6. 통계적 머신러닝

최근 통계학 분야는 회귀나 분류와 같은 예측 모델링을 자동화하기 위한 더 강력한 기술을 개발하는 것에 집중해왔다. 이러한 방법들은 **통계적 머신러닝**이라는 큰 틀안에 속하며, 데이터에 기반하며 전체적인 구조 (예를 들어 모델이 선형인지)를 가정하지 않는다는 점에서 고전적인 통계 방법과 구별된다.

예를 들면 K 최근접 이웃 방법은 아주 간단하다. 비슷한 레코드들이 어떻게 분류되는지에 따라 해당 레코드를 분류하는 방법이다. **앙상블 학습**을 적용한 **의사 결정 트리**가 지금까지는 가장 성공적이고 널리 사용되는 기술이다.

앙상블 학습의 기본 아이디어는 단일 모델로부터 결과를 얻는 것이 아닌 최종 예측을 얻기 위해 많은 모델을 사용하는 것이다. 의사 결정 트리는 예측변수와 결과변수 사이의 관계 규칙을 학습하는 유연하고 자동화된 기술이다. **앙상블 학습과 의사 결정 나무를 결합하는 방식은 최고의 성능을 얻을 수 있는 예측 모델링 기법이다.**

NOTE_ 머신러닝 대 통계학

예측 모델링의 관점에서 머신러닝과 통계학의 차이점은 무엇일까? 이 두가지 사이에 명확하게 선을 그을 수는 없다. 머신러닝 분야는 예측 모델을 최적화하기 위해서 많은 데이터를 효과적으로 처리하는 알고리즘을 개발하는 데 좀 더 집중하고 있다.

통계학은 확률론과 모델의 구조를 결정하는 데 좀 더 관심을 갖는다. 배깅과 랜덤 포레스트는 확실히 통계학에서 시작됐다고 볼 수 있다. 반면에 부스팅은 양쪽 분야에서 발전했다고 보는 것이 맞고 오히려 요즘에는 머신러닝 분야에서 더 관심을 갖는다. 역사적인 배경과 무관하게, 부스팅 자체는 양쪽 분야 모두에서 기대를 한 몸에 받고 있다.

6.1 K최근접 이웃

K 최근접 이웃 KNN(K-nearest neighbors)) 알고리즘의 아이디어는 아주 간단하다. 각 레코드를 다음과 같이 분류 혹은 예측한다.

1. **특징들이 가장 유사한(즉, 예측변수들이 유사한) K개의 레코드를 찾는다.**
2. **분류 : 이 유사한 레코드들 중에 다수가 속한 클래스가 무엇인지 찾은 후에 새로운 레코드를 그 클래스에 할당한다.**
3. **예측 ($KNN_{\text{회귀}}$ 라고도 함) : 유사한 레코드들의 평균을 찾아서 새로운 레코드에 대한 예측값으로 사용한다.**

용어정리

- **이웃 : 예측변수에서 값들이 유사한 레코드**
- **거리 지표 : 각 레코드 사이가 얼마나 멀리 떨어져 있는지를 나타내는 단일 값**
- **표준화 : 평균을 뺀 후에 표준편차로 나누는 일(유의어 : 정규화)**
- **z-점수 : 표준화를 통해 얻은 값**
- **K : 최근접 이웃을 계산하는 데 사용되는 이웃의 개수**

KNN은 가장 간단한 예측/분류 방법 중 하나다. 회귀와는 달리 모델을 피팅하는 과정이 필요 없다. 그렇다고 KNN이 완전히 자동화된 방법이라는 의미는 아니다. 특징들이 어떤 척도에 존재하는지, 가까운 정도를 어떻게 측정할 것인지, K를 어떻게 설정할 것인지에 따라 예측 결과가 달라진다.

또한 모든 예측변수들은 수치형이어야 한다. 분류 예제를 통해 더 자세히 알아보자.

6.1.1 예제: 대출 연체 예측

다음 표는 랜딩 클럽에서 얻은 개인 대출 정보의 일부이다. 랜딩 클럽은 투자자들이 모은 돈을 개인에게 대출해주는 P2P 방식의 대출 업체이며 이 분야 선두에 있다. 새 잠재 대출의 결과(상환 혹은 연체)를 예측하고자 한다.

예측변수 두 가지만을 고려한 가장 간단한 모델을 생각해보자. 변수 payment_inc_ratio는 소득에 대한 대출 상환 비율이며, 변수 dti는 소득에 대한 부채(모기지는 제외) 비율을 뜻한다.

두 변수 모두 100을 곱한 값을 사용한다. 이진 결과를 알고 있는 200개의 대출만을 뽑아 만든 작은 데이터 집합 loan200을 사용하겠다. 이제 K를 20으로 할 때, payment_inc_ratio = 9, dti = 22.5인 새로운 대출에 대한 KNN 예측 결과는 newloan을 다음과 같이 구할 수 있다.

```
> ## KNN
> ## the first row of loan200 is the target data
> newloan <- loan200[1, 2:3, drop=FALSE]
> newloan
  payment_inc_ratio  dti
1                 9 22.5
> knn_pred <- knn(train=loan200[-1,2:3], test=newloan, cl=loan200[-1,1], k=20)
> knn_pred == 'paid off'
[1] TRUE
```

KNN 결과 이 새로운 대출은 상환될 것으로 예상된다. R은 기본적으로 knn함수를 제공한다. 이외에도 FNN(빠른 최근접 이웃 알고리즘)과 같이 빅데이터를 다루기에 더 낫고, 좀 더 사용자에게 유연성을 제공하는 패키지도 있다.

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

위 그림은 이 예제의 내용을 시각화한 모습이다.

원 중심점은 새로운 대출에 대한 예측 결과이다.

엑스(연체)와 동그라미(상환)은 학습 데이터를 의미한다. 검은 실선은 가장 가까운 20개의 점들에 대한 경계선을 보여준다. 이 경우, 가까운 9개의 대출에서 연체가 발생하고, 11번의 상환이 이뤄졌다는 것을 볼 수 있다. 따라서 예측 결과는 상환이 된다.

NOTE_ 분류문제에서 KNN의 결과는, 대출 데이터에서의 연체 또는 상환처럼 보통은 이진형이지만, 때로는 0과 1 사이의 확률(경향)을 결과로 줄 수도 있다. K개의 가장 가까운 점들이 속한 클래스의 비율을 통해 확률을 정할 수 있다. 앞 예제에서, 결과가 연체일 확률은 12/20 = 0.6이 될 것이다.

확률 점수를 통해 간단한 다수결 투표 결과와는 다른 분류 규칙을 만들 수 있다. 이는 불균형 문제를 다루는 데 특히 중요하다. 5.5절을 참고하자. 예를 들면 희귀한 클래스의 데이터를 잘 분류하는 것이 목표라면 컷오프는 보통 50% 미만이 되어야 할 것이다. 희귀한 사건에 대한 확률로 컷오프를 정하는 것이 매우 일반적인 방법이다.

6.1.2 거리 지표

유사성similiarity(근접성 nearness)은 **거리 지표**를 통해 결정된다. 이 지표는 두 데이터$(x_1, x_2, \cdots, x_p)$와 $(u_1, u_2, \cdots, u_p)$ 가 서로 얼마나 멀리 떨어져 있는지를 측정하는 함수라고 할 수 있다. 두 벡터 사이에 가장 많이 사용되는 지표는 **유클리드 거리**이다. 이 두 벡터 사이의 유클리드 거리를 구하려면 서로의 차이에 대한 제곱합을 구한 후 그 값의 제곱근을 취한다.

$$\sqrt{(x_1 - u_1)^2 + (x_2 - u_2)^2 + \cdots + (x_p - u_p)^2}$$

유클리드 거리는 특별히 계산상 이점이 있다. KNN 은 $K \times n$(데이터 개수) 번 만큼 쌍대(pairwise)

비교가 필요하기 때문에 데이터 개수가 커질수록 계산량이 더 중요해진다.

또 다음으로 많이 사용하는 거리 지표는 **맨하튼 거리** 이다.

$$|x_1 - u_1| + |x_2 - u_2| + \cdots + |x_p - u_p|$$

유클리드 거리는 두 점 사이의 직선 거리라고 볼 수 있다. 반면, 맨하튼 거리는 한 번에 대각선이 아닌 한 축 방향으로만 움직일 수 있다고 할 때 (도심지에서 직사각형 건물들 사이을 이동한다고 할 때), 두 점 사이의 거리라고 할 수 있다. 따라서 맨하튼 거리는 점과 점 사이의 이동 시간으로 근접성을 따질 때 좋은 지표가 된다.

두 벡터 사이의 거리를 측정할 때, 상대적으로 큰 스케일에서 측정된 변수(특징)들이 측정치에 미치는 영향이 클 것이다. 예로 대출 데이터에서 소득과 대출 금액을 변수로 하는 거리 함수를 생각해보면 몇 천만 원에서 억 단위까지 생각할 수 있을 것이다. 이에 반해 비율을 나타내는 변수들은 상대적으로 거의 거리에 영향을 주지 못할 것이다. 데이터 표준화를 통해 이러한 문제를 다룰 수 있다.

이에 대해 자세히 알아보자.

NOTE_ 다른 거리 지표

벡터 사이의 거리를 측정하기 위해 수많은 거리 지표들이 있다. 수치 데이터를 다룰 때, 마할라노비스 거리는 두 변수 간의 상관관계를 사용하기 때문에 장점이 있다. 두 변수 사이에 높은 상관관계가 있다면 아주 유용하다. 마할라노비스 거리는 이를 거리라는 하나의 값으로 표현할 수 있다.

유클리드나 맨하튼 거리는 이러한 상관성을 고려하지 않다보니 상관성이 있는 변수들에서 원인이 되는 속성에 더 큰 가중치를 두고 계산해야 한다. 마할라노비스 거리를 사용할 때 단점은 많은 계산이 필요하고 복잡성이 증가한다는 점이다. 계싼에 공분산 행렬을 사용하기 때문이다.

6.1.3 원-핫 인코더

위 표는 몇 가지 요인(문자열) 변수를 포함하고 있다. 대부분의 통계 모델이나 머신러닝 모델에서 이러한 형태의 변수는, 다음 표와 같이 정보를 담고 있는 이진 가변수의 집합으로 변환해야 한다.

주택 소유 상태를 '저당이 있는 소유','저당이 없는 소유', '월세', '기타'로 나타내는 하나의 변수 대신에 네 개의 이진변수로 만드는 것이다.

이렇게 주택 소유 상태라는 하나의 예측변수를 하나의 1과 세 개의 0으로 이뤄진 벡터 형태로 변환하면 통계적 알고리즘에 사용하기 편리해진다. 이러한 **원-핫 인코딩**이라는 용어는 디지털 회로 분야에서 유래한 것으로, 하나의 비트만 양수로 허용되는 회로 설정을 말한다.

NOTE_ 선형회귀나 로지스틱 회귀에서 원-핫 인코딩은 다중공선성과 관련된 문제를 일으킨다. 이런 경우 한 기변수를 생략하는 방법이 있다(그 값은 다른 값들로부터 유추할 수 있기 때문에). 하지만 KNN이나 다른 방법에서는 이것이 문제가 되지 않는다.

6.1.4 표준화(정규화, z점수)

측정할 때는, 종종 값이 '얼마인지' 보다 '얼마나 평균과 차이가 나는지'에 더 관심이 있을 때가 있다.

표준화 혹은 **정규화**란, 모든 변수에서 평균을 빼고 표준편차로 나누는 과정을 통해 변수들을 모두 비슷한 스케일에 놓는다. 이러한 방식으로, 실제 측정된 값의 스케일 때문에 모델에 심한 영향을 주는 것을 막을 수 있다.

$$z = \frac{x - \bar{x}}{s}$$

이 값을 일반적으로 **z 점수**라고 부른다. 이는 평균으로부터 표준편차만큼 얼마나 떨어져 있는지를 의미한다. 이를 통해 변수의 스케일로 인한 영향을 줄일 수 있다.

CAUTION_ 통계적인 맥락에서의 정규화를 DB의 표준화와 혼동하지 않도록 하자. DB의 표준화란, DB 설계시 데이터의 중복을 줄이고 데이터 의존성을 확인하는 과정을 말한다.

KNN이나 다른 알고리즘(예로 주성분분석과 클러스터링)에서는 데이터를 미리 표준화하는 것이 필수다. 예시로 6.1.1절에서 사용한 대출 데이터 예제에 KNN을 적용해보겠다. 앞 예제에서 고려했던 변수 dti와 payment_inc_ratio, 그리고 또 다른 두 가지 변수를 추가로 고려한다. 이는 달러로 신청할 수 있는 총 회전 신용을 의미하는 변수 revol_bal와 이미 사용 중인 신용 비율을 의미하는 변수 revol_util이다. 예측할 새로운 데이터는 다음과 같다.

```
> newloan = loan_df[1,, drop=FALSE]
> newloan
  payment_inc_ratio dti revol_bal revol_util
1            2.3932   1      1687        9.4
```

달러로 표시된 revol_bal 값의 크기가 다른 변수들의 값보다 훨씬 큰 것을 알 수 있다. knn함수를 통해 새로운 데이터에서 가장 가까운 이웃들의 인덱스(nn.index 속성에 저장)를 구할 수 있다.

이를 통해, loan_df에서 가장 가까운 상위 5개 행을 표시할 수 있다.

```
loan_df <- model.matrix(~ -1 + payment_inc_ratio + dti + revol_bal + revol_util,
data=loan_data)
newloan = loan_df[1,, drop=FALSE]
loan_df = loan_df[-1,]
outcome <- loan_data[-1,1]
knn_pred <- knn(train=loan_df, test=newloan, cl=outcome, k=5)
knn_pred
> loan_df[attr(knn_pred,"nn.index"),]
      payment_inc_ratio  dti revol_bal revol_util
35537           1.47212 1.46      1686       10.0
33652           3.38178 6.37      1688        8.4
25864           2.36303 1.39      1691        3.5
42954           1.28160 7.14      1684        3.9
43600           4.12244 8.98      1684        7.2
```

이 이웃들의 revol_bal의 값은 새 데이터 값과 아주 비슷하지만, 다른 예측변수들은 넓게 퍼져 있는 것을 볼 수 있다. 이를 통해, 이웃들을 결정할 때, 다른 변수들의 역활이 별로 중요하지 않았다고 볼 수 있다.

이것을 R에서 제공하는 scale 함수를 이용해 데이터를 표준화한 후에 적용한 KNN과 비교해보자.

이 함수는 각 변수의 z 점수를 계산해준다.

```
loan_df <- model.matrix(~ -1 + payment_inc_ratio + dti + revol_bal + revol_util,
data=loan_data)
loan_std <- scale(loan_df)
target_std = loan_std[1,, drop=FALSE]
loan_std = loan_std[-1,]
outcome <- loan_data[-1,1]
knn_pred <- knn(train=loan_std, test=target_std, cl=outcome, k=5)
knn_pred
```

```
> loan_df[attr(knn_pred,"nn.index"),]
      payment_inc_ratio   dti revol_bal revol_util
2080           10.04400 19.89      9179       51.5
1438            3.87890  5.31      1687       51.1
30215           6.71820 15.44      4295       26.0
28542           6.93816 20.31     11182       76.1
44737           8.20170 16.65      5244       73.9
```

새롭게 얻은 5개의 최근접 이웃들은 모든 변수에서 훨씬 더 유사한 것을 볼 수 있다. 결과를 원래 스케일로 표시했지만, KNN안에서는 새로운 데이터에 대한 예측 결과를 얻는 데 표준화된 변수를 적용한다.

TIP_ z점수는 변수를 변환하는 여러 방법 중 한 가지일 뿐이다. 평균 대신에 중간값과 같은 좀 더 로버스트한 위치 추정값을 사용할 수도 있다. 마찬가지로, 표준편차 대신에 사분위범위와 같은 다른 척도 추정 방법을 사용할 수 있다. 가끔은 변수를 0과 1 사이로 축소하는 것이 무리한 것 처럼 보인다. 단위 분산을 갖도록 모든 변수를 조정하는 것이 다소 억지스러울 수 있다는 것을 인식하는 것도 중요하다. 이는 각 변수가 예측력 측면에서 갖는 중요성이 모두 같다는 것을 의미한다. 일부 변수가 다른 변수보다 중요하다는 주관적 지식이 있는 경우, 이러한 사실을 적용할 수 있다.

예를 들면 대출 데이터의 경우, 소득 대비 상환 비율이 매우 중요하다고 판단하는 것이 합리적이다.

6.1.5 K 선택하기

K를 잘 선택하는 것은 KNN의 성능을 결정하는 아주 중요한 요소이다. 가장 간단한 방법은 $K = 1$로 놓는 방법이다. 이는 1-최근접 이웃 분류기가 된다. 이 예측 방법은 매우 직관적이다.

새로 들어온 데이터와 가장 가까운 데이터를 찾아 예측 결과로 사용한다. 하지만 거의 대부분 $K = 1$이 가장 좋은 결과를 주진 않는다. $K > 1$ 일 때 더 좋은 결과를 보인다.

일반적으로, K 가 너무 작으면 데이터의 노이즈 성분까지 고려하는 오버피팅 문제가 발생한다.

K값이 클수록 결정 함수를 좀 더 부드럽게 하는 효과를 가져와 학습 데이터에서의 오버피팅 위험을 낮출 수 있다. 반대로 K를 너무 크게하면 결정함수가 너무 과하게 평탄화되어(오버스무딩) 데이터의 지역 정보를 예측하는 KNN의 기능을 잃어버리게 된다.

오버피팅과 오버 스무딩 사이의 균형을 맞춘 최적의 K값을 찾기 위해 정확도 지표들을 활용한다. 특히 홀드아웃 데이터 또는 타당성검사를 위해 따로 떼어놓은 데이터에서의 정확도를 가지고 K값을 결정하는데 사용한다. **물론 최적의 K를 결정하는 일반적인 규칙은 없다.** 데이터에 따라 크게 달라진다. **데이터에 노이즈가 거의 없고 아주 잘 구조화된 데이터의 경우 K값이 작을 수록 잘 동작한다.** 신호처리 분야에서 사용하는 전문용어를 잠시 빌려오자면, **신호 대 잡음비SNR(Signal to noise ratio)** 가 높은 데이터라는 의미이다. 일반적으로 손글씨 데이터 또는 음성 인식 데이터가 SNR이 높은 데이터라고 할 수 있다. 반면에 대출 데이터와 같이 노이즈가 많아 SNR이 낮은 데이터의 경우 K가 클수록 좋다. 보통 K를 1에서 20사이에 놓는다. 동률이 나오는 경우를 막기 위해 보통은 홀수를 사용한다.

NOTE_ 편향 분산 트레이드 오프

*과잉 평탄화(오버 스무딩)와 과잉 적합화(오버피팅) 사이의 이율배반 관계를 **편향-분산 트레이드 오프**라고 하며, 이 문제는 통계적 모델 적합화(피팅)에서 늘 존재하는 문제이다.*

분산은 학습 데이터 선정에 따라 발생하는 모델링 오차를 말한다. 즉 다른 학습 데이터 집합을 사용할 경우, 결과적으로 나오는 모델이 달라지는 정도를 의미한다.

편향은 모델이 실제 세계를 정확히 표현하지 못하기 때문에 발생하는 모델링 오차를 말한다. 이는 모델에 데이터를 추가한다고 해서 나아지지 않는다. 유연한 모델에서 오버피팅이 발생했다면 분산이 증가했다는 것을 의미한다. 이를 줄이기 위해 단순한 모델을 사용하는 것도 하나의 방법이다.

*하지만 모델이 실제 문제를 정확히 표현하지 못할 정도로 유연성을 잃어버린다면 편향이 증가할 수도 있다. 이러한 트레이브 오프를 다루기 위해서 일반적으로 **교차타당성검사** 방법을 사용한다.*

6.1.6 KNN을 통한 피처 엔지니어링

KNN은 구현이 간단하고 직관적이다 보니 널리 활용된다. 성능 면에서는, 다른 복잡한 분류 방법들에 비해 그렇게 경쟁력이 있다고 보기 어렵다. 하지만 실용적인 측면에서, 다른 분류 방법들의 특정 단계에 사용할 수 있게 모델에 '지역적 정보(local knowlegde)' 를 추가하기 위해 KNN을 사용할 수 있다.

1. KNN은 데이터에 기반하여 분류 결과(클래스에 속할 확률)를 얻는다.
2. 이 결과는 해당 레코드에 새로운 특징(피처) 으로 추가된다. 이 결과를 다른 분류 방법에 사용된다. 원래의 예측변수들을 두 번씩 사용하는 셈이 된다.

여기서 어떤 예측변수들을 두 번 사용한다는 것이 다중공선성 같은 문제를 야기하지는 않을까 궁금해할 수 있다. 하지만 이는 문제가 되지 않는다. 위의 2단계에서 얻은 정보는 소수에 접근한 레코드들로부터 얻은 매우 지엽적인 정보이기 때문이다. 따라서 새로 얻은 정보는 불필요하거나, 중복성이 있지 않다.

NOTE_ KNN을 이러한 식으로, 분류 단계의 일부로 사용하는 것은 마치 여러 가지의 모델링 방법을 합쳐 하나의 예측 결과를 만들어내는 앙상블 학습 형태로 볼 수 있다. 또한 예측 능력을 높이기 위해 피처(예측변수)들을 새롭게 만들어내기 위한, '피쳐 엔지니어링' 개념으로 볼 수도 있다. 피처 엔지니어링을 위해서는 데이터를 일일이 손수 재조사해야 할 수도 있는데, KNN은 이를 꽤나 자동화한 방법이라고 볼 수 있다.

예를 들어 킹 카운티 주택 데이터를 다시 생각해보자. 주택 가격을 산정할 때, 부동산 중계업자는 최근에 팔린 비슷한 집들의 가격(흔히 comps라고 한다)을 기준으로 삼을 것이다. 결국 중계업자들은 비슷한 주택의 매매 가격을 일일이 확인하면서 일종의 수동식 KNN을 하고 있다고 볼 수 있다.

이를 통해 이 집이 어느 정도의 가격에[팔릴지를 예측할 수 있다. 우리는 최근 거래 정보에 KNN을 적용해 부동산 전문가들이 하는 것을 통계 모델을 통해 모방하도록 새로운 특징을 하나 만들었다.

예측 결과는 판매 가격이 되고 예측 변수들은 지역, 평수, 구조, 대치 면적, 침실 수 ,욕실 수등을 포함했다.

KNN을 통해 추가하려고 하는 새로운 예측 변수는 각 레코드에 대한 (중계업자들의 comps와 유사한) KNN 예측변수이다. 예측 결과가 수치형 변수이기 때문에, 다수결 결과가 아닌 K 최근접 이웃값의 평균을 사용한다. (**KNN회귀**)

이와 비슷하게 대출 데이터에 대해, 대출 과정의 다른 측면을 나타내는 특징들을 만들 수 있다.

예를 들면 아래 과정은 대출자의 신용정보를 나타내는 피처를 만드는 것을 보여준다.

```
> ## Create a feature for borrowers
> borrow_df <- model.matrix(~ -1 + dti + revol_bal + revol_util + open_acc +
+                             delinq_2yrs_zero + pub_rec_zero, data=loan_data)
> borrow_knn <- knn(borrow_df, test=borrow_df, cl=loan_data[, 'outcome'],
prob=TRUE, k=20)
> prob <- attr(borrow_knn, "prob")
> borrow_feature <- ifelse(borrow_knn=='default', 1-prob, prob)
> summary(borrow_feature)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  0.050   0.400   0.500   0.499   0.600   1.000
```

신용 기록을 기초로 대출자가 대출을 갚지 못할 것으로 예상되는 정도를 나타내는 피쳐를 만들었다.

주요 개념

- **K 최근접 이웃(KNN) 방법이란 유사한 레코드들이 속한 클래스로 레코드를 분류하는 것이다.**

- **유사성(거리)은 유클리드 거리나 다른 관련 지표들을 이용해 결정한다.**
- **가장 가까운 이웃 데이터의 개수를 의미하는 K 는 학습 데이터에서 얼마나 좋은 성능을 보이는지를 가지고 결정한다.**
- **일반적으로 예측 변수들을 표준화한다. 이를 통해 스케일이 큰 변수들의 영향력이 너무 커지지 않도록 한다.**
- **예측 모델링의 첫 단계에서 종종 KNN을 사용한다. 이렇게 얻은 값을 다시 데이터에 하나의 예측변수로 추가해서 두 번째 단계의 (KNN이 아닌) 모델링을 위해 사용한다.**

6.2 트리 모델

트리 모델은 **회귀 및 분석 트리(classification and regression tree(CART)), 의사결정트리(decision tree),** 혹은 단순히 **트리** 라고도 불린다. 트리 모델들과 여기서 파생된 강력한 **랜덤 포레스트(Random Forest)** 와 **부스팅** 같은 방법들은 회귀나 분류 문제를 위해 데이터 과학에서 가장 널리 사용되는 강력한 예측 모델링 기법들의 기초라고 할 수 있다.

용어정리

- **재귀 분할(recursive partitioning)** : 마지막 분할 영역에 해당하는 출력이 최대한 비슷한(homogeneous) 결과를 보이도록 데이터를 반복적으로 분할 하는 것
- **분할값(split value)** : 분할값을 기준으로 예측변수를 그 값 보다 작은 영역과 큰 영역으로 나눈다.
- **마디(노드)** : 의사 결정 트리와 같은 가지치기 형태로 구성된 규칙들의 집합에서, 노드는 분할 규칙의 시각적인 표시라고 할 수 있다.
- **잎(leaf)** : if-then 규칙의 가장 마지막 부분, 혹은 트리의 마지막 가지(brench) 부분을 의미한다. 트리 모델에서 잎 노드는 어떤 레코드에 적용할 최종적인 분류 규칙을 의미한다.
- **손실(loss)** : 분류하는 과정에서 발생하는 오분류의 수. 손실이 클수록 불손도가 높다고 할 수 있다.
- **불순도(impurity)** : 데이터를 분할한 집합에서 서로 다른 클래스의 데이터가 얼마나 섞여 있는지를 나타낸다. 더 많이 섞여 있을수록 불순도가 높다고 할 수 있다. **(유의어 : 이질성(heterogeneity), 반의어 : 동질성(homogeneity), 순도)**
- **가지치기(pruning)** : 학습이 끝난 트리 모델에서 오버피팅을 줄이기 위해 가지들을 하나씩 잘라내는 과정

트리 모델이란 쉽게 말해 if-then-else 규칙의 집합체라고 할 수 있다. 따라서 이해하기도 쉽고 구현하기도 쉽다. 회귀나 로지스틱 회귀와 반대로 트리는 데이터에 존재하는 복잡한 상호관계에 따른 숨겨진 패턴들을 발견하는 발견하는 능력이 있다. 게대가 KNN이나 나이브 베이즈 모델과 달리, 예측 변수들 사이의 관계로 단순 트리 모델을 펴시할 수 있고 쉽게 해석이 가능하다.

CAUTION_ 운용과학에서의 의사 결정 트리

인간의 의사 결정 과정에 대해 연구하는 의사 결정 과학이나 운용과학 분야에서, 의사 결정 트리라는 용어는 다른 의미를 갖는다. 이 분야에서는 분기 다이어그램에 나타난 결정 지점, 가능한 결과와 예상 확률 등이 주어진 상태에서, 최대 기댓값을 갖는 의사 결정 경로를 선택하는 것을 의미한다.

6.2.1 간단한 예제

트리 모델을 얻기 위해 주로 사용하는 R 패키지로는 rpart와 tree가 있다. rpart 패키지를 이용해, 3,000개의 대출 데이터에 적합한 트리 모델을 다음과 같이 만들어 보자. 여기서는 payment_inc_ratio 와 borrower_score 두 변수를 고려한다.

```
library(rpart)
loan_tree <- rpart(outcome ~ borrower_score + payment_inc_ratio,
                   data=loan3000,
                   control = rpart.control(cp=.005))
loan_tree

## Figure 6-3: Rules for simple tree model (not same as in book)
png(filename=file.path(PSDS_PATH, 'figures', 'psds_rpart_tree.png'),
    width = 6, height=4, units='in', res=300)

par(mar=c(0,0,0,0)+.1)
plot(loan_tree, uniform=TRUE, margin=.05)
text(loan_tree, cex=.75)
```

위 코드에 대한 결과이다. 이러한 분류 규칙은 루트 노드에서 시작하여, 노드가 참이면 왼쪽으로 가고 아니면 오른쪽으로 가면서, 잎 노드에 도달할 때까지 계측구조의 트리를 통과하며 결정된다.

다음과 같이 손쉽게 트리 구조를 보기 좋은 형태로 출력할 수 있다.

```
> loan_tree
n= 3000

node), split, n, loss, yval, (yprob)
      * denotes terminal node

 1) root 3000 1445 paid off (0.5183333 0.4816667)  (상환 확률, 연체 확률) 3,000개 중, 1,445 개 참
   2) borrower_score>=0.575 878  261(loss 개수) paid off (0.7027335 0.2972665) * (true) leaf(3000 X 0.48 = 1445)
   3) borrower_score< 0.575 2122  938 default (0.4420358 0.5579642)
     6) borrower_score>=0.375 1639  802 default (0.4893228 0.5106772)
      12) payment_inc_ratio< 10.42265 1157  547 paid off (0.5272256 0.4727744)
        24) payment_inc_ratio< 4.42601 334  139 paid off (0.5838323 0.4161677) *
        25) payment_inc_ratio>=4.42601 823  408 paid off (0.5042527 0.4957473)
          50) borrower_score>=0.475 418  190 paid off (0.5454545 0.4545455) *
          51) borrower_score< 0.475 405  187 default (0.4617284 0.5382716) *
      13) payment_inc_ratio>=10.42265 482  192 default (0.3983402 0.6016598) *
     7) borrower_score< 0.375 483  136 default (0.2815735 0.7184265) *
```

트리의 깊이를 들여쓰기의 정도로 쉽게 파악할 수 있다. 각 노드는 해당 분할 규칙에 대해 우세한 쪽으로 결정하는 임시 분류를 의미한다. '손실'은 이러한 임시 분할에서 발생하는 오분류의 개수를 의미한다. 예로 노드2를 보면전체 1,139의 전체 데이터 가운데 261개의 오분류가 있었다.

괄호 안의 숫자는 해당 데이터에서 대출 상환과 대출 연체 각각의 비율을 의미한다. 예를 들면 노드 13에서는 해당 레코드를 중 60% 정도가 대출 연체라는 것을 말해준다.

6.2.2 재귀 분할 알고리즘

의사 결정 트리를 만들 때는 **재귀 분할**이라고 하는 알고리즘을 사용한다. 간단하면서도 직관적인 방법이다. 예측변수 값을 기주으로 데이터를 반복적으로 분할해나간다. 분할할 때에는 상대적으로 같은 클래스의 데이터틀끼리 구분되도록 한다. 다음 그림은 위 트리 구조를 통해 만들어진 분할 영역을 보여준다. 가장 첫 번째 규칙은 borrower_score >= 0.525이고 이는 규칙1(rule_number가 1인 실선)로 그림에 표시된다. 그림에서 두 번째 규칙은 payment_inc_ratio < 9.732이며 왼쪽 영역을 둘로 나누고 있다.

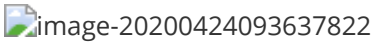
```
r_tree <- data_frame(x1 = c(0.575, 0.375, 0.375, 0.375, 0.475),
                     x2 = c(0.575, 0.375, 0.575, 0.575, 0.475),
                     y1 = c(0,          0, 10.42, 4.426, 4.426),
                     y2 = c(25,        25, 10.42, 4.426, 10.42),
                     rule_number = factor(c(1, 2, 3, 4, 5)))
r_tree <- as.data.frame(r_tree)

labs <- data.frame(x=c(.575 + (1-.575)/2,
                       .375/2,
                       (.375 + .575)/2,
                       (.375 + .575)/2,
                       (.475 + .575)/2,
                       (.375 + .475)/2
                       ),
                   y=c(12.5,
                       12.5,
                       10.42 + (25-10.42)/2,
                       4.426/2,
                       4.426 + (10.42-4.426)/2,
                       4.426 + (10.42-4.426)/2
                       ),
                   decision = factor(c('paid off', 'default', 'default', 'paid
off', 'paid off', 'default')))


png(filename=file.path(PSDS_PATH, 'figures', 'psds_0604.png'),
    width = 6, height=4, units='in', res=300)

ggplot(data=loan3000, aes(x=borrower_score, y=payment_inc_ratio)) +
  geom_point( aes(color=outcome, shape=outcome), alpha=.5) +
  scale_color_manual(values=c('blue', 'red')) +
  scale_shape_manual(values = c(1, 46)) +
  # scale_shape_discrete(solid=FALSE) +
  geom_segment(data=r_tree, aes(x=x1, y=y1, xend=x2, yend=y2,
linetype=rule_number), size=1.5, alpha=.7) +
  guides(colour = guide_legend(override.aes = list(size=1.5)),
         linetype = guide_legend(keywidth=3, override.aes = list(size=1))) +
  scale_x_continuous(expand=c(0,0)) +
  scale_y_continuous(expand=c(0,0), limits=c(0, 25)) +
  geom_label(data=labs, aes(x=x, y=y, label=decision)) +
  #theme(legend.position='bottom') +
  theme_bw()

dev.off()
```



응답변수 Y와 P개의 예측변수 집합 $X_j (j = 1, \cdots, P)$가 있다고 가정하자. 어떤 파티션 (분할 영역) A에 대해, A를 두 개의 하위 분할 영역으로 나누기 위한 가장 좋은 재귀적 분할 방법을 찾아야 한다.

1. **각 예측변수 X_j에 대해,**
    - **a. X_j에 해당하는 각 변수 s_j에 대해**
        - **i. A에 해당하는 모든 레코드를 $X_j < s_j$ 인 부분과 나머지 $X_j \geq s_j$ 인 부분으로 나눈다.**
        - **ii. A의 각 하위 분할 영역 안에 해당 클래스의 동질성을 측정한다.**

b. 하위 분할 영역 내에서의 클래스 동질성이 가장 큰 s_j 값을 선택한다.

2. **클래스 동질 성이 가장 큰 변수 X_j 와 s_j 값을 선택한다.**

이제 알고리즘의 재귀 부분이 나온다.

1. **전체 데이터를 가지고 A를 초기화한다.**
2. **A를 두 부분 A_1 과 A_2 으로 나누기 위해 분할 알고리즘을 적용한다.**
3. **A_1 과 A_2 각각에서 2번 과정을 반복한다.**
4. **분할을 해도 더 이상 하위 분할 영역의 동질성이 개선되지 않을 정도로 충분히 분할을 진행했을 때, 알고리즘을 종료한다.**

최종 결과는 2차원이 아니라 P 차원이라는 점에서 다를 뿐 위 그림과 같은 데이터 분할 영역이다. 각 영역은 해당 영역에 속한 응답변수들의 다수결 결과에 따라 0 또는 1로 예측 결과를 결정한다.

NOTE_ 0 또는 1의 이진 결과 외에, 트리 모델은 하위 분할 영역에 존재하는 0과 1의 개수에 따라 확률값을 구할 수도 있다. 간단히 분할 영역에 속하는 0 혹은 1의 개수를 영역에 속한 전체 데이터의 개수로 나누면 구할 수 있다.

$P(Y=1)$ = $\frac{\text{파티션 내 1의 개수}}{\text{파티션의 크기}}$

이렇게 얻은 확률 $P(Y=1)$ 을 통해 이진 결정을 할 수 있다. 예를 들어 $P(Y=1) > 0.5$ 인 경우, 1로 예측할 수 있다.

6.2.3 동질성과 불순도 측정하기

트리 모델링은 분할 영역 A(기록 집합)를 재귀적으로 만드는 과정이라고 볼 수 있다. 이렇게 만들어진 분할 영역을 통해 $Y=0$ 혹은 $Y=1$ 의 결과를 예측한다. 앞서 설명한 알고리즘에서 이미 소개했듯이, 각 분할 영역에 대한 동질성, 즉 **클래스 순도**를 측정하는 방법이 필요하다. 혹은 동일한 목적을 위해 불순도를 측정해도 된다. 해당 파티션 내에서 오분류된 레코드의 비율 p로 예측의 정확도를 표시할 수 있으며 이는 **0(완전)에서 0.5(순수 랜덤 추측) 사이의 값**을 갖는다.

정확도는 불순도를 측정하는 데 썩 좋지 않은 것으로 밝혀졌다. 대신 **지니 불순도**와 **엔트로피**가 대표적인 불순도 측정 지표이다. 이들 불순도 지표들은 클래스가 2개 이상인 분류 문제에도 적용 가능하지만 일단 여기서는 설명을 위해 이진 예제를 사용한다. 분할 영역 A의 지니 불순도는 다음과 같이 정의된다.

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

엔트로피의 경우는 다음과 같다.

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

```
## Gini coefficient and impurity

info <- function(x){
  info <- ifelse(x==0, 0, -x * log2(x) - (1-x) * log2(1-x))
  return(info)
}
x <- 0:50/100
plot(x, info(x) + info(1-x))

gini <- function(x){
  return(x * (1-x))
}
```

```
plot(x, gini(x))

impure <- data.frame(p = rep(x, 3),
                     impurity = c(2*x,
                                  gini(x)/gini(.5)*info(.5),
                                  info(x)),
                     type = rep(c('Accuracy', 'Gini', 'Entropy'), rep(51,3)))

## Figure 06-05: comparison of impurity measures
png(filename=file.path(PSDS_PATH, 'figures', 'psds_0605.png'), width = 5,
height=4, units='in', res=300)

ggplot(data=impure, aes(x=p, y=impurity, linetype=type, color=type)) +
  geom_line(size=1.5) +
  guides( linetype = guide_legend( keywidth=3, override.aes = list(size=1))) +
  scale_x_continuous(expand=c(0,0.01)) +
  scale_y_continuous(expand=c(0,0.01)) +
  theme_bw() +
  theme( legend.title=element_blank())

dev.off()
```

CAUTION_ 지니계수

지니 불순도를 지니계수와 혼동해서는 안 된다. 둘 다 모두 개념적으로 비슷하지만 지니 계수는 이진 분류 문제로 한정되며, AUC 지표와 관련이 있는 용어다.

앞에서 설명한 분할 알고리즘에서 이 불순도 측정 지표를 사용했다. 분할로 만들어지는 각 영역에 대해 불순도를 측정한 다음, 가중평균을 계산하고 (단계마다) 가장 낮은 가중평균을 보이는 분할 영역을 선택한다.

6.2.4 트리 형성 중지하기

트리가 더 커질수록 분할 규칙들은 더 세분화되고, 실제 믿을 만한 관계들을 확인 '큰' 규칙을 만드는 단계에서 노이즈까지 반영하는 '아주 작은' 규칙을 만드는 단계로 점점 변화한다. 앞에서의 순도가 완전히 100%가 될 때까지 다 자란 트리는 학습한 데이터에 대해 100%의 정확도를 갖게 된다.

물론 이 정확도는 오버피팅에 의해 얻은 허황된 것이다. 학습 데이터의 노이즈까지 학습한 결과이고 새로운 데이터를 분류할 때 이것은 방해가 된다.

NOTE_ 가지치기

트리의 크기를 줄이는 단순하면서도 가장 직관적인 방법은 트리 말단의 작은 가지들을 제거하는 가지치기를 하는 것이다. 그렇다면 가지치기를 어느 정도 해야할까? 가장 일반적인 방법은 홀드아웃 데이터에서의 에러가 최소가 되는 지점까지 가지치기를 진행하는 것이다. 여러 트리에서 얻은 예측 결과를 하나로 합칠 때에도 마찬가지로 트리 성장을 멈추는 방법이 필요하다. 가지치기는 앙상블 기법에서 사용할 트리들을 얼마나 크게 만들지 결정하기 위한 교차타당성검사에서 중요한 역할을 담당한다.

모델이 새로 들어오는 데이터에 대해 좋은 일반화 성능을 얻기 위해 언제 트리 성장을 멈춰야 하는지 결정하는 방법이 필요하다. 가지 분할을 멈추는 대표적인 두 가지 방법이 있다.

- *분할을 통해 얻어지는 하위 영역 또는 말단 잎의 크기가 너무 작다면 분할하는 것을 멈춘다. rpart 함수에서 minsplit나 minbucket 같은 파라미터를 이용해 최소 분할 영역 크기나 말단 잎의 크기를 조절할 수 있다. 기본값은 각각 20과 7이다.*
- *새로운 분할 영역이 '유의미'한 정도로 불순도를 줄이지 않는다면 굳이 분할하지 않는다. rpart 함수에서 트리의 복잡도를 의미하는 **복잡도 파라미터(complexity parameter)** 인 cp를 이용해 이를 조절한다. 트리가 복잡해질수록 cp의 값이 증가한다. 실무에서는 트리의 복잡도가 추가적으로 늘어나는 만큼 cp 를 벌점으로 간주해 트리 성장을 제한하는 데 사용된다.*

다소 임의적이라고 할 수 있는 첫 번째 방법은 탐색 작업에 유용할 수 있지만 최적값(새로운 데이터에 대한 예측 정확도를 최대화하기 위한 값)을 결정하기가 매우 어렵다. 복잡도 파라미터 cp를 이용하면 어떤 크기의 트리가 새로운 데이터에 대해 가장 좋은 성능을 보일지 추정할 수 있다.

cp가 매우 작다면 트리는 실제 의미 있는 신호 뿐 아니라 노이즈까지 학습하여 오버피팅되는 문제가 발생하게 될 것이다. 반면에 cp가 너무 크다면 트리가 너무 작아 예측 능력을 거의 갖지 못할 것이다.

rpart 함수의 기본 설정은 0.01로 다소 큰 값이다. 이전 예제에서 기본값을 그대로 사용하면 트리 분할이 단 한 번만 일어나므로, 예제에서는 cp를 0.005로 지정했다. 보통은 사전 탐색 분석에서 몇 가지 값을 테스트해보는 것으로도 충분하다.

최적의 cp를 결정하는 것은 평향=-분산 트레이드 오프를 보여주는 하나의 대표적인 예라고 할 수 있다. cp를 추정하는 가장 일반적인 방법은 교차타당성 검정을 이용하는 것이다.

1. *데이터를 학습용 데이터와 타당성검사용(홀드아웃) 데이터로 나눈다.*
2. *학습 데이터를 이용해 트리를 키운다.*
3. *트리를 단계적으로 계속해서 가지치기를 한다. 매 단계마다 **학습 데이터**를 이용해 cp를 기록한다.*
4. ***타당성검사 데이터**에 대해 최소 에러(손실)을 보이는 cp를 기록한다.*
5. *데이터를 다시 학습용 데이터와 타당성검사용 데이터로 나누고, 마찬가지로 트리를 만들고 가지치기를 하고 cp를 기록하는 과정을 반복한다.*
6. *이를 여러 번 반복한 후 각 트리에서 최소 에러를 보이는 cp 값의 평균을 구한다.*
7. *원래 데이터를 이용해 위에서 구한 cp의 최적값을 가지고 트리를 만든다.*

rpart 함수에서 변수 cptable을 이용해 cp값과 이에 해당하는 교차타당성검사 에러 (변수 xerror)를 보이는 테이블을 얻을 수 있다. 이를 참고해 교차타당성검사 에러가 가장 적은 cp값을 선정할 수 있다.

6.2.5 연속값 예측하기

트리 모델을 이용해 연속값을 예측하는 방법(**회귀분석**)은 위에서 소개한 것과 동일한 논리 구조와 과정을 거친다. 다만 각 하위 분할 영역에서 평균으로부터의 편차들을 제곱한 값을 이용해 불순도를 측정한다는 점과 제곱근 평균제곱오차(RMSE)를 이용해 예측 성능을 평가한다는 점에서 차이가 있다.

6.2.6 트리 활용하기

어떤 조작이든 예측 모델링을 수행하는 담당자달이 겪는 가장 큰 어려움 중 하나는 사용하는 모델이 '블랙박스'와 같은 성질을 갖는다는 것이다. 이는 조직의 다른 부서로부터 반대 의견의 빌미를 제공한다. 이러한 트리 모델은 아래 두 가지 장점을 갖는다.

- **트리 모델은 데이터 탐색을 위한 시각화가 가능하다. 이는 어떤 변수가 중요하고 변수 간에 어떤 관계가 있는지를 보여준다. 트리 모델은 예측변수들 간의 비선형 관계를 담아낼 수 있다.**
- **트리 모델은 일종의 규칙들의 집합(set of rules)이라고 볼 수 있다. 따라서 실제 구현 방법에 대해서, 아니면 데이터 마이닝 프로젝트 홍보에 대해 비전문가들과 대화하는데 아주 효과적이라고 할**

수 있다.

하지만 예측에 관해서는, 다중 트리에서 나온 결과를 이용하는 것이 단일 트리를 이용하는 것보다 보통은 훨씬 강력하다. 특히 랜덤 포레스트나 부스팅 트리 알고리즘은 거의 항상 우수한 예측 정확도나 성능을 보여준다. 물론 앞서 설명한 단일 트리의 장점들을 잃어버린다는 단점이 있다.

주요 개념

- **의사 결정 트리는 결과를 분류하거나 예측하기 위한 일련의 규칙들을 생성한다.**
- **이 규칙들은 데이터를 하위 영역으로 연속적으로 분할하는 것과 관련이 있다.**
- **각 분할 혹은 분기는 어떤 한 예측변수 값을 기준으로 데이터를 두 부분으로 나누는 것이다.**
- **각 단계마다, 트리 알고리즘은 결과의 불순도를 최소화하는 쪽으로 영역 분할을 진행한다.**
- **더 이상 분할이 불가능할 때, 트리가 완전히 자랐다고 볼 수 있으며 각 말단 노드 혹은 앞 노드에 해당하는 레코드들은 단일 클래스에 속한다. 새로운 데이터는 이 규칙 경로를 따라 해당 클래스로 할당된다.**
- **완전히 자란 트리는 데이터를 오버피팅하기 때문에, 노이즈를 제외한 신호에만 반응하도록 트리에 가지치기를 수행해야 한다.**
- **랜덤 포레스트나 부스팅 트리 같은 다중 트리 알고리즘은 우수한 예측 성능을 보장한다. 하지만 규칙에 기반을 둔 단일 트리의 장점이라고 할 수 있는 정보 전달 능력은 잃어버린다.**

6.3 배깅과 랜덤 포레스트

이 원리는 예측 모델에서도 적용된다. 즉 다중 모델의 평균을 취하는 방식(혹은 다수결 투표), 다른 말로 **앙상블** 모델은 단일 모델을 사용하는 것보다 더 나은 성능을 보인다.

용어정리

- **앙상블(ensemble)** : 여러 모델의 집합을 이용해서 하나의 예측을 이끌어내는 방식 (유의어 : 모델 평균화(model averaging)
- **배깅(bagging)** : 데이터를 부트스트래핑해서 여러 모델을 만드는 일반적인 방법(유의어 : 부트스트랩 종합(bootstrap aggregation)
- **랜덤 포레스트(random forest)** : 의사 결정 트리 모델에 기반을 둔 배깅 추정 모델(유의어 : 배깅 의사 결정 트리)
- **변수 중요도(variable importance)** : 모델 성능에 미치는 예측변수의 중요도

앙상블 접근법은 여러 다른 모델링 방법에도 적용되고 있다. 최근 넷플릭스가 진행한 넷플릭스 컨테스트에서 100만 달러 상금을 받은 참가자 역시 넷플릭스 고객들의 영화 평점 예측응ㄹ 10% 정도 향상시키는 데 앙상블 모델을 사용했다. 앙상블 방법의 가장 간단한 버전은 다음과 같다.

1. **주어진 데이터에 대해 예측 모델을 만들고 예측 결과를 기록한다.**
2. **같은 데이터에 대해 여러 모델을 만들고 결과를 기록한다.**
3. **각 레코드에 대해 예측된 결과들의 평균(또는 가중평균, 다수결 투표)을 구한다.**

앙상블 기법은 의사 결정 트리에 체계적이고 효과적으로 적용되어왔다. 앙상블 기법은 상대적으로 적은 노력만으로도 좋은 예측 모델을 만들 수 있다는 점에서 정말 파워플하다.

앞서 소개한 단순한 앙상블 방법 외에, 가장 많이 사용되는 **배깅**과 **부스팅**이라는 앙상블 방법이 있다. 앙상블 기법이 트리 모델에 적용될 경우, **랜덤 포레스트**와 **부스팅 트리**가 각각 이에 해당한다.

이번절에서는 배깅에 대해 알아보자.

6.3.1. 배깅

배깅이란 '부트스트랩 종합(bootstrap aggregating)'의 준말이다. 응답변수 Y와 P개의 예측변수 $X = X_1, X_2, \cdots, X_p$ 로 이뤄진 N개의 레코드가 있다고 가정하자 .

배깅은 다양한 모델들을 정확히 같은 데이터에 대해 구하는 대신, 매번 부트스트랩 재표본에 대해 새로운 모델을 만든다. 이 부분만 빼면 앞에서 설명한 기본 앙상블 방법과 동일하다. 알고리즘은 다음과 같다.

1. 만들 모델의 개수 M과 모델을 만드는 데 사용할 레코드의 개수 $n(n < N)$의 값을 초기화한다. 반복 변수 $m = 1$ 로 놓는다.
2. 훈련 데이터로부터 복원추출 방법으로 n개 부분 데이터 Y_m과 X_m 을 부트스트랩 재표본 추출한다.
3. 의사 결정 규칙 $\hat{f}_m(X)$ 를 얻기 위해, Y_m과 X_m을 이용해 모델을 학습한다.
4. $m = m + 1$ 로 모델 개수를 늘린다. $m \leq M$ 이면 다시 1단계로 간다.

$\hat{f}_m$ 이 $Y = 1$ 인 경우의 확률을 예측한다고 했을 때, 배깅 추정치는 다음과 같이 정의할 수 있다.

$$\hat{f} = \frac{1}{M}(\hat{f}_1(X) + \hat{f}_2(X) + \cdots + \hat{f}_M(X))$$

6.3.2 랜덤 포레스트

랜덤 포레스트는 의사 결정 트리 모델에 한 가지 중요한 요소가 추가된 배깅 방법을 적용한 모델이다. 바로 레코드를 표본추출 할 때, 변수 역시 샘플링하는 것이다. 일반적으로 의사 결정 트리에서는 하위 분할 영역 A를 만들 때, 지니 불순도와 같은 기준값이 최소화되도록 변수와 분할 지점을 결정했다. 랜덤 포레스트에서는 알고리즘의 각 단계 마다, 고를 수 있는 변수가 **랜덤하게 결정된 전체 변수들의 부분집합**에 한정된다. 기본 트리 알고리즘에 비해 랜덤 포레스트에는 앞서 언급한 배깅과 각 분할을 위한 변수의 부트스트랩 샘플링, 이 두 가지 단계가 다음과 같이 추가된다.

1. **전체 데이터로부터 부트스트랩 샘플링 (복원추출) 을 한다.**
2. **첫 분할을 위해 비복원 랜덤표본추출로 $p(p < P)$개의 변수를 샘플링한다.**
3. **샘플링된 변수 $X_{f(1)}, X_{f(2)}, \cdots, X_{f(p)}$ 에 대해 분할 알고리즘을 적용한다.**
    a. **$X_{f(k)}$의 각 변수 $s_{j(k)}$에 대해**
        i . **파티션 A에 있는 레코드들을 $X_{j(k)} < s_{j(k)}$인 하위 영역과 $X_{j(k)} \geq s_{j(k)}$인 하위 영역으로 나눔.**
        ii . **A의 각 하위 영역 내부의 클래스의 동질성을 측정한다.**
    b. **분할 영역 내부의 클래스 동질성을 최대로 하는 $s_{j(k)}$의 값을 선택한다.**
4. **분할 영역 내부의 클래스 동질성을 최대로 하는 $X_{j(k)}$ 와 $s_{j(k)}$ 값을 선택한다.**
5. **다음 분할을 진행하기 위해, 2단계부터 시작해 이전 단계들을 반복한다.**
6. **트리가 모두 자랄 때 까지 위와 같은 분할 과정을 반복한다.**
7. **1단계로 돌아가 또 다른 부트스트랩 표본을 추출해 같은 과정을 반복한다.**

그렇다면 각 단계마다 몇 개 정도의 변수를 샘플링 해야 할까? 보통은 전체 변수의 개수가 P일 때, $\sqrt{P}$ 개 정도를 선택한다. R에는 랜텀 포레스트를 구현해 놓은 randomForest 라는 패키지가 있다. 아래 코드는 대출 데이터를 이 패키지에 적용하는 예를 보여준다.

```
> rf <- randomForest(outcome ~ borrower_score + payment_inc_ratio,
+                    data=loan3000)
> rf
```

```
Call:
 randomForest(formula = outcome ~ borrower_score + payment_inc_ratio,      data
= loan3000)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 1

        OOB estimate of  error rate: 38.87%
Confusion matrix:
         paid off default class.error
paid off      982     573   0.3684887
default       593     852   0.4103806
```

기본적으로 500개의 트리를 학습을 통해 생성한다. 여기서는 예측변수를 두 개만 사용했기 때문에, 알고리즘은 매 단계마다 둘 중 한 변수를 랜덤하게 선택한다. (즉, 부트스트랩 샘플의 크기가 1이다.)

주머니 외부(Out-Of-Bag(OOB)) 추정 에러는 트리 모델을 만들 때 사용했던 학습 데이터에 속하지 않는 데이터를 사용해 구한 학습된 모델의 오차율을 말한다. 모델의 출력을 이용해 랜덤 포레스트에서 트리의 개수에 따른 OOB 에러의 변화를 그려볼 수 있다.

Predict 함수를 이용해 예측값을 구하고 이를 다음과 같이 그래프로 만들 수 있다.

랜덤 포레스트는 일종의 블랙박스 모델이다. 단순한 단일 트리보다 훨씬 정확한 예측 성능을 보이지만 간단한 트리를 통해 얻을 수 있는 객관적인 해석은 불가하다. 예측 결과 역시 다소 지저분하다. 신용점수가 높은 사람 중에도 대출을 다 갚지 못할 것이라는 예측 결과가 나오는 것을 볼 수 있다. 이것은 데이터에서 일반적이지 않은 예외 사항까지 학습해서 생기는 결과로, 랜덤 포레스트에 의한 오버피팅의 위험성을 보여준다.

6.3.3 변수 중요도

랜덤 포레스트는 피처와 레코드의 개수가 많은 데이터에 대해 예측 모델을 만들 때 장점을 발휘한다. 다수의 예측변수 중 어떤 것이 중요한지, 그리고 이들 사이에 존재하는 상관관계 항들에 대응되는 복잡한 관계들을 자동으로 결정하는 능력이 있다. 예를 들어, 대출 데이터에서 모든 변수를 고려한 모델을 찾는다고 하자.

```
> ## Fit random forest to all the data; this can take a while and
> ## may cause problems if you don't have enough RAM
> rf_all <- randomForest(outcome ~ ., data=loan_data, importance=TRUE)
> rf_all

Call:
 randomForest(formula = outcome ~ ., data = loan_data, importance = TRUE)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 4

        OOB estimate of  error rate: 33.96%
Confusion matrix:
         default paid off class.error
default    15123     7548   0.3329364
paid off    7852    14819   0.3463456
```

important = TRUE 설정은 **randomForest** 함수에 다른 변수들의 중요도에 관한 정보를 추가적으로 저장하도록 요청한다. vamImpPlot 함수는 변수들의 상대적인 성능을 그래프를 통해 보여준다.

```
imp1 <- importance(rf_all, type=1)
imp2 <- importance(rf_all, type=2)
idx <- order(imp1[,1])
nms <- factor(row.names(imp1)[idx], levels=row.names(imp1)[idx])
imp <- data.frame(Predictor = rep(nms, 2),
                  Importance = c(imp1[idx, 1], imp2[idx, 1]),
                  Type = rep( c('Accuracy Decrease', 'Gini Decrease'),
rep(nrow(imp1), 2)))
```

```
## Figure 6-8: Variable importance plot for random forest
png(filename=file.path(PSDS_PATH, 'figures', 'psds_0608.png'),  width = 4,
height=6, units='in', res=300)

ggplot(imp) +
  geom_point(aes(y=Predictor, x=Importance), size=2, stat="identity") +
  facet_wrap(~Type, ncol=1, scales="free_x") +
  theme(
    panel.grid.major.x = element_blank() ,
    panel.grid.major.y = element_line(linetype=3, color="darkgray") ) +
  theme_bw()

dev.off()
```

대출 데이터를 가지고 구한 모델에서 변수들의 중요도를 보여준다.

변수 중요도를 측정하는 데에는 두 가지 방법이 있다.

- **변수의 값을 랜덤하게 섞었다면, 모델의 정확도가 감소하는 정도를 측정한다. (type = 1) . 변수를 랜덤하게 섞는 다는 것은 해당 변수가 예측에 미치는 모든 영향력을 제거하는 것을 의미한다. 정확도는 OOB 데이터로부터 얻는다. (결국 교차타당성검사와 같은 효과를 얻는다.)**
- **특정 변수를 기준으로 분할이 일어난 모든 노드에서 불순도 점수와 평균 감소량을 측정한다. 이 지표는 해당 변수가 노드의 순도를 개선하는 데 얼마나 기여했는지를 나타낸다. 물론 이 지표는 학습 데이터를 기반으로 측정되기 때문에, OOB 데이터를 가지고 계산한 것에 비해 믿을 만 하지 못하다.**

위 그림은 각각 정확도와 지니 불순도가 감소하는 정도를 이용해 변수 정확도를 보여준다. 정확도 감소량을 기준으로 변수들의 순위를 결정했다. 두 지표를 통해 알아본 변수 중요도가 서로 다른 것을 볼 수 있다.

정확도 감소량이 좀 더 믿을만한 지표하면 지니 불순도를 굳이 고려할 필요가 있을까? 기본적으로 randomforest 함수는 지니 불순도만을 활용한다. 모델의 정확도는 추가적인 계산(랜덤 순열 조합으로 데이터를 뽑고 이 데이터를 이용해 예측 결과를 구하는 일련의 과정)이 필요한 반면, 지니 불순도는 알고리즘상에서 부차적으로 얻어지는 결과물이다.

예를 들어, 몇 천개의 모델을 만들어야 해서 계산 복잡도가 중요하다면, 프로덕션 상황에서 계산을 더해 얻는 이득이 거의 없을 수 있다.

또한 지니 불순도의 감소는 분할 규칙을 만드는 과정에서 어떤 변수를 사용해야 할지에 대해 큰 역활을 한다.

물론 간단한 트리모델이 갖고 있는 장점인, 쉬운 해석을 랜덤 포레스트에서는 찾아볼 수 없다. 지니 계수 감소와 모델 정확도 감소 사이의 차이를 잘 조사해보면, 변수 중요도를 통해 모델을 개선할 수 있는 몇 가지 아이디어를 얻을 수 있을 것이다.

6.3.4 하이퍼파라미터

다른 여러 머신러닝 알고리즘과 마찬가지로 랜덤 포레스트는 성능을 조절할 수 있는 손잡이가 달린 일정의 블랙박스 알고리즘이라고 할 수 있다. 이러한 손잡이를 **하이퍼파라미터**라고 부르고, 이 파라미터들은 모델을 학습하기 전에 미리 정해야 한다. 이 값들은 학습 과정 중에 최적화되지 않는다. 기존의 통계 모델에서는 이것이 선택 사항이었다면(예를 들어 회귀 모형에서 사용할 예측변수들을 선택하는 것), 랜덤 포레스트에서 하이터파라미터는 좀 더 결정적인 영향을 미치는 중요한 요소이다. 특히 오버피팅을 피하기 위해 매우 중요하다. 랜덤 포레스트에는 아래 두 가지 가장 중요한 하이퍼파라미터들이 존재한다.

- **nodesize : 말단노드(나무에서 앞 부분)의 크기를 의미한다. 분류 문제를 위한 기본 설정은 1이며, 회귀 문제에서는 5이다.**
- **maxnodes : 각 결정 트리에서 전체 노드의 최대 개수를 의미한다.**

물론 이러한 파라미터들을 무시하고 그냥 기본 설정으로 가고 싶은 유혹이 있을 수 있다. 하지만 랜덤 포레스트를 노이즈가 많은 데이터에 적용할 때, 기본 설정으로는 오버피팅에 빠질 수 있다. nodesize 와 maxnode를 크게 하면 더 작은 트리를 얻게 되고 거짓 예측 규칙들을 만드는 것을 피할 수 있다. 하이퍼파리미터에 다른 값들을 적용했을 때의 효과를 알아보려면 교차타당성검사를 이용할 수 있다.

주요개념

- **앙상블 모델은 많은 모델로부터 얻은 결과를 서로 결합해 모델 정확도를 높인다.**
- **배깅은 앙상블 모델 가운데 하나의 형태로, 부트스트랩 샘플을 이용해 많은 모델들을 생성하고 이 모델들을 평균화 한다.**
- **랜덤 포레스트는 매핑 기법을 의사 결정 트리 알고리즘에 적용한 특별한 형태이다. 랜덤 포레스트에서는 데이터를 재표본추출하는 동시에 트리를 분할할 때 예측변수 또한 샘플링한다.**
- **랜덤 포레스트로부터 나오는 출력 중 유용한 것은 예측변수들이 모델 정확도에 미치는 영향력을 의미하는 변수 중요도이다.**
- **랜덤 포레스트에서는 오버피팅을 피하기 위해 교차타당성검사를 통해 조정된 하이퍼파리미터를 사용한다.**

6.4 부스팅

앙상블 모델은 예측 모델링 쪽에서 표준 방법이 되고 있다. **부스팅**은 모델들을 앙상블 형태로 만들기 위한 일반적인 기법이다. 이는 **배깅**과 비슷한 시기에 개발되었다. 배깅과 마찬가지로 부스팅 역시 결정 트리에 주로 가장 많이 사용된다. 이런 비슷한 면도 있지만, 부스팅은 훨씬 많은 부가 기능을 갖고 있는 전혀 다른 방법이다. 배깅은 상대적으로 튜닝이 거의 필요 없지만 부스팅은 적용하고자 하는 문제에 따라 주의가 필요하다. 이 두 방법을 자동차에 비유하자면, 배깅은 (신뢰할 수 있고 안정적인) 혼다 어코드와 같고, 부스팅은 (강력하지만 더 많은 주의가 필요한) 포르쉐와 같다고 할 수 있다.

선형회귀 모델에서는 피팅이 더 개선될 수 있는지 알아보기 위해 잔차를 종종 사용했다. 부스팅은 바로 이러한 개념을 더 발전시켜, 이전 모델이 갖는 오차를 줄이는 방향으로 다음 모델을 연속적으로 생성한다. **에이다부스트, 그레이디언트 부스팅, 확률적 그레이디언트 부스팅**은 가장 자주 사용되는 변형된 형태의 부스팅 알고리즘이다. 이 중에서도 확률적 그레이디언트 부스팅이 일반적으로 가정 널리 사용된다. 실제로 파라미터만 올바르게 잘 선택한다면 이 알고리즘은 랜덤 포레스트를 그대로 에뮬레이션 할 수 있다.

용어 정리

- **앙상블 (ensemble) : 여러 모델들의 집합을 통해 예측 결과를 만들어내는 것(유의어 : 모델 평균화)**

- **부스팅 : 연속된 라운드마다 잔차가 큰 레코드들에 가중치를 높여 일련의 모델들을 생성하는 일반적인 기법**
- **에이다부스트(AdaBoost) : 잔차에 따라 데이터의 가중치를 조절하는 부스팅의 초기 버전**
- **그레이디언트 부스팅 : 비용함수를 최소화하는 방향으로 부스팅을 활용하는 좀 더 일반적인 형태**
- **확률적 그레이디언트 부스팅 : 각 라운드마다 레코드와 열을 재표본추출하는 것을 포함하는 부스팅의 가장 일반적인 형태**
- **정규화 : 비용함수에 모델의 파라미터 개수에 해당하는 벌점 항을 추가해 오버피팅을 피하는 방법**
- **하이퍼파라미터 : 알고리즘을 피팅하기 전에 미리 세팅해야 하는 파라미터**

6.4.1 부스팅 알고리즘

다양한 부스팅 알고리즘의 바탕에 깔려 있는 기본 아이디어는 모두 같다. 가장 이해하기 쉬운 에이다부스트 알고리즘부터 알아보자.

1. **먼저 피팅할 모델의 개수 M을 설정한다. 그리고 반복 횟수를 의미하는 $m = 1$ 로 초기화한다. 관측 가중치 $w_i = 1/N$ 으로 초기화한다($i = 1, 2, \cdots, N$). 앙상블 모델을 $\hat{F}_0 = 0$ 으로 초기화한다.**
2. **관측 가중치 $w_0, w_1, \cdots, w_N$ 을 이용해 모델 $\hat{f}_0$ 을 학습한다. 이때 잘못 분류된 관측치에 대해 가중치를 적용한 합을 의미하는 가중 오차 e_m 이 최소화되도록 학습한다.**
3. **앙상블 모델에 다음 모델을 추가한다. $\hat{F}_m = \hat{F}_{m-1} + \alpha_m \hat{f}_m$ 여기서 $\alpha_m = \frac{\log(1-e_m)}{e_m}$ 이다.**
4. **잘못 분류된 입력 데이터에 대한 가중치를 증가하는 방향으로 가중치 $w_0, w_1, \cdots, w_N$ 을 업데이트한다.**

   α_m 에 따라 증가 폭이 결정되며, α_m 이 클수록 가중치가 더 커진다.
5. **모델 반복 횟수를 $m = m + 1$ 으로 증가시키고 $m \leq M$ 이면 다시 1단계로 돌아간다.**

이 과정을 통해 얻은 부스팅 추정치는 다음과 같다.

$$\hat{F} = \alpha_1 \hat{f}_1 + \alpha_2 \hat{f}_2 + \cdots + \alpha_M \hat{f}_M$$

잘못 분류된 관측 데이터에 가중치를 증가시킴으로써, 현재 성능이 제일 떨어지는 데이터에 대해 더 집중해서 학습을 하도록 하는 효과를 가져온다. α_m값을 이용해 모델의 오차가 낮을 수록 더 큰 가중치를 부여한다.

그레이디언트 부스팅은 에이다부스팅과 거의 비슷하지만, 비용함수를 최적화하는 접근법을 사용했다는 점에서 차이가 있다. 그레이디언트 부스팅에서는 가중치를 조정하는 대신에 모델이 **유사잔차(pseudo-residual)**를 학습하도록 한다. 이는 잔차가 큰 데이터를 더 집중적으로 학습하는 효과를 가져온다. 확률적 그레이디언트 부스팅에서는 랜덤 포레스트와 유사하게, 매 단계마다 데이터와 예측변수를 샘플링하는 식으로 그레이딩언트 부스팅에 랜덤한 요소를 추가한다.

6.4.2 XG부스트

부스팅 방법 가운데 대중적으로 가장 많이 사용되는 오픈소스 SW는 **XG부스트(XGBoost)** 이다. 여러 가지 옵션이 효율적으로 구현되었고, 대부분의 데이터 과학 SW언어를 지원하는 패키지를 제공한다. R에서도 xgboost 패키지를 이용해 XG부스트를 사용할 수 있다.

xgboost 패키지를 이용해 XG부스트를 사용할 수 있다. xgboost 함수는 우리가 직접 조정할 수 있는 다양한 파라미터들을 제공한다. 이 가운데 가장 중요한 파라미터 두 가지는 **subsample과 eta** 이다.

subsample은 각 반복 구간 마다 샘플링할 입력 데이터의 비율을 조정한다.

eta는 부스팅 알고리즘에서 α_m 에 적용되는 축소 비율을 결정한다. subsample 설정에 따라 비복원 추출 한다는 점만 빼면 부스팅은 마치 랜덤 포레스트 같이 동작한다. 축소 파라미터 eta는 가중치의 변화량을 낮춰 오버피팅을 방지하는 효과가 있다. 가중치를 조금씩 변화하는 것은 알고리즘이 학습 데이터에 오버피팅될 가능성을 줄여준다. 다음은 xgboost를 대출 데이터에 적용하는 과정을 보여준다. 다음 예제에서는 두 가지 변수만들 고려한다.

```
predictors <- data.matrix(loan3000[, c('borrower_score', 'payment_inc_ratio')])
label <- as.numeric(loan3000[,'outcome'])-1
xgb <- xgboost(data=predictors, label=label, objective = "binary:logistic",
               params=list(subsample=.63, eta=0.1), nrounds=100)
```

xgboost는 수식이 포함된 문법을 지원하지 않기 때문에 예측변수는 data.matrix 형태로, 예측변수는 0/1 형태로 변형해서 사용해야 한다. objective 파라미터는 문제가 어떤 종류인지 설정하기 위한 것이다. 이에 따라 xgboost 함수를 이용해 예측 결과를 구해왔다. 두 가지 변수만을 고려했기에 다음과 같이 결과 그래프를 그려볼 수 있다.

```
pred <- predict(xgb, newdata=predictors)
xgb_df <- cbind(loan3000, pred_default=pred>.5, prob_default=pred)

## Figure 6-9: prediction from xgboost
png(filename=file.path(PSDS_PATH, 'figures', 'psds_0609.png'), width = 5,
    height=4, units='in', res=300)

ggplot(data=xgb_df, aes(x=borrower_score, y=payment_inc_ratio,
                        color=pred_default, shape=pred_default)) +
  geom_point(alpha=.6, size=2) +
  scale_shape_manual( values=c( 46, 4)) +
  scale_x_continuous(expand=c(.03, 0)) +
  scale_y_continuous(expand=c(0,0), lim=c(0, 20)) +
  theme_bw()

dev.off()
```

위 그림은 예측 결과를 보여준다. 전체적으로 랜덤 포레스트를 이용했을 때의 아래 그림과 예측 결과가 비슷한 것을 볼 수 있다. 예측 결과가 다소 복잡하고, 아직도 신용점수가 높은 경우 인데도 예측 결과가 연체로 나오기도 한다.

6.4.3 정규화 : 오버피팅 피하기

xgboost 함수를 무작정 사용할 경우, 학습 데이터에 **오버피팅**되는 불안정한 모델을 얻을 가능성이 있다. 오버피팅은 다음 두 가지 문제를 일으킬 수 있다.

- **학습 데이터에 없는 새로운 데이터에 대한 모델의 정확도가 떨어진다.**
- **모델의 예측 결과에 변동이 매우 심하고 불안정한 결과를 보인다.**

어떤 모델링 기법이든 오버피팅에 빠지기 쉽다. 예를 들어 회귀방정식에서 너무 많은 변수를 고려하다 보면 결국 모델은 잘못된 예측 결과를 만들어 낼 수 있다. 하지만 대부분의 통계적인 기법들은 예측변수들을 까다롭게 걸러내는 과정을 통해 오버피팅을 피할 수 있다. 심지어 랜덤 포레스트는 별다른 파라미터 튜닝 없이도 상당히 괜찮은 모델을 만들곤 한다. 하지만 xgboost의 경우는 그렇지 않다. 대출 데이터에서 모든 변수를 고려해 xgboost를 학습시켜보자.

```
> ## Create a test and training set and compare the learning rates under
```

```
different hyperparameter choices
> seed <- 400820
> predictors <- data.matrix(loan_data[,-which(names(loan_data) %in% 'outcome')])
> label <- as.numeric(loan_data$outcome)-1
> test_idx <- sample(nrow(loan_data), 10000)
> xgb_default <- xgboost(data=predictors[-test_idx,], label=label[-test_idx],
+                        objective = "binary:logistic", nrounds=250, verbose=0)
> pred_default <- predict(xgb_default, predictors[test_idx,])
> error_default <- abs(label[test_idx] - pred_default) > 0.5
> xgb_default$evaluation_log[250,]
   iter train_error
1:  250     0.13293
> mean(error_default)
[1] 0.3609
```

전체 데이터 가운데 랜덤하게 테스트를 위한 10,000개의 데이터를 선별하고, 나머지 데이터를 학습 데이터로 사용했다. 부스팅 모델을 학습한 결과, 학습 데이터에 대한 오차율은 13.2%였지만, 테스트 데이터에 대한 오차율은 그것보다 훨씬 높은 36.09%였다. 이는 오버피팅의 결과이다. 부스팅 결과는 학습 데이터에 대한 변동성을 잘 설명했지만 이 부스팅 모델의 새로운 데이터에는 잘 맞지 않는 다는 것을 알 수 있다.

앞서 살펴본 것 처럼 subsample이나 eta 같은 파라미터를 통해 부스팅 시 오버피팅을 방지 할 수 있다. 여기에서 소개할 또 다른 방법은 **정규화** 방법이다. 이는 모델의 복잡도에 따라 벌점을 추가하는 형태로, 비용함수를 변경하는 방법이다. 의사 결정 트리에서는 지니 불순도와 같은 비용 기준값을 최소화하는 쪽으로 모델을 피팅했다. xgboost에서는 모델의 복잡도를 의미하는 항을 추가하는 형태로 비용함수를 변경할 수 있다.

xgboost에는 모델을 정규화하기 위한 두 파라미터 alpha와 lambda가 존재한다. 이는 각각 맨하튼 거리와 유클리드 거리를 의미한다. 이 파라미터들을 크게 하면, 모델이 복잡해질수록 더 많은 벌점을 부여하고 결과적으로 트리의 크기는 작아진다. 예를 들어 만약 lambda의 값을 1,000으로 하면 무슨 일이 벌어지는지 알아보자.

```
> xgb_penalty <- xgboost(data=predictors[-test_idx,],
+                        label=label[-test_idx],
+                        params=list(eta=.1, subsample=.63, lambda=1000),
+                        objective = "binary:logistic", nrounds=250, verbose=0)
> pred_penalty <- predict(xgb_penalty, predictors[test_idx,])
> error_penalty <- abs(label[test_idx] - pred_penalty) > 0.5
> xgb_penalty$evaluation_log[250,]
   iter train_error
1:  250    0.308783
> mean(error_penalty)
[1] 0.3351
```

predict 함수는 좀 더 편리한 파라미터 ntreelimit 를 제공한다. 이는 예측을 위해 첫 i개의 트리 모델만을 사용하는 것을 가능하게 한다. 이를 통해 예측을 위해 사용하는 모델의 개수에 따른 표본 내 오차율과 표본 밖 오차율을 더 쉽게 비교할 수 있다.

```
error_default <- rep(0, 250)
error_penalty <- rep(0, 250)
for(i in 1:250)
{
  pred_default <- predict(xgb_default, predictors[test_idx,], ntreelimit = i)
  error_default[i] <- mean(abs(label[test_idx] - pred_default) >= 0.5)
  pred_penalty <- predict(xgb_penalty, predictors[test_idx,], ntreelimit = i)
  error_penalty[i] <- mean(abs(label[test_idx] - pred_penalty) >= 0.5)
}
```

모델 출력에서 xgb_default$evaluation_log은 학습 데이터에 대한 오차를 알려준다. 이를 표본 밖 오차와 결합하여 반복 횟수에 따른 오차율에 대한 그래프를 구할 수 있다.

```
errors <- rbind(xgb_default$evaluation_log,
                xgb_penalty$evaluation_log,
                data.frame(iter=1:250, train_error=error_default),
                data.frame(iter=1:250, train_error=error_penalty))
errors$type <- rep(c('default train', 'penalty train',
                     'default test', 'penalty test'), rep(250, 4))

## Figure 6-10: learning rates for different choices of hyperparameters
png(filename=file.path(PSDS_PATH, 'figures', 'psds_0610.png'), width = 6,
    height=4, units='in', res=300)

ggplot(errors, aes(x=iter, y=train_error, group=type)) +
  geom_line(aes(linetype=type, color=type), size=1) +
  scale_linetype_manual(values=c('solid', 'dashed', 'dotted', 'longdash')) +
  theme_bw() +
  theme(legend.key.width = unit(1.5,"cm")) +
  labs(x="Iterations", y="Error") +
  guides(colour = guide_legend(override.aes = list(size=1)))

dev.off()
```

이에 대한 결과는 다음과 같다. 기본 모형은 정확도가 학습 데이터에 대해서는 꾸준히 좋아지지만 테스트 데이터에 대해서는 나빠진다. 벌점을 추가한 모형은 그렇지 않다.

기존의 XG부스트와 정규화를 적용한 XG부스트의 오차율

능형회귀와 라소회귀

모델의 복잡도에 따라 벌점을 부여하는 방식이 오버피팅을 방지하는 효과가 있다는 것이 밝혀진다는 것은 1970년대로 거슬로 올라간다. 최소제곱회귀에서는 잔차제곱합(RSS)을 최소화한다는 것을 이미 앞(4.1.3절)에서 다뤘다. **능형회귀**에서는 잔차제곱합에 회귀계수의 개수와 크기에 따라 벌점을 추가한 값을 최소화한다.

$$\sum_{i=1}^{n}(Y_i - \hat{b}_0 - \hat{b}_1 X_i - \cdots - \hat{b}X_p)^2 + \lambda(\hat{b}_1^2 + \cdots + \hat{b}_p^2)$$

λ 는 계수에 대해 어느 정도 벌점을 부여할 것인가를 결정한다. 이 값이 클수록 모델이 데이터에 오버피팅할 가능성이 낮아진다. **라소(Lasso)** 회귀 역시 이와 비슷한데, 다만 벌점 항에 유클리드 거리 대신 맨하탄 거리를 이용한다.

$$\sum_{i=1}^{n}(Y_i - \hat{b}_0 - \hat{b}_1 X_i - \cdots - \hat{b}X_p)^2 + \alpha(|\hat{b}_1^2| + \cdots + |\hat{b}_p^2|)$$

xgboost 함수의 파라미터 lambda와 alpha가 이 같은 방식으로 동작한다.

6.4.4 하이퍼파라미터와 교차타당성검사

xgboost 함수에는 결정이 쉽지 않은 여러 하이퍼파라미터들이 존재한다. 파라미터들에 대한 자세한 소개는 아래 'XG부스트의 하이퍼파라미터' 박스를 참고하자. 이전 절에서 살펴봤듯이 경우에 따라 특정 파라미터 선정이 모델 성능에 큰 차이를 가져온다. 동시에, 설정해야 하는 파라미터 수가 많아지면 어떤 가지고 이 파라미터들을 골라야 할까? 여기에 대한 해답은 바로 **교차타당성검사**를 활용하는 것이다. 교차타당성검사를 위해 일단 데이터를 K개의 서로 다른 그룹(**폴드**)으로 랜덤하게 나눈다. 각 폴드마다 해당 폴드에 속한 데이터를 제외한 나머지 데이터를 가지고 모델을 학습한 후, 폴드에 속한 데이터를 이용해 모델을 평가한다. **이는 결국 표본 밖 데이터에 대한 모델의 성능을 보여준다.** 설정한 하이퍼파라미터 조합마다 폴드에 대한 오차의 평균을 계산해서 전체적으로 가장 낮은 평균 오차를 갖는 최적의 하이퍼파라미터 조합을 찾는다.

이 방법을 xgboost 함수의 파라미터를 찾는데 이용해보자. 이 예제에서 축소 파라미터 **eta**와 트리의 최대 깊이 max_depth, 이 두 파라미터를 최적화해보겠다. max_depth 파라미터는 트리의 뿌리에서 잎 노드까지 최대 깊이를 의미하고 기본값은 6이다. 이 값은 오버피팅을 조절하는 또 다른 방법을 제시한다. 먼저 폴드들과 테스트할 파라미터 값들의 리스트를 설정한다.

```
## Cross validation
N <- nrow(loan_data)
fold_number <- sample(1:5, N, replace = TRUE)
params <- data.frame(eta = rep(c(.1, .5, .9), 3),
                     max_depth = rep(c(3, 6, 12), rep(3,3)))
```

이제 5개의 폴드를 이용해 각 폴드마다 각 모델에 대한 오차를 계산한다.

```
rf_list <- vector('list', 9)
error <- matrix(0, nrow=9, ncol=5)
for(i in 1:nrow(params)){
  for(k in 1:5){
    cat('Fold', k, 'for model', i, '\n')
    fold_idx <- (1:N)[fold_number == k]
    xgb <- xgboost(data=predictors[-fold_idx,], label=label[-fold_idx],
                   params = list(eta = params[i, 'eta'],
                                 max_depth = params[i, 'max_depth']),
                   objective = "binary:logistic", nrounds=100, verbose=0)
    pred <- predict(xgb, predictors[fold_idx,])
    error[i, k] <- mean(abs(label[fold_idx] - pred) >= 0.5)
  }
}
```

전체 45개의 모델을 얻어야 하기에 시간이 꽤 오래 걸릴 수 있다. 오차는 각 모델을 의미하는 행과 각 폴드를 의미하는 열로 이뤄진 행렬 형태로 저장되었다. 이제 rowMeans 함수를 이용하면 다른 파라미터 조합 간의 오차율을 비교할 수 있다.

```
> avg_error <- 100 * round(rowMeans(error), 4)
> cbind(params, avg_error)
  eta max_depth avg_error
1 0.1         3     33.07
2 0.5         3     33.56
3 0.9         3     34.62
4 0.1         6     33.16
5 0.5         6     35.60
6 0.9         6     37.64
7 0.1        12     34.66
8 0.5        12     37.03
9 0.9        12     38.38
```

교차타당성검사를 통해 eta 값이 작으면서 깊이가 얕은 트리를 사용하는 것이 좀 더 정확한 성능을 보인다는 사실을 알게 됐다. 이러한 모델이 좀 더 안정적이기에, 최적의 파라미터는 eta=0.1이고 max_depth=3 (또는 max_depth =6)이라고 할 수 있다.

XG부스트의 하이퍼파라미터

xgboost 함수의 하이퍼파라미터들은 주로 오버피팅, 정확도, 계산 복잡도 사이의 균형을 잡기 위해 사용된다.

- *eta : 부스팅 알고리즘에서 α 에 적용되는 0과 1 사이의 축소인자(shrinkage Factor), 기본값은 0.3 이지만 노이 즈가 있는 데이터에 대해서는 더 작은 값을 추천한다. (예를 들어 0.1)*
- *nround : 부스팅 라운드 횟수, eta가 작은 값이라면 알고리즘의 학습 속도가 늦춰지기 때문에 라운드 횟수를 늘 려야 한다. 오버피팅을 방지하는 파라미터 설정이 포함된 경우, 라운드 횟수를 좀 더 늘려도 괜찮다.*
- *max_depth : 트리의 최대 깊이(기본값은 6). 깊이가 아주 깊은 트리를 만드는 랜덤 포레스트와는 반대로 부스팅 트리의 깊이는 얕다. 이는 노이즈가 많은 데이터에 대해 모델이 복잡한 모델 거짓 상호작용을 회피하는 데 도움이 된다.*
- *subsample 및 colsample_bytree : 전체 데이터에서 일부 데이터를 비복원 샘플링하는 비율 및 예측변수 중 일부 변수를 샘플링하는 비율. 이는 랜덤 포레스트에서 오버피팅을 피하기 위해 사용했던 것들과 유사하다.*
- *lambda 및 alpha : 오버피팅을 조절하기 위해 사용되는 정규화 파라미터들(6.4.3 절 참고)*

주요 개념

- **부스팅 방법은 일련의 모델들을 피팅할 때, 이전 라운드에서 오차가 컸던 레코드들에 가중치를 더하는 방식을 사용하는 앙상블 모델의 한 부류이다.**
- **확률적 그레이디언트 부스팅은 부스팅은 부스팅 가운데 가장 일반적으로 사용되며 가장 높은 성능을 보인다. 확률적 그레이디언트 부스팅의 가장 일반적인 형태는 트리 모델을 사용한다.**
- **XG부스트는 확률적 그레이디언트 부스팅을 사용하기 위한 가장 유명한 SW 패키지이다. 데이터 과학에서 활용되는 거의 대부분의 언어를 지원한다.**
- **부스팅은 데이터에 오버피팅되기 쉽다. 이를 피하기 위해 하이퍼파라미터를 잘 설정해야 한다.**
- **정규화는 파라미터의 개수(예를 들어: 트리의 크기)에 관한 벌점 항목을 모델링에 포함하여 오버피팅을 피하는 방법이다.**
- **부스팅에서 여러 개의 하이퍼파라미터들의 조합을 찾아야 할 때, 교차타당성검사는 아주 중요하다.**

6.5 마치며

이번 장에서는 전체 데이터에 맞는 형태가 딱 정해진 모델(예를 들면 선형회귀) 보다는 데이터에 따라 유연하면서 지역적으로 학습하는 두 가지 분류와 예측 방법을 다뤘다. K최근접이웃(KNN) 방법은 해당 레코드와 비슷한 주변 데이터를 찾아보고 주변 데이터들이 가장 많이 속한 클래스 (혹은 평균)을 찾아 그것을 해당 레코드에 대한 예측값으로 할당하는 아주 간단한 방법이다.

트리 모델에서는 여러 가지 가능한 예측변수의 (분할) 값들을 기준으로 나눠보고 분할 영역의 클래스에 대한 동질성이 가장 많이 증가하는 방향으로 데이터를 분할한다. 가장 효과적인 분할값들은 분류 혹은 예측에 대한 하나의 길 혹은 규칙을 이룬다. 트리 모델은 다른 방법들보다 자주 우수한 성능을 보이는 아주 강력한 예측 기법이다. 또한 다양한 앙상블 방법(랜덤 포레스트, 부스팅, 배깅) 은 트리의 예측 능력을 한층 더 강력하게 한다.