Fəsil 8. Dimensionality Reduction (Boyut Azaldılması)

Bir çox maşın öyrənmə problemləri hər bir nümunə üçün minlərlə, hətta milyonlarla xüsusiyyəti əhatə edir. Bütün bu xüsusiyyətlər nəinki təlimi son dərəcə yavaşladır, həm də görəcəyiniz kimi yaxşı həll tapmağı xeyli çətinləşdiririr. Bu problemə çox vaxt *curse of dimensionality* (boyut lənəti) deyirlər.
Xoşbəxtlikdən, real dünyada baş verən hadisələrdə həlli mümkün olmayan problemi mümkün problemə çevirərək xüsusiyyətlərin sayını əhəmiyyətli dərəcədə azaltmaq olur. Məsələn, MNIST şəkillərini (3-cü Fəsildə təqdim olunmuş) nəzərdən keçirək: təsvirin sərhədlərindəki piksellər demək olar ki, həmişə ağ rəngdədir, ona görə də çox məlumat itirmədən bu pikselləri təlim dəstindən tamamilə çıxara bilərsiniz. Əvvəlki fəsildə gördüyümüz kimi (Şəkil 7-6) bu piksellər təsnifat tapşırığı üçün tamamilə əhəmiyyətsizdir. Bundan əlavə, iki qonşu piksel çox vaxt yüksək korrelyasiyaya malikdir: onları bir pikselə birləşdirsəniz (məsələn, iki piksel intensivliyinin ortasını götürməklə), çox məlumat itirməyəcəksiniz.

> **XƏBƏRDARLIQ**
>
> Şəkli JPEG formatına sıxışdırarkən keyfiyyətin azalması kimi boyutların azadılması da müəyyən məlumat itkisi ilə nəticələnir. Buna görə də o təlimi sürətləndirsə də, sisteminizin işini bir qədər pisləşdirə bilər. O, həmçinin pipeline-ı bir az daha mürəkkəb edir və beləliklə, onlara qulluq etməyi çətinləşdirir. Buna görə də, boyutların azaldılmasından istifadə etməzdən əvvəl ilk növbədə sisteminizi orijinal məlumatlarla train etməniz tövsiyə olunur. Bəzi hallarda, təlim məlumatlarının boyutunun azaldılması bəzi lazımsız detalları süzgəcdən keçirə bilər və beləliklə, daha yüksək performansla nəticələnə bilər, lakin ümumiyyətlə belə bir şey yaşanmayacaq; proses, sadəcə təlimi sürətləndirəcək.

Təlimin sürətləndirilməsi ilə yanaşı, boyutların sayının azaldılması həm də məlumatların vizuallaşdırılması üçün son dərəcə faydalıdır. Boyutların sayının ikiyə (və ya üçə) endirilməsi yüksək ölçülü təlim dəstinin qısaldılmış görünüşünü qrafikdə çəkmək üçün imkan yaradır və çox vaxt klasterlər kimi nümunələri vizual şəkildə aşkar etməklə (detect) bəzi mühüm fikirlər əldə edir. Üstəlik, məlumatların vizuallaşdırılması öz nəticələrinizi alim olmayan insanlara, xüsusən də nəticələrinizdən istifadə edəcək qərar qəbul edənlərə çatdırmaq üçün vacibdir.

Bu fəsildə biz ilk növbədə boyut lənətini (curse of dimensionality) müzakirə edəcəyik və yüksək boyutlu (high-dimensional) sahədə nələr baş verdiyini anlayacağıq. Daha sonra biz boyutların azaldılması (dimensionality reduction) ilə bağlı iki əsas yanaşmanı (proyeksiya və manifold öyrənmə) nəzərdən keçirəcəyik və boyutların azaldılması üçün ən məşhur üç üsulu keçəcəyik: PCA, təsadüfi proyeksiya və yerli xətti yerləşdirmə (LLE).

Boyut lənəti (The Curse of Dimensionality)

Biz üç boyutda yaşamağa o qədər öyrəşmişik ki, yüksək boyutlu məkanı təsəvvür etməyə çalışdığımız zaman intuisiyamız məğlub olur. Hətta adi 4D hiperkubu belə təsəvvürümüzdə canlandırmaq olduqca çətindir (bax Şəkil 8-1), o ki qaldı 1000 ölçülü fəzada əyilmiş 200 ölçülü ellipsoid.

Şəkil 8-1. Nöqtə, seqment, kvadrat, kub və tesserakt (0D - 4D hiperkublar)

Belə çıxır ki, çox-boyutlu məkanda bir çox şey fərqli hərəkət edir. Məsələn, vahid kvadratda (1 × 1 kvadrat) təsadüfi bir nöqtə seçsəniz, onun sərhəddən 0,001-dən az məsafədə yerləşmə şansı təxminən 0,4% olacaq (başqa sözlə, təsadüfi bir nöqtənin hər hansı boyut boyu "ifrat" ("extreme") olması ehtimalı çox azdır). Lakin 10.000 boyutlu vahid hiperkubda bu ehtimal 99,999999%-dən çoxdur. Çox-boyutlu hiperkubun əksər nöqtələri sərhədə çox yaxındır.
Burada daha çətin bir fərq var: vahid kvadratda təsadüfi iki nöqtə seçsəniz, bu iki nöqtə arasındakı məsafə orta hesabla təxminən 0,52 olacaqdır. 3D vahid kubda iki təsadüfi nöqtə seçsəniz, aradakı məsafə orta hesapla təxminən 0,66 olacaq. Bəs 1.000.000 ölçülü vahid hiperkubda təsadüfi seçilmiş iki nöqtə haqqında nə demək olar? Orta məsafə, inanın ya da inanmayın, təxminən 408.25 (təxminən 1.000.0006) olacaq! Bu, ziddiyyətdir: iki nöqtə eyni hiperkubda yerləşdikdə necə bir-birindən bu qədər uzaq ola bilər? Bəli, çox-boyutlu sahələrdə sadəcə bir qədər yer var. Nəticə olaraq, çox-boyutlu verilənlər dəstləri çox seyrək olmaq riski altındadır: əksər təlim nümunələrinin bir-birindən uzaq olma ehtimalları var. Bu həm də o deməkdir ki, yeni nümunə qalan hər hansı təlim nümunəsindən uzaqda olacaq və proqnozları az-boyutlulara nisbətən daha az etibarlı edəcək, çünki onlar daha böyük ekstrapolyasiyalara əsaslanacaqlar. Bir sözlə, təlim dəstinin ölçüləri nə qədər çox olarsa, onun həddən artıq yerləşmə (overfitting) riski də bir o qədər çox olar.
Nəzəri olaraq, boyut lənətinə həll yolu təlim nümunələrinin kifayət qədər sıxlığa çatması üçün təlim dəstinin ölçüsünün artırılması ola bilər. Təəssüf ki, praktikada müəyyən bir sıxlığa çatmaq üçün tələb olunan təlim nümunələrinin sayı ölçülərin sayı ilə eksponent olaraq artır. Cəmi 100 xüsusiyyətlə (MNIST problemindən əhəmiyyətli dərəcədə azdır) hamısı 0-dan 1-ə qədər olmaqla, təlim nümunələrinin orta hesabla bir-birindən 0,1 məsafədə olması üçün müşahidə olunan kainatdakı atomlardan daha çox təlim nümunələrinə ehtiyacınız olacaq, bütün boyutlara bərabər şəkildə yayıldığını güman etsək

Ölçülərin azaldılması üçün əsas yanaşmalar

Ölçülərin azaldılması alqoritmlərinə keçməzdən əvvəl gəlin ölçüləri azaltmaq üçün iki əsas yanaşmaya nəzər salaq: proyeksiya və manifold öyrənmə.

Proyeksiya

Əksər real dünya problemlərində təlim nümunələri bütün boyutlara bərabər şəkildə yayılmır. Bir çox xüsusiyyətlər demək olar ki, sabitdir, digərləri isə yüksək korrelyasiyaya malikdir (əvvəllər MNIST haqqında müzakirə edildiyi kimi). Nəticədə, bütün təlim nümunələri çox-boyutlu məkanın daha az-boyutlu alt fəzasında (və ya ona yaxın) yerləşir. Bu çox mücərrəd səslənir, ona görə də bir nümunəyə baxaq. Şəkil 8-2-də siz kiçik kürələrlə təmsil olunan 3D verilənlər toplusunu görə bilərsiniz.

Şəkil 8-2. 2D alt fəzasına yaxın olan 3D verilənlər toplusu

Diqqət yetirin ki, bütün təlim nümunələri müstəviyə yaxın yerləşir: bu, çox-boyutlu (3D) fəzanın az-boyutlu (2D) alt fəzasıdır. Hər bir təlim nümunəsini bu alt fəzaya perpendikulyar şəkildə proyeksiya etsək (nümunələri müstəvi ilə birləşdirən qısa xətlərlə təmsil olunduğu kimi), Şəkil 8-3-də göstərilən yeni 2D məlumat dəstini alırıq. Ta-da! Biz indicə verilənlər toplusunun ölçüsünü 3D-dən 2D-ə endirdik. Qeyd edək ki, oxlar yeni xüsusiyyətlərə z_1 və z_2 -ə uyğundur: onlar müstəvidəki proyeksiyaların koordinatlarıdır.

Şəkil 8-3. Proyeksiyadan sonra yeni 2D verilənlər toplusu

Manifold öyrənmə (Manifold Learning)

Bununla belə, proyeksiya ölçüləri azaltmaq üçün həmişə ən yaxşı yanaşma hesap olunmur. Bir çox hallarda alt fəza fırlana və dönə bilər, məsələn, Şəkil 8-4-də göstərilən məşhur İsveçrə rulonlu oyuncaq (Swiss roll toy) məlumat toplusunda olduğu kimi.

Şəkil 8-4. İsveçrə rulon məlumat toplusu

Sadəcə olaraq müstəviyə proyeksiya etmək (məsələn, x_3-ü atman) Şəkil 8-5-in sol tərəfində göstərildiyi kimi İsveçrə rulonunun müxtəlif təbəqələrini bir-birinə sıxar. Ancaq çox güman ki, sizin istədiyiniz şey Şəkil 8-5-in sağ tərəfindəki 2D məlumat toplusunu əldə etmək üçün İsveçrə rulonunu açmaqdır.

Şəkil 8-5. Müstəviyə proyeksiya etmək(solda) və İsveçrə rulonunu açmaq (sağda)

İsveçrə rulonu 2D manifolduna bir nümunədir. Sadə dillə desək, 2D manifold daha çox-boyutlu məkanda əyilə və bükülə bilən 2D formatıdır. Daha ümumi şəkildə d-boyutlu manifold yerli olaraq d-boyutlu hipermüstəviyə bənzəyən n-boyutlu fəzanın (burada $d < n$) bir hissəsidir. İsveçrə rulonunda, $d = 2$ və $n = 3$: yerli olaraq 2D müstəviyə bənzəyir, lakin üçüncü ölçüdə yuvarlanır.

Bir çox boyutların azaldılması alqoritmləri təlim nümunələrinin yerləşdiyi manifoldu modelləşdirməklə işləyir; buna manifold öyrənmə deyilir. O, manifold hipotezisi (hypothesis) də adlandırılan manifold fərziyyəsinə əsaslanır ki, bu da real dünyadakı çox-boyutlu məlumat toplularının çoxunun daha az-boyutlu manifolda yaxın olduğunu təsdiqləyir. Bu fərziyyə çox vaxt empirik şəkildə müşahidə olunur.

Bir daha MNIST məlumat dəsti haqqında düşünün: bütün əl ilə yazılmış rəqəmsal şəkillərin bəzi oxşarlıqları var. Onlar bir-birinə bağlı xətlərdən ibarətdir, haşiyələri ağ rəngdədir və az-çox mərkəzləşmişdir. Təsadüfi şəkillər yaratsanız, onların yalnız gülünc dərəcədə kiçik bir hissəsi əlyazma rəqəmlərinə bənzəyəcək. Başqa sözlə, rəqəmsal bir şəkil yaratmağa çalışdığınız zaman əldə edə biləcəyiniz azadlıq, istədiyiniz hər hansı bir görüntü yaratmağa icazə verildiyi təqdirdə əldə etdiyiniz azadlıqdan kəskin şəkildə aşağıdır. Bu məhdudiyyətlər verilənlər bazasını daha aşağı ölçülü manifolda sıxmağa meyllidir.

Manifold fərziyyəsi tez-tez başqa bir gizli (implicit) fərziyyə ilə müşayiət olunur: tapşırıq (məsələn, təsnifat və ya reqressiya) manifoldun az-boyutlu fəzasında ifadə edildiyi təqdirdə daha sadə olacaq. Məsələn, Şəkil 8-6-nın yuxarı cərgəsində İsveçrə

rulonu iki sinfə bölünür: 3D məkanında (solda) qərar sərhədi (decision boundary) kifayət qədər mürəkkəb olardı, lakin 2D açılmamış manifold boşluğunda (sağda) qərar sərhədi düz xəttdir.

Lakin bu gizli fərziyyə həmişə özünü doğrultmur. Məsələn, Şəkil 8-6-nın alt cərgəsində qərar sərhədi $x_1 = 5$-də yerləşir. Bu qərar sərhədi orijinal 3D məkanında (şaquli müstəvidə) çox sadə görünür, lakin açılmamış manifoldda daha mürəkkəb görünür (dörd müstəqil xətt seqmentinin toplusu).

Qısacası, modeli train etməzdən əvvəl təlim dəstinizin boyutunu azaltmaq adətən təlimi sürətləndirəcək, lakin bu, həmişə daha yaxşı və ya daha sadə həllə gətirib çıxarmaya bilər; hamısı verilənlər toplusundan asılıdır.

Ümid edirik ki, indi siz boyut lənətinin nə olduğunu və boyutları azaltma alqoritmlərinin bununla necə mübarizə apara biləcəyini yaxşı başa düşürsünüz, xüsusən də manifold fərziyyəsi qüvvədə olduqda. Bu fəslin qalan hissəsində boyutların azaldılması üçün ən məşhur alqoritmlərdən bəziləri nəzərdən keçiriləcək.

Şəkil 8-6. Qərar sərhədi az-boyutlarda həmişə daha sadə deyil

PCA

Əsas komponent analizi (Principal component analysis) (PCA) boyutların azaldılması üçün ən populyar alqoritmidir. Əvvəlcə o, verilənlərə ən yaxın olan hipermüstəvini (hyperplane) müəyyən edir, sonra isə Şəkil 8-2-də olduğu kimi məlumatları onun üzərinə proyeksiya edir.

Fərqliliyin qorunması

Təlim dəstini daha az-boyutlu hipermüstəviyə proyeksiya etməzdən əvvəl, ilk növbədə düzgün hipermüstəvini seçməlisiniz. Məsələn, Şəkil 8-7-də solda üç müxtəlif ox (yəni, 1D hipermüstəvilər) ilə birlikdə sadə 2D məlumat toplusu təqdim olunur. Sağdakı isə məlumat toplusunun bu oxların hər birinə proyeksiyasının nəticəsidir. Gördüyünüz kimi, bərk xətt üzərində proyeksiya maksimum dispersiyanı (yuxarı), nöqtəli xəttə proyeksiya çox az dispersiyanı (aşağı) və kəsik xəttin üzərindəki proyeksiya aralıq miqdarını (orta) qoruyur.

Şəkil 8-7. Proyeksiya olunacaq alt məkanın seçilməsi

Maksimum fərqlilik miqdarını qoruyan oxu seçmək məqsədəuyğun görünür, çünki o, digər proqnozlara nisbətən daha az məlumat itirəcək. Bu seçimi əsaslandırmağın başqa bir yolu budur ki, bu ox verilənlər toplusu ilə onun bu oxa proyeksiyası arasındakı orta kvadrat məsafəni minimuma endirir. Bu, PCA-ə əsaslanan olduqca sadə fikirdir.

Əsas komponentlər

PCA təlim dəstindəki ən böyük fərq miqdarını hesaba alan oxu müəyyən edir. Şəkil 8-7-də bu düz xəttdir. O, həmçinin birinciyə ortoqonal olan ikinci oxu tapır ki, bu da qalan fərq miqdarının ən böyük qiymətini təşkil edir. Bu 2D nümunəsində başqa seçim yoxdur: bu nöqtəli xəttdir. Daha çox-ölçülü verilənlər toplusu olsaydı, PCA həm də əvvəlki oxlara ortoqonal olan üçüncü ox sonra dördüncü, beşinci və s. - verilənlər bazasındakı boyutların sayı qədər ox tapardı.

i-ci ox verilənlərin i-ci əsas komponenti (PC) adlanır. Şəkil 8-7-də birinci PC c_1 vektorunun, ikinci PC isə c_2 vektorunun yerləşdiyi oxdur. Şəkil 8-2-də ilk iki PC proyeksiya müstəvisində, üçüncü PC isə həmin müstəviyə ortoqonal oxdur. Proyeksiyadan sonra Şəkil 8-3-də birinci PC z_1 oxuna, ikinci PC isə z_2 oxuna uyğun gəlir.

> **NOTE**
>
> Hər bir əsas komponent üçün PCA PC istiqamətinə işarə edən sıfır mərkəzli vahid vektor tapır. İki əks vahid vektor eyni oxda yerləşdiyinə görə, PCA tərəfindən qaytarılan vahid vektorlarının istiqaməti sabit deyil: əgər siz təlim dəstini bir az narahat etsəniz və PCA-nı yenidən işə salsanız, vahid vektorları orijinal vektorlar kimi əks istiqamətə yönələ bilər. Bununla belə, onlar ümumiyyətlə yenə də eyni oxlar üzərində uzanacaqlar. Bəzi hallarda, vahid vektor cütü hətta dönə və ya dəyişdirilə bilər (əgər bu iki ox boyunca fərqlər çox yaxındırsa), lakin onların müəyyən etdiyi müstəvi ümumiyyətlə eyni qalacaq.

Beləliklə, təlim dəstinin əsas komponentlərini necə tapmaq olar? Xoşbəxtlikdən, təlim çoxluğu X matrisini üç U Σ V^T matrisi, bərabərlik 8-1-də göstərildiyi kimi burada V sizin istifadə etdiyiniz bütün əsas komponentləri təyin edən vahid vektorları ifadə edir, matris vurmasına parçalaya bilən tək dəyər parçalanması (singular value decomposition (SVD) adlanan standart matrisin faktorizasiyası texnikası var.

Equation 8-1. Əsas komponentlər matrisi

$$V = | | | c1 c2 \cdots cn | | |$$

Aşağıdakı Python kodu Şəkil 8-2-də göstərilən 3D təlim dəstinin bütün əsas komponentlərini əldə etmək üçün NumPy-nin svd() funksiyasından istifadə edir, sonra ilk iki fərdi PC-ni təyin edən iki vahid vektoru çıxarır:

```
import numpy as np
X = [...] # create a small 3D dataset

X_centered = X - X.mean(axis=0)

U, s, Vt = np.linalg.svd(X_centered)

c1 = Vt[0]

c2 = Vt[1]
```

XƏBƏRDARLIQ

PCA məlumat dəstinin mənbə ətrafında mərkəzləşdiyini güman edir. Gördüyünüz kimi, Scikit-Öyrənmənin PCA dərsləri məlumatların sizin üçün mərkəzləşdirilməsinə diqqət yetirir. Əgər siz PCA-nı özünüz tətbiq edirsinizsə (əvvəlki nümunədə olduğu kimi) və ya başqa library-lərdən istifadə edirsinizsə, əvvəlcə məlumatları mərkəzləşdirməyi unutmayın.

D Ölçülərinə qədər Proyeksiya

Bütün əsas komponentləri müəyyən etdikdən sonra, ilk d əsas komponentləri ilə müəyyən edilmiş hipermüstəviyə proyeksiya edərək məlumat toplusunun boyutlarını d boyutuna qədər azalda bilərsiniz. Bu hipermüstəvinin seçilməsi proyeksiyanın mümkün qədər çox fərqlilik saxlamasını təmin edir. Məsələn, Şəkil 8-2-də 3D verilənlər toplusu verilənlər dəstinin dispersiyasının böyük hissəsini saxlayaraq ilk iki əsas komponent tərəfindən müəyyən edilmiş 2D müstəvisinə qədər proqnozlaşdırılır. Nəticədə, 2D proyeksiya orijinal 3D məlumat toplusuna çox bənzəyir.

Təlim dəstini hipermüstəviyə proyeksiya etmək və d boyutlu azaldılmış $X_{d\text{-}proj}$ verilənlər toplusunu əldə etmək üçün aşağıda göstərildiyi kimi V-nin ilk d sütunlarından təşkil olunan matris kimi təyin olunmuş Wd matrisi ilə təlim dəsti X matrisinin matris vurmasını (multiplication) hesablayın. Bərabərlik 8-2.

Bərabərlik 8-2. Təlimin d boyutlarına qədər proyeksiyası

$$Xd - proj = XWd$$

Aşağıdakı Python kodu təlim dəstini ilk iki əsas komponent tərəfindən müəyyən edilmiş müstəviyə proyeksiya edir:

```
W2 = Vt[:2].T
X2D = X_centered @ W2
```

Budur! İndi siz fərqlənməni mümkün qədər qoruyaraq (preserving), istənilən saylı boyuta qədər proyeksiya etməklə hər hansı bir məlumat toplusunun boyutunu azaltmağın yolunu bilirsiniz.

Scikit-Öyrənmədən istifadə

Scikit-Öyrənmənin PCA sinfi, bu fəsildə əvvəllər etdiyimiz kimi, PCA-nı həyata keçirmək üçün SVD-dən istifadə edir. Aşağıdakı kod, verilənlər toplusunun boyutlarını iki boyuta endirmək üçün PCA tətbiq edir (qeyd edək ki, o, məlumatların mərkəzləşdirilməsinə avtomatik diqqət yetirir):

```
from sklearn.decomposition import PCA

pca = PCA(n_components=2)
X2D = pca.fit_transform(X)
```

PCA transformatorunu verilənlər toplusuna quraşdırdıqdan sonra onun komponentləri_ atributu W_d-nin yerini tutur: o, ilk d əsas komponentlərin hər biri üçün bir sıra ehtiva edir.

İzahlı Fərqlənmə Nisbəti (Ratio)

Başqa bir faydalı məlumat isə hər bir əsas komponentin izahlı fərqlənmə nisbətidir və izahlı_fərqlənmə_nisbəti dəyişəni vasitəsilə əldə edilə bilər. Bu qiymət məlumat toplusunun hər bir əsas komponentdə olan fərqlərinin nisbətini göstərir. Məsələn, Şəkil 8-2-də göstərilən 3D məlumat dəstinin ilk iki komponentinin izahlı fərqlənmə nisbətlərinə baxaq.

```
>>> pca.explained_variance_ratio_
array([0.7578477 , 0.15186921])
```

Bu output bizə məlumat dəstinin təqribən 76%-nin birinci PC-də, təxminən 15%-nin isə ikinci PC-də olduğunu göstərir. Bu o deməkdir ki, üçüncü PC üçün təxminən 9% qalır, buna görə də üçüncü PC-in çox az məlumat daşıdığını güman etmək məqsədəuyğundur.

Boyutların sayının düzgün seçilməsi

Azaltmaq üçün boyutların sayını ixtiyari olaraq seçmək əvəzinə, fərqlənmənin kifayət qədər böyük olduğu bir hissəsini - məsələn, fərqlənməsi 95% -ə çatan saylı boyutları seçmək daha sadədir (əlbəttə ki, bu qaydaya istisna olaraqç siz məlumatların vizuallaşdırılması üçün boyutları azaldırsınız, bu halda siz ölçüləri 2 və ya 3-ə endirmək istəyəcəksiniz).

Aşağıdakı kod MNIST məlumat dəstini yükləyir və bölür (3-cü Fəsildə təqdim olunur) və boyutları azaltmadan PCA-i yerinə yetirir, sonra təlim dəstinin fərqlənməsinin 95%-ni qorumaq (preserve) üçün tələb olunan minimum boyut sayını hesablayır:

```
from sklearn.datasets import fetch_openml

mnist = fetch_openml('mnist_784', as_frame=False)
X_train, y_train = mnist.data[:60_000], mnist.target[:60_000] X_test, y_test = mnist.data[60_000:],
mnist.target[60_000:]

pca = PCA()
pca.fit(X_train)
cumsum = np.cumsum(pca.explained_variance_ratio_)

d = np.argmax(cumsum >= 0.95) + 1 # d equals 154
```

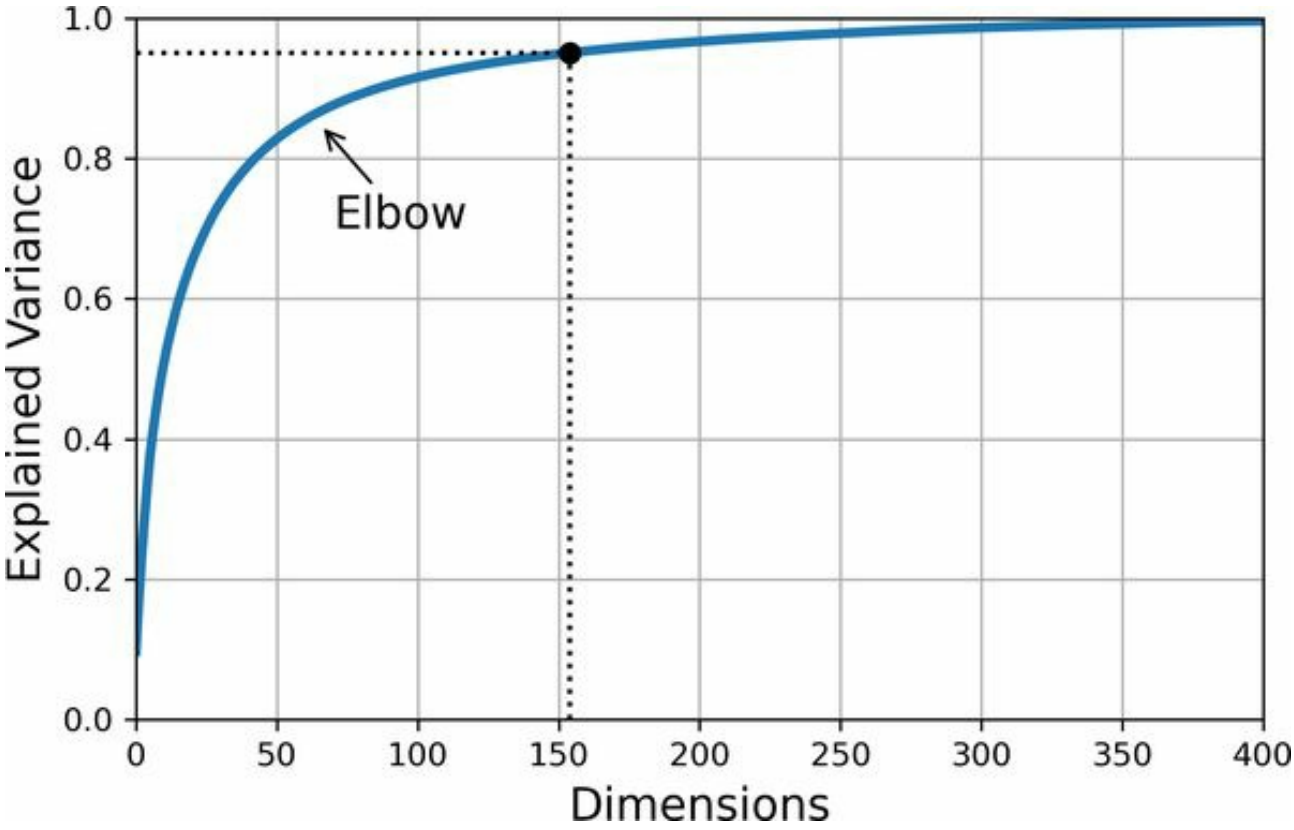
Daha sonra n_components=d təyin edib PCA-nı yenidən işə sala bilərsiniz, lakin daha yaxşı seçim var. Saxlamaq istədiyiniz əsas komponentlərin sayını göstərmək əvəzinə, n_components-i qorumaq istədiyiniz fərqlənmə nisbətini göstərən 0.0 və 1.0 arasında float kimi təyin edə bilərsiniz:

```
pca = PCA(n_components=0.95)
X_reduced = pca.fit_transform(X_train)
```

Komponentlərin faktiki sayı təlim zamanı müəyyən edilir və o, n_components_ atributunda saxlanılır:

```
>>> pca.n_components_
154
```

Başqa bir seçim boyutların sayından asılı olaraq izah edilmiş fərqlənməni çəkməkdir (sadəcə cumsum-ın qrafikini tərtib etmək; Şəkil 8-8-ə baxın). Əyridə adətən bir dirsək (elbow) olacaq, burada izah edilən fərq sürətlə böyüməyi dayandırır. Bu halda siz görə bilərsiniz ki, boyutu təxminən 100-ə endirsəniz, izah edilən dəyişkənliyi itirməyəcəksiniz.



Şəkil 8-8. Fərqlənmə boyutların sayından asılı olaraq izah edilmişdir

Nəhayət, əgər siz nəzarət edilən öyrənmə tapşırığı (supervised learning task) (məsələn, təsnifat) üçün qabaqcadan emal addımı kimi boyutların azaldılmasından istifadə edirsinizsə, onda siz hər hansı digər hiperparametrdə olduğu kimi boyutların sayını tənzimləyə bilərsiniz (2-ci Fəslə baxın). Məsələn, aşağıdakı kod nümunəsi iki addımlı pipeline yaradır, əvvəlcə PCA-dən istifadə edərək boyutları azaldır, sonra təsadüfi meşədən (random forest) istifadə edərək təsnif edir. Sonra, həm PCA, həm də təsadüfi meşə təsnifatı (random forest classifier) üçün hiperparametrlərin yaxşı birləşməsini tapan RandomizedSearchCV-dən istifadə edir. Bu nümunə sürətli axtarış edir, yalnız 2 hiperparametri tənzimləyir (tuning), cəmi 1000 nümunə üzərində məşq edir və cəmi 10 iterasiya üçün çalışır, lakin vaxtınız varsa, daha ətraflı axtarış etməkdən çəkinməyin:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RandomizedSearchCV
 from sklearn.pipeline import make_pipeline
clf = make_pipeline(PCA(random_state=42),
                    RandomForestClassifier(random_state=42))

param_distrib = {
                    "pca__n_components": np.arange(10, 80),

"randomforestclassifier__n_estimators": np.arange(50, 500)
}
rnd_search = RandomizedSearchCV(clf, param_distrib, n_iter=10, cv=3, random_state=42)
```

```
rnd_search.fit(X_train[:1000], y_train[:1000])
```

Tapılan ən yaxşı hiperparametrlərə baxaq:

```
>>> print(rnd_search.best_params_)
{'randomforestclassifier__n_estimators': 465, 'pca__n_components': 23}
```

Komponentlərin optimal sayının nə qədər aşağı olduğunu qeyd etmək maraqlıdır: 784 boyutlu məlumat dəstini cəmi 23 boyuta endirdik! Bu, olduqca güclü bir model olan təsadüfi meşədən (random forest) istifadə etməyimizlə bağlıdır. Əvəzində SGDClassifier kimi xətti modeldən istifadə etsək, axtarış daha çox boyutları (təxminən 70) saxlamalı olduğunu görər.

Sıxılma üçün PCA

Boyutların azaldılmasından sonra təlim dəsti daha az yer tutur. Məsələn, MNIST məlumat toplusuna PCA tətbiq etdikdən sonra onun fərqlənməsinin 95% -ni qoruyaraq, biz orijinal 784 xüsusiyyət əvəzinə 154 xüsusiyyətlə qalırıq. Beləliklə, verilənlər toplusu indi orijinal ölçüsünün 20%-dən azdır və biz onun fərqinin yalnız 5%-ni itirdik! Bu, ağlabatan sıxılma nisbətidir və belə bir boyut azalmasının təsnifat alqoritmini necə sürətləndirəcəyini təxmin etmək asandır.

PCA proyeksiyasının tərs çevrilməsini tətbiq etməklə azaldılmış verilənlər toplusunu 784 boyuta qaytarmaq da mümkündür. Bu, sizə orijinal məlumatları geri verməyəcək, çünki proyeksiya bir az məlumatı itirdi (düşmüş 5% fərq daxilində), lakin çox güman ki, orijinal məlumatlara yaxın olan nəticə aldə edəcəksiniz. İlkin verilənlərlə yenidən qurulan verilənlər arasındakı orta kvadrat məsafəyə yenidənqurma xətası deyilir.

İnverse_transform() metodu bizə azaldılmış MNIST məlumat dəstini 784 boyuta çıxarmağa imkan verir:

```
X_recovered = pca.inverse_transform(X_reduced)
```

Şəkil 8-9 orijinal təlim dəstindən bir neçə rəqəmi (solda) və sıxılmadan sonrakı müvafiq rəqəmləri göstərir. Şəkil keyfiyyətində kiçik bir itki olduğunu görə bilərsiniz, lakin rəqəmlər hələ də əsasən toxunulmazdır (intact).

Şəkil 8-9. Fərqlənmənin 95%-ni saxlayan MNIST sıxılması

Tərs çevrilmə tənliyi bərabırlik 8-3-də göstərilmişdir.

Tənlik 8-3. PCA tərs çevrilmə, ölçülərin orijinal sayına qayıt

$$Xrecovered = Xd - projWd^{\intercal}$$

Təsadüfi PCA

Əgər svd_solver hiperparametrini "təsadüfiləşdirilmiş" olaraq təyin etsəniz, Scikit-Öyrənmə randomizə edilmiş PCA adlı stoxastik alqoritmdən istifadə edir ki, bu da ilk d əsas komponentlərin təxmini qiymətini tez tapır. Tam SVD yanaşmasına görə onun hesablama yolu $O(m \times n^2) + O(n^3)$ əvəzinə $O(m \times d^2) + O(d^3)$ təşkil edir, ona görə də d n-dən müəyyən qədər kiçik olduqda tam SVD-dən kəskin bir şəkildə sürətlidir.

```
rnd_pca = PCA(n_components=154, svd_solver="randomized", random_state=42)
X_reduced = rnd_pca.fit_transform(X_train)
```

TIP

Varsayılan (default) olaraq, svd_solver əslində "avtomatik" olaraq təyin edilmişdir: Scikit-Öyrənmə, əgər max(m, n) > 500 və n_components min(m, n)-in 80%-dən kiçik tam ədəddirsə avtomatik olaraq təsadüfi PCA alqoritmini istifadə edir, əks təqdirdə tam SVD yanaşmasından istifadə edir. Beləliklə, əvvəlki kod $154 < 0{,}8 \times 784$ olduğundan svd_solver="randomized" arqumentini çıxarsanız belə, təsadüfi PCA alqoritmi istifadə olunacaq. Əgər Scikit-Öyrənməni daha dəqiq nəticə əldə etmək üçün tam SVD dən istifadə etməyə məcbur etmək istəyirsinizsə, svd_solver hiperparametrini "tam" olaraq təyin edin.

Artan PCA (Incremental PCA)

PCA-nın əvvəlki tətbiqləri ilə bağlı bir problem, alqoritmin işləməsi üçün bütün təlim dəstinin yaddaşa uyğun olmasını tələb etməsidir. Xoşbəxtlikdən, artan PCA (IPCA) alqoritmləri işlənib hazırlanmışdır ki, bu da sizə təlim dəstini mini-dəstələrə bölməyə və onları bir anda bir mini-dəstədə bəsləməyə imkan verir. Bu, böyük təlim dəstləri üçün və PCA-nın onlayn tətbiqi üçün faydalıdır (yəni, yeni nümunələr gəldikdə).

Aşağıdakı kod MNIST təlim dəstini 100 mini-dəstəyə bölür (NumPy-nin array_split() funksiyasından istifadə etməklə) və əvvəlki kimi MNIST məlumat dəstinin boyutunu 154-ə endirmək üçün onları Scikit-Öyrənmənin IncrementalPCA sinfinə verir. Nəzərə alın ki, bütün təlim dəsti üçün fit() metodundan çox, hər bir mini-dəstə üçün partial_fit() metodunu çağırmalısınız:

```
from sklearn.decomposition import IncrementalPCA

n_batches = 100
inc_pca = IncrementalPCA(n_components=154)

for X_batch in np.array_split(X_train, n_batches):
        inc_pca.partial_fit(X_batch)

X_reduced = inc_pca.transform(X_train)
```

Alternativ olaraq, diskdəki ikili faylda saxlanılan böyük massivi tamamilə yaddaşda olan kimi idarə etməyə imkan verən NumPy-nin memmap sinifindən istifadə edə bilərsiniz; sinif yalnız ehtiyac duyduğu məlumatları yaddaşa yükləyir. Bunu nümayiş etdirmək üçün gəlin əvvəlcə memory-mapped (memmap) faylı yaradaq və MNIST təlim dəstini ona kopyalayaq, sonra hələ də cache-də olan hər hansı məlumatın diskdə saxlanmasını təmin etmək üçün flush() funksiyasına yönələk. Real həyatda X_train adətən yaddaşa sığmaz, ona görə də siz onu hissə-hissə yükləyərdiniz və hər bir parçanı yaddaş massivinin sağ hissəsinə saxlayardınız:

```
filename = "my_mnist.mmap"
X_mmap = np.memmap(filename, dtype='float32', mode='write', shape=X_train.shape) X_mmap[:] = X_train # could be a loop instead, saving the data chunk by chunk

X_mmap.flush()
```

Sonra, memmap faylını yükləyə və onu adi NumPy massivi kimi istifadə edə bilərik. Onun ölçüsünü azaltmaq üçün IncrementalPCA sinifindən istifadə edək. Bu alqoritm istənilən vaxt massivin yalnız kiçik bir hissəsindən istifadə etdiyi üçün yaddaş istifadəsi nəzarət altında qalır. Bu, partial_fit() əvəzinə adi fit() metodunu istifadə etməyə imkan verir ki, bu da olduqca rahatdır:

```
X_mmap = np.memmap(filename, dtype="float32", mode="readonly").reshape(-1, 784) batch_size = X_mmap.shape[0]
// n_batches
inc_pca = IncrementalPCA(n_components=154, batch_size=batch_size) inc_pca.fit(X_mmap)
```

XƏBƏRDARLIQ

Diskdə yalnız binary verilənlər saxlanılır, ona görə də siz onu yükləyərkən massivin məlumat tipini və formasını təyin etməlisiniz. Formanı buraxsanız,

Çox yüksək ölçülü verilənlər dəstləri üçün PCA çox yavaş ola bilər. Daha əvvəl gördüyünüz kimi, təsadüfi PCA-dan istifadə etsəniz belə, onun hesablama mürəkkəbliyi hələ də $O(m \times d^2) + O(d^3)$ olur, ona görə də d ölçülərinin hədəf sayı çox böyük olmamalıdır. Əgər siz on minlərlə və ya daha çox funksiyaya (məsələn, şəkillər) malik verilənlər toplusu ilə məşğul olursunuzsa, onda təlim çox yavaş ola bilər: bu halda, bunun əvəzinə təsadüfi proyeksiyadan istifadə etməyi düşünməlisiniz.

Təsadüfi proyeksiya

Adından da göründüyü kimi, təsadüfi proyeksiya alqoritmi verilənləri təsadüfi xətti proyeksiyadan istifadə edərək daha boyutlu ölçülü fəzaya proyeksiya edir. Bu, çılğın səslənə bilər, lakin belə bir təsadüfi proyeksiyanın əslində məsafələri kifayət qədər yaxşı qoruyacağı ehtimalı böyükdür, bunu məşhur lemmada William B. Conson və Joram Lindenstrauss riyazi olaraq nümayiş etdirdilər. Beləliklə, iki oxşar nümunə proyeksiyadan sonra da oxşar qalacaq, iki çox fərqli nümunə isə proyksiyadan sonra da çox fərqli olaraq qalacaq.

Aydındır ki, nə qədər çox boyut düşürsən, bir o qədər çox məlumat itirilir və daha çox məsafələr təhrif olunur. Beləliklə, ölçülərin optimal sayını necə seçə bilərsiniz? Yaxşı, Conson və Lindenstrauss, məsafələrin müəyyən bir dözümlülükdən çox dəyişməyəcəyini təmin etmək üçün qorunmaq üçün ölçülərin minimum sayını təyin edən bir tənlik yaratdılar. Məsələn, hər biri n = 20.000 funksiyaya malik m = 5.000 nümunədən ibarət məlumat dəstiniz varsa və hər hansı iki instansiya arasındakı kvadrat məsafənin ε = 10% -dən çox dəyişməsini istəmirsinizsə, o zaman məlumatları d ölçülərinə qədər, $d \geq 4 \log(m) / (1/2\ \varepsilon^2 - 1/3\ \varepsilon^3)$ ilə, hansı ki 7300 boyutdur, proyeksiya etməlisiniz. Bu olduqca əhəmiyyətli boyut azalımıdır! Diqqət yetirin ki, tənlik n-dən istifadə etmir, yalnız m və ε-a əsaslanır. Bu tənlik johnson_lindenstrauss_min_dim() funksiyası ilə həyata keçirilir:

```
>>> from sklearn.random_projection import johnson_lindenstrauss_min_dim >>> m, ε = 5_000, 0.1
>>> d = johnson_lindenstrauss_min_dim(m, eps=ε)
>>> d

7300
```

İndi biz [d, n] formalı təsadüfi P matrisini yarada bilərik, burada hər bir element Qauss paylanması sayəsindəç ədədi ortası 0 və fərqlənməsi 1/d olaraq təsadüfi nümunələnir və ondan məlumat dəstini n boyutundan d-ə qədər proyeksiya etmək üçün istifadə olunur:

```
n = 20_000

np.random.seed(42)
P = np.random.randn(d, n) / np.sqrt(d) # std dev = square root of variance

X = np.random.randn(m, n) # generate a fake dataset

X_reduced = X @ P.T
```

Bu qədər! Bu sadədir, səmərəlidir və heç bir təlim tələb olunmur: alqoritmin təsadüfi matris yaratmaq üçün ehtiyac duyduğu yeganə şey verilənlər toplusunun formasıdır. Məlumatların özü ümumiyyətlə istifadə olunmur.

Scikit-Öyrənmə bizim etdiyimizi tam olaraq yerinə yetirmək üçün GaussianRandomProjection sinifini təklif edir: siz onun fit() metodunu istifadə etdiyiniz zaman zaman o, output boyutunu təyin etmək üçün johnson_lindenstrauss_min_dim() istifadə edir, sonra komponentlər_ atributunda saxladığı təsadüfi matris yaradır. Sonra transform() funksiyasına yönləndiyiniz zaman proyeksiyanı yerinə yetirmək üçün bu matrisdən istifadə edir. Transformatoru yaradarkən, siz ε (default olaraq 0.1-ə bərabərdir) cizmaq istəyirsinizsə eps-i, konkret d hədəf boyutunu məcbur etmək istəyirsinizsə, n_components-i təyin edə bilərsiniz Aşağıdakı kod nümunəsi əvvəlki kodla eyni nəticəni verir (həmçinin gaussian_rnd_proj.components_ parametrinin P-yə bərabər olduğunu yoxlaya bilərsiniz):

```
from sklearn.random_projection import GaussianRandomProjection

gaussian_rnd_proj = GaussianRandomProjection(eps=ε, random_state=42)

X_reduced = gaussian_rnd_proj.fit_transform(X) # same result as above
```

Scikit-Öyrənmə həmçinin SparseRandomProjection kimi tanınan ikinci təsadüfi proyeksiya transformatorunu təqdim edir. O, eyni şəkildə hədəf boyutu müəyyən edir, eyni formalı təsadüfi matris yaradır və proyeksiyanı eyni şəkildə yerinə yetirir. Əsas fərq təsadüfi matrisin seyrək olmasıdır. Bu o deməkdir ki, daha az yaddaş istifadə edir: əvvəlki misalda demək olar ki, 1,2 GB əvəzinə təxminən 25 MB! Həm də təsadüfi matris yaratmaq və boyutları azaltmaq üçün daha sürətli: bu məsələdə təxminən 50% daha sürətli. Bundan əlavə, əgər giriş seyrəkdirsə, transformasiya onu seyrək saxlayır (dense_output=True təyin etməsəniz). Nəhayət, əvvəlki yanaşma ilə eyni məsafəni qoruyan xüsusiyyətə malikdir və boyutların azaldılması keyfiyyəti müqayisə edilə bilər. Bir sözlə, adətən birincisi əvəzinə bu transformatordan istifadə eyməyə üstünlük verin, xüsusən də böyük və ya seyrək verilənlər topluları üçün.

Seyrək təsadüfi matrisdəki sıfırdan fərqli elementlərin r nisbətinə onun sıxlığı deyilir. Varsayılan olaraq, 1/n-ə bərabərdir. 20.000 funksiya ilə bu o deməkdir ki, təsadüfi matrisdəki ~141 xanadan yalnız 1-i sıfırdan fərqlidir: bu, olduqca seyrəkdir! İstəyirsinizsə, sıxlıq hiperparametrini başqa bir dəyərə təyin edə bilərsiniz. Seyrək təsadüfi matrisin hər bir xanasının sıfırdan fərqli olma ehtimalı r və hər sıfırdan fərqli qiymət ya –v, ya da +v (hər ikisi eyni dərəcədə mümkündür), burada v = 1/dr.

Əgər tərs çevrilməni həyata keçirmək istəyirsinizsə, əvvəlcə SciPy-nin pinv() funksiyasından istifadə edərək komponentlər matrisinin psevdo-tərsini (pseudo-inverse) hesablamalı, sonra azaldılmış məlumatları psevdo-tersin transpozisiyasına vurmalısınız:

```
components_pinv = np.linalg.pinv(gaussian_rnd_proj.components_)

X_recovered = X_reduced @ components_pinv.T
```

XƏBƏRDARLIQ

Komponentlərin matrisi böyükdürsə, psevdo-tersin hesablanması çox uzun vaxt apara bilər, çünki pinv()-in hesablama mürəkkəbliyi $d < n$ olarsa $O(dn^2)$, əks halda $O(nd^2)$-dir.

Xülasə, təsadüfi proyeksiya sadə, sürətli, yaddaşa qənaət edən və təəccüblü dərəcədə güclü boyutların azaldılması alqoritmidir, onu xüsusən də yüksək ölçülü verilənlər topluları ilə məşğul olarkən yada salmalısınız.

NOTE

Təsadüfi proyeksiya həmişə böyük verilənlər toplusunun boyutlarını azaltmaq üçün istifadə edilmir. Məsələn, Sanjoy Dasgupta və başqalarının 2017-ci il məqaləsi göstərdi ki, meyvə milçəyinin beyni sıx az-boyutlu qoxu girişlərinin seyrək çox-boyutlu binary çıxışlarını xəritələmək üçün təsadüfi proyeksiyanın analoqunu həyata keçirir: hər bir qoxu üçün çıxış neyronlarının yalnız kiçik bir hissəsi aktivləşir, lakin oxşar qoxular bir çoxunu aktivləşdirir. eyni neyronlardan. Bu, oxşar sənədləri qruplaşdırmaq üçün adətən axtarış sistemlərində istifadə edilən lokallığa həssas hashing (LSH) adlı məşhur alqoritmə bənzəyir.

LLE

Yerli xətti daxiletmə (Locally linear embedding) (LLE) qeyri-xətti ölçülərin azaldılması (NLDR) texnikasıdır. Bu, PCA və təsadüfi proyeksiyadan fərqli olaraq, proyeksiyalara etibar etməyən çoxşaxəli öyrənmə texnikasıdır. Bir sözlə, LLE əvvəlcə hər bir məşq nümunəsinin ən yaxın qonşuları ilə xətti əlaqəsini ölçməklə işləyir, sonra isə bu yerli əlaqələrin ən yaxşı şəkildə qorunduğu təlim dəstinin aşağı ölçülü təsvirini axtarır (qısaca daha detallıdır). Bu yanaşma, xüsusilə çox noise olmadıqda, bükülmüş manifoldların açılmasında xüsusilə yaxşı işləyir.

Aşağıdakı kod İsveçrə rulonu yaradır, sonra onu açmaq üçün Scikit-Öyrənmənin LocallyLinearEmbedding sinfindən istifadə edir:

```
from sklearn.datasets import make_swiss_roll
from sklearn.manifold import LocallyLinearEmbedding

X_swiss, t = make_swiss_roll(n_samples=1000, noise=0.2, random_state=42)
lle = LocallyLinearEmbedding(n_components=2, n_neighbors=10, random_state=42)

 X_unrolled = lle.fit_transform(X_swiss)
```

Dəyişən t İsveçrə rulonunun yuvarlanmış oxu boyunca hər bir nümunənin mövqeyini ehtiva edən 1D NumPy massivdir. Biz onu bu nümunədə istifadə etmirik, lakin o, qeyri-xətti reqressiya tapşırığı üçün hədəf kimi istifadə edilə bilər.

Nəticədə 2D verilənlər bazası Şəkil 8-10-da göstərilmişdir. Gördüyünüz kimi, İsveçrə rulonu tamamilə açılıb və nümunələr arasındakı məsafələr yerli olaraq yaxşı qorunub saxlanılır. Bununla belə, məsafələr daha böyük miqyasda qorunmur: açılmış İsveçrə rulonu bu cür uzanmış və bükülmüş bant deyil, düzbucaqlı olmalıdır. Buna baxmayaraq, LLE manifoldun modelləşdirilməsində olduqca yaxşı iş gördü.

Şəkil 8-10. LLE istifadə edərək açılmış İsveçrə rulonu

LLE belə işləyir: hər bir təlim nümunəsi x(i) üçün alqoritm özünün k-yaxın qonşularını müəyyən edir (əvvəlki kodda k = 10), sonra x(i)-ni bu qonşuların xətti funksiyası kimi yenidən qurmağa çalışır. Daha dəqiq desək, $w_{i,j}$ -i elə tapmağa çalışır ki, x(i) ilə $\sum j=1\ m\ w\ i,j\ x\ (j)$ arasındakı kvadrat məsafə mümkün qədər kiçik olsun, əgər $x^{(j)}$, $x^{(i)}$-nin k-yaxın qonşularından biri deyilsə, $w_{i,j} = 0$ götürürük . Beləliklə, LLE-nin ilk addımı 8-4-cü tənlikdə təsvir edilən məhdud optimallaşdırma problemidir, burada W bütün çəkiləri (weights) $w_{i,j}$ ehtiva edən çəki matrisidir. İkinci məhdudiyyət sadəcə olaraq hər bir məşq nümunəsi üçün çəkiləri normallaşdırır x(i).

Tənlik 8-4. LLE addım 1: yerli münasibətlərin xətti modelləşdirilməsi

W ^ = argmin W ∑ i=1 m x (i) -∑ j=1 m w i,j x (j) 2 w i,j = 0-a tabedirsə, əgər x (j) k n.n-dən biri deyilsə. x (i) ∑ j=1 m w i,j = 1 üçün i = 1 , 2 , ⋯ , m

Bu addımdan sonra W ^ çəki matrisi (w^i,j çəkilərindən ibarətdi) təlim nümunələri arasında yerli xətti əlaqələri kodlaşdırır. İkinci addım, bu yerli əlaqələri mümkün qədər qoruyaraq, təlim nümunələrini d-ölçülü fəzaya (burada d < n) uyğunlaşdırmaqdır. Əgər $z^{(i)}$ bu d ölçülü fəzada x(i)-nin şəklidirsə, onda biz $z^{(i)}$ ilə ∑ j=1 m w ^ i,j z (j) arasındakı kvadrat məsafənin mümkün qədər kiçik olmasını istəyirik. . Bu fikir Tənlik 8-5-də təsvir edilən məhdudiyyətsiz optimallaşdırma probleminə gətirib çıxarır. Bu, ilk addıma çox bənzəyir, lakin nümunələri sabit saxlamaq və optimal çəkiləri tapmaq əvəzinə, biz bunun əksini edirik: çəkiləri sabit saxlamaq və nümunələrin şəkillərinin az-boyutlu məkanda optimal mövqeyini tapmaq. Qeyd edək ki, Z bütün $z^{(i)}$-dan ibarət matrisdir.

Bərabərlik 8-5. LLE addım 2: əlaqələri qoruyarkən ölçülərin azaldılması

Z ^ = arqmin Z ∑ i=1 m z (i) -∑ j=1 m w ^ i,j z (j) 2

Scikit-Öyrənmənin LLE tətbiqi aşağıdakı hesablama mürəkkəbliyinə malikdir: k-ən yaxın qonşuları tapmaq üçün O(m log(m)n log(k)), çəkiləri optimallaşdırmaq üçün $O(mnk^3)$ və aşağı səviyyəni qurmaq üçün $O(dm^2)$ -boyutlu təsvirlər. Təəssüf ki, son dövrdə m^2 bu alqoritmi çox böyük verilənlər toplusu üçün zəif miqyaslandırır.

Gördüyünüz kimi, LLE proyeksiya üsullarından tamamilə fərqlidir və əhəmiyyətli dərəcədə daha mürəkkəbdir, lakin o, həmçinin, xüsusən də məlumatlar qeyri-xətti olduqda, daha yaxşı az-boyutlu təsvirlər qura bilər.

Digər Boyutların Azaldılması Texnikaları

Bu fəsli bitirməzdən əvvəl gəlin Scikit-Öyrənmədə mövcud olan bir neçə digər məşhur boyutları azaltma üsullarına qısaca nəzər salaq:

sklearn.manifold.MDS

Çox-boyutlu miqyaslama (Multidimensional Scaling) (MDS) nümunələr arasındakı məsafələri qorumağa çalışarkən boyutları azaldır. Təsadüfi proyeksiya bunu cox-boyutlu məlumatlar üçün edir, lakin az-boyutlu məlumatlarda yaxşı işləmir.

sklearn.manifold.Isomap

Isomap hər bir nümunəni ən yaxın qonşularına birləşdirərək qrafik yaradır, sonra nümunələr arasında geodeziya məsafələrini qorumağa çalışarkən ölçüləri azaldır. Qrafikdəki iki qovşaq arasındakı geodezik (geodesic) məsafə bu qovşaqlar arasında ən qısa yolda olan qovşaqların sayıdır.

sklearn.manifold.TSNE

t-paylanmış stoxastik qonşu yerləşdirmə (t-distributed stochastic neighbor embedding) (t-SNE) oxşar nümunələri yaxın və fərqli nümunələri bir-birindən ayırmağa çalışarkən boyutları azaldır. O, daha çox vizuallaşdırma üçün, xüsusən də çox-boyutlu məkanda nümunə qruplarını vizuallaşdırmaq üçün istifadə olunur. Məsələn, bu fəslin sonundakı məşqlərdə siz MNIST şəkillərinin 2D xəritəsini vizuallaşdırmaq üçün t-SNE-dən istifadə edəcəksiniz.

sklearn.discriminant_analysis.LinearDiscriminantAnalysis

Xətti diskriminant analizi (Linear discriminant analysis) (LDA) xətti təsnifat alqoritmidir ki, təlim zamanı siniflər arasında ən diskriminativ oxları öyrənir. Bu baltalar daha sonra məlumatların proyeksiya ediləcəyi hiperplanı müəyyən etmək üçün istifadə edilə bilər. Bu yanaşmanın üstünlüyü ondan ibarətdir ki, proyeksiya sinifləri mümkün qədər bir-birindən uzaq tutacaq, ona görə də LDA başqa təsnifat alqoritmini işə salmazdan əvvəl boyutları azaldan (yalnız LDA kifayət etmədikdə) yaxşı bir texnikadır.

Şəkil 8-11 İsveçrə rulonunda MDS, Isomap və t-SNE nəticələrini göstərir. MDS qlobal əyriliyini itirmədən İsveçrə rulonunu düzəltməyi bacarır, Isomap isə onu tamamilə buraxır. Aşağı axın vəzifəsindən asılı olaraq, genişmiqyaslı strukturun qorunması yaxşı və ya pis ola bilər. t-SNE, bir az əyriliyi qoruyaraq, İsveçrə rulonunu hamarlamaq üçün ağlabatan bir iş görür və eyni zamanda rulonu parçalayaraq

çoxluqları gücləndirir. Yenə də bu, aşağı axın vəzifəsindən asılı olaraq yaxşı və ya pis ola bilər.

Şəkil 8-11. İsveçrə rulonunu 2D-ə endirmək üçün müxtəlif üsullardan istifadə edin

Exercises

1. Məlumat dəstinin boyutunu azaltmaq üçün əsas motivasiyalar hansılardır? Əsas çatışmazlıqlar hansılardır?

2. Boyutların lənəti nədir?

3. Verilənlər toplusunun boyutu azaldılandan sonra əməliyyatı geri qaytarmaq mümkündürmü? Əgər belədirsə, necə? Əgər yoxsa, niyə?

4. PCA xətti olmayan verilənlər toplusunun boyutunu azaltmaq üçün istifadə edilə bilərmi?

5. Tutaq ki, 1000 ölçülü verilənlər toplusunda PCA-nı yerinə yetirirsiniz, izahlı variasiya nisbətini 95%-ə təyin edirsiniz. Nəticə məlumat dəstinin neçə boyutu olacaq?

6. Hansı hallarda müntəzəm PCA, artımlı PCA, randomizə edilmiş PCA və ya təsadüfi proyeksiyadan istifadə edərdiniz?

7. Verilənlər toplusunda boyutların azaldılması alqoritminin işini necə qiymətləndirə bilərsiniz?

8. İki fərqli boyutlu azaldılması alqoritmini birləşdirməyin mənası varmı?

9. MNIST verilənlər dəstini (3-cü fəsildə təqdim olunur) və bölünmüş təlim dəstini və test dəstini yükləyin (təlim üçün ilk 60,000 nümunəni, sınaq üçün qalan 10,000 nümunəni götürün). Verilənlər toplusunda təsadüfi meşə təsnifatını və bunun nə qədər vaxt aparacağını öyrədin, sonra test toplusunda əldə edilən modeli qiymətləndirin. Sonra, 95% izah edilmiş dispersiya nisbəti ilə məlumat dəstinin boyutunu azaltmaq üçün PCA-dən istifadə edin. Azaldılmış verilənlər bazasında yeni təsadüfi meşə təsnifatını hazırlayın və bunun nə qədər vaxt apardığını görün. Məşq daha sürətli idi? Sonra, test dəstindəki təsnifatı qiymətləndirin. Əvvəlki təsnifatçı ilə necə müqayisə olunur? SGDClassifier ilə yenidən cəhd edin. PCA indi nə qədər kömək edir?

10. MNIST məlumat dəstinin ilk 5000 şəklini 2 boyuta endirmək üçün t-SNE-dən istifadə edin və Matplotlib istifadə edərək nəticəni tərtib edin. Hər bir təsvirin

hədəf sinfini təmsil etmək üçün 10 fərqli rəngdən istifadə edərək səpələnmə qrafikindən istifadə edə bilərsiniz. Alternativ olaraq, səpələnmə qrafikindəki hər bir nöqtəni müvafiq nümunənin sinfi ilə (0-dan 9-a qədər rəqəm) əvəz edə bilərsiniz və ya hətta rəqəm şəkillərinin özlərinin kiçildilmiş versiyalarını tərtib edə bilərsiniz (bütün rəqəmləri tərtib etsəniz, vizuallaşdırma çox qarışıq olacaq, ona görə də siz ya təsadüfi bir nümunə çəkməlisiniz, ya da yalnız yaxın məsafədə başqa heç bir nümunə çəkilmədiyi halda nümunə çəkməlisiniz). Yaxşı ayrılmış rəqəm qrupları ilə gözəl vizuallaşdırma əldə etməlisiniz. PCA, LLE və ya MDS kimi digər utların boyazaldılması alqoritmlərindən istifadə etməyə çalışın və nəticədə ortaya çıxan vizualları müqayisə edin.

Bu məşqlərin həlli bu fəslin sonunda, https://homl.info/colab3 ünvanında mövcuddur.

Yaxşı, əgər vaxtı hesablasanız, dörd boyut, nəzəriyyəçisinizsə, bir az çox boyut.
Https://homl.info/30 ünvanında 3D məkana proqnozlaşdırılan fırlanan tesseraktı izləyin. Şəkil Vikipediya istifadəçisi NerdBoy1392 (Creative Commons BY-SA 3.0). Https://en.wikipedia.org/wiki/Tesseract saytından bərpa edilib.
Əyləncəli fakt: kifayət qədər boyutları nəzərə alsanız, tanıdığınız hər kəs, ehtimal ki, ən azı bir boyutda (məsələn, qəhvəsinə nə qədər şəkər qoyduqları) ekstremistdir.
Karl Pearson, “On Lines and Planes of Closest Fit to Systems of Points in Space”, The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science, № 2. 11 (1901): 559– 572.
Scikit-Öyrnəmə David A. Ross et al., “Incremental Learning for Roust Visual Tracking”, International Journal of Computer Vision 77, №-də təsvir olunan alqoritmdən istifadə edir. 1–3 (2008): 125–141.
ε yunan hərfi epsilondur, tez-tez kiçik dəyərlər üçün istifadə olunur.
Sanjoy Dasqupta və başqaları, “A neural algorithm for a fundamental computing problem”, Elm. No 358. 6364 (2017): 793–796.
Sam T. Roweis və Lawrence K. Saul, “Nonlinear Dimensionality Reduction by Locally Linear Embedding ”, Science 290, №. 5500 (2000): 2323–2326

Train a system, model	
Dimensionality reduction	Boyutların azaldılması
Curse of dimensionality	Boyut lənəti
Pipeline	
Training data	
Equation	Bərabərlik
Training set	Təlim dəsti
Dataset	Verilənlər toplusu
Random forest	
Binary	
Noise	