Principal Components Analysis

Dimensionality

Let's start thinking about our data in terms of *dimensions in space*.

Each predictor is an axis.

The values of the predictors for a certain observation define a point in space.

When we compute distances between observations for KNN, we are computing distance in space.

When we fit a regression, we are drawing a straight line through the points.

The curse of dimensionality

We run into trouble when we have **too many dimensions**.

What does "too many" mean?

Parametric estimation -> We can't estimate 7000 coefficients from only 44 observations!

Interpretability -> Do we really want to translate our model into meaning for thousands of predictors???

Flexibility -> More predictors = more flexibility = overfitting?

Principal Components Analysis

PCA is a way to *transform our data* (prior to modeling!) so that it has fewer dimensions in space.

Instead of:

axis 1 = Predictor A

axis 2 = Predictor B

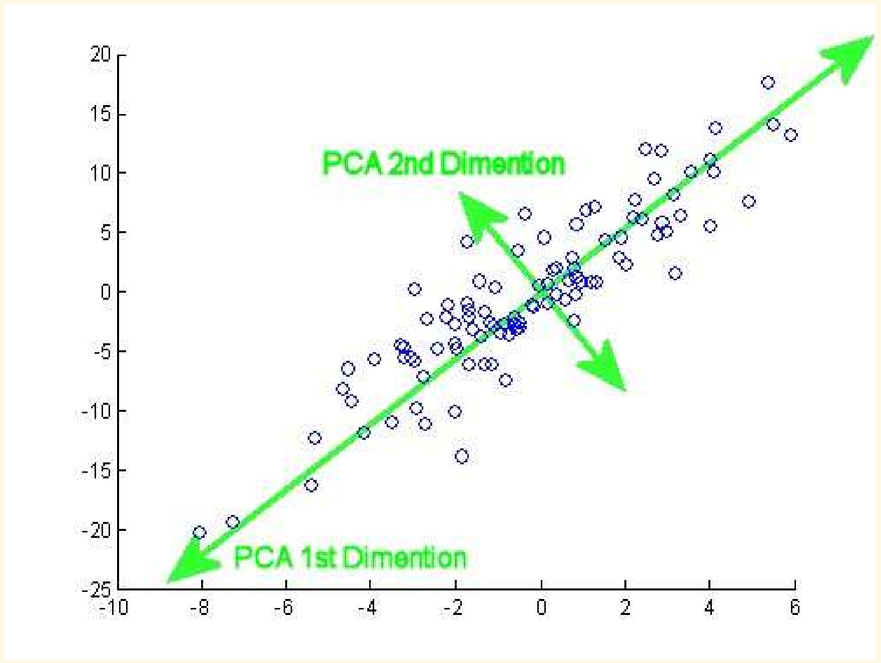
axis 3 = Predictor C

axis 1 = 0.5 (Pred A) + 0.2 (Pred B) + 0.3 (Pred C)

axis 2 = 0.1 (Pred A) + 0.7 (Pred B) + 0.2 (Pred C)

axis 3 = 0.1 (Pred A) + 0.2 (Pred B) + 0.8 (Pred C)

PCA


PCA 2nd Dimention
PCA 1st Dimention
20
15
10
5
0
-5
-10
-15
-20
-25
-10
-8
-6
-4
-2
0
2
4
6

PCA

1. Standardize all axes.
2. Find the axis of highest variance: This is PC 1.
3. Find the axis of highest variance that is perpendicular to PC 1: This is PC2.
4. Continue until you have p PCs, where p = number of predictors (or, if $p > n$, until you have n PCs).
5. Use only the first k predictors in your analysis, where $k < p$ and $k < n$.

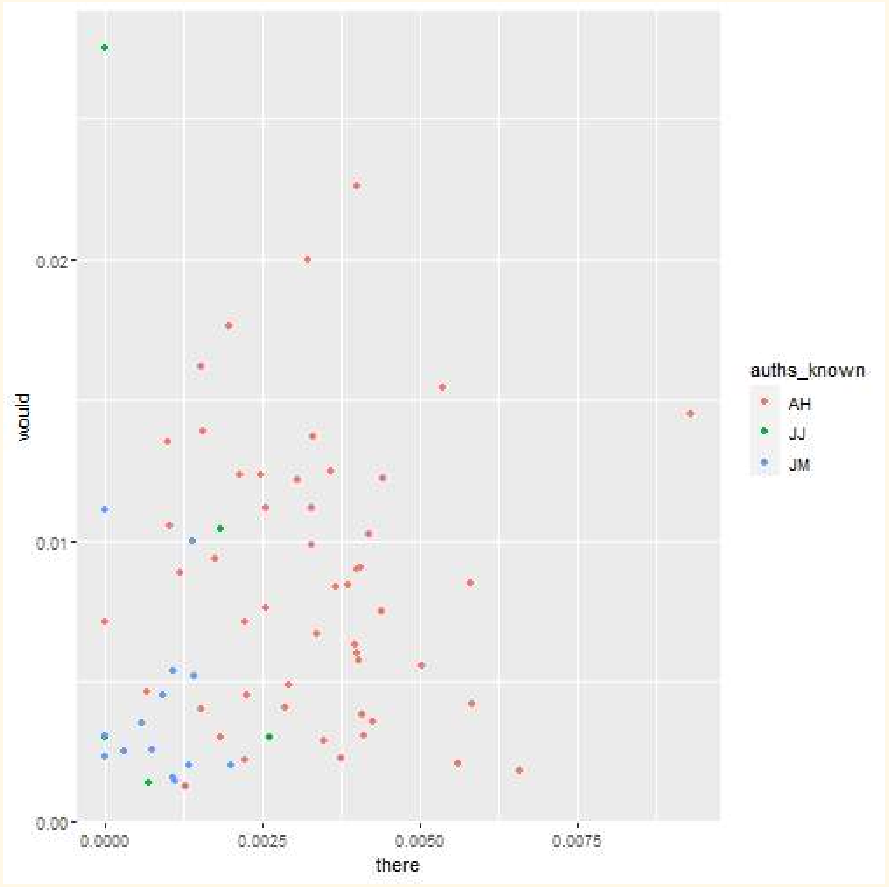
Example

The Federalist papers are a series of essays written by John Jay, Alexander Hamilton, and James Madison.

Data: How many of each word was used in each essay (for the most common 200 words only).

Example

If we choose a couple words and plot our data...


would
0.02
0.01
0.00
0.0000
0.0025
0.0050
0.0075
there
auths_known
AH
JJ
JM

Example

Instead, let's apply pca:

```
fed_matrix <- fed_ex %>% select(-auths_known) %>% as.matrix()
pc <- prcomp(fed_matrix, center = TRUE, scale = TRUE)
```

Example

Combinations of variables that create new axes:

```
##               PC1        PC2        PC3        PC4        PC5       PC6
## a       0.1981759 -0.0007576  0.0935479 -0.1214028 -0.1896307  0.072464
## do     -0.0302646 -0.0727378 -0.0926549 -0.0626465 -0.1555949  0.036756
## is      0.1336907  0.0797518 -0.1785689  0.0632460 -0.1423981 -0.053693
## or     -0.1089584 -0.1039416 -0.0132435  0.0042732  0.0412992 -0.177685
## this    0.2334027  0.0630079 -0.0218351 -0.0906833 -0.1114024  0.034180
## all     0.0844019  0.0628760  0.0426171  0.1138584  0.1803859 -0.052679
## down    0.0292295 -0.0751349 -0.0179187  0.0165084  0.0005723 -0.140894
## it     -0.0292278 -0.0288241 -0.2022608 -0.2491168  0.0007965  0.065907
## our    -0.1253024 -0.0447230  0.0946293 -0.0059994  0.1822235 -0.004053
## to      0.1133667 -0.1822340 -0.0075929 -0.1504039  0.1180770  0.042107
## also   -0.1895352  0.0803221  0.0144453  0.1038795 -0.2159480 -0.021350
## even    0.0412006 -0.0053735  0.1988205 -0.1080156 -0.0070907 -0.085335
## its     0.0925980 -0.1364080 -0.1003783  0.0562824  0.1573799 -0.087837
## shall   0.0275950  0.0069202 -0.0851937  0.0941129 -0.2189935  0.061184
## up      0.0440077 -0.0079942  0.1198425 -0.1167414  0.0746236 -0.092781
## an      0.1987379  0.0008641  0.0911709 -0.1070558 -0.0145571  0.066112
## every   0.1006472  0.0201706 -0.1635864  0.1268195 -0.1723828 -0.016974
## may    -0.0002659 -0.1579650 -0.1937055  0.0956668 -0.1181080  0.055328
## should -0.0633278  0.0130290 -0.0238200 -0.2547560  0.0931800 -0.055491
## upon    0.2414271 -0.1504749  0.0467621 -0.1254395  0.0574721  0.071282
## and    -0.2969149  0.0829995 -0.0079523  0.0648104  0.1141941 -0.069459
```

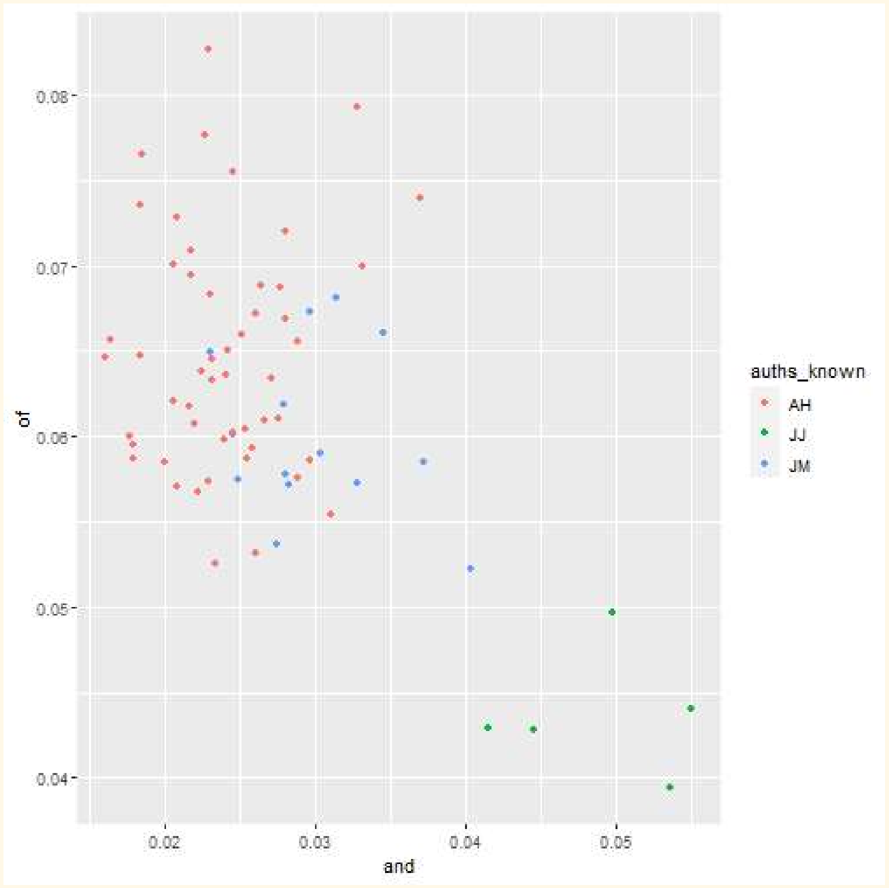
Example

What variables matter most?

```
##    rowname        PC1        PC2        PC3        PC4        PC5       PC6
## 1      and -0.2969149  0.0829995 -0.0079523  0.0648104  0.1141941 -0.069459
## 2       of  0.2532369  0.0465962  0.1820845  0.0849970 -0.0677631 -0.024695
## 3     upon  0.2414271 -0.1504749  0.0467621 -0.1254395  0.0574721  0.071282
## 4     this  0.2334027  0.0630079 -0.0218351 -0.0906833 -0.1114024  0.034180
## 5      one -0.2310547 -0.0273376  0.0517796  0.0560865 -0.2252644 -0.001185
## 6     more -0.2192323 -0.1299366  0.1103656  0.1446700  0.0100714  0.140034
## 7    their -0.2098190  0.0526373 -0.0558632 -0.1481704  0.1857935  0.126837
## 8       an  0.1987379  0.0008641  0.0911709 -0.1070558 -0.0145571  0.066112
## 9        a  0.1981759 -0.0007576  0.0935479 -0.1214028 -0.1896307  0.072464
## 10     the  0.1959157  0.0652826 -0.0653657  0.0971773 -0.0635347 -0.142684
## 11    also -0.1895352  0.0803221  0.0144453  0.1038795 -0.2159480 -0.021350
## 12    into -0.1646578 -0.0768915  0.0947230  0.1017412 -0.0845516  0.010113
## 13   there  0.1570871 -0.2135796  0.0912550 -0.0034868 -0.0925714  0.003078
## 14     in.  0.1517917  0.0252016 -0.0993965 -0.1591765 -0.1155705  0.118976
## 15   which  0.1406579  0.0833763 -0.0633439  0.1863356  0.1818572 -0.010853
## 16    been  0.1397300  0.2585870 -0.0832399  0.0552028 -0.0521795  0.108904
## 17      is  0.1336907  0.0797518 -0.1785689  0.0632460 -0.1423981 -0.053693
## 18    than -0.1292803 -0.2098651  0.0407369  0.1246189 -0.1711510  0.261084
## 19     our -0.1253024 -0.0447230  0.0946293 -0.0059994  0.1822235 -0.004053
## 20    only -0.1252312  0.0404905 -0.1818237 -0.1444484 -0.0538031 -0.062422
## 21    with -0.1202768 -0.0353634  0.1731743  0.1397497 -0.0882275 -0.029550
```

Example

This doesn't really help us visualize the data...



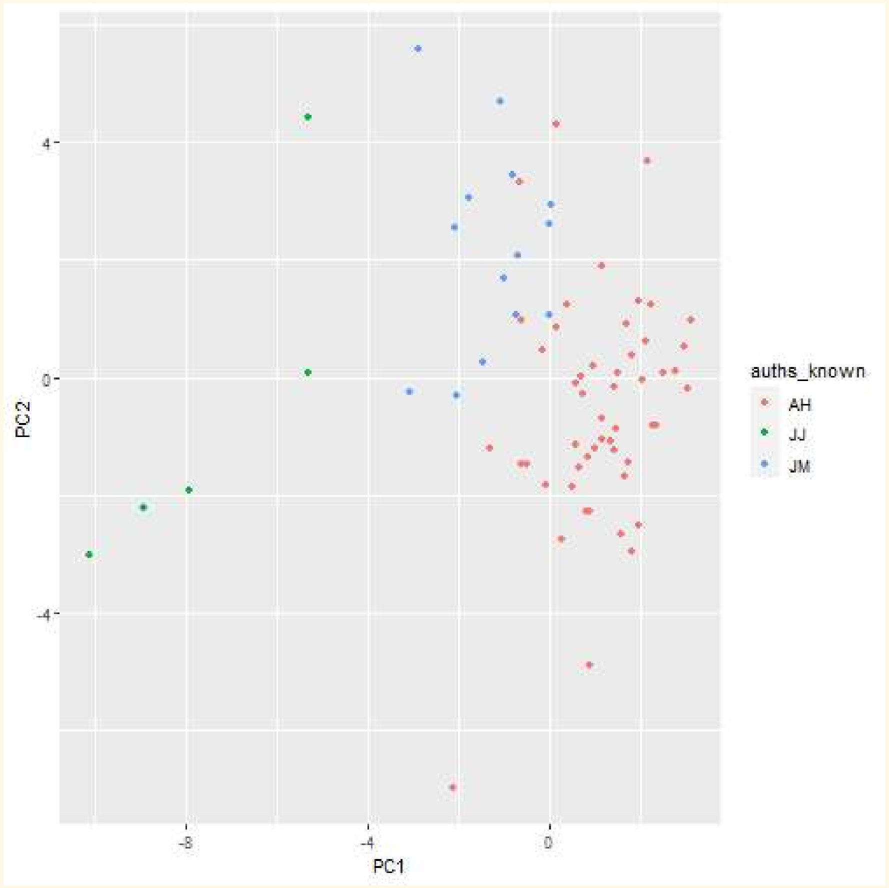
0.08
0.07
0.06
0.05
0.04
of
0.02
0.03
0.04
0.05
and
auths_known
AH
JJ
JM

Example

Locations of observations on new axes:

```
##            PC1      PC2      PC3      PC4       PC5      PC6      PC7       PC8
## 1    0.255883 -2.75723 -0.84061  4.22604  3.580825  7.87903 -0.74834 -2.742008
## 2   -5.333328  4.41925  0.10285 -1.85960  0.385100  2.77020 -2.05465  0.310339
## 3   -7.933306 -1.90187 -2.06613  0.17152  0.444284 -0.88954  2.73854  1.482792
## 4  -10.132733 -3.01485 -0.76659  0.22149  0.388497 -0.82803 -1.44149  2.264506
## 5   -8.922169 -2.22472  3.78289 -0.08058 -0.187915  0.18637 -3.19444 -4.563181
## 6   -0.664084  3.30507  3.19383  1.00454  1.497993  1.68512  0.78693  2.868702
## 7    1.140376  1.88974  5.25083 -1.66296  1.699322 -0.41244 -0.62420 -0.832843
## 8    0.358401  1.23172  2.36159  0.81826  1.955048 -1.53049 -1.06244  1.153950
## 9    0.140885  4.29456  2.16608  0.56263 -0.004672  0.68167  0.30781  1.427079
## 10  -2.035723 -0.31634 -1.02594  1.82065  1.242992 -0.64046  0.86073 -0.908341
## 11  -0.637666 -1.45658  3.78830  0.35541 -0.251304 -0.36100 -3.51373  0.569111
## 12   1.392538 -0.17138  1.80856  1.19238  1.713521 -1.00059 -2.12927  1.822116
## 13  -2.132165 -6.97474  4.04474  4.79525 -6.312285  0.88882 -2.32670  1.957892
## 14  -1.017369  1.67820 -1.35167  2.20414  2.951174  1.91323 -0.15203 -1.600599
## 15   2.746062  0.09324  1.33704  2.02277  2.416346  0.17314 -2.49471  2.170778
## 16   1.346831 -1.09305  2.17638 -2.53938  0.597100 -0.88129 -0.16961 -0.264660
## 17   0.945306  0.21010  4.27734  2.09572  2.968605  0.35147  1.51480 -1.561375
## 18   3.014215 -0.18193  0.45892  1.34565 -1.801528 -1.42809 -1.18131  2.533578
## 19   1.785756  0.37943  0.92432 -0.41027  0.024650 -0.17518 -1.19383  0.674022
## 20   1.699876 -1.45300 -1.79566  1.69022  3.632504 -1.03967  0.60535  1.341652
## 21   0.688843  0.02942  2.43645 -3.92072  0.986641 -0.41756 -2.71591 -0.324139
```

```
new_dims_df %>%
  ggplot(aes(x = PC1, y = PC2, color = auths_known)) +
  geom_point()
```



Example

Standard deviations of PC scores:

```
##  [1] 2.584293976481497168 2.224602605692588053 2.133495077907519910
##  [4] 2.009679212438630014 1.771199010414047370 1.737846153429807972
##  [7] 1.658192994671916720 1.593432897204484666 1.567765465416049331
## [10] 1.498153968179446682 1.448627829241023512 1.383202255873344555
## [13] 1.348835290469535098 1.329864883494040306 1.290075723747528214
## [16] 1.263656822413196323 1.211546834748383983 1.179468039028208848
## [19] 1.140273064226865252 1.136685249904107353 1.091881595973659413
## [22] 1.024359742653421224 1.002539216556411761 0.988309265374064272
## [25] 0.966758063464953188 0.953955378557446387 0.913371285706331415
## [28] 0.879219144916352668 0.866397791514436300 0.835362796306243660
## [31] 0.813744862497518318 0.795836559214300077 0.777288905706550737
## [34] 0.755507991799209577 0.729325759585313982 0.723553021174748179
## [37] 0.668529227774796841 0.644640218695886280 0.608063670676834200
## [40] 0.599862980594791706 0.589869259793525114 0.549984140873157168
## [43] 0.518576401942239196 0.490365964788197883 0.485951132126856200
## [46] 0.473420966039042657 0.447648096786177840 0.410247671033032724
## [49] 0.398219462937036084 0.381729038085061145 0.356072979106936693
## [52] 0.322926868470927830 0.302289519970545262 0.278418759166978558
## [55] 0.254888335158211798 0.246279930847948220 0.227060020169778221
## [58] 0.209288942626959323 0.201384112256443981 0.181411412891078455
## [61] 0.162645660545995191 0.152090980040969159 0.125916504292499787
## [64] 0.109322167263386283 0.082184427523104023 0.065913614423482467
```

Example

Cumulative variances:

```
cumul_vars <- cumsum(pc$sdev^2)/sum(pc$sdev^2)
cumul_vars
```

```
##  [1] 0.09541 0.16611 0.23113 0.28883 0.33365 0.37679 0.41607 0.45234 0.48745
## [10] 0.51952 0.54950 0.57683 0.60282 0.62809 0.65186 0.67467 0.69564 0.71552
## [19] 0.73409 0.75255 0.76958 0.78457 0.79893 0.81288 0.82623 0.83923 0.85115
## [28] 0.86219 0.87292 0.88289 0.89235 0.90139 0.91003 0.91818 0.92578 0.93326
## [37] 0.93964 0.94558 0.95086 0.95600 0.96097 0.96529 0.96914 0.97257 0.97594
## [46] 0.97915 0.98201 0.98441 0.98668 0.98876 0.99057 0.99206 0.99337 0.99447
## [55] 0.99540 0.99627 0.99700 0.99763 0.99821 0.99868 0.99906 0.99939 0.99961
## [64] 0.99979 0.99988 0.99994 0.99997 1.00000 1.00000 1.00000
```

Example

```
plot(cumul_vars)
```

Details

How many PCs should we use?

No single answer; people often do "enough for 90% variance covered" or similar.

How do you do modeling with PCA?

You don't. It's a data *preprocessing* step.

You would then use `PC1`, `PC2`, etc **instead of** your original predictors.

OR you might use the variable importance measures ("pc loadings") to help decide which predictors to keep.

Pros and Cons

Pros:

- Reduces dimension while still letting all original predictors be "involved"
- Computationally fast for big data
- Axis rotations are interpretable!
- Dropping off lower PCs gets rid of noise (maybe)

Cons:

- Using PCs in interpretable models makes them uninterpretable. (What does the coefficient of PC1 mean in real life?)
- No magic answer for how many PCs to use.
- Dropping off lower PCs gets rid of useful info (maybe)

But wait - isn't this LDA?

Recall that LDA also found *scores* for your observations based on a *linear combination* of the original predictors.

So what's the difference?

LDA loadings are trying to maximize the difference in mean scores across categories.

(*supervised* method)

PCA loadings are trying to maximize the variance of the scores on the PC axes.

(*unsupervised* method)

Try it!

Open Activity-PCA.Rmd

Apply PCA to the cannabis data

Interpret the PC rotations

Plot the data on the first two axes, and color by Type.

Choose a "good" number of PCs to use.

Fit a KNN classifier using only your chosen PCs. How does the accuracy compare to when you use all the original predictors?