Dimensionality Reduction

(part 1 of 2)

Ryan Henning

- Why reduce dimensionality?
- Methods for reducing dimensionality
- One common technique: *Principal Components Analysis* (PCA)
- Crude facial recognition with PCA and kNN (i.e. *Eigenfaces*)

Before we go on:
What is the *dimensionality* of our data?

Here are some different words for the same thing…

- "dimension" = "feature" = "predictor"
- "dimensionality" = "number of features" = "number of predictors"

	mpg	cylinders	displacement	horsepower	weight	acceleration	model	origin	car_name
0	18	8	307	130.0	3504	12.0	70	1	chevrolet chevelle malibu
1	15	8	350	165.0	3693	11.5	70	1	buick skylark 320
2	18	8	318	150.0	3436	11.0	70	1	plymouth satellite
3	16	8	304	150.0	3433	12.0	70	1	amc rebel sst
4	17	8	302	140.0	3449	10.5	70	1	ford torino

How many dimensions is this dataset?

How many dimensions is this dataset?

Why reduce dimensionality?

HIgh dimensional data causes many problems. Here are a few:

- **The Curse of Dimensionality:**
  - Points are “far away” in high dimensions, and it’s easy to overfit small datasets. (see kNN / Decision Tree slides for a review of *The Curse*)
- **Visualization:**
  - It’s hard to visualize anything with more than 3 dimensions.
- **Latent Features:**
  - Often (especially with image/video data) the most relevant features are not explicitly present in the high dimensional (raw) data.
- **Remove Correlation:**
  - With many many features (dimensions) you can bet on there being a ton of correlation (e.g. consider neighboring pixels in an image dataset).

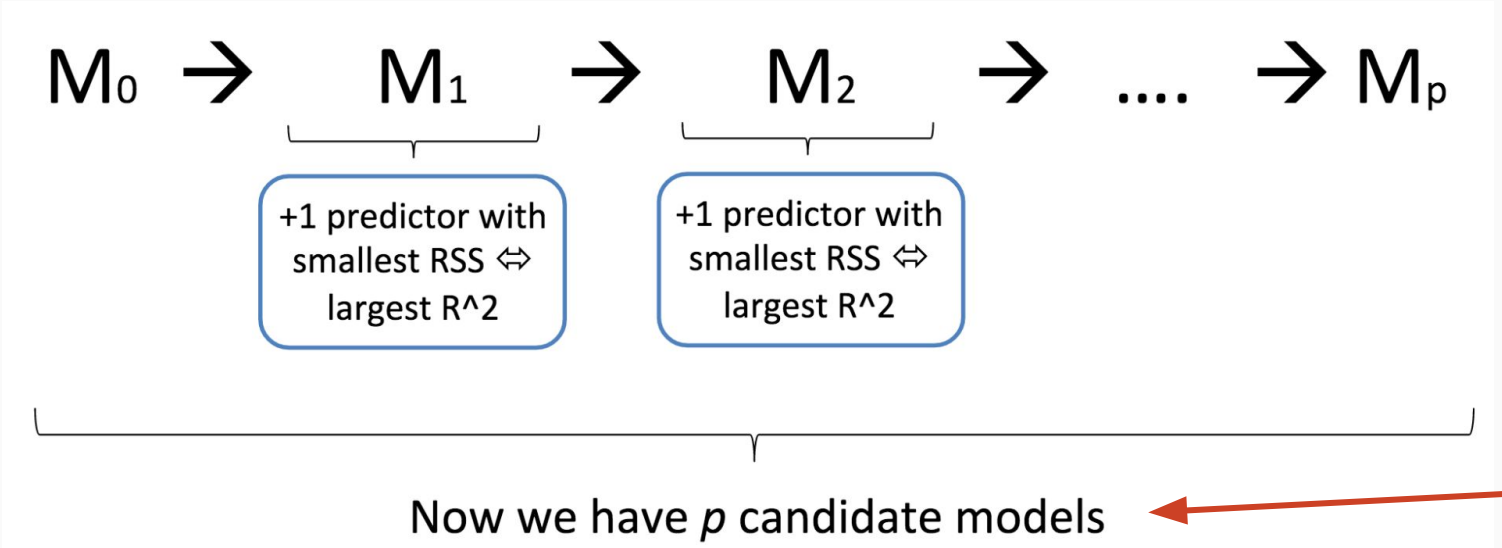
Methods for reducing dimensionality

There are a number of ways to reduce the dimensionality of your dataset:

- Subset selection of features; e.g. forward stepwise selection
- LASSO regression
- Relaxed LASSO:
  - (1) do LASSO regression, (2) throw away unused features, (3) do OLS regression
- Neural networks:
  - For labeled data: feature extraction via upper-layer outputs
  - For labeled or unlabeled data: autoencoders
- Principal Components Analysis (PCA)

This is the primary topic today, but first let's review these other methods.

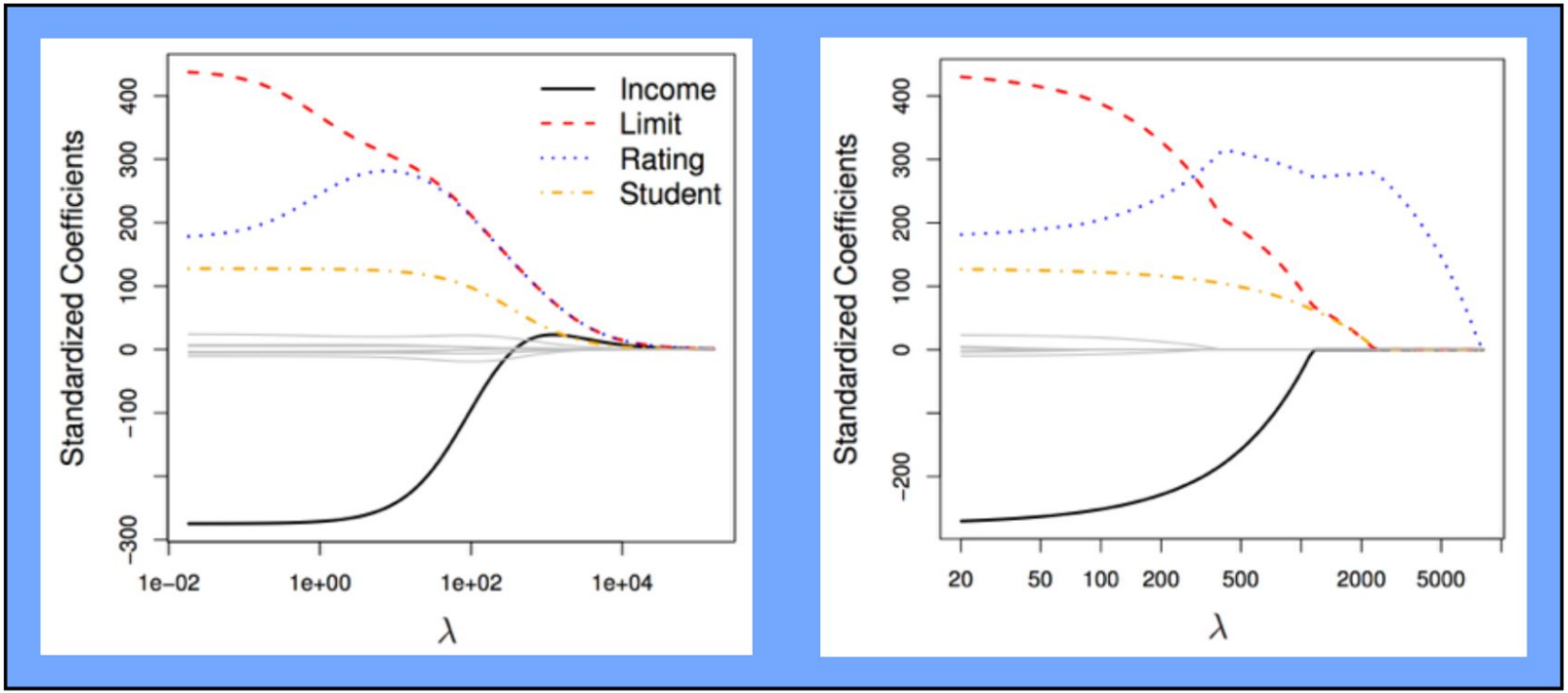
Review: Forward stepwise subset selection of features



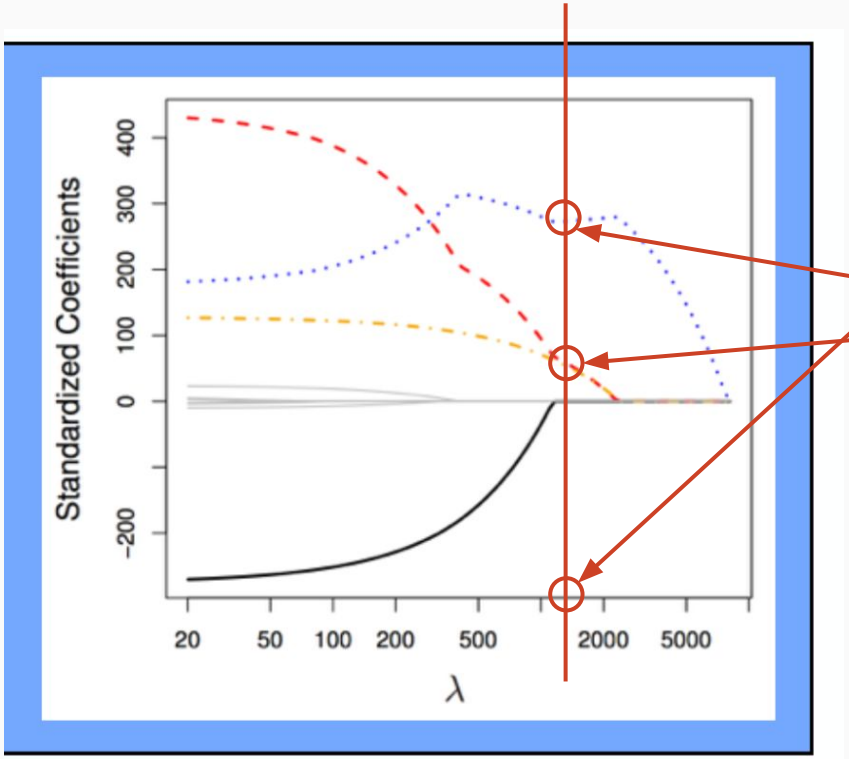
Use Mallow's C_p, or AIC, or BIC, or Adjusted R^2, or cross-validation to choose which of these *p* models to use in production.

Review: LASSO regression for creating sparse models

Ridge vs. Lasso



Relaxed LASSO (Meinshausen, 2006)

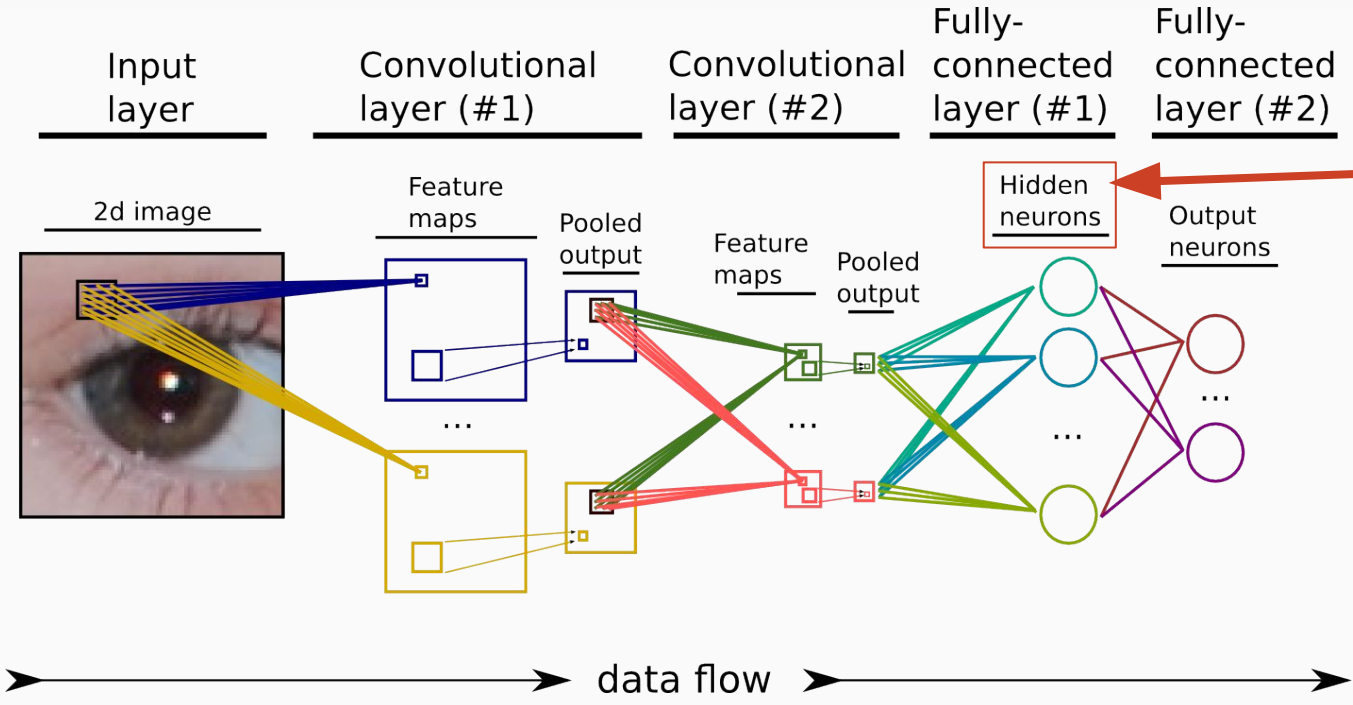


1. Use cross-validation and LASSO regression; find the best value for lambda.

2. Keep only the features with non-zero coefficients.

3. Re-fit using ordinary least squares (OLS) regression. (I.e. Re-fit using no regularization!)

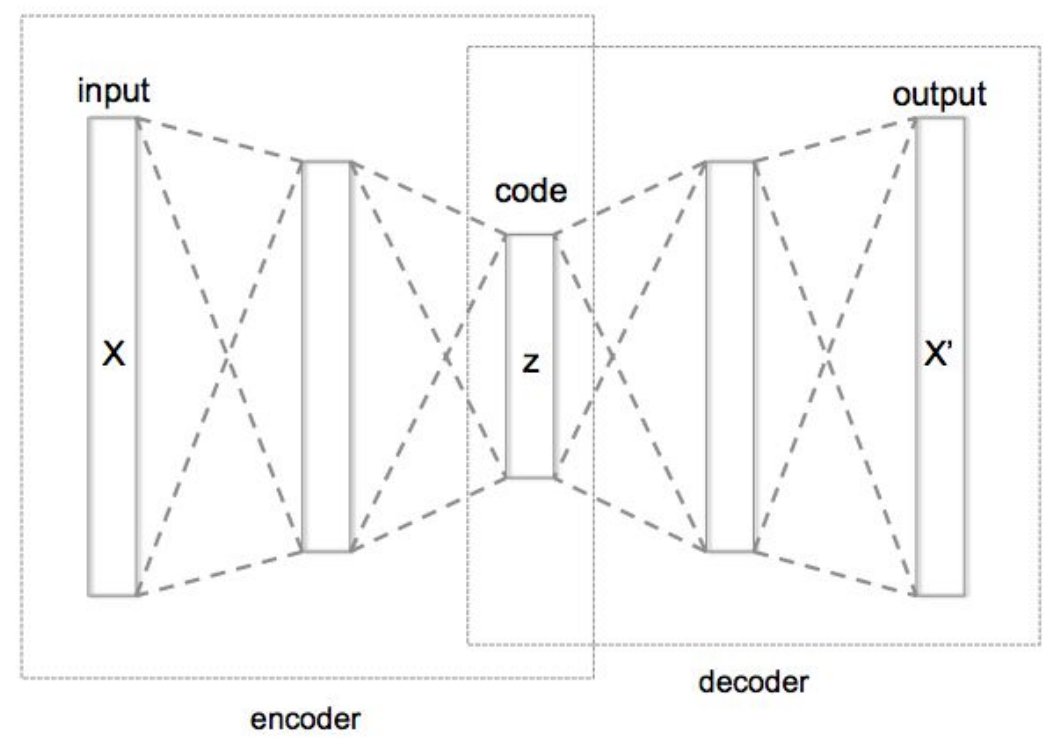
Upper-layer features in Neural Networks
(for labeled training data)



Train the network, then interpret the output of these neurons as high-level features!

(This is called *feature extraction*.)

Autoencoders
(for labeled or unlabeled training data)



Autoencoders are neural networks. Instead of having the network learn a *target*, you have the network learn to reconstruct the *input*.

The catch is that you force the information through a bottleneck hidden layer.

Converges on PCA...

Principal Components Analysis (PCA)

PCA is common technique for dimensionality reduction. It doesn't require labeled data!

The first goal of PCA is to remove correlation between features.

A side effect is that we can use the process to reduce the dimensionality of our data while preserving most of the variance in our data.

Let's derive PCA on the board!

Principal Components Analysis (PCA)
(summary from boardwork)

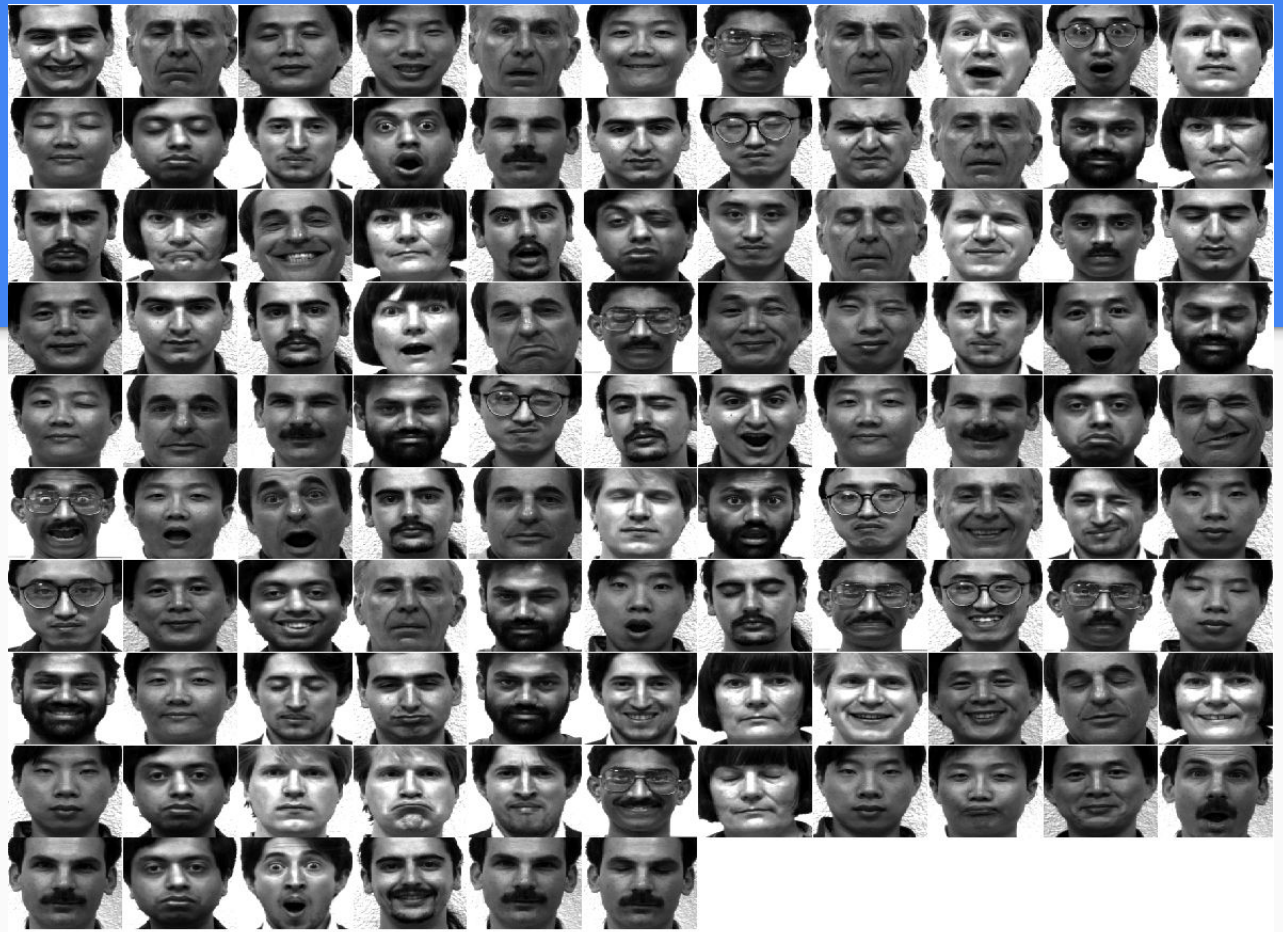
1. Create the *centred design matrix*.
   Center your data by subtracting the mean. Put your centered data in matrix form where each row is one example. Call this the *centered design matrix* $\boldsymbol{M}$.

2. Compute the covariance matrix.
   The covariance matrix is computed by $\boldsymbol{M^T M}$.

3. The principal components are the eigenvectors of the covariance matrix.
   Ordering the eigenvectors by decreasing corresponding eigenvalues, you get an uncorrelated and orthogonal basis capturing the directions of most-to-least variance in your data.

Our dataset

- From the *Yale Face Database* (using subset of 105 images)
- 320x243 pixels each, grayscale
- Centered and cropped identically



Let’s do PCA on this dataset of faces!

Pause… let’s make sure we’re all on the same page.

What do you expect will happen?

This! We call these **eigenfaces**, because they are the eigenvectors of the face database covariance matrix.

What do you see here? What is each eigenface capturing?

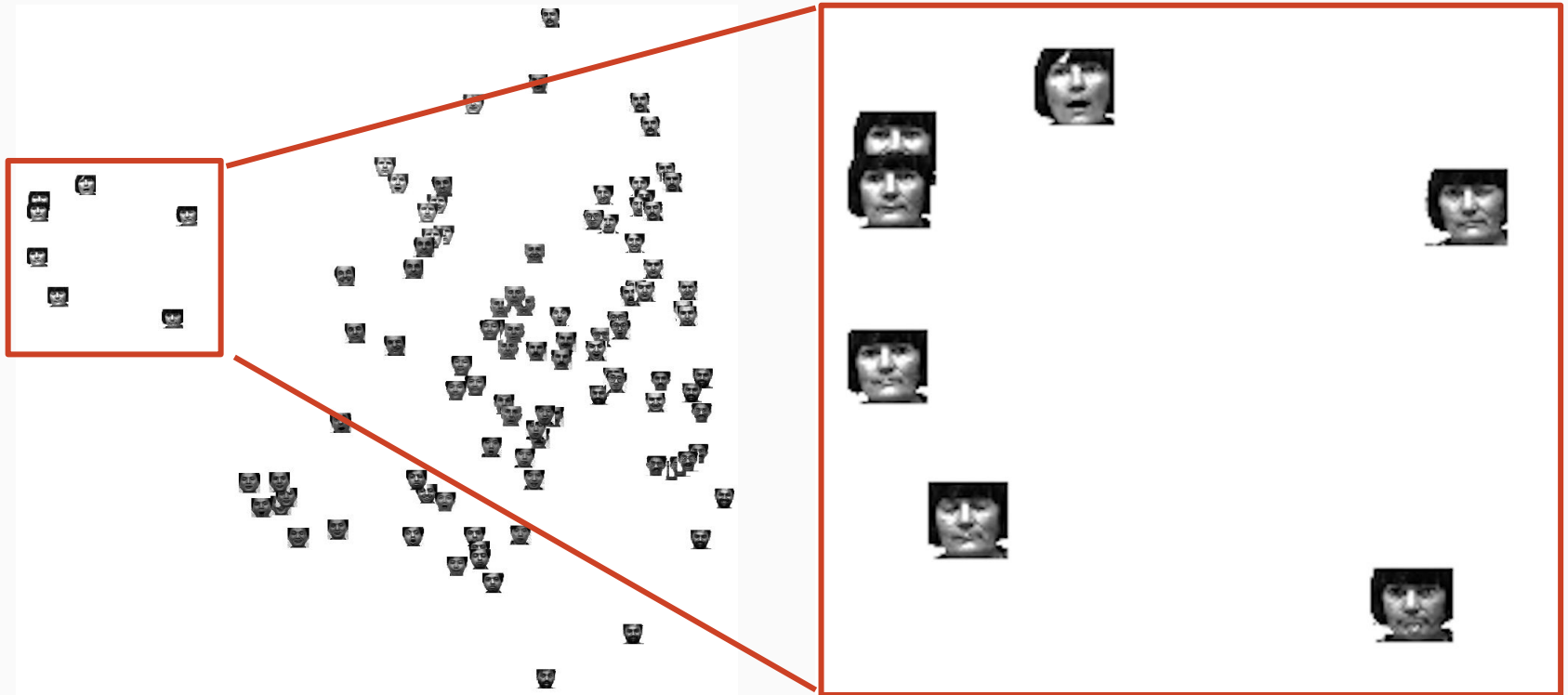
Drawbacks of Eigenfaces

- Faces must be aligned eyes-to-eyes, mouth-to-mouth -- differences in translation and scale are captured by PCA (which isn't what we want)
- Faces must be lit the same -- differences in lighting are captured by PCA (which isn't what we want).
- Old method: 1987, 1991

- Improvements:
  - "Fisherfaces": uses LDA and labels to help remove lighting effects
  - Use PCA on shapes detected in the image (search for Active Shape Model)

Crude facial recognition

Using PCA on cropped face images (i.e. eigenfaces), combined with kNN, we can do very crude facial recognition! To show this, let's look at the embedding onto 2d.



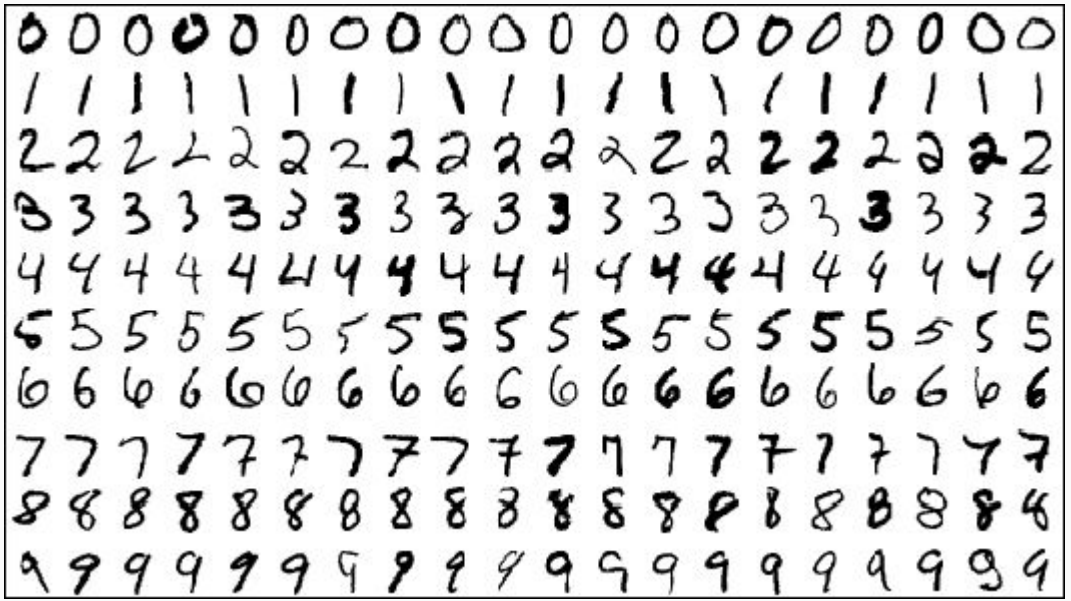
Crude facial recognition

MNIST Dataset

- Dataset of handwritten digits
- 10 classes (0-9)

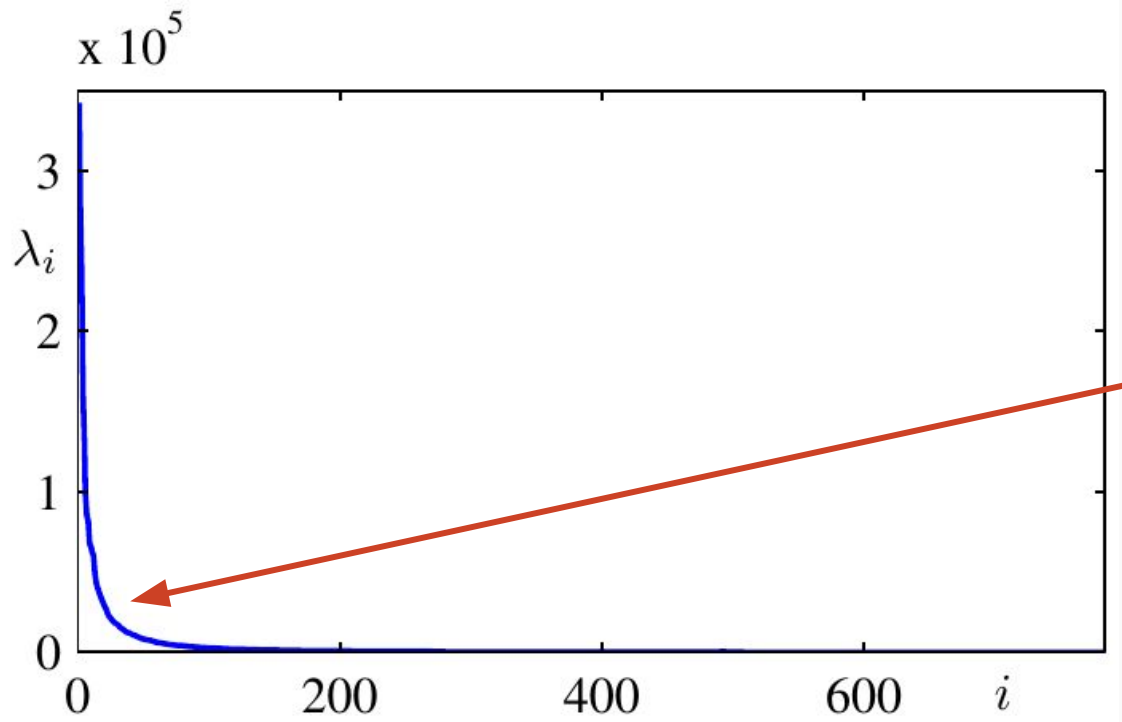
- 28x28 pixels, grayscale
  = 784 dimensions

- 60,000 training images
- 10,000 test images



PCA on MNIST: What are the sizes of the eigenvalues?

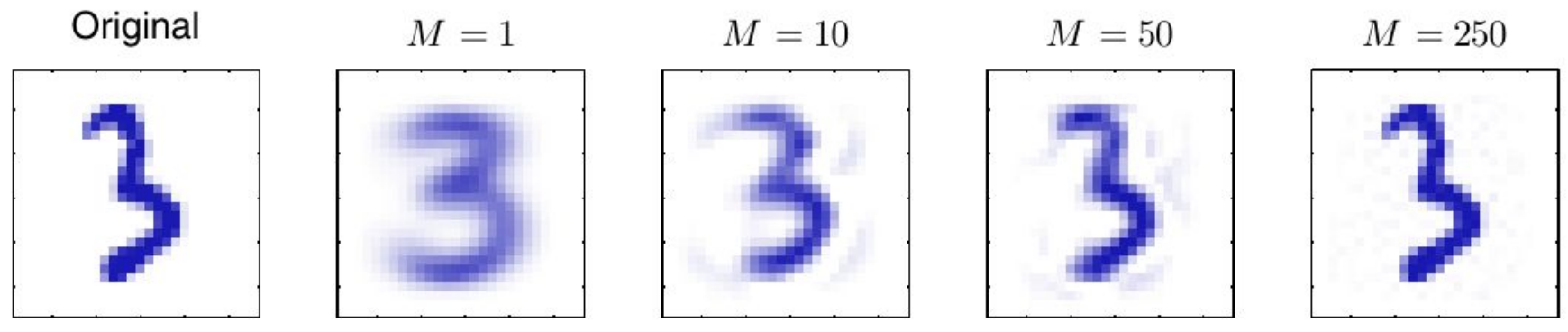
Recall: The size of each eigenvector's eigenvalue denotes the amount of variance captured by that eigenvector.



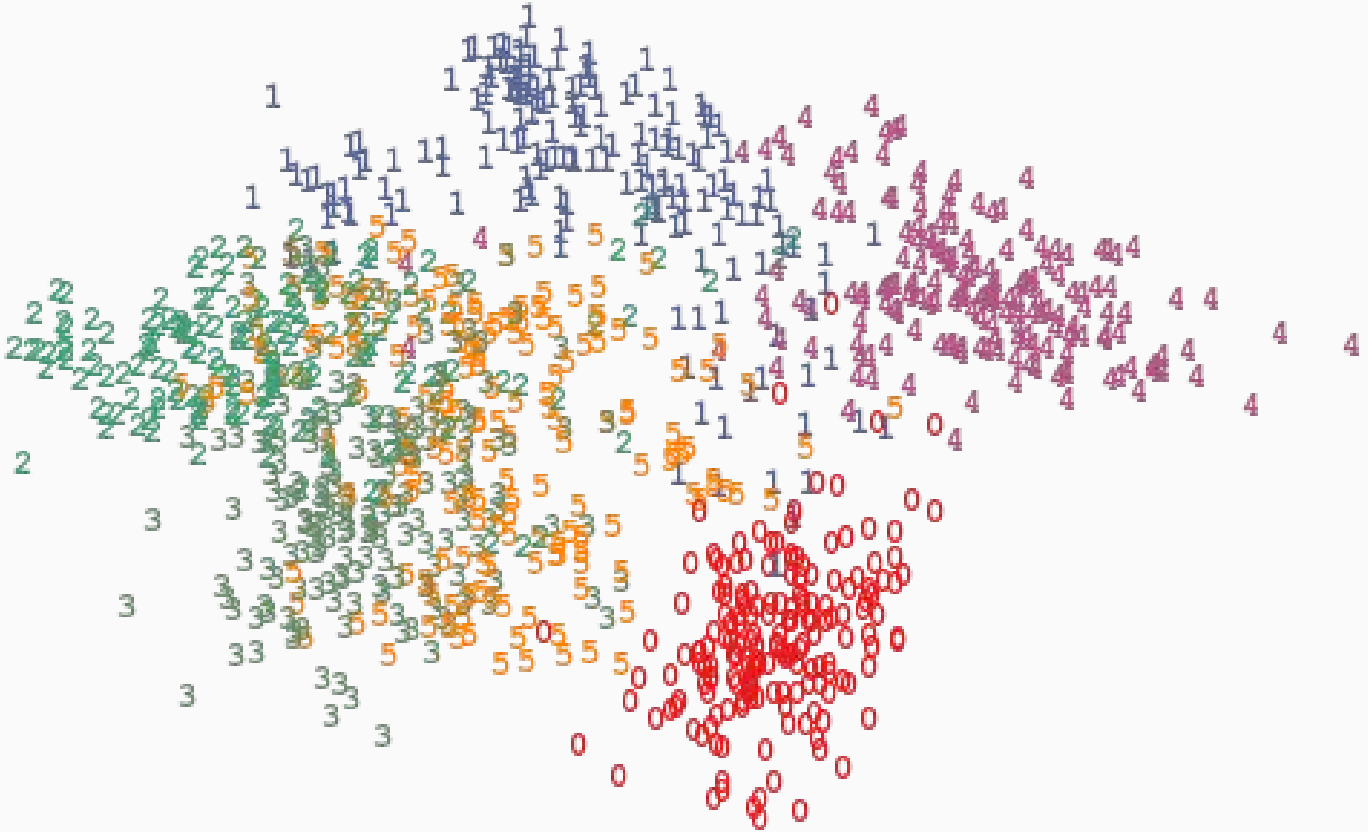
Look how fast the size of the eigenvalues drop!

Look for this "elbow".

PCA on MNIST: Reconstructing the input by a linear combination of eigenvectors



PCA on MNIST: Embedding in 2d



Dimensionality Reduction

(part 2 of 2)

Ryan Henning

- *Singular Value Decomposition* (SVD)
- SVD vs PCA
- SVD for capturing latent features

When to use PCA
(general advice only!)

Use when:

- kNN on high dimensional data
- Clustering high dimensional data
- Visualization
  (e.g. embeddings)
- Working with images
  (e.g. would it work well to feed an image into a decision tree?)

Don't use when:

- You need to retain interpretability of your feature space
- Your model doesn't *need* reduced dimensional data
  (e.g. OLS on relatively small data)

Singular Value Decomposition (SVD)

Boardwork...

SVD vs PCA

Boardwork...

SVD for capturing latent features

See IPython notebook…

The idea goes like this: Say our dataset maps from space *X* to space *Y*. E.g. *X* might be “user space”, and *Y* might be “movie space”.

After doing SVD on this dataset, we can interpret the SVD as mapping from *X* space to a “concept space”, then mapping from that “concept space” to Y space.

We then interpret this “concept space” as ***latent features*** of our dataset. We also sometimes call this space the “topic space”.