5. 기계학습과 인공신경망

5.1. 기계학습

기계학습(machine learning)이란 데이터로부터 패턴을 찾아내는 컴퓨터 과학의 한 분야이다.

5.1.1. 모형

모형(model)이란 데이터의 패턴을 일정한 수학적 형태로 표현한 것이다. 기계학습은 이러한 모형을 바탕으로 이뤄진다. 모형에 따라 잘 학습할 수 있는 패턴과 그렇지 않은 패턴이 다르기 때문에 데이터의 종류에 따라 서로 다른 모형을 적용하거나 여러 가지 모형을 테스트 해보고 가장 성능이 좋은 모형을 사용하게 된다.

지도 학습에서 널리 사용되는 기계 학습 모형에는 선형 모형, 최근접 이웃, 인공 신경망, 의사 결정 나무 등이 있다.

선형 모형(linear model)은 데이터의 패턴이 직선적인 형태를 가지고 있다는 가정에 바탕을 둔다. 단순한 모형이지만 결과가 안정적이고 해석이 쉽기 때문에 널리 쓰인다.

최근접 이웃(nearest neighbor)은 새로운 사례가 입력으로 들어왔을 때 기존의 사례 중에서 가장 유사한(최근접) 사례들(이웃)을 찾아 이들의 평균을 내거나 다수결하여 출력한다. 간단하지만 정확한 방법이다. 다만 기존의 사례가 너무 많거나 복잡한 데이터의 경우에는 사용하기 어렵다는 단점이 있다.

인공 신경망과 의사 결정 나무는 연구와 응용 양쪽에서 최근 가장 활발한 모형이다. 인공 신경망(artificial neural network)은 생물의 신경망, 인간의 두뇌와 같은 것을 모방하는 방법이다. 학습이 느리고 많은 데이터가 필요하다는 단점이 있었으나 빅데이터의 등장과 컴퓨터 속도의 향상으로 최근 각광을 받게 되었다. 흔히 말하는 딥러닝(deep learning)이 인공 신경망에 바탕을 둔 것이다. 인공 신경망은 특히 언어, 소리, 이미지 같은 비정형(unstructured) 데이터에 강점이 있다.

의사 결정 나무(decision tree)는 스무고개와 같은 식으로 예/아니오로 답할 수 있는 질문들을 거쳐 예측을 할 수 있는 방법이다. 데이터를 바탕으로 이러한 질문의 종류와 순서를 결정하는데 정확성이 매우 높다. 최근의 경향은 수 백 개의 의사 결정 나무들을 합쳐서 예측 능력을 높이는 앙상블(ensemble) 방법을 사용한다. 의사 결정 나무는 표의 형태로 나타낼 수 있는 정형(structured) 데이터에 강점이 있다.

5.1.2. 손실 함수

손실 함수(loss function) 또는 비용 함수(cost function)은 모형에 학습된 패턴이 실제 데이터와 얼마나 떨어졌는지를 측정하는 함수이다. 기계학습은 손실을 최소화하도록 학습된 패턴

조정하는 방식으로 이뤄진다. 손실 함수의 대표적인 예는 평균제곱오차(mean squared error: MSE)이다. MSE는 모든 사례에 대해 실제값(y)과 모형에 의한 예측값($\hat{y}$)의 오차를 제곱해서 평균낸 것이다.

$$\frac{1}{N}\sum_i (y_i - \hat{y}_i)^2$$

5.1.3. 최적화

최적화(optimization)란 주어진 제약 조건 아래서 목적 함수의 값을 최대화 또는 최소화하는 것을 말한다. 예를 들어 초콜렛이 10개 사탕이 15개 있다고 하자. 이때 다음과 같이 2종류의 세트가 있다.

세트	구성	가격
A	초콜렛 1개, 사탕 1개	1,000
B	초콜렛 1개, 사탕 2개	1,500

그렇다면 세트 A와 세트 B를 각각 몇 개씩 만들어야 가장 많은 돈을 벌 수 있겠는가?

이 문제를 수식으로 바꿔보면 아래와 같다.

$$\begin{array}{ll} \max & 1000a + 1500b \\ \text{subject to} & a + b \leq 10 \quad \text{초콜렛} \\ & a + 2b \leq 15 \quad \text{사탕} \end{array}$$

여기서 첫번째 식이 목적 함수, 이하의 내용이 제약조건이 된다.

문제의 종류는 다르지만 기계학습도 주어진 모형과 데이터 아래서 손실 함수를 최소화하는 것이므로 최적화 문제로 바꿔서 풀 수 있다.

대부분의 기계학습 문제는 문제를 넣으면 바로 답이 나오는 공식이 없기 때문에 점진적으로 해를 개선해나가는 방식의 최적화 알고리즘들이 널리 쓰인다.

5.1.4. 기계학습의 종류

컴퓨터는 입력된 자료를 바탕으로 계산한 결과를 출력한다. 출력되는 결과의 성격에 따라 기계학습은 지도 학습, 비지도 학습, 강화 학습로 구분된다.

지도 학습

지도 학습(supervised learning)은 출력되는 결과의 올바른 답을 알고 있는 문제이다. 예를 들어 X레이 사진에서 암을 진단하는 문제의 경우 입력은 X레이 사진, 출력은 암 진단(양성/음성)이 된다. 이 경우 X레이 사진과 각 사진의 진단이 있으면 컴퓨터에 이 두 가지를 넣고 학습시켜서 새로운 X레이 사진을 입력으로 주었을 때 암을 진단하게 할 수 있다.

지도 학습은 기계 학습에서 가장 널리 다뤄지는 문제이다. 기계 번역, 추천 시스템, 이미지 인식 등이 모두 지도 학습에 속한다.

지도 학습은 출력의 형태에 따라 회귀(regression)와 분류(classification)로 나뉜다. 회귀는 연속적인 값(예: 온도, 나이, 가격, 만족도 등)을, 분류는 이산적인 값(예: 성별, 물체의 종류, 질병 진단 등)을 출력한다.

지도학습을 비스지스에 활용하는 사례로는 다음과 같은 것들이 있다.

- 제품의 특성, 가격 등으로 판매량 예측
- 채무자의 직업, 소득 등으로 상환 가능성 예측
- 피부 손상의 사진으로 상처가 치료될 때까지 기간을 예측
- 이메일의 내용으로 스팸 여부 판단
- 건물의 여러 가지 요소를 바탕으로 화재 위험 진단

비지도 학습

비지도 학습(unsupervised learning)은 출력되야할 결과의 답을 모르는 문제이다. 예를 들어 고객들을 비슷한 몇 가지 집단으로 나눠서 마케팅을 하고자 할 때 어떤 고객이 어떤 집단에 속하게 될지는 실제로 나눠보기 전에는 알 수 없다. 이런 비지도 학습에서는 '비슷함'만 정의하고 컴퓨터에게 알아서 고객을 나누게 한다. 컴퓨테에게 정답을 알려주지 않기 때문에 '비지도'학습이라고 한다.

비지도 학습도 출력의 형태에 따라 차원 축소(dimensionality reduction)와 군집(clustering)으로 나눈다. 차원 축소는 연속적인 값을, 군집은 이산적인 값을 출력한다. 앞의 고객들을 집단으로 나누는 경우가 군집에 해당한다.

차원 축소의 조금 낯선 개념일 수도 있는데, 우리가 익숙한 예가 있다. 시험의 '총점'이다. '총점'은 여러 과목의 점수를 더해서 하나의 점수로 만든다. 세 과목의 점수를 더 해 하나의 총점으로 만들면 3차원에서 1차원으로 줄어드는 것과 같다. 그래서 차원 '축소'라고 하는 것이다. 기계학습에서 차원 축소는 단순히 총점을 내는 것보다는 훨씬 복잡한 방법이지만 기본적인 개념은 동일하다.

비지도 학습은 비즈니스에 직접적으로 활용하는 경우는 지도학습보다 적다. 마케팅에서 고객들을 군집하거나, 추천시스템에서 활용되기도 한다.

비지도 학습은 지도 학습의 전처리 과정으로 활용되기도 한다. 차원을 축소하면 입력의 크기가 줄어들기 때문에 지도 학습을 할 때 패턴을 파악하기가 더 쉬워질 수 있기 때문이다.

비지도학습을 비즈니스에 활용하는 사례로는 다음과 같은 것들이 있다.

- 소비패턴이 비슷한 고객들끼리 군집화
- 고객과 시청패턴이 비슷한 다른 고객들이 즐겨보는 영화를 추천 (추천시스템)
- 게시판 등에서 고객 의견을 비슷한 것끼리 묶어서 정리

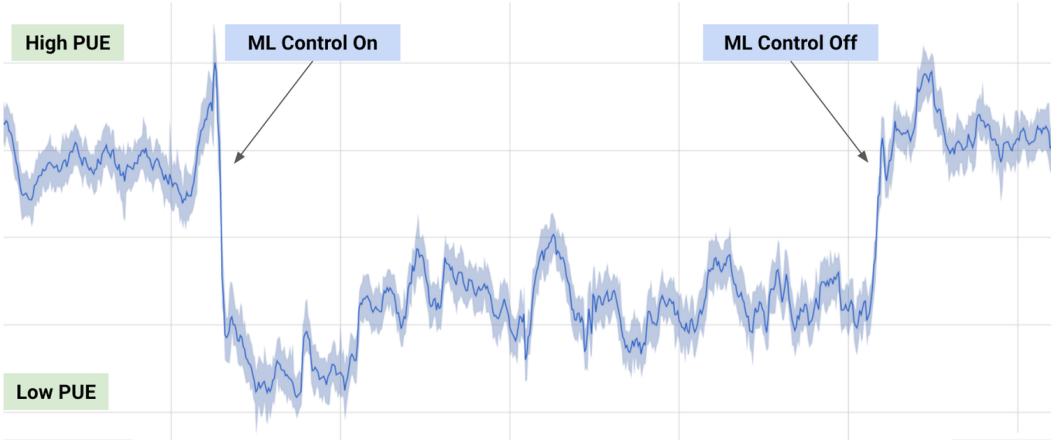
강화 학습

강화 학습(reinforcement learning)은 원래 심리학 용어로 동물에게 보상과 처벌을 통해 학습을 시키는 것을 말한다. 기계학습에서 강화 학습은 이러한 심리학적 과정을 컴퓨터로 흉내낸 것이다.

강화학습은 지도 학습과 비슷하지만 정답 대신 그 답이 얼마나 '좋은 지'만 아는 경우에 활용한다. 예를 들어 인터넷 쇼핑몰에 똑같은 버튼에 "지금 구매하세요"와 "주문하기" 두 가지 문구를 쓸 수 있다고 해보자. 어떤 버튼이 더 좋은지는 처음부터 알 수 없다. 하지만 고객들에게 버튼을 보여주었을 때 고객들이 그 버튼을 누른다면 그 버튼은 '좋다'고 말할 수 있다. 강화학습을 적용하면 컴퓨터가 시행착오를 거쳐 더 나은 문구의 버튼을 선택하게 할 수 있다.

쇼핑몰의 버튼은 한 단계로 끝나는 문제지만 여러 단계의 문제에도 강화학습을 적용할 수 있다. 바둑의 경우 '신의 한 수'가 무엇인지는 알 수 없다. 그러나 한 국의 바둑이 끝났을 때 승패는 바둑의 규칙을 아는 사람이라면 누구나 알 수 있다. 바둑에 강화학습을 적용하면 컴퓨터가 승패로부터 수를 되짚어서 '신의 한 수'에 가까운 수를 스스로 터득할 수 있게 된다. 구글의 알파고(AlphaGo)는 이러한 방식으로 바둑을 깨우쳐 세계 정상급 바둑기사인 한국의 이세돌과 중국의 커제를 상대로 압도적인 승리를 거두었다.

강화학습을 비즈니스에 활용한 사례는 아직 많지 않다. 알파고를 만든 구글 딥마인드는 2016년 강화학습을 통해 구글 데이터 센터의 냉방 비용을 40% 절감했다고 밝혔다.



앞으로 강화학습은 자율 주행 자동차, 실시간 트레이딩 등에 활용될 수 있을 것으로 기대된다.

5.1.5. 기계학습의 과정

기계 학습의 과정은 다음과 같은 순서로 진행된다.

- 데이터 분할
- 다양한 모형과 하이퍼 파라미터로 훈련
- 테스트
- 모형 선택

데이터 분할은 데이터를 훈련용(training)과 테스트용(test)으로 나누는 것이다. 컴퓨터가 데이터를 모두 '외워' 버리면 우리가 가진 데이터에서는 성능이 높아 보일 수도 있으나 실제에 적

용하면 성능이 떨어질 수 있다. 따라서 컴퓨터가 단순히 데이터를 '외웠'는지 아니면 데이터의 패턴을 잘 깨우쳤는지 구별하기 위해 데이터 중 일부를 테스트용으로 나눠놓는 것이다.

데이터를 나눈 후에는 다양한 모형에 학습 또는 훈련을 시킨다. 대부분의 기계학습 모형들은 모형의 특성을 조절하는 하이퍼 파라미터(hyperparameter)를 가지고 있다. 같은 모형이라도 하이퍼파라미터에 따라 성격이 달라진다. 따라서 다양한 모형과 또 모형마다 다양한 하이퍼파라미터로 훈련을 시킨다.

이렇게 다양한 모형/하이퍼파라미터를 시도하는 이유는 실제로 학습을 시켜보기 전에는 어떤 것이 잘 작동할지 알 수 없기 때문이다. 모형이 잘 작동하려면 모형의 형태와 데이터가 가진 패턴의 형태가 맞아야 한다. 그런데 데이터가 가진 패턴의 형태는 매우 복잡해서 우리가 파악하기 어렵기 때문에 일단 학습을 시켜보고 잘 작동하는 모형을 고른다.

테스트는 말 그대로 앞에서 훈련시킨 다양한 모형들을 테스트 데이터로 테스트 하는 것이다.

모형 선택은 테스트에서 성능이 가장 좋은 모형을 선택한다. 여기서는 여러 가지 비즈니스적인 고려가 들어가기도 한다. 예를 들면 예측 자체는 잘하지만 계산이 오래 걸리는 모형보다 조금 예측이 부정확하더라도 계산이 빠른 모형을 선택할 수도 있다.

5.1.6. 기계학습과 통계학의 차이

기계학습과 통계학은 모두 데이터에서 패턴을 발견하는 데 초점을 맞춘 학문이다. 둘의 내용은 거의 비슷하지만 접근 방식에 약간의 차이가 있다. 간단히 말하자면 통계학은 수학 또는 과학의 하위 분과이고, 기계학습은 공학의 하위 분과라고 할 수 있다.

현대적인 통계학은 19세기 후반에서 20세기 초반에 성립한 수학의 응용 분야이다. 주로 데이터의 패턴을 **해석**하고 인과관계를 이해하는데 많은 비중을 두고 발전해왔다.

기계학습은 20세기 후반에 발전한 인공지능의 한 분야로 통계학에서 많은 영향을 받았다. 혹자는 **통계적 학습**(statistical learning)이라고 부르기도 한다. 기계학습은 통계학을 인공지능을 만들기 위한 수단으로서 활용하기 때문에 데이터에 나타낸 패턴을 해석하거나 인과 관계를 이해하는데는 큰 비중을 두지 않는다.

5.2. 감정 분석

감정 분석(sentiment analysis)은 텍스트에 나타난 긍/부정 형태의 감정을 분석하는 방법이다. 감정 분석을 할 수 있으면 단어의 빈도를 넘어 어떤 사안에 대한 사람들의 의견과 감정을 분석할 수 있게 된다.

5.2.1. 감정 분석 방법

감정 분석에는 크게 사전에 의한 방법과 기계 학습에 의한 방법 2가지가 있다.

단어 중에는 긍/부정의 감정을 나타내는 단어들이 있다. 이런 단어를 감정 단어라 한다. 사전에 의한 방법은 전문가가 감정 단어를 수집하여 감정 사전을 만들고, 이 사전을 이용해 텍스트

의 감정을 분석하는 것이다. 예를 들어 낮은 가격을 나타낼 때 '합리적 가격'이라고 하면 긍정적 표현이지만, '싸구려'는 부정적 표현이다. 이와 같은 표현들을 수집하여 사전으로 만든다. 감정 사전을 만들기 위해서는 높은 전문성과 많은 노력이 필요하다.

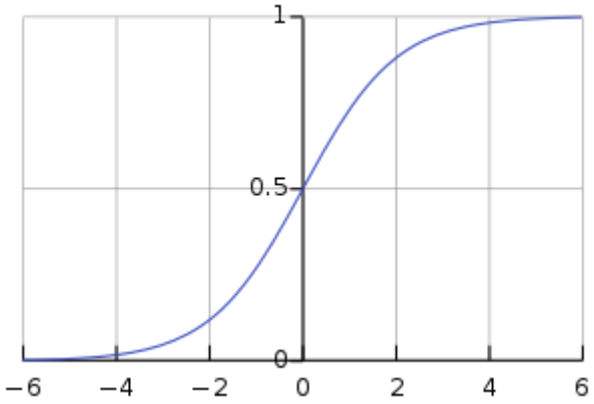
기계학습에 의한 방법은 미리 긍/부정으로 분류된 문장들을 수집하여 기계학습 모형에 학습시킨다. 그리고, 새로운 텍스트에 학습된 모형을 적용하여 긍/부정을 예측한다. 기계학습에 의한 방법은 대량의 데이터가 필요하다는 단점이 있다.

5.3. 로지스틱 회귀

5.3.1. 로지스틱 함수

로지스틱 함수는 다음과 같은 형태의 함수이다.

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$



로지스틱 함수는 $(-\infty, \infty)$ 범위의 y를 $(0, 1)$까지 범위로 찌그러트린다(squash). 그래서 스쿼시 함수라고도 하고 S자 형태로 생겼다 해서 시그모이드(sigmoid) 함수라고도 한다. 확률이 $[0, 1]$ 범위의 값이기 때문에 확률로 해석할 수 있게 된다.

참고로 범위를 나타낼 때 둥근 괄호는 범위의 양쪽 끝을 포함하지 않는다는 의미이고, 각 괄호는 범위의 양쪽 끝을 포함한다는 의미이다.

시그모이드가 S자 형태로 생겼다는 뜻인 이유는 '오이드(-oid)'가 "~의 형태로 생긴"이라는 뜻의 어미이기 때문이다. '시그마(sigma)'는 라틴 문자 S에 해당하는 그리스 문자를 가리킨다. 인간형 로봇을 휴머노이드(humanoid)라고 한다.

5.3.2. 로지스틱 회귀

로지스틱 회귀는 선형 모형과 로지스틱 함수를 결합시킨 형태다. 이름은 '회귀'이지만 실제로는 분류에 쓰이는 모형이다. 독립변수(x)에 일정한 가중치(w)를 곱하는 방식이고, 출력이 $(0, 1)$ 사이의 확률 형태이므로 해석이 간편한다는 장점이 있다.

$$\sigma(wx + b)$$

5.3.3. 크로스 엔트로피

그렇다면 로지스틱 회귀분석의 손실 함수는 어떻게 정의할까? 가장 대표적인 방법은 최대우도법(maximum likelihood)을 이용한다. 우도(likelihood)란 모형의 파라미터를 정했을 때, 학습용 데이터의 확률을 말한다. 최대우도법은 이 우도를 가장 크게 만드는 파라미터를 찾는 방법이다.

로지스틱 회귀 분석은 각각의 x에 대해 한 클래스에 속할 확률 p를 알려주므로 그 확률들을 모두 곱하면 우도를 구할 수 있다.

$$\prod_k \left[\sigma^y (1-\sigma)^{1-y}\right]$$

그런데 확률의 곱셈은 번거로우므로 로그(log)를 사용해서 덧셈으로 바꿔서 **로그 우도**를 사용한다. 마이너스를 붙인 이유는 로그 우도를 최소화했을 때 확률이 최대화되도록 만들기 위해서이다.

$$-\sum_k \left[y \log \sigma + (1-y) \log (1-\sigma)\right]$$

- y: k번째 데이터의 클래스(0 또는 1)
- σ: k번째 데이터의 클래스가 1일 확률

위의 로그 우도는 정보 이론의 크로스 엔트로피와 형태가 같기 때문에 크로스 엔트로피라고도 한다.

5.4. 로지스틱 회귀를 이용한 감정 분석

5.4.1. 데이터 불러오기

```
from sklearn.externals import joblib
```

```
with open('amazon.pkl', 'rb') as f:
    data = joblib.load(f)
locals().update(data)
```

`tdm` 을 확인해보자.

```
tdm
```

```
<1000x1000 sparse matrix of type '<class 'numpy.float64'>'
        with 4060 stored elements in Compressed Sparse Row format>
```

5.4.2. 데이터 분할

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(tdm, sentiment, test_size=.2, rand
```

5.4.3. Logistic Regression 모형 만들기

교차검증이 포함되어있는 `LogisticRegressionCV` 를 불러오자. `LogisticRegressionCV` 는 자동으로 최적의 파라미터를 찾아준다.

```
from sklearn.linear_model import LogisticRegressionCV
```

`model` 에 로지스틱 회귀분석을 불러온다.

```
model = LogisticRegressionCV()
```

`.fit()` 으로 `tdm` 과 점수를 나타내는 `score` 를 학습시킨다.

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

```
LogisticRegressionCV(Cs=10, class_weight=None, cv=None, dual=False,
           fit_intercept=True, intercept_scaling=1.0, max_iter=100,
           multi_class='ovr', n_jobs=1, penalty='l2', random_state=None,
           refit=True, scoring=None, solver='lbfgs', tol=0.0001, verbose=0)
```

`.score()` 으로 정확도를 확인할 수 있다.

```
model.score(X_train, y_train)
```

```
0.9625
```

테스트 데이터로 정확도를 확인해보자

```
model.score(X_test, y_test)
```

```
0.785
```

훈련 데이터에 비해 더 낮은 성능을 보인다.

5.4.4. 감정사전 만들기

회귀계수 확인

`model.coef_` 로 회귀계수를 확인할 수 있다.

첫 10개 단어의 회귀계수를 확인해보자.

```
model.coef_.shape
```

```
(1, 1000)
```

```
model.coef_[0, :10]
```

```
array([ 0.3980101 ,  0.40024896, -0.19642719, -0.34152841,  0.        ,
        0.19202989, -0.33896717,  0.23679641,  0.        ,  0.        ])
```

```
import pandas as pd
```

`zip()` 를 사용하면 두개의 리스트 값을 짝지을 수 있다.

```
sent_df = pd.DataFrame({'단어': vectorizer.get_feature_names(),
                        '계수': model.coef_.flat})
```

```
sd.tail()
```

	계수	단어
995	-0.251266	wrongly
996	0.047629	year
997	1.441947	years
998	-0.556230	yell
999	0.000000	yes

데이터 프레임으로 단어와 회귀계수를 볼 수 있게 되었다.

5.4.5. 부정단어 사전

```
neg_df = sent_df[sent_df['계수'] < 0].sort_values(by='계수')
```

```
neg_df.head()
```

	계수	단어
492	-2.563357	poor
103	-2.239632	bad
803	-2.128382	terrible
988	-1.892567	worst
227	-1.884050	don

5.4.6. 긍정단어 사전

```
pos_df = sent_df[sent_df['계수'] > 0].sort_values(by='계수', ascending=False)
```

```
pos_df.head()
```

	계수	단어
310	5.266542	great
985	3.297022	works
114	3.264917	best
395	3.105162	love
255	3.062604	excellent

5.4.7. 감정사전 저장

```
sent_df.to_csv('sent_df.csv')
```

5.5. 감정사전을 이용한 감정 분석

이번 시간에는 감정사전을 사용해 문장에 감정점수를 부여해보자.

예시로 이전에 로지스틱 회귀분석으로 만들어 놓은 감정 사전을 사옹해보자.

```
import pandas as pd
```

```
sent_df = pd.read_csv('sent_df.csv', index_col=0)
```

감정단어 데이터 프레임을 확인해보자.

```
sent_df.head()
```

	계수	단어
492	-2.563357	poor
103	-2.239632	bad
803	-2.128382	terrible
988	-1.892567	worst
227	-1.884050	don

5.5.1. 극성

감정분석에서 긍정/부정을 극성(polarity)라고 부른다. 극성의 계산은 긍정 단어 수(N_{pos})와부정단어수(N_{neg})를 바탕으로 다음과 같이 계산한다.

$$\frac{N_{pos} - N_{neg}}{N_{pos} + N_{neg}}$$

극성이 +이면 긍정, -이면 부정으로 분류한다.

회귀분석을 통해 만든 감정 사전에서도 계수가 양수면 +1, 음수면 -1로 변환한다.

```
import numpy
```

```
sent_df['극성'] = numpy.sign(sent_df['계수'])
```

극성 계산의 편의를 위해 `단어` 를 인덱스로 설정한다.

```
sent_df.set_index('단어', inplace=True)
```

```
sent_df.head()
```

	계수	극성
단어		
poor	-2.563357	-1.0
bad	-2.239632	-1.0
terrible	-2.128382	-1.0
worst	-1.892567	-1.0
don	-1.884050	-1.0

5.5.2. 극성 계산

다음 문장의 극성을 계산해보자.

```
sentence = 'poor screen but reasonable price'
```

문장을 토큰 단위로 쪼갠다.

```
words = sentence.split()
```

토큰에 해당하는 행을 사전에서 찾는다.

```
sent_df.reindex(words)
```

	계수	극성
단어		
poor	-2.563357	-1.0
screen	0.175747	1.0
but	NaN	NaN
reasonable	0.297161	1.0
price	2.489535	1.0

사전에 없는 토큰을 지운다.

```
sent = sent_df.reindex(words).dropna()
```

```
sent
```

	계수	극성
단어		
poor	-2.563357	-1.0
screen	0.175747	1.0
reasonable	0.297161	1.0
price	2.489535	1.0

공식을 이용해 극성을 구한다.

```
sent['극성'].sum() / sent['극성'].abs().sum()
```

```
0.5
```

5.6. 인공신경망

5.6.1. 퍼셉트론

인공신경망(artificial neural network)는 생물의 신경망을 흉내낸 모형이다. 신경망은 신경 세포, 뉴런(neuron)의 네트워크다. 뉴런의 기본 작동방식은 이렇다.

첫째, 뉴런은 다른 뉴런들로부터 신호를 받는다.

둘째, 뉴런 사이의 연결 강도에 따라 신호는 크고 작게 변한다.

셋째, 받아들인 신호의 총량이 역치(threshold)를 넘어서면 뉴런은 활성화되어 다른 뉴런들로 신호를 전달한다.

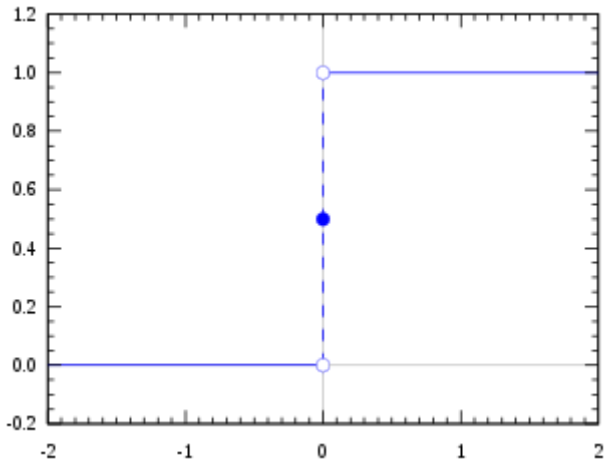
이러한 뉴런의 특성을 따른 최초의 모형은 1957년에 발표된 퍼셉트론(perceptron)이다. 뉴런의 작동방식에서 첫째와 둘째를 수식으로 나타내면 선형모형과 똑같다.

$$y = wx + b$$

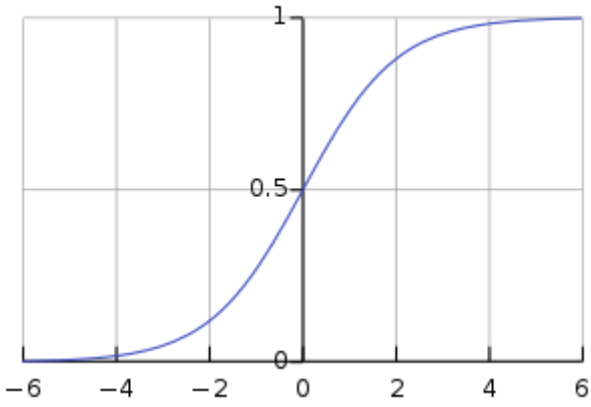
고전적인 퍼셉트론은 $y > 0$이면 활성화되서 1의 신호를 내보내고, 이외의 경우에는 0의 신호를 내보냈다.

5.6.2. 활성화 함수

다른 말로 표현하면 고전적 퍼셉트론의 활성화 함수는 계단 함수(step function)이다.



이런 함수는 기울기가 모든 점에서 0이고, 0에서는 불연속이기 때문에 경사하강법을 쓸 수가 없다. 그래서 매끄럽게 변하는 로지스틱 함수를 활성화 함수로 흔히 쓴다.



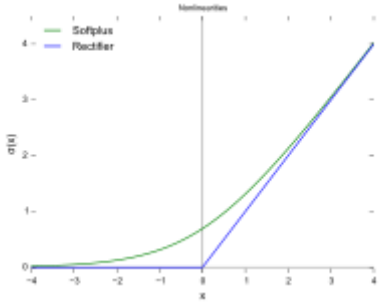
퍼셉트론에 로지스틱 함수를 활성화 함수로 적용하면 로지스틱 회귀분석과 완전히 동일한 형태가 된다. 로지스틱 함수는 출력값이 0에서 1사이의 범위를 갖는다.

신경망의 맥락에서는 로지스틱 함수는 시그모이드(sigmoid) 함수라는 이름으로 더 많이 부른다. 시그모이드는 그리스어로 "S자 모양의"라는 뜻이다.

이와 비슷하게 쌍곡 탄젠트 함수를 사용하기도 한다. 쌍곡 탄젠트는 출력값이 -1에서 1사이의 범위를 갖는다.

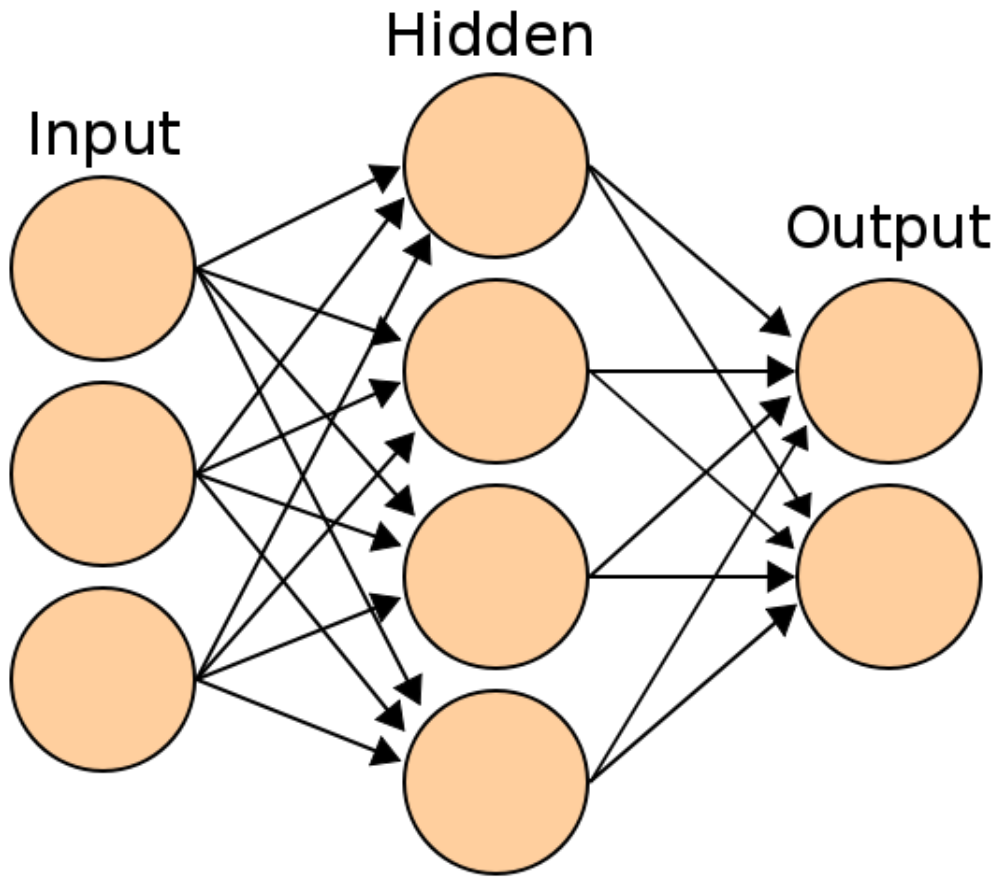
쌍곡탄젠트는 시그모이드에 비해 몇 가지 장점이 있다. 첫째, 출력값이 +, -로 나오기 때문에 편향되지 않는다. 둘째, 0 주변에서 기울기가 더 급해서 사라지는 경사 문제가 적다.

최근에는 ReLU라는 활성화 함수도 많이 사용된다. ReLU는 $\max(0, x)$ 형태의 함수로 입력값이 0보다 크면 그대로 출력값으로 내보내고 0보다 작으면 0을 내보내는 함수다.



5.6.3. 다층 퍼셉트론

아래 그림과 같이 퍼셉트론을 여러 겹으로 덧붙인 것이 다층 퍼셉트론(Multi-Layer Perceptron: MLP)이다. 앙상블에서 스택킹과도 비슷한 구조가 된다.



(출처: https://en.wikipedia.org/wiki/Artificial_neural_network)

MLP에서는 입력층(input layer)과 출력층(output layer) 사이에 여러 겹의 은닉층이 들어가게 된다. 입력층의 신호가 은닉층으로, 은닉층의 신호가 출력층으로 앞으로 앞으로 전달되는 방식이기 때문에 앞먹임 신경망(feedforward network)라고도 한다.

식으로 나타내보자면 아래와 같이 쓸 수 있다.

$$h = a(W_1 x + b_1)$$
$$y = a(W_2 h + b_2)$$

$a(w \cdot + b)$를 간단히 f라고 쓴다면 다음과 같다.

$$h = f_1(x)$$
$$y = f_2(x)$$

더 간단히 쓰자면 이렇게도 쓸 수 있다.

$$y = f_2(f_1(x))$$

5.6.4. 보편적 근사 정리

충분히 큰 은닉층을 가진 인공 신경망은 파라미터만 잘 조정하면 우리가 흔히 다루는 거의 모든 형태의 모형과 비슷하게 만들 수 있다. 이를 **보편적 근사 정리**(universal approximation theorem)라고 한다.

멋진 정리지만 2가지 함정이 숨어 있다. 하나는 '잘 조정하면'이다. 즉, 어떤 파라미터가 존재한다는 것이지 그걸 반드시 찾을 수 있다는 것은 아니다. 경사하강법이든 어떤 학습 알고리즘든 항상 국소최적에 빠지거나 과적합될 수 있다.

또 다른 함정은 '충분히 큰'(sufficiently large)이다. 수학에서 이 말은 어떤 명제가 어떤 크기 N에서 참이면 $N+1$이든 $N+2$이든 N보다 더 큰 크기에서도 참이라는 말이다. 예를 들면 충분히 큰 n에 대해 다음 식이 성립한다.

$$\frac{1}{n} < 0.001$$

보편적 근사 정리를 다시 풀어서 써보면 은닉층의 크기가 N인 신경망을 어떤 모형과 비슷하게 만들 수 있으면, 그 신경망보다 은닉층이 더 큰 신경망도 그 모형과 비슷하게 만들 수 있다. 실제로는 N은 다룰 수 없을 정도로 매우 클 수도 있다.

5.6.5. 심층 신경망 또는 딥러닝

경험적으로 작은 은닉층을 여러 층으로 겹쳐 쌓으면 큰 은닉층 하나와 비슷한 성능을 보인다. 이를 심층 신경망(deep neural network) 또는 딥러닝이라고 한다. 은닉층이 몇 개부터 심층인가에 대해서는 명확한 기준이 있는 것은 아니다.

은닉층을 많이 넣으면 예측력이 좋아진다는 것은 1980년대부터 알려져 있었다. 그렇지만 2000년대 초반까지도 딥러닝은 현실화되지 못했다.

시그모이드 함수의 기울기는 최대가 .25, 쌍곡탄젠트도 1이다. 위에서 봤듯이 신경망에 경사하강법을 적용하면 뒤쪽의 기울기가 여러 번 곱해지게 된다. 1보다 작은 수를 계속 곱하면 점점 작아지므로 앞쪽의 레이어에는 경사가 거의 0이 된다. 이를 사라지는 경사(vanishing gradient)라고 한다.

경사하강법은 말그대로 경사를 따라 내려가는 식으로 파라미터를 개선하는 방법이다. 그런데 경사가 사라지면 파라미터 개선이 안되고, 학습도 안된다. 은닉층을 많이 넣으면 성능이 좋아져야 하지만 실제로는 학습이 안되서 성능이 좋아지지 못하는 것이다.

2000년 초반에 이 문제는 한 가지 해결책이 제시되었다. 그것은 사전 훈련(pretraining)이라는 방식으로, 매 층을 따로 학습시켜서 쌓아 올리는 방법이었다. 그러나 현재는 사전 훈련 없이 한 번에 학습시키는 방식을 쓰고 있다.

5.6.6. 딥러닝이 가능한 이유

현재 딥러닝이 활성화된 이유는 바꿔말하면 사라지는 경사가 어느 정도 해결되었기 때문이다. 대표적인 것이 ReLU다. ReLU는 0보다 큰 범위에서는 기울기가 항상 1이기 때문에 사라지는 경사 문제가 적다.

그러나 가장 큰 이유는 데이터가 많아지고 컴퓨터가 빨라졌기 때문이다. 딥러닝에서 경사가 완전히 사라지는 것은 아니기 때문에 학습이 느려질 뿐 멈추지는 않는다. 현재는 데이터도 많

고 컴퓨터도 빠르기 때문에 많은 데이터로 오래 학습을 하면 결국에는 충분한 성능이 나올 때까지 학습을 시킬 수 있다.

바꿔말하면 데이터 양이나 컴퓨터 성능이 충분치 않다면 딥러닝은 충분한 효과를 보여주지 못한다.

5.7. 케라스를 이용한 로지스틱 회귀분석

5.7.1. 모형 만들기

```
from keras import Sequential
from keras.layers import Dense, Input
```

```
NUM_WORDS = tdm.shape[1]
```

단어의 갯수(`NUM_WORDS`)만큼 입력을 받아 1개의 출력을 내놓는 레이어를 추가한다.
`Dense` 는 선형 모형, `sigmoid` 는 로지스틱 함수와 같다.

```
m1 = Sequential()
m1.add(Dense(1, activation='sigmoid', input_shape=(NUM_WORDS,)))
```

```
m1.summary()
```

5.7.2. 학습

```
from keras.optimizers import Adam
```

데이터로 학습을 시킨다. 손실 함수는 크로스엔트로피를 쓴다. 문제에 따라 크로스엔트로피를 효과적으로 계산하는 방법이 달라지는데 둘 중에 하나로 분류할 때는
`binary_crossentropy` 를 사용한다. 3개 이상의 분류에는
`sparse_categorical_crossentropy` 를 사용한다. 최적화 알고리즘으로는 `Adam` 을 사용한다.

```
m1.compile(loss='binary_crossentropy', metrics=['accuracy'], optimizer=Adam())
```

```
m1.fit(X_train, y_train, epochs=30)
```

5.7.3. 예측

```
y_pred = m1.predict_classes(X_test)
```

5.7.4. 평가

```
from sklearn.metrics import accuracy_score
```

```
accuracy_score(y_test, y_pred)
```

```
0.73
```

5.7.5. 다층 신경망 모형 만들기

다층 신경망은 2개의 레이어를 겹쳐 쌓은 것이다. 1층은 단어 갯수만큼 입력을 받아 100개의 출력을 내놓는다. 2층은 100개의 입력을 받아 1개의 출력을 내놓는다. 여기서 100개는 임의로 정한 것으로 수치를 늘리면 학습이 잘 되지만 과적합이 일어나기 쉽고 줄이면 반대가 된다. 따라서 여러 가지 수치를 시도해보고 가장 성능이 좋은 것을 고른다.

```
m2 = Sequential()
m2.add(Dense(100, activation='relu', input_shape=(NUM_WORDS,)))
m2.add(Dense(1, activation='sigmoid'))
```

```
m2.summary()
```

```
m2.compile(loss='binary_crossentropy', metrics=['accuracy'], optimizer=Adam())
```

```
m2.fit(X_train, y_train, epochs=30, verbose=0)
```

```
<keras.callbacks.History at 0x1a5e2f88550>
```

```
y_pred2 = m2.predict_classes(X_test)
accuracy_score(y_test, y_pred2)
```

```
0.8
```

5.8. 텐서플로를 이용한 로지스틱 회귀분석

인공신경말을 포함하여 여러 기계학습 모형은 행렬이나 또는 행렬을 일반화한 텐서(tensor)를 이용해 계산한다. 텐서플로는 텐서를 쉽게 계산할 수 있도록 만든 라이브러리로 특히 인공신경망 모형을 만드는데 많이 사용되고 있다.

5.8.1. 즉시 실행

텐서플로에는 지연 실행(lazy execution) 모드와 즉시 실행(eager execution) 모드가 있다. 지연 실행은 여러 가지 계산을 '계산 그래프'라는 형태로 만들어두었다가 한 번에 실행한다. 속도는 빠르지만 사용이 어렵다. 즉시 실행은 말 그대로 계산을 즉시 실행하는 것이다.

텐서 플로는 지연 실행이 기본이므로 임포트 후 즉시 실행을 바로 켜줘야 한다.

```
import tensorflow as tf

tf.enable_eager_execution()
tfe = tf.contrib.eager
```

간단하게 행렬을 하나 만들어 보자.

```
a = tf.constant([[1, 2],
                 [3, 4]])
a
```

```
<tf.Tensor: id=6, shape=(2, 2), dtype=int32, numpy=
array([[1, 2],
       [3, 4]])>
```

텐서에서 넘파이 어레이로 변환하려면 `.numpy()` 메소드를 사용한다.

```
a.numpy()
```

```
array([[1, 2],
       [3, 4]])
```

행렬의 덧셈을 해보자.

```
tf.add(a, a)
```

```
<tf.Tensor: id=10, shape=(2, 2), dtype=int32, numpy=
array([[2, 4],
       [6, 8]])>
```

행렬의 곱셈을 해보자.

```
tf.matmul(a, a)
```

```
<tf.Tensor: id=12, shape=(2, 2), dtype=int32, numpy=
array([[ 7, 10],
       [15, 22]])>
```

5.8.2. 경사하강법

x가 1과 2 두 건의 데이터가 있고 y가 3과 5 두 건의 데이터라고 할 때 x를 이용해 y를 예측하는 선형 모형을 만들어보자.

```
x = tf.constant([[1.0], [2.0]])
y = tf.constant([[3.0], [5.0]])
```

```
x.numpy()
```

```
array([[1.],
       [2.]], dtype=float32)
```

```
y.numpy()
```

```
array([[3.],
       [5.]], dtype=float32)
```

선형 모형의 파라미터를 `Variable` 로 만든다.

```
w = tfe.Variable([1.0])  # 계수
b = tfe.Variable([0.0])  # 절편
```

먼저, x에 계수를 곱하고 절편을 더해서 예측값 `pred` 를 만든다. 다음으로 평균제곱오차(실제값과 예측값의 차이의 제곱의 평균)을 손실 `loss` 로 정한다. 이 과정을 `GradientTape` 에 기록하면 텐서플로가 자동미분을 실행한다.

```
with tf.GradientTape() as tape:
    pred = w * x + b
    loss = tf.reduce_mean((y - pred) ** 2)
```

예측값은 각각 1과 2가 된다.

```
pred.numpy()
```

```
array([[1.],
       [2.]], dtype=float32)
```

실제값은 3과 5이므로 평균제곱오차는 6.5가 된다.

```
loss.numpy()
```

```
6.5
```

이제 `loss` 에 대한 `w` 와 `b` 의 기울기를 구해보면 각각 -8과 -5가 된다. 따라서 `w` 와 `b` 를 증가시키면 손실이 감소한다.

```
w_grad, b_grad = tape.gradient(loss, [w, b])
```

```
w_grad.numpy()
```

```
array([-8.], dtype=float32)
```

```
b_grad.numpy()
```

```
array([-5.], dtype=float32)
```

학습률(η)을 .05로 정해주고 $w \leftarrow w - \eta \nabla w$의 식에 따라 파라미터를 업데이트 한다.

```
learning_rate = .05
```

```
w.assign_sub(learning_rate * w_grad)
b.assign_sub(learning_rate * b_grad)
w.numpy(), b.numpy()
```

```
(array([1.4], dtype=float32), array([0.25], dtype=float32))
```

w와 b가 업데이트 되었다. 이제 위의 과정을 10회 반복해보자.

```
for _ in range(10):
    with tf.GradientTape() as tape:
        pred = w * x + b
        loss = tf.reduce_mean((y - pred) ** 2)

    print(w.numpy(), b.numpy(), loss.numpy())

    w_grad, b_grad = tape.gradient(loss, [w, b])
    w.assign_sub(learning_rate * w_grad)
    b.assign_sub(learning_rate * b_grad)
```

직접 계산해보면 파라미터가 $w = 2, b = 1$일 때 손실이 0으로 최적이다. 위의 과정을 보면 점차 손실이 줄어들면서 파라미터가 최적화되어가는 것을 볼 수 있다.

5.8.3. 로지스틱 회귀 모형 만들기

```
import numpy
```

```
NUM_TRAIN, NUM_FEATURE = X_train.shape
```

```
NUM_TRAIN, NUM_FEATURE
```

```
(800, 1000)
```

`w` 는 단어의 수만큼 만들어 무작위로 초기화한다.

```
w = tfe.Variable(tf.random_normal([NUM_FEATURE, 1]))
```

```
b = tfe.Variable(0.0)
```

간단한 테스트를 위해 10건의 데이터만 가져온다. 자료형을 32비트 실수(`float32`)로, 형태는 어레이로 바꿔준다.

```
x = tf.constant(X_train[:10].astype(numpy.float32).toarray())
```

변수가 여러 개이기 때문에 행렬곱을 하는 것을 제외하면 선형 모형은 위에서 만들었던 것과 같다. 선형 모형에 로지스틱 함수(`tf.sigmoid`)를 적용한다.

```
z = tf.matmul(x, w) + b
p = tf.sigmoid(z)
```

```
p.numpy()
```

```
array([[0.62886614],
       [0.59435606],
       [0.64281607],
       [0.64559805],
       [0.1694062 ],
       [0.31435224],
       [0.3503407 ],
       [0.4511888 ],
       [0.41870806],
       [0.4376156 ]], dtype=float32)
```

크로스 엔트로피로 손실을 구해보자. 직접 계산하는 대신 텐서플로에 내장된 함수를 사용한다. 확률인 `p` 대신 로짓 `z` 를 넣어주는 점에 주의.

```
y = tf.constant(y_train.values[:10])
```

```
loss = tf.losses.sigmoid_cross_entropy(tf.expand_dims(y, 1), z)
loss.numpy()
```

```
0.8674763
```

학습

`Dataset` 을 이용하면 학습 데이터를 편리하게 다룰 수 있다.

```
from tensorflow.data import Dataset
```

`X_train` 과 `y_train` 을 데이터셋으로 만든다.

```
dataset = Dataset.from_tensor_slices((X_train.toarray().astype(numpy.float32), y_train
```

파라미터를 직접 업데이트 하는 대신에 미리 만들어져 있는 `GradientDescentOptimizer` 를 이용한다. 학습률은 10으로 한다.

```
optimizer = tf.train.GradientDescentOptimizer(learning_rate=10)
```

이제 30 에포크에 걸쳐 학습을 반복한다.

```
for _ in range(30):

    # 데이터를 갯수만큼 섞고 32개씩 학습시킨다
    for batch_x, batch_y in dataset.shuffle(NUM_TRAIN).batch(32):

        # 예측을 해서 손실을 구한다
        with tf.GradientTape() as tape:
            z = tf.matmul(batch_x, w) + b
            loss = tf.losses.sigmoid_cross_entropy(tf.expand_dims(batch_y, 1), z)

        # 경사하강법으로 파라미터를 업데이트 한다
        w_grad, b_grad = tape.gradient(loss, [w, b])
        optimizer.apply_gradients(
            [(w_grad, w), (b_grad, b)],
            global_step=tf.train.get_or_create_global_step())

    print(loss.numpy())
```

예측

```
x = tf.constant(X_test.astype(numpy.float32).toarray())
y = tf.constant(y_test.values)
z = tf.matmul(x, w) + b
p = tf.sigmoid(z)
loss = tf.losses.sigmoid_cross_entropy(tf.expand_dims(y, 1), z)
```

평가

```
from sklearn.metrics import accuracy_score
```

```
y_pred = (p.numpy() > .5).astype(int)
```

```
accuracy_score(y_test, y_pred)
```

```
0.785
```