**Chapter 8. Dimensionality Reduction**

1. **The Curse of Dimensionality**
2. **Main Approaches for Dimensionality Reduction** 
   1. Projection
   2. Manifold Learning
3. **PCA** 
   1. Preserving the Variance
   2. Principal Components
   3. Projecting Down to d Dimensions
   4. Using Scikit-Learn
   5. Explained Variance Ratio
   6. Choosing the Right Number of Dimensions
   7. PCA for Compression
   8. Incremental PCA
   9. Randomized PCA
4. **Kernel PCA   
   4.1)** Selecting a Kernel and Tuning Hyperparameters
5. **LLE**5.1) Other Dimensionality Reduction Techniques

**Chapter 8. Dimensionality Reduction**

많은 기계 학습 문제는 각 트레이닝 인스턴스에 대해 수천 개 또는 수백만 개의 기능을 포함합니다. 이것은 트레이닝을 극도로 느리게 만들 뿐만 아니라 좋은 솔루션을 찾기가 훨씬 어려워 질 수 있습니다.(차원의 저주 The curse of dimensionality)

현실 세계의 문제에서 기능의 수를 상당히 줄여 다루기 어려운 문제를 취급하기 쉬운 것으로 만들 수 있습니다.(예를 들어, MNIST 이미지)

* 차원을 줄이면 일부 정보가 손실됩니다
* 트레이닝 속도가 빨라지더라도 시스템 성능이 약간 저하 될 수 있습니다.
* 트레이닝이 너무 느리면 차원 감소를 고려하기 전에 먼저 원본 데이터로 시스템을 트레이닝 해야합니다.
* 파이프 라인을 좀 더 복잡하게 만들고 유지 관리하기가 더 어려워집니다.
* 경우에 따라 교육 데이터의 차원을 줄이면 일부 잡음과 불필요한 세부 정보가 필터링되어 더 높은 성능을 얻을 수 있습니다(일반적으로 그렇지는 않지만 교육 속도가 빨라집니다).

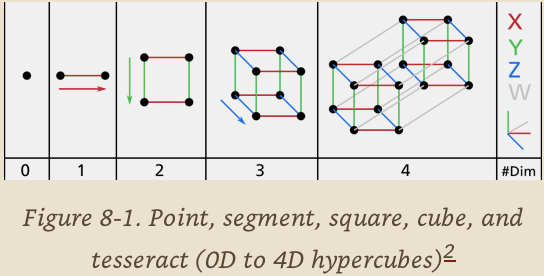
교육 속도를 높이는 것 외에도, 차원 감소는 데이터 시각화에도 매우 유용합니다.

차원 수를 2 개(또는 3 개)로 줄이면 고차원 학습 세트를 그래프에 표시하고 클러스터와 같은 패턴을 시각적으로 감지하여 중요한 통찰력을 얻을 수 있습니다.

이 장에서는 The curse of dimensionality에 대해 논의하고 고차원 공간에서 일어나는 일에 대해 살펴보겠습니다. 그리고 차원 감소(**projection and Manifold Learning**)에 대한 두 가지 주요 접근 방식을 제시하고 가장 널리 사용되는 차원 축소 기술 인 PCA, Kernel PCA 및 LLE를 살펴 보겠습니다.

1. **The Curse of Dimensionality**

우리는 고차원적인 공간을 상상하려고 할 때 3 차원에 너무 익숙합니다. 심지어 기본적인 4D 하이퍼큐브 조차도 1000 차원 공간에서 구부러진 200 차원 타원체는 말할 것도 없고, 우리 머리 속에 그림을 그리는 것이 힘듭니다.



고차원 공간에서는 많은 것들이 매우 다르게 작동한다는 것이 밝혀졌습니다.

임의의 점은 모든 차원에서 "극한"이됩니다

고차원적인 하이퍼큐브의 대부분의 점들은 경계에 매우 가깝습니다.

단위 사각형에서 두 점을 임의로 선택하면 이 두 점 사이의 거리는 평균 약 0.52가됩니다. 단위 3D 큐브에서 임의의 두 점을 선택하면 평균 거리는 약 0.66이됩니다. 그러나 1,000,000 차원 하이퍼 큐브에서 무작위로 추출한 두 점은 무엇입니까? 평균 거리는 408.25일 것입니다. 이것은 매우 직관적이지 못합니다 : 둘 다 같은 단위 하이퍼 큐브에 있을 때 어떻게 두 점을 너무 멀리 떨어뜨릴 수 있습니까? 이 사실은 고차원 데이터 세트가 매우 희박해질 위험이 있음을 의미합니다. 대부분의 교육 인스턴스는 서로 멀리 떨어져 있을 가능성이 큽니다. 물론 이는 또한 새로운 인스턴스가 교육 인스턴스와 멀리 떨어져 있을 가능성이 높다는 것을 의미하며 훨씬 큰 외삽법을 기반으로 하기때문에 하위 차원보다 예측이 훨씬 덜 신뢰할 수 있습니다. 간단히 말해서, 훈련 세트의 차수가 많을수록 overfitting의 위험이 커집니다.

The curse of dimensionality에 대한 한 가지 해결책은 훈련 세트의 크기를 훈련 인스턴스의 충분한 밀도에 도달하도록 증가시키는 것일 수 있습니다. 그러나 실제로는 주어진 밀도에 도달하는데 필요한 훈련 인스턴스의 수는 차원 수에 따라 기하 급수적으로 커집니다. 100개의 feature(MNIST 문제보다 훨씬 적음)를 사용하면, 모든 차원에서 균일하게 분산되었다고 가정하면, 훈련 인스턴스가 평균적으로 서로 0.1 이내가되도록 관찰 가능한 우주의 원자보다 많은 훈련 인스턴스가 필요할 것입니다 .

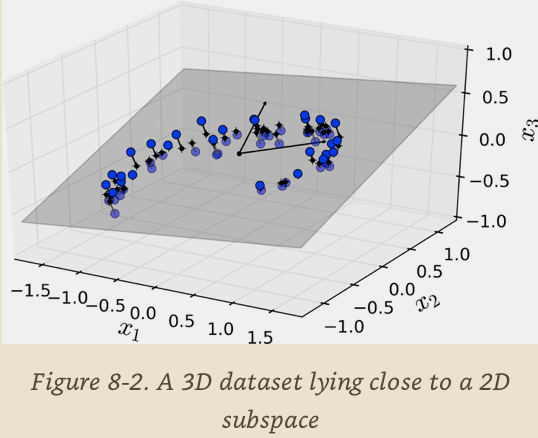
1. **Main Approaches for Dimensionality Reduction**

특정 차원 감소 알고리즘에 대해 알아보기 전에 차수를 줄이기 위한 두 가지 주요 방법인 projection and Manifold Learning을 살펴 보겠습니다.

* 1. **Projection**

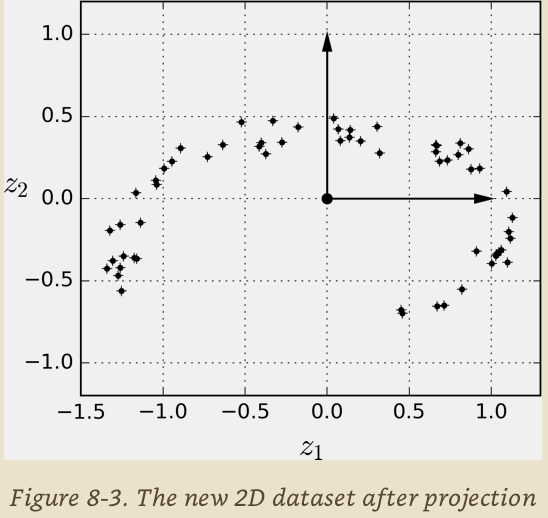
현실 세계의 대부분의 문제에서 트레이닝 인스턴스는 모든 차원에서 균등하게 분산되어 있지 않습니다. 많은 특징들이 거의 일정한 반면, 다른 것들은 고도로 상호. 결과적으로, 모든 트레이닝 인스턴스는 실제로 고차원 공간의 훨씬 더 낮은 차원의 부분 공간 내에 (또는 가까이에) 놓여있다. 이는 매우 추상적으로 들리므로 예제를 살펴 보겠습니다.

그림 8-2에서 원으로 표시된 3D 데이터 세트를 볼 수 있습니다.

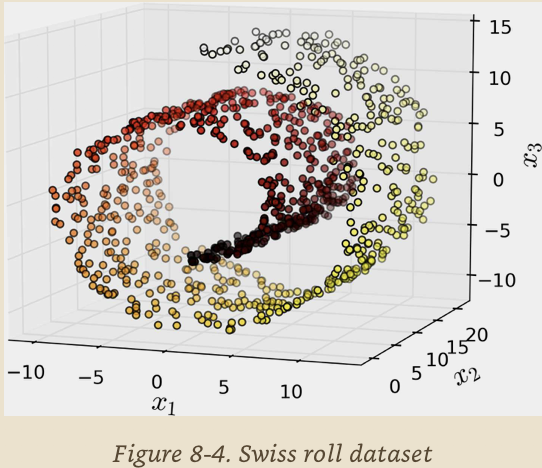


모든 트레이닝 인스턴스가 평면에 가깝게 위치한다는 것을 주목하십시오 : 이것은 고차원 (3D) 공간의 하위 차원 (2D) 부분 공간입니다. 이제 모든 부분을 이 부분 공간에 수직으로 투영하면 (인스턴스를 평면에 연결하는 짧은 선으로 나타남)

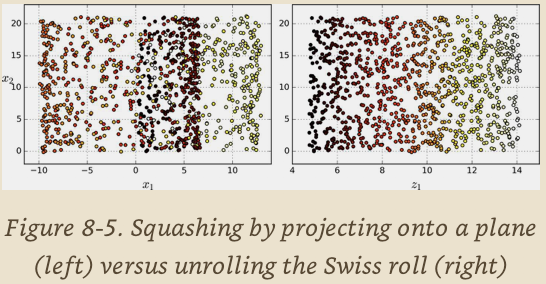
그림 8-3과 같은 새로운 2D 데이터 세트가 생성됩니다. 우리는 방금 데이터 세트의 차원을 3D에서 2D로 줄였습니다. 축은 새로운 feature z1 및 z2 (평면의 투영 좌표)에 해당합니다.



그러나 투영은 항상 차원 감소에 대한 최선의 방법은 아닙니다. 많은 경우에 부분 공간은 그림 8-4에 나와있는 유명한 Swiss roll toy 데이터 세트와 같이 비틀어 회전 할 수 있습니다.



x3을 내fu 평면에 단순히 투영하면 그림 8-5의 왼쪽에 표시된 것처럼 Swiss roll toy의 서로 다른 레이어가 겹치게 됩니다. 그러나 실제로 원하는 것은 그림 8-5의 오른쪽에 있는 2D 데이터 세트를 얻기 위해 스위스 롤을 펼치는 것입니다.



* 1. **Manifold Learning**

Swiss roll toy 은 2D Manifold 의 예입니다. 간단히 말하면, 2D 매니폴드는 고차원 공간에서 구부러지고 꼬여 질 수 있는 2D모양입니다. 더 일반적으로, d 차원 매니폴드는 국부적으로 d 차원 평면과 유사한 n 차원 공간 (d <n)의 일부입니다. 스위스 롤의 경우 d = 2 및 n = 3(2D 평면과 국부적으로 유사하지만 3 차원으로 롤링 됩니다.)

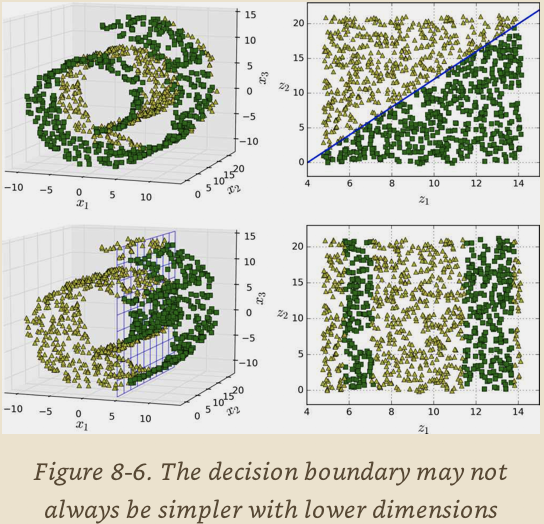
많은 차원 감소 알고리즘은 트레이닝 인스턴스가 있는 매니폴드를 모델링하여 작동합니다. 이것을 매니폴드 러닝 (Manifold Learning)이라고 합니다. 이것은 매니폴드 가설 (Manifold hypothesis)이라고도 불리는 매니폴드 가설에 의거합니다. 이 가설은 현실 세계의 고차원 데이터 세트가 훨씬 더 낮은 차원의 다양성에 가깝다고 말합니다.

MNIST 데이터 세트에 대해 생각해보십시오 : 모든 필기 숫자 이미지는 몇 가지 유사점을 가지고 있습니다. 그것들은 연결된 선들로 이루어져 있고, 테두리는 흰색이고, 중심은 더 많거나 적습니다. 이미지를 무작위로 생성한 경우, 그 중 아주 작은 부분만 손으로 ​​자른 자릿수처럼 보일 것입니다. 다시 말해, 숫자 이미지를 만들려고 할 때 사용할 수 있는 자유도는 원하는 이미지를 생성 할 수 있는 자유도보다 훨씬 낮습니다. 이러한 제약 조건은 데이터 집합을 하위 차원 매니 폴드로 집어 넣는 경향이 있습니다.

[5X5로 나눴을 때 작은 부분을 확대 할때 25개의 모든 값을 가지면 높은 자유도

엣지가 상하로 있다는 것은 위 아래가 같은 밝기를 가질 확률이 큼]

매니폴드 가정은 종종 매니폴드의 더 낮은 차원 공간에서 표현된다면 현재의 작업(예 : 분류 또는 회귀)이 더 간단하다는 또 다른 암시적 가정을 수반한다. 예를 들어, 그림 8-6의 맨 윗줄에서 스위스 롤은 두 개의 클래스로 나뉩니다. 3D 공간 (왼쪽)에서 결정 경계는 상당히 복잡하지만 2D 말려지지 않은 매니폴드 공간의(오른쪽) 결정 경계는 단순한 직선입니다.



그러나 이 가정이 항상 성립되는 것은 아닙니다. 예를 들어, 그림 8-6의 하단 행에서 결정 경계는 x1 = 5에 있습니다. 이 결정 경계는 원본 3D 공간 (수직면)에서 매우 단순 해 보이지만 밀려지지 않은 매니폴드에서는 더욱 복잡해 보입니다(4 개의 독립적 인 선분의 집합).

즉, 모델을 교육하기 전에 교육의 차원을 줄이면 교육 속도가 빨라지지만 항상 더 좋거나 간단한 솔루션으로 이어지지는 않습니다. 그것은 모두 데이터 집합에 따라 다릅니다.

이제는 차원의 저주(The curse of dimensiona)가 무엇인지, 특히 다양성 가정이 성립 할 때 차원 감소 알고리즘이 이를 어떻게 다룰 수 있는지에 대해 잘 알아보았습니다.

이 장의 나머지 부분에서는 가장 널리 사용되는 일부 알고리즘을 살펴 보겠습니다.

1. **PCA**

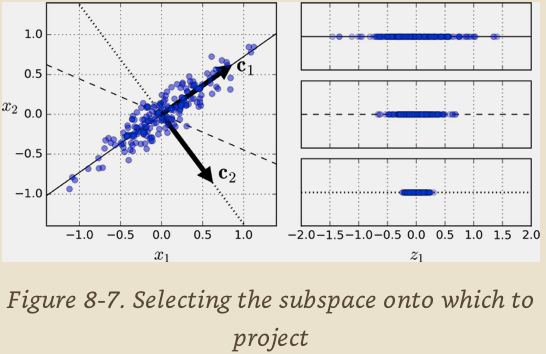
PCA (Principal Component Analysis)는 가장 널리 사용되는 차원 감소 알고리즘입니다.

먼저 데이터에 가장 가까운 초평면을 식별 한 다음 데이터를 그 위에 투영합니다.

[전체 공간보다 차원이 하나 낮은 공간을 초평면이라고 한다. 평면 안에서는 직선이 초평면이고, 삼차원 공간 안에서는 보통의 평면이 초평면이다.]

* 1. **Preserving the Variance**

훈련 세트를 저차원 초평면에 투영하기 전에 먼저 올바른 초평면을 선택해야합니다. 예를 들어, 단순한 2D 데이터 세트가 그림 8-7의 왼쪽에 세 개의 다른 축 (일차원의 초평면)과 함께 표시됩니다. 오른쪽은 각 축에 데이터 세트를 투영 한 결과입니다. 보시다시피, 실선으로 투영하면 최대 분산이 유지되지만 점선으로 투영하면 매우 적은 차이가 유지되고 파선으로 투영하면 중간 정도의 차이가 유지됩니다.



가장 큰 차이를 유지하는 축을 선택하는 것이 합리적입니다. 다른 투영보다 정보가 거의 손실되지 않기 때문입니다. 이 선택을 정당화하는 또 다른 방법은 원본 데이터 세트와 그 축으로의 투영 사이의 평균 제곱 거리를 최소화하는 축입니다.

이것은 PCA의 간단한 아이디어 입니다.

[c1 축과 c2축을 기준으로 점들을 정사영할 때 그 사영거리를 최소화하는 축을 선택

거리가 크면 오류가 크다]

* 1. **Principal Components**

PCA는 훈련 세트에서 가장 큰 분산을 설명하는 축을 식별합니다.(그림 8-7에서는 실선)

또한 첫 번째 축과 직각을 이루는 가장 큰 양의 나머지 분산을 설명하는 두 번째 축을 찾습니다. 이 2D 예제에서는 점선입니다. 고차원 데이터 세트인 경우 PCA는 이전 축과 직각을 이루는 세 번째 축과 네 번째, 다섯 번째 축 등 데이터 세트의 차원 수만큼의 축을 찾습니다.

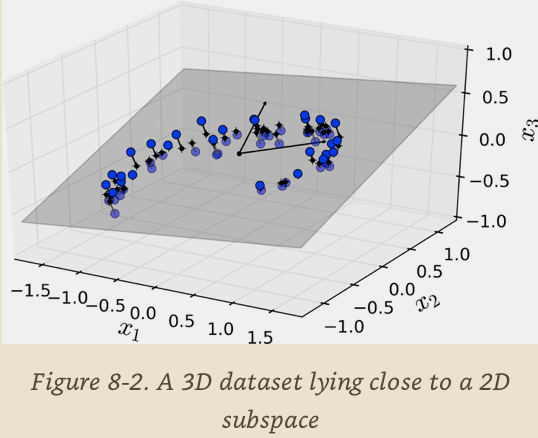
i 번째 축을 정의하는 단위 벡터를 i 번째 주 구성 요소 (PC)라고 합니다.

그림 8-7에서 첫 번째 PC는 c1이고 두 번째 PC는 c2입니다.

그림 8-2에서 첫 두개의 PC는 평면에서 직교 화살표로 표시되며

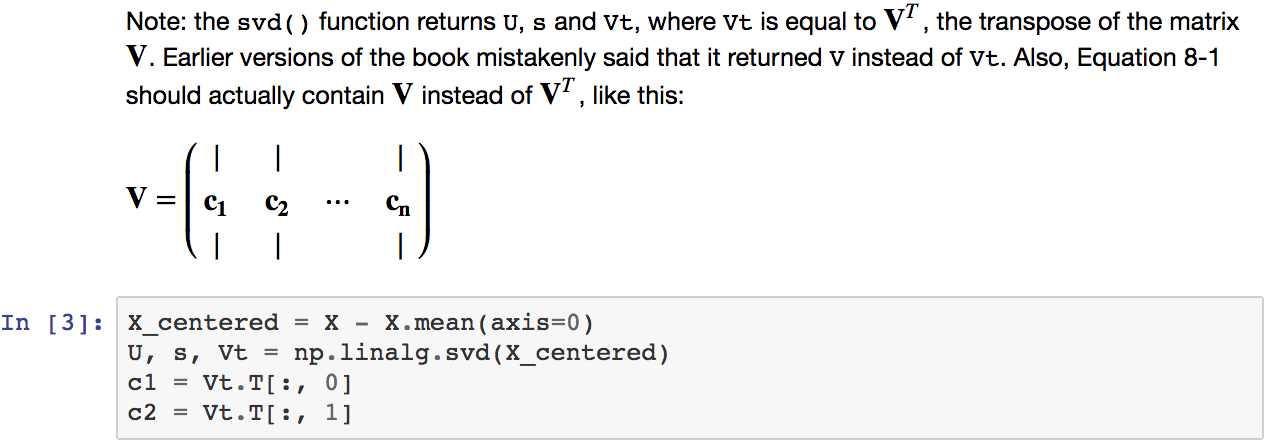
세 번째 PC는 평면과 수직입니다 (위 또는 아래를 가리킴)

[기준이 되는 방향과 가장 먼 방향을 예로 든다 그래서 c1 c2가 직교]



교육 세트의 주요 구성 요소를 어떻게 찾을 수 있을까?

훈련 집합 행렬 X를 3 개의 행렬 U · Σ · VT의 내적으로 분해 할 수 있는 SIGN (Singular Value Decomposition)이라는 표준 행렬 인수 분해 기법이 있습니다. 여기서 V는 우리가 찾고있는 모든 주요 구성 요소를 포함하고 있습니다. (식 8-1).



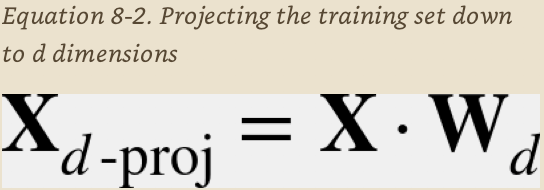
PCA는 데이터 세트가 원점을 중심으로 한다고 가정합니다. 이제 알게 되겠지만, Scikit-Learn의 PCA learning은 여러분을 위해 데이터 센터링을 담당합니다. 그러나 PCA를 직접 구현 한 경우 (앞의 예와 같이) 또는 다른 라이브러리를 사용하는 경우 데이터를 먼저 가운데에 배치하는 것을 잊지 마십시오.

* 1. **Projecting Down to d Dimensions**

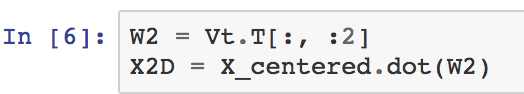
모든 주요 구성 요소를 확인한 후에는 첫 번째 주요 구성 요소로 정의된 초평면에 투영하여 데이터 차원의 차원을 d 차원으로 줄일 수 있습니다. 이 초평면을 선택하면 영사가 가능한 한 많은 차이를 유지하게 됩니다.

예를 들어, 그림 8-2에서 3D 데이터 세트는 처음 두 주 구성 요소에 의해 정의 된 2D 평면으로 투영되어 데이터 세트의 분산의 상당 부분을 보존합니다. 결과적으로 2D 투영은 원래 3D 데이터 세트와 매우 유사하게 보입니다.

트레이닝 세트를 초평면에 투영하려면 트레이닝 집합 행렬 X의 내적을 행렬 Wd로 계산하면 됩니다. 행렬 Wd는 첫 번째 d 주 구성 요소를 포함하는 행렬 (즉, V의 첫 번째 d 열로 구성된 행렬 ), 식 8-2와 같다.



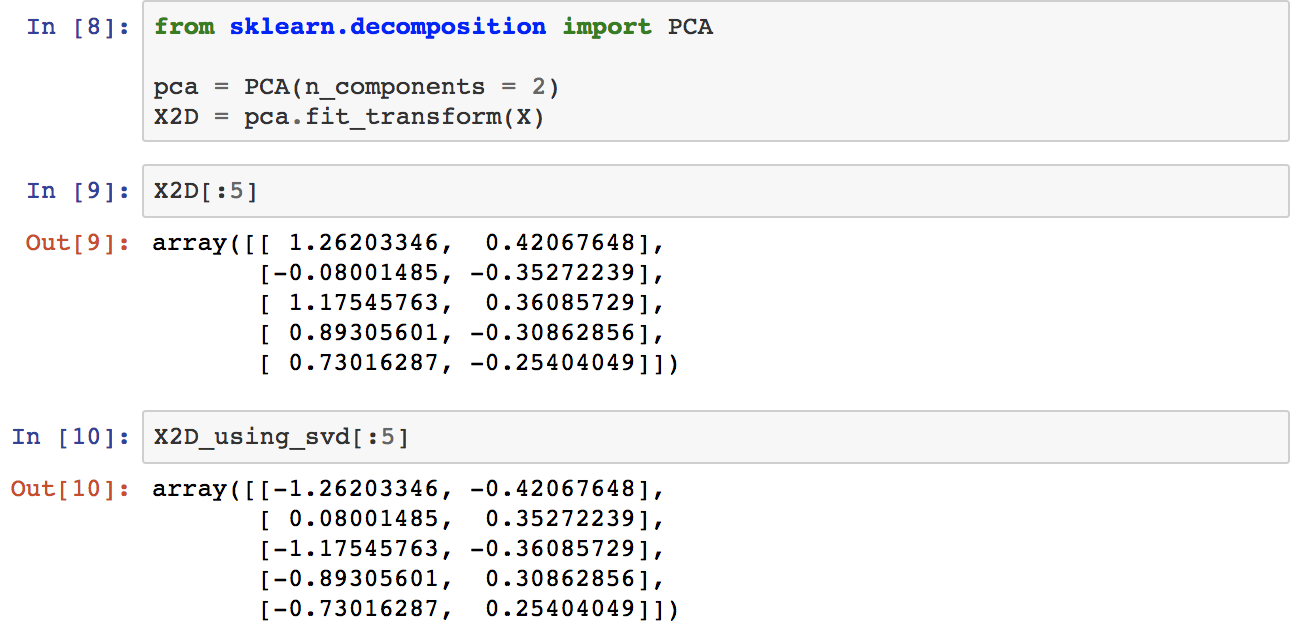
다음 Python 코드는 처음 두 가지 주요 구성 요소로 정의 된 평면에 교육 세트를 투영합니다.



가능한 한 많은 차이를 유지하면서 데이터 집합의 차원을 임의의 차원으로 축소하는 방법을 알았습니다.

* 1. **Using Scikit-Learn**

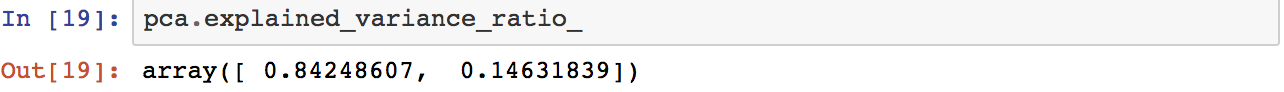
Scikit-Learn의 PCA 클래스는 이전과 마찬가지로 SVD 분해를 사용하여 PCA를 구현합니다. 다음 코드는 데이터 집합의 차원을 2 차원으로 줄이기 위해 PCA를 적용합니다 (데이터 센터링이 자동으로 처리됩니다).



PCA 변환기를 데이터 세트에 맞추면 components\_ 변수를 사용하여 주 구성 요소에 액세스 할 수 있습니다 (PC에 수평 벡터가 포함되어 있으므로 첫 번째 주 구성 요소는 pca.components\_. T [:, 0]와 같다).

* 1. **Explained Variance Ratio**

또 다른 매우 유용한 정보는 explain\_variance\_ratio\_ 변수를 통해 사용 가능한 각 주 구성 요소의 설명된 분산 비율입니다. 각 주 구성 요소의 축을 따라 데이터 집합의 분산 비율을 나타냅니다. 예를 들어 그림 8-2에 표시된 3D 데이터 세트의 처음 두 구성 요소에 대해 설명한 분산 비율을 살펴 보겠습니다.

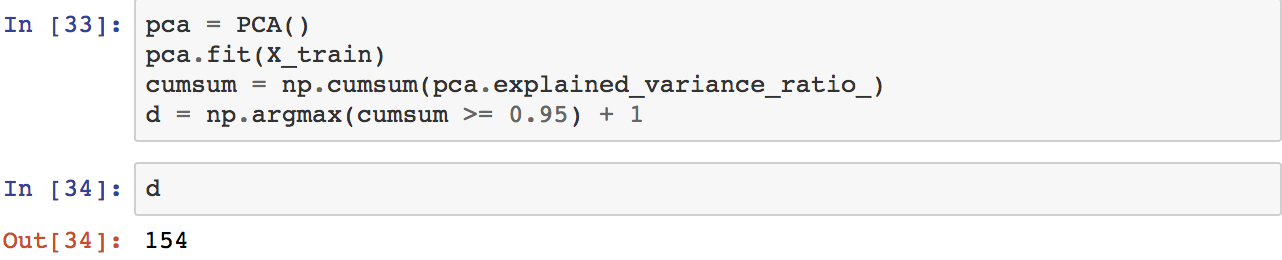


이것은 데이터 세트의 분산의 84.2 %가 첫 번째 축에 있고 14.6 %가 두 번째 축에 놓여 있음을 나타냅니다. 이는 세 번째 축에 대해 1.2 % 미만을 유지하므로 정보가 거의 전달되지 않는다고 가정하는 것이 합리적입니다.

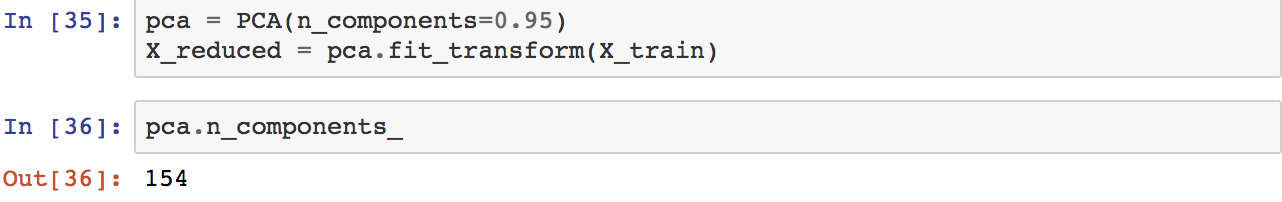
* 1. **Choosing the Right Number of Dimensions**

감소시킬 차수의 수를 임의로 선택하는 대신에, 분산의 충분히 큰 부분(예를 들어, 95 %)까지 합친 치수의 수를 선택하는 것이 일반적으로 바람직하다. 물론 데이터 시각화를 위한 차원을 축소하지 않는 한, 일반적으로 차원을 2 또는 3차원을 사용

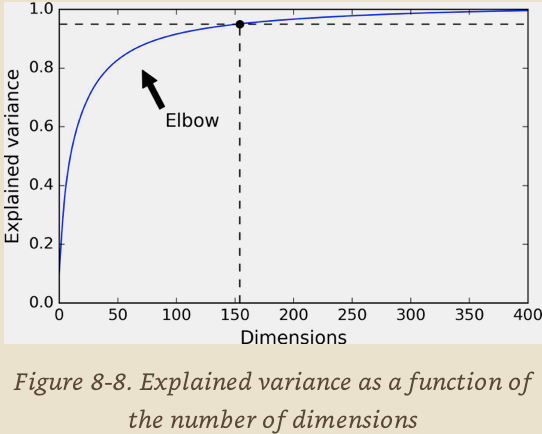
다음 코드는 차원을 줄이지 않고 PCA를 계산 한 다음 훈련 세트의 분산의 95 %를 보존하는 데 필요한 최소 치수 수를 계산합니다.



그런 다음 n\_components = d를 설정하고 PCA를 다시 실행할 수 있습니다. 그러나 훨씬 더 나은 옵션이 있습니다. 보존하려는 주 구성 요소의 수를 지정하는 대신 보존 할 분산의 비율을 나타내는 n\_components를 0.0에서 1.0 사이의 부동 소수점으로 설정할 수 있습니다.



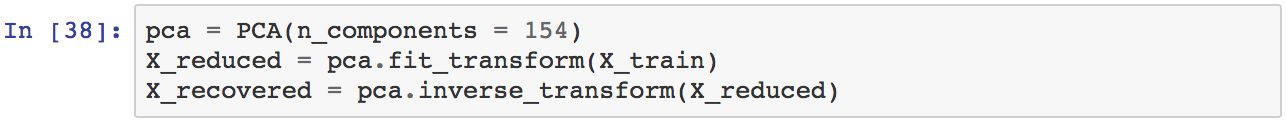
또 다른 옵션은 설명 된 분산을 차원 수의 함수로 나타내기 위한 것입니다 (단순히 그림 cumsum, 그림 8-8 참조). 설명 된 분산이 빠르게 커지는 것을 멈추는 곡선에 보통 팔꿈치가 있습니다. 이것을 데이터 세트의 본질적인 차원으로 생각할 수 있습니다. 이 경우 차원을 약 100 차원까지 줄이면 너무 많은 분산을 잃지 않습니다.

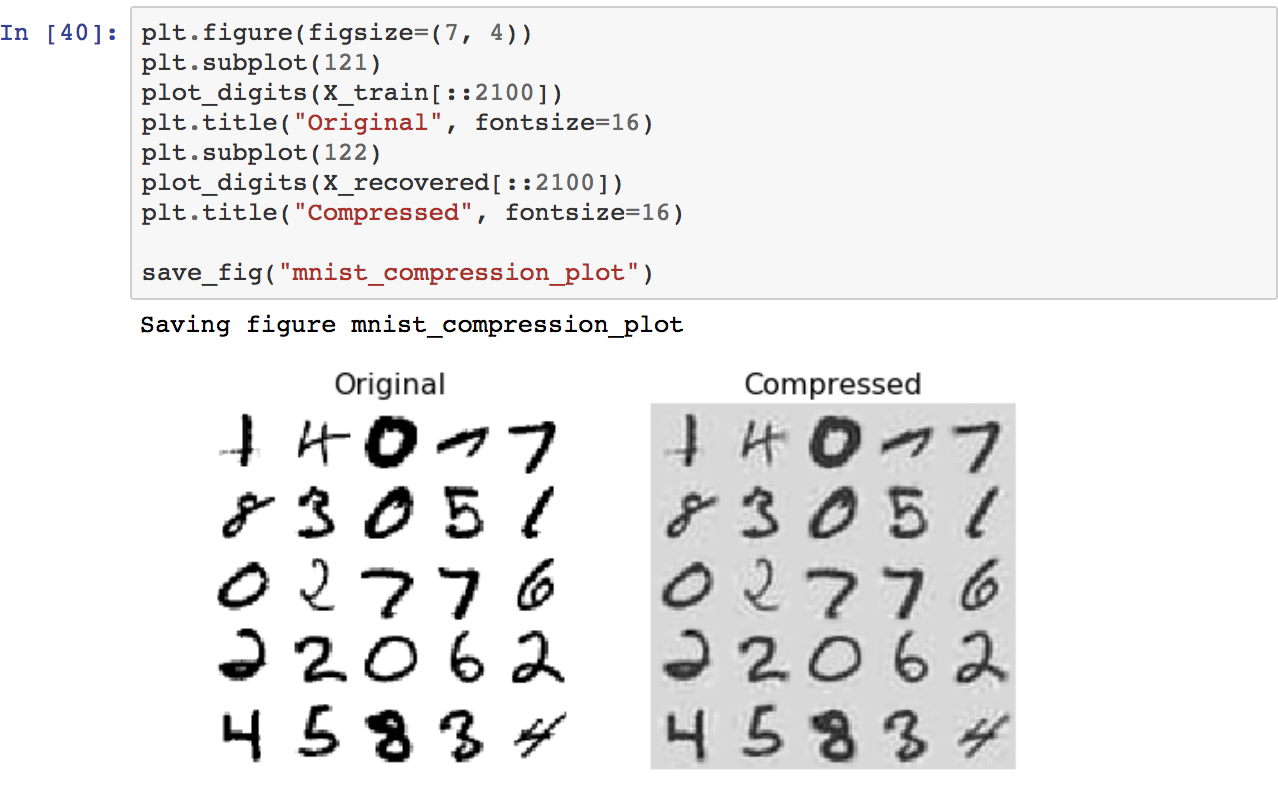


* 1. **PCA for Compression**

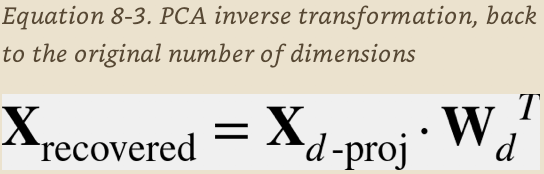
분명히 차원 감소 후에, 트레이닝 세트는 훨씬 적은 공간을 차지합니다. 예를 들어, MNIST 데이터 세트에 PCA를 적용하면서 그 분산의 95 %를 보존하십시오. 각 인스턴스에는 원래 784 기능 대신 150 개가 넘는 기능 만 있습니다. 따라서 대부분의 분산은 보존되지만 데이터 세트는 원래 크기의 20 % 미만입니다! 이는 합리적인 압축 비율이며, 이것이 분류 알고리즘 (예 : SVM 분류 자)의 속도를 대폭 향상시킬 수 있음을 알 수 있습니다.

PCA 투영의 역변환을 적용하여 축소 된 데이터 세트를 다시 784 차원으로 압축을 풀 수도 있습니다. 물론 이것은 투영법이 약간의 정보를 잃어 버렸기 때문에 원래의 데이터를 되돌릴 수는 없지만 원래의 데이터와 매우 유사 할 것입니다. 원래 데이터와 재구성 된 데이터 (압축 된 후 압축 해제 된 데이터) 간의 평균 제곱 거리를 재구성 오류라고 합니다. 예를 들어, 다음 코드는 MNIST 데이터 집합을 154 차원으로 압축 한 다음 inverse\_transform () 메서드를 사용하여이를 다시 784 차원으로 압축 해제합니다. 그림 8-9는 원본 트레이닝 세트의 몇 자리 (왼쪽)와 압축 및 압축 해제 후의 해당 숫자를 보여줍니다. 약간의 화질 손실이 있음을 알 수 있지만 숫자는 여전히 거의 손상되지 않습니다.





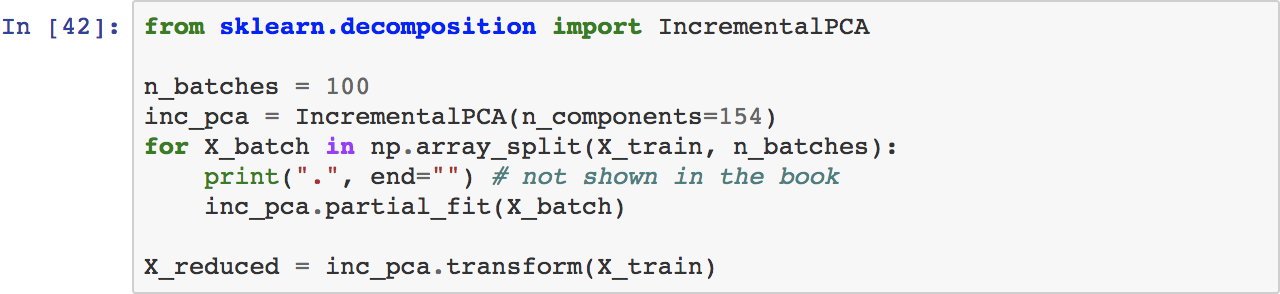
역변환의 공식은 식 8-3과 같습니다.



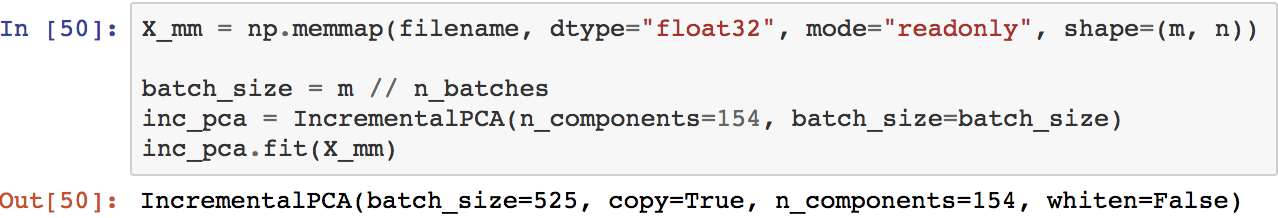
* 1. **Incremental PCA**

앞선 PCA 구현의 한 가지 문제점은 SVD 알고리즘을 실행하기 위해 전체 교육 세트가 메모리에 적합하도록 설정해야한다는 것입니다. 다행히도 Incremental PCA (IPCA) 알고리즘이 개발되었습니다. 교육 세트를 미니 배치로 분할하고 IPCA 알고리즘에 한 번에 하나의 미니 배치를 공급할 수 있습니다. 이는 대규모 교육 세트와 PCA를 온라인 (즉, 새로운 인스턴스가 도착하면 즉석에서)에 적용하는 데 유용합니다.

다음 코드는 MNIST 데이터 세트를 100 개의 미니 배치 (NumPy의 array\_split () 함수 사용)로 분할하고이를 Scikit-Learn의 IncrementalPCA class5에 제공하여 MNIST 데이터 세트의 차원을 이전과 같이 154 차원까지 줄입니다. 전체 트레이닝 세트를 사용하여 fit () 메소드 대신 각 미니 배치로 partial\_fit () 메소드를 호출해야합니다.

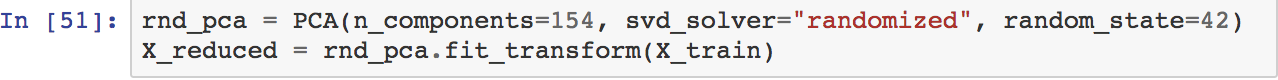


또는 numPy의 memmap 클래스를 사용할 수 있습니다.이 클래스는 메모리에있는 것처럼 디스크의 이진 파일에 저장된 큰 배열을 조작 할 수 있습니다. 클래스는 필요할 때 메모리에 필요한 데이터 만로드합니다. IncrementalPCA 클래스는 주어진 시간에 배열의 작은 부분만을 사용하기 때문에 메모리 사용은 여전히 ​​통제하에 있습니다. 이렇게하면 다음 코드에서 볼 수 있듯이 일반적인 fit () 메서드를 호출 할 수 있습니다.



* 1. **Randomized PCA**

Scikit-Learn은 PCA를 수행하는 또 다른 옵션 인 Randomized PCA를 제공합니다. 이것은 첫 번째 주요 구성 요소의 근사를 빠르게 찾는 확률 적 알고리즘입니다. 계산 복잡도는 O (m × n2) + O (n3) 대신 O (m × d2) + O (d3)이므로 d가 n보다 훨씬 작 으면 이전 알고리즘보다 훨씬 빠릅니다.



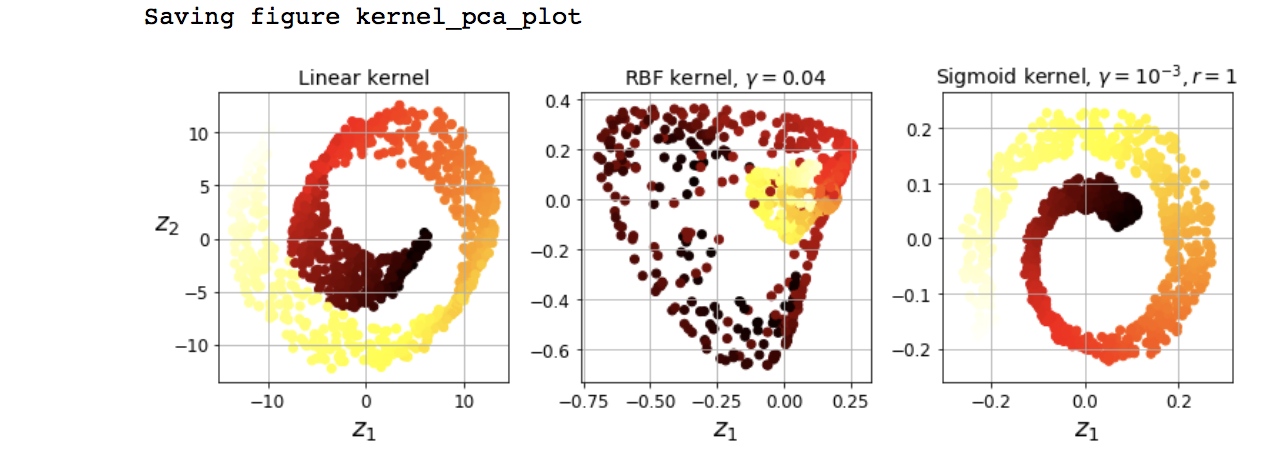
1. **Kernel PCA**

5 장에서는 인스턴스를 매우 고차원 공간 (피쳐 공간이라고 함)으로 암시 적으로 매핑하는 수학적 기법 인 커널 트릭에 대해 설명했으며 지원 벡터 머신을 사용하여 비선형 분류 및 회귀를 가능하게했습니다. 고차원 형상 공간에서의 선형 결정 경계는 원래 공간에서 복잡한 비선형 결정 경계에 해당한다는 것을 상기하자.

동일한 트릭을 PCA에 적용 할 수 있으므로 차원 감소를 위해 복잡한 비선형 예측을 수행 할 수 있습니다. 이를 커널 PCA (kPCA)라고합니다. 투영 후 인스턴스 클러스터를 보존하거나 때로는 뒤틀린 매니 폴드에 가깝게 놓여있는 데이터 세트를 언롤링하는 것이 좋습니다.

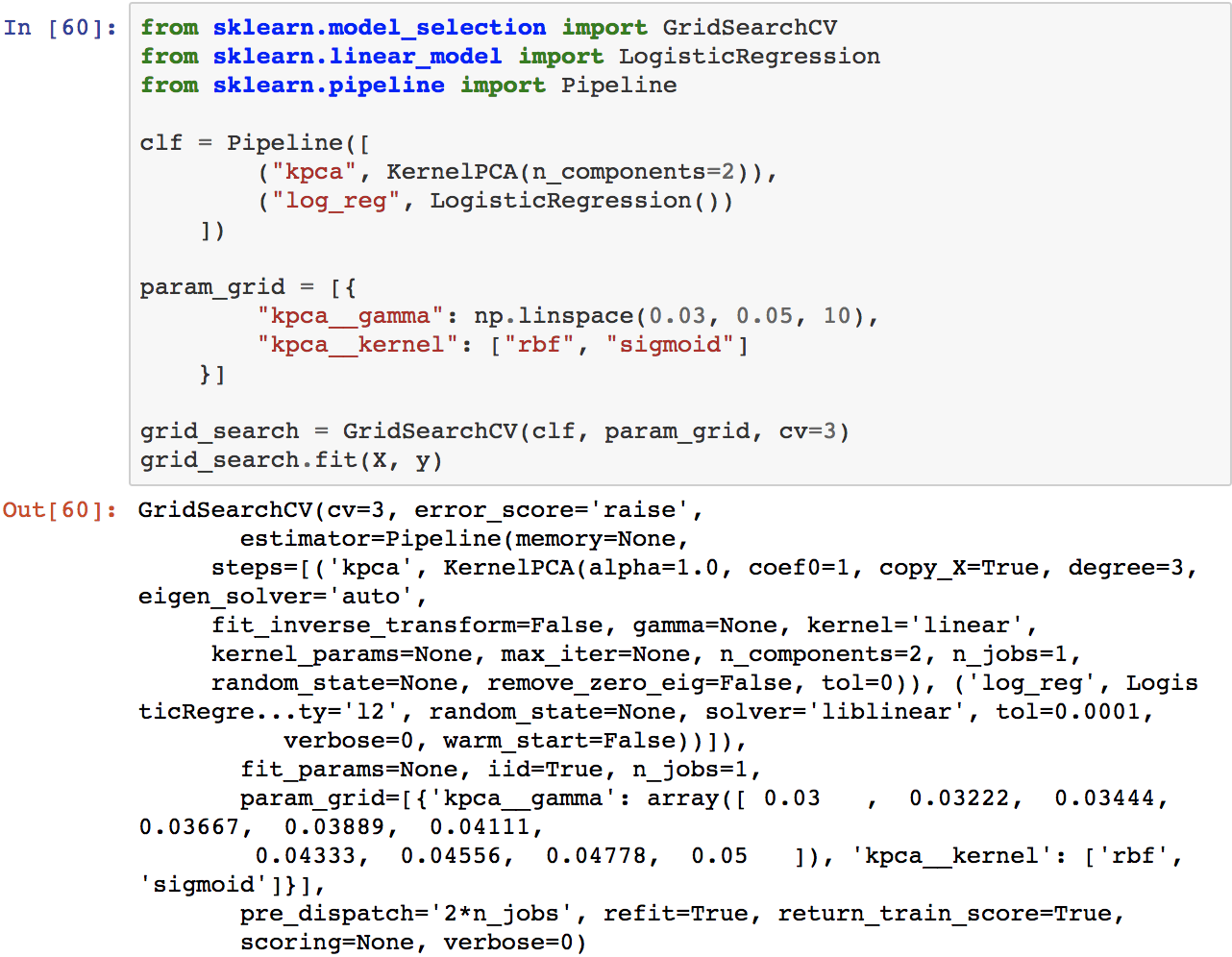
예를 들어 다음 코드는 Scikit-Learn의 KernelPCA 클래스를 사용하여 RBF 커널에서 kPCA를 수행합니다 (RBF 커널 및 기타 커널에 대한 자세한 내용은 5 장 참조).

 그림 8-10은 선형 커널 (간단히 PCA 클래스 사용), RBF 커널 및 시그모이드커널 (Logistic)을 사용하여 2 차원으로 축소 된 스위스 롤을 보여줍니다.

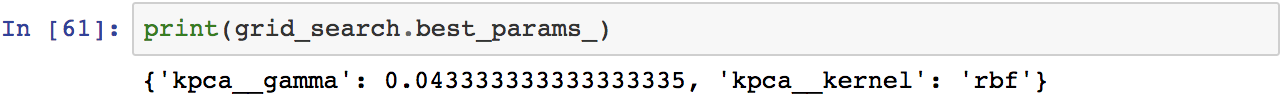


**4.1) Selecting a Kernel and Tuning Hyperparameters**

kPCA는 감독되지 않은 학습 알고리즘이므로 최상의 커널 및 하이퍼 매개 변수 값을 선택할 때 도움이되는 명확한 성능 측정 방법은 없습니다. 그러나 차원 감축은 종종 감독 학습 과제 (예 : 분류)의 준비 단계이기 때문에 그리드 검색을 사용하여 해당 작업에서 최상의 성능으로 이어지는 커널 및 하이퍼 파라미터를 선택할 수 있습니다. 예를 들어 다음 코드는 kPCA를 사용하여 차원을 2 차원으로 먼저 축소 한 다음 분류에 로지스틱 회귀를 적용하여 2 단계 파이프 라인을 만듭니다. 그런 다음 GridSearchCV를 사용하여 kPCA에 대한 최상의 커널 및 감마 값을 찾아서 파이프 라인 끝 부분에서 최상의 분류 정확도를 얻습니다.



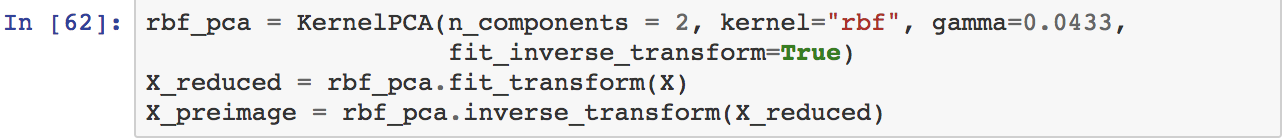
best\_params\_ 변수를 통해 최상의 커널 및 하이퍼 패러미터를 사용할 수 있습니다.



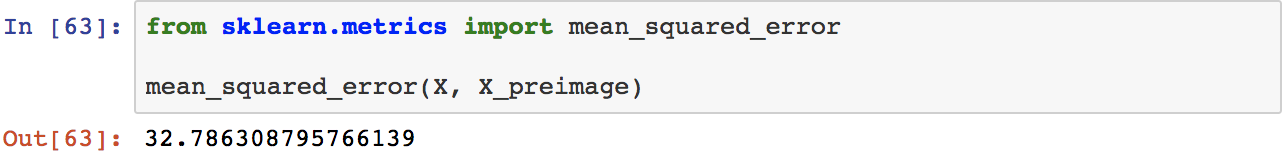
이번에 완전히 감독되지 않은 또 다른 접근법은 가장 낮은 재구성 오차를 산출하는 커널과 하이퍼 파라미터를 선택하는 것입니다. 그러나 재구성은 선형 PCA만큼 쉽지 않습니다. 이유가 여기 있습니다. 그림 8-11은 원래의 스위스 롤 3D 데이터 세트 (왼쪽 위)와 kPCA 후의 결과 2D 데이터 세트를 RBF 커널 (오른쪽 상단)을 사용하여 적용한 것입니다. 커널 트릭 덕분에, 이것은 수학적으로는 피쳐 맵 φ를 사용하여 무한 차원의 피쳐 공간 (오른쪽 하단)에 트레이닝 세트를 매핑한 다음 선형 PCA를 사용하여 변형된 트레이닝 세트를 2D로 투영하는 것과 동일합니다. 축소 된 공간에서 주어진 인스턴스에 대해 선형 PCA 스텝을 반전시킬 수 있다면, 재구성 된 포인트는 원래 공간이 아닌 (예를 들어, 다이어그램에서 x로 표시되는 것과 같은) 특징 공간에 놓여 있음을 주목하십시오. 특징 공간은 무한 차원이기 때문에 재구성 된 점을 계산할 수 없으므로 실제 재구성 오류를 계산할 수 없습니다. 다행히 원래 공간에서 재구성된 점에 가까운 점을 찾을 수 있습니다. 이를 재구성 사전 이미지라고 합니다. 이 사전 이미지를 얻으면 원 인스턴스와의 제곱 거리를 측정 할 수 있습니다. 그런 다음이 재구성 사전 이미지 오류를 최소화하는 커널 및 하이퍼 패러미터를 선택할 수 있습니다.



이 재구성을 수행하는 방법을 궁금해하실 수 있습니다. 한 가지 해결책은 예상 집합을 학습 집합으로, 원래 인스턴스를 대상으로 감독 회귀 모델을 학습하는 것입니다. Scikit-Learn은 다음 코드와 같이 fit\_inverse\_transform = True로 설정하면 자동으로 이 작업을 수행합니다.



그런 다음 재구성 사전 이미지 오류를 계산할 수 있습니다.

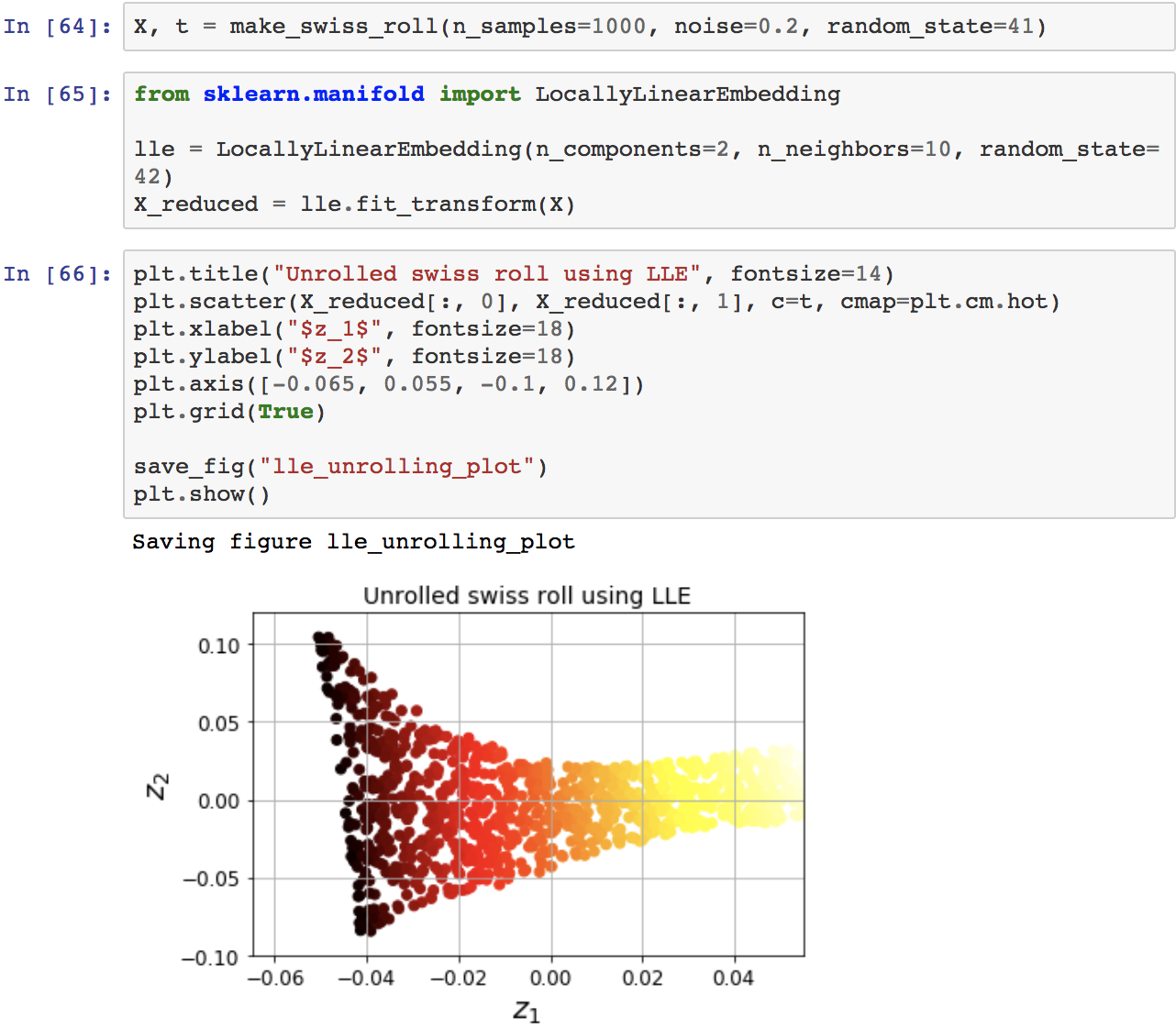


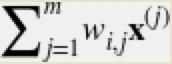
이제 교차 검증을 사용하여 그리드 검색을 사용하여이 사전 이미지 재구성 오류를 최소화하는 커널 및 하이퍼 매개 변수를 찾을 수 있습니다.

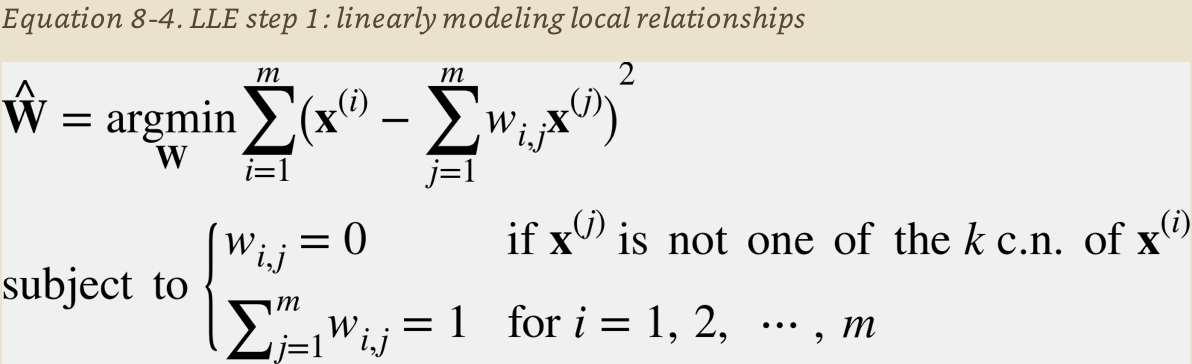
1. **LLE**

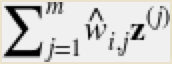
로컬 선형 임베딩 (LLE) 8은 매우 강력한 비선형 차원 감소 (NLDR) 기술입니다. 이전 알고리즘과 같은 투영법에 의존하지 않는 매니 폴드 학습 기법입니다. 간단히 말해서 LLE는 먼저 각 교육 인스턴스가 가장 가까운 이웃과 선형 적으로 관련되어 있는지 측정 한 다음이 로컬 관계가 가장 잘 유지되는 교육 집합의 저 차원 표현을 찾습니다 (자세한 내용은 곧 참조). 특히 너무 많은 소음이없는 경우에는 꼬인 매니 폴드를 풀 때 특히 유용합니다.

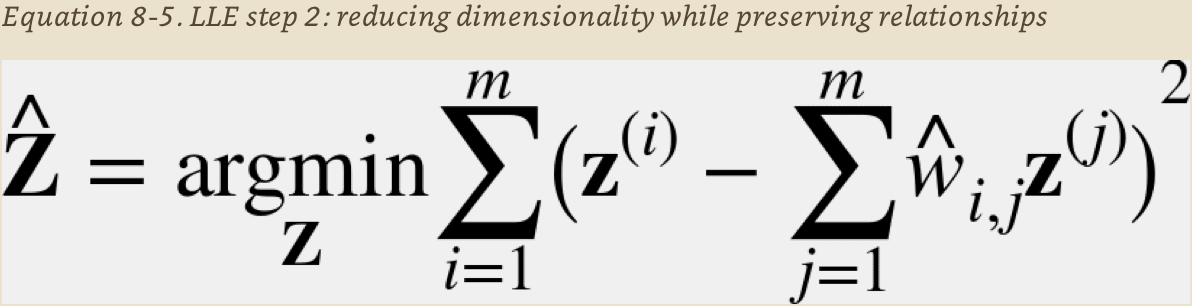
예를 들어, 다음 코드는 Scikit-Learn의 LocallyLinearEmbedding 클래스를 사용하여 스위스 롤을 풀어줍니다. 결과로 얻은 2D 데이터 세트가 그림 8-12에 나와 있습니다. 보시다시피, 스위스 롤은 완전히 펼쳐지고 인스턴스 간의 거리는 국부적으로 잘 보존됩니다. 그러나 거리가 더 크게 유지되지는 않습니다. 즉, 풀린 스위스 롤의 왼쪽 부분이 압착되고 오른쪽 부분이 늘어납니다. 그럼에도 불구하고, LLE는 매니 폴드를 모델링 할 때 꽤 좋은 작업을 수행했습니다.



다음은 LLE의 작동 방식입니다. 먼저, 각 교육 인스턴스 x (i)에 대해 알고리즘이 k 개의 가장 가까운 이웃을 식별하고 (앞의 코드 k = 10에서) x (i)를 이러한 이웃의 선형 함수로 재구성합니다. 더 구체적으로, x (i)와  사이의 제곱 거리가 가능한 한 작게되도록 가중치 wi, j를 찾으며, x (j)가 x의 가장 가까운 이웃 중 하나가 아닌 경우 w 나는). 따라서 LLE의 첫 번째 단계는 식 8-4에 설명 된 제약 최적화 문제입니다. 여기서 W는 모든 가중치 wi, j를 포함하는 가중치 행렬입니다. 두 번째 제약은 단순히 각 트레이닝 인스턴스 x (i)에 대한 가중치를 정규화합니다.



이 단계 후에 가중치 행렬  (가중치 포함)은 학습 인스턴스 간의 로컬 선형 관계를 인코딩합니다. 이제 두 번째 단계는 훈련 인스턴스를 가능한 한 많이 유지하면서 d- 차원 공간 (d <n)으로 매핑하는 것입니다. z (i)가이 d 차원 공간에서 x (i)의 이미지라면 z (i)와  사이의 제곱 거리를 가능한 한 작게해야 합니다. 이 아이디어는 식 8-5에 설명 된 제약되지 않은 최적화 문제를 발생시킵니다. 첫 번째 단계와 매우 유사하게 보이지만, 인스턴스를 고정시키고 최적의 가중치를 찾는 대신, 가중치를 고정시키고 저 차원 공간에서 인스턴스 이미지의 최적 위치를 찾는 것과 반대의 작업을 수행합니다. Z는 모든 z (i)를 포함하는 행렬입니다.



Scikit-Learn의 LLE 구현은 k 개의 가장 가까운 이웃을 찾는 O (m log (m) n log (k)), 가중치를 최적화하기위한 O (mnk3) 및 저조도를 구성하기위한 O (dm2) 차원 표현. 불행하게도, 마지막 기간의 m2는이 알고리즘을 매우 큰 데이터 집합으로 잘못 조정합니다.

**5.1) Other Dimensionality Reduction Techniques**

다른 많은 차원 감소 기술이 있으며, 그 중 일부는 Scikit-Learn에서 사용할 수 있습니다. 다음은 가장 인기있는 몇 가지 예입니다.

* 다차원 스케일링 (Multidimensional Scaling, MDS)은 인스턴스 간의 거리를 유지하면서 차원을 줄입니다 (그림 8-13 참조).
* Isomap은 각 인스턴스를 가장 가까운 이웃에 연결하여 그래프를 만든 다음 인스턴스 사이의 측지 거리(geodesic distances)를 유지하면서 차원을 줄입니다.
* t-SNE (Distributed Stochastic Neighbor Embedding)는 비슷한 인스턴스를 가깝고 다른 인스턴스를 분리하여 유지하면서 차원을 줄입니다. 주로 시각화, 특히 고차원 공간에서 인스턴스의 클러스터를 시각화 (예 : 2D에서 MNIST 이미지를 시각화)하는 데 사용됩니다.
* 선형 판별 분석 (Linear Discriminant Analysis, LDA)은 실제로 분류 알고리즘이지만, 학습하는 동안 클래스간에 가장 차별적 인 축을 학습하고,이 축을 사용하여 데이터를 투영 할 초평면을 정의 할 수 있습니다. 이점은 프로젝션이 클래스를 최대한 멀리 유지할 것이므로 LDA는 SVM 분류 자와 같은 다른 분류 알고리즘을 실행하기 전에 차원을 줄이는 좋은 기술입니다.

