1/29/15
Lec 3, CS 4786
intuitions behind PCA.

(these notes assume you have the handout for the lecture on hand)

- ~~review~~ overview of our 3 topics:

  the data matrix X gives you:

  ... the covariance matrix Σ, whose largest-eigenvalue eigenvectors are the principal directions onto which PCA projects the data vectors.

  ...

About the data matrix: rows = instances or objects; columns = variables or features.

examples: each row is a student; each column is a grade in a certain class, a standardized test score on some test, etc.

It may not be 'fair' or 'correct' to reduce ~~the~~ living student before you into a bunch of numbers but it's often useful to use these numbers as our representation.

<aside: statistician George Box has been quoted as saying "all models are wrong, but some are useful" > (*)

: another example - the "vector-space model" of documents takes a text as complex Hamlet and reduces it to a set of counts of the words it contains.

In some sense it's absurd to reduce all that complexity to a single point.

But on the other hand, you could say, what else is a text, but just made up of the words that are in it?

(*) Neil Mason on CrossValidated says that Box clarified, "the practical question is how wrong do they have to be not to be useful?"

First, let's just look @ the n×d shape. These are meaningful, and something you hear about in data-science conversations.

~~a big~~ "tall"/"deep" matrix: n big, d small. → ("traditional"/"classic")

Exemplar: "~~old-fashioned~~" survey data: you reached a lot of people, but only got the answers to a few questions.

"fat"/"wide" matrix: n small, d big

Exemplar: "ethnographic" research, where there's a small community or collection of items ~~that you know a lot~~, for each of which you have a tremendous amount of info/knowledge.

→ "big data": both n and d are big.

Exemplar: Facebook has data on a 'gillion' (a billion active users) people, and a lot of info about each person.

Now...

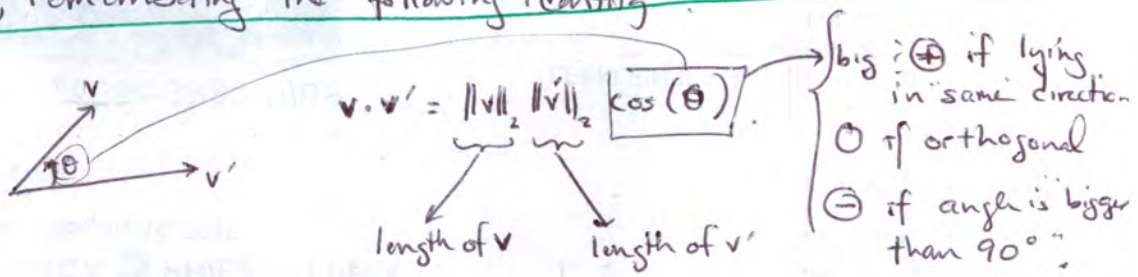
- Let's look @ the examples on your handout to get some intuitions as to what the data matrix tells us.

⟶ Blue & pink documents are about cars; orange doc is about linguistics; these are things we as humans know by reading these documents.

- Suppose we make a data matrix: rows = docs, entry(i,j) = # of times the j^{th} term shows up in doc i. So you can see from the term overlaps (# circled) that the pink & blue docs share the terms "car", "hood", "trunk".

- now let's visualize these document vectors geometrically.
Since the pink & blue docs share terms, their vectors are going to lie closer together than to the orange vector.

You can compute such relationships using the inner product between two vectors, remembering the following identity:



so notice that if you length-normalize your vectors, then the inner product is just the cosine, and tells you how much your vectors lie "in the same direction."

But, if some of your vectors can be arbitarily big, then the length ~~term~~ parts of the inner product will be big, and so even things that don't point "in the same direction" will have large inner product.

(Indeed, in the old days spammers tried to trick search engines by ~~writing~~ repeating ~~lots of key terms~~ lots of key terms in the same color as the webpage's background, to get artificially high similarity scores, even tho' people wouldn't see all those extra terms.)

Anyway, here this representation did seem to represent the "true" relationships btwn the documents well.

Does this representation always work well? ~~too~~

<Handout does say, "a "bad" feature space ☺>.

2nd example: due to the use of synonyms (features that "look different" but aren't) and ambiguous features (two actually-diff features that are being treated as the same, like the word "model"):
the pink & blue doc. vectors are far apart, and the pink & orange ones are close.

- we're in the wrong feature space, which is something PCA can help us with.

~~Principle comp~~

Principal Components analysis: ~~looks at (orthogonal)~~

~~as I~~ we presented it in last lecture as being based on the eigenvectors of the covariance matrix $\boxed{\Sigma}$ of X. (Altho' in practice it can be computed in other ways.)

↑ box to distinguish from 'summation sign'

So, what does $\boxed{\Sigma}$ mean? It tells you how the features covary.

- Since there are d features, $\boxed{\Sigma}$ will be d×d.
- Last time we expressed $\boxed{\Sigma}$ as: $\frac{1}{n}\sum_{t=1}^{n} \underbrace{(x_t-\mu)}_{d\times 1}\underbrace{(x_t-\mu)^T}_{1\times d}$, an outer product.

~~The outer pr~~

Outer products like these look like:

$$\begin{bmatrix} v[1] \\ v[2] \\ \vdots \\ v[d] \end{bmatrix} [v[1]\ v[2]\ \cdots\ v[d]] = \begin{bmatrix} v[1]v[1] & v[1]v[2] & \cdots & v[1]v[d] \\ & \vdots & & \\ v[d]v[1] & v[d]v[2] & \cdots & v[d]v[d] \end{bmatrix}$$

(d × d)

↑ all pairwise combinations of the entries.

So the entries of $(x_t-\mu)(x_t-\mu)^T$ look like: $(x_t-\mu)[i] \times (x_t-\mu)[j]$.

- $(x_t-\mu)[i]$: observation for i^{th} feature
- $(x_t-\mu)[j]$: observation for feature j.

$$\begin{bmatrix} x_t[1] \\ x_t[2] \\ \vdots \\ x_t[d] \end{bmatrix} - \begin{bmatrix} \mu[1] \\ \vdots \\ \mu[d] \end{bmatrix} \quad \begin{bmatrix} (x_t-\mu)[1](x_t-\mu)[1] & \cdots \\ \vdots & \ddots \end{bmatrix}$$

(x_t)

If we sum over all n objects, then dividing by n, we've computed the average of observations of (feature i − its mean) × (feature j − its mean).

But that's a [biased] (*) estimate of the covariance!

(defn: $\text{cov}(F_i, F_j) = E[(F_i - \text{its mean}) \cdot (F_j - \text{its mean})]$, if you recall from intro probability).

(*) dividing by $\frac{1}{n-1}$ instead of $\frac{1}{n}$ gives an unbiased estimate. Be careful when you use a software package which does normalization for you.

- alternately, you can think of the covariance matrix in terms of inner products:

matrix multiplication: $\begin{bmatrix} -i- \end{bmatrix} \begin{bmatrix} \vdots \\ j \\ \vdots \end{bmatrix} = \begin{bmatrix} \cdot & \cdot \\ \cdot & \nearrow \\ \cdot & \cdot \end{bmatrix}$

(i,j) entry is the inner product of the first matrix's i^{th} row & the 2^{nd} matrix's j^{th} ~~col~~ column.

If we let x_t^c (c = "centered") stand for $x_t - \mu$, then it's also the case that

$$\boxed{\Sigma} = \frac{1}{n} \left\{ \overbrace{\begin{bmatrix} | & | & & | \\ x_1^c & x_2^c & \cdots & x_n^c \\ | & | & & | \end{bmatrix}}^{n} \underbrace{\begin{bmatrix} - & x_1^c & - \\ - & x_2^c & - \\ & \vdots & \\ - & x_n^c & - \end{bmatrix}}_{d} \right\}_n \qquad \text{(product is } d \times d\text{)}$$

Notice that we have inner products between the rows of the 1st matrix & the columns of the 2nd matrix:

- meaning inner products between the (recentered) features
- so again, we're comparing "similarities" btwn the features, consider as vectors.

("how ~~similar is the word~~ correlated are ~~scores on science~~ scores on science tests to scores on math tests?"

"how does the usage of word 'Twitter' ~~was~~ vary w/ the usage of the word 'Facebook'?"

So ~~the~~ what ~~Answer~~ can we learn by looking @ the covariance matrix?

Let's look @ an example (run pcaEx.py, kill after cov. matrix printed out)

- note: numpy (which I only started working with that night) turns out to want transposed data matrices. I lost so much time trying to figure out why the ~~correlation~~ covariance matrix was the wrong dim, etc.

BUT: knowing what dimensions things should be help you debug (instead of unknowingly computing wrong answers!)

6 data items, 4 features. Presumably you don't see any obvious patterns. (Maybe: students, scores on math, english, physics, vocabulary)

But now look @ the covariance matrix.

1st row says 1st feature has high ⊕ inner product w/ itself, very ⊖ inner product w/ the 2nd feature, high ⊕ inner product w/ 3rd, very ⊖ inner product w/ 4th. And it's symmetric, which makes sense from the inner product being order-independent.

(*) Now, what about PCA??

We saw last time that the eigenvectors of Σ give us the orthogonal directions of biggest variation.

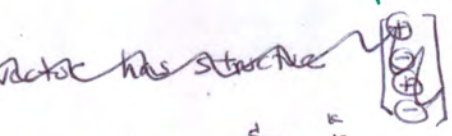
<Look @ pcaEx.py to see how to compute the covariance matrix - cov - and get eigenvectors - eig. And then a little code to get the column eigenvectors sorted in decreasing eigenvalue order>.

numpy gives us a $d \times d$ non-symmetric matrix. → (this is W^T is the notation of lec. 2) (other lectures may have things transposed!)

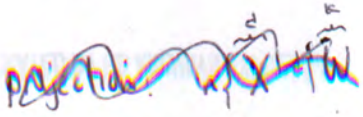
The columns are unit-length eigenvectors.

(in fact, there's a sanity check in the code to see that the lengths are as they should be (which turned out to be handy when someone asked if these were really unit-length!)

The 1st eigenvector has structure:



projection



If you take the centered data project along

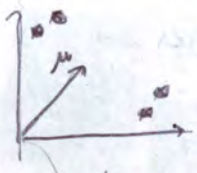
<Look @ pcaEx2.py>

(data - data.mean(axis=0))

(the centered matrix already happens to make the structure clear, in this case: the 1st 3 objects are quite diff. from the last 3 objects.

To get an idea of what PCA is going to do:

- our original data looks like the 4-d analog of this:



either big-first, small second coord,
or,
small-first, big-second coord.

- centering the data will give us something like this:

skip for lecture

so the direction of biggest variance is "northwest" (negative 1st coord, positive 2nd) / "southeast" (positive 1st coord, negative 2nd), (depending on)

Using pcaEx2.py, you can indeed see that the centered data has the form

skip for lecture

$$\begin{bmatrix} - & - \\ - & - \\ - & - \\ - & - \\ + & + \\ + & + \\ + & + \\ + & + \end{bmatrix}$$

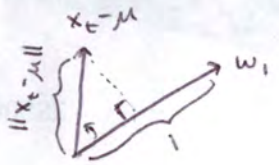
(so in this case, even the centered ~~matrix~~ data matrix itself makes the data's structure clear)

And the computed eigenvectors (sorted by decreasing eigenvalue) look like this:

first one: $w_1 = \begin{bmatrix} + \\ - \\ + \\ - \end{bmatrix}$ = on the "northwest/southeast axis", as predicted.
(in 1st two features' plane).

~~the projection~~

The projection on to the 1st eigenvector is given by: $w_1 \cdot (x_t - \mu)$, which is a single #.
Recall this is equal to: $\|w_1\| \cdot \|x_t - \mu\| \cos(\angle(w_1, x_t - \mu))$ since we have unit-length eigenvectors.
$= \|x_t - \mu\| \cos(\angle(w_1, x_t - \mu))$



cosine = "adjacent ÷ hypotenuse"

~~so also~~

so $\|x_t - \mu\| \cos(\angle(w_1, x_t - \mu)) = \text{hyp} \times \frac{\text{adj}}{\text{hyp}}$

= adjacent
(length of) projection!
geometrically!

The projections onto all selected e-vectors is thus:

$$\begin{bmatrix} — & w_1^T & — \\ — & w_2^T & — \\ & \vdots & \\ — & w_k^T & — \end{bmatrix} \cdot (x_t - \mu)$$

(get a k×1 vector of dot-products).

For pcaEx2.py, ~~you have the signs for w~~ if we project just on to w_1, we see that ~~the signs~~ for ~~y_1, y_2, y_3 all differ~~ the projections of the 1st 3 instances are all diff than for the last 3 instances, confirming our intuition that there's big separation of the 1st 3 from the last 3.

Finally, some computational notes: 6

We've shown that the principal directions correspond to the eigenvectors of the covariance matrix.

But, you don't have to compute the covariance matrix explicitly! (And in fact, you would want to avoid doing this if d is large).

There exist singular-value decomposition (SVD) methods that will recover the principal directions just from the data matrix.