NAVAL
POSTGRADUATE
SCHOOL

OS4118
Statistical and Machine Learning

Introduction

Prof. Sam Buttrey

Fall AY 2020

Who Am I?

- Prof. Sam Buttrey, buttrey@nps.edu
- A.B., Princeton, Statistics; M.A., Ph.D., U. California-Berkeley, Statistics
- Naval Postgraduate School, Department of Operations Research, 1996-Present
- Interests: Data Analysis, Data Mining, Big Data Analytics and Computing, Classification, Modeling and Applications
- I have taught this course once before…
  - …with a slightly different set-up…
  - …apologies in advance

Admin Notes

- Course via Sakai & Collaborate
- I love email! buttrey@nps.edu
- Telephone: 831-656-3035
- **Travel**: I will miss 6 Nov and 13 Nov classes. My current plan:
  - Pre-record the lecture of 6 Nov
  - Guest lecturer 13 Nov; she will take emails during my absence
  - Sorry for the hassle here

Admin Notes 2

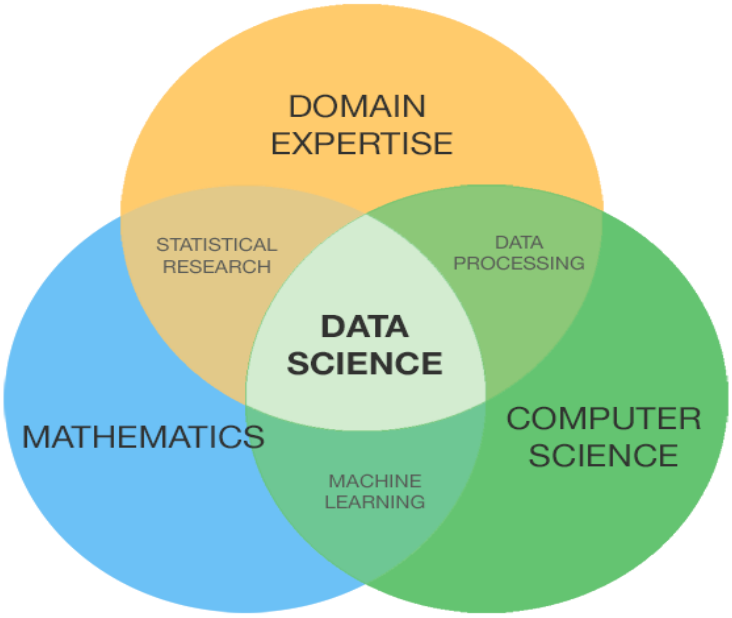
- Exercises: one per week, graded
- Grades: I'll record grades for the exercises on Sakai, but the final grade is mine
- R: I plan to demonstrate stuff in (regular) R; I'm willing to use RStudio too. We may have to turn to AWS – we'll discuss.
- CS approach vs Statistics approach

NAVAL POSTGRADUATE SCHOOL

Data Science vis-à-vis Statistics

Math and Statistics Competencies

- Statistical modeling
- Machine learning
- Bayesian inference
- Optimization
- Simulation
- Network science
- Model development



Source: Palmer, Shelly. Data Science for the C-Suite. New York: Digital Living Press, 2015. Print.

Domain Expertise Competencies

- Specific functional area
- Curious about data
- Influence with leaders
- Problem solver
- Make narratives w/data
- Visual design and communication
- Creative, innovative, and collaborative

Computer Science Competencies

- Scripting (Python)
- Statistical computing (R)
- Databases (No/SQL)
- Distributed storage (Hadoop Distributed FS)
- Distributed processing (MapReduce)
- Cloud computing (AWS)

Data pipelines (Pig/Hive)

Data Science is a team sport!

So Far (1)

- Great problems of statistics/analytics:
- **Regression**: we have a series of $(y_i, \boldsymbol{X}_i)$ observations, where the y_i are **numeric**
  - Typically meaning "continuous"
- Goal: predict E $(y \mid \boldsymbol{X}_0)$ for some $\boldsymbol{X}_0$ – or predict some other attribute of $y \mid \boldsymbol{X}_0$
- Tools: linear regression, regularization, regression trees/ensembles…
- Issues: categorical predictors; transformations; interactions;

Issues in Regression

- Ongoing issues
  - Incorporating **categorical** predictors
  - Incorporating **transformations** of *y*, or of one or more X's, or both
  - **Interactions**, where the relationship between *y* and *Xp* depends on the value of *Xq* (and higher-order interactions)
  - **Missing value** handling (both at the time we build the model and at prediction time)
  - Resisting the effect of **outliers**, anomalies, points of high influence

Extending the Regression Model

- Extensions: **integer** data
  - There is a version of GLM that allows the distribution of $y \mid \boldsymbol{X}$ to be Poisson
  - Negative binomial also sometimes used
  - Often there is over-dispersion or weirdly frequent zero counts ("zero-inflated Poisson")
  - **Log-linear models** for counts in tables
- **Repeated measures**
  - Common in experiments on humans
  - Introduces correlation structure for measurements made on the same person

Extending Regression (cont'd)

- **Random effects**
  - When, e.g.. the experimental units are a sample from a population of units – e.g. people – and we're interested in the distribution of effects across the whole population
- Multivariate ANOVA, for a vector-valued *y*
- Non-linear regression, common in chemical and biological applications
- Lots of extensions to these models!

So Far (2)

- **Classification**: we have a series of $(y_i, \boldsymbol{X}_i)$ observations, where the y_i are **categorical**
  - Often binary
- Goal: predict Pr $(y = 1 \mid \boldsymbol{X}_0)$ for some $\boldsymbol{X}_0$ – or the distribution of $y \mid \boldsymbol{X}_0$
- Tools: logistic regression, regularization, classification trees/ensembles…
- Evaluate performance through cross-validation, **test set** performance (but maybe with AUC rather than misclass rate)

Considerations

- Almost all of the issues with regression apply to classification:
  - Transformation, Interactions, missings etc.
- Other issues: **multinomial outcomes** (logistic regression difficult here)
- **Very rare** (or very common) events
  - Where the "naïve model" is very good!
- There is less information per observation in the classification problem

Model Evaluation

- It is critical to evaluate your model on new data – not on the same data you used to build it
- Where possible, an independent **test set** should be used. If not, cross-validation
- In a time series set-up, test set might refer to data after a cutoff (or rolling horizons)
- In a voice-recognition problem, test set would contain **people** not in training set
  - Not just **observations** not in training set

Model Evaluation (cont'd)

- The measure of evaluation should be relevant to the problem at hand
- In regression, "mean square prediction error" (or its square root) is common

$$= \frac{1}{n_{\text{test}}} \sum (y - \hat{y}_i)^2$$

- In two-class classification, we often use **sensitivity** and **specificity**, compute AUC
- Alternatively, **precision** (= #true pos / #pred. pos) and **recall** (= sensitivity = #true pos/#actual pos), ignoring #true neg

Multinomial Classification

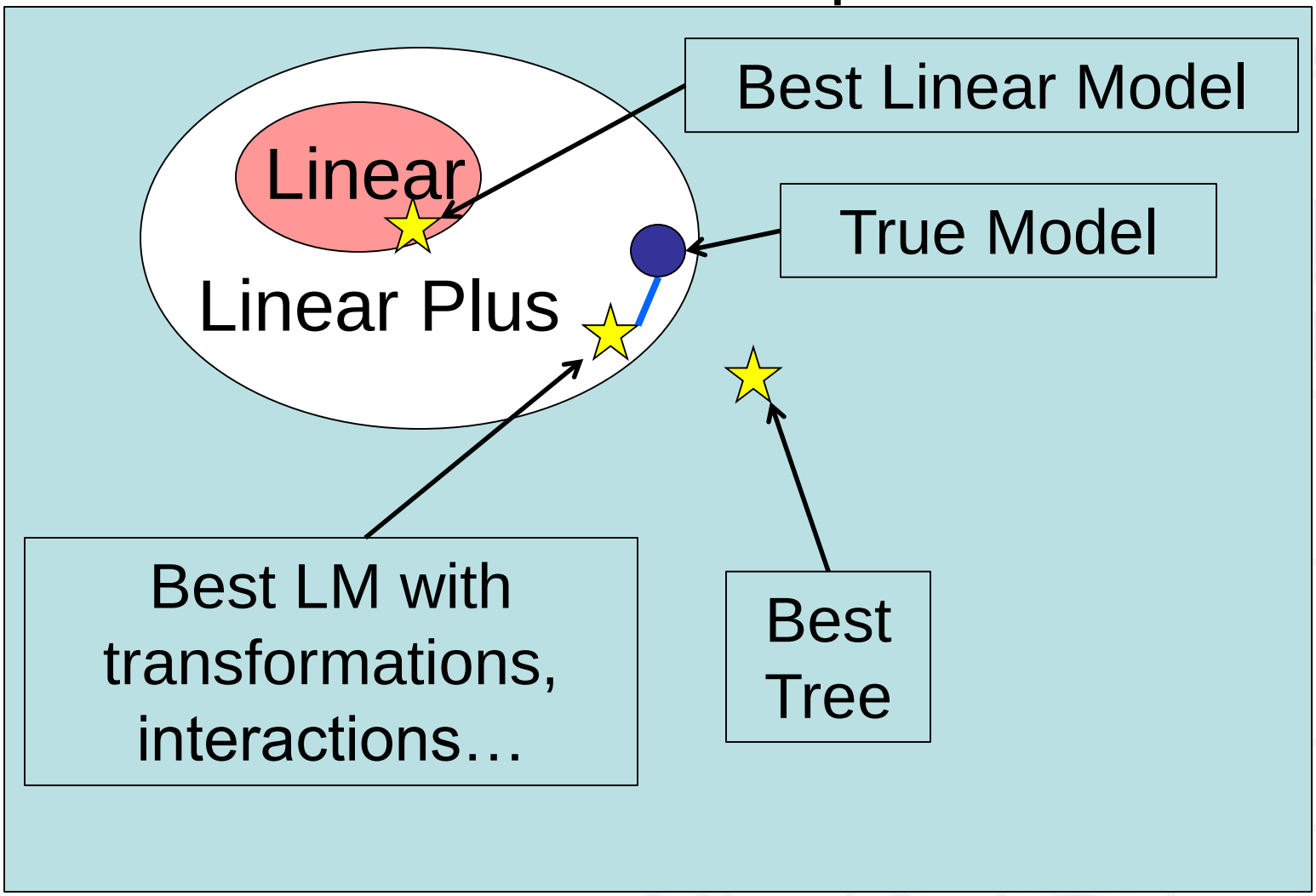
- In the multi-class case the approach is less clear
- The naïve model is often very bad
  - Mean of class-specific error rates? Maybe weighted by sample sizes?
  - $\sum_{i=1}^{n}\sum_{c=1}^{C}(y_{ic}-\hat{y}_{ic})^2$, where y_{ic} = 1 if obs. y is in class *c* and 0 otherwise? (Brier score)
  - Average of AUCs for “one-versus-all-other” classification models?
- I claim this problem is unsolved

Ensembles

- The two most exciting directions in machine learning over the last ten years have been in **ensembles** and neural networks
- We'll come back to neural networks
- Ensembles of automatically-generated "weak learners" can "vote" to produce classification or regression models that are very often better than our "one best model"

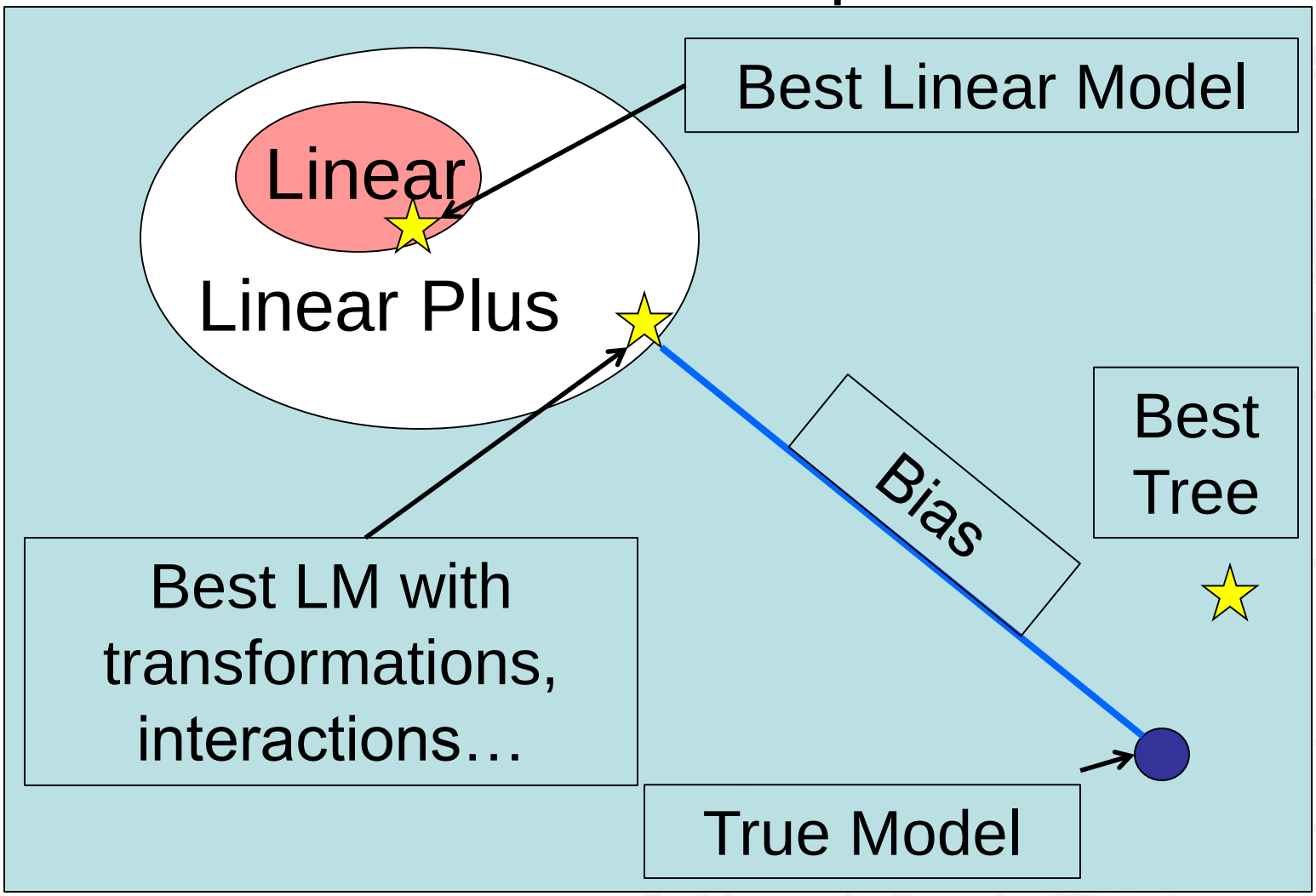
Bias vs Variance

All Relationships

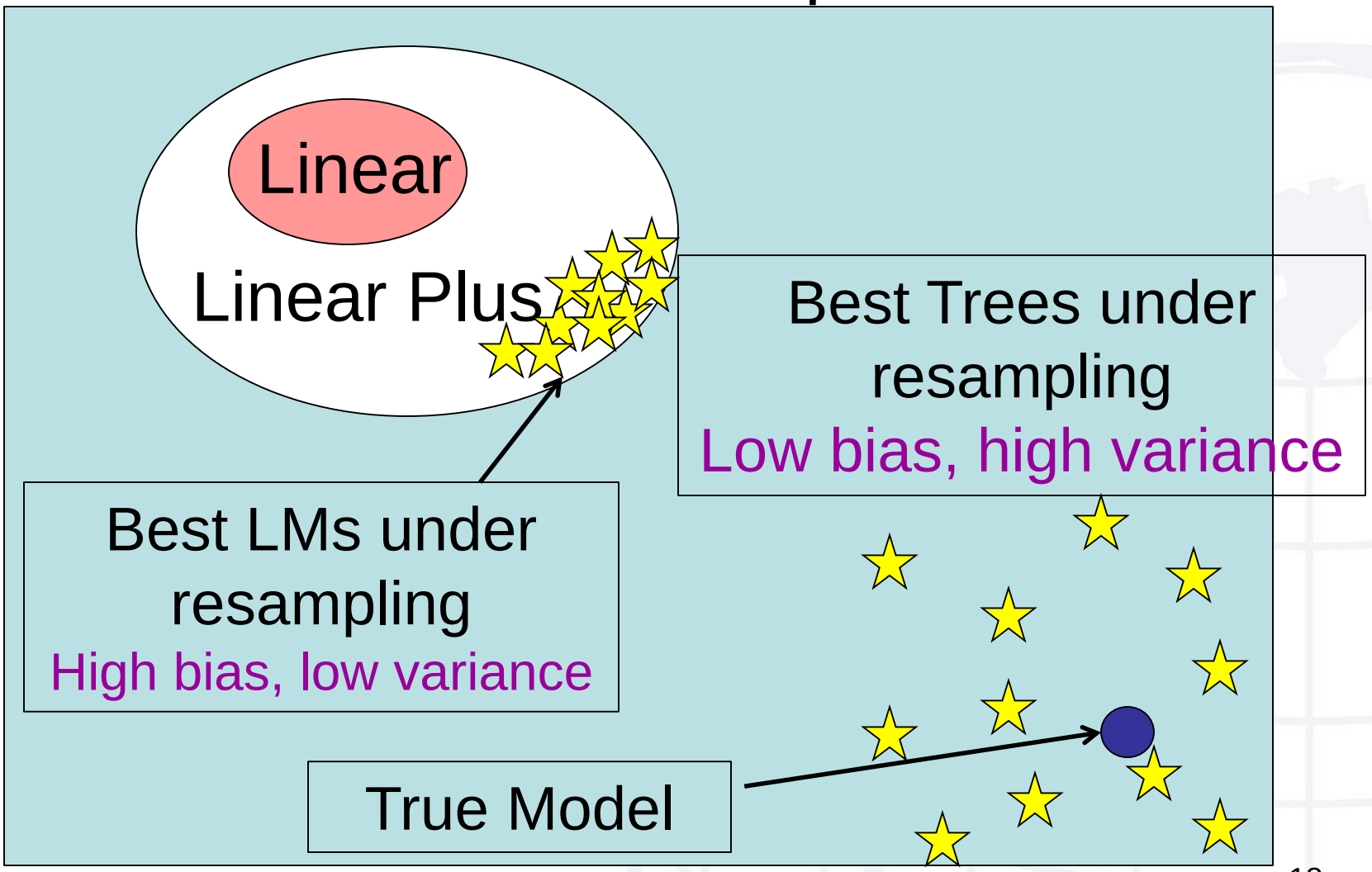


Bias vs Variance

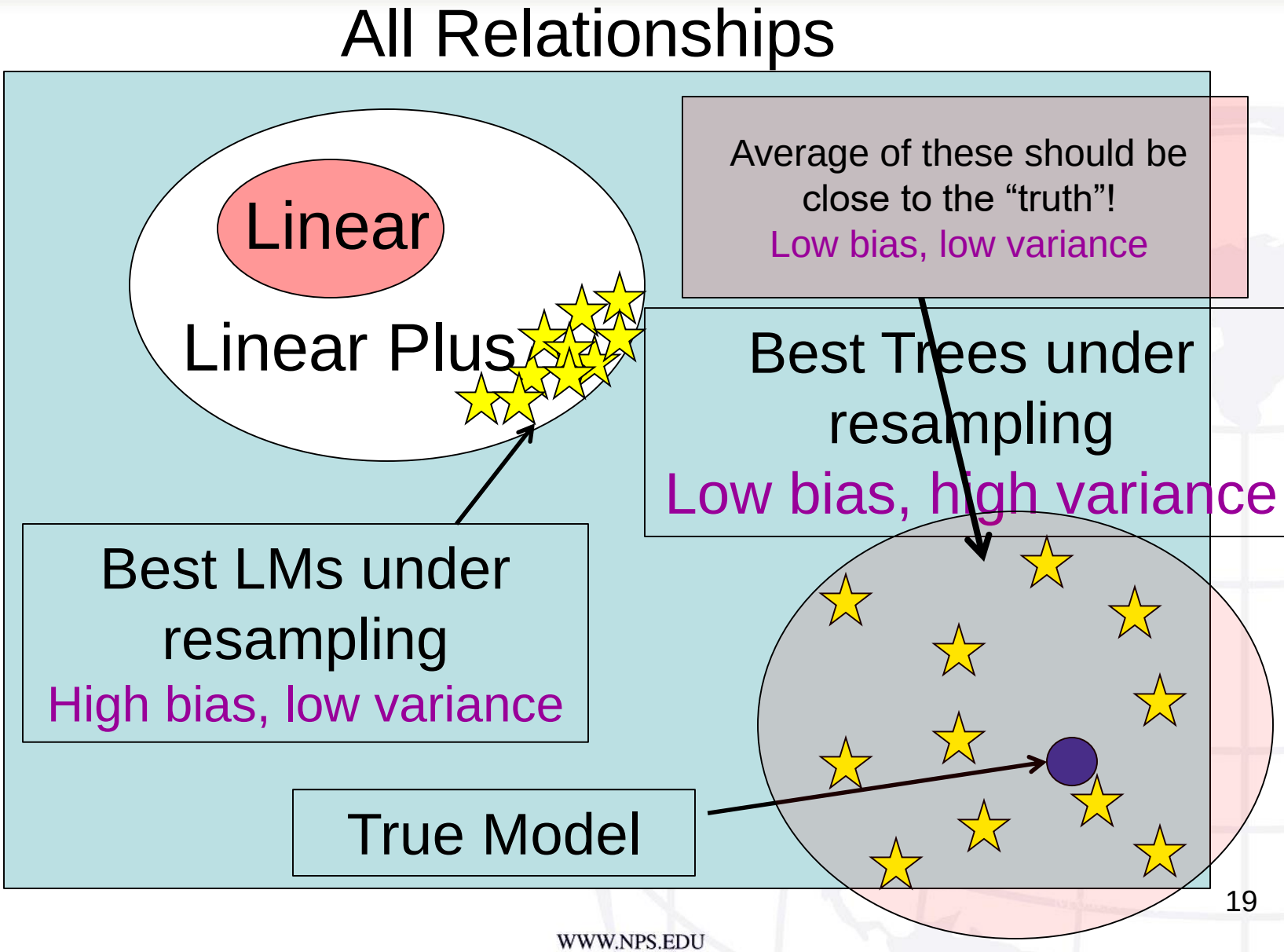
All Relationships



Averaging (Ensembles) Might Help



Averaging (Ensembles) Might Help



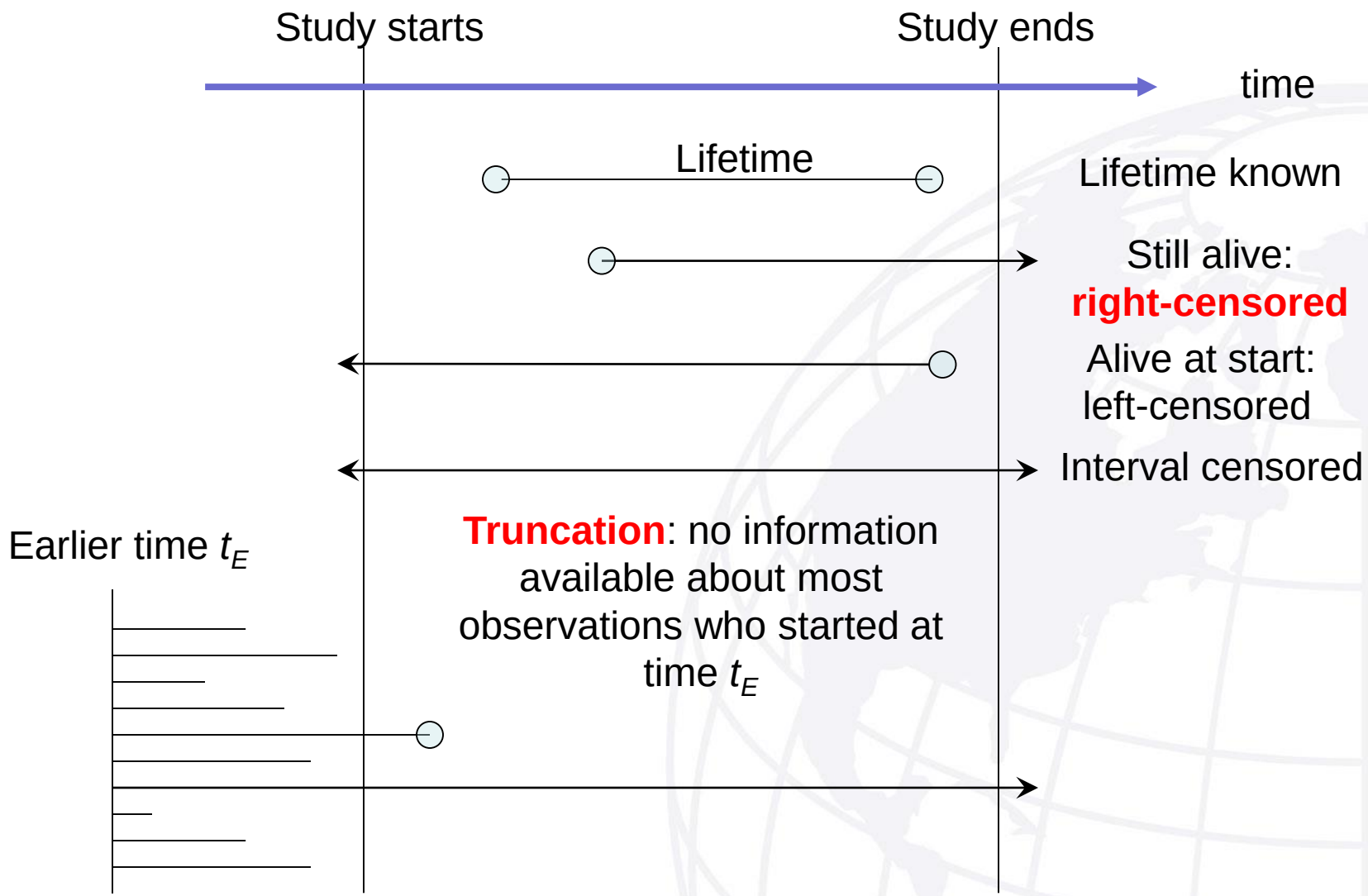
Other Important Problems

- **Time Series**, where observations share time dependency
  - This is a huge subject about which you saw some good info in OS 4106
- **Spatial Statistics**, where there is correlation, possibly directional, between observations close together
  - We will look briefly at some elementary aspects of this, especially making maps

Other Problems (cont'd)

- **Survival Analysis**: analysis of lifetime data
  - Issues: lots of the lifetimes are "right-censored," that is, not completely observed (observations are still living)
  - There can be left-censoring (birth dates / starting times unknown)….
  - Maybe more often we have "truncation," where we haven't observed all the people who were alive when we started

Survival Analysis



Supervised Learning

- **Supervised** learning is when there are known *y*'s
  - Classification and regression
  - With perhaps a linguistic stretch, time series, spatial analysis and survival analysis
- As opposed to **unsupervised**, in which there is just a bunch of *x*'s with no known *y*'s
  - Examples to follow, but the primary example is always clustering – dividing into groups

Learning, cont'd

- In **semi-supervised** learning we have a small amount of supervised data, plus lots of unsupervised data to help out
- In **reinforcement learning**, the algorithm gets "rewarded" for smart guesses
  - Like when your sibling hides something…
  - An important foundation for Artificial Intelligence algorithms and one very much worthy of serious study…
  - …but not in this course

Unsupervised Learning

Unsupervised learning: techniques with no specified response variable

1.) Principal Components (PCA)

- For numeric data, reduce dimensionality
- Remove noise; can also compress data

- Projection Pursuit (different criteria)
- Blind Source Separation
  - "Cocktail party problem"
- No Y variable

Unsupervised Learning

Unsupervised learning: techniques with no specified response variable

1.) Principal Components (PCA)

- For numeric data, reduce dimensionality
- Remove noise; can also compress data

- Projection Pursuit (different criteria)
- Blind Source Separation
  - "Cocktail party problem"
- No Y variable

Details to Follow

Unsupervised Learning

Techniques with no specified response variable, cont'd

2.) Clustering

- Find structure ("lumps") in observations
  - Group X's without knowing their y's
  - Usually we don't know number of clusters
- Important questions:
  - How do we measure the **distance** between observations – or clusters – if our measures are on different scales or categorical?

Unsupervised Learning

Techniques with no specified response variable, cont'd

2.) Clustering

- Find structure ("lumps") in observations
  - Group X's without knowing their y's
  - Usually we don't know number of clusters
- Important questions:
  - How do we measure the **distance** between observations – or clusters – if our measures are on different scales or categorical?

Details to follow

Unsupervised Learning

3.) Association rules (market baskets)

- Given a set of categorical data, deduce rules like "if a $\in$ A and b $\in$ B, then c $\in$ C"
- Each rule carries **coverage** (proportion of pop'n for which a $\in$ A and b $\in$ B) and **confidence** (proportion of obs with a $\in$ A and b $\in$ B for which c $\in$ C)
- No particular response specified; any variable can be "predictor" or "response"

Unsupervised Learning

3.) Association rules (market baskets)

- Given a set of categorical data, deduce rules like "if a $\in$ A and b $\in$ B, then c $\in$ C"
- Each rule carries **coverage** (proportion of pop'n for which a $\in$ A and b $\in$ B) and **confidence** (proportion of obs with a $\in$ A and b $\in$ B for which c $\in$ C)
- No particular response specified; any variable can be "predictor" or "response"

Details to follow

Unsupervised Learning

4.) Density Estimation

- Given data, what density might it have come from?
- One form: “Bump hunting”
- Particularly tough in high dimensions where data is very sparse
- Can sometimes be cast as a classification problem

- Anomaly / outlier detection

Unsupervised Learning

5.) Multidimensional scaling

- Map high-d data into 2 (or 3) dimensions while preserving inter-point distances (or maybe their ordering) to the extent possible

Details to follow, briefly

- Self-Organizing (Kohonen) Map

6.) Search

- PageRank
- Recommender Systems (Amazon, Pandora…)

Principal Components

- **Goals**:
  - Reduce dimensionality, maybe de-noise
  - Explore linear dependence among predictors
  - Produce uncorrelated X's for use in regression
- Basic idea: replace original measurements by a set of specially chosen weighted averages ("linear combinations")
- Basic PC is for **numeric data only**

Example

- Defense Language Institute study concerning program effectiveness
- $x_1, x_2, \ldots, x_6$ is a sequence of quarterly exam scores
- Do we need all six values?
- Maybe replace the six exam scores with
  - z_1 = average of the six exam scores
  - z_2 = (avg. of last 3) – (avg. of first 3)
  - This creates a **two-dimensional** data set
  - Some information is lost

Principal Components

- In principal components, we first find p orthogonal linear combinations of the p original regressors
  - "Orthogonal" means both "at right angles" and also "uncorrelated" here
  - No information created or destroyed, although…
  - We often then drop some of the new combinations, achieving…
  - **Dimensionality reduction**

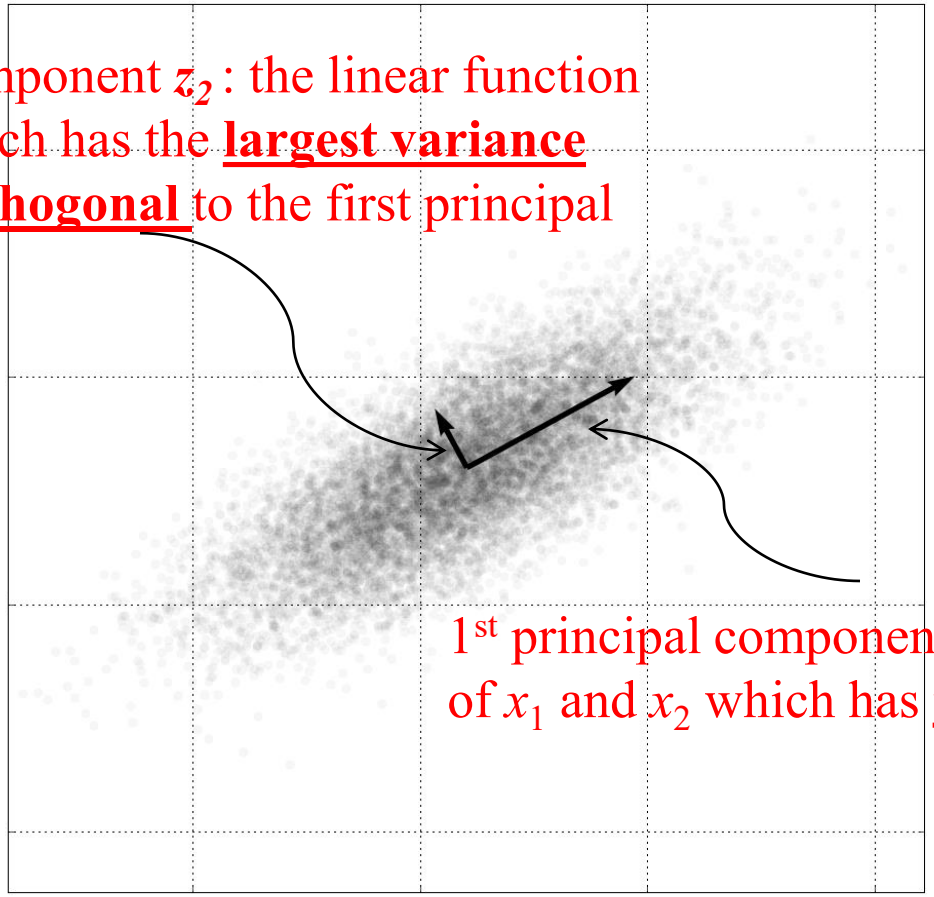
Principal Components

- These new linear combinations are the principal components – i.e. we re-express our X's in terms of p orthonormal basis vectors
- The first principal component points in the direction of the principal axis of the data – the direction in which there is most spread (that is, the "largest variance")

Two-dimensional Example

If we were to replace x_1 and x_2 with **one** variable, z_1 would be a natural choice; it has more variablity than either variable x_1 or x_2.

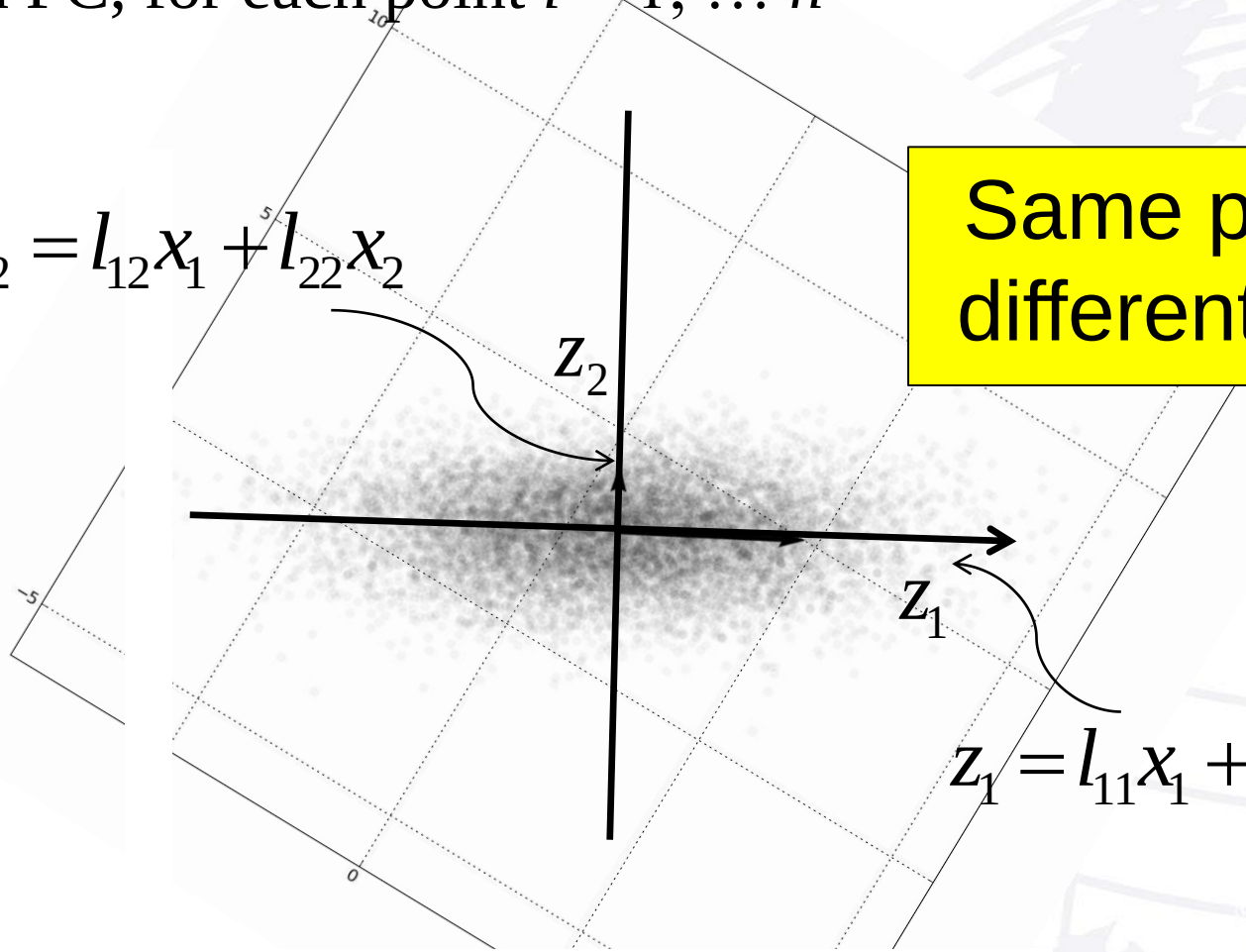
2nd principal component z_2 : the linear function of x_1 and x_2, which has the **largest variance** among those **orthogonal** to the first principal component z_1 .



1st principal component z_1 : the linear function of x_1 and x_2 which has **the largest variance**

Transforming Rotates the Axes

Here we plot (z_{i1}, z_{i2}) where $z_{i1} = l_{11}x_{i1} + l_{21}x_{i2}$ are the scores from the first PC and $z_{i1} = l_{11}x_{i1} + l_{21}x_{i2}$ are the scores from the second PC, for each point $i = 1, \ldots n$



Same points, different axes

For p variables there are p p.c.'s

- The p principal components are:

$$z_1 = l_{11}x_1 + l_{21}x_2 + \ldots + l_{p1}x_p = \mathbf{l_1}^{\mathrm{T}}\mathbf{x}$$
$$z_2 = l_{12}x_1 + l_{22}x_2 + \ldots + l_{p2}x_p = \mathbf{l_2}^{\mathrm{T}}\mathbf{x}$$
$$\vdots \qquad \vdots \qquad \vdots \qquad \vdots$$
$$z_p = l_{1p}x_1 + l_{2p}x_2 + \ldots + l_{pp}x_p = \mathbf{l_p}^{\mathrm{T}}\mathbf{x}$$

loadings

where the vectors of **loadings** are orthogonal to one another and have length 1

- The loadings are just weights; we take p weighted averages of our p variables

PC Scores Have Maximal Variance

- The **scores** are the values of the PCs for each observation
- E.g., $z_{i1} = l_{11}x_{i1} + \ldots + l_{p1}x_{ip}$ is the **score** of the i^{th} observation on the first PC
- We choose the loadings so that the first PC scores have the largest variance among all linear functions of the x's
  - (for which the coefficient vector has length 1)
- The second PC scores have the largest variance among all linear functions (w/len 1) which are orthogonal to the first…etc.

Why Variance?

- A direction with no variability is uninformative; maybe one in which there is lots of variability is very informative
- Choosing scores with maximum variance leads to straightforward computation
- Other indices of “interestingness” can be used
- PCs widely used, seem to be successful

Finding PCs in Practice

- In R, use `prcomp()` or `princomp()`
  - `prcomp()` (stronger but less friendly) uses the *n* – 1 divisor for variances, calls scores `x`
  - `princomp()` uses *n*, calls scores `scores`
  - Both **center** the data by subtracting means; we'll need those values for scoring new observations
  - Neither **scales** by default; we usually do that
- Remember: we convert data to scores
- Start with *p* columns, convert to *p* scores

How do we find the loadings?

- Let *S* be the (sample) cov. matrix of the x's
- cov (*X*) =

$$\begin{bmatrix} s_{x_1}^2 & \widehat{\text{cov}(x_1, x_2)} & \cdots & \widehat{\text{cov}(x_1, x_p)} \\ \widehat{\text{cov}(x_2, x_1)} & s_{x_2}^2 & \cdots & \widehat{\text{cov}(x_2, x_p)} \\ \vdots & \vdots & \ddots & \vdots \\ \text{cov}(x_p, x_1) & \widehat{\text{cov}(x_p, x_2)} & \cdots & s_{x_p}^2 \end{bmatrix}$$

$$s_{x_j}^2 = \frac{\sum_{i=1}^{n} (x_{ij} - \bar{x}_j)^2}{n-1} \qquad \hat{\text{cov}}(x_j, x_k) = \frac{\sum_{i=1}^{n} (x_{ij} - \bar{x}_j)(x_{ik} - \bar{x}_k)}{n}$$

Some programs divide by *n* rather than *n*– 1 when computing PC's

Finding Loadings

- We want to find l such that $l\,S\,l^T$ is maximized, subject to the constraint that $l\,l^T = 1$
- Method of Lagrange multipliers: construct $f = l\,S\,l^T - \lambda\,(l\,l^T - 1)$, take derivatives...
- Solution: we want $l\,S = \lambda\,l$
- **That is**, the first principal component is precisely the first eigenvector of the matrix S
- Subsequent eigenvectors give subsequent principal components

Decomposing the S Matrix

- The loadings are the eigenvectors of S
- $S = L\,\Lambda\,L^T$
- $[\boldsymbol{L}_1, \boldsymbol{L}_2, \ldots, \boldsymbol{L}_p] \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_p \end{bmatrix} \begin{bmatrix} \boldsymbol{L}_1 \\ \boldsymbol{L}_2 \\ \vdots \\ \boldsymbol{L}_p \end{bmatrix}$

Column eigenvectors are the loadings for each PC

The j^{th} eigenvalue is the sample variance of the scores corresponding to the j^{th} PC.

i.e. $\lambda_j = s_{z_j}^2$ for $j = 1, \ldots, p$

And $\lambda_1 \geq \lambda_2 \ldots \lambda_p \geq 0$

$\sum s_{x_j}^2 = \sum \lambda_j = \sum s_{z_j}^2$

Scaling the X's

- The sum of the original data columns' variances is equal to the sum of the score columns' variances
  - But differently apportioned
  - Some have lots, some have hardly any
- If the x's are in different units, like lb and °F, their variances won't be additive
  - The analysis doesn't care, but we do, because changing to kg and °C shouldn't change anything

Scaling the X's

- "If the x's are in different units, their variances won't be additive"
- In that case, it's usual to **scale** the columns by dividing each by its SD
  - In practice we will usually do this, with `scale = TRUE` for `prcomp`
  - Loadings are **`rotation`**, scores are **`x`**
- After scaling, the total variance across the score columns will add up to p
- Example 0: Made-up, low-d data
- Example 1: The iris data

Iris Example

```
> (iris.pc <- prcomp (iris[,-5])) # extra () says "print"
```

I didn't scale here because all the measurements all already in inches

```
Standard deviations:
[1] 2.0562689 0.4926162 0.2796596 0.1543862
```

```
> round (prcomp (iris[,-5])$rotation, 3)
```

	PC1	PC2	PC3	PC4
Sepal.Length	0.361	-0.657	0.582	0.315
Sepal.Width	-0.085	-0.730	-0.598	-0.320
Petal.Length	0.857	0.173	-0.076	-0.480
Petal.Width	0.358	0.075	-0.546	0.754

Signs within a column can all be reversed and still give the same answer

We hope for a few interpretable components, but interpretation is often difficult

Interpreting Coefficients

- Each iris is described by four numbers
- Suppose you wanted one number – one weighted average – for each flower, such that those numbers were as spread out as possible. How would you do that?
- Answer: .361 (Sepal Len) – .085 (Sepal Wid) + .857 (Petal Len) + .358 (Petal Wid)

  $-.361^2 + (-.085)^2 + .857^2 + .358^2 = 1$
- That linear combination accounts for 92% of all the variability in all 4 original columns

Scores

- Scores (`x`) are returned automatically
- The score columns are uncorrelated, as are the columns of loadings
- Using `library (rgl)`, compare:
- `cc <- rep (c("red", "blue", "black"), each=50)`
- `plot3d (iris[,1:3], col=cc)`
- `plot3d(iris.pc$x[,1:3], col=cc)`

How Many PCs Should Be Kept?

```
> summary (iris.pc)
Importance of components:
                          PC1     PC2     PC3     PC4
Standard deviation     2.0563 0.49262 0.2797 0.15439
Proportion of Variance 0.9246 0.05307 0.0171 0.00521
Cumulative Proportion  0.9246 0.97769 0.9948 1.00000
```

Standard deviations of Score Columns s_j for $j = 1, \ldots, p$

Proportion of Variance of Score Columns: $s_j^2 / \sum_k s_k^2$

Rules of Thumb:

Keep the PCs as long as

- Proportion of the Variance > .10
- Cumulative Proportion > .90, .99
- If scaling, keep any w/ $s > 1$

Or

- Look for a "knee" in the scree plot

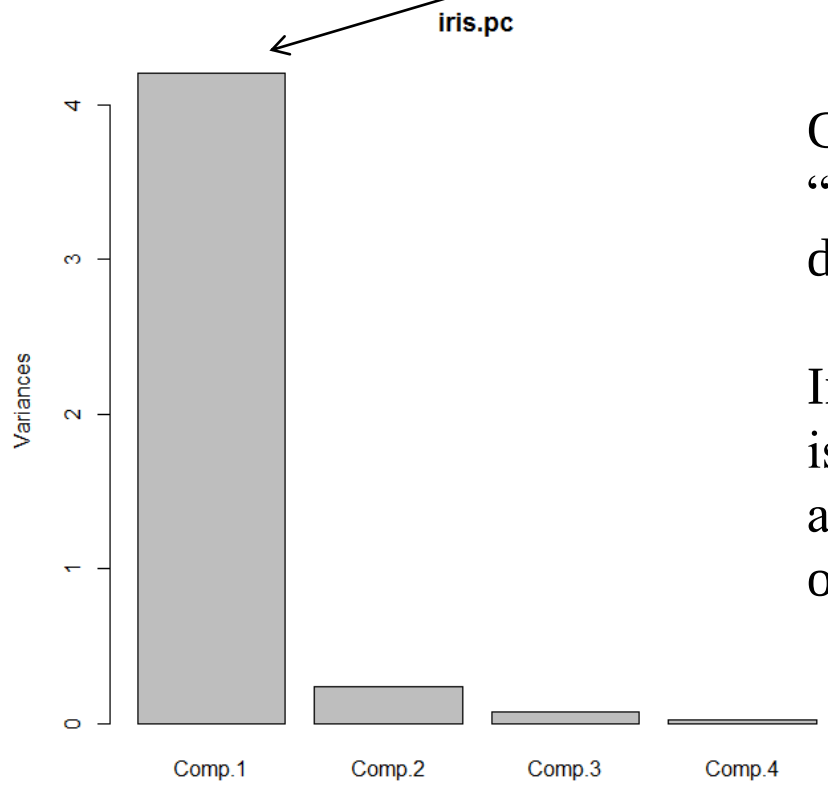
Cumulative Proportion of Variance

$$\frac{\sum_{i=1}^{j} s_{z_i}^2}{\sum_{i=1}^{p} s_{z_i}^2} \quad \text{for } j = 1, \ldots, p$$

Scree plot

```
> plot(iris.pc)
```

$s^2_{z1} = 4.2$



Common rule: look for a "knee" in the plot, where the descent slows

In the iris example, 1 or 2 PCs is enough to capture "almost all" of the variability in the original data

State Example

- Built in `state.x77` data
- Why is scaling necessary?
- R uses %*% for matrix multiplication
- Standardized data <- (data – center) / scale
  - Remember R acts columnwise; see example
- Then `stddata %*% rotation = scores`, and `scores %*% t(rotation) = stddata`
  - No information lost or created, but…
  - What if we drop a few PCs here?

PC Recap: States Example

- In the `state.x77` example, data is 50 × 8
  - 50 points lying in an 8-dimensional space
- Full set of PC loadings was 8 × 8
  - Instructions for converting from one space to another
- Scores = data `%*%` loadings
  - 50 points' coordinates in the new 8-d space
  - Sometimes we explicitly scale and/or center; then scores are (data – center)/scale `%*%` loadings

PC Recap: States Example

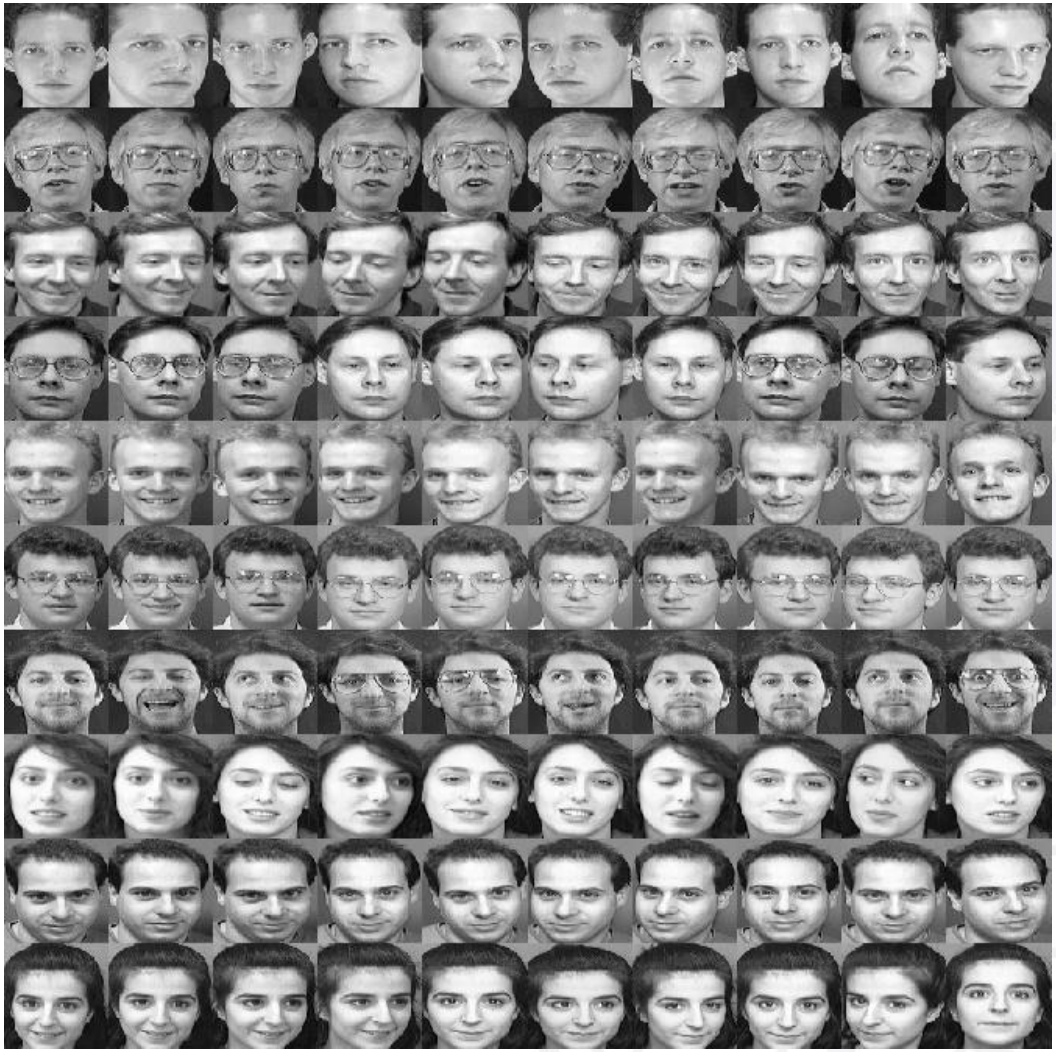
- Scores = data %*% loadings (weights)
- data = Scores %*% t(loadings)
  - Or (Scores %*% t(loadings)) * scale + center
  - No information gained or lost, until…
- We reduce dimensionality by using only a subset of scores, and a subset of the columns of loadings
- Then we lost a little information but may still have a decent approximation

Principal Comp Bigger Example

- Here's one example of a way that principal components can be used in practice: facial recognition
- Training set: Olivetti Research Lab, Cambridge UK (1994)[1]
- 40 subjects × 10 images/subject × (92×112) pixels/image × 255 gray levels per pixel
- Task 1: Read 400 images into R object `faces`
  - I used `read.pnm()` from `library (pixmap)`
  - Plot method available
  - This produces an S4 object; use `@` to extract
  - This data has 400 rows and 10,304 columns
  - We can keep fewer columns and still do well

1. F. Samaria and A. Harter, "Parameterisation of a stochastic model for human face identification," 2nd IEEE Workshop on Applications of Computer Vision, December 1994, Sarasota (Florida).

Examples



The Approach

- Computing distances between a training 10,000-vector and a test one is expensive, and the pixels are highly correlated
- Let's reduce the dimensionality by finding 382 PCs and keeping, say, 200 or so
- Each training set picture can be represented **exactly** with a score vector of 382 numbers, or approximately with 200
- For a test set item we get its 200-vector of scores and compare it to training set scores

The Eigenfaces

- Use `prcomp()` to get loadings, scores
- The loadings are called **eigenfaces** here
- Each "real" face is a linear combination of eigenfaces
- Perhaps we only need to keep a "small" number of these, $m' < m$ – perhaps 200?
- Test set: Subject 17 plus some randoms
- For a new face, convert it into a vector of 200 scores, compare it to all other vectors of scores, and see where it falls

Let's Do This Thing

- Lots of typing, so just follow along
- But the story is:
  - Create training and test sets
  - Compute PCs for training set; keep 200
  - Use those loadings to compute PC scores for test set, too
  - For each test set item, measure its distance to each of the training set items
  - Assign the test set item to the person depicted in the nearest-neighbor within the training set

Or some other reasonable number

Experiment

- How well did we do?
- What happens if you project a random pixel vector into face space?
- Answer: you get a vaguely face-like thing, a weird combination of eigenfaces
- Its scores won't be near any actual faces'
- "Rule": if distance to some picture is small, this is a face we've seen; if it's medium, this is a new face; if it's large, this isn't a face at all

Final Thoughts

- PC not well-suited to categorical data…
  - …Or data that varies with time and space
- In many problems we represent our data with a rich set of basis functions
  - Original measurements
  - Polynomials and products (interactions)
  - Transformations like log, sin/cos…
- In PC we select a small set of basis functions from a large original set, but..
- Lots of good choices aren't in the PC basis