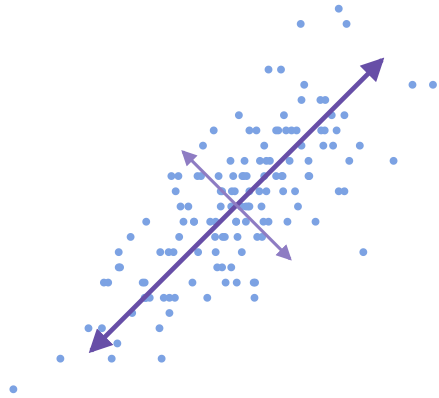


UNIT 02 · CHAPTER 1

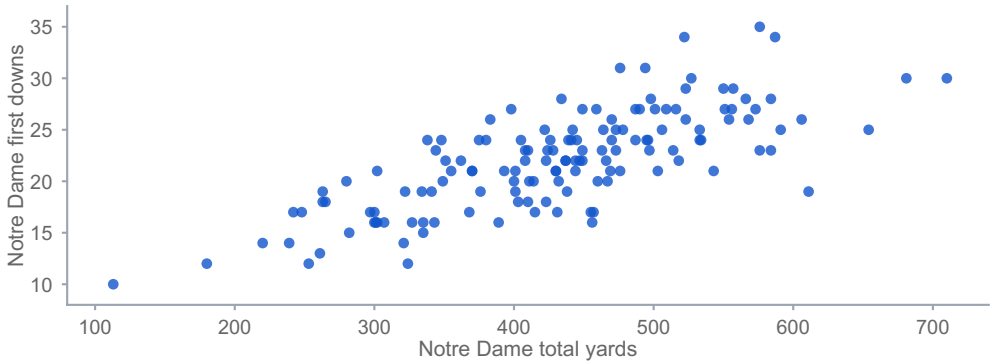
Principal Component Analysis

CSE 30124 · Introduction to Artificial Intelligence
William Theisen · University of Notre Dame

Redundant Features

Last chapter we clustered Notre Dame's games on the whole box score, and one cluster came out at `70` wins and only `4` losses. Unfortunately we couldn't look at it. As we saw in High Dimensional Data, each game is a point with one dimension per stat, `26` of them here.

A lot of those stats tell us the same thing, though. Pictured below is Notre Dame's total yards against its first downs in all `141` games.



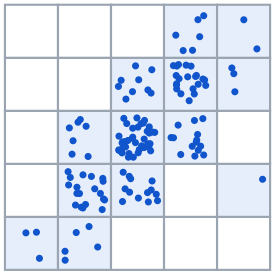
Games where Notre Dame gained a lot of yards are, unsurprisingly, games where it picked up a lot of first downs. We're storing **two columns but only getting a little more than one column's worth of information** out of them. Spotting that in a plot is easy, the harder question is how we get a computer to find the few directions that actually matter in all `26` stats, where we can't draw a plot at all.

Dimensionality Reduction

Fewer columns would help our models too. As we saw in kNN, every stat we add spreads our games thinner (the curse of dimensionality). Pictured below, split total yards into 5 equal ranges and all 5 have a game in them, add first downs and only 13 of the 25 squares do. With all 26 stats every game sits in a square by itself.



total yards: 5 cells, all 5 used

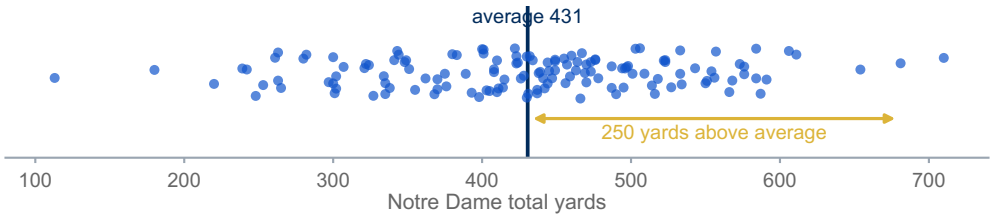


total yards and first downs: 25 cells, 13 used

Squeezing our games into fewer numbers while keeping what makes them different is called **dimensionality reduction**, and like clustering it's **unsupervised learning**.

Variance

To keep "what makes our games different" we first need a way to measure it. Look at our games along a single stat, total yards, and each game is just one number.



The games spread from `113` to `710` yards around an average of `431`, and the `2015` UMass game sits `250` yards above that average. If we square every game's distance from the average and then average the squares, we get the stat's **variance**:

[1.1]

over n games (some books use n − 1)

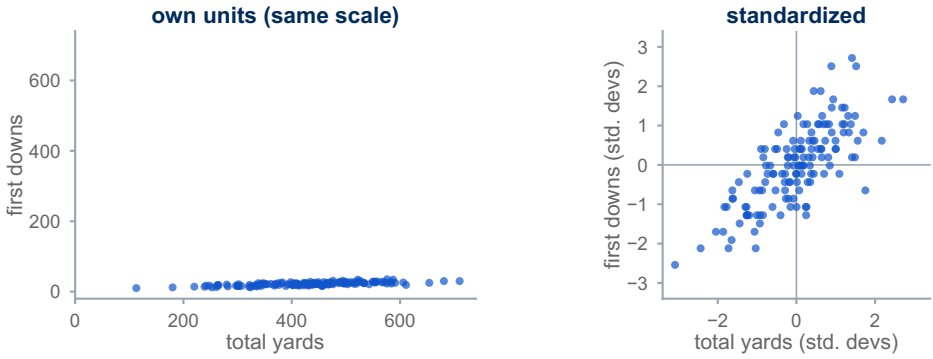
$$\mathrm{Var}(x) = \frac{1}{n}\sum_{i}(x_i - \mu)^2$$

For total yards that's about `10,600`, in yards squared, which is why we usually quote its square root instead (the **standard deviation**, `103` yards).

A stat with no variance would be useless. If Notre Dame gained exactly `400` yards every game, the yards would tell us nothing about which game we were looking at. Generally speaking, **the more a stat varies, the more information it can carry**.

Standardization

Unfortunately, variance depends on units. First downs have a variance of about `23` and total yards about `10,600`. That doesn't make yards `470` times more informative, a yard is just a much smaller unit than a first down. Give both stats the same range (on the left below) and first downs barely leave the floor.



The fix is the standardization we've used since High Dimensional Data, subtracting each stat's average and dividing by its standard deviation. A `534`-yard game becomes `(534 − 431)/103 = 1.0`, one standard deviation above average. Afterwards **every stat has an average of 0 and a variance of exactly 1** (on the right), so no stat gets extra variance just from its units.

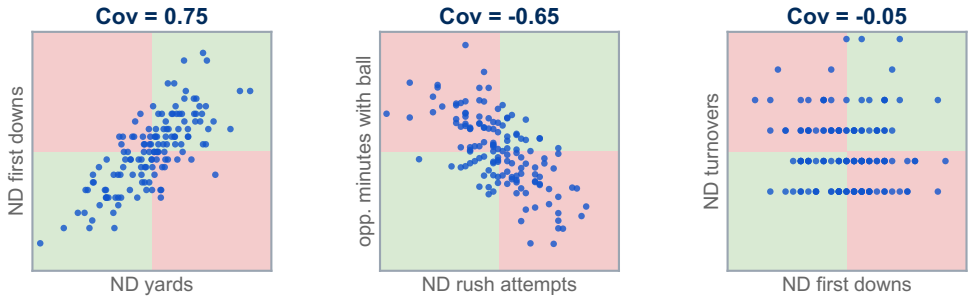
Covariance

Variance looks at one stat at a time, but our problem was two stats moving together. Take the standardized versions and multiply each game's yards by its first downs. A game above average in both (or below in both) gives a positive product, and a game above in one and below in the other gives a negative one.

Averaging those products gives the **covariance** of the two stats:

covariance of two stats

[1.2] $$\mathrm{Cov}(a, b) = \frac{1}{n}\sum_{i}(a_i - \mu_a)(b_i - \mu_b)$$



On the left, almost every game lands in a green quarter and the covariance is `0.75`. In the middle, the more Notre Dame runs the ball the less time the opponent has it (`-0.65`). On the right, first downs and turnovers have almost nothing to do with each other (`-0.05`). We can think of covariance as **how much "informational" overlap two stats have** (for standardized stats it's also called the correlation).

The Covariance Matrix

With 26 stats there's a covariance for every pair, so we collect them all in one table, the **covariance matrix** C. Its diagonal holds each stat's variance (a stat's covariance with itself is just its variance), everything off the diagonal is a covariance, and it's symmetric. For yards and first downs:

[1.3] all 141 games, standardized

$$C = \begin{bmatrix} 1 & 0.75 \\ 0.75 & 1 \end{bmatrix}$$

If we stack our standardized games as the rows of a matrix X (one row per game, one column per stat), one line computes the whole thing:

[1.4] covariance matrix of standardized data

$$C = \frac{1}{n} X^{\mathsf{T}} X$$

The total variance is the sum of the diagonal, $1 + 1 = 2$, and each stat holds half of it. Squaring the covariance ($0.752^2 \approx 0.57$) tells us that about 57% of the game-to-game variation in first downs can be explained by yards alone. That's the overlap we'd like to squeeze out.

Principal Components

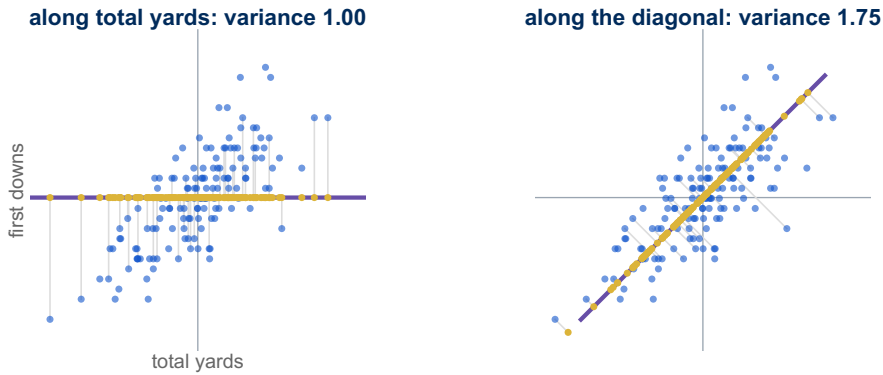
So far a game's position on the yards axis has just been its yards. But nothing forces us to look along the axes we were given. A **direction** is a row v of length 1, and a game's position along it is a dot product, just like a prediction in Linear Regression:

```
[1.5]
```

game x's position along direction v

$$z = x\,v^{\mathsf{T}}$$

Pictured below, every game is dropped straight onto two different directions.



Along total yards the positions have a variance of 1, as they should. Along the diagonal, $v = [0.71, 0.71]$, which adds the two stats together, they have a variance of 1.75. The diagonal sees more of what separates our games than either stat on its own. The direction with the most variance is called the **first principal component** (PC 1), the direction with the most of what's left, at right angles to it, is the second (PC 2), and so on.

Eigenvectors and Eigenvalues

We don't have to measure every direction one at a time to find PC 1. The covariance matrix already knows the variance along any direction:

[1.6] variance of the positions along v

$$\mathrm{Var}(z) = v\,C\,v^T$$

So given C, our goal is to find the v that makes $v\ C\ v^T$ as large as possible, but only among directions of length 1 (or we'd just make v longer). That's an optimization with a constraint, so we use a Lagrange multiplier λ:

variance, with a penalty for leaving length 1

$$L(v, \lambda) = v\,C\,v^T - \lambda\,(v\,v^T - 1)$$

$$\downarrow \frac{\partial L}{\partial v} = 2\,C\,v^T - 2\lambda\,v^T = 0 \quad \text{(C is symmetric), divide by 2}$$

[1.7] the eigenvector equation

$$C\,v^T = \lambda\,v^T$$

Multiplying v by C doesn't turn it, it only stretches it by λ. Directions like that are called the **eigenvectors** of C, and their stretch factors are its **eigenvalues**. Putting [1.7] back into [1.6] gives $\mathrm{Var}(z) = v(\lambda v^T) = \lambda$ (since $v\ v^T = 1$), so **an eigenvalue is the variance along its eigenvector**. That's all PCA is, **the principal components are the eigenvectors of C, sorted by eigenvalue**.

PCA by Hand

Here are six games, one from each of six seasons:

Game	Yards	First downs	Yards (std.)	First downs (std.)	Product
2012 Wake Forest	584	23	1.28	0.73	0.93
2014 Rice	576	23	1.22	0.73	0.89
2015 Clemson	432	20	0.13	0.00	0.00
2018 Vanderbilt	380	24	−0.26	0.97	−0.26
2017 Georgia	265	18	−1.14	−0.49	0.55
2022 Ohio State	253	12	−1.23	−1.94	2.38

Step 1 standardizes. These games average `415` yards and `20` first downs (standard deviations `132` and `4.12`), so Wake Forest's `584` yards become $(584 - 415)/132 = 1.28$.

Step 2 is the covariance matrix. Both columns now have variance `1` and average `0`, so the covariance is the average of the products:

[1.8]

the average of the products

$$\frac{0.93 + 0.89 + 0.00 - 0.26 + 0.55 + 2.38}{6} = 0.748$$

our six games' covariance matrix

$$C = \begin{bmatrix} 1 & 0.748 \\ 0.748 & 1 \end{bmatrix}$$

Eigenvalues by Hand

Step 3 starts with the eigenvalues. Moving everything in [1.7] to one side gives $(C - \lambda I)v^T = 0$, and a direction that isn't zero can only be squashed to zero if $C - \lambda I$ has a determinant of 0. That leaves an equation with only λ in it:

[1.9] solve for λ

$$\det(C - \lambda I) = 0$$

$$\det\begin{bmatrix} 1-\lambda & 0.748 \\ 0.748 & 1-\lambda \end{bmatrix} = 0$$

↓ 2×2 determinant: $ad - bc$

$$(1-\lambda)^2 - 0.748^2 = 0$$

↓ square root of both sides

$$1 - \lambda = \pm 0.748$$

[1.10] one for each new axis

$$\lambda_1 = 1.748 \qquad \lambda_2 = 0.252$$

Notice that they add up to 2, the total variance we started with. **Rotating our axes moves variance around, it never creates or destroys any.** PC 1 now holds $1.748/2 = 87\%$ of it, where yards and first downs held 50% each.

Two stats: with two standardized stats the eigenvalues are always 1 plus and minus the covariance, so this step is quick. With 26 stats it is not.

Eigenvectors by Hand

Next, the direction for each eigenvalue. Put $\lambda_1 = 1.748$ into $(C - \lambda I)v^T = 0$, writing $v_1 = [a, b]$:

$$\begin{bmatrix} -0.748 & 0.748 \\ 0.748 & -0.748 \end{bmatrix}\begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

↓ both rows say the same thing

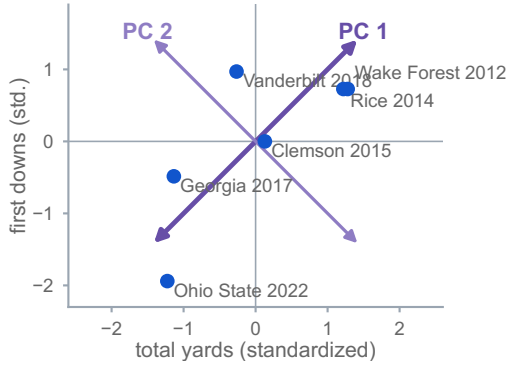
$$a = b$$

↓ scale to length 1

[1.11] PC 1

$$v_1 = \frac{1}{\sqrt{2}}[1,\ 1] \approx [0.71,\ 0.71]$$

Doing the same with $\lambda_2 = 0.252$ gives $a = -b$, so $v_2 \approx [-0.71, 0.71]$.



Notice that $v_1\ v_2^T = -0.5 + 0.5 = 0$, the two directions are at right angles, or **orthogonal**. That's no accident, **a symmetric matrix always has orthogonal eigenvectors**, so no variance gets counted twice.

Projecting onto the Components

Step 4 sorts the components by eigenvalue (already done, PC 1 first) and **step 5** picks how many to keep. With `87%` of the variance on PC 1, let's keep just that one. **Step 6** moves every game into the new axes with `[1.5]`. For Wake Forest, $z = 0.71(1.28) + 0.71(0.73) = 1.42$, and each game is now one number instead of two.

Did it work? We wanted new axes where the stats stop overlapping. Here are all six games on both components, with the product of the two positions:

Game	PC 1	PC 2	Product
2012 Wake Forest	1.42	−0.39	−0.55
2014 Rice	1.38	−0.35	−0.48
2015 Clemson	0.09	−0.09	−0.01
2018 Vanderbilt	0.50	0.87	0.44
2017 Georgia	−1.15	0.46	−0.53
2022 Ohio State	−2.24	−0.50	1.13

This time the products cancel out, they add up to `0`. In the new axes the covariance matrix has `1.748` and `0.252` on its diagonal and zeros everywhere else, so the new features are **uncorrelated**. **All the overlap between yards and first downs has been rotated onto PC 1.** Compare that with the Product column on the PCA by Hand page, where only one game was negative and the covariance came out at `0.748`.

Reconstruction

We kept one number per game. Can we get both stats back? Almost, walking z steps along the direction we kept rebuilds the game:

[1.12] rebuild a game from its position along v

$$\hat{x} = z\,v$$

For Wake Forest, $\hat{x} = 1.42\,[0.71, 0.71] = [1.00, 1.00]$. Its real standardized stats were $[1.28, 0.73]$, so we miss by $[0.28, -0.28]$, which is exactly its part along PC 2, the axis we dropped. Undoing the standardizing (multiply by σ, add μ) gives back about **548** yards and **24.1** first downs. It really had **584** and **23**.

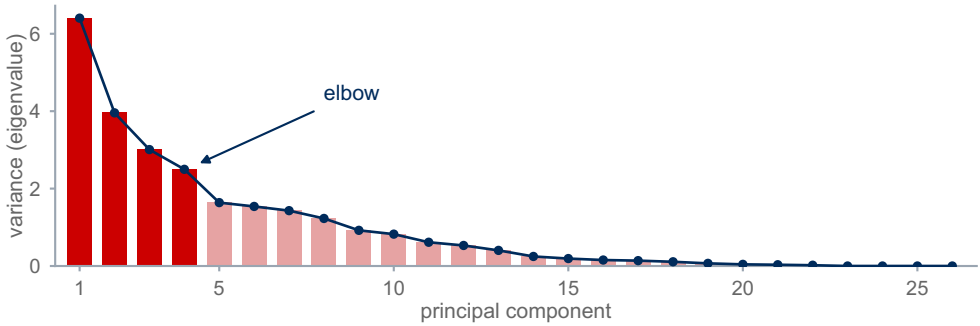
Keep every component and we get every game back exactly, since PCA was only a rotation. Keep fewer and **what we lose is the variance on the components we dropped**, here $\lambda_2 = 0.252$, the missing **13%**.

keeping k components

$$\hat{x} = z_1 v_1 + z_2 v_2 + \cdots + z_k v_k$$

Scree Plots

With all `141` games, yards and first downs give eigenvalues of `1.752` and `0.248`, so PC 1 holds `88%`, very nearly what our six games said. Now let's hand PCA the whole box score, all `26` stats for both teams, standardized. That gives us `26` components, and plotting their eigenvalues in order is called a **scree plot**:



A scree plot typically drops quickly and then flattens out. The bend is called the **elbow** (ours isn't a very sharp one, somewhere around the fourth component). After it each component adds very little, **the new axes "front-load" the variance**.

The last four eigenvalues are `0`, or close enough. Total yards is pass yards plus rush yards and plays is pass attempts plus rush attempts, for both teams, so four columns can be built from the others. PCA gives those directions no variance at all (the same columns broke the inverse in Linear Regression).

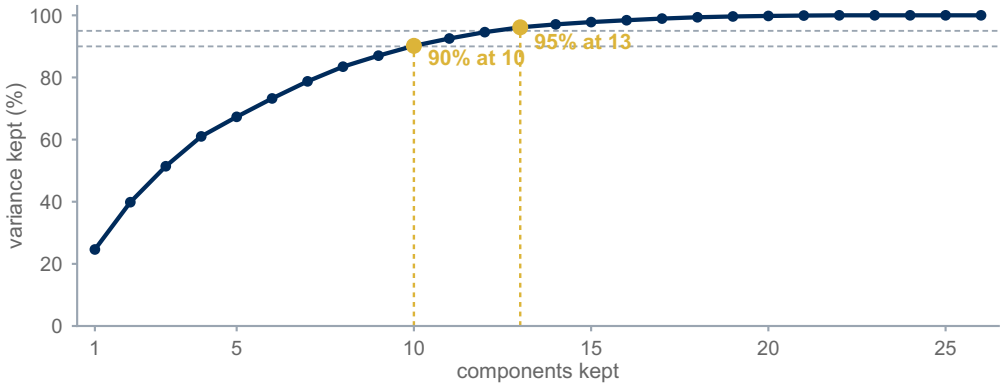
Scree is the loose rock piled at the bottom of a cliff, which is what the flat tail of the plot looks like.

How Many to Keep

The elbow is a judgment call. A more precise rule is to add up each component's share of the total variance, its **explained variance ratio**, and stop once we've kept enough:

[1.13] share of the variance on component k

$$\frac{\lambda_k}{\lambda_1 + \lambda_2 + \cdots + \lambda_d}$$

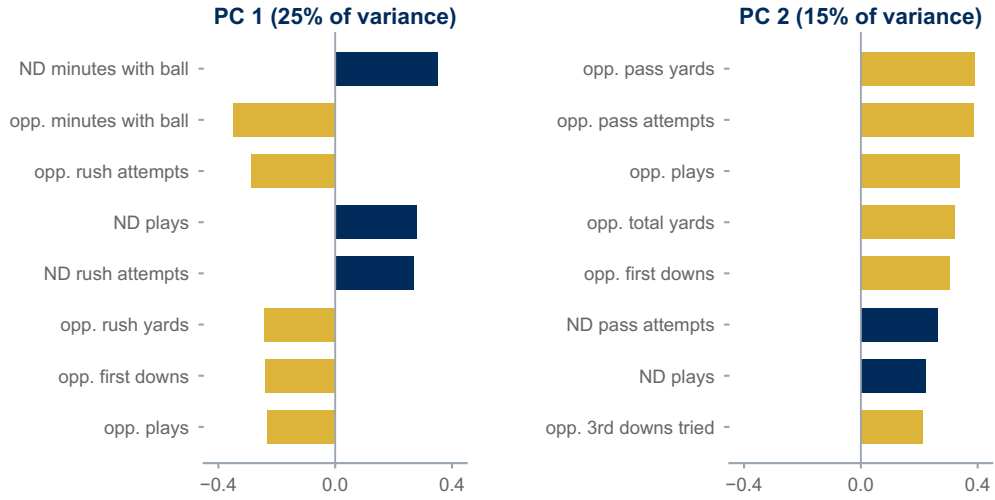


PC 1 alone holds `25%` of the variance and the first four hold `61%`. We need `10` of the `26` components to keep `90%`, and `13` to keep `95%`. **We can throw away half our columns and lose less than a twentieth of the variance.**

No right answer: `90%` and `95%` are common choices, not laws. The number of components is a hyperparameter (like the k in kNN), so the honest way to pick it is on games the model didn't see.

Interpreting the Components

Each component is a mixture of all 26 stats, but the biggest entries in its direction usually tell a story. Pictured below are the eight biggest entries of PC 1 and PC 2 (Notre Dame's stats in navy, the opponent's in gold).

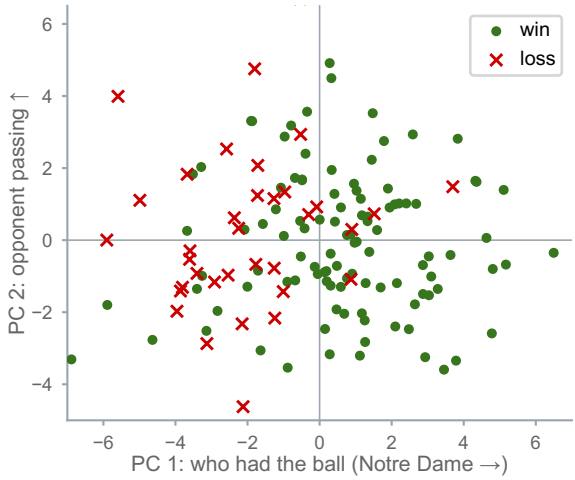


PC 1 is **who had the ball**. It goes up with Notre Dame's minutes of possession, plays and rush attempts, and down with the opponent's minutes, rush attempts, first downs and plays. PC 2 is mostly **the opponent throwing a lot**, its pass yards, pass attempts, plays and total yards all point the same way.

Unfortunately, it isn't usually this easy. Most components are a muddle of many stats, and further down the list they get very hard to name at all.

Visualizing with PCA

This answers the question Clustering left us with. We can't plot 26 dimensions, but we can plot every game's position on the first two components and color it by whether Notre Dame won.



PCA never saw who won, yet the wins pile up on the right. Of the 75 games with a positive PC 1 (Notre Dame had the ball more than usual) Notre Dame won 95%, and of the 66 with a negative one it won 53%. That's with only 40% of the variance. PC 2 on the other hand barely separates anything, losses happen at every height.

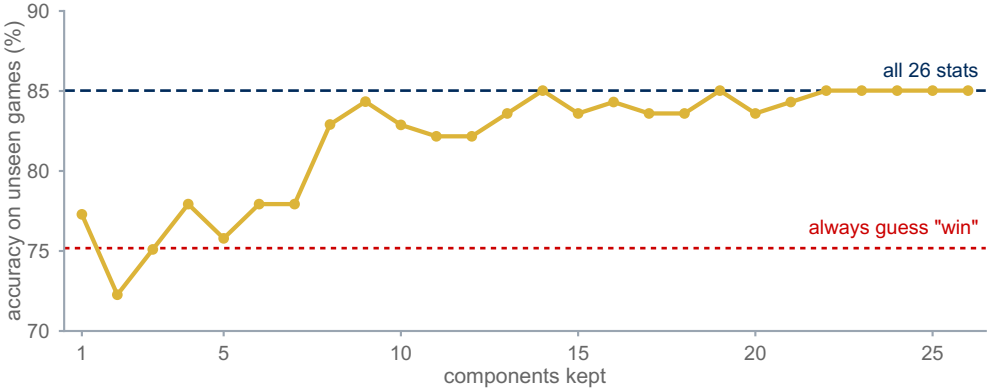
Input Space and Feature Space

Until now every model in this book has worked in **input space**, the space of our original measured stats. When we plot our games along the principal components instead, as on the last page, they're in a new space that PCA found for us, called **feature space** (or latent space). "Who had the ball" was never a column in our data.

In 1991 Turk and Pentland did the same thing to photographs of faces, treating every pixel as a feature. Their principal components, turned back into pictures, looked like ghostly faces (they called them **eigenfaces**), and a new face could be recognized by its position on the first few. Much of deep learning is about the same idea, **transforming our input into a latent space**, except that the transformation is learned rather than handed to us by an eigenvector.

Why Use PCA

Variance kept is nice, but what we really care about is whether our model still works. Back to "Did we win?". Below is **kNN** (k = 5) trained on only the first few components and scored on games it didn't see (five-fold cross-validation, as in kNN). PCA learns its axes from the data, so it's fit on the training games only, like the standardizing.



With all 26 standardized stats kNN gets the 85% we saw in kNN. Keeping 13 components (95% of the variance) gets 84%, and 14 is back to 85%. Keeping only two does worse than always guessing "win" (72% against 75%), so **a nice picture isn't the same thing as a good model**.

Fewer dimensions means faster training and prediction, less to store, and games that sit closer together, so "nearby" means something again. To be honest, with only 141 games a difference of a point or two is mostly noise.

The Singular Value Decomposition

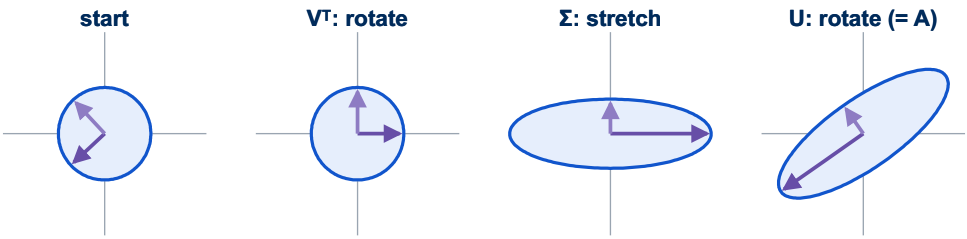
With 26 stats the characteristic equation is a polynomial of degree 26, and above degree 4 there isn't even a formula for its roots, so computers don't find eigenvalues this way at all. Building C is a problem too, it multiplies our data by itself, which **makes rounding errors worse**.

Instead we use the **Singular Value Decomposition** (SVD), which works on the data directly:

[1.14] singular value decomposition of any matrix

$$X = U \Sigma V^T$$

Any matrix is a rotation, then a stretch along the axes, then another rotation. V^T rotates, Σ stretches (its diagonal holds the **singular values** $\sigma_1 \geq \sigma_2 \geq \ldots$, already sorted), and U rotates again. Pictured below is one matrix acting on a circle, a piece at a time.



Perpendicular directions, sorted by how much they stretch. That should sound a lot like principal components!

The SVD Hands Us PCA

Put the SVD into [1.4], and because U is a rotation, $U^T U = I$ cancels:

$$C = \frac{1}{n} X^T X = \frac{1}{n} V \Sigma U^T U \Sigma V^T$$

$$\downarrow\ U^T U = I$$

rotate, stretch, rotate back

$$C = V \frac{\Sigma^2}{n} V^T \qquad [1.15]$$

That's C as rotate, stretch, rotate back, just what eigenvectors and eigenvalues do. **The rows of V^T are our principal components** and $\lambda_k = \sigma_k^2 / n$, and we never build C at all. Our six games again:

```python
import numpy as np

X = np.array([[584, 23], [576, 23], [432, 20],
              [380, 24], [265, 18], [253, 12]])
Z = (X - X.mean(axis=0)) / X.std(axis=0)  # step 1
U, S, Vt = np.linalg.svd(Z, full_matrices=False)
print(Vt.round(2))                # the directions
print((S**2 / len(X)).round(3))   # their variances
print((Z @ Vt[0]).round(2))       # step 6, on PC 1
```

OUTPUT

```
[[-0.71 -0.71]
 [-0.71  0.71]]
[1.748 0.252]
[-1.42 -1.38 -0.09 -0.5   1.15  2.24]
```

The same **1.748** and **0.252** (numpy's PC 1 just points the other way).

PCA in sklearn

In practice we let sklearn do all of it. Its `PCA` centers the data and runs the SVD, but it doesn't scale, so we standardize first with a `StandardScaler`:

```
import pandas as pd
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

games = pd.read_csv("nd_football.csv")
stats = [c for c in games.columns
         if c.startswith(("nd_", "opp_"))
         and c not in ("nd_points", "nd_home")]
X = StandardScaler().fit_transform(games[stats])

pca = PCA().fit(X)
print(len(stats))
print(pca.explained_variance_ratio_[:4].round(3))
print(PCA(n_components=0.95).fit(X).n_components_)
```

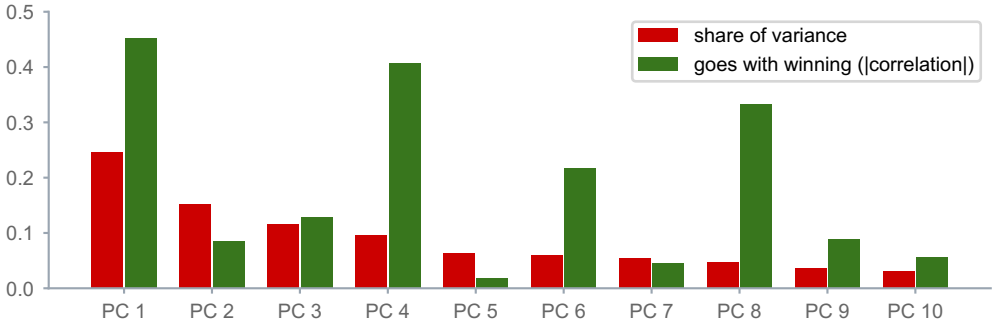
OUTPUT

```
26
[0.246 0.152 0.116 0.096]
13
```

Our `26` stats, the same shares as the scree plot (`25%`, `15%`, `12%` and `10%` for the first four), and asking for `n_components=0.95` keeps however many components it takes to reach `95%`, which is `13`. `pca.transform` then gives every game's positions on them.

Limitations of PCA

PCA is unsupervised. It keeps the directions with the most variance, and unfortunately **the most variance isn't always the most useful**. Pictured below is each of the first `10` components' share of the variance, next to how strongly it goes with the result.



PC 1 holds the most variance and goes with winning the most. But PC 2 holds `15%` of the variance and has almost nothing to do with the result (a correlation of `0.08`), while PC 4, with only `10%`, is tied to it nearly as strongly as PC 1 (`-0.41`, it goes with losing). PC 4 is mostly Notre Dame throwing a lot, not running much and turning the ball over (what a team that's behind tends to do), and keeping only the first three components would throw it away. PCA is also **linear**, it can't unbend anything curved.

Notice that none of this touched kNN itself. PCA hands a model better features, and with our `26` stats that bought us about nothing in accuracy. If we want more games right, the model is the thing that has to get better.