가 있음을 알 수 있습니다. 이는 체질량 지수(bmi)가 결측값이 있기 때문입니다. 주성분 분석에서 결측값이 있으면 분석을 할 수 없으므로 결측값을 제거합니다.

```
df2 = df1.dropna() # 데이터프레임 df1 의 결측값이 있는 행을 제거한 후 df2 에 저장
df2.describe()     # 데이터프레임 df2 의 기초 통계량 표시
```

	age	avg_glucose_level	bmi
count	4909.000000	4909.000000	4909.000000
mean	42.865374	105.305150	28.893237
std	22.555115	44.424341	7.854067
min	0.080000	55.120000	10.300000
25%	25.000000	77.070000	23.500000
50%	44.000000	91.680000	28.100000
75%	60.000000	113.570000	33.100000
max	82.000000	271.740000	97.600000

체질량 지수(bmi)의 결측값이 있는 행들을 데이터프레임에서 제거했더니 데이터 행 수(count)가 4,909개로 통일되었습니다.

머신러닝 팁!
주성분 분석을 하기 전에는 데이터 스케일(측정 단위)에 대한 표준화(Standardization)가 필요합니다. 이는 주성분 분석이 다루는 변수들의 측정 스케일이 비슷해야 주성분 분석의 결과가 좋아지기 때문입니다. 원본 데이터 X에 대한 표준화는 다음과 같은 변환을 통해 이루어집니다.

$$Z = (X - U) / S$$

여기서 Z는 표준화된 데이터, U는 데이터의 평균, S는 데이터의 표준편차를 의미합니다.

주성분 분석을 하기 전에 데이터 스케일을 표준화합니다.

```
from sklearn.preprocessing import StandardScaler # StandardScaler 를 불러오기
X_scaled = StandardScaler().fit_transform(df2)   # df2 데이터 표준화
```

위의 두 번째 명령문에서 입력은 데이터프레임 형식인 df2로 했지만, 출력된 결과물인 X_scaled는 넘파이 배열 형식(numpy.ndarray)을 띕니다.

```
type(X_scaled)
```

```
numpy.ndarray
```

넘파이 배열 형식은 데이터프레임 df2에 포함되어 있었던 변수명을 제거하고, 데이터값만 배열 형태로 저장합니다. 이렇게 표준화된 입력 데이터를 가지고 주성분 분석을 실시합니다.

```
from sklearn.decomposition import PCA   # PCA 를 사이킷런 라이브러리에서 불러오기
pca = PCA(n_components=2)               # 주성분을 2 개로 지정
X_pca = pca.fit_transform(X_scaled)     # X_scaled 주성분 분석을 실시 후
                                        # 그 결과를 X_pca 에 저장
```

주성분 분석 결과가 저장된 X_pca를 눈으로 확인하기 위해 다음 코드를 실행합니다.

```
# X_pca 가 numpy array 형태인데, 이를 데이터프레임으로 변환
```

```
# 변수명은 pc_1, pc_2 로 지정
df_pca = pd.DataFrame(data = X_pca, columns = ['pc_1', 'pc_2'])
df_pca.shape
```

```
(4909, 2)
```

앞에서 X_scaled가 넘파이 배열 형식이라고 했습니다. X_scaled를 입력하여 주성분 분석 과정을 거친 결과물인 X_pca도 넘파이 배열 형식입니다. 데이터프레임은 변수명을 포함하지만, 넘파이 배열에서는 변수명이 사라집니다. 따라서 위의 코드를 보면 pd.DataFrame 기능을 통해 넘파이 배열 X_pca를 데이터프레임 df_pca로 바꿔줄 때 변수명 pc_1과 pc_2를 추가합니다. pd.DataFrame 구문의 괄호 안에 있는 columns = ['pc_1', 'pc_2']가 그 역할을 합니다.

그리고 shape 명령어를 통해서 확인한 df_pca의 열(변수) 개수는 2개입니다. 이제 df_pca 데이터를 5줄까지만 나열해보겠습니다.

```
df_pca.head(5)
```

	pc_1	pc_2
0	2.650888	-1.634549
1	1.315375	0.552382
2	1.332632	-0.846288
3	1.417073	-1.289867
4	1.987158	-1.184098

원래 df2에 표준화된 입력 변수는 나이(age), 평균 혈당치(avg_glucose_level), 체질량 지수(bmi), 세 개였습니다. 그런데 주성분 분석 작업을 거치고 난 결과물인 df_pca에는 주성분1(pc_1), 주성분2(pc_2), 두 개의 변수만 들어있습니다. 변수 3개로 표현된 정보량을 변수 2개로 나타낼 수 있도록 차원 축소를 한 것입니다. 차원 축소를 하면 소실되는 정보량은 다음과 같이 확인합니다.

```
print(pca.explained_variance_ratio_)    # 각 주성분이 설명하는 분산 비율 표기
```

```
[0.5008308  0.27977091]
```

위 코드는 주성분1과 주성분2가 설명하는 원본 데이터의 분산 비율을 표기합니다. 각기 약 50%와 28%의 분산을 설명합니다. 대략적으로 차원 축소를 통해서 나머지 22%의 분산 설명력을 상실했습니다. 분산 설명력 소실의 대가로 얻은 것은 줄어든 입력 변수의 개수입니다.

머신러닝 팁!
여기서는 3개의 입력 변수를 2개의 주성분으로 줄였기 때문에 입력 변수를 줄인 효과가 크게 보이지 않지만, 만약 20개의 입력 변수를 4~5개의 주성분으로 줄이면 차원 축소로 인해 고려해야 하는 입력 변수를 대폭 줄이는 효과를 볼 수 있습니다.

이상으로 주성분 분석 코드 실습을 마치겠습니다. 여기서 쓰인 표준화된 입력 변수는 K-평균 군집에서 다시 쓰입니다. 이때 다시 쓰기 위하여 표준화된 입력 변수를 담은 X_scaled를 df_scaled

라는 데이터프레임으로 저장합니다.

```
# X_scaled 가 numpy array 형태인데, 이를 데이터프레임으로 변환
# 변수명은 cols 로 지정
# cols 에는 기존 3 개 변수명이 들어 있음
df_scaled = pd.DataFrame(data = X_scaled, columns = cols)
df_scaled.shape
```

```
(4909, 3)
```

```
df_scaled.head(3)
```

	age	avg_glucose_level	bmi
0	1.070138	2.777698	0.981345
1	1.646563	0.013842	0.459269
2	0.272012	1.484132	0.701207

이제 이 df_scaled라는 데이터프레임을 csv 파일 형태로 저장합니다. 파일명은 stroke-scaled.csv로 저장합니다.

```
# df_scaled 를 csv 형태의 파일로 저장
df_scaled.to_csv('stroke-scaled.csv', index=False)
```

A.9 군집 분석

대표적인 비지도 학습 분류 작업인 군집 분석의 개념 및 K-평균 군집을 소개하고 예제를 살펴봅니다.

A.9.1 군집 분석 개요

다수의 입력 변숫값이 다차원 공간에 흩뿌려져 있으면 이들을 직관적으로 그룹화하기는 쉽지 않습니다. 입력 변수가 1~2개를 넘어 3개 이상이 되면 시각적으로도 입력 데이터의 분포를 파악하기 쉽지 않습니다. 그러나 이러한 입력 데이터를 비슷한 특성(그래프로 말하자면 비슷한 분포)끼리 모아서 그룹화하면 방대한 자료를 몇 개의 자료군으로 파악할 수 있게 되어 분석이 용이해집니다.

예를 들어 우리나라 프로 야구 KBO 리그의 모든 타자 데이터를 가지고 분석을 한다고 해보겠습니다. 수백 명이 넘는 타자의 타율, 타점, 출루율 등 야구 관련 통계 지표는 최근에 만들어진 지표까지 합치면 수십 개가 넘습니다. 이 수십 개의 지표를 보고 직관만으로는 선수의 유형(그룹)을 찾기 어렵습니다. 만약 선수들을 특정 몇 개의 그룹으로 분류할 수 있다면 분석이 편리해질 것입

니다. 이렇듯 머신러닝과 딥러닝 분석에서도 입력 데이터를 그룹별로 분류할 필요가 있는데, 이를 비지도 학습 분류라고 합니다.

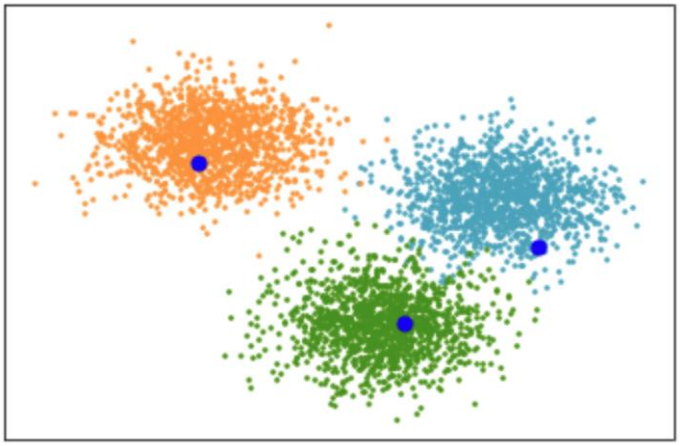
비지도 학습 분류는 출력 변수(타깃 변수)를 고려하지 않고, 입력 변수의 유사성(similarity)에 의존하여 학습 데이터세트를 그룹별로 나눕니다. 그룹화 결과물은 군집(cluster) 혹은 세그먼트(segment)라고 부르며, 이러한 그룹화 과정을 군집 분석이라고 합니다.

마케팅 분야에서는 대표적으로 고객을 공통의 특성을 중심으로 분류한 군집 분석이 유용합니다. 분류된 고객 세그먼트별로 차별화된 마케팅을 할 수 있기 때문입니다. 다만, 입력 데이터가 출력 변수와 결합되어 있지 않은 비지도 학습에서는 입력 데이터를 그룹화하는 기준이 필요합니다. 여기서는 대표적인 그룹화 기준으로 K-평균 알고리즘을 사용한 K-평균 군집을 소개하겠습니다.

A.9.2 K-평균 군집

K-평균 군집(K-Means Clustering) 알고리즘은 직관적으로 이해하기 쉽고 대규모 데이터세트에서도 잘 작동하기 때문에 가장 많이 쓰이는 군집 기법 중 하나입니다. K-평균 군집은 데이터간의 거리를 측정하는 과정을 포함하기 때문에 입력 데이터가 구간 변수(Interval variable)이어야 합니다. 대부분의 머신러닝 데이터는 구간 변수와 범주형 변수가 혼재돼 있습니다. 비즈니스 분석 자료는 특히 그렇습니다. 때문에 K-평균 군집 알고리즘은 구간 변수에만 적용해야 한다는 점에 주의하기 바랍니다.

K-평균 군집 알고리즘은 다음과 같습니다. 그림에서 주황색, 하늘색, 초록색 점들은 입력 데이터를 나타냅니다.



[그림 A-10] K-평균 군집 입력 데이터 분포

K-평균 군집 알고리즘의 동작 과정은 다음과 같습니다.

Step 1. 입력 데이터에서 K 개의 군집 중심점을 임의로 선택합니다. 위의 그림에서 하늘색 점이 임으로 주어진 3 개의 초기 중심점입니다. 이때 K 값을 지정해야 합니다(예: K=3).

Step 2. 모든 개별 입력 데이터와 K 개 중심점과의 거리(유클리드 거리)를 계산합니다. 그리고 계산 결과에 의해 개별 입력 데이터를 가장 가까운 중심점으로 소속시킵니다. 이 과정을 통해 모든 데이터의 소속이 K 개의 군집으로 정해집니다.

Step 3. 군집별로 소속 데이터의 평균을 구해 이를 새로운 중심점으로 설정하여 K 개의 새로운 군집 중심점을 정합니다.

Step 4. 재조정된 중심점의 위치를 가지고 모든 데이터에 대해 **Step 2** 작업부터 다시 실시합니다.

Step 5. 위 과정을 반복하다가 중심점의 위치가 수렴하면 과정을 끝냅니다. 이 결과로 최종적인 K 개의 군집이 얻어집니다. 즉, 위의 그림에서 설정된 최초의 중심점과 최종 결과의 중심점은 위치가 달라질 수 있습니다.

머신러닝 팁!
앞에서 설명한 주성분 분석(PCA)처럼 군집 분석 또한 비지도 학습으로서 분석 대상으로 입력 변수 중 구간 변수만 다룹니다. 군집 분석은 입력 데이터를 군집으로 그룹화하기 위해 거리 측정이 필요하기 때문에 입력 데이터가 실수나 정수 등의 숫자형이어야 합니다.
아울러 주성분 분석처럼 군집 분석도 데이터 스케일에 대한 표준화가 필요합니다. 이는 군집 분석이 거리 측정을 통해 군집을 형성할 때 변수마다 측정 스케일이 크게 차이 나면 특정 변수가 거리 측정 시 미치는 영향력이 커지기 때문입니다.

위에서 언급한 대로 군집 분석도 구간 변수 데이터를 표준화해야 합니다. 우리는 앞에서 저장한 구간 변수 표준화 데이터세트 stroke-scaled.csv를 불러와서 K-평균 군집 분석을 실시하겠습니다.

```
import pandas as pd
import numpy as np
df = pd.read_csv('stroke-scaled.csv')
df.shape
```

```
(4909, 3)
```

```
df.head(3)
```

	age	avg_glucose_level	bmi
0	1.070138	2.777698	0.981345
1	1.646563	0.013842	0.459269
2	0.272012	1.484132	0.701207

불러온 데이터세트는 구간 변수인 나이(age), 평균 혈당치(avg_glucose_level), 그리고 체질량 지수(bmi), 세 가지 변수의 표준화된 데이터를 담고 있습니다. 이제 K-평균 군집을 실시합니다.

```
# 사이킷런 라이브러리에서 KMeans 불러오기
from sklearn.cluster import KMeans
# Kmeans 기능에서 군집 개수를 3 개로 설정
km = KMeans(n_clusters=3, random_state=0)
# 데이터프레임 df 를 군집 분석한 결과로서 군집 결괏값(레이블)을 변수명 cluster 에 저장
```

```
df['cluster'] = km.fit_predict(df)
df
```

	age	avg_glucose_level	bmi	cluster
0	1.070138	2.777698	0.981345	1
1	1.646563	0.013842	0.459269	0
2	0.272012	1.484132	0.701207	1
3	1.602222	1.549193	-0.623083	1
4	1.690903	1.821368	0.013595	1
...	...	...	...	...

결과 화면을 보면 cluster라는 변수가 추가로 생성되었고 0, 1, 2 이렇게 세 개의 레이블(값)을 갖는 것을 알 수 있습니다. 이는 코드에서 n_clusters=3 문구를 통해 군집 개수를 3으로 미리 설정했기 때문입니다.

머신러닝 팁!
머신러닝 모델이나 알고리즘은 실행 과정에서 랜덤한 설정이나 절차에 의존하는 경우가 있습니다. 이럴 경우, 동일한 입력 데이터로 동일한 코드를 실행했음에도 불구하고 결괏값이 미묘하게 달라질 수 있습니다. 이는 지극히 정상적인 결과입니다.
책과 부록에서는 혼동을 방지하기 위해 머신러닝을 실행할 때 random_state를 설정하는 옵션을 사용합니다. random_state값은 정수라면 어떤 값이라도 상관 없습니다. 여기서는 random_state=0으로 설정했습니다.
여기서 잠깐! 앞의 주성분 분석(PCA) 코드에서는 random_state값을 설정하지 않았습니다. 왜일까요? 주성분 분석은 실행 과정에서 랜덤한 설정이나 절차에 의존하지 않기에 아예 random_state를 설정하는 옵션이 없습니다.

이렇게 만들어진 3개의 군집 중심점의 좌표는 다음과 같습니다.

```
km.cluster_centers_
```

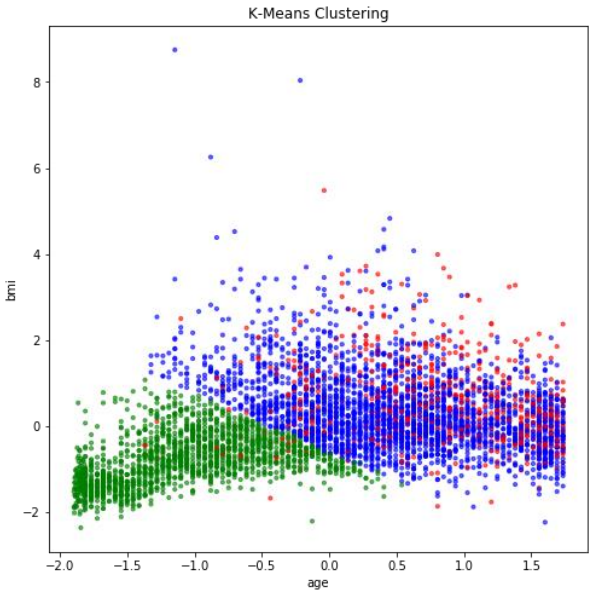
```
array([[ 0.49163911, -0.36676422,  0.36229297],
       [ 0.78171714,  2.23498965,  0.57702459],
       [-1.07091421, -0.28226236, -0.78953804]])
```

중심점 좌표는 넘파이 배열 형태로 나타납니다. 결과 화면에서 세 개의 숫자 조합은 각기 나이(age), 평균 혈당치(avg_glucose_level), 체질량 지수(bmi)의 값을 나타냅니다. 이는 첫 번째 군집(cluster 레이블 0)의 중심점 좌표입니다. 마찬가지로 두 번째 숫자의 조합은 두 번째 군집(cluster 레이블 1)의 중심점 좌표이고, 세 번째 숫자의 조합은 세 번째 군집(cluster 레이블 2)의 중심점 좌표입니다.

이제 총 4,909개의 입력 데이터를 3개의 군집으로 그룹화하여 색으로 구분한 그래프를 그려보겠습니다.

```
# 클러스터 레이블별 colors 를 다음과 같이 저장
# cluster0: 파랑, cluster1: 빨강, cluster2: 초록
colors = ['b', 'r', 'g']
```

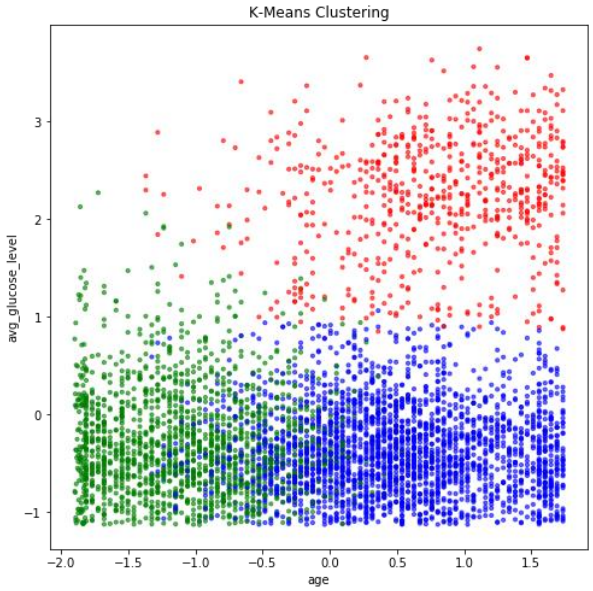
```
df['c'] = df['cluster'].map({0:colors[0], 1:colors[1], 2:colors[2]})

import matplotlib.pyplot as plt  # matplotlib 라이브러리를 호출
%matplotlib inline               # 주피터 노트북에서 그래프를 출력

# 그래프 그리기
fig, ax = plt.subplots(1, figsize=(8,8))
scatter = ax.scatter(df['age'],df['bmi'],c=df['c'], alpha = 0.6, s=10)


# 제목과 x 축, y 축 이름 설정
ax.set_title('K-Means Clustering')
ax.set_xlabel('age')
ax.set_ylabel('bmi')

# 파랑 cluster0: 나이가 아주 어리지는 않은 그룹
# 빨강 cluster1: 나이가 아주 어리지는 않은 그룹
# 초록 cluster2: 나이가 어리고 체질량이 작은 그룹
```



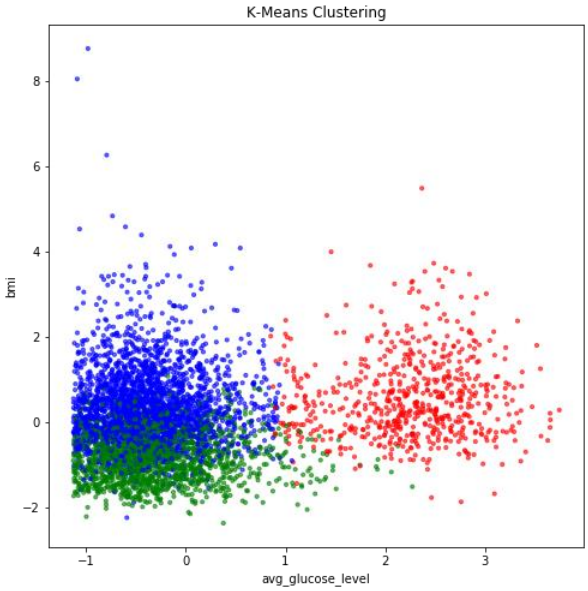
위의 그래프에서 X축과 Y축은 나이(age)와 체질량 지수(bmi)입니다. 초록색 군집과 파란색 군집은 잘 구분되는데, 빨간색 군집은 다른 군집과 명확히 구분되지 않고 섞여 있는 것처럼 보입니다. 따라서 X축과 Y축을 바꿔보겠습니다.

```
# 그래프 그리기
fig, ax = plt.subplots(1, figsize=(8,8))
scatter = ax.scatter(df['age'],df['avg_glucose_level'],c=df['c'],alpha = 0.6,
                     s=10)

# 제목과 x 축, y 축 이름 설정
```

```
Ax.set_title('K-Means Clustering')
ax.set_xlabel('age')
ax.set_ylabel('avg_glucose_level')

# 파랑 cluster0: 혈당량이 낮은 성인
# 빨강 cluster1: 혈당량이 높은 성인
# 초록 cluster2: 어린이 및 청소년
```

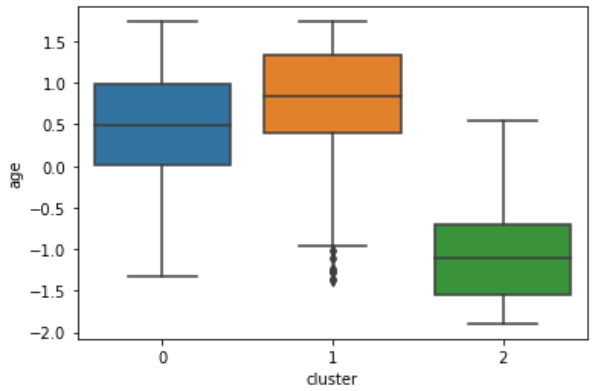



```
# 그래프 그리기
Fig, ax = plt.subplots(1, figsize=(8,8))
scatter = ax.scatter(df['avg_glucose_level'], df['bmi'],c=df['c'], alpha = 0.6,
                     s=10)

# 제목과 x 축, y 축 이름 설정
ax.set_title('K-Means Clustering')
ax.set_xlabel('avg_glucose_level')
ax.set_ylabel('bmi')

# 파랑 cluster0: 혈당량이 낮고 체질량은 높은 그룹
# 빨강 cluster1: 혈당량이 높은 그룹
# 초록 cluster2: 혈당량과 체질량이 모두 낮은 그룹
```



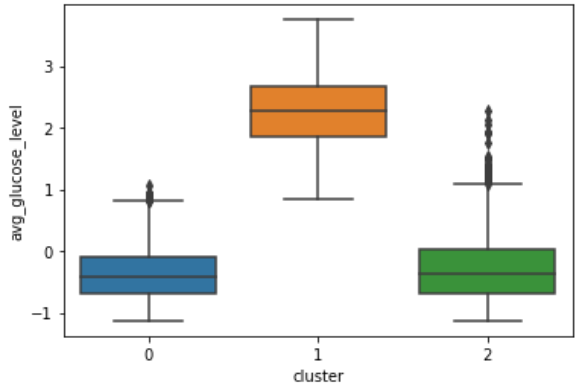
새로 출력한 두 개의 그래프는 각 군집이 명백하게 구분됩니다. 아울러 군집별로 입력 변수의 분포를 상자그림(box plot)으로도 비교할 수 있습니다. 참고로 위 그래프에서 빨간색 군집은 상자그림에서 주황색 군집이고, 파란색과 초록색은 약간 채도가 다르게 나타납니다.

```
import seaborn as sns                    # seaborn 라이브러리 불러오기
sns.boxplot(df['cluster'],df['age']); # cluster 별로 나이(age)의 상자그림 그리기
```



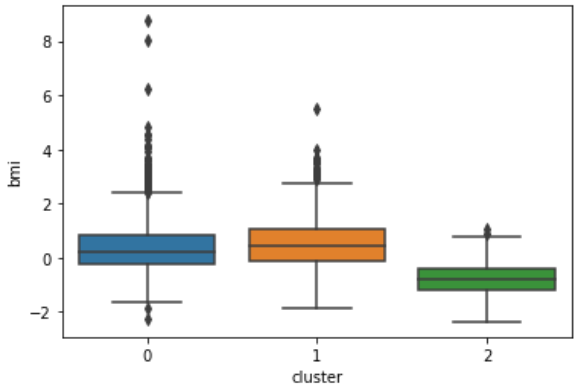
위 상자그림은 군집별 나이(age)의 분포를 나타냅니다. 예를 들어 파란색 박스의 가운데 선은 파란색 군집의 나이 중위수(median)를 나타냅니다. 파란색 박스에서 나이 중위수를 나타내는 가로선은 0.5를 가리키는데, 데이터를 표준화하기 전 나이를 기준으로 환산하면 나이가 많은 축에 속합니다. 주황색 군집의 나이 중위수는 이보다 더 많고, 초록색 군집의 나이 중위수는 매우 어린 것을 알 수 있습니다.

```
# cluster 별로 평균 혈당량의 상자그림 그리기
sns.boxplot(df['cluster'],df['avg_glucose_level']);
```



군집별 평균 혈당량(avg_glucose_level) 분포를 보면 주황색 군집의 위치가 높고, 나머지 둘은 낮습니다.

```
# cluster 별로 체질량 지수(bmi)의 상자그림 그리기
sns.boxplot(df['cluster'],df['bmi']);
```



군집별 체질량 지수(bmi) 분포를 보면 파란색 군집과 주황색 군집이 비슷하고, 초록색 군집이 나머지 둘보다 낮은 위치에 있습니다.