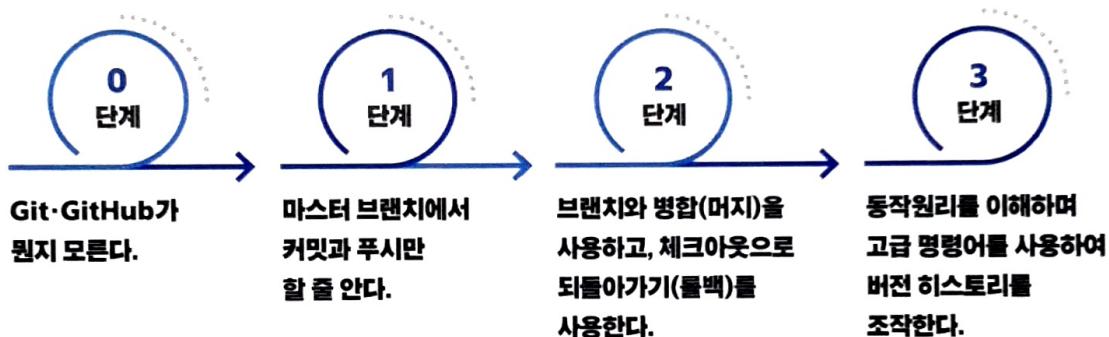


실무에서 바로 쓰는 Git 협업 시나리오 GUI로 이해하고 CLI로 정복한다!



여러분은 몇 단계 사용자인가요?

가장 기본적인 명령어부터 그래픽 툴로 Git 사용하기, Git 내부구조 들어보기, 그리고 오픈소스 컨트리뷰션까지 실무에서 사용하는 Git, GitHub에 필요한 내용을 모두 다룹니다.



무조건 쉽게 쓰자. 단, 제대로!

이 책의 집필 목표입니다. 이 목표를 위해 이 책은 세 개의 큰 덩어리로 구성했습니다.

CHAPTER 0 빠른 실습으로 감 잡기

간단한 명령어를 통해 Git, GitHub로 버전 관리하는 하나의 사이클을 실습합니다.

PART 1 친숙한 그래픽 툴로 Git 이해하기

GUI 환경에서 비주얼하게 Git을 배웁니다.
그림을 통해 Git의 핵심 원리를 설명하고 실무 사례를 들어 실습합니다.

PART 2 명령어로 개발자스러움을 즐기며 Git 정복하기

CLI 환경에서 Git 명령을 학습하고 동작원리를 함께 이해합니다.
다양한 명령어를 응용하며 Git을 숙달하고 정복합니다.

소프트웨어공학/개발방법론



정가 18,000원

ISBN 979-11-6224-203-2

```
>>> with tf.device("/gpu:0"):
...     i = tf.Variable(3)
[...]
>>> sess.run(i.initializer)
Traceback (most recent call last):
[...]
tensorflow.python.framework.errors.InvalidArgumentError: Cannot assign a device
to node 'Variable': Could not satisfy explicit device specification
```

초깃값이 정수이기 때문에 텐서플로는 그 변수가 int32 형일 것으로 추측합니다. 만약 초깃값을 3이 아니라 3.0으로 바꾸거나 변수를 만들 때 dtype=tf.float32로 명시하면 에러가 발생하지 않을 것입니다.

간접 배치

기본적으로 커널이 없는 장치에 연산을 할당하면 텐서플로가 그 장치에 연산을 배치할 때 앞서 본 것처럼 에러를 발생시킵니다. 에러를 발생시키는 대신 텐서플로가 CPU를 사용하도록 하려면 allow_soft_placement 환경 설정을 True로 지정합니다.

```
with tf.device("/gpu:0"):
    i = tf.Variable(3)

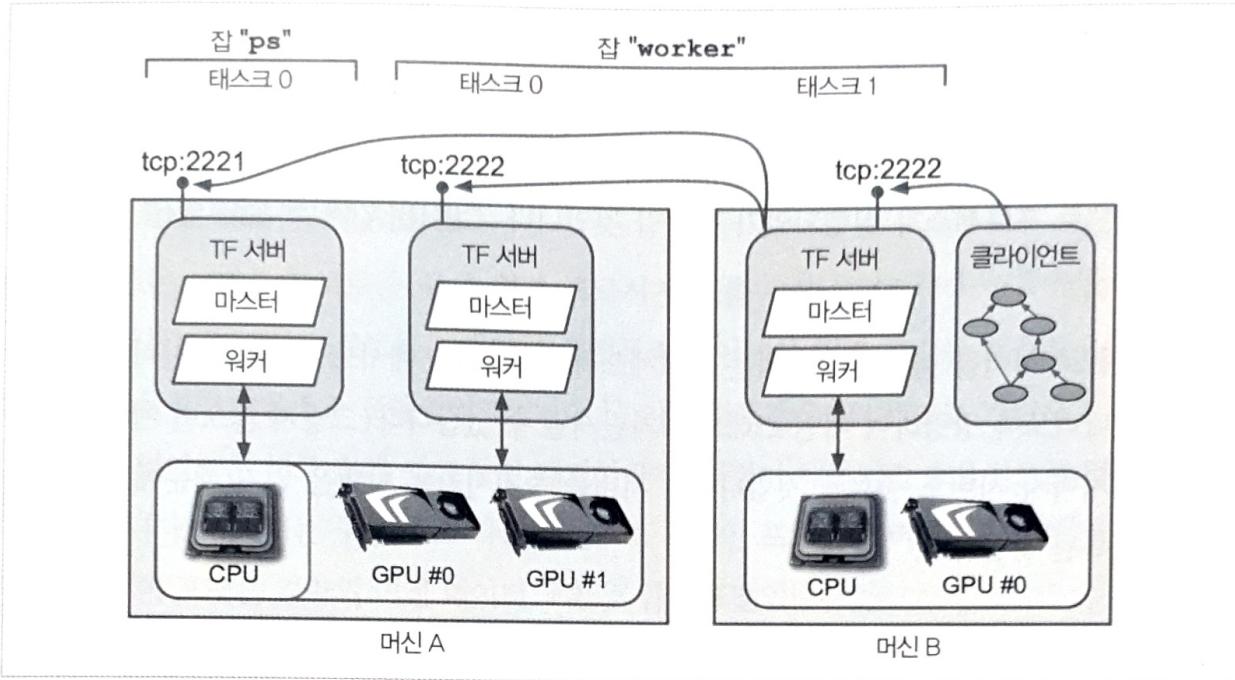
config = tf.ConfigProto()
config.allow_soft_placement = True
sess = tf.Session(config=config)
sess.run(i.initializer) # 배치자가 실행되면 결국 /cpu:0에 배치됩니다.
```

지금까지 노드를 다른 장치에 할당하는 법에 대해 이야기했습니다. 이제 텐서플로가 이런 노드를 어떻게 병렬로 실행하는지 살펴보겠습니다.

12.1.4 병렬 실행

텐서플로는 계산 그래프를 실행할 때 먼저 평가해야 할 연산을 찾고, 각 연산이 다른 연산에 얼마나 많이 의존하는지 카운트합니다. 그런 다음 의존성이 전혀 없는 노드(즉, 소스 노드 source

그림 12-6 텐서플로 클러스터



다음의 **클러스터 명세** cluster specification는 각각 한 개와 두 개의 태스크를 가지는 잡 "ps"와 "worker"를 정의합니다. 이 예제에서 머신 A는 두 개의 텐서플로 서버(즉, 태스크)를 다른 포트로 서비스하고 있습니다. 하나는 "ps" 잡이고 다른 하나는 "worker" 잡의 일부분입니다. 머신 B는 하나의 텐서플로 서버에서 "worker" 잡의 일부분을 서비스하고 있습니다.

```
cluster_spec = tf.train.ClusterSpec({
    "ps": [
        "machine-a.example.com:2221",  # /job:ps/task:0
    ],
    "worker": [
        "machine-a.example.com:2222",  # /job:worker/task:0
        "machine-b.example.com:2222",  # /job:worker/task:1
    ]})
```

텐서플로 서버를 시작하기 위해서는 Server 객체를 만들고 (다른 서버와 통신을 위해) 클러스터 명세와 잡 이름, 태스크 번호를 전달해야 합니다. 예를 들어 첫 번째 "worker" 태스크를 시작하려면 머신 A에서 다음 코드를 실행합니다.

```
server = tf.train.Server(cluster_spec, job_name="worker", task_index=0)
```

그다음에는 어떤 파일을 읽어야 할지 리더에 알려주기 위해 큐를 만듭니다. enqueue 연산을 만들고 원하는 파일 이름을 큐에 넣기 위해 플레이스홀더를 만듭니다. 더 이상 읽을 파일이 없을 때 큐를 종료하기 위해 종료 연산도 만듭니다.

```
filename_queue = tf.FIFOQueue(capacity=10, dtypes=[tf.string], shapes=[()])  
filename = tf.placeholder(tf.string)  
enqueue_filename = filename_queue.enqueue([filename])  
close_filename_queue = filename_queue.close()
```

이제 한 번에 하나의 레코드(즉, 한 줄)를 읽어서 키/값 쌍을 반환하는 read 연산을 만들 준비가 되었습니다. 키는 레코드의 고유 식별자로, 파일 이름과 콜론(:) 그리고 줄 번호로 이루어져 있습니다. 값은 그냥 그 줄의 내용이 들어 있는 문자열입니다.

```
key, value = reader.read(filename_queue)
```

파일을 한 줄씩 읽기 위해 필요한 모든 준비를 마쳤습니다! 하지만 아직 완전히 끝난 것은 아닙니다. 이 문자열을 특성과 타깃으로 나누어야 합니다.

```
x1, x2, target = tf.decode_csv(value, record_defaults=[[-1.], [-1.], [-1.]])  
features = tf.stack([x1, x2])
```

첫 번째 줄은 텐서플로의 CSV 파서 parser를 사용해 현재 줄에서 값을 추출합니다. 필드가 비어 있을 경우(여기에서는 세 번째 샘플의 x2 특성)에는 기본값이 사용됩니다. 또한 기본값을 보고 각 필드의 데이터 타입을 결정합니다(여기에서는 두 개의 실수와 하나의 정수).

마지막으로 훈련 샘플과 타깃을 RandomShuffleQueue에 넣어 훈련 그래프에 공유하고(큐에서 미니배치 방식으로 꺼낼 수 있습니다), 샘플을 모두 넣었을 때 큐를 종료하기 위한 연산을 만듭니다.

```
instance_queue = tf.RandomShuffleQueue(  
    capacity=10, min_after_dequeue=2,  
    dtypes=[tf.float32, tf.int32], shapes=[[2], []],  
    name="instance_q", shared_name="shared_instance_q")  
enqueue_instance = instance_queue.enqueue([features, target])  
close_instance_queue = instance_queue.close()
```

과 출력으로(그리고 아마도 네트워크를 건너서) 데이터를 옮기는 데 드는 시간이 계산 부하를 분할해서 얻는 속도보다 더 클 것이기 때문입니다. 이런 점에서 더 많은 GPU를 추가하면 대역폭을 더 포화시키고 훈련을 느리게 만들 것입니다.

TIP 비교적 규모가 작고 매우 큰 훈련 세트에서 훈련되는 모델일 경우 단일 GPU의 단일 머신에서 모델을 훈련시키는 것이 종종 더 낫습니다.

대역폭 포화는 전송해야 할 파라미터와 그래디언트가 많기 때문에 대규모 밀집 모델에서 더욱 심각합니다. 작은 모델의 경우에는 덜 심하고(하지만 병렬화의 이득이 작습니다). 크지만 희소한 모델의 경우에는 대부분 그래디언트가 0이므로 효율적으로 통신할 수 있습니다. 구글 브레인 프로젝트의 창시자이자 리더인 제프 딘 Jeff Dean이 <http://goo.gl/E4ypxo>²²에서 밀집 모델을 50개의 GPU에 훈련(계산을 분산)시킬 때 보통 25~40배 빨라지고 희소 모델을 500개의 GPU에 훈련시킬 때 300배 빨라진다고 보고했습니다. 다음은 일부 구체적인 사례입니다.

- 신경망 기계 번역: 8개 GPU에서 6배 속도 증가
- 인셉션 이미지넷: 50개 GPU에서 32배 속도 증가
- 랭크브레인 RankBrain²³: 500개 GPU에서 300배 속도 증가

이 수치는 2016년 1분기 시점에서 최고 수준의 성능입니다. 밀집 모델에서 수십 개의 GPU와 희소 모델에서 수백 개의 GPU를 넘어서면 대역폭이 포화되기 시작하고 성능이 감소됩니다. 이 문제를 해결하기 위한 연구가 많이 진행되고 있습니다(중앙 집중화된 파라미터 서버 대신 피어 투피어 peer-to-peer 구조 탐구, 손실 모델 압축, 복제 모델의 통신 시기와 양에 대한 최적화 등의 기술을 사용해서). 그러므로 향후 몇 년 안에 신경망의 병렬화에 많은 진전이 있을 것입니다.

그동안에는 대역폭 포화를 감소시키기 위해 취할 수 있는 몇 가지 방법이 있습니다.

- 많은 서버에 분산하지 말고 몇 대의 서버에 GPU를 모읍니다. 이렇게 하면 불필요한 네트워크 연결^{hop}을 피할 수 있습니다.
- 여러 대의 파라미터 서버에 파라미터를 분산시킵니다(앞서 설명한 대로).
- 모델 파라미터의 실수 정밀도를 32비트(tf.float32)에서 16비트(tf.float16)로 감소시킵니다. 이렇게 하면 수령 속도나 모델 성능에 크게 영향을 끼치지 않고 전송하는 데이터 양을 절반으로 줄일 수 있습니다.

22 옮긴이_ 이 동영상에서 41분 39초 쯤(<https://goo.gl/voChor>)에 관련 내용이 언급됩니다.

23 옮긴이_ 랭크브레인은 2015년부터 구글 검색 엔진에 적용된 인공 신경망 알고리즘입니다. 인셉션과 랭크브레인에 대한 자료는 구글 클라우드 엔지니어인 카츠 사토(Kaz Sato)가 발표한 자료에서 확인할 수 있습니다(<https://goo.gl/nAjLWZ>).

상충되는 목표를 가지고 있습니다. 마진 오류를 최소화하기 위해 가능한 한 슬랙 변수의 값을 작게 만드는 것과 마진을 크게 하기 위해 $\frac{1}{2} \mathbf{w}^T \cdot \mathbf{w}$ 를 가능한 한 작게 만드는 것입니다. 여기에 하이퍼파라미터 C 가 등장합니다. 이 파라미터는 두 목표 사이의 트레이드오프를 정의합니다. 결국 [식 5-4]에 있는 제약을 가진 최적화 문제가 됩니다.

식 5-4 소프트 마진 선형 SVM 분류기의 목적 함수²⁰

$$\underset{\mathbf{w}, b, \zeta}{\text{minimize}} \quad \frac{1}{2} \mathbf{w}^T \cdot \mathbf{w} + C \sum_{i=1}^m \zeta^{(i)}$$

[조건] $i = 1, 2, \dots, m$ 일 때 $y^{(i)}(\mathbf{w}^T \cdot \mathbf{x}^{(i)} + b) \geq 1 - \zeta^{(i)}$ 이고 $\zeta^{(i)} \geq 0$

5.4.3 콴드라틱 프로그래밍

하드 마진과 소프트 마진 문제는 모두 선형적인 제약 조건이 있는 볼록 함수의 이차 최적화 문제입니다. 이런 문제를 콴드라틱 프로그래밍 Quadratic Programming (QP) 문제라고 합니다. 여러 가지 테크닉으로 QP 문제를 푸는 알고리즘이 많이 있지만 이 책의 범위를 벗어납니다.²¹ 일반적인 문제 공식은 [식 5-5]와 같습니다.

식 5-5 QP 문제

$$\underset{\mathbf{p}}{\text{minimize}} \quad \frac{1}{2} \mathbf{p}^T \cdot \mathbf{H} \cdot \mathbf{p} + \mathbf{f}^T \cdot \mathbf{p}$$

[조건] $\mathbf{A} \cdot \mathbf{p} \leq \mathbf{b}$

여기서 $\left\{ \begin{array}{l} \mathbf{p} \text{는 } n_p \text{ 차원의 벡터} (n_p = \text{모델 파라미터 수}) \\ \mathbf{H} \text{는 } n_p \times n_p \text{ 크기 행렬} \\ \mathbf{f} \text{는 } n_p \text{ 차원의 벡터} \\ \mathbf{A} \text{는 } n_c \times n_p \text{ 크기 행렬} (n_c = \text{제약 수}) \\ \mathbf{b} \text{는 } n_c \text{ 차원의 벡터} \end{array} \right.$

20 옮긴이_SVM 회귀를 위한 목적 함수는 분류의 경우와 조금 다릅니다. 회귀에서는 결정 경계의 양쪽으로 모든 샘플을 담기 위한 도로의 오차 폭을 두 개의 슬랙 변수 ζ, ζ^* 라고 할 때, $y^{(i)} - (\mathbf{w}^T \cdot \mathbf{x}^{(i)} + b) \leq \varepsilon + \zeta^{(i)}$ 와 $(\mathbf{w}^T \cdot \mathbf{x}^{(i)} + b) - y^{(i)} \leq \varepsilon + \zeta^{(i)*}$ 두 조건을 만족하는 목적 함수를 구성합니다.

21 QP에 대해 더 자세한 내용을 알려면 스티븐 보이드(Stephen Boyd)와 리벤 반덴버그(Lieven Vandenberghe)의 「Convex Optimization」(<http://goo.gl/FGXuLw>) (영국 캠프리지 대학교, 2004)나 리차드 브라운(Richard Brown)의 동영상 강의 (<http://goo.gl/rTo3Af>)를 참고하세요.

6.7 규제 매개변수

결정 트리는 훈련 데이터에 대한 제약사항이 거의 없습니다(반대로 선형 모델은 데이터가 꼭 선형일 거라 가정합니다). 제한을 두지 않으면 트리가 훈련 데이터에 아주 가깝게 맞추려고 해서 대부분 과대적합되기 쉽습니다. 결정 트리는 모델 파라미터가 전혀 없는 것이 아니라(보통 많습니다) 훈련되기 전에 파라미터 수가 결정되지 않기 때문에 이런 모델을 **비파라미터 모델** nonparametric model이라고 부르곤 합니다. 그래서 모델 구조가 데이터에 맞춰져서 고정되지 않고 자유롭습니다. 반대로 선형 모델 같은 **파라미터 모델** parametric model은 미리 정의된 모델 파라미터 수를 가지므로 자유도가 제한되고 과대적합될 위험이 줄어듭니다(하지만 과소적합될 위험은 커집니다).

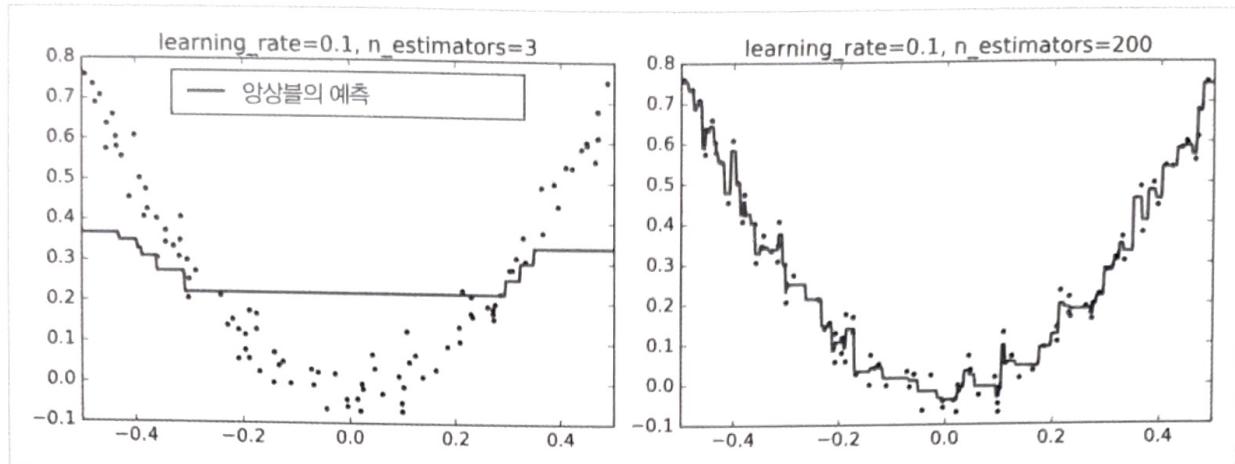
훈련 데이터에 대한 과대적합을 피하기 위해 학습할 때 결정 트리의 자유도를 제한할 필요가 있습니다. 이미 알고 있듯이 이를 규제라고 합니다. 규제 매개변수는 사용하는 알고리즘에 따라 다르지만, 보통 적어도 결정 트리의 최대 깊이는 제어할 수 있습니다. 사이킷런에서는 `max_depth` 매개변수로 이를 조절합니다(기본값은 제한이 없는 것을 의미하는 `None`입니다). `max_depth`를 줄이면 모델을 규제하게 되고 과대적합의 위험이 감소합니다.

`DecisionTreeClassifier`에는 비슷하게 결정 트리의 형태를 제한하는 다른 매개변수가 몇 개 있습니다. `min_samples_split`(분할되기 위해 노드가 가져야 하는 최소 샘플 수), `min_samples_leaf`(리프 노드가 가지고 있어야 할 최소 샘플 수), `min_weight_fraction_leaf` (`min_samples_leaf`와 같지만 가중치가 부여된 전체 샘플 수에서의 비율), `max_leaf_nodes` (리프 노드의 최대 수), `max_features`(각 노드에서 분할에 사용할 특성의 최대 수)가 있습니다. `min_`으로 시작하는 매개변수를 증가시키거나 `max_`로 시작하는 매개변수를 감소시키면 모델에 규제가 커집니다.⁹

[그림 6-3]은 (5장에서 소개한) moons 데이터셋에 훈련시킨 두 개의 결정 트리를 보여줍니다. 왼쪽 결정 트리는 기본 매개변수를 사용하여 훈련시켰고(즉, 규제가 없습니다), 오른쪽 결정 트리는 `min_samples_leaf=4`로 지정하여 훈련시켰습니다. 왼쪽 모델은 확실히 과대적합되었고 오른쪽 모델은 일반화 성능이 더 좋을 것 같아 보입니다.

⁹ 옮긴이_ 사이킷런 0.19에서는 이 외에도 분할로 얻어질 최소한의 불순도 감소량을 지정하는 `min_impurity_decrease`와 분할 대상이 되기 위해 필요한 최소한의 불순도를 지정하는 `min_impurity_split`가 추가되었습니다. `DecisionTreeRegressor`에도 동일한 매개변수가 있습니다.

그림 7-10 예측기가 부족한 경우(왼쪽)와 너무 많은 경우(오른쪽)의 GBRT 양상을



최적의 트리 수를 찾기 위해서는 조기 종료 기법(4장 참조)을 사용할 수 있습니다. 간단하게 구현하려면 `staged_predict()` 메서드를 사용합니다. 이 메서드는 훈련의 각 단계(트리 하나, 트리 두 개 등)에서 양상을에 의해 만들어진 예측기를 순회하는 반복자^{iterator}를 반환합니다. 다음 코드는 120개의 트리로 GBRT 양상을 훈련시키고 최적의 트리 수를 찾기 위해 각 훈련 단계에서 검증 오차를 측정합니다. 마지막에 최적의 트리 수를 사용해 새로운 GBRT 양상을 훈련시킵니다.

```

import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

X_train, X_val, y_train, y_val = train_test_split(X, y)

gbrt = GradientBoostingRegressor(max_depth=2, n_estimators=120)
gbrt.fit(X_train, y_train)

errors = [mean_squared_error(y_val, y_pred)
          for y_pred in gbrt.staged_predict(X_val)]
bst_n_estimators = np.argmin(errors)

gbrt_best = GradientBoostingRegressor(max_depth=2, n_estimators=bst_n_estimators)
gbrt_best.fit(X_train, y_train)

```

[그림 7-11]의 왼쪽은 검증 오차이고 오른쪽은 최적 모델의 예측입니다.

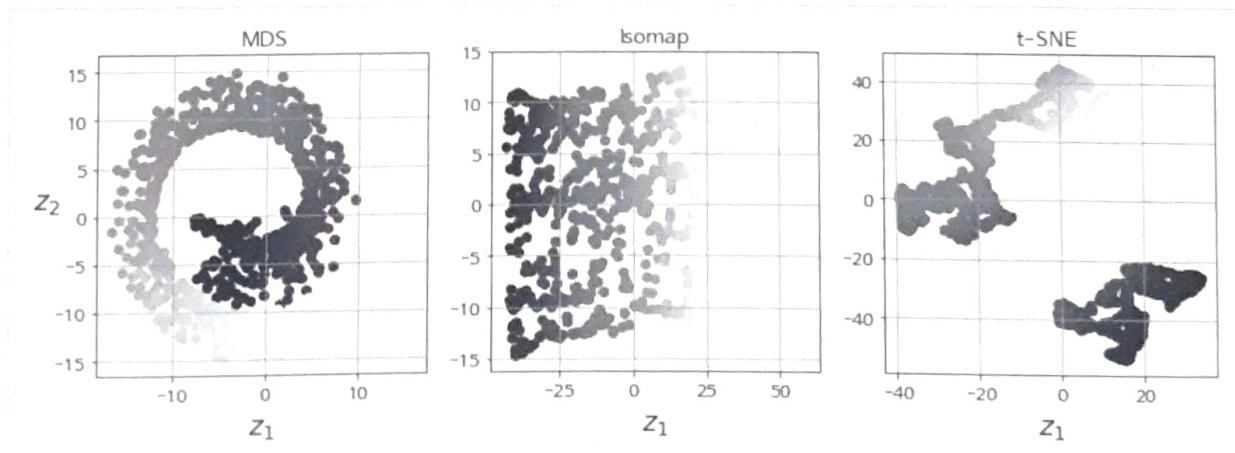
불행하게도 마지막 항의 m^2 때문에 이 알고리즘을 대량의 데이터셋에 적용하기는 어렵습니다.²⁸

8.6 다른 차원 축소 기법

사이킷런은 다양한 차원 축소 기법을 제공합니다. 다음은 그중에서 가장 널리 사용되는 것들입니다.

- **다차원 스케일링** Multidimensional Scaling (MDS)은 샘플 간의 거리를 보존하면서 차원을 축소합니다(그림 8-13).
- **Isomap**은 각 샘플을 가장 가까운 이웃과 연결하는 식으로 그래프를 만듭니다. 그런 다음 샘플 간의 **지오데식 거리** geodesic distance²⁹를 유지하면서 차원을 축소합니다.
- **t-SNE** t-Distributed Stochastic Neighbor Embedding는 비슷한 샘플은 가까이, 비슷하지 않은 샘플은 멀리 떨어지도록 하면서 차원을 축소합니다. 주로 시각화에 많이 사용되며 특히 고차원 공간에 있는 샘플의 군집을 시각화할 때 사용됩니다(예를 들면 MNIST 데이터셋을 2D로 시각화할 때).
- **선형 판별 분석** Linear Discriminant Analysis (LDA)은 사실 분류 알고리즘입니다. 하지만 훈련 과정에서 클래스 사이를 가장 잘 구분하는 축을 학습합니다. 이 축은 데이터가 투영되는 초평면을 정의하는 데 사용할 수 있습니다. 이 알고리즘의 장점은 투영을 통해 가능한 한 클래스를 멀리 떨어지게 유지시키므로 SVM 분류기 같은 다른 분류 알고리즘을 적용하기 전에 차원을 축소시키는 데 좋습니다.

그림 8-13 여러 가지 기법을 사용해 스위스 롤을 2D로 축소하기



28 옮긴이_ 이 절은 LocallyLinearEmbedding의 method 매개변수가 기본값인 'standard'일 때를 설명하고 있습니다. method 매개변수에 지정할 수 있는 다른 옵션에 대한 자세한 설명은 사이킷런의 API 문서(<https://goo.gl/yms7EH>)와 가이드 문서(<https://goo.gl/3XFAVB>)를 참고하세요.

29 그래프에서 두 노드 사이의 지오데식 거리는 두 노드 사이의 최단 경로를 이루는 노드의 수입니다.