Introduction to Machine Learning with Python

A Guide for Data Scientists

Andreas Muller & Sarah Guido / O’REILLY

1. Introduction

머신 러닝은 데이터로부터 지식을 추출해내는 작업이다. 머신 러닝은 통계학, 인공지능, 컴퓨터 사이언스와 교차되는 영역을 가지고 있고, 예측 분석 또는 통계적 학습 등으로도 알려져 있다.

머신 러닝 기법을 포함한 애플리케이션들이 요즘 우리의 일상생활 속에 나타나고 있다. 볼만한 영화 추천, 음식이나 상품 추천에서 개인화된 온라인 라디오, 내 사진 속의 친구 얼굴 인식까지 수많은 웹사이트와 디바이스가 그 핵심에 머신 러닝 알고리즘을 가지고 있다. 상업적인 영역 외에서도 머신 러닝은 오늘날 이뤄지고 있는 데이터 중심의 리서치에 막대한 영향을 끼치고 있다. 머신 러닝 알고리즘은 별을 이해하고, 외계 행성을 찾고, 새로운 물질을 발견하고, DNA 서열을 분석하고, 개인화된 암 치료법을 제공하는 등의 다양한 과학 문제에도 적용되고 있다.

왜 머신 러닝인가?

요즘 모든 스마트폰은 이미지에서 얼굴 인식을 한다. 하지만 얼굴 인식은 2001년까지만 해도 풀리지 않던 문제였다. 그 주된 원인은 컴퓨터가 인식하는 픽셀(pixel)은 인간이 얼굴을 인지하는 방식과 매우 다르다는 것이다. 사람이 디지털 이미지에서 얼굴을 구성하는 요소를 일일히 지정하는 것은 불가능하다. 그러나 머신 러닝을 이용하면 대량의 얼굴 이미지 데이터를 모은 후 얼굴을 인식하는데 필요한 특성이 무엇인지 결정하는 알고리즘을 적용한 프로그램을 실행하면 된다.

가장 유용한 머신 러닝 알고리즘 중 하나는 이미 알고 있는 사례들로부터 일반화를 위한 의사결정 프로세스를 자동으로 만들어 내는 것이다. 사용자가 알고리즘에 입력 데이터와 원하는 결과값을 주면, 알고리즘은 주어진 입력값을 기초로 원하는 결과값을 만들기 위한 방법을 발견해 낸다. 더 나아가, 알고리즘은 인간의 도움 없이도 이전에 전혀 볼 수 없었던 새로운 결과를 만들어 내기도 한다. 머신 러닝을 이용한 스팸 메일을 분류하는 사례의 경우, 사용자는 알고리즘에 입력 데이터로 어느 메일이 스팸인지 표시된 대량의 이메일을 제공한다. 새로운 메일이 오면, 도출된 의사결정 프로세스를 기반으로 알고리즘은 새 메일이 스팸인지 아닌지 예측을 한다.

이처럼 input/output 쌍으로 된 데이터로부터 학습하는 머신 러닝 알고리즘을 지도학습(supervised learning) 알고리즘이라고 부른다. 한 “교사”가 각각의 데이터로부터 원하는 결과가 나오도록 지도하기(supervise) 때문이다. 지도학습을 위한 데이터셋을 만드는 것은 종종 고생스런 수작업 과정이기도 하지만, 지도학습 알고리즘은 이해하기 쉽고 그 성과를 측정하기도 쉽다. 지도학습 머신 러닝이 적용되는 사례는 다음과 같다.

편지봉투에서 손으로 쓰여진 우편번호를 인식하여 자동 분류

의료용 이미지를 기반으로 종양이 양성인지 음성인지 판별

신용카드 사용 기록을 통해 부정 사용 실시간 감시

그와 다른 타입의 알고리즘으로 비지도학습(unsupervised learning) 알고리즘이 있다. 비지도학습은 입력 데이터만 알고리즘에 제공하고, 알고리즘에서 어떤 결과를 낼지 사전에 알 수 없다. 이 머신 러닝 알고리즘은 수많은 성공적인 애플리케이션에서 사용되고 있지만, 일반적으로 이해하기 어렵고, 그 결과를 평가하기도 쉽지 않다. 비지도학습 머신 러닝이 적용되는 사례는 다음과 같다.

블로그 포스팅들의 주요 주제 식별

고객들을 비슷한 취향을 가진 그룹으로 세분화

웹사이트에 대한 비정상적인 접근 패턴 감시

머신 러닝에 있어서 컴퓨터가 이해할 수 있도록 잘 준비된 입력 데이터는 매우 중요하다. 데이터 테이블의 각 행(row) 또는 객체(entity)를 머신 러닝에서 “샘플(sample)” 또는 “데이터 포인트(data point)”이라고 하며, 각 객체의 특성을 설명하는 컬럼(column)은 변수 또는 “피쳐(feature)”라고 부른다.

나중에 우리는 당신의 데이터를 더 잘 대표할 수 있는 “파생 변수 추출”(feature extraction, feature engineering)과 관련하여 자세히 알아볼 것이다.

머신 러닝 프로세스에서 가장 중요한 부분은 당신이 작업하고 있는 데이터와 그것이 해결하고자 하는 문제와 어떻게 관련되어 있는지에 대해 정확히 이해하는 것이다. 각 알고리즘은 필요로 하는 데이터의 종류가 무엇인지, 어떤 문제 해결에 효과적인지 모두 다르기 때문이다. 당신이 머신 러닝 솔루션을 개발중이라면 다음 질문에 답을 하거나, 아니면 최소한 마음에 새기고 있는 것이 좋을 것이다.

* 내가 답하려고 하는 질문은 무엇인가? 수집한 데이터로 그 질문에 답을 할 수 있다고 생각는가?
* 내 질문을 머신 러닝 문제로 표현하는 최상의 방법은 무엇인가?
* 해결하고자 하는 문제를 보여줄 데이터를 충분히 수집했는가?
* 데이터에서 내가 새로이 추출한 변수(feature)는 무엇이며, 그것으로 올바른 예측이 가능한가?
* 예측 성공을 어떻게 측정할 것인가?
* 머신 러닝 솔루션을 다른 비즈니스들과 어떻게 연동할 것인가?

왜 파이썬인가?

파이썬은 수많은 데이터 과학 애플리케이션의 공용어가 되었다. 파이썬은 데이터 입출력, 시각화, 통계, 자연어 처리, 이미지 처리 등과 관련된 라이브러리들을 가지고 있다. 파이썬을 사용하는 가장 큰 장점 중 하나는 Jupyter Notebook 같은 도구를 사용하면 코드의 실행 결과를 즉각적으로 알 수 있다는 것이다. 이런 도구들은 반복적인 작업을 쉽고 빠르게 처리하는데 도움이 된다.

scikit-learn (<http://scikit-learn.org/)>

scikit-learn은 파이썬에서 가장 많이 사용하는 머신 러닝 라이브러리이며, 사용과 배포가 자유로운 오픈 소스 프로젝트이다. scikit-learn 프로젝트는 아주 활동적인 사용자 커뮤니티를 가지고 있으며, 계속 기능이 발전하고 향상되고 있는 중이다. 따라서 최첨단 머신 러닝 알고리즘을 포함하고 있으며, 각 알고리즘에 대해 이해하기 쉬운 문서들도 제공하고 있다.

문서 링크 <http://scikit-learn.org/stable/documentation.html>

사용자 가이드 <http://scikit-learn.org/stable/user_guide.html>

scikit-learn은 NumPy와 SciPy, 이 두 가지 파이썬 패키지와 깊은 연관이 있다. 또한 차트를 그리는 등의 대화형(interactive) 개발을 원한다면 matplotlib, Jupyter Notebook 등의 패키지도 설치해야 한다.

* 아나콘다(Anaconda)

대규모 데이터 처리, 예측 분석, 과학적 컴퓨팅 등을 위해 만들어진 파이썬 배포판(Python distribution)이다. 맥OS, 윈도우즈, 리눅스 모두에서 사용 가능하며, 파이썬 패키지들의 설치를 쉽게 해주는 솔루션이다. 아나콘다에는 NumPy, SciPy, matplotlib, pandas, Ipython, Jupyter Notebook, scikit-learn 등과 같은 패키지들이 포함되어 있으며, 이 패키지들은 아니콘다 설치 후 콘솔창에서 다음과 입력하면 쉽게 설치가 된다.

$ conda install <packagename>

현재 아나콘다에는 무료로 사용할 수 있는 인텔의 MKL 라이브러리가 포함되어 있으며, 아나콘다 설치시 자동으로 설치된다. MKL을 사용하면 scikit-learn의 수많은 알고리즘들의 성능을 눈부시게 향상시킬 수 있다.

아나콘다 다운로드 : [www.continuum.io/downloads](http://www.continuum.io/downloads)

* Jupyter Notebook

Jupyter Notebook은 웹브라우저에서 코드를 실행할 수 있는 대화형 개발도구이다. 이것은 탐색적 데이터 분석에 매우 유용한 도구이기 때문에 데이터 많은 데이터 사이언티스트들이 사용하고 있다. Jupyter Notebook은 파이썬뿐만 아니라 다른 프로그래밍 언어도 지원한다. 아나콘다 설치시 Jupyter Notebook은 자동으로 함께 설치된다.

* NumPy

NumPy는 파이썬에서 과학적인 컴퓨팅을 위해 기본이 되는 패키지 중 하나이다. NumPy는 다차원 배열(multidimensional array), 선형 대수와 같은 고차원 수학연산 기능, 푸리에 변환, 난수 생성 등과 같은 기능을 포함하고 있다.

scikit-learn에서 NumPy array는 기본적인 데이터 구조이다. NumPy는 2차원, 3차원의 배열 구조를 만들 수 있으며, NumPy array에 포함된 데이터들은 반드시 동일한 데이터 타입이어야 한다.

* SciPy

SciPy는 과학적 연산 기능을 모아놓은 파이썬 패키지이다. scikit-learn 알고리즘들은 SciPy의 함수들을 많이 끌어다 쓴다. SciPy에서 우리에게 중요한 부분은 scipy.sparse 이다. 이것은 scikit-learn에서 데이터를 처리할 때 많이 사용하는 희소 행렬(sparse matrix)을 제공해 준다.

* matplotlib

matplotlib는 파이썬의 핵심적인 플롯 라이브러리이다. 이 라이브러리는 라인 차트, 히스토그램, 산점도 등등의 수준높은 시각화 기능을 제공한다. 데이터와 분석결과의 다양한 측면을 시각화하는 것은 우리에게 중요한 통찰력을 제공해 줄 것이다.

Jupyter Notebook에서 작업할 때 “%matplotlib notebook”과 “%matplotlib inline” 명령어를 사용하면 웹브라우저에서 차트를 직접 확인할 수 있다.

* pandas

pandas는 데이터를 처리하고 분석하기 위한 파이썬 라이브러리이다. pandas는 데이터프레임(DataFrame)이라 불리는 데이터 구조를 사용하는데, 이것은 R 의 데이터프레임을 참조한 것이다. DataFrame은 간단히 말해서 엑셀의 스프레드시트와 비슷한 테이블이다. pandas는 이 테이블을 다루는 매우 다양한 기능을 제공하고 있으며, 특히 테이블을 SQL 형태로 조회하고 조인할 수 있다. NumPy array이가 동일한 데이터 타입의 객체만 가지고 있는 반면에, pandas dataframe은 각각의 컬럼이 서로 다른 데이터 타입을 가질 수 있다. pandas의 아주 유용한 특징 중 하나는 다양한 형태의 파일과 데이터베이스에서 데이터를 가져올 수 있다는 것이다.

첫번째 애플리케이션 : 붓꽃의 종 분류

이제 간단한 머신 러닝 애플리케이션과 첫번째 모델을 만들어 보자.

식물학이 취미인 한 여성은 자신이 발견한 어떤 붓꽃이 무슨 종인지 알고 싶어한다. 그녀는 이 붓꽃의 꽃잎 길이와 너비, 꽃받침 길이와 너비를 센티미터 단위로 측정했다. 그녀는 또한 전문 식물학자가 많은 붓꽃들을 측정하여 각 꽃들을 setosa, versicolor, virginica 의 세 종으로 분류한 데이터를 가지고 있다.

우리의 목적은 어떤 종에 속하는지 이미 알고 있는 붓꽃들의 측정 데이터로부터 학습할 수 있는 머신 러닝 모델을 만든 후 새로 발견한 붓꽃이 어떤 종인지 예측해 보는 것이다.

우리는 각 붓꽃의 종이 정확히 무엇인지 알고 있는 데이터를 가지고 있기 때문에, 이것은 지도학습 문제이다. 또한 우리는 여러 개의 옵션들(3개의 종) 중에 하나를 예측해야 하기 때문에 이것은 분류(classification) 문제라고 할 수 있다. 도출될 수 있는 결과들을 우리는 클래스(class)라고 부르기 때문이다. 데이터셋에 포함된 모든 붓꽃은 세 개의 클래스 중 하나에 속해 있다.

하나의 데이터 포인트(붓꽃)에 대해 도출되길 원하는 결과는 이 꽃의 ‘종’이다. 하나의 특정한 데이터 포인트에 대해서 그것이 속해있는 종을 우리는 “레이블(label)”이라고 한다.

모델의 성과 측정 : 트레이닝 데이터와 테스트 데이터

우리는 이 데이터를 기반으로 새로운 측정치를 가진 데이터셋에서 붓꽃의 종을 예측하는 머신 러닝 모델을 만들려고 한다. 그런데 예측 모델을 만들기 전에 우리는 그것이 실제로 어떻게 동작하는지, 다시 말해서 우리가 그 예측을 신뢰할 수 있는지 알아야 할 필요가 있다.

안타깝게도 우리는 모델을 만드는 데 사용한 데이터로 그 모델을 평가할 수 없다. 왜냐하면 단순하게 말해서 우리의 모델은 트레이닝 데이터셋을 다시 기억해내기 때문이다. 이 ‘기억해낸다’는 것이 우리 모델이 일반화가 잘 되어 있다는, 다시 말해서 다른 새로운 데이터에 대해서도 잘 동작한다는 것을 가리키지는 않는다.

모델의 성과를 측정하기 위해서 우리는 모델에게 레이블 되어 있는(목표값이 표시된) 새로운(이전에 학습해보지 않은) 데이터를 던져 주어야 한다. 이를 위해서 가지고 있는 레이블된 데이터를 두 개로 분리시킨다. 분리된 데이터 중 하나는 머신 러닝 모델을 만들기 위해 사용하며, 이것을 트레이닝 데이터(training data) 또는 트레이닝셋(training set)이라고 부른다. 데이터의 나머지 부분은 모델이 얼마나 잘 동작하는지 검증하기 위해 사용하며, 테스트 데이터(test data) 또는 테스트셋(test set)이라고 한다.

scikit-learn은 데이터셋을 뒤섞고 분리하는 train\_test\_split이라는 함수를 가지고 있다. 이 함수는 레이블의 분포를 고려하여 전체 행의 75%를 트레이닝 데이터에 할당하고, 나머지 25%를 테스트 데이터로 할당한다. 트레이닝 데이터와 테스트 데이터에 각각 얼만큼씩 할당할 것인지는 어느정도 임의적으로 결정할 수 있지만, 경험에 근거했을 때 테스트 데이터에 25% 정도가 포함되는 것이 가장 좋다고 할 수 있다.

scikit-learn에서 데이터는 일반적으로 대문자 X로 표시하고, 레이블은 소문자 y로 표시한다. 이것은 수학의 표준 공식인 f(x) = y 에서 영향을 받은 것이다. 하지만 머신 러닝에서 데이터는 2차원 배열(매트릭스 matrix)이기 때문에 대문자 X를 사용하고, 목표 변수는 1차원 배열(벡터 vector)이기 때문에 소문자 y를 사용한다.

<code>

train\_test\_split 함수는 데이터를 분리하기 전에 난수 생성을 통해 데이터셋을 뒤섞는다. 만약 원본 데이터셋에서 뒤쪽의 25%를 테스트 데이터로 가져온다면, 모든 데이터 포인트가 앞에서 조회해 본 것처럼 레이블 ‘2’를 가지고 있게 된다. 3개의 클래스 중 하나만 포함하고 있는 테스트셋은 모델이 얼마나 일반화가 잘 되어 있는지 말해줄 수 없기 때문에, 모든 클래스가 테스트 데이터에 골고루 포함되도록 우리의 데이터를 잘 섞어야 한다.

같은 함수를 여러번 실행해도 동일한 결과가 나오도록 하기 위해 난수 생성기에 random\_state 파라미터를 사용하여 고정된 시드값(seed value)을 정해준다.

train\_test\_split 함수의 결과물은 X\_train, X\_test, y\_train, y\_test 이며 이것들은 모두 NumPy array이다. X\_train에는 전체 데이터셋에서 75%의 데이터가 포함되어 있고, X\_test 에는 나머지 25%의 데이터가 들어 있다.

가장 중요한 걸 먼저 하자 : 데이터 탐색

머신 러닝 모델을 만들기 전에 데이터를 잘 살펴보는 것이 좋다. 머신 러닝 없이도 문제를 쉽게 해결할 수 있을지도 모르고, 또는 원하는 결과를 내는데 필요한 정보가 데이터에 포함되어 있지 않을 수도 있다.

또한 데이터를 잘 살피는 것은 이상값이나 특이값을 가진 데이터 포인트를 발견하는데 도움이 된다. 예를 들어, 아마 어떤 붓꽃은 센티미터가 아닌 인치로 측정된 값을 가지고 있을 수도 있다. 실제 상황에서도 데이터 내의 모순과 예측하지 못한 값들이 들어있는 경우는 매우 흔하다.

데이터를 점검하기 위해 가장 좋은 방법 중 하나는 시각화하는 것이다. 그 중 하나로 산점도(scatter plot)를 살펴보자. 산점도는 데이터의 한 피쳐를 x 좌표에 놓고, 다른 한 피쳐를 y 좌표로 삼아 각각의 데이터 포인트를 점으로 나타낸 것이다. 컴퓨터 화면은 아쉽게도 2차원이기 때문에 우리는 산점도에서 한번에 2가지 (또는 3가지) 피쳐에 대해서만 볼 수 있다. 이 문제를 극복하기 위해 가능한 모든 피쳐들의 쌍에 대해서 보여주는 pair plot를 사용하기도 한다. 만약 가지고 있는 데이터의 피쳐 갯수가 적다면 추천할 만하다. 하지만 기억해야 할 것은 pair plot이 모든 피쳐들의 상호작용을 한번에 보여주지 못한다는 것이다. 그래서 데이터 어떤 측면은 이 방법으로 시각화를 해도 드러나지 않을 수 있다.

그림 1-3은 트레이닝 데이터의 피쳐들에 대한 pair plot이다. 각 데이터 포인트는 자신이 속한 종에 따라서 색상을 표시했다. pair plot을 그리기 위해 우리는 우선 데이터의 NumPy array를 pandas의 DataFrame으로 변환해야 한다. pandas는 scatter\_matrix라는 pair plot을 위한 함수를 가지고 있다. 이 매트릭스의 대각선은 각 피쳐에 대한 히스토그램으로 되어 있다.

<코드>

pair plot을 통해 우리는 세 개의 클래스가 꽃받침과 꽃잎의 측정치에 따라 어느 정도 구분되어 있음을 알 수 있다. 이는 머신 러닝 모델이 그것들을 구분할 수 있도록 학습시킬 수 있음을 의미한다.

첫번째 모델 : k-최근접 이웃

이제 실제적인 머신 러닝 모델을 만들어 보자. scikit-learn에는 사용할 수 있는 분류분석 알고리즘이 매우 많다. 여기서 우리가 사용할 것은 이해하기 쉬운 k-최근접 이웃 분류기(k-nearest neighbors classifier)이다. 모델을 생성하기 위해 트레이닝 데이터를 넣어 주기만 하면 된다. 새 데이터 포인트에 대해 예측을 하기 위해서 알고리즘은 트레이닝 데이터의 포인트 중에서 새 데이터와 가장 가까운 포인트를 찾아낸다. 그리고 그 지점의 트레이닝 데이터가 가진 레이블을 새 데이터 포인트의 레이블로 할당한다.

k-최근접 이웃에서 k는 트레이닝 데이터에서 새로운 데이터 포인트와 가장 가까운 데이터 포인트를 몇 개까지 선택할 것이지를 지시하는 고정된 숫자이다. 이 이웃들 중에서 우세한 클래스를 예측된 결과로 반환한다. 더 자세한 내용은 “제2장”에서 살펴보고, 여기서는 최근접 이웃을 한 개만 사용한다.

scikit-learn의 모든 머신 러닝 모델은 Estimator 클래스라고 불리는 자기 소유의 클래스에서 구현된다. k-최근접 이웃 분류 알고리즘은 neighbors 모듈에 있는 KNeighborsClassifier에서 구현된다. 그래서 우리는 모델을 사용하기 전에 클래스를 객체로 인스턴스화해야 한다. KNeighborsClassifier의 가장 중요한 파라미터는 이웃의 갯수이며, 우리는 여기서 1로 설정한다.

<코드>

knn 객체는 트레이닝 데이터로부터 모델을 생성하기 위해 사용될 알고리즘 뿐만 아니라, 새로운 데이터 포인트에 대하여 예측을 수행할 알고리즘도 포함하고 있다.

트레이닝셋으로 모델을 만들기 위해서 knn 객체의 fit 메소드를 호출한다. 이 메소드는 트레이닝 데이터인 X\_train과 트레이닝 레이블인 y\_train을 인수로 취한다.

<코드>

fit 메소드는 knn 객체 자체, 즉 우리의 분류기인 KNeighborsClassifier를 반환한다. 반환된 결과에서 우리는 모델을 생성하는데 사용된 파라미터들을 확인할 수 있다. 아마도 거의 모두가 기본 설정값(default value)이고, n\_neighbors=1 하나만 우리가 지정한 값일 것이다. scikit-learn의 모델들은 많은 파라미터를 가지고 있는데, 그 대부분은 처리 속도 최적화 아니면 아주 특별한 경우에 사용하는 것이다. 그러므로 위 반환값이 보여주는 다른 파라미터들은 신경쓰지 않아도 된다.

예측

이제 이 모델을 이용해서 정확한 레이블이 무엇인지 모르는 새로운 데이터에 대한 예측을 수행해 보자. 우리가 길가에서 새로 발견한 붓꽃의 꽃받침 길이가 5cm, 너비가 2.9 cm 이고, 꽃잎 길이가 1 cm, 너비가 0.2 cm 라고 가정해보자. 이 붓꽃의 종은 무엇일까? 우선 이 데이터를 샘플 수(1)와 피쳐 수(4)로 된 형태의 NumPy array로 만든다.

<코드>

scikit-learn에서는 항상 2차원 NumPy array를 데이터 타입으로 원하고 있음을 기억할 것이다.

예측 결과를 얻기 위해 knn 객체의 predict 메소드를 호출한다.

<코드>

우리의 모델은 새 붓꽃이 클래스 0에 속한다고 예측했다. 하지만 이 모델을 얼마나 신뢰할 수 있을지 알 수 있을까?

모델 평가

우리가 처음에 테스트셋을 만든 이유가 여기에 있다. 이 데이터는 모델을 만들 때 사용되지 않았지만, 각 붓꽃이 정확히 무슨 종인지 그 결과를 알고 있다. 그러므로 우리는 테스트 데이터의 각 붓꽃에 대해 예측을 해본 후 그것을 각각의 레이블(알고 있는 종)과 비교해 볼 수 있다. 레이블된 종과 예측한 종을 비교하면 그 정확도(accuracy)를 계산할 수 있기 때문에 이 모델이 얼마나 잘 작동하는지 측정할 수 있다.

<코드>

knn 객체의 score 메소드를 사용하면 우리가 계산한 정확도와 같은 값을 바로 얻을 수 있다.

<코드>

이 모델의 테스트 데이터에 대한 정확도는 약 0.97이며, 이것은 테스트셋 중 97%의 붓꽃에 대해 정확한 예측을 했음을 의미한다. 그러므로 우리가 새로 발견한 붓꽃에 대한 모델의 예측이 97% 정확하다고 말할 수 있다. 나중에 우리는 이 성능을 더 향상시키는 방법과 그에 따른 위험에 대해 더 논의할 것이다.

1장 정리

이제 우리가 1장에서 배운 내용을 정리해 보자. 우리는 머신 러닝에 대한 간략하게 살펴본 후 지도학습과 비지도학습의 차이점을 얘기했다. 그 다음으로 우리는 꽃들을 물리적으로 측정한 데이터를 이용하여 특정한 붓꽃이 어떤 종에 속하는지 예측하는 방법을 계획했다. 지도학습의 방법론을 따라 전문 식물학자가 측정하여 각 꽃의 정확한 종을 명시해놓은 데이터를 모델을 만드는데 사용한다. 한 붓꽃이 속한 종을 “레이블”이라고 부르며, 여기에는 setosa, versicolor, virginica의 세 가지 종이 있기 때문에 우리는 이것은 3개 클래스에 대한 분류분석 문제로 정의했다.

붓꽃 데이터셋은 2개의 NumPy array로 구성되어 있다. 하나는 데이터이며 scikit-learn에서 X로 지칭된다. 다른 하나는 정확한(도출되길 원하는) 목표값이 포함된 배열이며 y로 지칭된다. X는 피쳐를 가진 2차원 배열로서 데이터 포인트 당 하나의 행(row), 피쳐 당 하나의 열(column)을 가지고 있다. y는 1차원 배열로서 각 샘플 당 하나의 클래스 레이블을 0에서 2까지 범위의 정수로서 가지고 있다.

우리는 데이터셋을 모델을 만드는데 사용하는 트레이닝 데이터와, 모델이 이전에 보지 못한 새로운 데이터에 대한 예측력을 얼마나 가지는지 평가할 때 사용하는 테스트 데이터로 분리했다.

우리는 k-최근접 이웃 분류분석 알고리즘을 선택했다. 이 알고리즘은 새로운 데이터 포인트에 대해 트레이닝 데이터 중 가장 가까운 이웃을 찾아 예측 결과를 만든다. 모델 생성 및 예측은 알고리즘을 가지고 있는 KNeighborsClassifier를 통해 구현했다. 먼저 이 클래스를 파라미터와 함께 인스턴스화 시킨다. 그 다음 fit 메소드를 통해 트레이닝 데이터를 넘겨주고 모델을 생성한다. 그리고 score 메소드로 테스트 데이터에 대한 정확도를 알 수 있다. 이 정확도가 새로운 데이터에 적용할 수 있을만한 신뢰를 준다면 predict 메소드를 호출하여 새로운 데이터 포인트에 대한 예측을 수행할 수 있다.

여기까지의 코드를 정리하면 다음과 같다.

<코드>

이것은 scikit-learn에 있는 머신 러닝 알고리즘의 핵심을 포함하고 있다. fit, predit, score 메소드는 scikit-learn 내부의 지도학습 알고리즘들의 공통 인터페이스이다.

다음 장에서 우리는 다른 유형이 지도학습 모델들을 좀 더 깊이있게 다룰 것이다.

제2장 지도학습 (Supervised Learning)

앞에서 살펴보았듯이 지도학습 머신 러닝은 가장 널리 쓰이는 성공적인 타입의 머신 러닝이다. 이번 장에서 우리는 supervised learning 에 대해 좀 더 깊이 다루고, 몇 가지 유명한 알고리즘들에 대해 살펴볼 것이다. 우리는 이미 1장에서 붓꽃의 물리적인 측정치를 이용하여 그 종을 분류하는 지도학습 머신 러닝 애플리케이션을 경험했다.

지도학습은 주어진 입력 데이터로부터 사전에 정해진 어떤 특정한 결과를 예측하길 원하고, 입력/출력 데이터 쌍을 가지고 있을 때 사용된다는 것을 기억할 것이다. 우리는 트레이닝셋으로 불리는 입력/출력 데이터로부터 머신 러닝 모델을 만들었다. 우리의 목적은 이전에 보지 못한 새로운 데이터에 대해서 정확한 예측값을 도출하는 것이다.

분류 분석과 회귀 분석

지도학습 머신 러닝에서 주로 사용되는 두 가지 타입은 분류분석(Classification)과 회귀분석(Regression)이다.

분류분석의 목적는 클래스 레이블을 예측하는 것이다. 클래스 레이블은 도출되기를 원하는 결과를 사전에 정의해 놓은 목록이다. 1장에서 우리는 선택가능한 세 가지 종 중의 하나로 붓꽃의 종을 분류했다. 분류분석은 종종 이진 분류분석(binary classification)과 다중클래스 분류분석(multiclass classification)으로 나뉜다. 이진 분류는 단 두 개의 클래스 중 하나로 구분하는 특수한 경우이며, 다중클래스 분류의 클래스는 두 개 이상이다. 이진 분류분석은 단순히 질문에 “예/아니오”로 답하는 것이라 생각하면 된다. 이메일이 스팸인지 아닌지 분류하는 것을 그 예로 들 수 있다. 반면에, 붓꽃의 종을 분류하는 예는 다중클래스 분류분석이라고 할 수 있다. 특정 웹사이트에 사용된 언어를 분류하는 일을 또 다른 예로 들 수 있다. 선택 가능한 대상 언어들은 물론 미리 정의되어 있을 것이다.

회귀분석의 목적은 연속된 숫자, 또는 수학용어로 실수값을 예측하는 것이다. 어떤 사람의 교육 수준, 나이, 주거지에 따라 그의 연수입을 예측하는 것은 회귀분석의 한 예이다. 예측된 값은 주어진 범위내의 어떠한 값이라도 될 수 있다. 회귀분석의 또 다른 예로는 옥수수 농장의 지난해 생산량, 날씨, 일꾼의 수를 근거로 올해의 생산량을 예측하는 것이다. 생산량은 임의의 어떤 수라도 될 수 있다.

회귀분석과 분류분석을 구별하는 가장 쉬운 방법은 결과값에 연속성이 있는지 없는지를 보는 것이다. 만약 도출 가능한 여러 결과값들 사이에 연속성이 있으면 회귀분석 문제라고 할 수 있다. 어떤 사람의 연수입 금액에는 확실한 연속성이 있다. 그의 1년 수입이 40,000 달러이든, 40,001 달러이든 실제적인 차이는 미미하다. 즉 40,000 달러로 예측이 나와야하는 알고리즘에서 결과가 39,999 또는 40,001 어느 쪽으로 나와도 큰 문제가 되지는 않는다.

하지만, 웹사이트에 사용된 언어를 판별하는 문제에 있어서는(이것은 분류분석 문제이다) 매우 다르다. 한 웹사이트는 A 또는 B 언어로 되어 있다. 두 언어 사이에는 연속성이라는 개념이 있을 수 없다. 이를테면 프랑스어와 영어 사이에 해당하는 언어란 존재하지 않는다.

Generalization, Overfitting, and Underfitting

지도학습에서 우리는 트레이닝 데이터를 기반으로 모델을 만들고, 그 다음에 우리가 사용한 트레이닝셋과 동일한 특성을 가진 새로운 데이터에 대한 예측을 한다. 만약 그 모델이 새로운 데이터에 대해 정확한 예측을 할 수 있다면, 우리는 트레이닝셋으로부터 테스트셋으로 일반화(generalize)가 가능하다고 말할 수 있다. 우리가 원하는 것은 가능한 한 정확하게 일반화된 모델을 만들어 내는 것이다.

일반적으로 우리는 트레이닝 데이터에 맞추는 방식으로 모델을 만든다. 만약 트레이닝셋과 테스트셋이 서로 공통적인 요인을 충분히 가지고 있다면 모델의 테스트셋에 대한 예측이 정확할 것이라고 기대할 수 있다. 하지만, 그렇지 못한 경우가 종종 있다.

이 점에 대한 자세한 설명을 위해 만든 예제를 살펴보자. 신참 데이터 사이언티스트가 이전에 보트를 구매한 고객과 그렇지 않은 고객에 대한 데이터를 바탕으로 어느 고객이 보트를 구매할지 예측하려고 한다. 실제 구매가 이루어질 가능성이 높은 사람들에게 프로모션 이메일을 보내야 하지만, 전혀 관심이 없는 사람들에게 까지 보내서 귀찮게 하면 안 된다.

아래의 고객 테이블을 보자.

<테이블 이미지>

테이블2-1. 보트 구매 관련 고객 데이터

잠시 데이터를 살펴본 신참 데이터 사이언티스트는 다음과 같은 규칙을 생각해냈다: "45세 이상, 자녀 3명 미만이며 이혼하지 않은 상태인 고객은 보트를 구매하고 싶어한다." 이 규칙이 얼마나 정확한지 질문을 한다면 이 데이터 사이언티스트는 "100% 정확합니다"라 대답할 것이다. 사실, 테이블에 나타난 데이터를 보면 이 규칙은 정확하다. 데이터상 같은 나이가 두번 나타나는 경우는 없기 때문에 나이가 66, 52, 53, 58세인 고객들은 보트를 구매하길 원할 것이고 나머지는 아니라고 말할 수 있다.

이 데이터셋을 바탕으로 그 외에도 많은 규칙들을 만들어낼 수 있지만, 이 데이터셋에 대해서 예측을 수행하는 것이 목표가 아님을 기억해야 한다. 이 고객들에 대한 답을 우리는 이미 알고 있기 때문이다. 우리가 원하는 것은 신규 고객들에게 보트 구매 의사가 있는지 알아내는 것이다. 우리에게 필요한 것은 새로운 고객들에 대해 잘 동작하는 규칙을 찾는 것이기 때문에, 트레이닝셋에 대해 100%의 정확성을 갖는 모델은 전혀 도움이 되지 않는다. 그 데이터 사이언티스트의 규칙이 새로운 고객에게도 잘 동작하리라고 기대하긴 어려울 것이다.

알고리즘이 새로운 데이터에 대해서도 잘 동작하는지 아닌지 측정하는 유일한 방법은 테스트셋으로 검증하는 것이다. 하지만 우리는 직감적으로 단순한 모델일수록 새 데이터에 대해 좀 더 일반화가 잘 된다는 것을 알 수 있다. “50세 이상인 사람은 보트를 사고 싶어한다”는 규칙은 여기에 자녀 수와 결혼 상태를 추가한 규칙보다 다른 고객들에 대해 더 잘 설명할 것이다. 그렇기 때문에 우리는 항상 가장 단순한 모델을 찾길 원한다. 그래서 “가지고 있는 정보의 양에 비해 모델이 너무 복잡한” 경우 과적합(overfitting) 되었다고 말한다.

모델이 트레이닝셋의 특성에 너무 최적화되어 새로운 데이터에 대해서 일반화를 잘 하지 못할 때 overfitting이 발생한다. 그와 반대로, 만약 당신의 모델이 너무나 단순하다면 (예를 들어 “집을 소유한 모든 사람은 보트 구매를 원한다”) 그 모델은 데이터의 다양한 측면을 제대로 잡아내지 못한 것이며, 트레이닝셋에 대해서도 제대로 동작하지 못할 것이다. 너무 단순한 모델을 선택하는 것을 underfitting 이라고 한다.

우리의 모델을 조금 더 복잡하게 만들면 트레이닝 데이터에 대한 예측력은 좋아질 것이다. 하지만 이 모델이 지나치게 복잡해진다면 트레이닝셋의 각 개별 데이터 포인트에 너무 집중하게 되어서 새로운 데이터에 대해서는 일반화가 전혀 되지 않을 것이다.

여기서 최상의 일반화 성능(generalization performance)을 내는 sweet spot이 있음을 알 수 있다. 이 지점이 우리가 찾는 모델이다. 그림 2-1은 overfitting과 underfitting 사이의 상호 모순적 관계를 나타낸 것이다.

<그림2-1. 모델 복잡도와 트레이닝셋/테스트셋 예측 정확도 간의 관계>

데이터셋의 크기와 모델 복잡도의 관계

모델의 복잡도는 트레이닝 데이터셋에 포함된 입력값의 분포와 매우 밀접한 관계를 가지고 있다. 매우 다양한 데이터 포인트들을 가지고 있다면 과적합을 피하면서 좀더 복잡한 모델을 사용할 수 있다. 일반적으로 데이터의 양이 늘어날수록 데이터의 다양성도 커지기 때문에 대용량의 데이터셋이 좀더 정교한 모델을 만들기에 좋다. 하지만 동일한 데이터 포인트의 중복이 많거나 서로 매우 비슷한 데이터가 많은 것은 도움이 되지 않는다.

앞의 예제로 돌아가서, 만약 우리가 10,000개 이상의 고객 데이터를 가지고 있다면 “45세 이상이고, 자녀가 3명 이하, 이혼하지 않은 상태의 고객이라면 보트 구매에 관심을 가지고 있다.”라는 규칙은 12개 데이터를 사용했을 때보다 훨씬 더 신뢰성이 가질 것이다.

이 책에서 우리는 고정된 크기의 데이터셋을 사용할 것이다. 하지만 실제 업무에서는 얼마나 많은 데이터를 수집해야 하는지 결정하는 능력이 당신의 모델을 개선하는데 많은 도움이 될 것이다. 대량의 데이터가 주는 이점을 절대 과소평가하지 말라.

지도학습 머신 러닝 알고리즘

우리는 이제 가장 많이 알려진 머신 러닝 알고리즘을 살펴보고, 데이터로부터 학습하는 방법과 예측을 수행하는 방법에 대해 설명할 것이다. 또한 모델의 복잡도에 따라 모델이 어떻게 달라지는지 논의하고, 각 알고리즘이 모델 생성 원리에 대해 간략하게 살펴볼 것이다. 각 알고리즘의 장점과 단점, 그리고 어떤 형태의 데이터를 적용하는 것이 좋을지도 알아볼 것이다. 그리고 알고리즘에서 중요한 파라미터들과 옵션들에 대해서도 설명할 것이다.

각 알고리즘에 대한 설명을 너무 상세하게 읽을 필요는 없지만, 그 모델들을 이해하는 것은 머신 러닝 알고리즘들이 서로 다른 방식으로 동작함을 알게 해줄 것이다.

샘플 데이터셋

우리는 알고리즘들을 설명하기 위해 몇가지 데이터셋을 사용할 것이다. 어떤 데이터셋은 알고리즘의 특정한 측면을 강조하기 위해 임의적으로 만든 것이며, 또 다른 것은 사이즈가 큰 실무용 데이터이다.

한 예로 이진 분류를 설명하기 위해 만든 forge 데이터셋은 2개의 피쳐를 가지고 있다. 다음의 코드는 이 데이터셋의 모든 데이터 포인트를 보여주는 산점도(그림2-2)를 만드는 것이다. 산점도의 x축은 첫번째 피쳐이고, y축은 두번째 피쳐이다. 산점도에서는 항상 하나의 데이터 포인트는 하나의 점으로 표현된다. 점의 색깔과 모양은 해당 데이터의 클래스를 가리킨다. X.shape 로 볼 수 있듯이 이 데이터셋은 26개의 데이터 포인트를 가지고 있다.

<코드>

회귀분석 알고리즘을 설명하기 위해서는 wave 데이터셋을 사용할 것이다. wave 데이터셋은 1개의 입력 변수와 1개의 연속형 목표 변수를 가지고 있다. (그림2-3)

<코드>

이렇게 단순하고 저차원의 데이터셋을 사용하는 이유는 그것을 2차원 공간에 시각화하기가 좋기 때문이다. 2개 이상의 피쳐를 가진 데이터는 모든 특성을 다 보여주기 쉽지 않다.

우리는 또한 scikit-learn에 포함되어 있는 실무적인 데이터셋도 함께 사용한다. 그 중 하나는 위스콘신 유방암 데이터셋(앞으로 cancer로 지칭)으로 유방암과 관련된 종양의 의학적인 측정치들이 포함되어 있다. 각 종양은 “benign”(온화한, 무해한 종양) 또는 “malignant”(악성의, 암이 되는 종양)으로 레이블되어 있다.

실제 회귀분석용 데이터인 보스턴 주택가격 데이터셋도 사용할 것이다. 이 데이터셋은 1970년대에 보스턴 여러 지역의 주택가격 중앙값(median)을 예측하기 위한 자료로, 범죄율, 찰스 강과의 거리, 고속도로 접근성 등등의 정보를 포함하고 있다. 이 데이터셋은 13 개의 피쳐에 의해 설명되고 있는 506 개의 데이터 포인트를 가지고 있다.

여기서 우리의 한 가지 목적은 이 데이터셋의 입력값이 되는 13개 피쳐들 사이의 결합을 통해 생성 가능한 모든 상호작용을 만들어내어 데이터셋을 확장시키는 것이다. 즉, 현재 가지고 있는 범죄율과 고속도로 접근성 피쳐를 결합하여 새로운 피쳐를 만드는 것이다. 이처럼 새로운 파생 컬럼을 만드는 것은 피쳐 엔지니어링(feature engineering)이라고 부르며, 나중에 제4장에서 자세히 논의할 것이다. 파생변수들이 추가된 데이터셋은 load\_extended\_boston 함수로 불러올 수 있다.

기존의 13 개 피쳐에, 이것들을 결합하여 새로 만든 피쳐 91개를 더해 전체 104 개의 피쳐를 가지고 있다.

이제 다양한 머신 러닝 알고리즘들의 특성들을 살펴보자. 가장 먼저 1장에서 본 k-최근접 이웃(k-NN, k-Nearest Neighbors) 알고리즘을 다시 만나 보자.

k-최근접 이웃

k-NN 알고리즘은 거의 틀림없이 가장 단순한 머신 러닝 알고리즘이다. 새로운 데이터 포인트에 대해 예측을 할 때, 트레이닝 데이터 내에서 가장 가까운 데이터 포인트(즉, 최근접 이웃)를 찾는 알고리즘이다.

k-최근접 이웃 분류분석

k-NN 알고리즘의 가장 단순한 버전은 최근접 이웃을 단 한 개만 고려하는 것이다. 그림2-4는 forge 데이터셋에 이 케이스(최근접 이웃 1개)를 적용한 분류분석을 나타낸 것이다.

세 개의 새로운 데이터 포인트는 별모양의 점으로 표시되어 있다. 각각에 대해 가장 가까운 트레이닝 데이터가 무엇인지 표시했다. 바로 그 지점의 데이터 포인트가 가진 레이블(색상으로 구분된 클래스)이 1-NN 알고리즘이 내리는 예측 결과이다.

최근접 이웃의 수로 1만이 아닌 임의의 숫자 k개를 적용하는 것도 고려해 볼 수 있다. 여기서 k-최근접 이웃 알고리즘의 이름이 유래했다. 이웃의 수를 1개 이상 사용할 경우에는 표결(voting)을 통해 레이블을 할당한다. 각 테스트 데이터 포인트의 최근접 이웃들을 클래스 0에 속한 갯수와 클래스 1에 속한 갯수를 계산한 후 더 많은 쪽의 클래스로 정하는 것이다. 다음의 예는 최근접 이웃을 3개 사용한 경우이다.

왼쪽 위에 있는 신규 데이터 포인트에 대한 예측이 1개의 이웃만 사용한 예측 결과와 달라진 것을 볼 수 있다.

k-NN 알고리즘은 이처럼 이진 분류 문제뿐만 아니라 클래스의 갯수가 여러 개인 데이터셋에도 적용할 수 있다. 클래스가 여러 개인 경우의 예측에도 이웃들이 속한 클래스 중에서 그 수가 가장 많은 클래스를 레이블로 결정한다.

이제 scikit-learn에서 k-NN 알고리즘을 어떻게 사용하는지 알아보자. 우선 1장에서 배운 것처럼 데이터를 트레이닝 데이터와 테스트 데이터로 분리한다.

그 다음, k-NN 분류기 클래스를 임포트하고 인스턴스화 시킨다. 이 때 이웃의 숫자 등과 같은 파라미터를 설정할 수 있다. 여기서는 3으로 설정하자.

트레이닝셋을 사용하여 분류기의 fit 메소드를 호출해 모델을 생성한다.

predict 메소드를 호출하여 테스트 데이터로 예측을 수행한다. 테스트셋의 각 데이터 포인트는 트레이닝셋에서 해당 갯수의 최근접 이웃을 계산하고 그 중 가장 빈도가 많은 클래스를 찾는다.

이 모델이 얼마나 일반화가 잘 되었는지 평가하기 위해 테스트 데이터 및 테스트 레이블과 함께 score 메소드를 호출한다.

이 모델의 정확도가 86%임을 알 수 있다. 이것은 테스트 데이터의 샘플 중 86%의 클래스를 정확히 예측했음을 말해준다.

KNeighborsClassifier 분석

2차원 데이터셋에 대한 가능한 예측을 xy 평면에 표현할 수 있다. 그러면 할당되는 클래스에 따라 색깔로 표시된 영역이 나타난다. 이것은 알고리즘이 어느 선에서 클래스 0과 클래스 1을 나누는지 알 수 있는 의사결정 경계선(ecision boundary)을 보기 위한 것이다. 다음 코드는 최근접 이웃을 1, 3, 9개로 설정했을 때 나오는 decision boundary를 시각화한 것이다.

1개의 이웃을 사용한 왼쪽 그림에서 decision boundary가 트레이닝 데이터와 가깝게 진행되고 있는 것을 볼 수 있다. 그리고 이웃의 갯수가 늘어날수록 decision boundary는 점점 매끄러워지고 있다. 매끄러운 경계선은 단순한 모델에 해당한다. 다시 말해서, 작은 이웃 갯수는 모델의 높은 복잡도를 나타내며(그림2-1의 오른쪽), 많은 이웃은 낮은 모델 복잡도를 표현한다(그림2-1의 왼쪽).

모델 복잡도와 일반화 성능 사이에서 어떻게 결정을 내려야 하는지 알아보자. 여기서는 유방암 데이터셋을 사용한다. 데이터를 먼저 트레이닝 데이터와 테스트 데이터로 나눈다. 그리고 나서 이웃 갯수가 달라짐에 따라 트레이닝셋과 테스트셋의 성능을 평가한다. 그림2-7이 그 결과를 보여주고 있다.

위 그림은 x축에 표시된 (모델이 사용한) 이웃 갯수에 따른 트레이닝셋과 테스트셋에 대한 예측 성공률을 y축에 보여준다.

k-최근접 이웃 회귀분석

k-최근접 이웃 알고리즘을 이용해 회귀분석을 할 수도 있다. 이번에는 wave 데이터셋을 사용해서 1개의 이웃으로 시작해보자. x축에 초록색 별로 표시된 것처럼 3개의 테스트 데이터 포인트를 추가했다. 1개의 이웃을 사용한 경우 예측값은 가장 가까운 이웃의 목표변수 값(target value)이다. 이것은 그림2-8에 파란색 별로 표시되어 있다.

회귀분석도 1개보다 많은 최근접 이웃을 사용할 수 있다. 여러 개의 이웃을 사용할 때 예측값은 해당 이웃들의 평균이다.(그림2-9)

회귀분석을 위한 k-NN 알고리즘은 scikit-learn의 KNeighborsRegressor 클래스에서 구현된다. 사용방법은 KNeighborsClassifier와 비슷하다.

이 모델도 역시 score 메소드를 통해 평가를 할 수 있는데, 회귀분석의 경우에는 R2 (R square) 값을 리턴한다. R square는 0에서 1사이의 값으로 회귀 모델의 예측력(또는 설명력)을 말해준다. 1은 완벽한 예측력을 나타낸다. 이 모델에서는 0.83이 나왔는데, 상대적으로 모델의 예측력이 좋음을 가리킨다.

KNeighborsRegressor 분석

1차원 데이터셋일 경우에 모든 가능한 피쳐값에 대해서 예측값이 어떻게 나오는지 살펴보자.

1개의 이웃만 사용한 경우에 트레이닝셋의 각 포인트가 예측값에 명확하게 영향을 주고 있음을 볼 수 있다. 즉, 예측된 값들(라인)이 모든 데이터 포인트 위를 지나가고 있다. 이런 특징은 매우 불안정한 예측을 유도할 수 있다. 예측에 사용하는 이웃 수가 많아지자 예측 결과가 매끈해졌지만, 이것은 오히려 트레이닝 데이터에 제대로 fitting이 안된 것일 수도 있다.

장점, 단점, 그리고 파라미터들

원칙적으로 KNeighborsClassifier에는 2개의 주요 파라미터가 있다. 최근접 이웃 갯수와 데이터 포인트 간의 거리를 측정하는 방법이다. 실무적으로 3 ~ 5개의 작은 수로 이웃 갯수를 설정하면 대부분 잘 동작하지만, 어떤 경우에는 이 파라미터를 조절해 줄 필요가 있다. 거리 측정의 방법을 선택하는 것은 이 책의 범위를 어느정도 벗어나는 것이다. 기본 설정값인 유클리드 거리(Euclidean distance)를 사용하면 일반적인 경우 잘 동작한다.

k-NN 알고리즘의 장점 중 하나는 매우 이해하기 쉽다는 것이며, 많은 조정이 필요없이도 괜찮은 성과를 낸다는 것이다. 이 알고리즘은 다른 고급 기법들을 시도해 보기 전에 기본적으로 사용하는 것이 좋다. 이 모델은 처리 속도가 매우 빠르지만, 트레이닝 데이터가 매우 큰 경우 (피쳐의 갯수 또는 샘플의 갯수 모두) 예측 수행 속도가 느릴 수도 있다. k-NN 알고리즘을 사용할 때, 데이터 전처리(제3장)는 매우 중요하다. 데이터셋의 피쳐가 매우 많은 경우 (100개 이상) 종종 제대로 동작하지 않을 수도 있으며, 대부분의 피쳐에 0 값이 아주 많이 들어가 있는 경우(이를 sparse dataset이라 부른다) 특히 좋지 않다.