Chp.6 통계적 머신러닝

▼ 통계학이 레시피라면, 머신러닝은 조리 도구이다.

레시피(통계학)는 요리의 구조에 집중, 조리 도구(머신러닝)은 요리를 빠르고 효율적으로 하기 위해 발전한다.

▼ 통계학의 집중 분야

확률론 & 모델 구조
>> 배깅과 랜덤포레스트는 통계학에서 시작. 부스팅에 관심이 많다.

▼ 머신 러닝의 집중 분야

신속한 데이터 처리를 위한 알고리즘 개발
>> 부스팅은 양쪽에서 개발했지만, 요즘은 머신러닝에서 더 유심히 개발 중

최근 통계학 분야는 회귀와 분류를 자동화하는 통계적 머신러닝인 정답을 사용하는 학습, 즉 지도학습에 집중해왔다. 많은 모델을 사용하여 결과를 도출하는 앙상블 모델이 가장 많이 사용되어 왔다.

K-Nearest Neighbor

▼ K근접 알고리즘은 비슷한 레코드들이 어떻게 분류 되는지에 따라 예측 및 분류한다.

▼ Terms

▼ 이웃(Neighbors)

예측 변수에서 값들이 유사한 레코드

▼ 거리 지표

각 레코드 사이가 얼마나 멀리 떨어져 있는가를 나타내는 단일 값

▼ 표준화

평균을 뺀 후에 표준편차로 나누는 일(유의어: 정규화)

▼ z-score

표준화를 통해 얻은 값

▼ k

최근접 이웃을 계산하는데 사용된 이웃의 개수

▼ ex) 대출 연체 예측 (Chp.5에서도 사용됨)

R에서는 FNN함수가 존재하고, FNN(빅데이터 + 유연성) 모델도 사용 가능하다.

파이썬의 sklearn에서 제공하는 KNeighborClassifier

features = [’payment_inc_ratio’, ‘dti’]

Target = [’Paid-off’, ‘Default’]

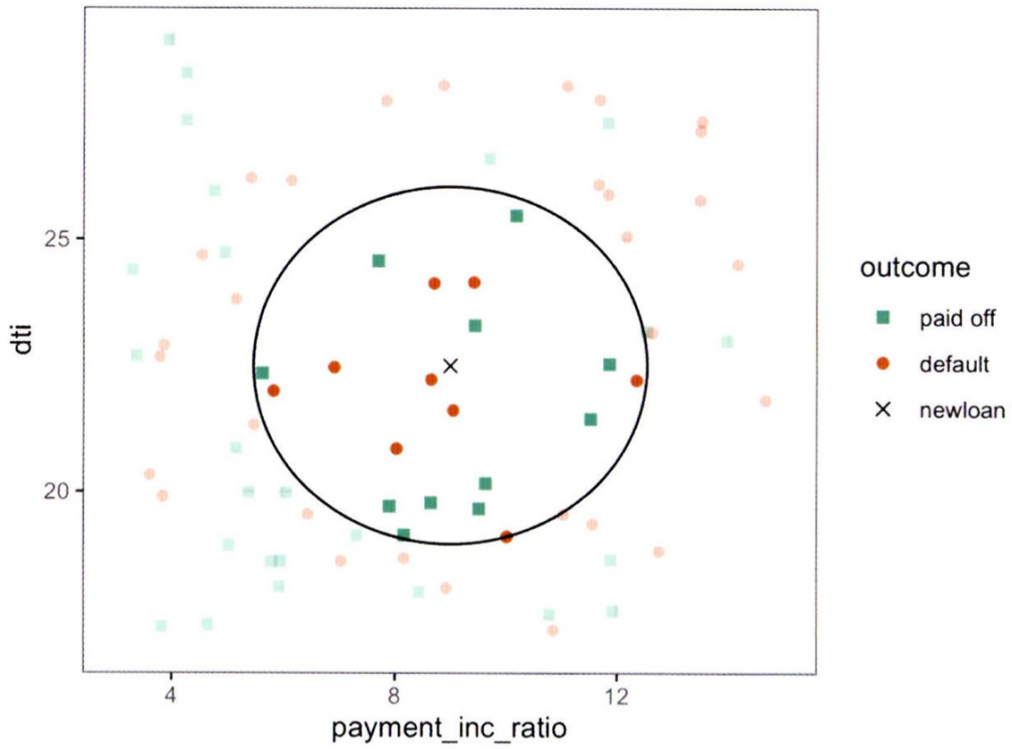


그림 6-2 두 변수(총 소득에 대한 부채 비율과 상환 비율)를 이용한 KNN의 대출 연체 예측

클래스 불균형이라면 생기는 문제는? k값을 다르게 설정하는 지에 따라 예측 결과는 어떻게 달라지는가? 확률 반환도 가능

▼ 거리 지표

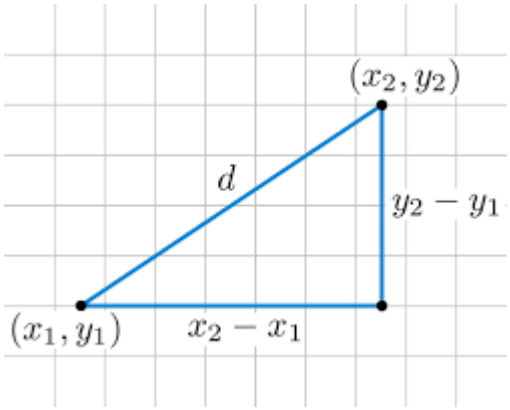
유사성과 근접성을 결정하는 지표

▼ 두 데이터가 얼마나 멀리 떨어져 있는가를 보기 위해 유클리드 거리와 맨해튼 거리를 사용한다.

▼ 유클리드 거리

▼ 두 벡터 사이의 차이를 제곱합한 후 제곱근을 씌워준 값이다.

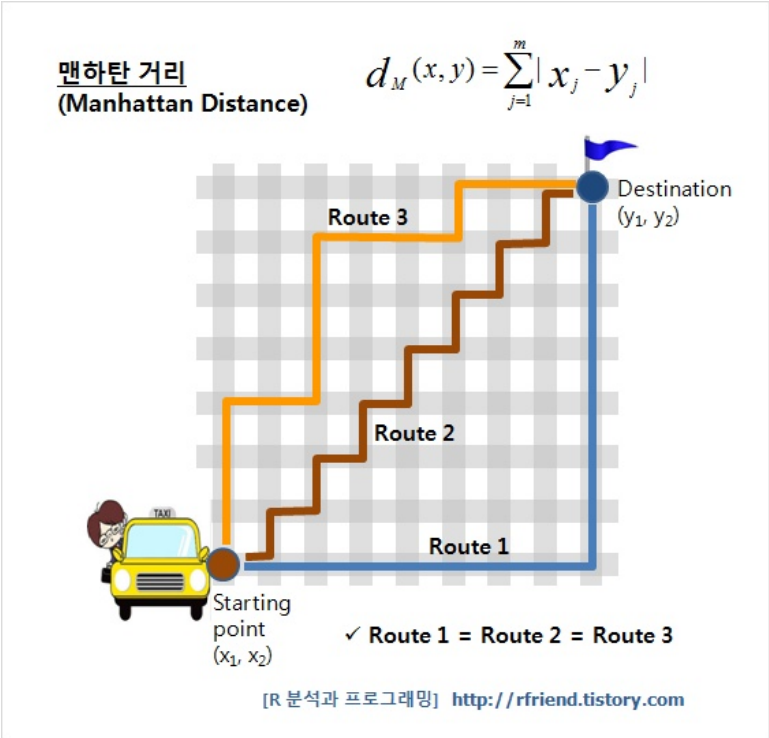
우리가 알고있는 피타고라스 정의와 같은 식이며 두 점 사이의 직선을 표현한다.



▼ k값* 데이터 개수만큼 연산량이 커지기 때문에
벡터의 크기와 k번 반복하는 k값이 커지면 계산이 복잡해진다.

▼ 맨해튼 거리

직선으로가 아닌 한 번에 한 축으로만 움직일 수 있다는 원리 안에서의 점과 점 사이의 거리이다.



▼ 정규화, 표준화, z-점수

두 벡터 사이의 거리를 측정할 때 정규화를 생각하고 고려해보아야 한다.

수치 스케일이 다르면 더 크게/작게 영향을 끼치는 변수를 스케릴링한다 (6.1.4에서 다룸)

금액: 100,000, 비율: 0.2

❖ **최소-최대 정규화**

$$X_{new} = \frac{X - \min(X)}{\max(X) - \min(X)}$$

❖ **Z점수 표준화**

$$X_{new} = \frac{X - \mu}{\sigma} = \frac{X - \text{Mean}(X)}{\text{StdDev}(X)}$$

❖ 더미 코딩

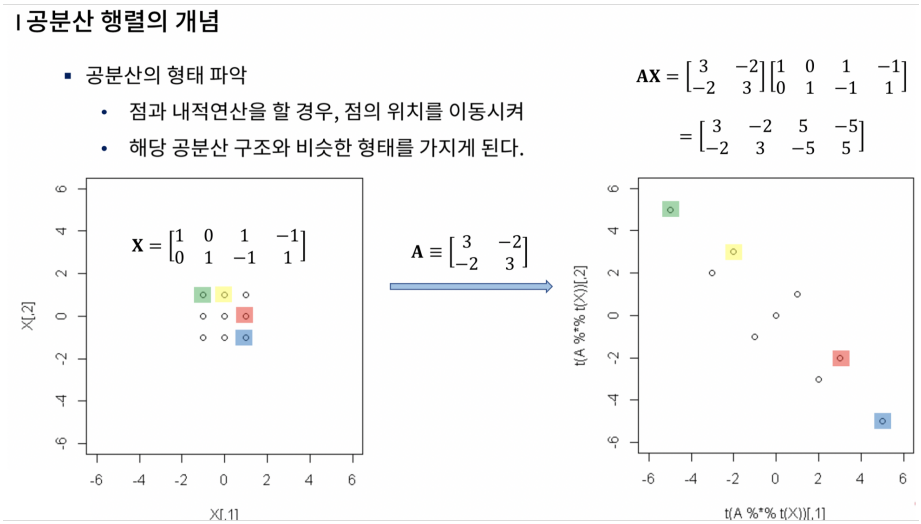
▼ 마할라노비스 거리

- 평균과의 거리가 표준편차의 몇 배인지를 나타내는 값이다.
- 변화가 거의 없는 사건에서 평균을 벗어난 값에서는 마할라노비스 거리 값이 크다.
- 변화가 들쑥날쑥한 사건에서 평균을 벗어난 값에서는 마할라노비스 거리 값이 작다.
- 표준정규분포로 바꾼 후에 z 값을 구하는 것이 바로 Mahalanobis distance를 구하는 과정
- 두 변수간의 상관관계와 공분산행렬을 사용한다.

▼ 공분산행렬 (5.2.1 참고)

- range(-∞, ∞)
- 공분산은 두 변수 사이의 관계를 의미하는 지표이다
- 다수의 두 변량 값들 간의 공분산 값을 행렬로 표현한 것이다.

- 변수를 z-score로 변환하기 위해서 표준 편차를 사용했는데, 확장해서 보면
  다변량 분석에서 **<u>표준화 처리를 하기 위해 공분산 행렬을 사용한다</u>**.



▼ 상관관계

- range(-1, 1)
- 유클리드와 맨해튼은 두 변수의 상관관계가 높을 때 원인에 집중하게 되는데,
  주성분 사이의 유클리드 거리를 구하기 때문에 주요한 특성만 사용할 수 있다.

▼ 원핫인코더

- 문자열 형태의 변수는 같은 정보를 담고 있는 **<u>이진 가변수</u>**의 집합으로 변환해야 한다.
- 선형회귀나 로지스틱 회귀에서 다중공선성 관련 문제가 나타날 수 있으니 주의.
  >> KNN + 다른 모델에서는 문제되지 않음

▼ 표준화 (a.k.a 표준화, z-score)

- 얼마나 평균과 파이가 나는지를 보고 싶은데 변수마다 스케일이 다 다르다.

- **평균**을 뺀 후 **표준 편차**로 나누는 것으로 한눈에 알아볼 수 있음
  >> mean 대신 median을 사용할 수도 있다.
  >> 표준 편차 대신에 사분위 범위를 사용할 수도 있음
- KNN, PCA, Clustering에는 필수적인 과정
- 표준화를 통해 skew된 데이터를 정규분포로 만들어주지는 않는다.
- 파이썬: sklearn.preprocessing.StandardScaler

▼ scaler을 적용하기 전에 생각해보아야 할 질문

>> 한 변수가 다른 변수보다 중요할 수도 있으면 어떻게 할까?

▼ K 선택하기

▼ K값의 크기

▼ k=1로 시작을 한 뒤 값을 크게 할 수록 성능이 좋아진다. why?

- k값이 1일때 근처에 하나의 데이터 포인트만 있어도 분류
  >> 노이즈까지 잡기 때문에 과적합 발생
- k값이 커질 수록 결정 함수를 부드럽게(?) 만들어준다. (과적합 방지 느낌)

▼ k값이 너무 크면?

결정함수가 과하게 평탄화 되어 오버스무딩 된다. (과소적합과 동의어)

>> 데이터의 지역 정보를 정확히 분류 하지 못하고 패턴을 잃게 된다.

▼ 분산 편향 트레이드 오프

- 반비례 관계
- 오버피팅 → 분산의 증가했다는 의미
- 오버스무딩 → 편향이 증가했다는 의미

4.2.3에 나오는 교차 타당성 검사 참조
>> Cross-validation 사용을 통해 최적의 값을 찾는다.

- 데이터에 따라 k값이 결정된다.
  - 노이즈가 별로 없으면? → k값이 작게
    ex) *전문용어로 신호대 잡음비(SNR)이 높은 데이터를 말한다.*
    손글씨 데이터, 음성 인식 데이터 → k값 작아도 됨

- 노이즈가 많으면? → k값 크게
  SNR이 낮은 데이터 → ex) *대출 데이터*
- k 값은 **1~20** 사이가 적절. 동률을 피하기 위해 **홀수** 사용 권장

▼ KNN을 사용한 "***자동화 된 feature engineering***"

▼ 복잡한 모델을 사용해도 되는데 왜 KNN을 사용하는가?

▼ 지역적 정보(Local Knowledge)를 추가하는 일종의 feature engineering 방법이다

ex) 2 features 사용 → 최근접(KNN)으로 **클래스의 속할 확률**을 구한다.
>> 더욱 복잡한 모델에서 새로운 피처로 사용한다. 예측 변수를 2번 사용하는 방법.

▼ y를 사용한 도출값을 다른 모델에 넣는다면 다중공선성의 문제가 있지 않은가?

소수의 근접한 레코드들로 부터 얻은 매우 지역적인 정보라서 괜찮다(?)
>> 새로 얻은 정보는 불필요하거나 중복성이 있지 않다고 설명

- 여러 모델링 방법을 합쳐 최종 학습 결과를 내는 앙상블 기법의 일종이라고도 볼 수 있다.

Tree Models

▼ 결정 트리는 의사 결정 트리, 회귀 및 분석 트리, 트리라고도 불린다.

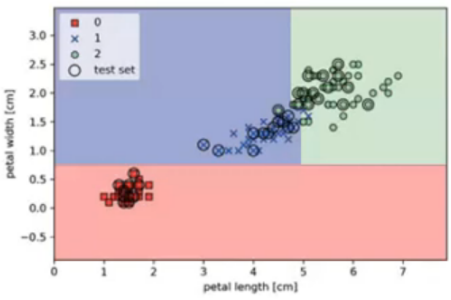
- 랜덤 포레스트 & 부스팅의 기본 모델
- **복잡한 상호관계의 패턴을 파악하는데 용이**
- KNN 및 Naive-Bayes model과 다르게 예측 변수들 사이의 관계로 단순 트리 모델을 표시할 수 있고 해석이 쉽다.

▼ 용어

- 재귀 분할 (Recursive Partitioning)

    - 마지막 분할값이 최대한 비슷하게 보이도록 반복적으로 분할하는 방법
- 분할값 (Split Value)
    - 예측 변수를 이분할 하는 기준값
- 노드 (Node)
    - 분할 규칙의 시각적인 표시
- 잎 (Leaf)
    - if-then-else의 마지막 가지 부분 (끝 부분)
    - 최종적인 분류 규칙을 의미
- 손실 (Loss)
    - 분류하는 과정에서 발생한 오분류의 수. 클수록 분순도가 높아짐
- 불순도 (Impurity)
    - 서로 다른 클래스의 데이터가 얼마나 분류되었는가를 나타냄
      (a.k.a 이질성 ↔ 동질성)
- 가지치기 (Pruning)
    - 학습이 끝난 모델에서 과적합을 방지하기 위해 가지를 하나씩 잘라주는 과정

▼ 재귀 분할 알고리즘 (Recursive Divisive Algorithm)



- 예측 변수를 반복적으로 분할하는 알고리즘
- 같은 클래스의 데이터끼리 구분되도록 한다.
- 특정 값을 이분화하여 하위 분할내에서 클래스의 동질성이 가장 큰 결과값으로 나눔
    - 더는 개선되지 않을 때까지 분할을 반복

▼ 동질성과 불순도 측정

해당 파티션 내에서 오분류 된 레코드의 비율

정확도는 불순도 측정에 좋지 않은 결과를 보이기 때문에 아래 와 같은 지표를 사용

▼ 지니 불순도

$$I(A) = p(1-p)$$

- 한 노드의 모든 샘플이 같은 클래스에 속해있을 때 값이 0이 되고, 순수하다고 말함.
  >>범주들이 섞여 있을 수록 수치가 커짐

▼ 엔트로피

$$I(A) = -p\log_2(p) - (1-p)\log_2(1-p)$$

- 한 노드의 모든 샘플이 같은 클래스에 속해있을 때 값이 0이 된다.

- *지니 인덱스는 구간 [0, 0.5] 안에 값이 있는 반면 엔트로피 구간은 [0, 1]*
- *엔트로피는 로그 를 사용해서 더 복잡하기 때문에, 지니 복잡도 계산이 더 빠름*

▼ 트리 형성 중지 시점

- 트리 모델은 큰 규칙에서 작은 규칙까지 계속 분할해 간다.
- 100% 정확도를 가지면 과적합 된 모델로 완성

▼ 하이퍼 파라미터 설정으로 과적합을 방지할 수 있다.

▼ 분할했는데 규칙의 크기가 너무 작다면?

- min_samples_split
- min_samples_leaf

▼ 불순도를 유의미하게 줄이지 않는다면?

- min_impurity_decrease

▼ 트리 복잡도 제어

아래의 하이퍼 파라미터를 GridSearchCV에 사용해서 최적의 값을 찾게 한다.

- cp_alpha
  >> 0이면 무성한 트리, 커질수록 트리는 작아진다.

- max_depth: 5~30
- min_samples_split: 20~100

▼ DecisionTreeRegressor

▼ 불순도 측정 방법

하위 분할 영역에서 평균으로부터의 편차를 제곱한 값을 이용해 불순도를 측정

▼ 회귀 성능 지표

RMSE

▼ (단일)트리 모델의 장점

블랙박스 문제 해결

1. 데이터 탐색을 위한 시각화 가능
2. 어떤 방식으로 분류되었는지 시각화가 가능
   >> 비전문가에게 설명 용이

Bagging & Random Forest

단일 트리의 장점인 블랙박스 문제 해결이라는 장점을 잃는 대신,

단일 트리를 사용하는 것보다 **다중 트리**를 사용하는게 성능은 좋음

▼ Ensemble

여러 모델의 집합을 이용해서 하나의 예측을 이끌어 내는 방식

- 다수결 투표
- 다중 모델의 평균
- 1906년에 진행된 수소 무게 예측 대회 → 800명 참가
  >> 예측 평균과 중간값이 1% 오차 범위 이내
- 넷플릭스 고객 영화 평점 예측 모델 성능 10% 향상 방법
- 적은 노력 → 좋은 예측 모델

▼ 앙상블을 적용한 트리 모델

▼ Bagging

- 데이터를 부트스트래핑해서 여러 모델을 만드는 일반적인 방법
- Bootstrap Aggregation의 줄임말

- 복원 추출; 같은 데이터를 사용하지 않고 재표본 추출

▼ Random Forest

- 결정 트리 모델에 기반을 둔 배깅 추정 모델
- 블랙박스 모델?? → 모델을 통해 직관적인 해석은 불가능하다
- 예외 사항까지 학습하는 과적합 문제 → 결정 트리처럼 하이퍼 파라미터 조정 필요

▼ 관측치 수에 비해 변수의 수가 많은 고차원 데이터에서 중요 변수 선택 기법으로 활용

레코드(행)를 표본추출 할 때 최적의 기준 **변수를 랜덤 샘플링**

▼ 각 단계마다 몇 개의 변수를 샘플링하는게 좋을까?

전체 특성 p개의 제곱근

▼ 주머니 외부 (Out-of-bag, OOB)

학습에 사용하지 않은 데이터를 이용해 테스트 → 모델의 오차율 계산

파이썬 라이브러리에서는 트리 개수에 대한 oob_score를 추정하기가 쉽지 않음

책에서는 for문으로 트리 개수를 변경해서 학습한 점수를 리스트에 담아 시각화

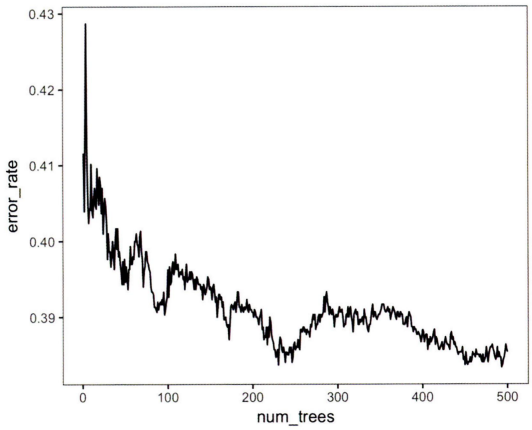


그림 6-6 랜덤 포레스트에서 트리를 추가할 때마다 정확도가 개선되는 예를 볼 수 있다.

▼ Feature Importance

- 피처와 레코드가 많은 데이터에서 상관관계 항들에 대응하는 변수들의 중요도를 파악할 수 있다

- model.fit → model.feature_importances_로 자동 계산
  >> 정확도 감소를 바로 비교하고 사용할 수 없어 다시 for문 사용

1. 정확도 감소량
   변수를 섞고 모델을 돌려 OOB를 통해 정확도를가 감소하는 정도를 측정하는 방법
2. 지니 불순도
   특정 변수를 기준 → 분할이 일어난 모든 노드에서 불순도 점수의 평균 감소량 측정
   >> OOB로 테스트한 데이터가 아닌, 학습 데이터로만 측정 → 신뢰 어려울 수도...

▼ 그럼 지니 불순도는 필요 없는 것인가?

▼ 연산 속도에서의 장점 확보
프로덕션에서는 정확도를 위한 반복 학습을 하기 힘들다.
>> 알고리즘에서 자동 계산하면서 부차적으로 얻어지는 결과물이라 빠르다

Boosting

▼ 모델 → 앙상블

- 배깅과 비슷한 시기에 개발 됨
  배깅 = 집단 지성
  부스팅 = 학사→석사→박사
- 결정 트리에 가장 많이 사용 됨
- 배깅 → 튜닝 많이 필요 없음
  부스팅 → 많은 부가 기능을 다루기 위한 핸들링 필요
- 선형 회귀에서 사용한 잔차를 모티브
  → 이전 모델이 갖는 오차를 줄이는 방향으로 모델 개선

▼ 종류

▼ Adaboost

- 잔차에 따라 데이터의 가중치를 조절하는 부스팅 초기 버젼
- 잘못 분류된 관측 데이터에 가중치를 증가 시킴으로써 현재 성능이 제일 떨어지는 데이터에 대해서 집중 학습
- 모델의 오차가 적을수록 더 큰 가중치를 준다(= 더 중요하게 여긴다)

▼ Gradient Boosting

▼ 비용함수를 최소화 하는 좀 더 보편화 된 모델

가중치 조절 X →유사 잔차 학습

잔차가 큰 데이터를 더 집중적으로 학습

▼ Stochastic Gradient Boosting

▼ 각 라운드 마다 레코드와 열을 재표본추출하는 방식을 추가 (rf와 유사)

매 단계마다 데이터와 특성을 샘플링 → 랜덤 요소 추가

- 대표적으로 XGBOOST가 있다.

▼ XGBOOST

가장 많이 사용되는 오픈 소스