

A Quick Introduction to Machine Learning

Paul Rodriguez
(and Mai Nguyen)
SDSC



Overview

- **Terminology and Key concepts**
- **Modeling and Machine Learning**
- **Main Activities of Modeling**
- **R and HPC**
- **Deep Learning and Digit Recognition**

Lots of Terms:



Lior Rokach
Department of Information Systems Engineering
Ben-Gurion University of the Negev

Lots of Terms: what are the key ideas?



Lior Rokach
Department of Information Systems Engineering
Ben-Gurion University of the Negev

Lots of Terms: what are the key ideas?



Lots of Terms: what are the key ideas?



*Some human-like processing
e.g. face detection, person tracking,
object recognition*

Lots of Terms: what are the key ideas?



Lots of Terms: what are the key ideas?

gathering data, looking for patterns
and associations –
e.g. who buys what products?



Lots of Terms: what are the key ideas?

BIG DATA:

*e.g. social networks, internet activity,
big science, etc...*

Lots of Terms: what are the key ideas?

BIG DATA:

*e.g. social networks, internet activity,
big science, etc...*

PREDICTIVE ANALYTICS:

*predicting unobserved data,
e.g. recommending movies
(in contrast to classical inferencing
about in-sample statistics)*

Lots of Terms: what are the key ideas?



Lots of Terms: what are the key ideas?



improving system performance with data
e.g. statistical learning,
e.g. models with algorithms for fitting parameters

Machine Learning Steps

- Gather & Prepare data
- Explore data – *e.g. know your variables*
- Build Model – *e.g. simple and complex*
- Evaluate

then perhaps iterate

Gather and Prepare Data

- Cleaning
- Transforming
- Organizing the data matrix
(aka 'data wrangling' or 'data munging')
- Variable Selection/Dimension Reduction

Explore Data – know your variables

- Summary statistics
- Check missing values
- Visualize:
 - Plot pairwise correlations
 - Plot histograms

Summary Statistics on Variable in R

- Summary statistics on individual variables
(*df* is a data frame)

```
> mean(df$MinTemp)
```

```
[1] 7.265574
```

```
>
```

```
> var(df$MinTemp)
```

```
[1] 36.31026
```

```
>
```

```
> sd(df$MinTemp)
```

```
[1] 6.0258
```

```
~ |
```

```
> summary(df$MinTemp)
```

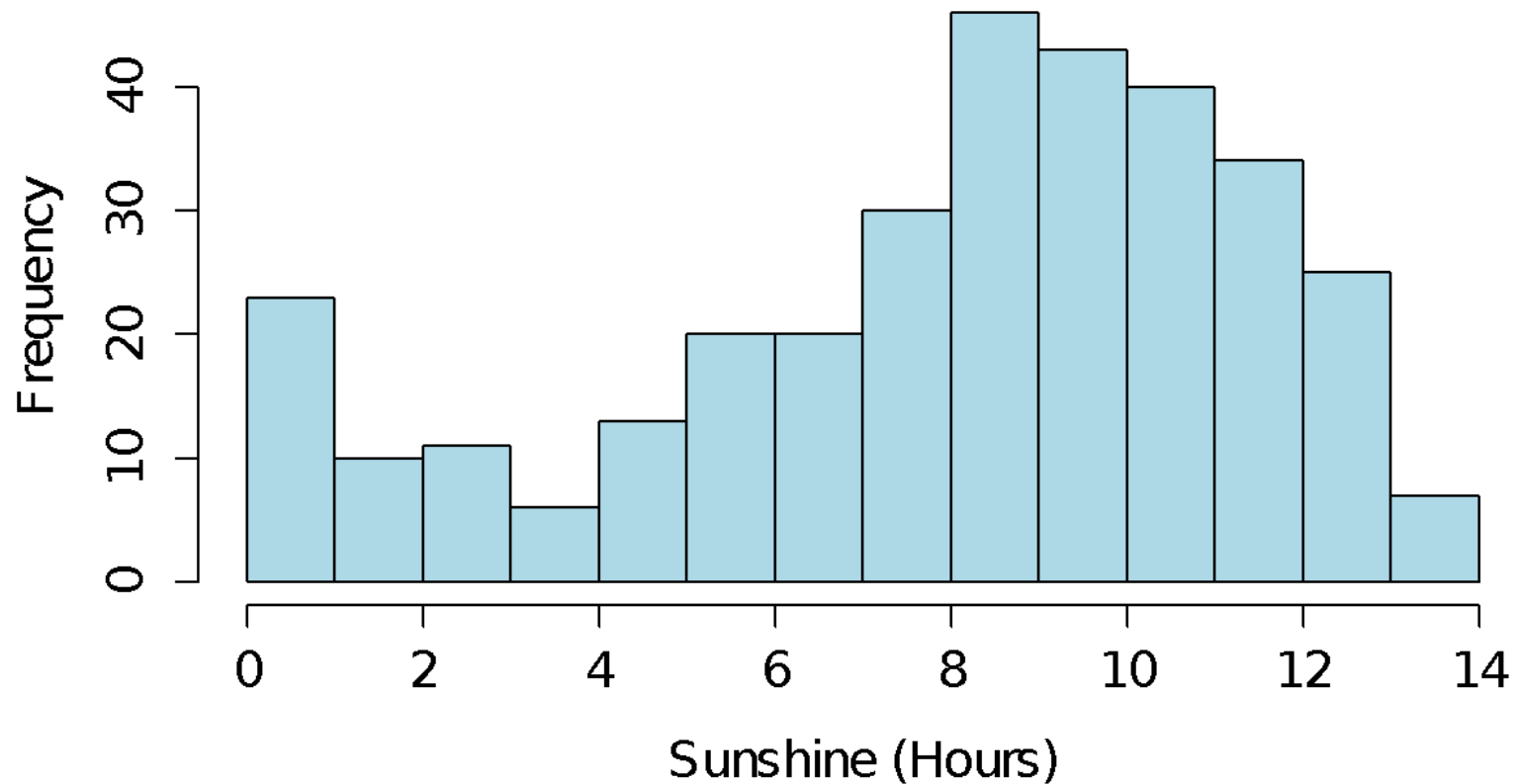
Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
-5.300	2.850	7.900	7.743	12.800	20.900

```
~ |
```

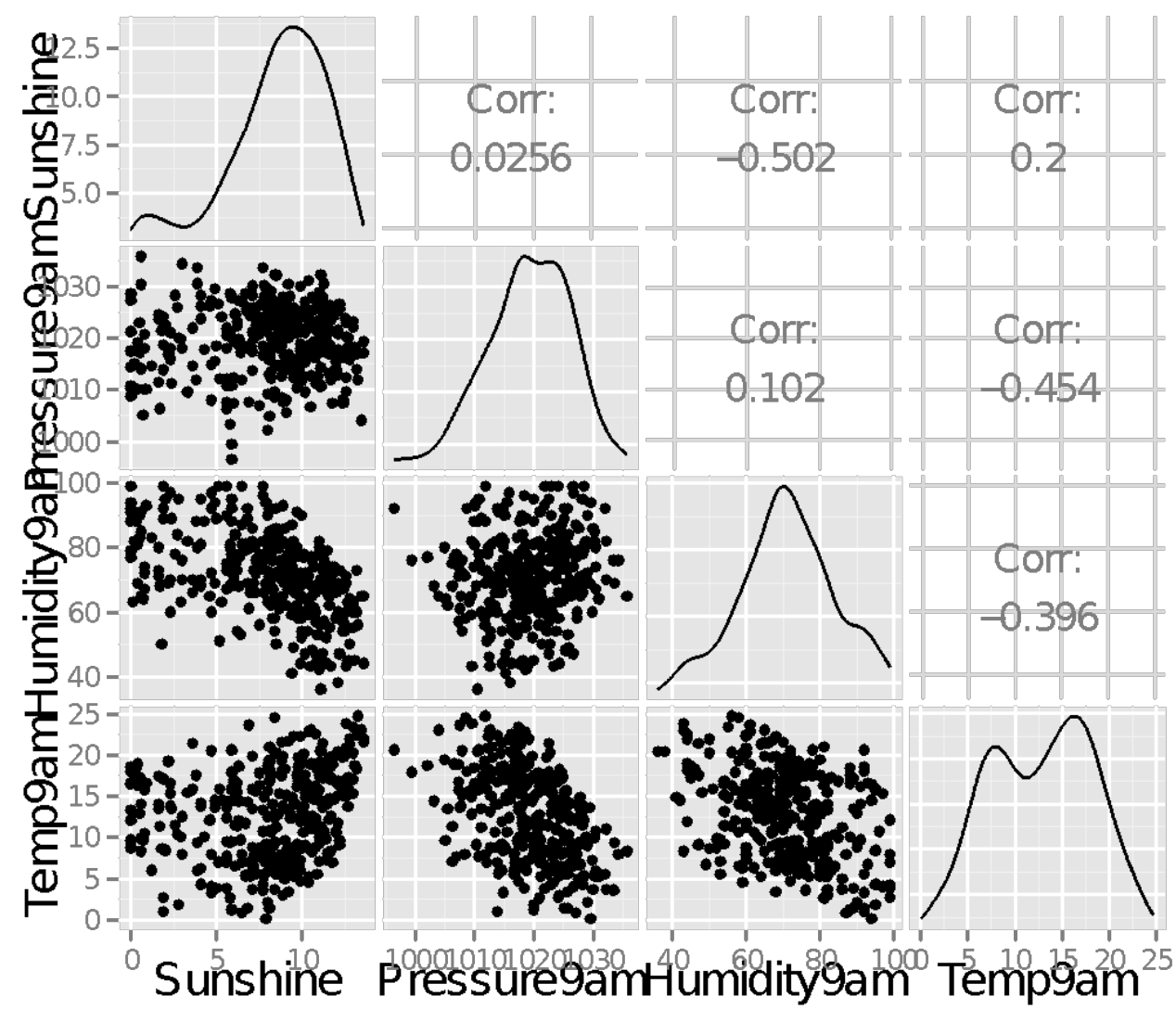

Histogram in R

```
hist(df$Sunshine,col="lightblue",main="Histogram of Daily Sunshine",xlab="Sunshine (Hours)")
```

Histogram of Daily Sunshine




```
ggpairs(df[c("Sunshine", "Pressure9am", "Humidity9am", "Temp9am")])
```



**Pairwise
Correlation
Plot in R**


Machine Learning Models

- **Classification**
- **Regression/Predictive**



Supervised (dependent variable or outcome labels given)

- **Cluster**
- **Matrix Factorization**

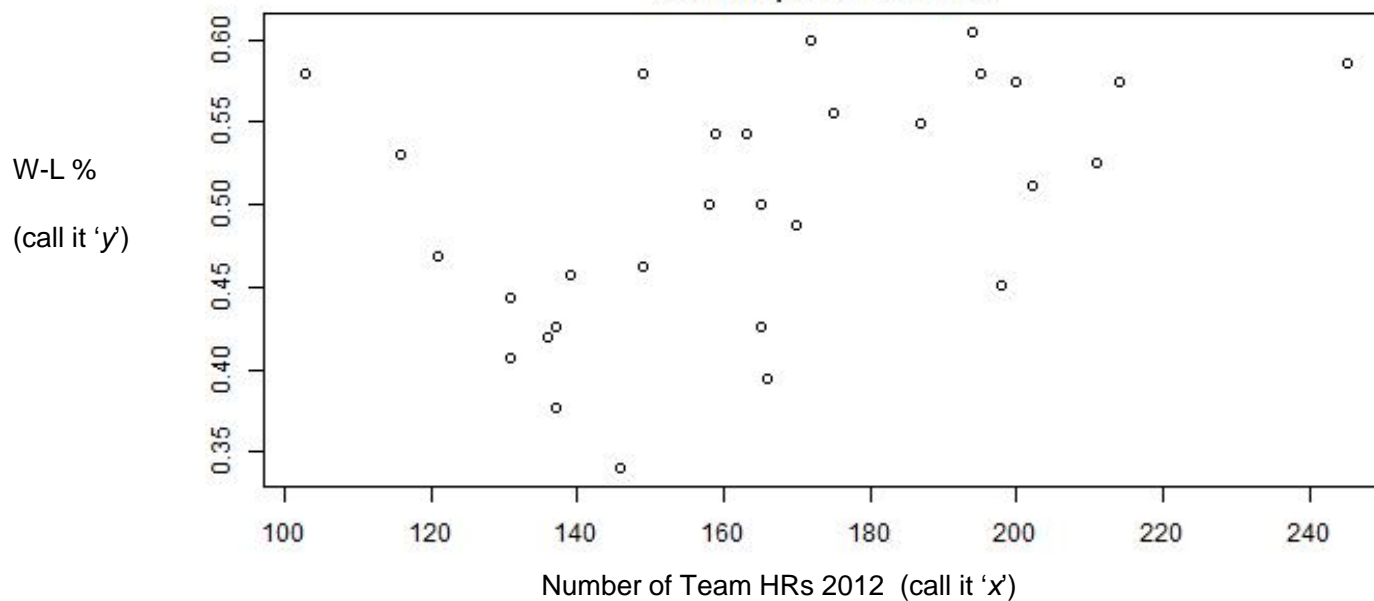


Unsupervised (no labels)

- **Bayesian (i.e. learning probability distributions)**

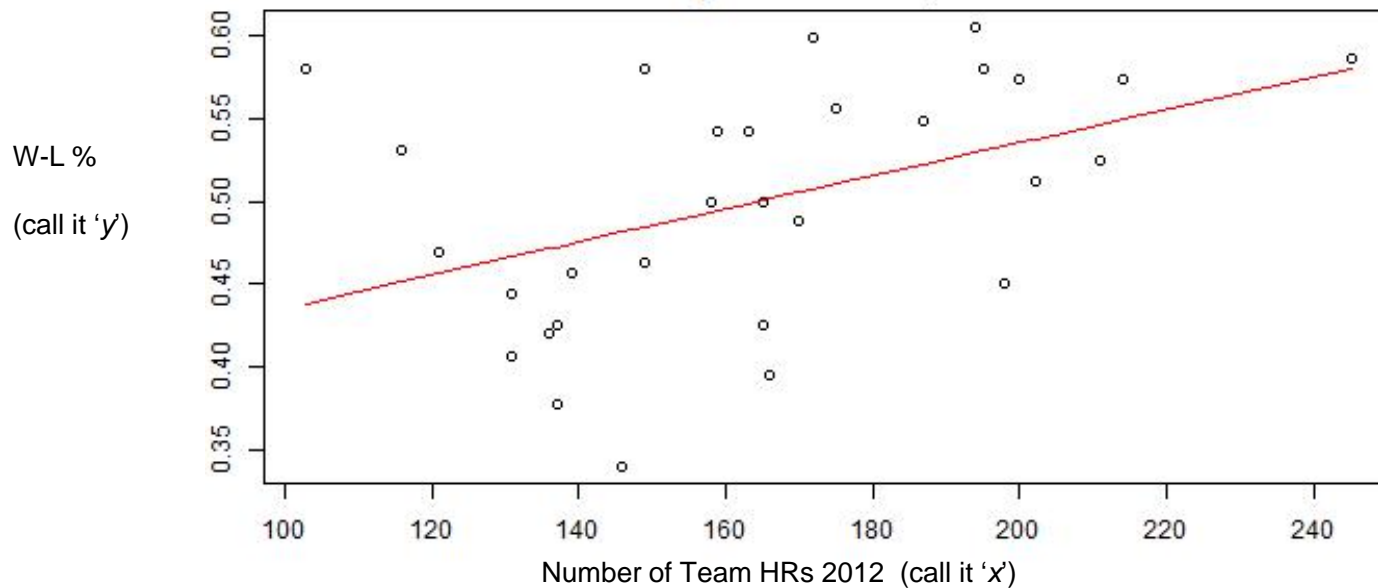
Statistical but comes up in HPC settings

A data example: Home Runs and W-L percent



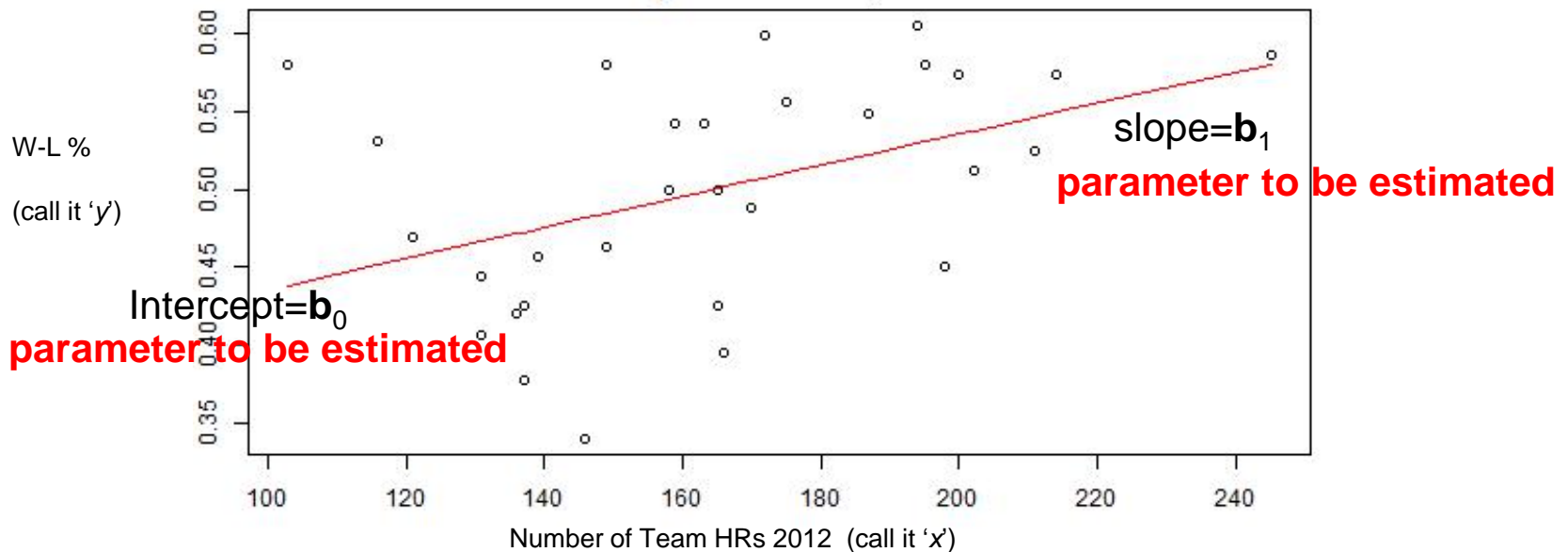
Recall Linear Regression is Fitting a Line

the Model: $y = f(x, b) = b_0 * 1 + b_1 * x$

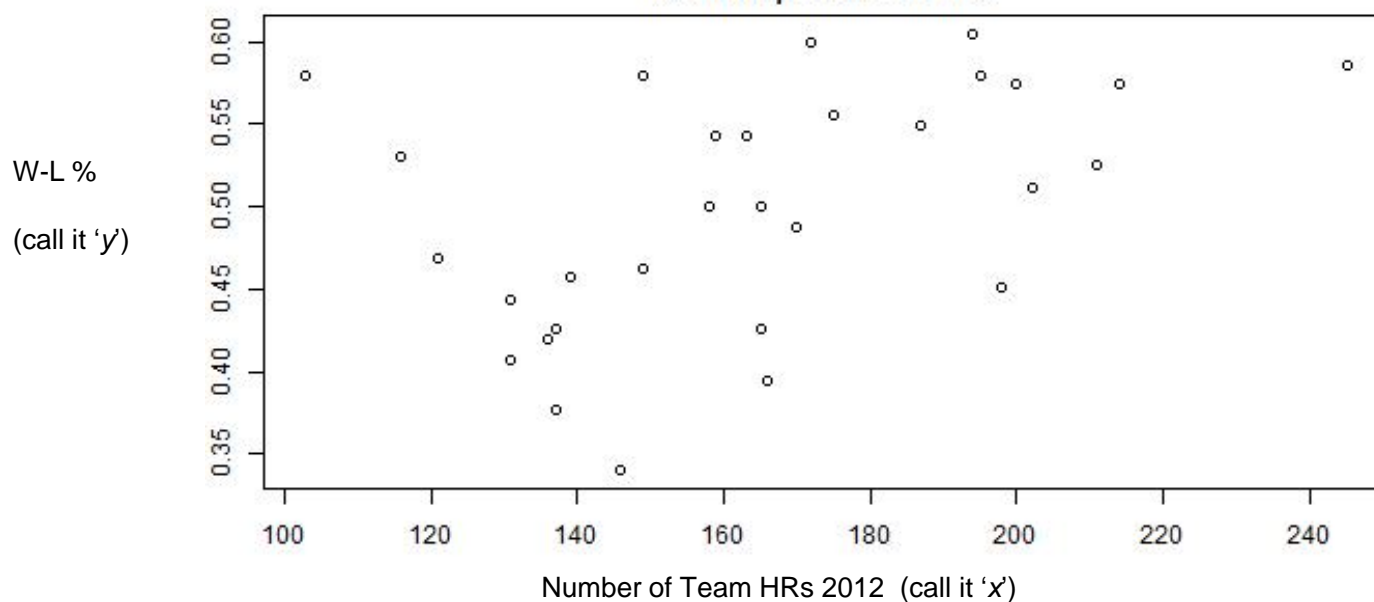


Recall Linear Regression is Fitting a Line – to minimize error

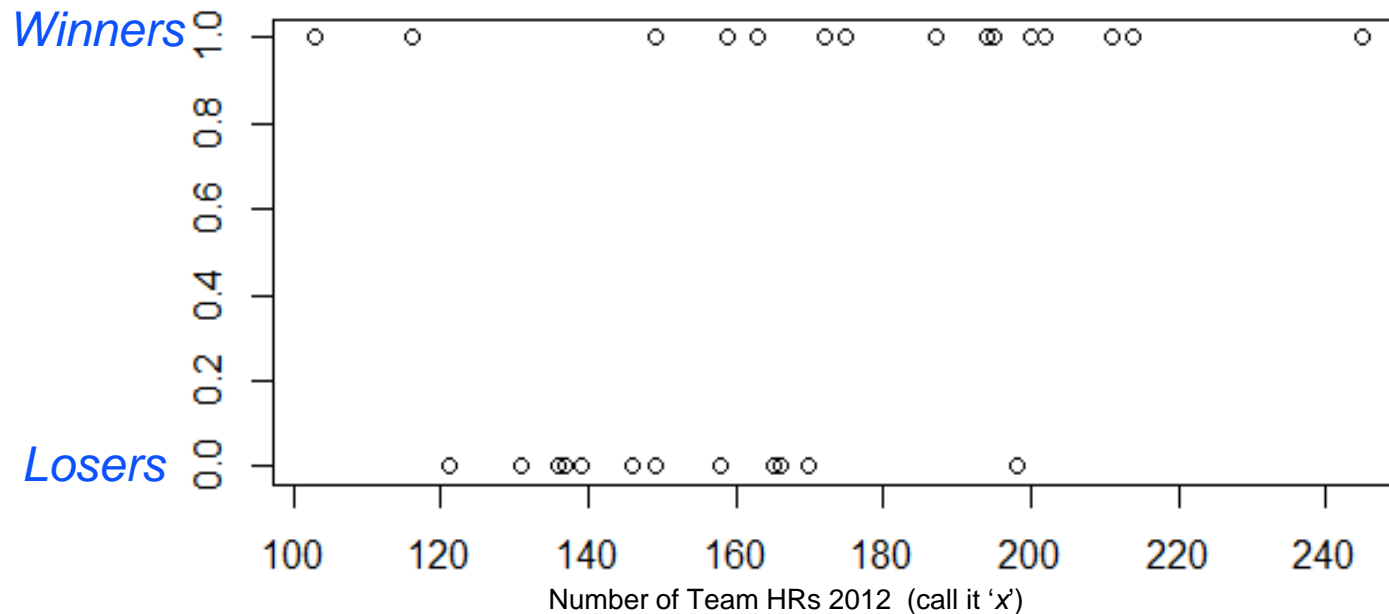
the Model: $y = f(x, b) = b_0 * 1 + b_1 * x$



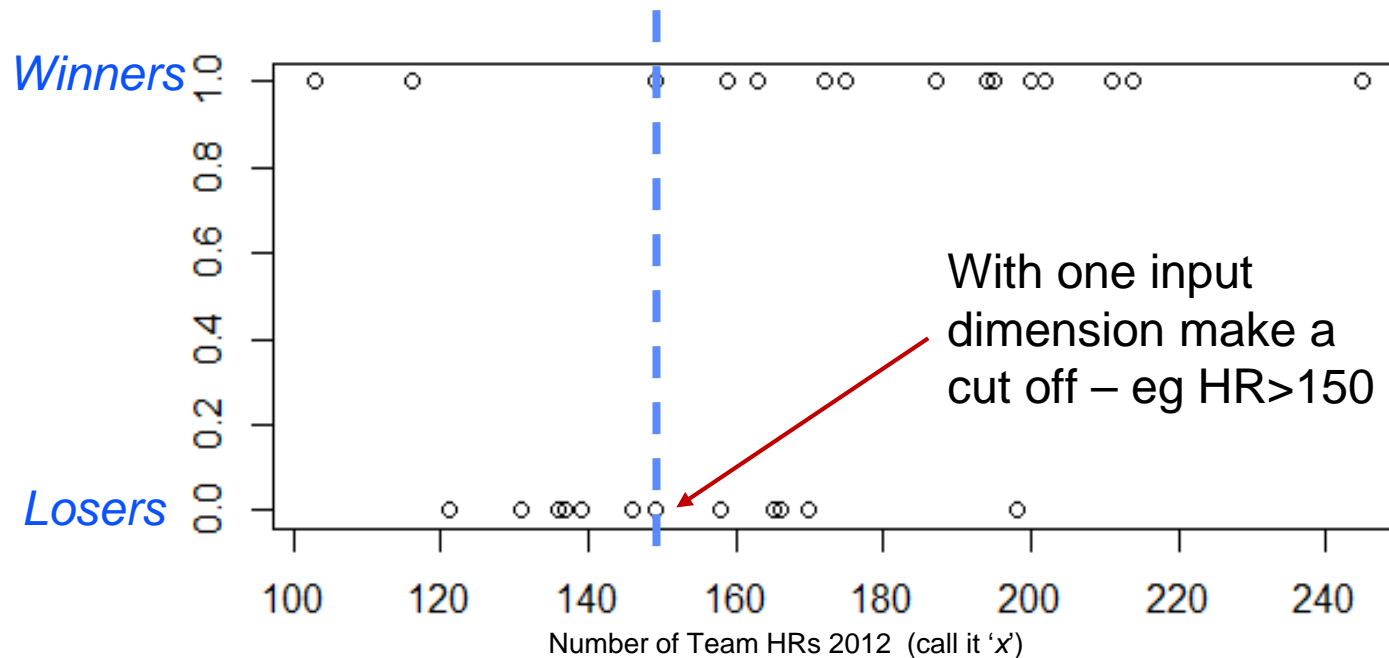
Can we just classify winners vs losers based on home runs?



Classification uses labelled outcomes



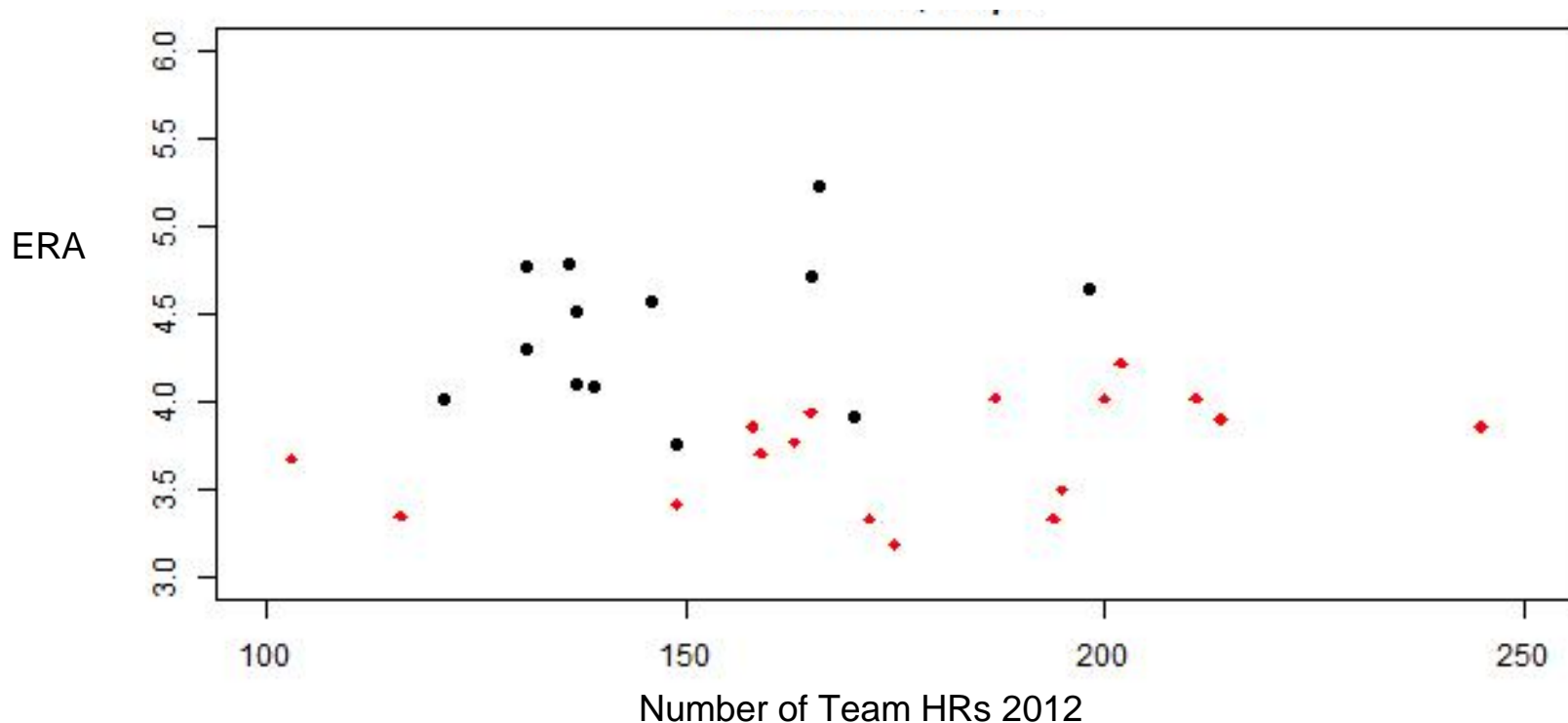
Classification uses labelled outcomes



A Linear Model for Classification

- 2 classes: +1=Black (WL% \geq .5) -1=Red (WL% $<$.5)

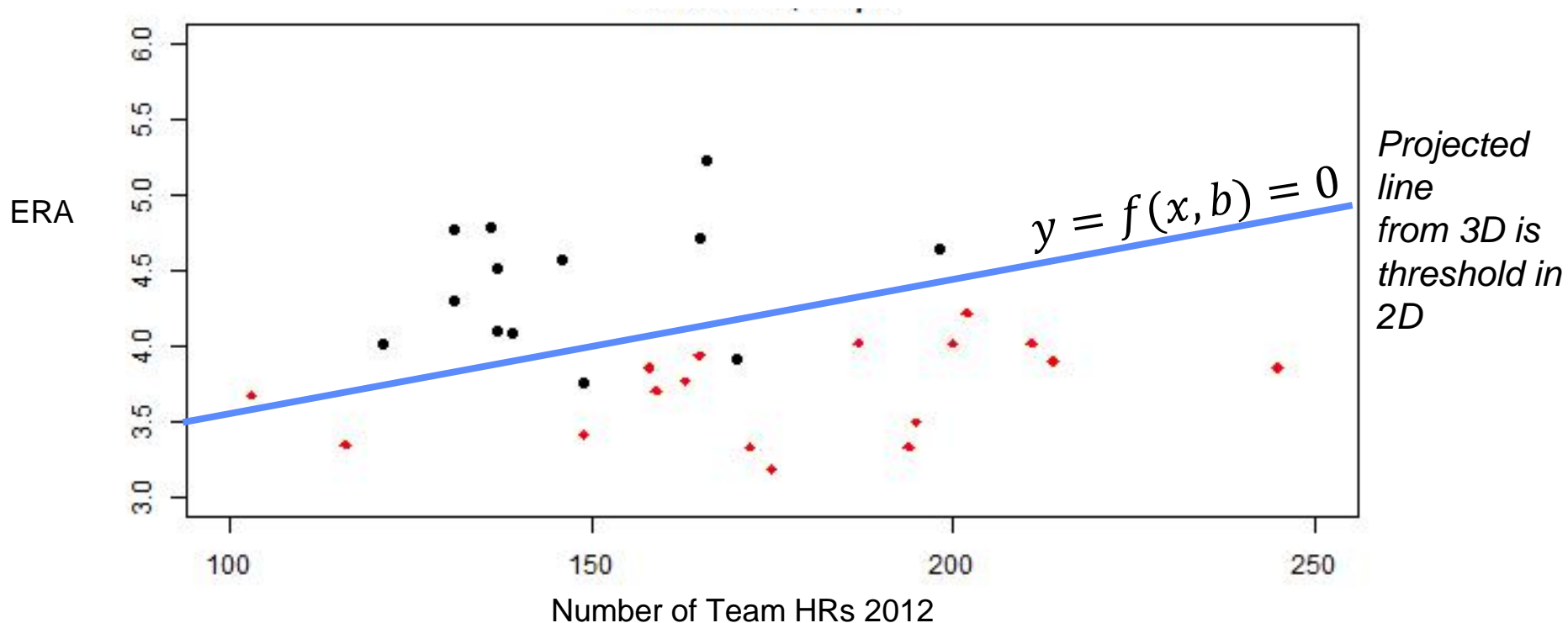
Q: Classify winning records based on HRs and ERA?



A Linear Model for Classification

- 2 classes: +1=Black (WL% \geq .5) -1=Red (WL% $<$.5)

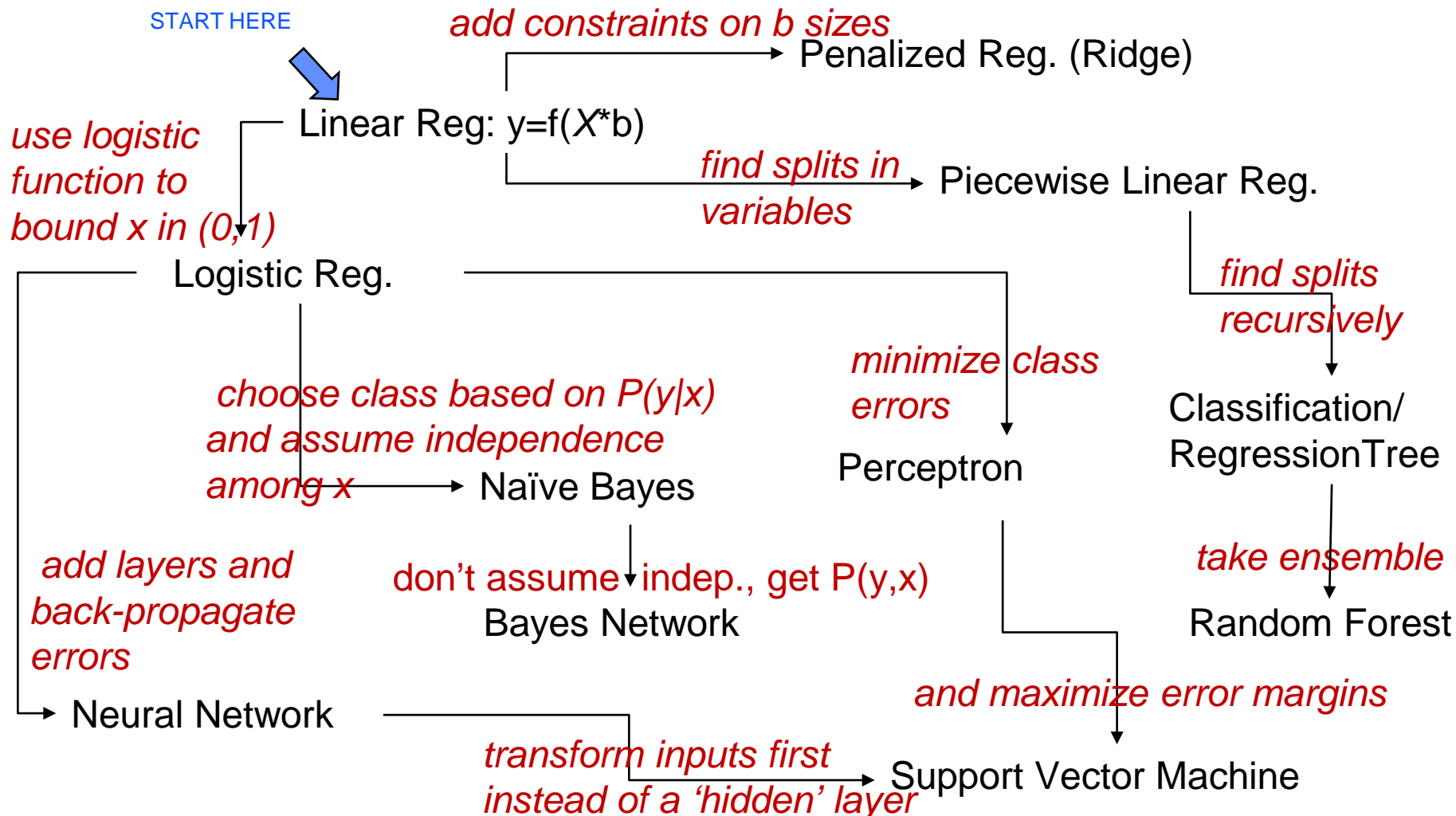
Q: Classify winning records based on HRs and ERA?



Model Choices

- **What kinds of functions to use**
 - e.g. Linear vs NonLinear
- **What to Optimize**
 - Minimize Prediction Error
 - Minimize Classification Errors
 - Maximize Probabilities
- **How to Find Parameters**
 - Search space of solutions
 - Constraints and Assumptions

Model Space Map – in a nutshell



Model Complexity vs Overfitting

more parameters =>

more complex =>

more potential to overfit

(so use training and test datasets)

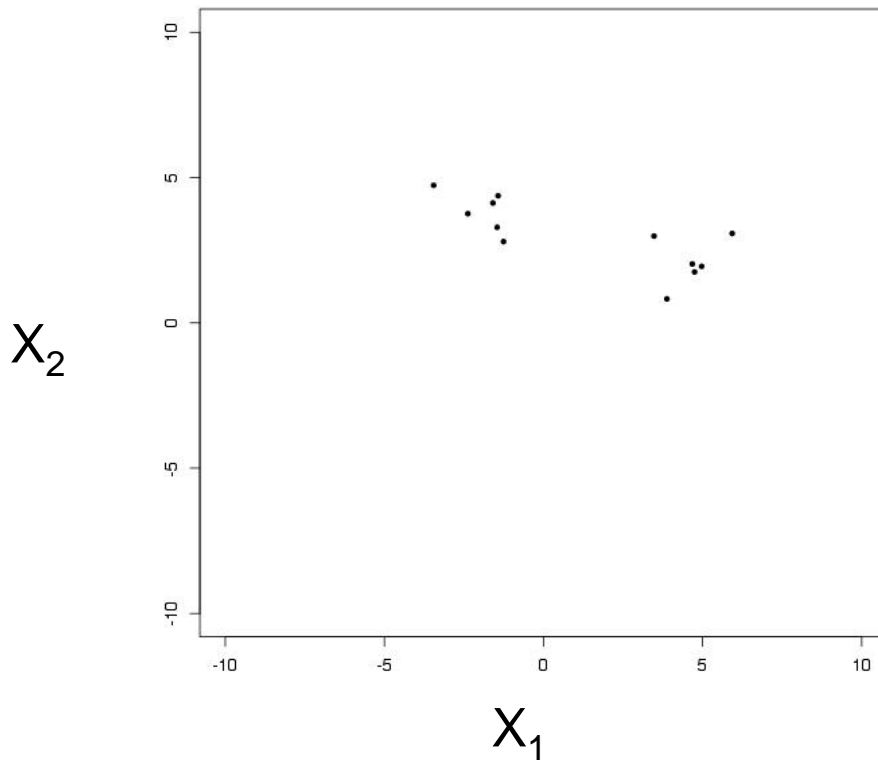
Modeling Recommendations

- **Start simple**
 - **Consider trade off as you go more complex**
 - **Find what works in your domain**
 - **Find what works for this model**
-
- **R, Python, Matlab: scripting languages with train/predict/test functions**
 - **Weka, KNIME: GUI tools**

Pause

Clustering Idea

- Given a set of data can we find a natural grouping?



Essential R commands:

```
D =rnorm(12,0,1) #generate 12
```

```
#random normal
```

```
X1 =matrix(D,6,2) #put into 6x2 matrix
```

```
X1[,1]=X1[,1]+4; #shift center
```

```
X1[,2]=X1[,2]+2;
```

```
#repeat for another set of points
```

```
#bind data points and plot
```

```
plot(rbind(X1,X2),  
      xlim=c(-10,10),ylim=c(-10,10));
```

Clustering

- **A good grouping implies some structure**

Interpret and label clusters

Characterize new points by the closest cluster

- **Kmeans is a standard algorithm**

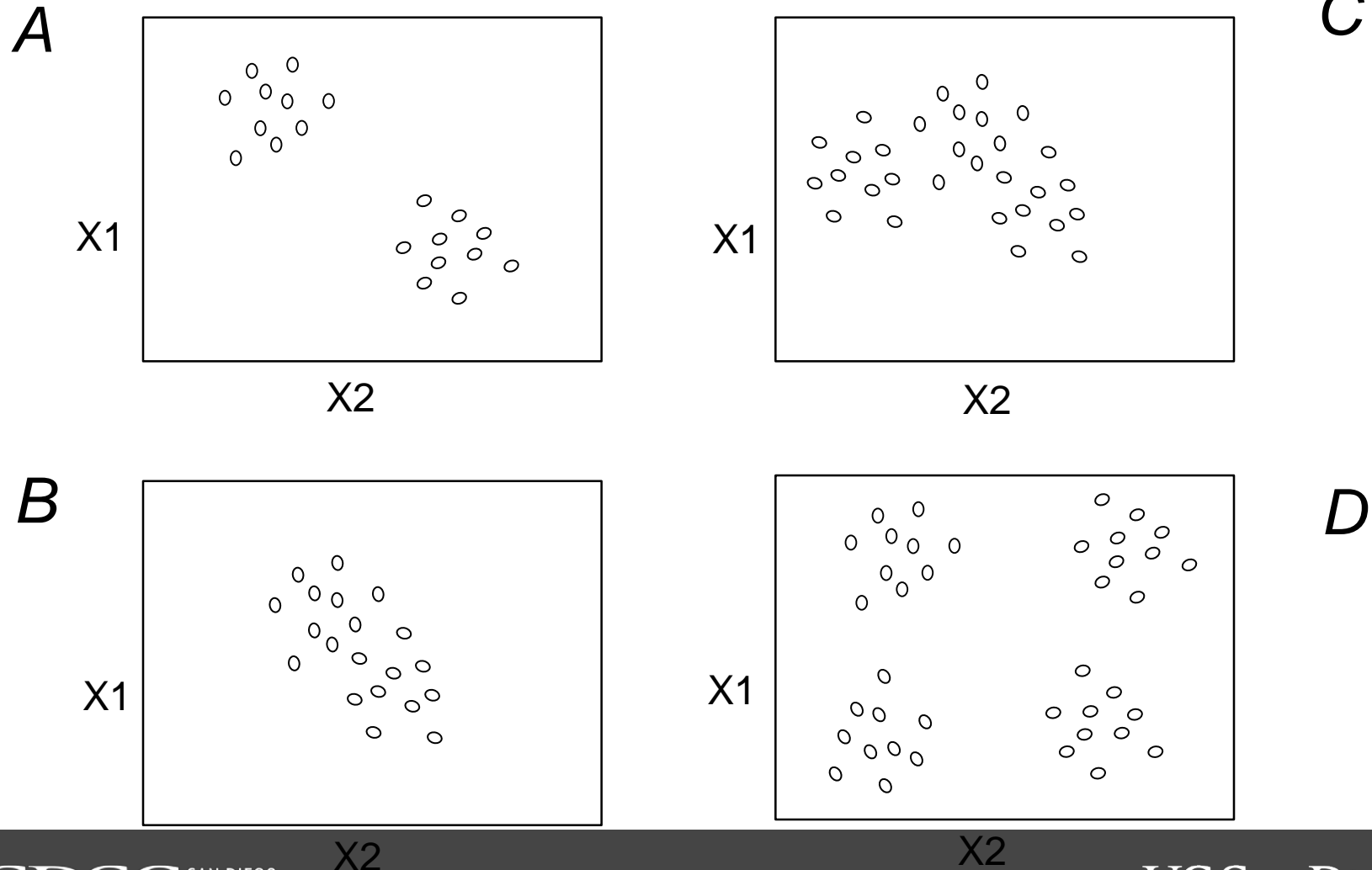
Get distances of all points to K cluster centers

Assign points to closest center

Recalculate cluster centers

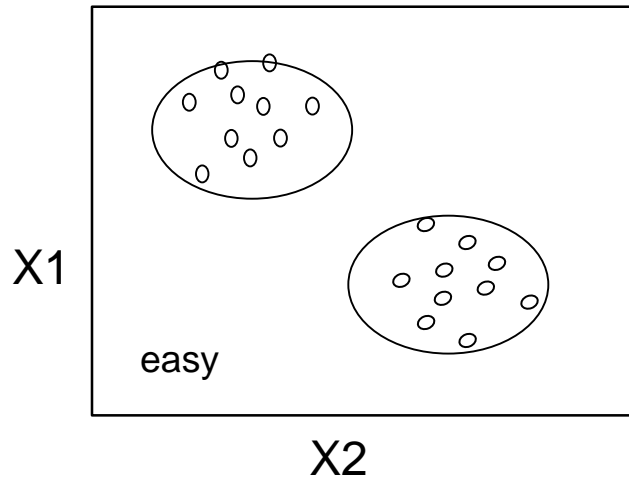
Try different K and select best inter- vs intra- class separations.

Imagine these 2 dimensional input spaces:
Which of these is easy or hard to cluster? (no class labels)

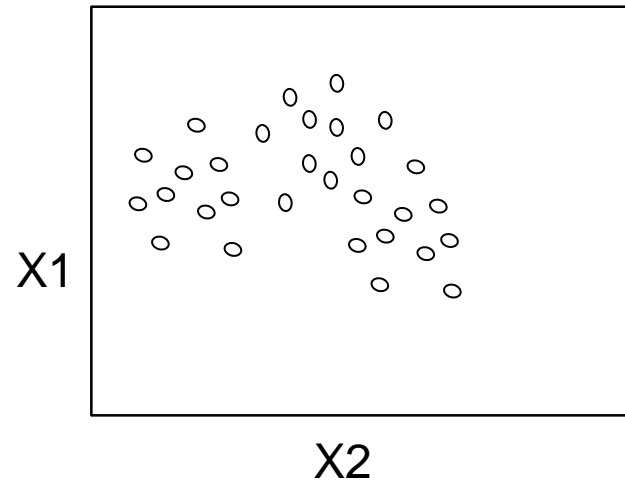


Potential clusters

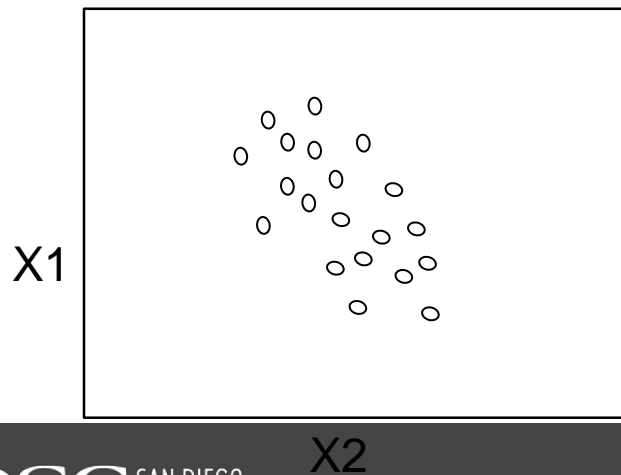
A



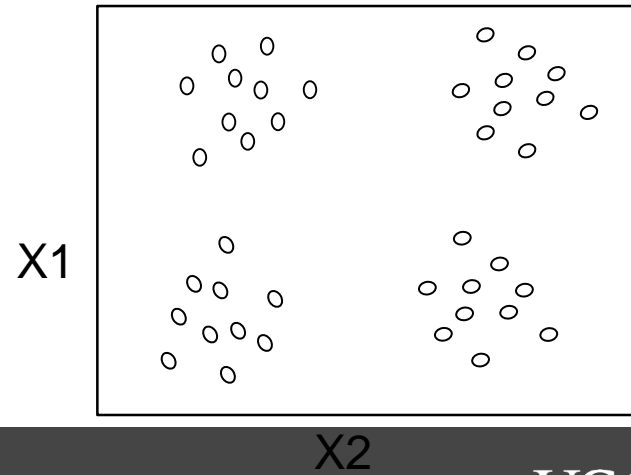
C



B

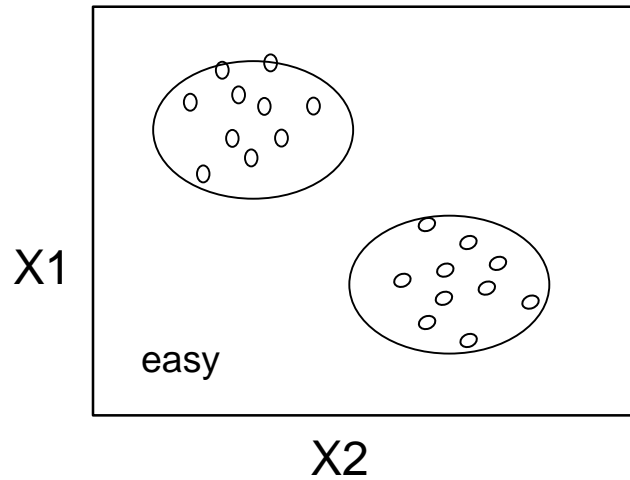


D

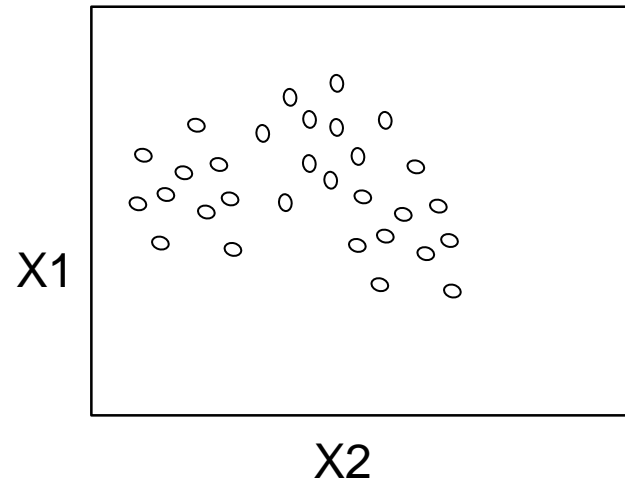


Potential clusters

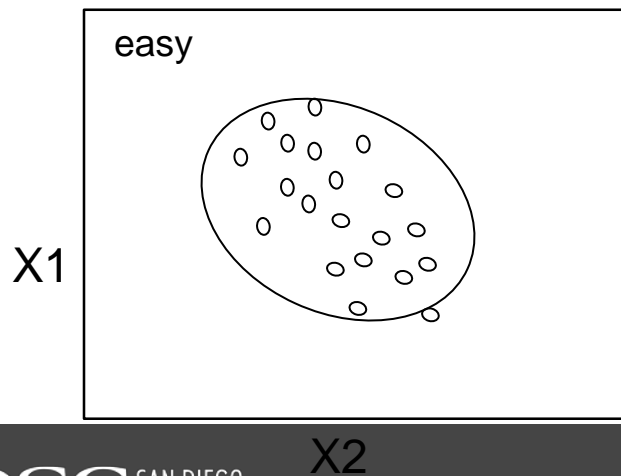
A



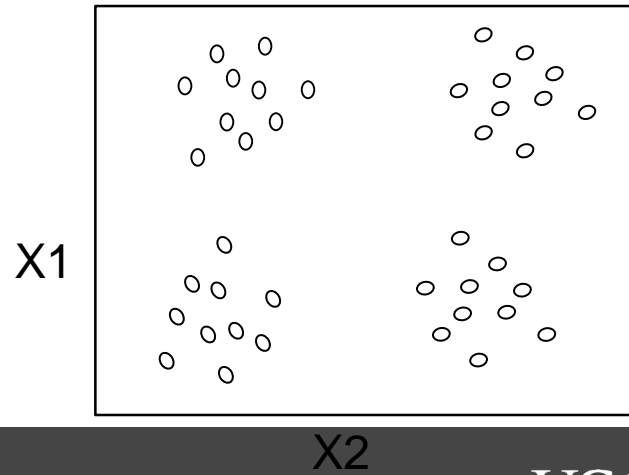
C



B

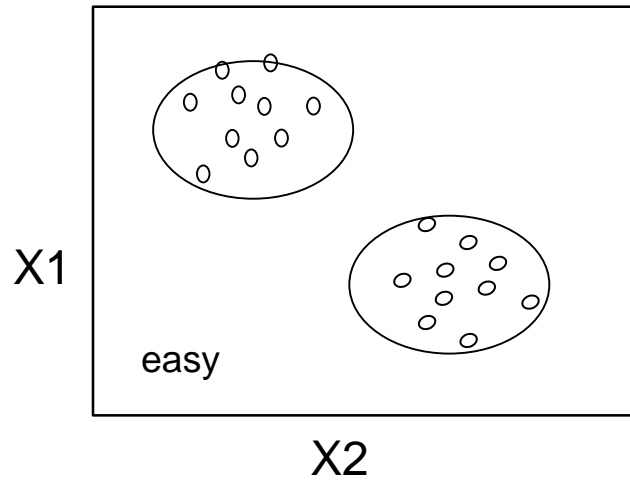


D

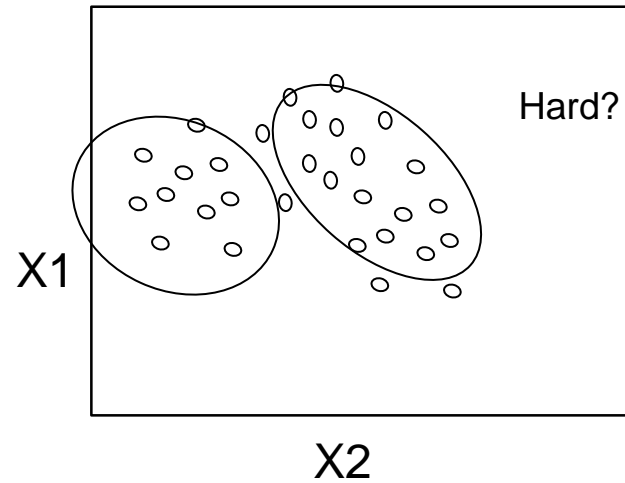


Potential clusters

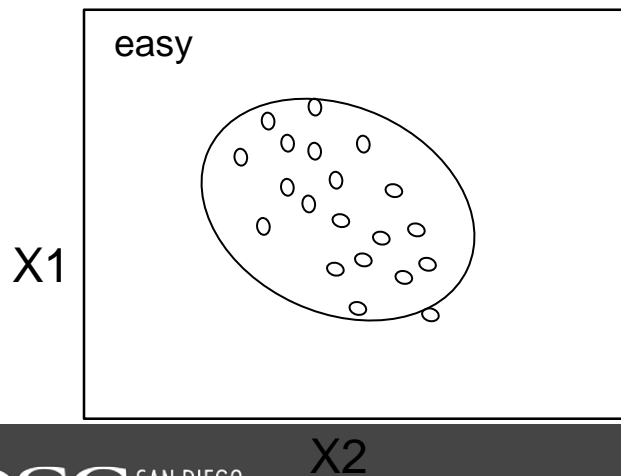
A



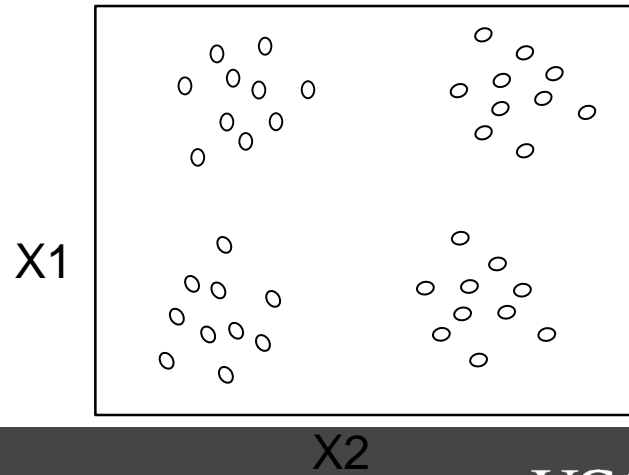
C



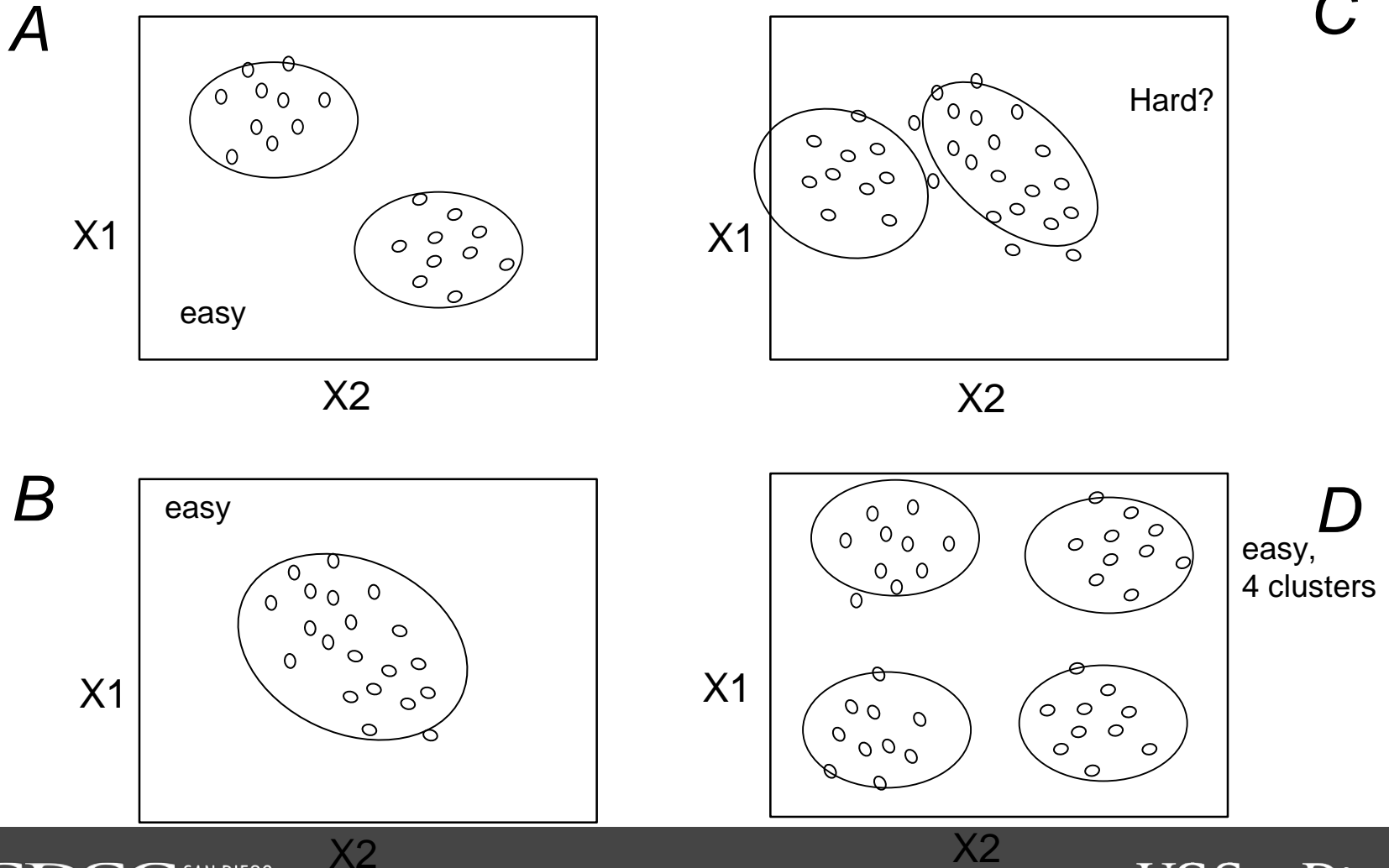
B



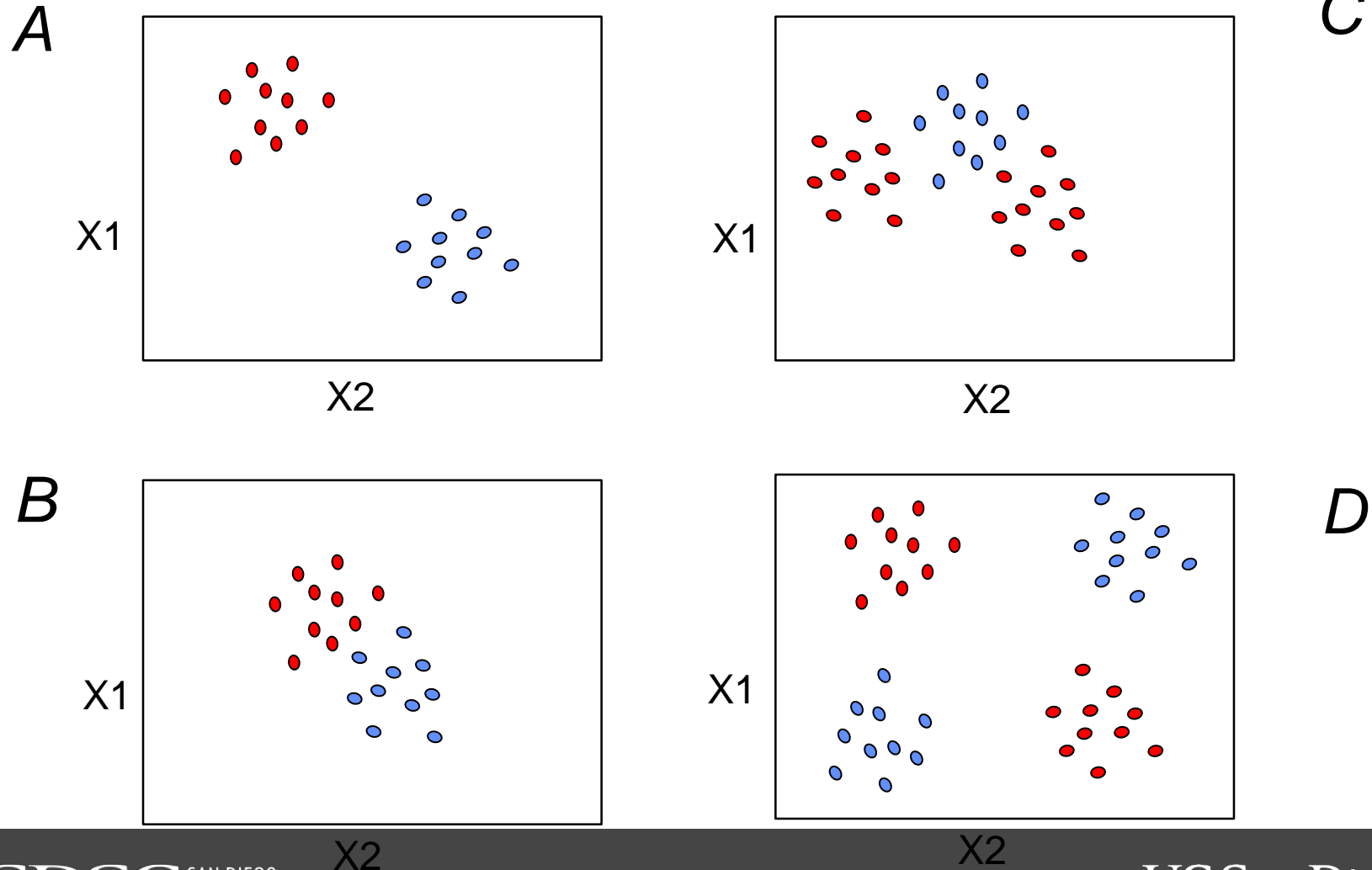
D



Potential clusters

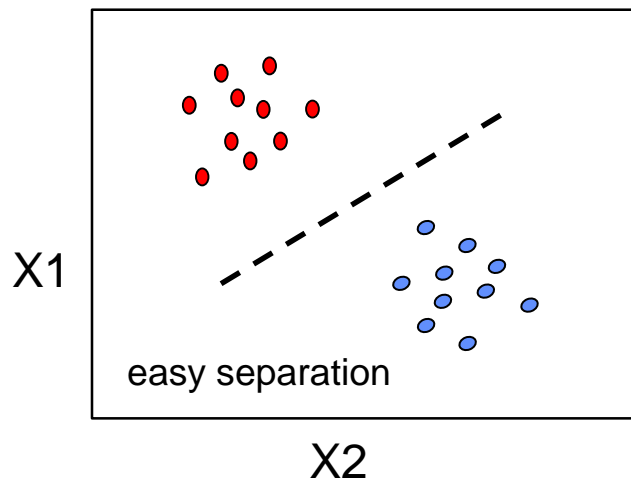


Now imaging there are two classes

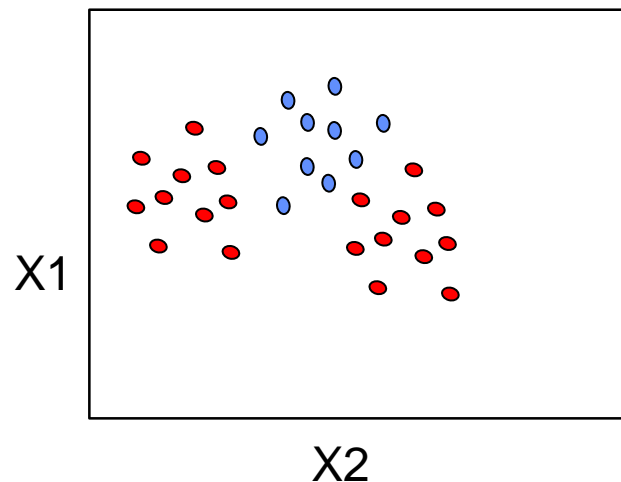


Which are easy or hard to classify?
(ie separate red or blue with lines)

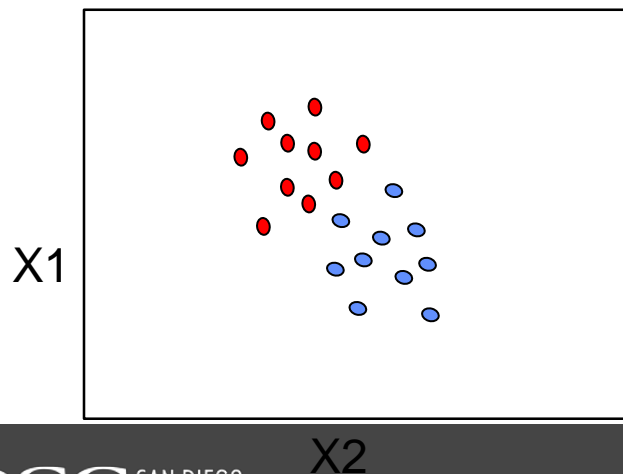
A



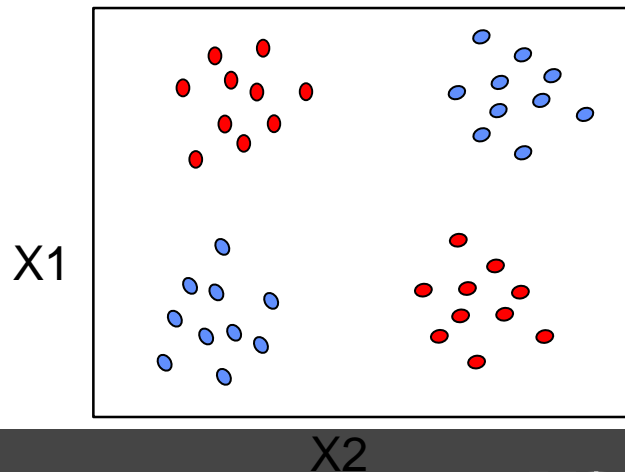
C



B

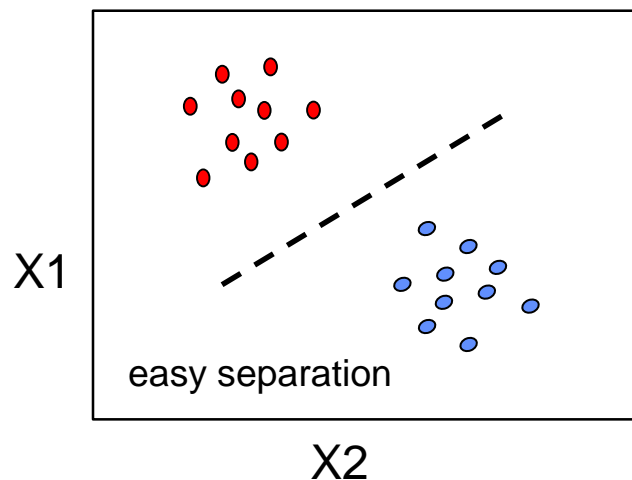


D

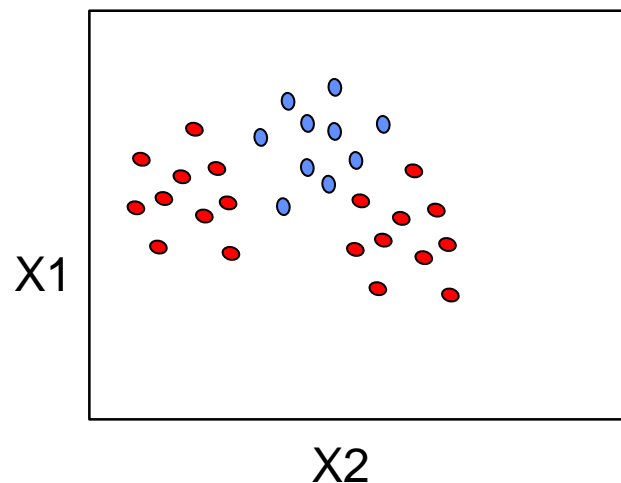


Which are easy or hard to classify?
(ie separate red or blue with lines)

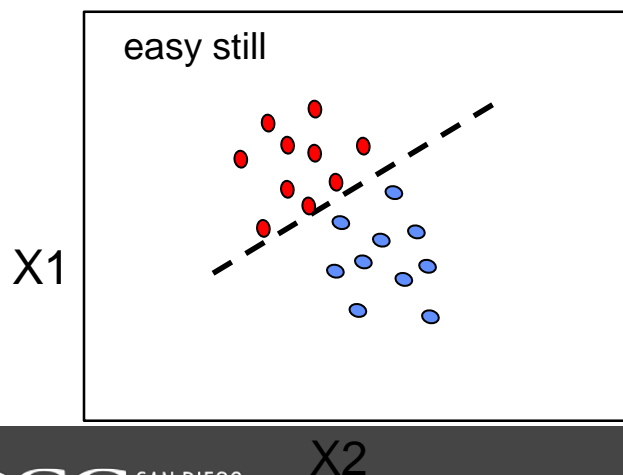
A



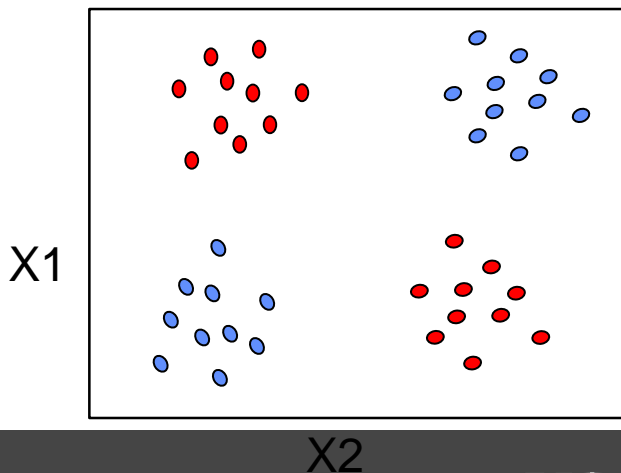
C



B

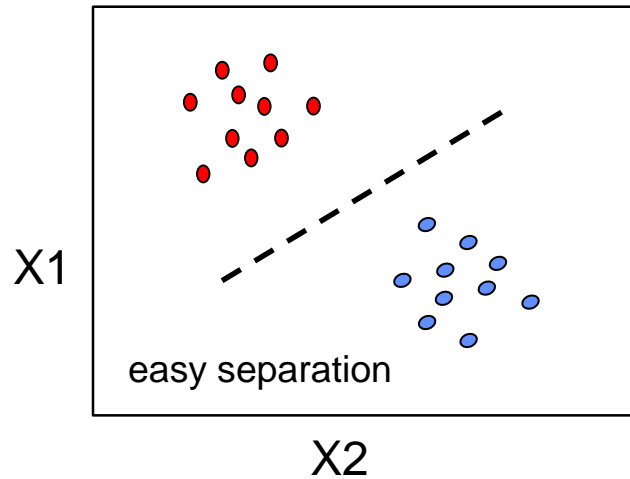


D

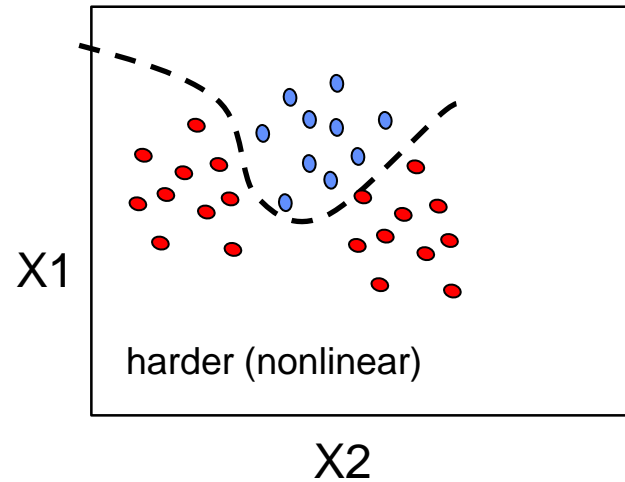


Which are easy or hard to classify?
(ie separate red or blue with lines)

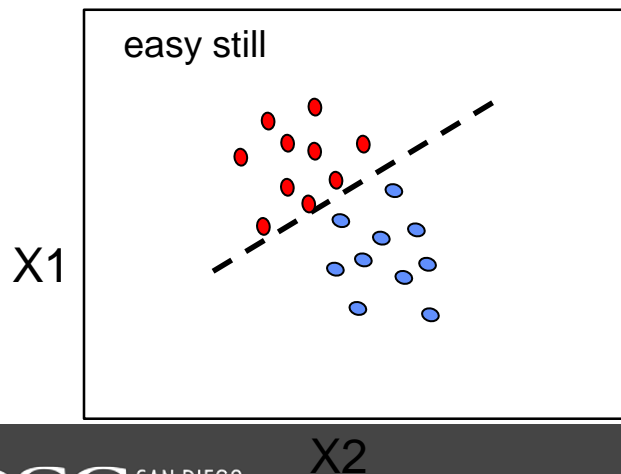
A



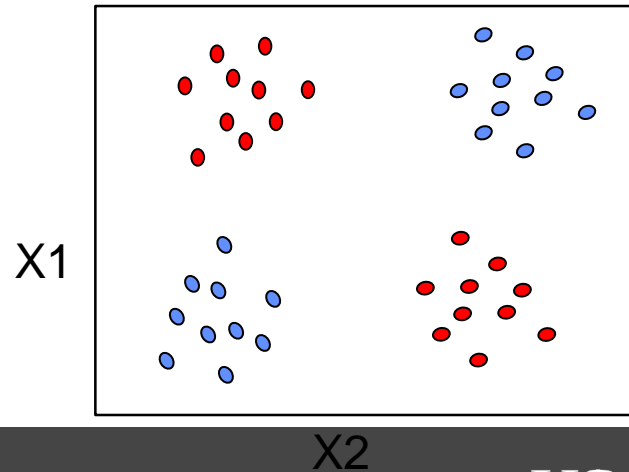
C



B

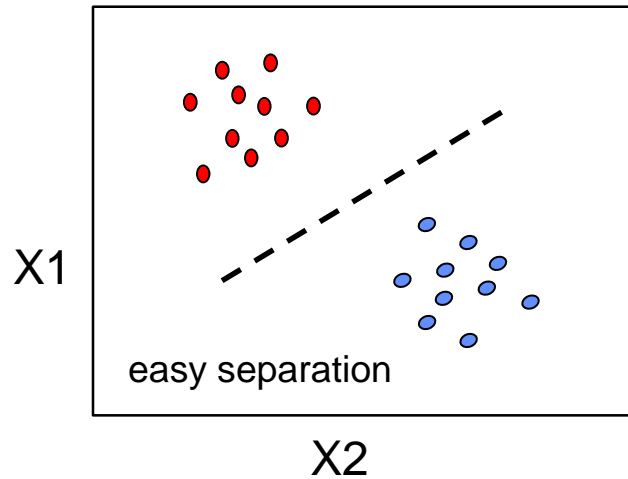


D

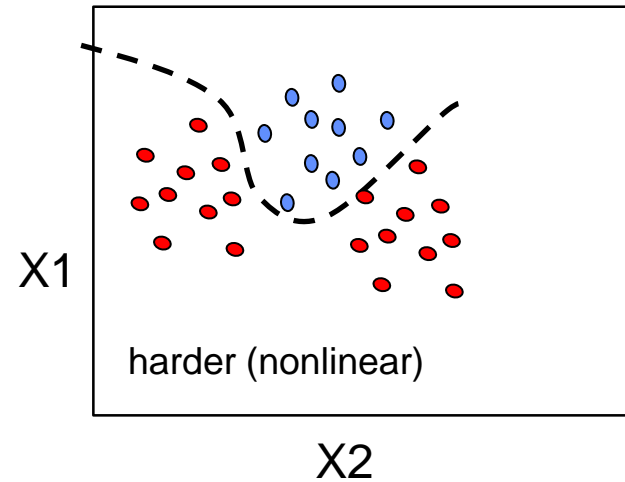


Which are easy or hard to classify?
(ie separate red or blue with lines)

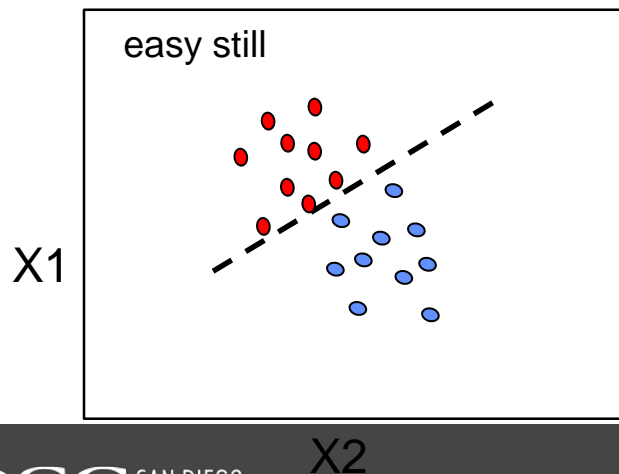
A



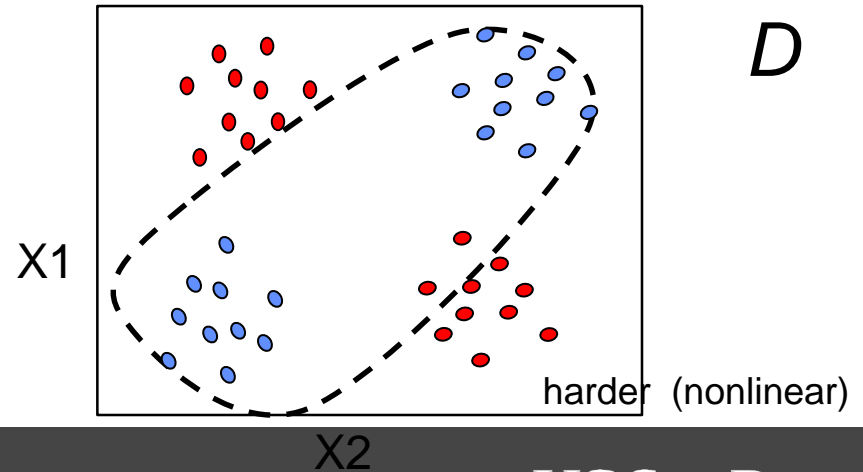
C



B

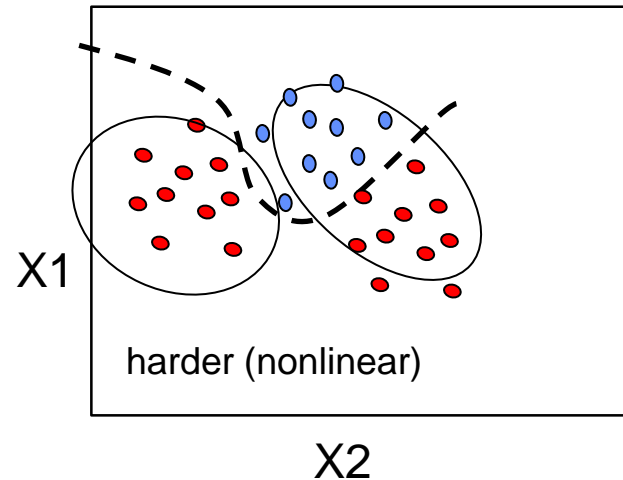
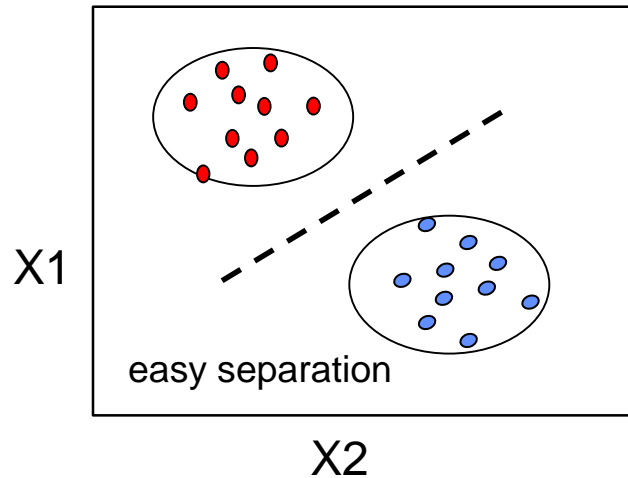


D



Which are easy or hard to classify?
(ie separate red or blue with lines)

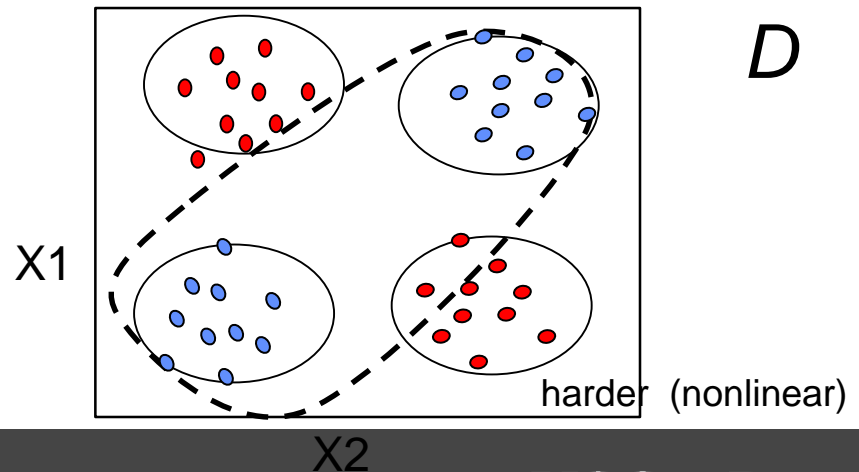
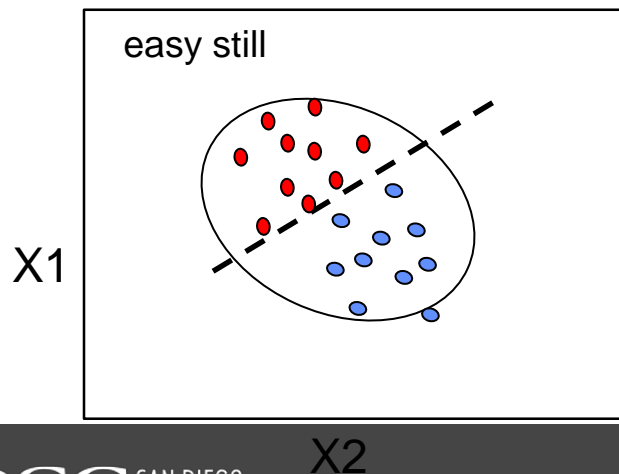
A



C

In summary:
**No easy
relationship
between
clusters
and
classification**

B



D

Pause

Matrix Factorization:

Given a numeric matrix, can we reduce the number of columns?

Matrix Factorization:

Given a numeric matrix, can we reduce the number of columns?

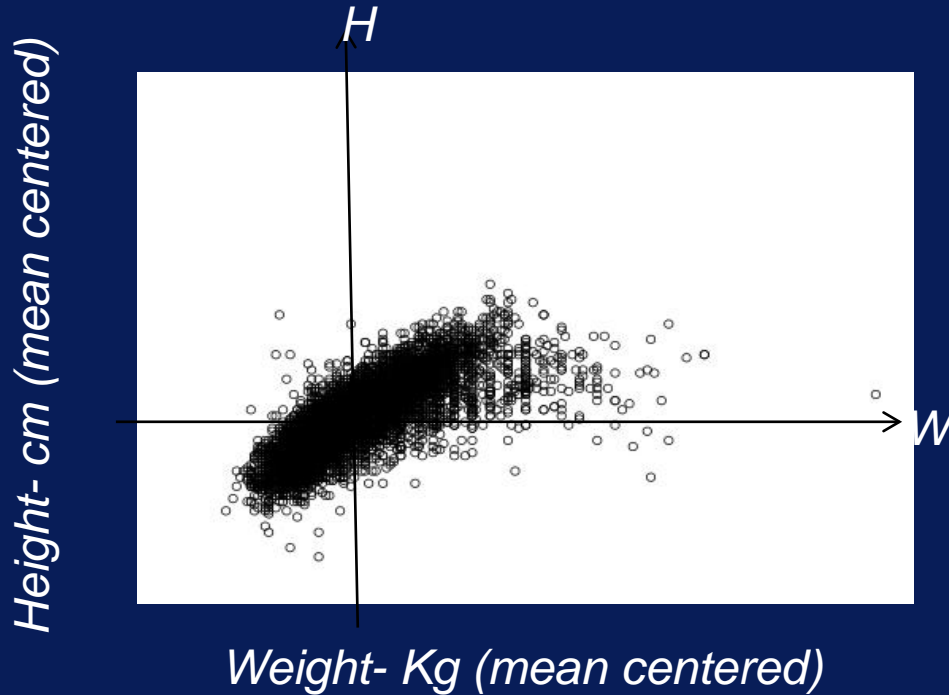
- Yes, if features are constant or redundant

Matrix Factorization:

Given a numeric matrix, can we reduce the number of columns?

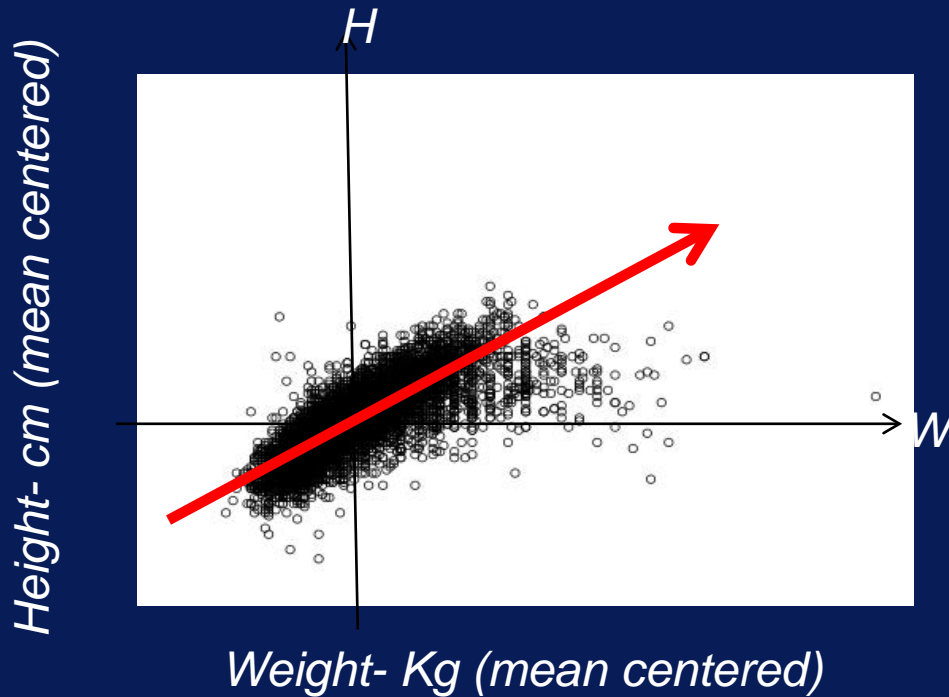
- Yes, if features are constant or redundant
- Yes, if features only contribute noise
(conversely, want features that contribute to variations of the data)

Example: Athletes' Height by Weight

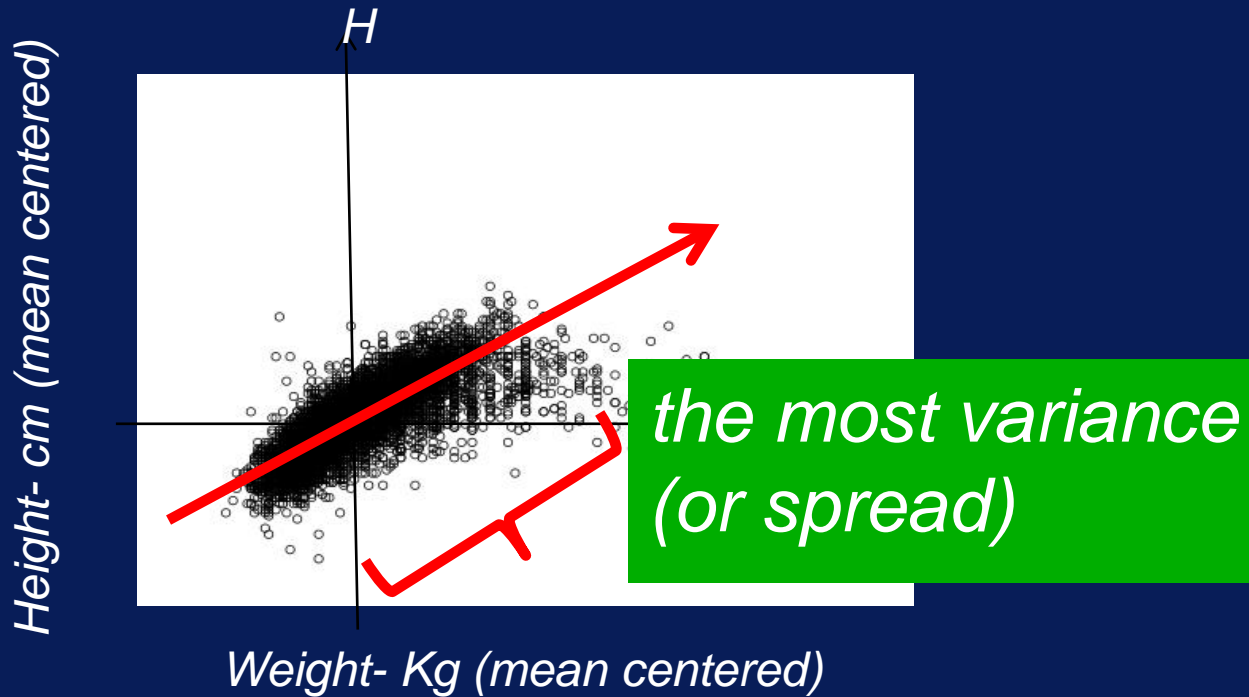


Find a line that aligns with the data.

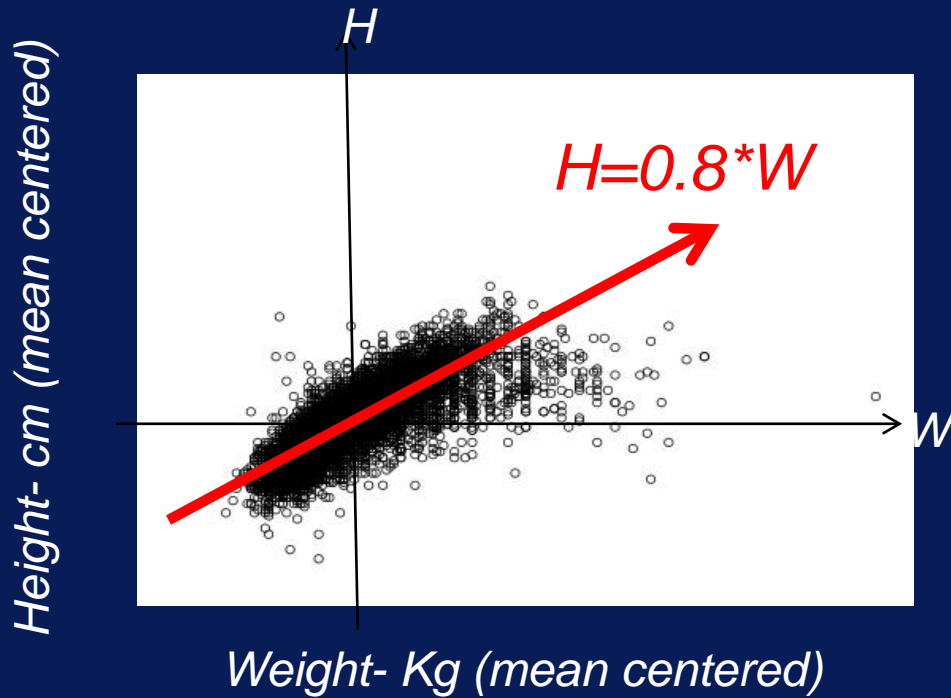
Example: Athletes' Height by Weight



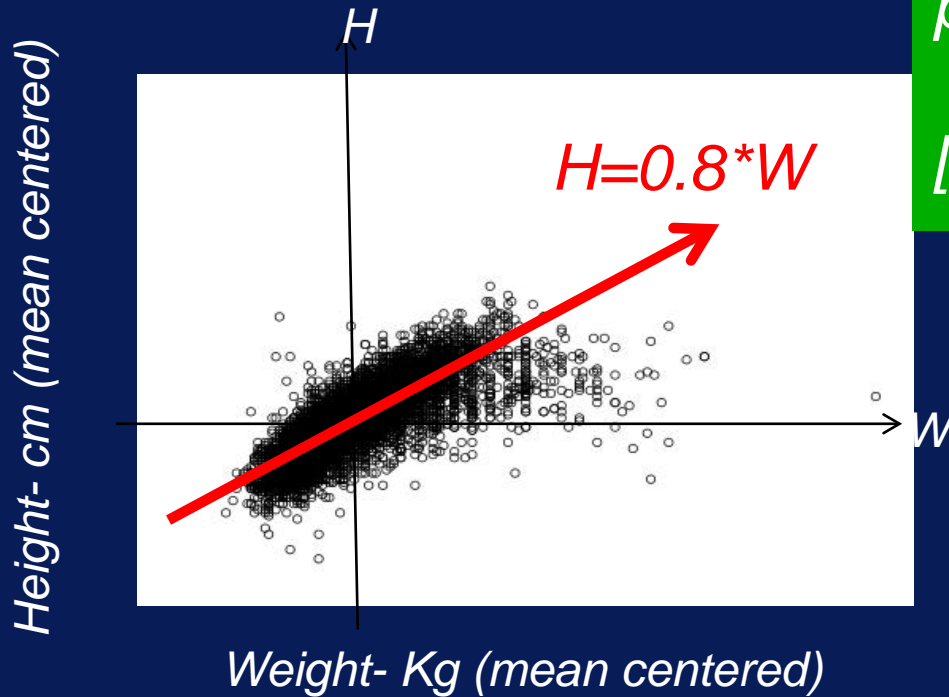
Find a line that aligns with the data.



Find a line that aligns with the data.



Find a line that aligns with the data.



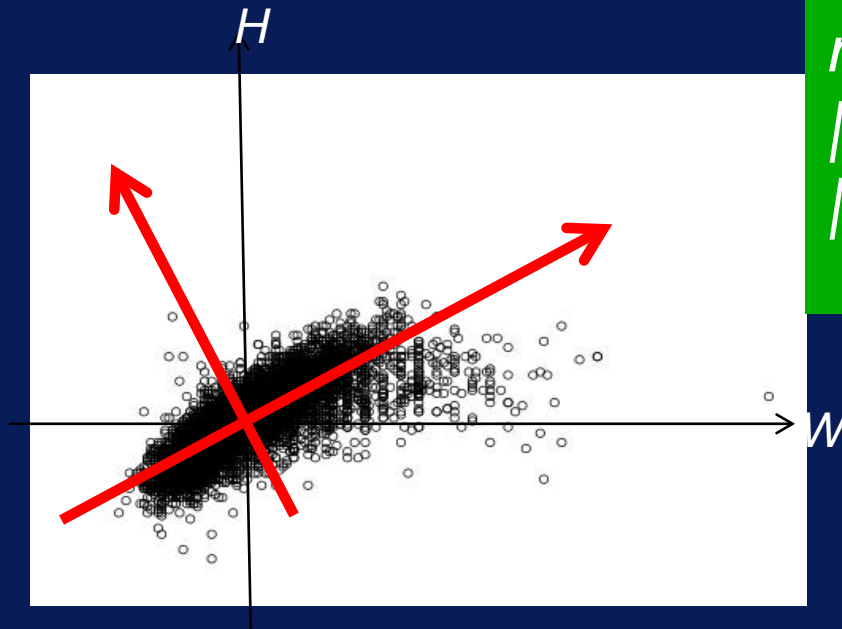
Note that $(0,0)$ and $(1,0.8)$ are points on the line

- OR -

$[1 \ 0.8]$ describes a vector

Find a line that aligns with the data.

Height- cm (mean centered)



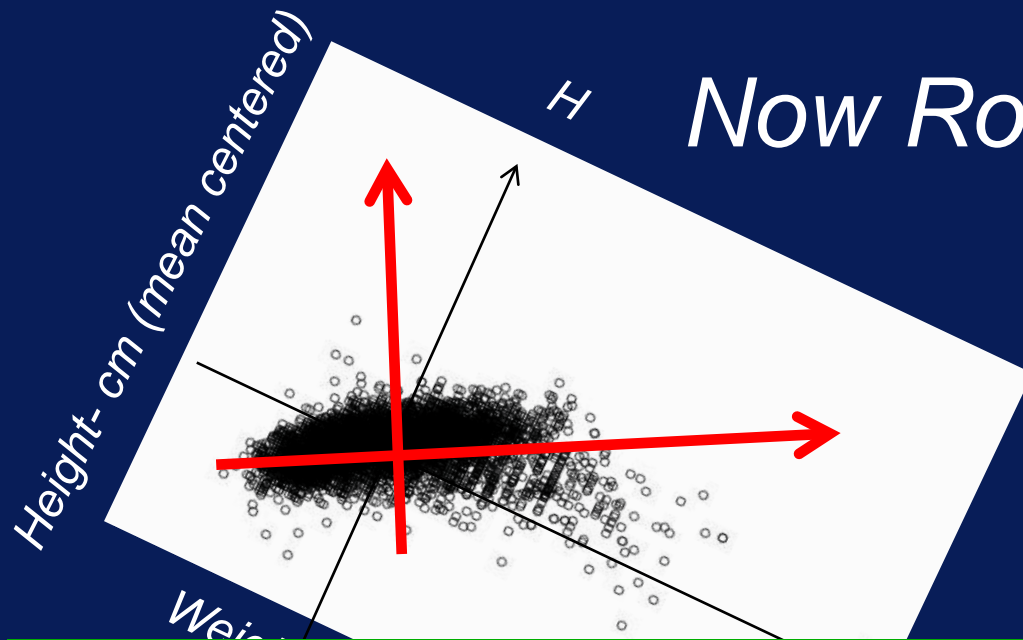
Weight- Kg (mean centered)

$[-0.8 \ 1]$ describes 2nd vector
- AND -

Both vectors are columns in a
matrix:

$$\begin{bmatrix} 1 & -0.8 \\ 0.8 & 1 \end{bmatrix}$$

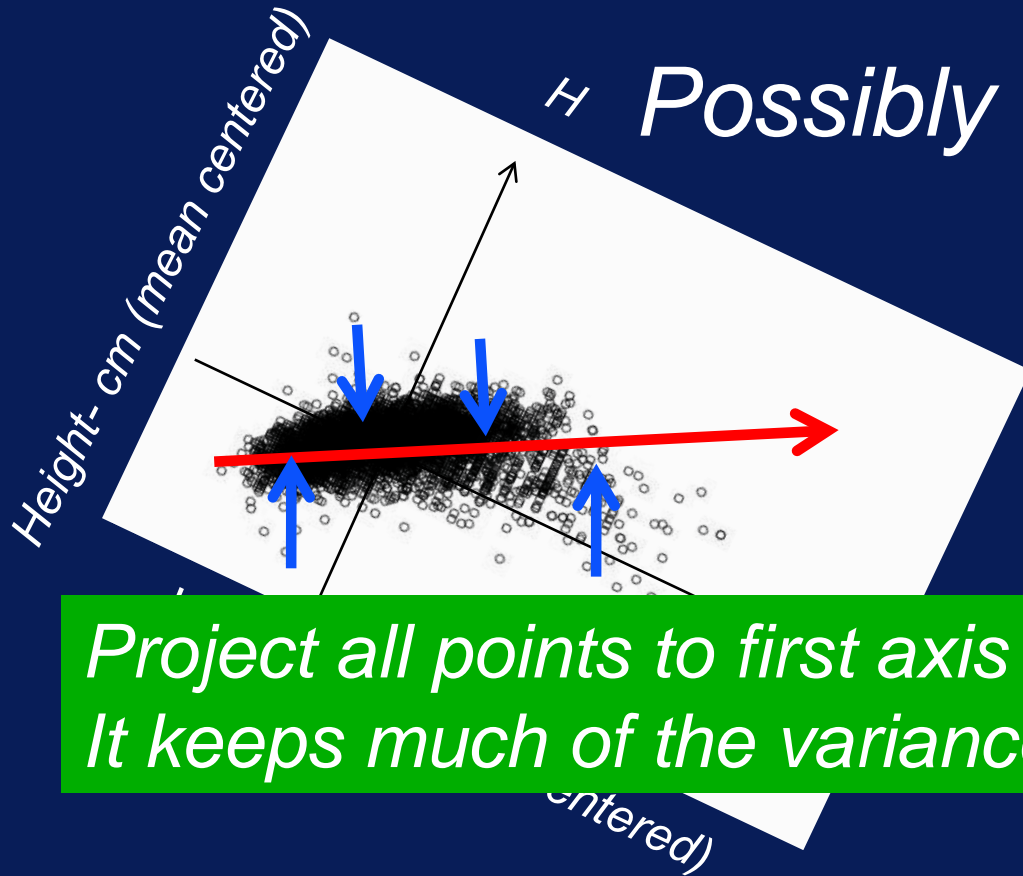
The next direction of most variance.



Now Rotate Axis

*New axis (AKA features)
defined as combinations of
old features*

H Possibly reduce dimensions



*Project all points to first axis
It keeps much of the variance*

- Best Known Factorization Algorithms:
 - SVD (singular value decomposition)
 - PCA (principle component analysis)

SVD more generally works on non square matrices

- **pause**

R on Comet

1. Get a compute node:

```
[Unix]$ : srun --partition=debug --pty --nodes=1 --  
ntasks-per-node=24 -t 00:30:00 --wait=0 --export=ALL -A your-  
account /bin/bash
```

2. Start R

```
[Unix]$ module load R
```

```
[Unix]$ R (this gets an interactive R session)
```

quit() to exit R

exit to exit the compute node

Running Jupyter notebook on Comet

1. Login to comet
2. Access compute node: `srun --partition=debug --pty --nodes=1 --ntasks-per-node=24 -t 00:30:00 --wait=0 --export=ALL -A your-account /bin/bash`
3. Start singularity shell
 1. `module load singularity`
 2. `IMAGE=/oasis/scratch/comet/zonca/temp_project/datascience-notebook-e1677043235c_fixjulia_keras_tf.img`
 3. `singularity exec $IMAGE jupyter notebook --ip=*`
4. on local machine, in browser url edit box, enter the http string shown, but replace localhost with comet-XX-XX.sdsc.edu
5. Open R-introHPC.ipynb or LabMNIST_Final.ipynb
6. After logging out in browser shutdown notebook on Comet with Ctrl-C

The image is a composite of three screenshots illustrating the process of running a Jupyter notebook on Comet.

Terminal Screenshot (Left): Shows a Singularity shell session. The user runs `srun` to access a compute node, then `module load singularity`, and finally `singularity exec $IMAGE jupyter notebook --ip=*`. The output shows the Jupyter notebook server starting, with a warning about the deprecated `ipython notebook` command. The server is running at `http://localhost:8888/?token=579bbc5380ffdc8cd8f34d9c37aa023187a10b5da12ee39`. A message at the bottom instructs the user to copy/paste this URL into their browser.

Browser Screenshot (Right): Shows the Jupyter web interface in a browser. The address bar contains the URL `comet-04-13.sdsc.edu:8888/?token=579bbc5380ffdc8cd8f34d9c37aa023187a10b5da12ee39`. The interface shows a file list with the following items:

Name	Last Modified
LabMNIST_v2.ipynb	8 hours ago
\$	a day ago
an0aet.txt	9 hours ago
an0W.txt	9 hours ago
an0Wmyen.txt	9 hours ago
an0Wsc.txt	9 hours ago
an0Wsc10x10.txt	9 hours ago
OctMNIST_Try	a day ago

April 4, 2018

1 In this introduction we will explore some useful R functions for data preparation. We will look very quickly at clustering and classification

```
In [1]: #1. First read the data from a CSV file into an R dataframe
        W_df_orig = read.table('weather_orig.csv',
                               header=TRUE,
                               sep=";",
                               stringsAsFactors = TRUE)

        dim(W_df_orig)
```

```
1. 366 2. 24
```

2 Quick way to view some rows:

```
In [2]: head(W_df_orig)
```

Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	Wind
2007-11-01	Canberra	8.0	24.3	0.0	3.4	6.3	NW	30
2007-11-02	Canberra	14.0	26.9	3.6	4.4	9.7	ENE	39
2007-11-03	Canberra	13.7	23.4	3.6	5.8	3.3	NW	85
2007-11-04	Canberra	13.3	15.5	39.8	7.2	9.1	NW	54
2007-11-05	Canberra	7.6	16.1	2.8	5.6	10.6	SSE	50
2007-11-06	Canberra	6.2	16.9	0.0	5.8	8.2	SE	44

```
In [3]: tail(W_df)
```

```
Error in tail(W_df): object 'W_df' not found
Traceback:
```

```
1. tail(W_df)
```

```
In [4]: str(W_df)      #Quick view of the basic 'structure' of the data frame
```



```
head(W_long)
#optional: write.csv(W_cast,file='Weather_castwide.csv')
```

Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindDir9am	Wind
2007-11-01	Canberra	8.0	24.3	0.0	3.4	6.3	SW	NW
2007-11-02	Canberra	14.0	26.9	3.6	4.4	9.7	E	W
2007-11-03	Canberra	13.7	23.4	3.6	5.8	3.3	N	NNE
2007-11-04	Canberra	13.3	15.5	39.8	7.2	9.1	WNW	W
2007-11-05	Canberra	7.6	16.1	2.8	5.6	10.6	SSE	ESE
2007-11-06	Canberra	6.2	16.9	0.0	5.8	8.2	SE	E

5 Get factors using SVD

```
In [8]: #1 Get numeric columns only
cols_numeric = sapply(W_df,is.numeric) #get column classes as a list
W_dfnum      = W_df[,which(cols_numeric)]
dim(W_dfnum)
```

```
1.328 2.16
```

```
In [9]: #2 turn it into a matrix
W_matrix = as.matrix(W_dfnum)
```

```
In [10]: #3 mean center data
W_mncntr=scale(W_dfnum,center=TRUE,scale=FALSE)
```

```
In [11]: #4 run SVD command
Wsvd=svd(W_mncntr)
str(Wsvd)
```

```
List of 3
```

```
$ d: num [1:16] 367 307 215 164 109 ...
$ u: num [1:328, 1:16] -0.03129 -0.01506 0.03569 0.00638 0.00357 ...
$ v: num [1:16, 1:16] -0.0775 -0.2114 0.0446 -0.0778 -0.128 ...
```

6 Get some kmean cluster and plot onto first two SVD factors

```
In [12]: #get Kmeans for 4 clusters, with 10 iterations and 1 starting points
k4      = kmeans(W_mncntr,4,10,1)

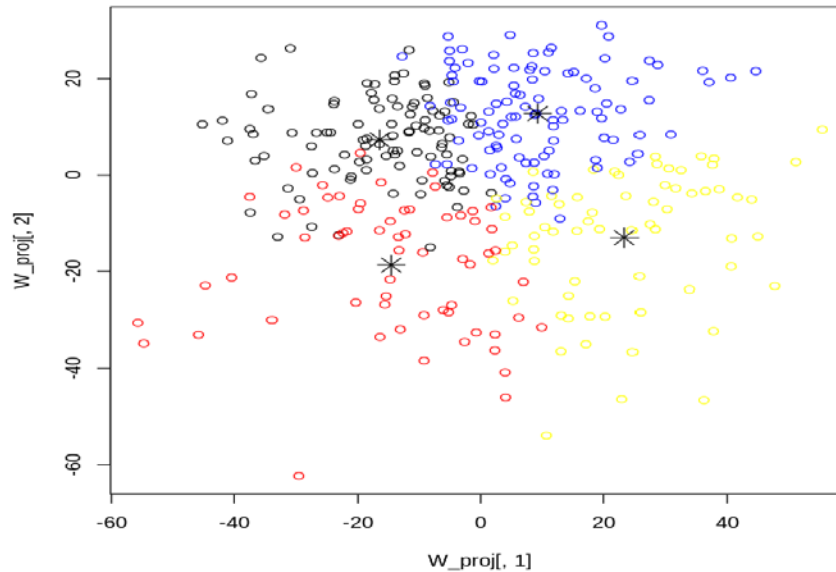
#set color scheme
col2use  = c('red','blue','black','yellow')
#set cluster assignment in colors
colassignments = col2use[k4$cluster]

W_proj = as.matrix(W_mncntr) %*% Wsvd$v[,1:3] #project data onto 3 components
```

```
plot(W_proj[,1],W_proj[,2],col=colassignments,main='data pts project to 1,2 SVD compone

# to plot center points, first project them into components
c3 = k4$centers%*% Wsvd$v[,1:3]
points(c3[,1],c3[,2],pch=8,cex=2)
```

data pts project to 1,2 SVD components, colored by kmeans



```
In [13]: #Y was created above, use it to select 2 colors

#get class assignment in colors
colassignments = col2use[Y]

plot(W_proj[,1],W_proj[,2],col=colassignments,main='data pts project to 1,2 SVD compone
```

```
In [16]: #get model predictions (more generally we would use a test set to get prediction accuracy)
Y_pred=linmodel_result$fitted.values
```

```
#get the indices of predictions NO vs YES
```

```
Y_pred1_indices =which(Y_pred<1.5)
```

```
Y_pred2_indices =which(Y_pred>=1.5)
```

```
#set up No, Yes predictions
```

```
Y_pred_class =matrix(1,length(Y),1)
```

```
Y_pred_class[Y_pred2_indices]=2
```

```
#show a confusion matrix
```

```
table(Y,Y_pred_class)
```

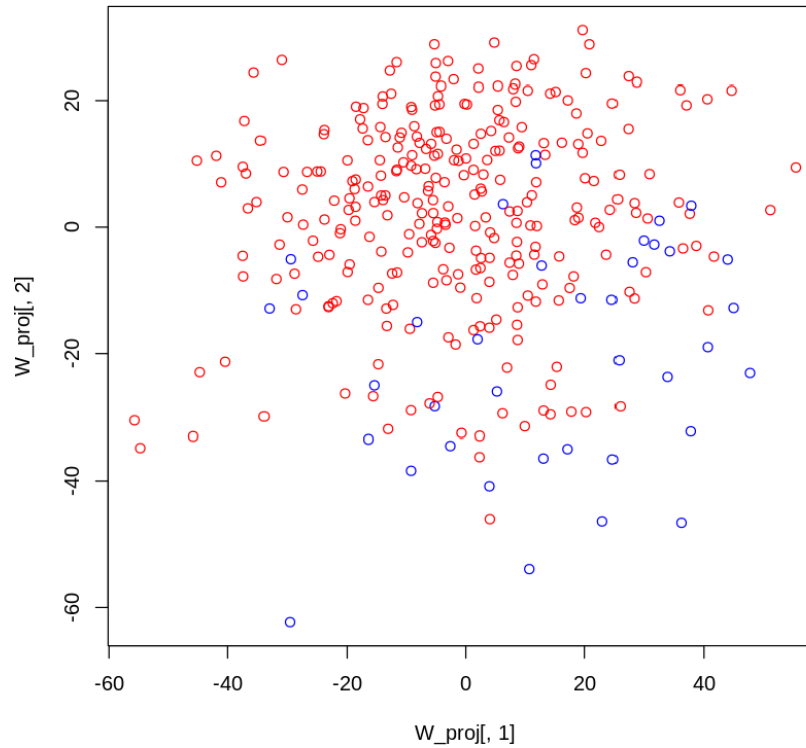
```
      Y_pred_class
Y      1      2
1 259      9
2   31     29
```

```
In [18]: #Now color the predictions onto the 2 SVD dimensions
```

```
colassignments = col2use[Y_pred_class]
```

```
plot(W_proj[,1],W_proj[,2],col=colassignments,main='data pts project to 1,2 SVD components')
```

data pts project to 1,2 SVD components, colored by PREDICTED class



#Now plot the incorrect cases in different colors

Ycol_ind =Y_pred_class

Ycol_ind[Yerr1_ind]=3 #set the color for errors, where true-value=1

Ycol_ind[Yerr2_ind]=4 # and true-value =2

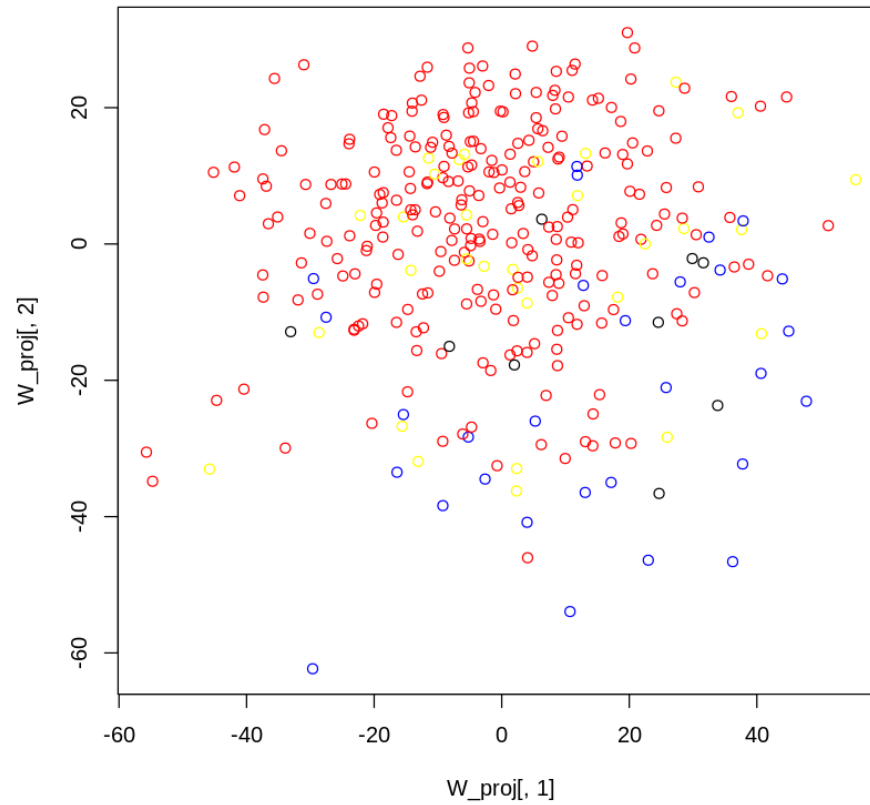
colassignments = col2use[Ycol_ind]

plot(W_proj[,1],W_proj[,2],col=colassignments,main='data pts project to 1,2 SVD components, Correct (R,BI) and Errors (Y,Bk)')

#Notice some of the incorrect class-1 cases are on the 'edges' of class 1 data in these dimensions, but other error cases

are next to correctly predicted class-1 cases, which suggest that other dimensions have useful information

data pts project to 1,2 SVD components, Correct (R,BI) and Errors (Y,BI)



- **pause**

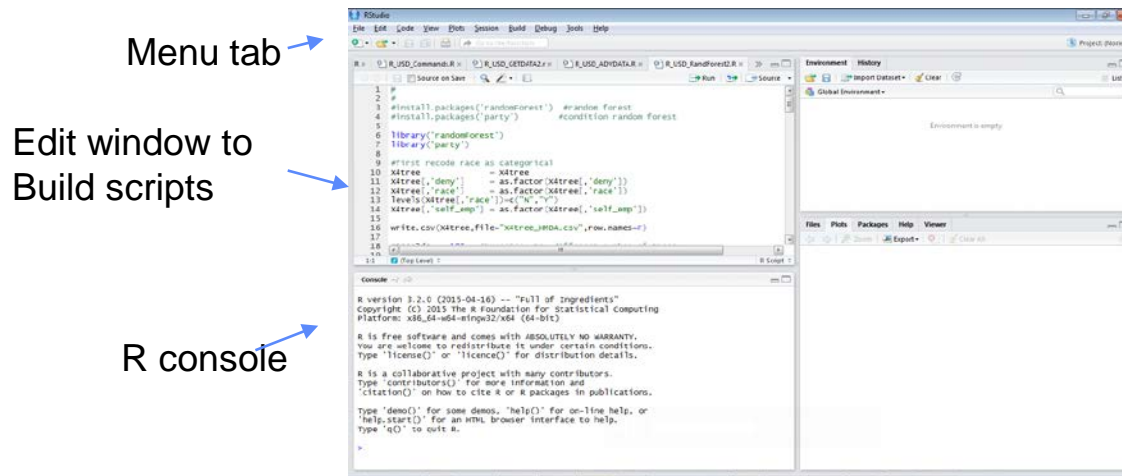
Quick R in HPC

Paul Rodriguez
SDSC



A typical R development workflow

- R studio: An Integrated development environment for R on your local machine – good for development



R commands in brief

- A typical R code workflow:

#READ DATA (housing mortgage cases)

```
X = read.csv('hmda_aer.csv', header=T, stringsAsFactors=T)
```

#SUBSET DATA

```
indices_2keep = which(X[, 's13'] %in% c(3,4,5))
```

```
X = X[unique(indices_2keep),]
```

#CREATE/TRANSFORM VARIABLES

```
pi_rat = as.numeric(X[, 's46']/100) #debt2income ratio
```

```
race = as.numeric(X[, 's13'] %in% c(3,4)) #make race values 1-4 into values 0 or 1
```

```
deny = as.numeric(X[, 's7']==3) #make deny values into 0 or 1,  
1 only for deny='3'
```

#RUN MODEL and SHOW RESULTS

```
lm_result = lm(deny~race+pi_rat) #lm is 'linear model'
```

```
summary(lm_result)
```

R strengths for HPC

- **Sampling/bootstrap methods**
- **Data Gathering and Preparation**
- **Particular Statistical procedures that you won't find implemented anywhere else, e.g.**

Multiple Imputation methods,

Instrument Variable (2 stage) Regression

Matching subjects for pairwise analysis

MCMC routines

R Scaling In a nutshell

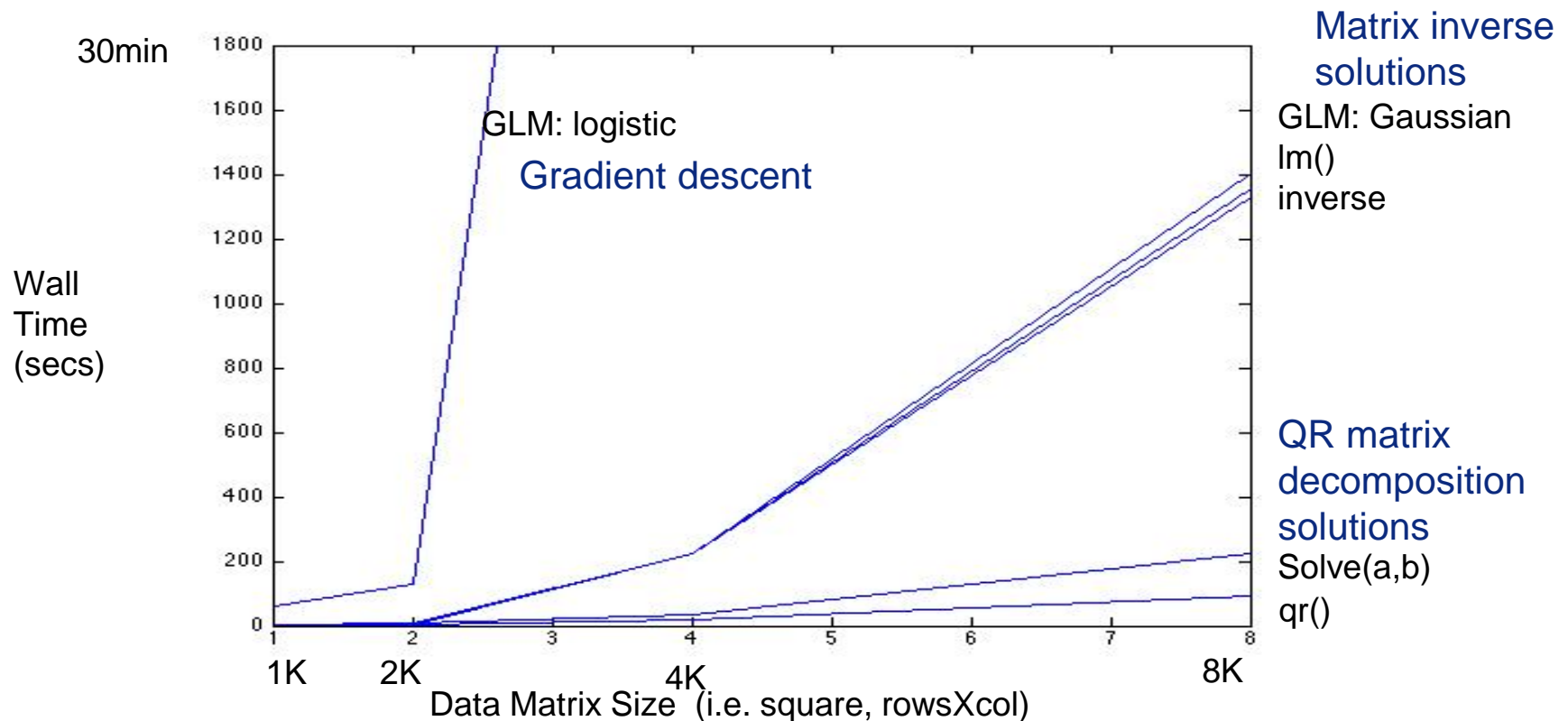
- R takes advantage of math libraries for vector operations
- R packages provide multicore, multimode, or distributed data (SparkR) options
- However, model implementations not necessarily built to use parallel backends
 - Some models more amenable to parallel versions

Solving Linear Systems: Performance with R, 1 compute node

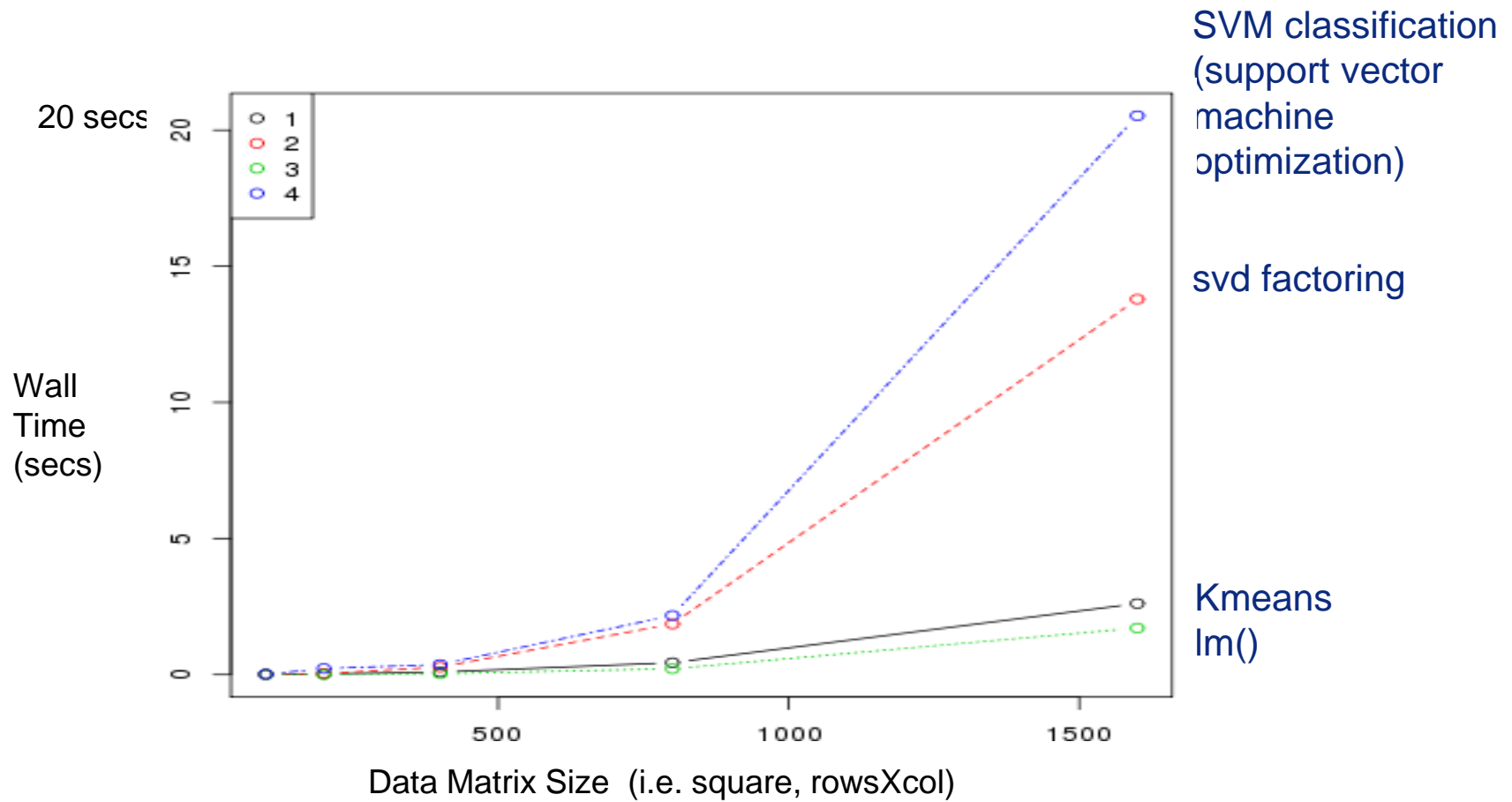
R:

```
glm(Y~X,family=gaussian) #gaussn regrssn (like lm)
```

```
glm(Y~X,family=binomial) # logistic regrssn (Y=0 or 1)
```



Machine learning models: Performance on 1 compute node



R multicore

- Run loop iterations on separate cores

```
install.packages(doParallel)  
library(doParallel)  
registerDoParallel(cores=24)
```

allocate workers



R multicore

- Run loop iterations on separate cores

```
install.packages(doParallel)  
library(doParallel)  
registerDoParallel(cores=24)
```

allocate workers



```
my_data_frame = .....
```

```
my_results = foreach(i=1:24,.combine=rbind) %dopar%  
{ ...  
    your code here  
  
    return( a variable or object )  
})
```

%dopar% puts loops
across cores,
(loops are independent)
%do% runs it serially



R multicore

- Run loop iterations on separate cores

```
install.packages(doParallel)  
library(doParallel)  
registerDoParallel(cores=24)
```

allocate workers

%dopar% puts loops
across cores,
(loops are independent)
%do% runs it serially

```
my_data_frame = .....
```

```
my_results = foreach(i=1:24,.combine=rbind) %dopar%  
{ ...  
  your code here
```

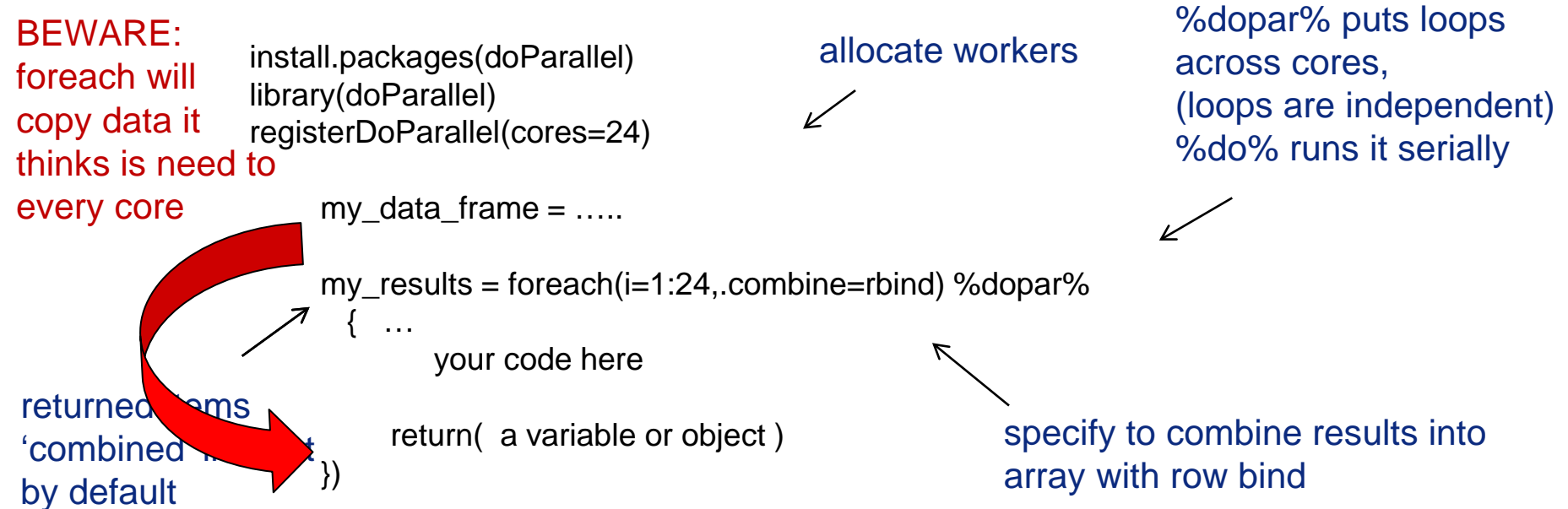
```
  return( a variable or object )
```

specify to combine results into
array with row bind

returned items
'combined' into list
by default

R multicore

- Run loop iterations on separate cores



R multinode: parallel backend


- Run loop iterations on separate nodes

```
install.packages('doSNOW')  
library('doSNOW')  
...  
cl <- makeCluster( mpi.universe.size()-1, type='MPI' )  
clusterExport(cl,c('data'))  
registerDoSNOW(cl)  
  
results = foreach(i=1:47,.combine=rbind) %dopar%  
  { ... your code here  
  
    return( a variable or object )  
  }  
stopCluster(cl)
```

allocate cluster as
parallel backend



%dopar% puts loops
across cores and
nodes



R multinode: parallel backend

- Run loop iterations on separate nodes

BEWARE:
foreach will
copy data it
thinks is need to
every node –
that can take a
long time!

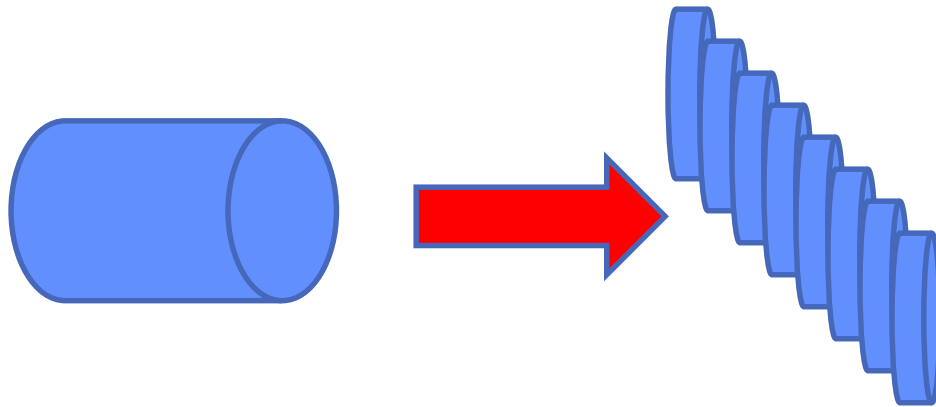
```
install.packages('doSNOW')  
library('doSNOW')  
...  
cl <- makeCluster( mpi.universe.size()-1, type='MPI' )  
clusterExport(cl,c('data'))  
registerDoSNOW(cl)  
results = foreach(i=1:47,.combine=rbind) %dopar%  
  { ... your code here  
  }  
return( a variable or object )  
})  
stopCluster(cl)
```

allocate cluster as
parallel backend

%dopar% puts loops
across cores and
nodes

Another option for (embarrassingly) Parallel R

1. Split up
data into N
parts



Another option for (embarrassingly) Parallel R

1. Split up data into N parts
2. In slurm batch script:
`ibrun -np processors My-perl-script`

My-perl-script:
get cpu-id &
pass it to R

Another option for (embarrassingly) Parallel R


1. Split up
data into N
parts

2. In slurm batch script:

```
ibrun -np processors My-perl-script
```

Init MPI and get MPI
rank

No other MPI calls
made



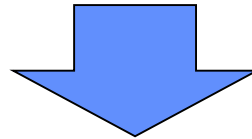
My-perl-script:
get cpu-id &
pass it to R

Another option for (embarrassingly) Parallel R

1. Split up
data into N
parts

2. In slurm batch script:

```
ibrun -np processors My-perl-script
```



CPU Core 1

My-perl-script:
get cpu-id &
pass it to R

CPU Core 2

My-perl-script:
get cpu-id &
pass it to R

...

CPU Core N

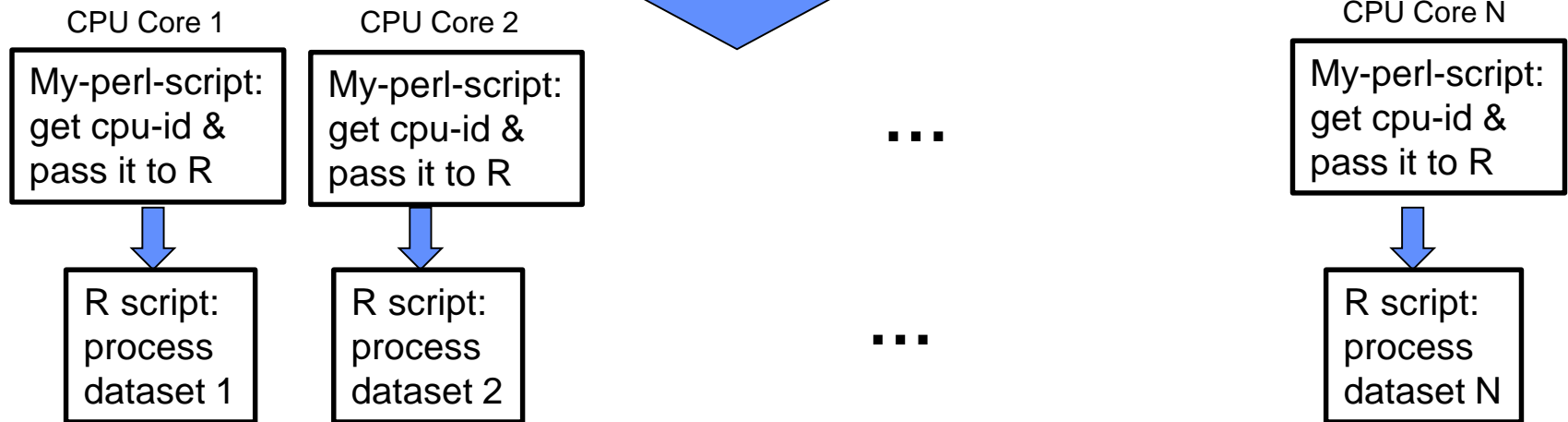
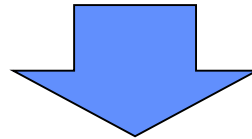
My-perl-script:
get cpu-id &
pass it to R

Another option for (embarrassingly) Parallel R

1. Split up data into N parts

2. In slurm batch script:

```
ibrun -np processors My-perl-script
```

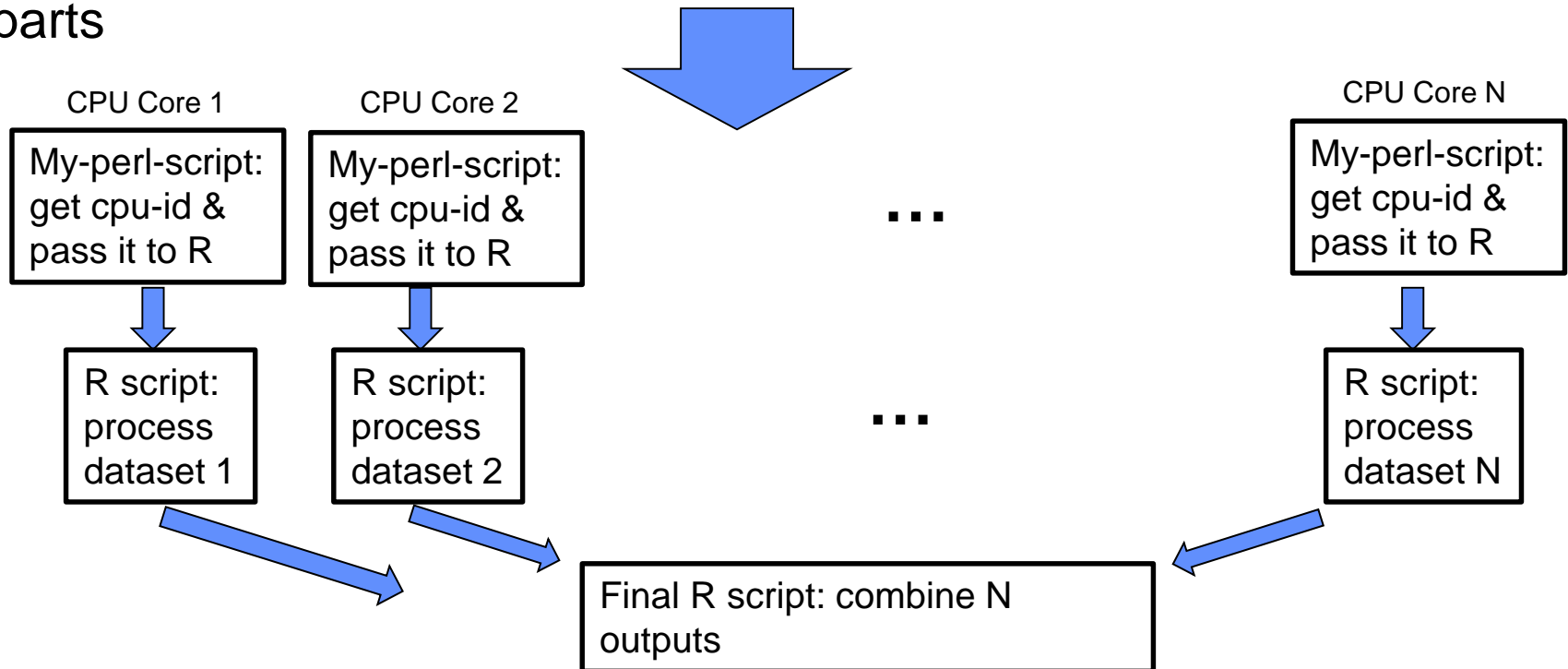


Another option for (embarrassingly) Parallel R

1. Split up data into N parts

2. In slurm batch script:

```
ibrun -np processors My-perl-script
```

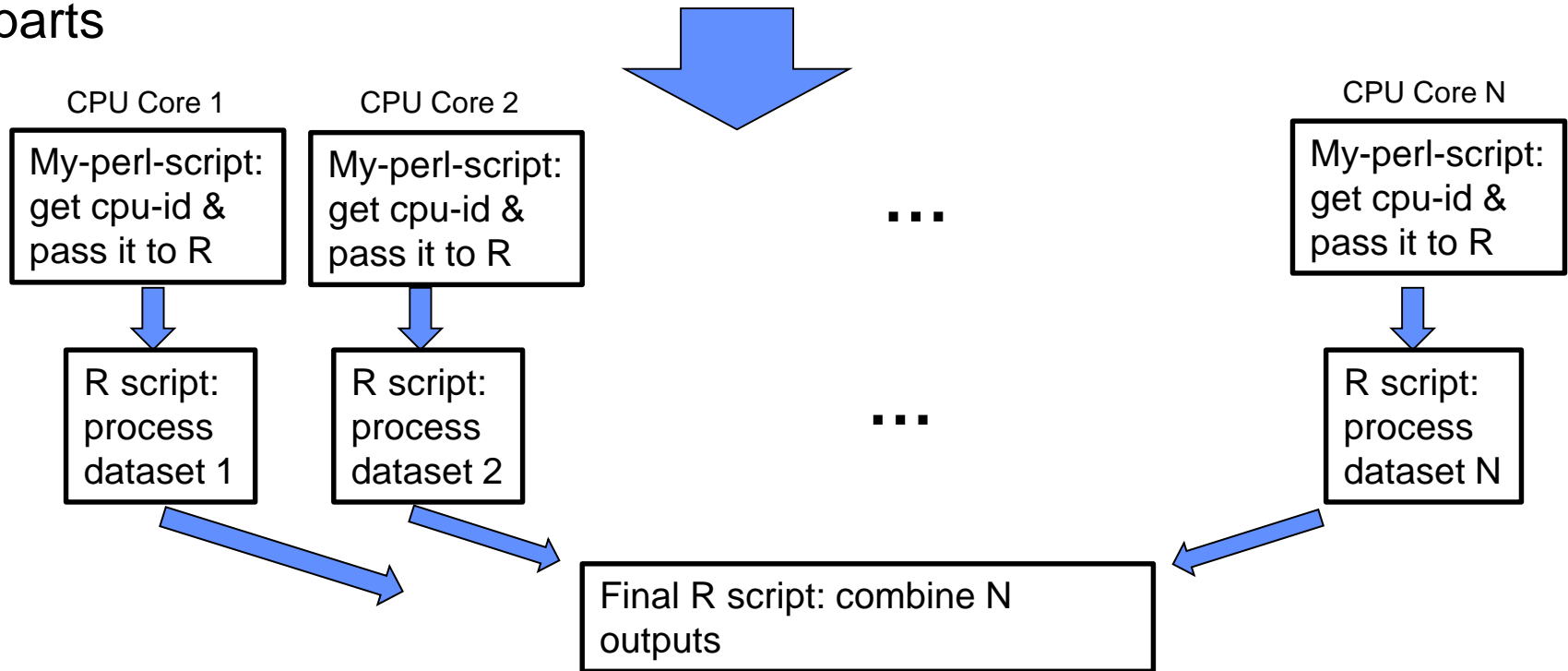


Another option for (embarrassingly) Parallel R

1. Split up data into N parts

2. In slurm batch script:

```
ibrun -np processors My-perl-script
```



More programming but more flexible

Normal
batch job
info

```
#!/bin/bash
# -----
# slurm script for a batch job on comet
# to run a task on individual cores
# -----
#SBATCH --job-name="packR"
#SBATCH --output="serial-pack.%j.%N.out"
#SBATCH --partition=compute
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=24
#SBATCH --export=ALL
#SBATCH -t 1:00:00
#SBATCH -A sds164

bash

#Generate a hostfile from the slurm node list
export SLURM_NODEFILE=`generate_pbs_nodefile`
module load R

#launch 24x2=48 tasks on 48 cores,
# and start this perl script on each task
ibrun --npernode 24 --tpp 1 perl ./bundlerxP.pl

#One can also run hybrid:
# launch 1 process per node, with 24 threads, and
# use doParallel
ibrun --npernode 1 --tpp 24 perl ./bundlerxP.pl
```

ibrun the
'bundler' perl
script on 24
cores per
nodes, and 1
thread each

the
'bundler'
Perl script

```
#!/usr/bin/perl  
use strict;  
use warnings;
```

the backtick executes
system command

```
my ($myid, $numprocs) = split(/\s+/, `./getid`);  
  
# -----  
# launch an R session for this task  
# -----  
my $task_index = $myid+1;  
`module load R;/opt/R/bin/Rscript Test_PackingR.R $task_index >  
  Rstd_out.$task_index.txt`;
```

Get current cpu
id and number
of processes

execute R and
pass the rank id
as an argument

Scaling doParallel vs 'Packing' R sessions

- **Packing *independent* R sessions onto cores is more flexible for:**
 - data management
 - large number of separate models
 - large variation in time per model
 - large matrix operations repeated
 - hybrid multimode/multicore scripts

But requires more programming or preprocessing

Example: scaling MCMC

Distributed Markov Chain Monte Carlo for Bayesian Hierarchical Models,
Frederico Bumbaca, UCIrvine, et al in print

- Probabilities of user web activity interdependent through a hierarchical model
- MCMC search for probabilities made independent through a phased approach.
- Ran on SDSC Comet with '**serial packing**' parallelization

(Using rhierMnIRwMixturefunction in the R package, bayesm)

# Individuals	Cores	Individ per Core	Total Minutes (I/O time)
100 million	1,7282 (max)	~ 58K	206 (38)

Example: scaling MCMC

Localizing social media hot spots (work in progress with UCIrvine)

- Individual spatial mixture models for users' geocoded social media use
- MCMC search for location probabilities are independent across users, but convergence time varies depending on user variations
- Ran on SDSC Comet with '**serial packing**' parallelization, with many cores for short runs, then few cores for longer runs

(using Rgeoprofile package with MCMC)

# Individuals	Cores	Approx Hours
~3000	192-288	2-3
~2000	48-96	4-8
~100	24	12-24

Example: scaling likelihoods

Social network evolution (work in progress with UTDallas)

- A large model of users' connections with interdependent variance terms for different actions
- Optimization, with ~70M observations (5-8Gb), takes > 48 hours on 1 compute node.
- R parallel copies too much data across nodes or cores
- R-mpi not flexible enough with nodes and cores
- Ran with '**serial packing**' parallelization on parts of data across nodes, with R parallel across cores (but not all cores),

(using Optim, doParallel, and send results back to main node through files)

# Connections	Cores	Approx Hours
~70M	288	2-3

Installing your own R Packages

- **In R:**

install.packages('package-name')

(see <https://cran.r-project.org/> for package lists and reviews)

- **on Comet:**

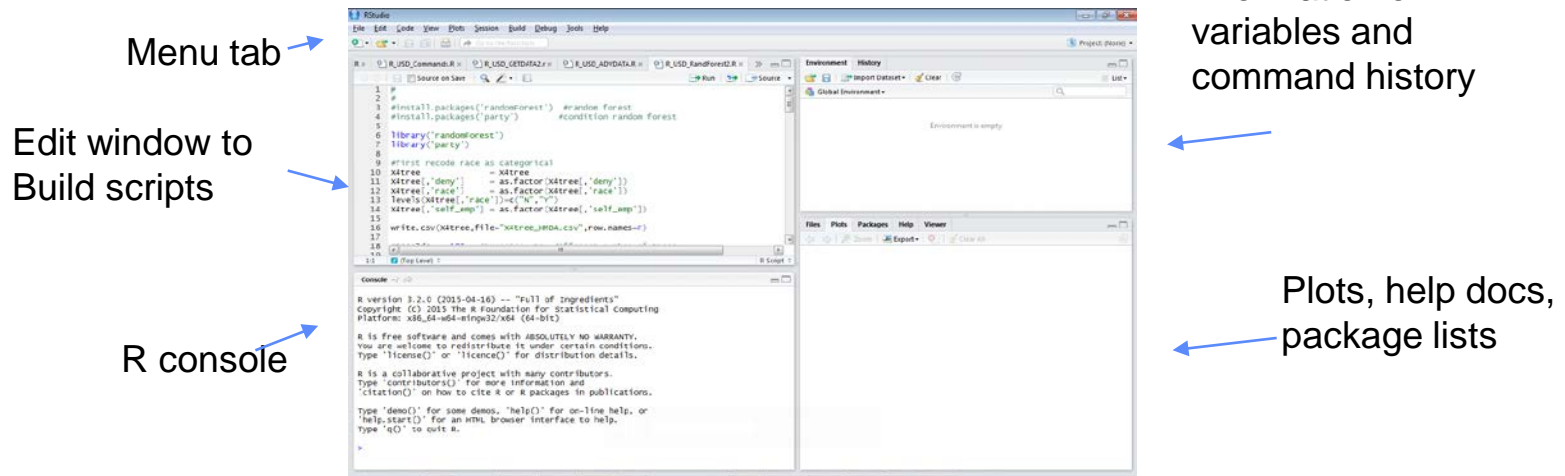
*install.packages('package-name',
repos='http://cran.us.r-project.org',dependencies=TRUE)*

If compiling is required and you get an error, call support

R-studio

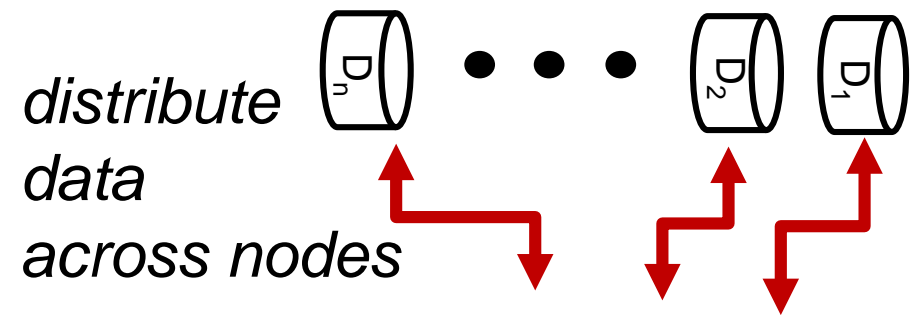
- R studio: An Integrated development environment for R on your local machine – good for development

now available on XSEDE through Jetstream cloud



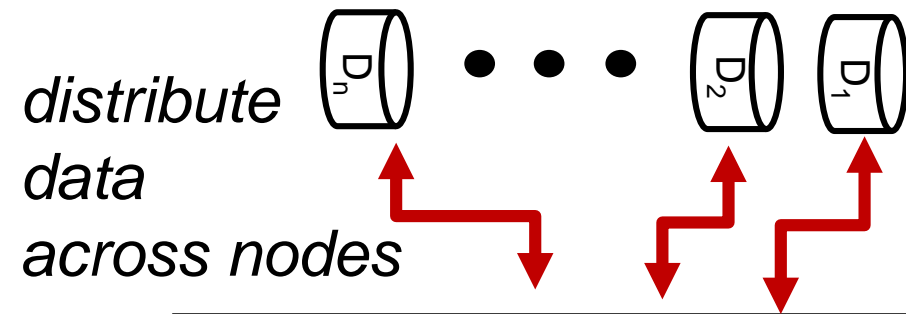
Other R packages:

- **Rspark** - R interface to Spark
- **pdbR** - MPI-based support for distributed matrix
(better for dense matrices vs Spark)
- **Rgputools** – GPU support
- **Ff, bigmemory; Revolution Scale R** – map data to files
- **Quick-R cheat sheets:**
<https://www.statmethods.net/index.html>



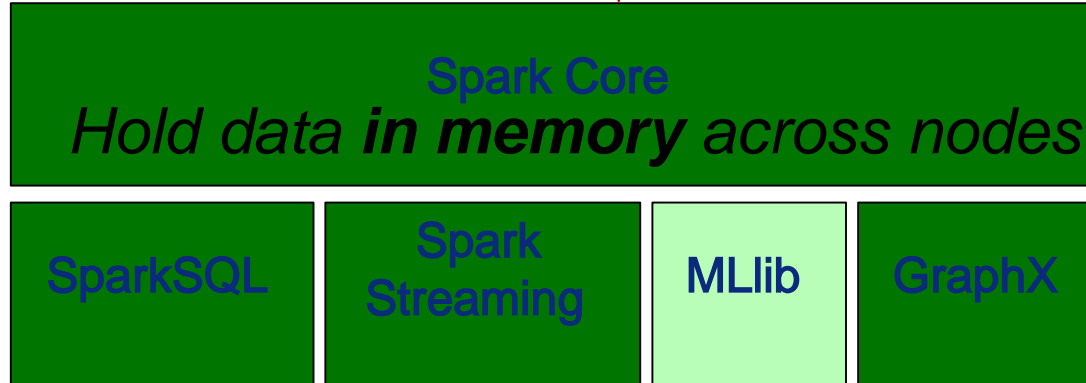
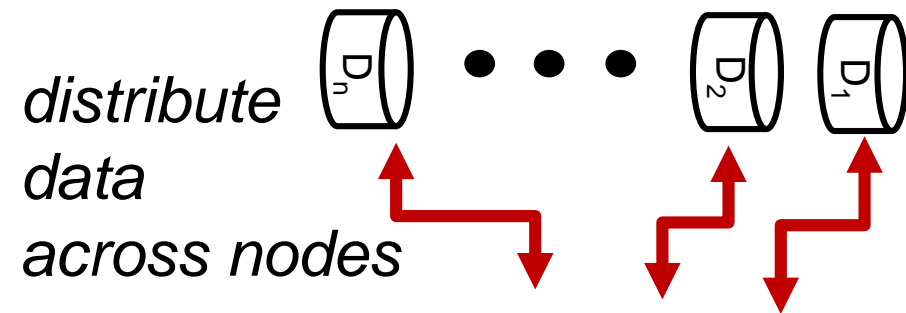
Spark MLlib

Spark MLlib



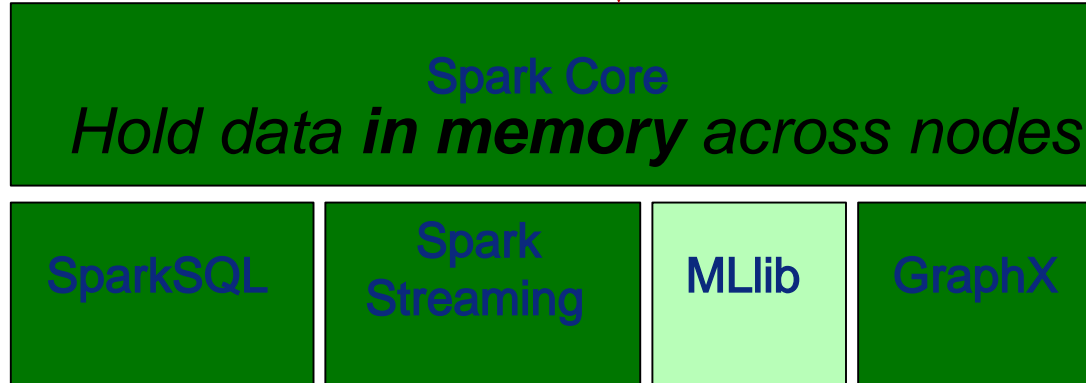
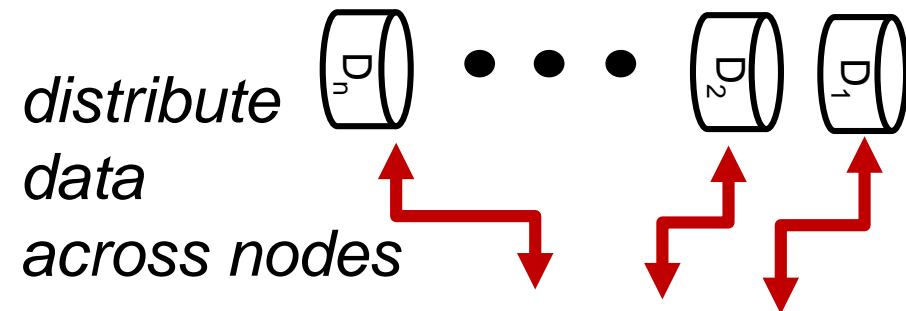
Spark Core
*Hold data **in memory** across nodes*

Spark MLlib



Run code on each part and gather as requested

Spark MLlib



Run code on each part and gather as requested

- Distributed implementations of common ML algorithms and utilities
- APIs for Scala, Java, Python, and R
- Scales well for independent processes

- **pause**

Quick Deep Learning

Paul Rodriguez SDSC



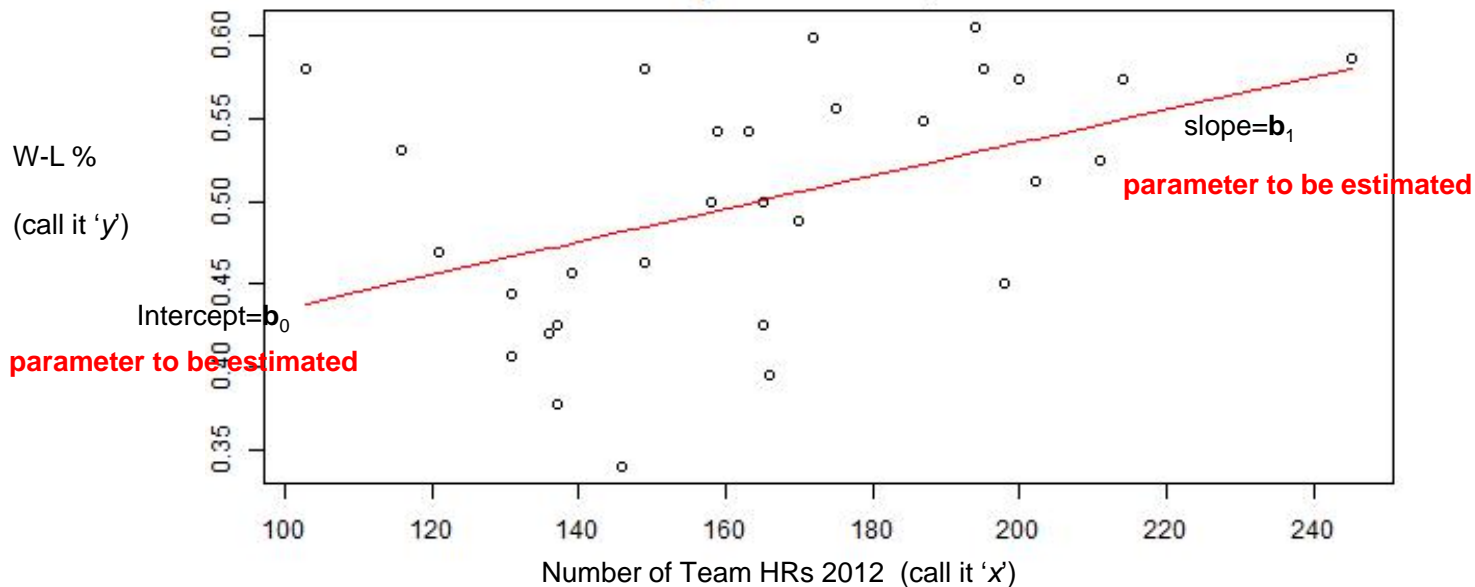
Deep Learning

- **3 characterizations:**
 1. Learning complicated interactions about input
 2. Discovering complex feature transformations
 3. Using neural networks with many layers

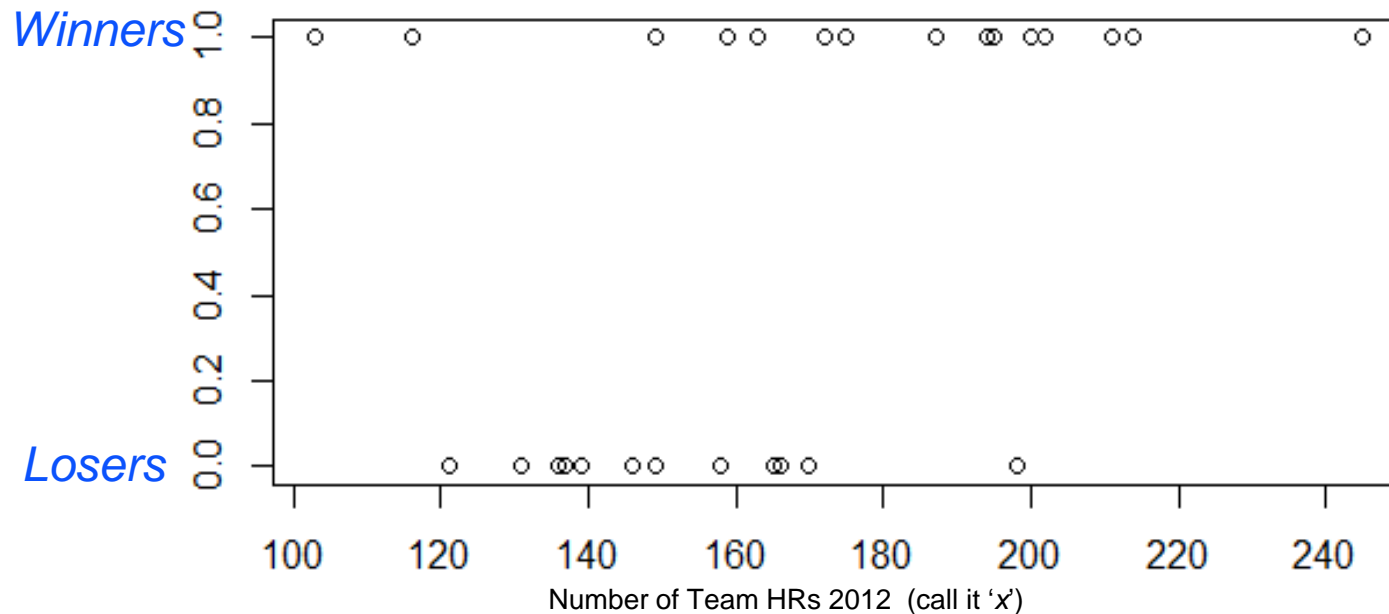
Explanation Strategy: Start with linear regression and go deep

Recall Linear Regression is Fitting a Line

the Model: $y = f(x, b) = b_0 * 1 + b_1 * x$

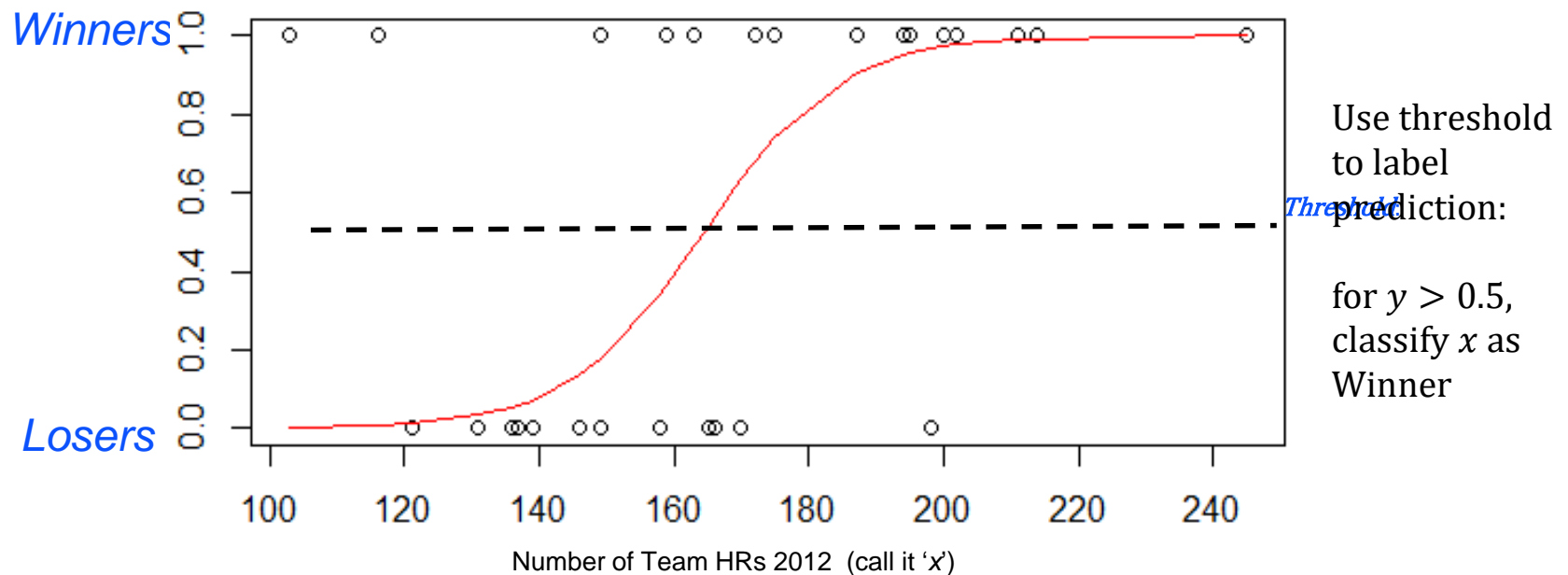


Classification uses labelled outcomes



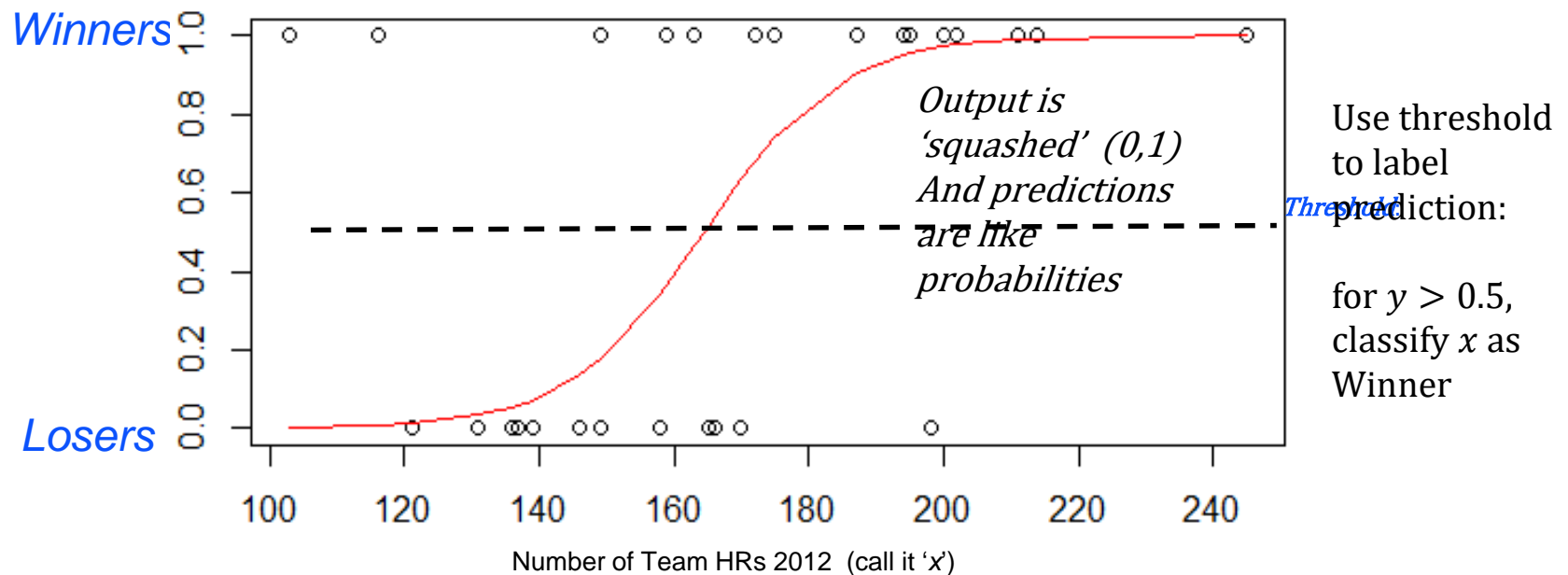
Can do better: fit a nonlinear function

the Model: $y = f(x, b) = 1/(1 + \exp[-(b_0 * 1 + b_1 * x)])$

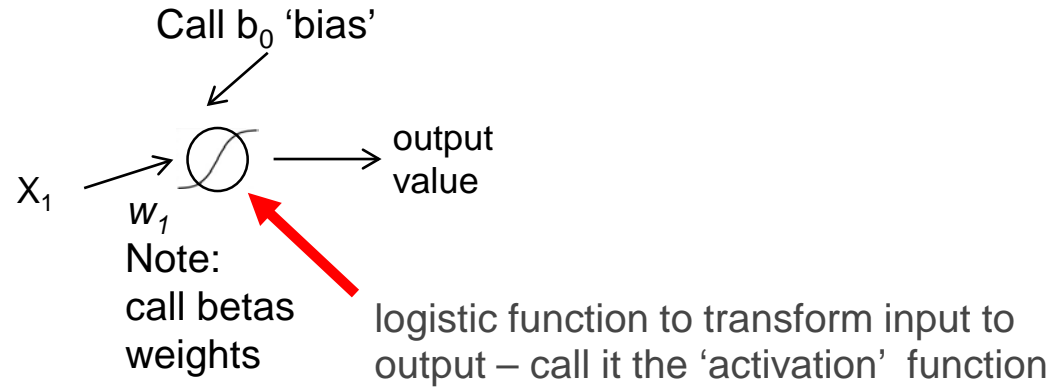


Can do better: fit a nonlinear function

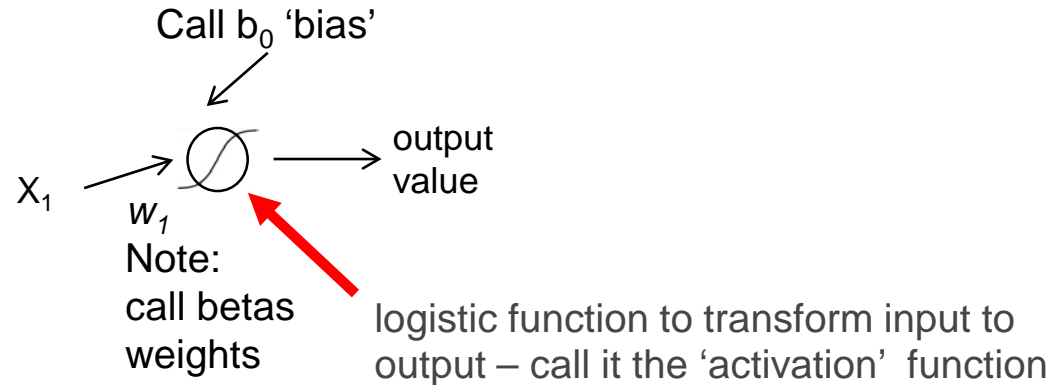
the Model: $y = f(x, b) = 1/(1 + \exp[-(b_0 * 1 + b_1 * x)])$



Logistic Regression as 1 node network

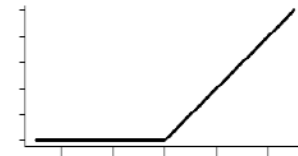


Logistic Regression as 1 node network

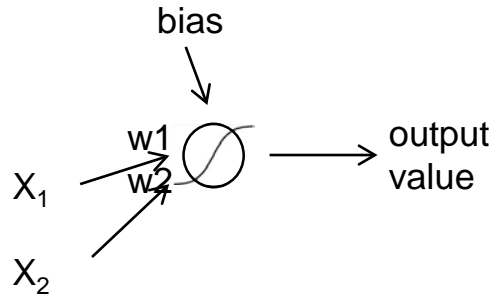


Note: other activations are possible,

RELU (rectified linear unit)



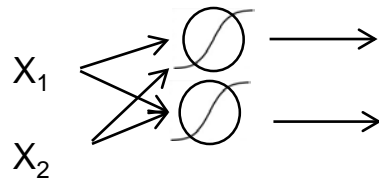
Next step: More general networks



Add input variables

More general networks

(assume bias present)

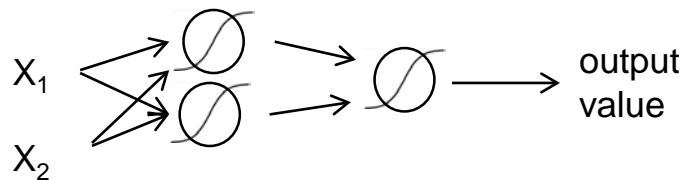


Add input variables

Add logistic transformations ...

More general networks

(assume bias)

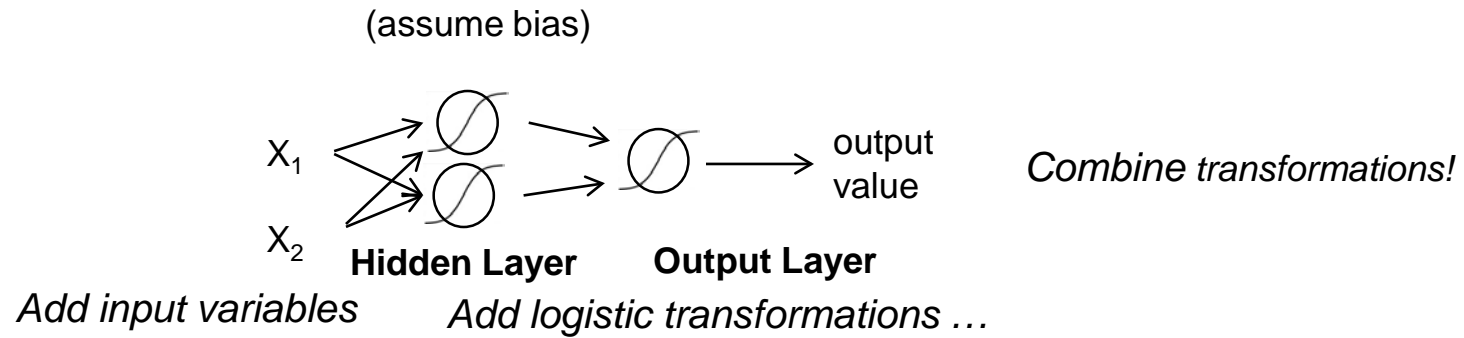


Combine transformations!

Add input variables

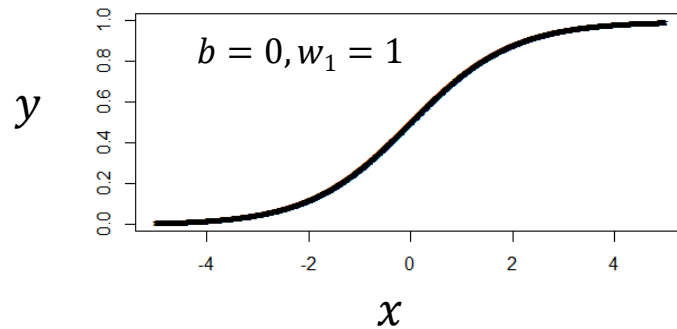
Add logistic transformations ...

More general networks



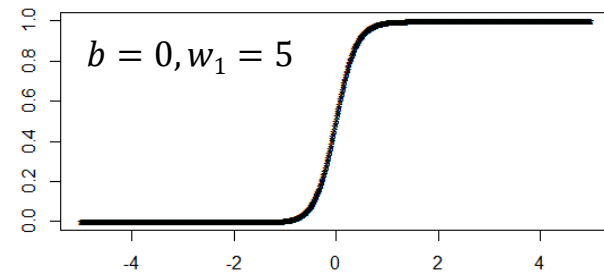
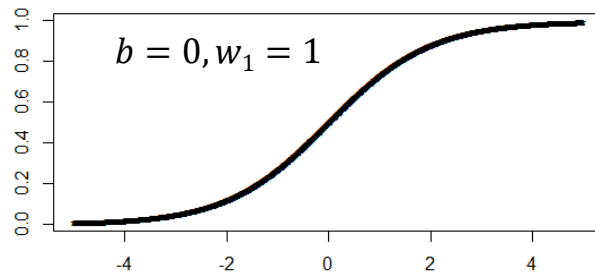
Logistic function w/various weights

$$\text{for } y = 1 / (1 + \exp(-(b + w_1 * x)))$$



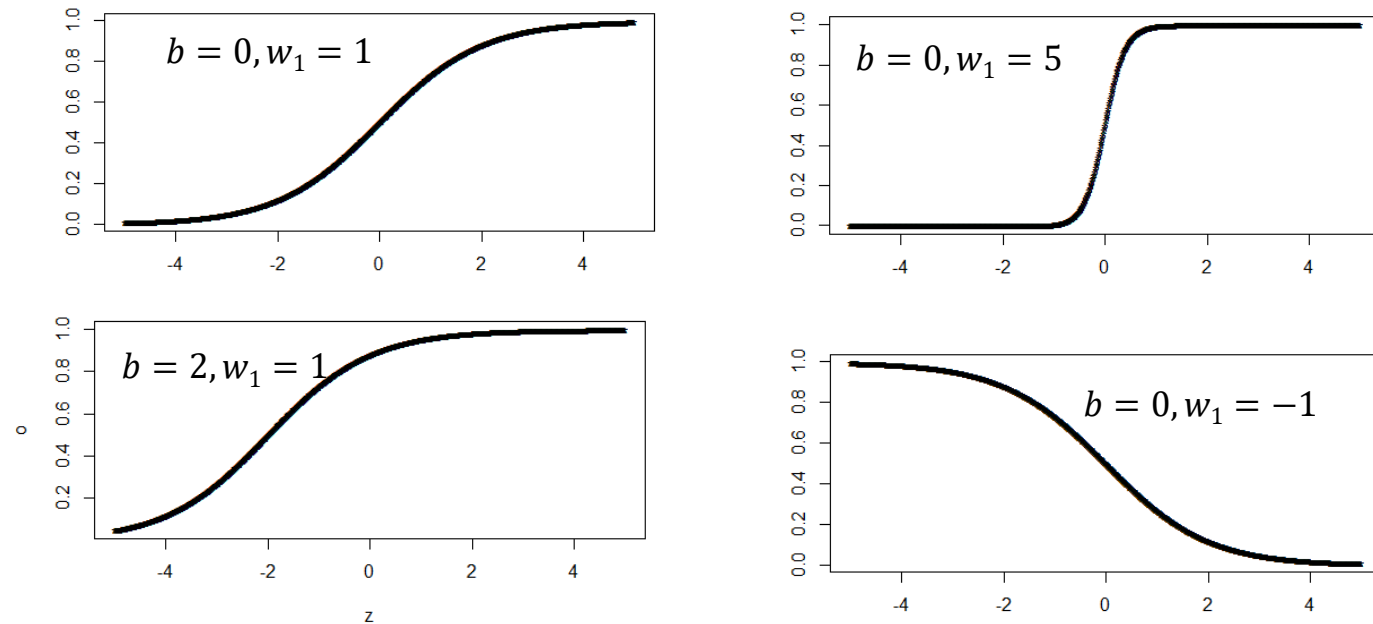
Logistic function w/various weights

$$\text{for } y = 1 / (1 + \exp(-(b + w_1 * x)))$$

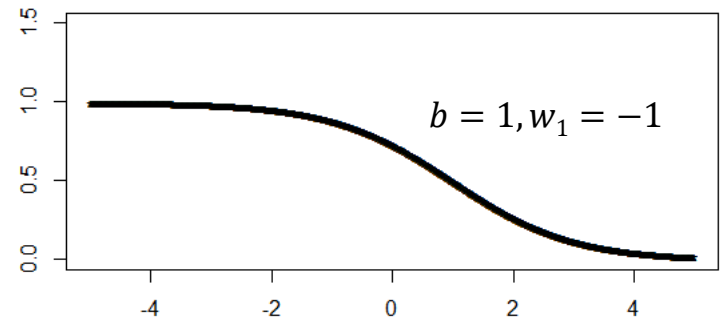
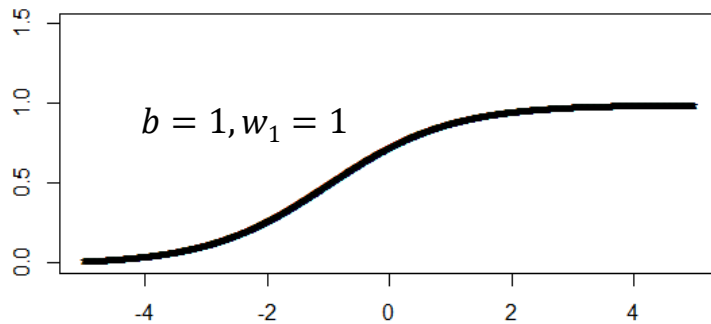


Logistic function w/various weights

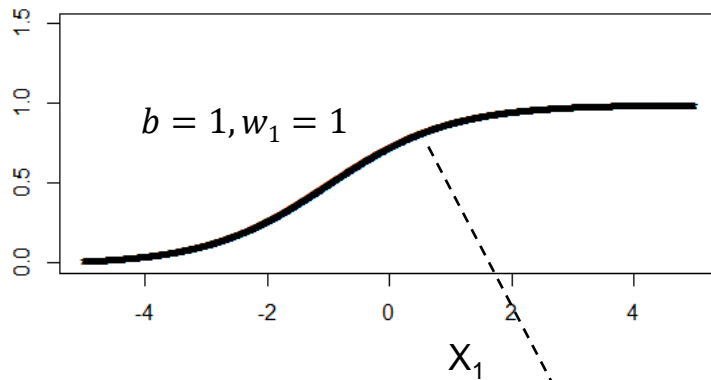
$$\text{for } y = 1 / (1 + \exp(-(b + w_1 * x)))$$



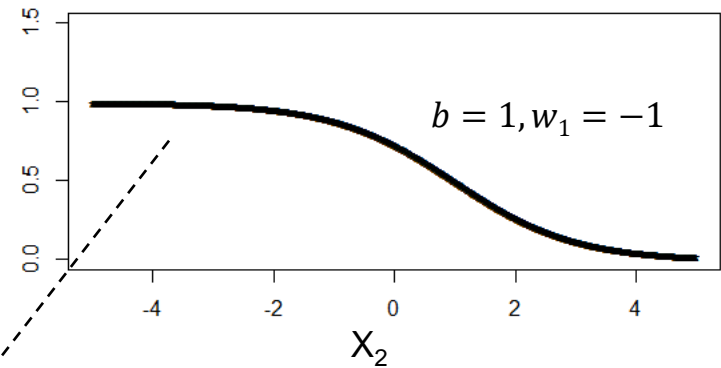
So combinations are highly flexible and nonlinear



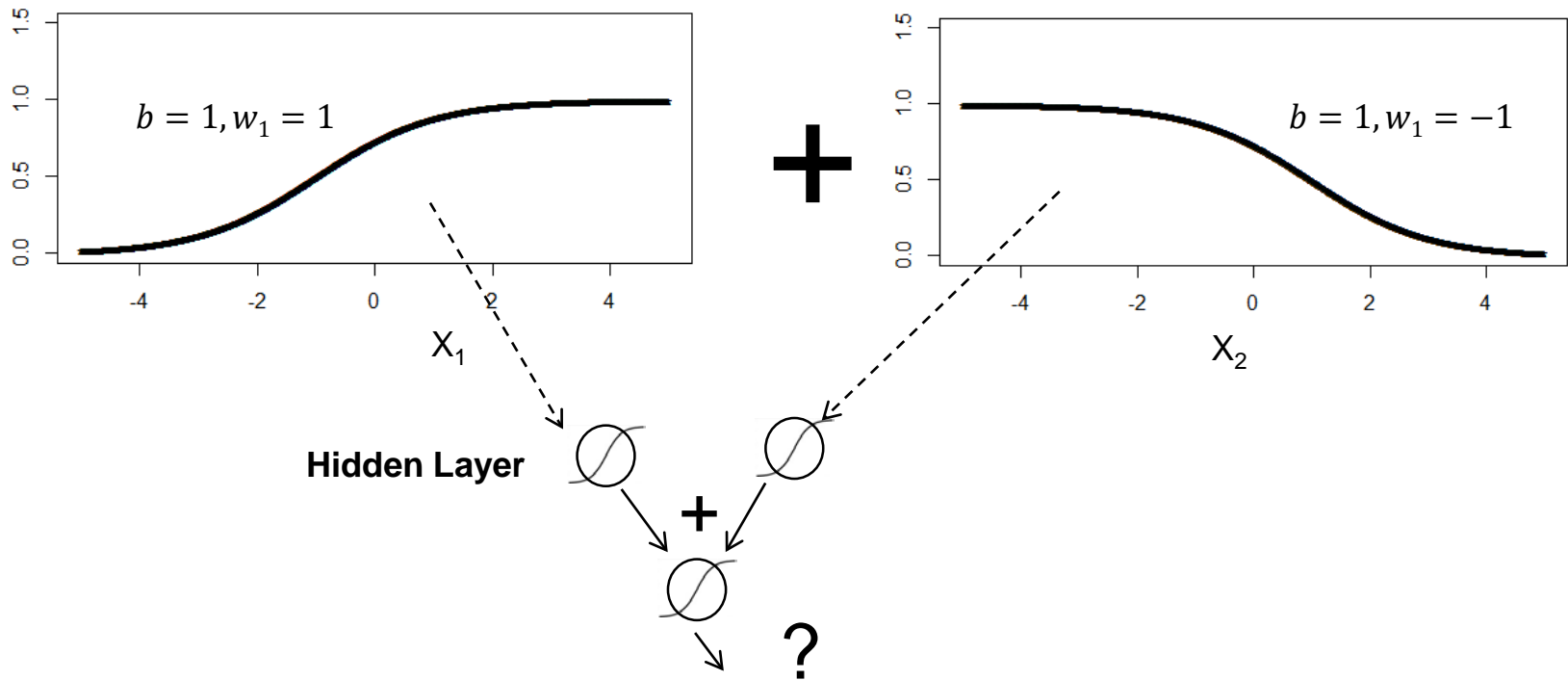
So combinations are highly flexible and nonlinear



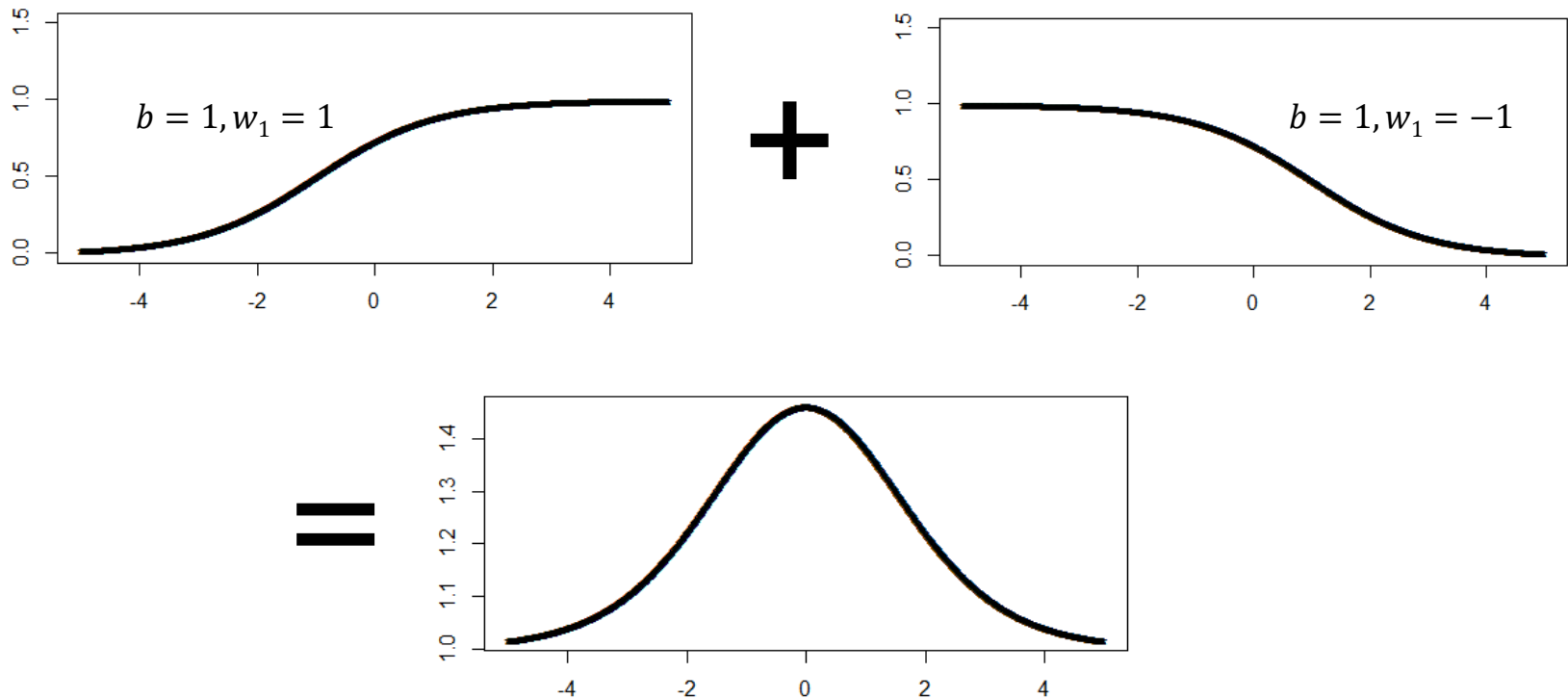
Hidden Layer



So combinations are highly flexible and nonlinear

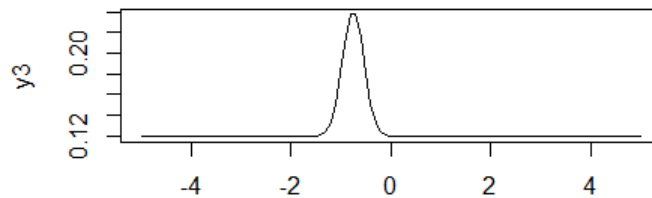


So combinations are highly flexible and nonlinear

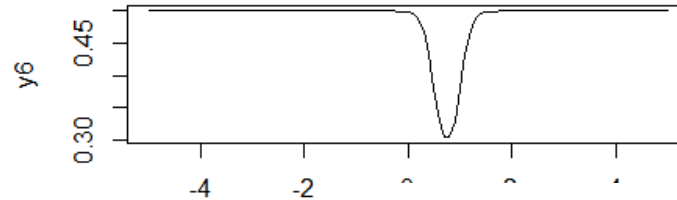


More function combinations

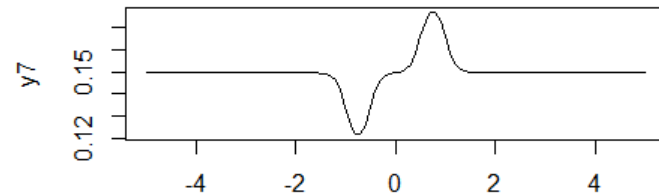
```
x=seq(-5,5,.1)
y1=1/(1+exp(10+ 10*x))
y2=1/(1+exp(-5+(-10)*x))
y3=1/(1+exp(1+1*y1+1*y2))
plot(x,y3,type="l")
```



```
y4=1/(1+exp(10+ (-10)*x))
y5=1/(1+exp(-5+(10)*x))
y6=1/(1+exp(1-1*y4-1*y5))
plot(x,y6,type="l")
```

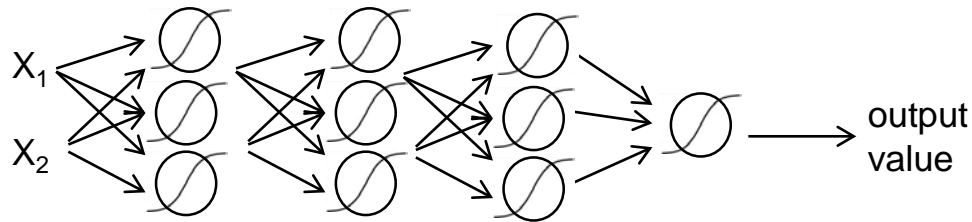


```
y7=1/(1+exp(1+2*y3+1*y6))
plot(x,y7,type="l")
```

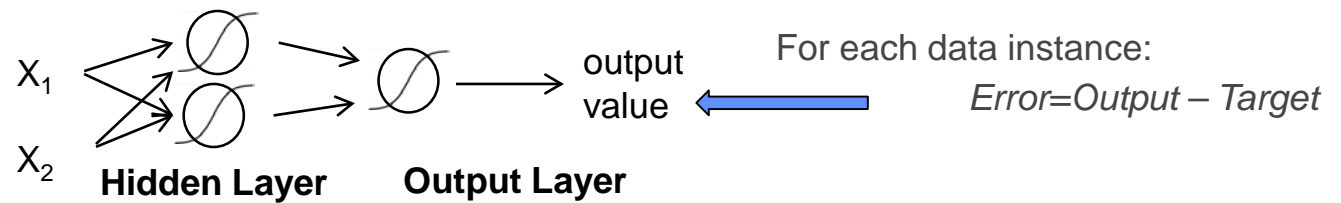


Why stop at 1 hidden layer?

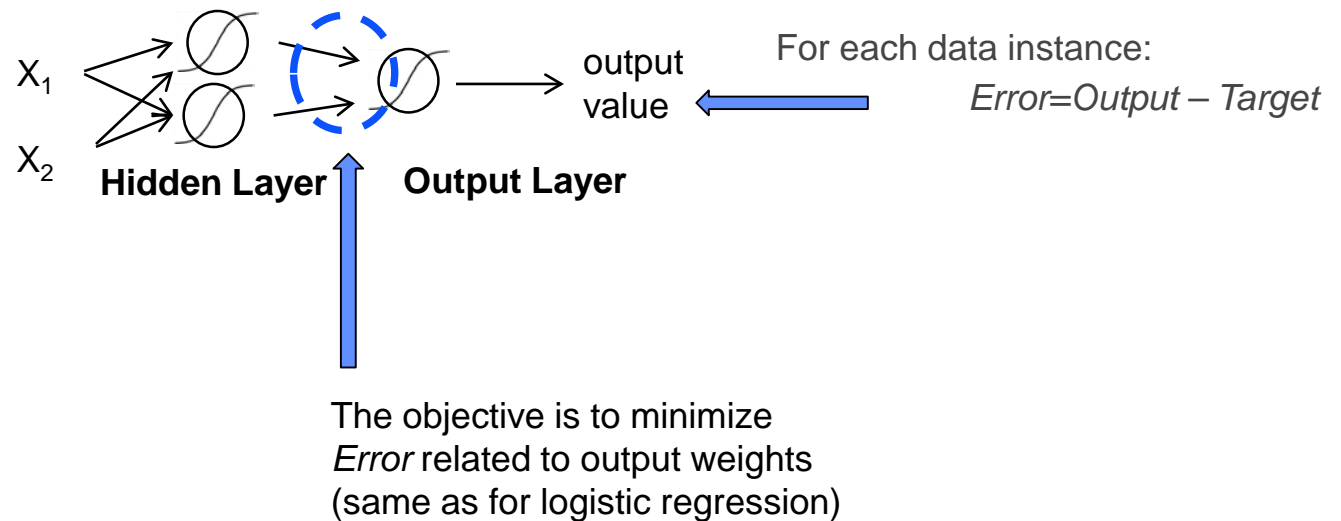
More hidden layers => More varied features and transformation



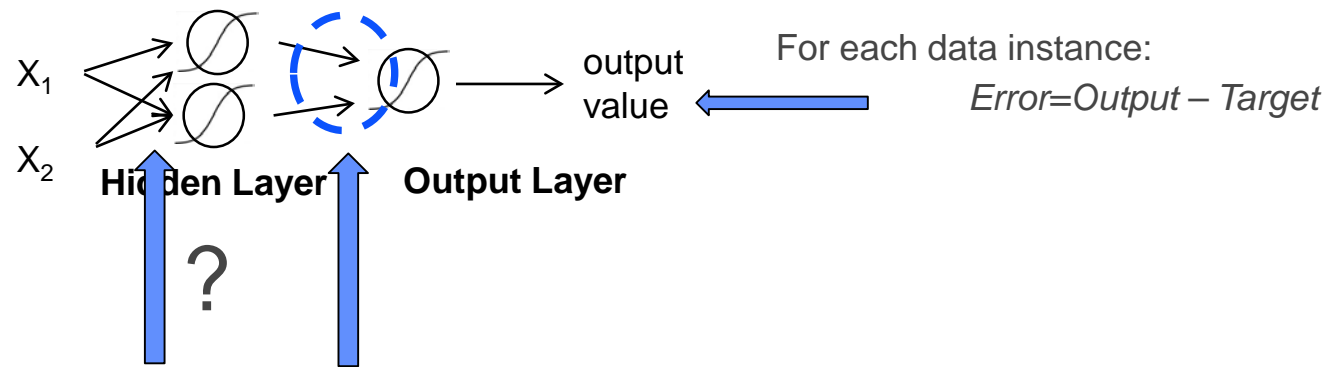
But parameter fitting is harder too



But parameter fitting is harder too



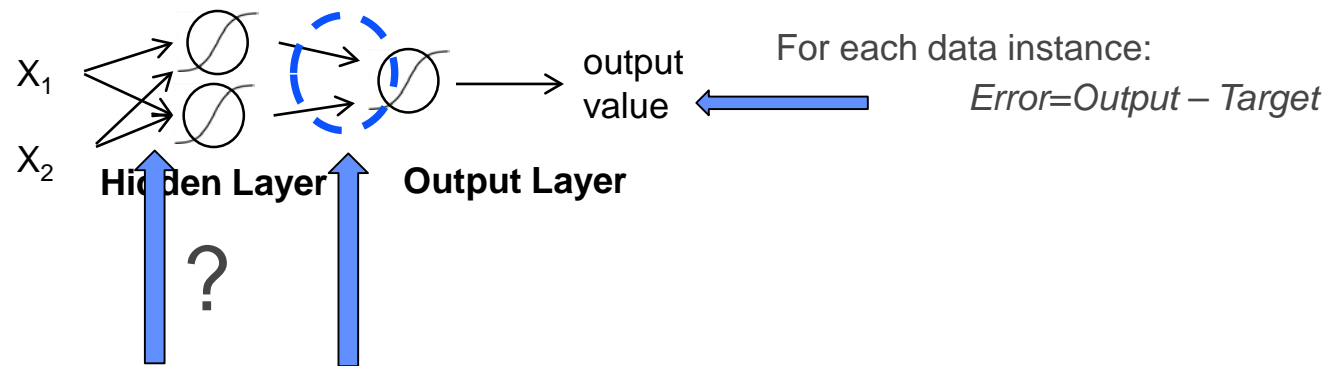
But parameter fitting is harder too



But, error signals are only known for output layer, what is error for hidden layer?

The objective is to minimize *Error* related to output weights (same as for logistic regression)

But parameter fitting is harder too



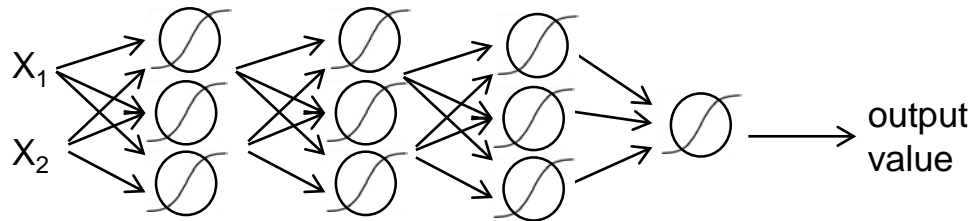
But, error signals are only known for output layer, what is error for hidden layer?

The objective is to minimize *Error* related to output weights (same as for logistic regression)

Solution: Minimize *Error* related to output weights, that is also related to hidden weights
(Use derivatives to 'back-propagate' errors, "stochastic gradient descent")

Train with Care

More hidden layers => More varied features and transformations

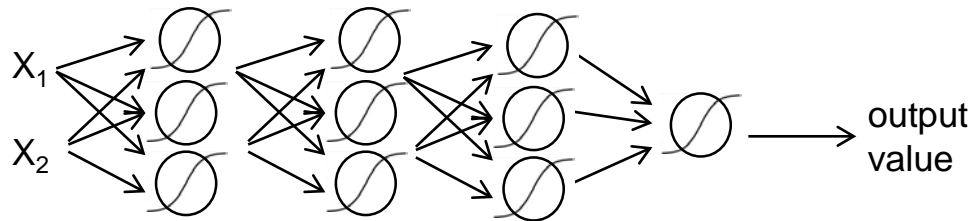


But:

More layers => more parameters

Train with Care

More hidden layers => More varied features and transformations

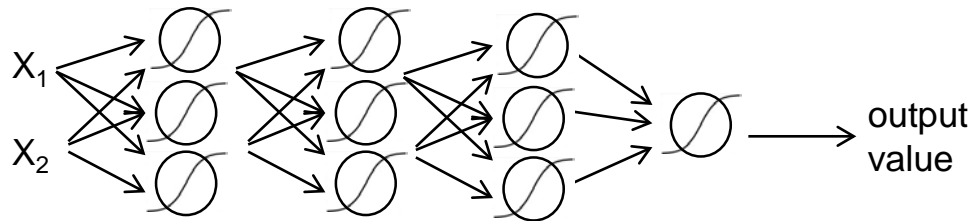


But:

**More layers => more parameters => Smaller error for each
especially at lower layers**

Train with Care

More hidden layers => More varied features and transformations



But:

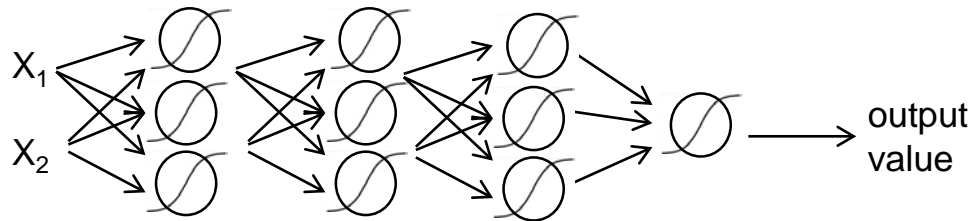
**More layers => more parameters => Smaller error for each
especially at lower layers**

Need:

More data and computing power (gpu)

Train with Care

More hidden layers => More varied features and transformations



But:

**More layers => more parameters => Smaller error for each
especially at lower layers**

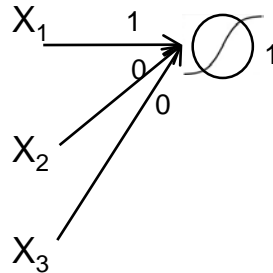
Need:

More data and computing power (gpu), functions that don't saturate(RELU)

Feature Transformations, Projections, and Convolutions

A Simple Transformation

3 input variables fully connected (dense) to 3 hidden nodes
(assume $b_0=0$, assume all X normalized between 0 and 1)



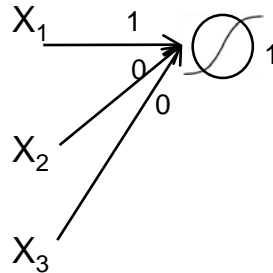
Call the connection parameters 'weights'.

For node 1 let $[w_1 \ w_2 \ w_3] = [1 \ 0 \ 0]$

*What feature transformation W^*X is that?*

A Simple Transformation

3 input variables fully connected (dense) to 3 hidden nodes
(assume $b_0=0$, assume all X normalized between 0 and 1)



Call the connection parameters 'weights'.

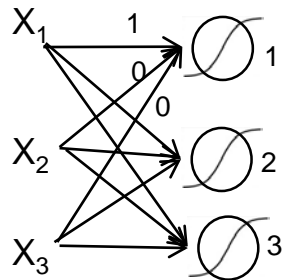
For node 1 let $[w_1 \ w_2 \ w_3] = [1 \ 0 \ 0]$

*What feature transformation W^*X is that?*

Informally, squash X_1 and ignore X_2, X_3

A Simple Transformation

3 input variables fully connected (dense) to 3 hidden nodes



For node 1 let $[w_1 \ w_2 \ w_3] = [1 \ 0 \ 0]$

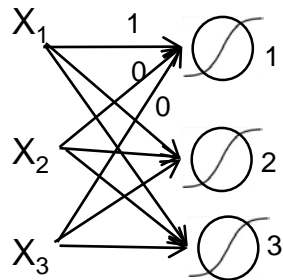
For node 2 let $[w_1 \ w_2 \ w_3] = [0 \ 1 \ 0]$

For node 3 let $[w_1 \ w_2 \ w_3] = [0 \ 0 \ 1]$

*What feature transformation W^*X are these together?*

A Simple Transformation

3 input variables fully connected (dense) to 3 hidden nodes



For node 1 let $[w_1 \ w_2 \ w_3] = [1 \ 0 \ 0]$

For node 2 let $[w_1 \ w_2 \ w_3] = [0 \ 1 \ 0]$

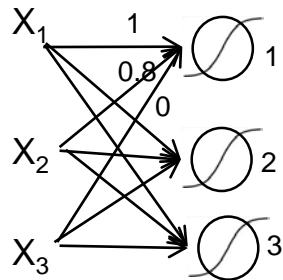
For node 3 let $[w_1 \ w_2 \ w_3] = [0 \ 0 \ 1]$

*What feature transformation W^*X are these together?*

Informally, squash 3D to another 3D space

A Factor Transformation

3 input variables fully connected (dense) to 3 hidden nodes



For node 1 let $[w_1 \ w_2 \ w_3] = [1 \ 0.8 \ 0]$

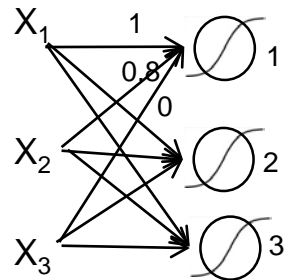
For node 2 let $[w_1 \ w_2 \ w_3] = [-0.8 \ 1 \ 0]$

For node 3 let $[w_1 \ w_2 \ w_3] = [0 \ 0 \ 0]$

*What feature transformation W^*X are these together?*

A Factor Transformation

3 input variables fully connected (dense) to 3 hidden nodes



For node 1 let $[w_1 \ w_2 \ w_3] = [1 \ 0.8 \ 0]$

For node 2 let $[w_1 \ w_2 \ w_3] = [-0.8 \ 1 \ 0]$

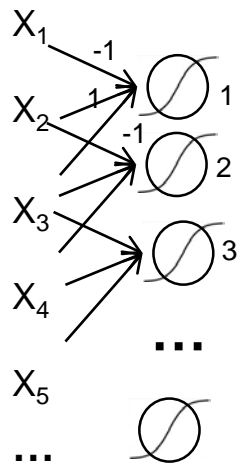
For node 3 let $[w_1 \ w_2 \ w_3] = [0 \ 0 \ 0]$

What feature transformation are these together?

Informally, like projection onto 2 orthogonal dimensions
(recall PCA example on Athletes Height and Weight!)

A Filter

Many X input, but only 3 connections to each hidden node
(a *local connectivity pattern*, aka *receptive field*)

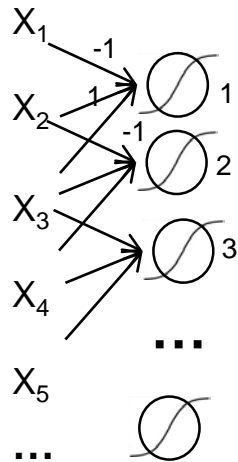


For node 1 let $W = [w_1 \ w_2 \ w_3] = [-1 \ 1 \ -1]$

What is the node 1 doing?
(assuming W are just ± 1)

A Filter

Many X input, but only 3 connections to each hidden node
(a *local connectivity pattern*, aka *receptive field*)



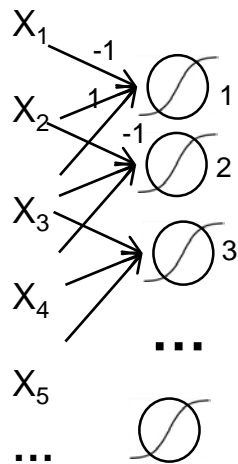
For node 1 let $W = [w_1 \ w_2 \ w_3] = [-1 \ 1 \ -1]$

What is the node 1 doing?
(assuming x are just ± 1)

Informally, node 1 has max activation for a 'spike',
e.g. when X_2 is positive and X_1, X_3 are negative

A Filter

Many X input, but only 3 connections to each hidden node
(a *local connectivity pattern*, aka *receptive field*)



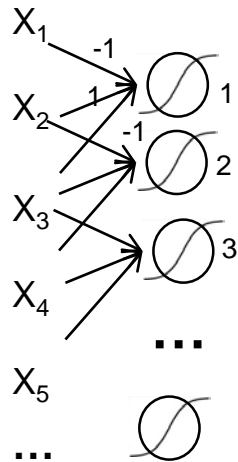
For node 1 let $W = [w_1 \ w_2 \ w_3] = [-1 \ 1 \ -1]$

For node 2,3, etc... copy W for node 1

What is the hidden layer doing?

A Filter

Many X input, but only 3 connections to each hidden node
(a *local connectivity pattern*, aka *receptive field*)



For node 1 let $W=[w_1 \ w_2 \ w_3] = [-1 \ 1 \ -1]$

For node 2,3, etc... copy W for node 1

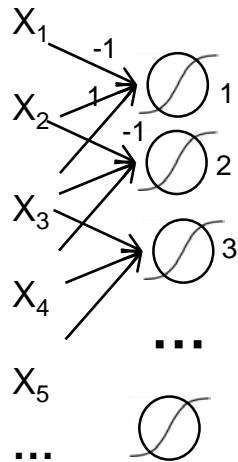
What is the hidden layer doing?

Informally, looking for a spike everywhere.

This is essentially a convolution operator,
where W is the kernel.

A Filter

Many X input, but only 3 connections to each hidden node
(a *local connectivity pattern*, aka *receptive field*)



For node 1 let $W=[w_1 \ w_2 \ w_3] = [-1 \ 1 \ -1]$

For node 2,3, etc... copy W for node 1

What is the hidden layer doing?

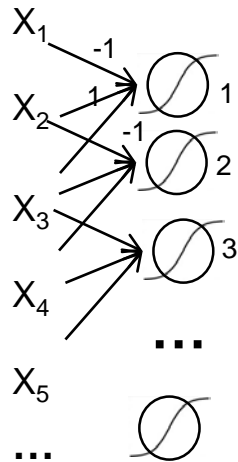
Informally, looking for a spike everywhere

This is essentially a convolution operator, where W is the kernel.

Note: sharing weights is like *sliding* W across input

A Filter

Many X input, but only 3 connections to each hidden node
(a *local connectivity pattern*, aka *receptive field*)



For node 1 let $W=[w_1 \ w_2 \ w_3] = [-1 \ 1 \ -1]$

For node 2,3, etc... copy W for node 1

What is the hidden layer doing?

Informally, looking for a spike everywhere

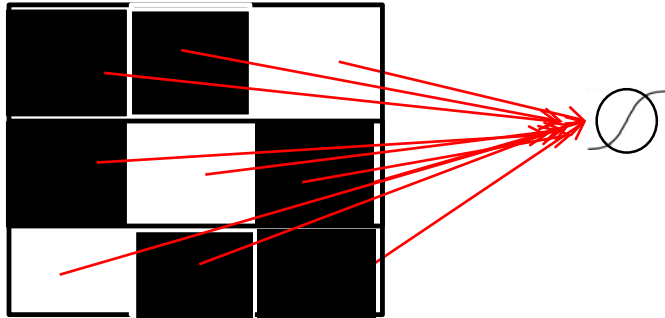
This is essentially a convolution operator, where W is the kernel.

Note: sharing weights is like *sliding W* across input

Note: if we take max activation across nodes ('Max Pool') then it's like looking for a spike *anywhere*.

2D Convolution

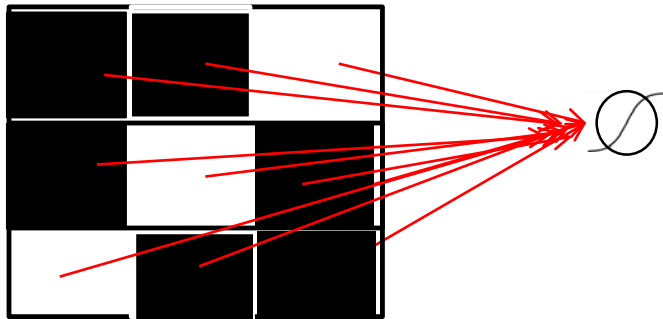
Now let input be a 2D binary matrix, e.g. a binary image) fully connected to 1 node



What W matrix would 'activate' for a upward-toward-left diagonal line?

2D Convolution

Now let input be a 2D binarized 3x3 matrix fully connected to 1 node



(black= -1 white=1)

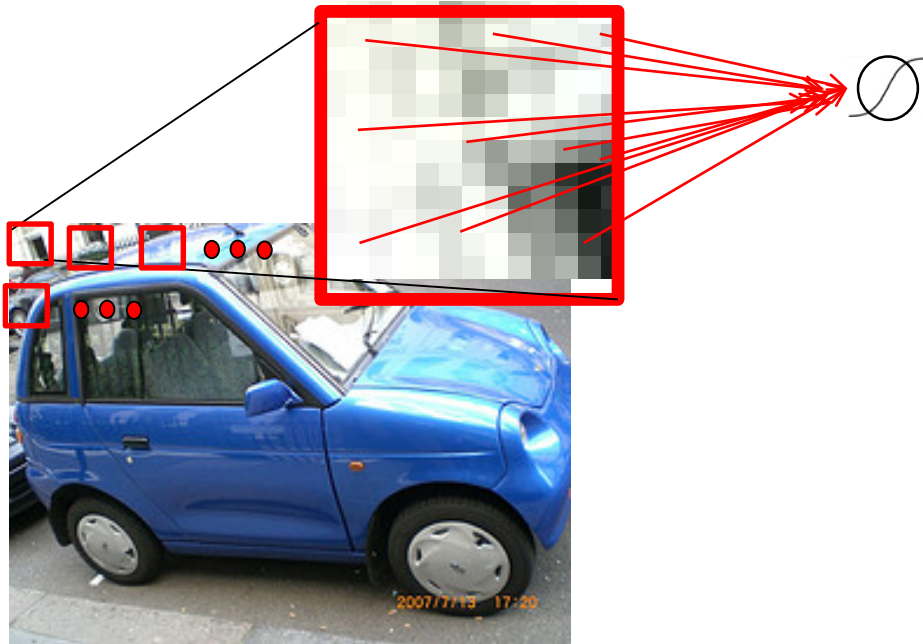
What W matrix would 'activate' for a upward-toward-left diagonal line?

How about:

$$W = \begin{bmatrix} -1 & -1 & 1 \\ -1 & 1 & -1 \\ 1 & -1 & -1 \end{bmatrix}$$

2D Convolution

For full image, 1 filter is applied to 1 region in 1 color channel at a time, and then slid across regions (or done in parallel with shared weights) and produces 1 new 2D image (hidden) layer



Convolution Layer parameters:

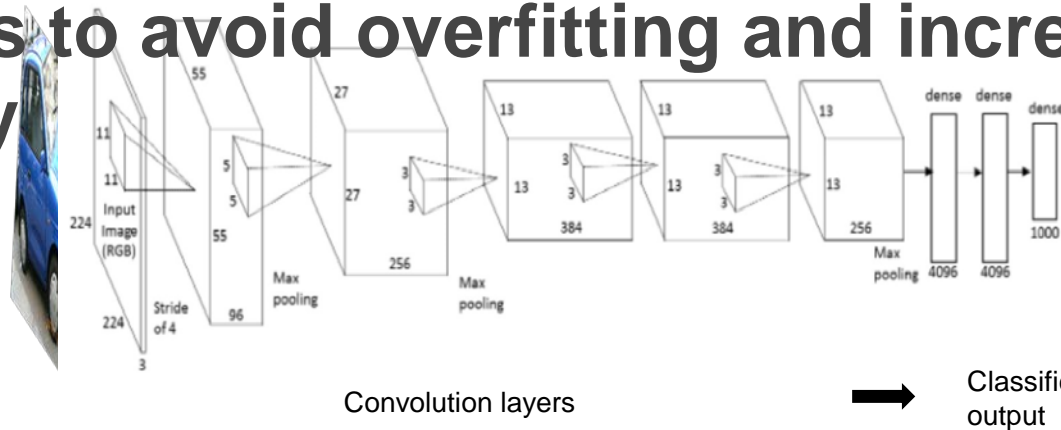
- filter size depends on input:
smaller filters for smaller details
2 layers of 3x3 ~ 1 layer of 5x5
- sliding amount
smaller better but less efficient
- number of filters
depends on task
each filter is a new 2D layer

Convolution Network :

many layers and architecture options

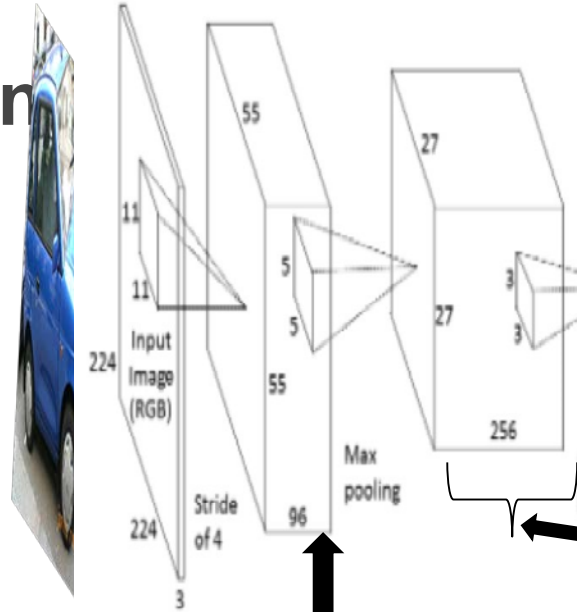
Large Scale Versions

- Large (deep) Convolution Networks are turning out to be feasible with GPUs (some are 100+ layers)
- Need large amounts of data and many heuristics to avoid overfitting and increase efficiency



Large Scale Versions

- Zooming in

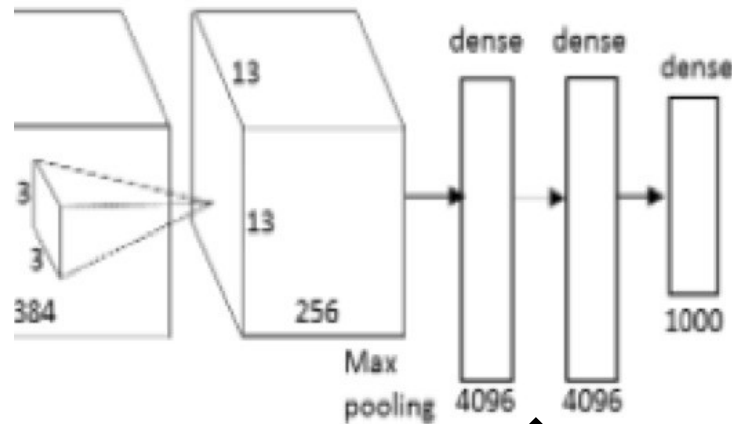


The thickness is the number of different convolutions, i.e. different transformations, sometimes called "channels"

Each convolution layer uses RELU (rectified linear activation units instead of logistic function) and is followed by Max Pooling layer (over 2D regions with sliding)

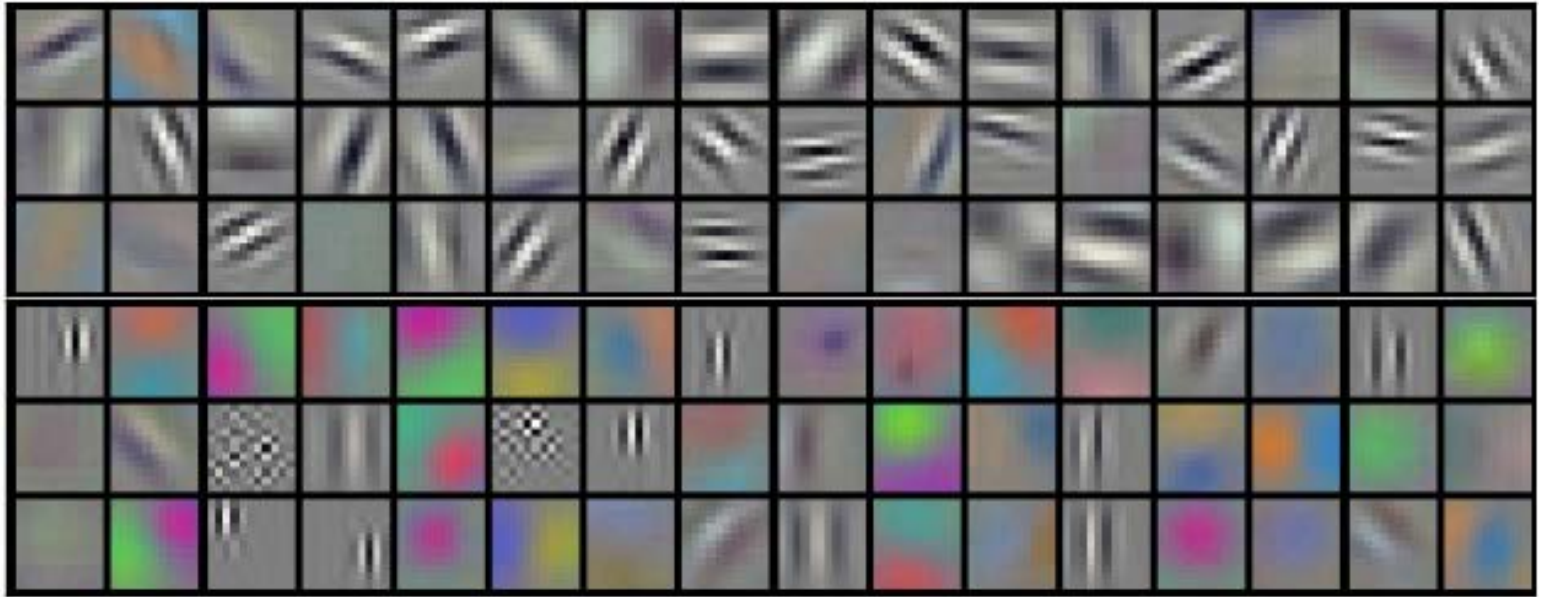
Large Scale Versions

- **Zooming in:**



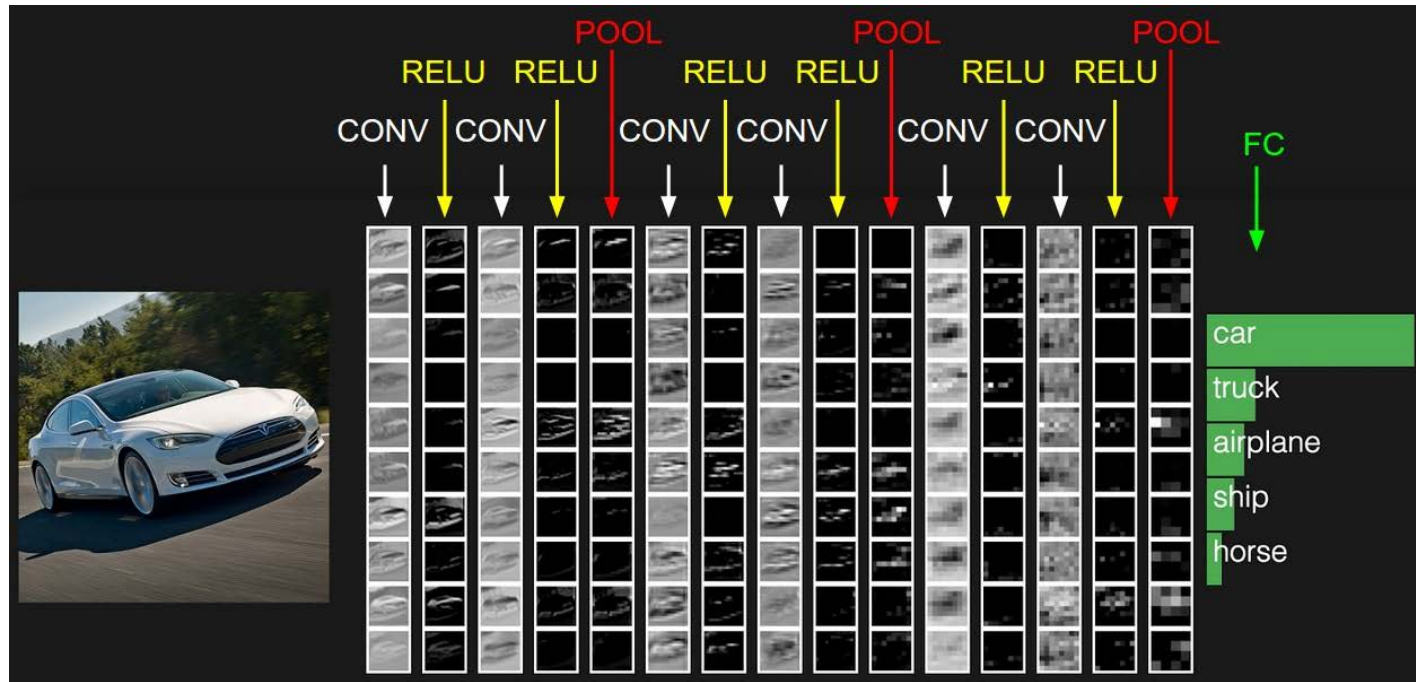
Last convolution layer is laid out as a vector for input into classification layers.
Classification uses dense, i.e. fully connected, hidden layers and output layer.

What Learned Convolutions Look Like



Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks." Advances in neural information processing systems. 2012.

What Learned Convolutions Look Like



Summarizing Deep Layers

- **Hidden layers transform input into new features:**
 - Feature can be highly nonlinear
 - Features as a new space of input data
 - Features as projection onto lower dimensions (compression)
 - Features as filters, which can be used for convolution
- **But also:**
 - Many algorithm parameters
 - Many weight parameters
 - Many options for stacking layers

Feature Coding vs Discovery

- **Edge detection with Support Vector Machine
OR
Convolution Neural Network?**
- **With small datasets and obvious features, SVMs can work well**
- **But building features is hard, and large classification problems can benefit from common features, so CNNs are better to discover features for multiclass outputs**

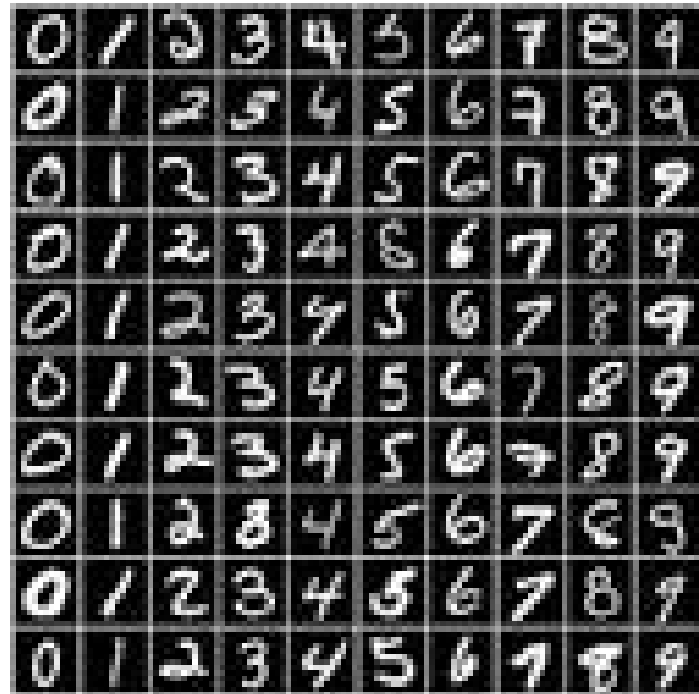
References

- **Book:** <https://mitpress.mit.edu/books/deep-learning>
- **Documentation:** <https://keras.io/>
- **Tutorials I used (borrowed):**
 - <http://cs231n.github.io/convolutional-networks/>
 - <https://hackernoon.com/visualizing-parts-of-convolutional-neural-networks-using-keras-and-cats-5cc01b214e59>
 - https://github.com/julienr/ipynb_playground/blob/master/keras/convmnist/keras_cnn_mnist.ipynb

- **pause**

Tutorial

- MNIST database of handwritten printed digits
- The 'hello world' of Conv. Neural Networks
- Use Keras front end (high level neural functions) to Tensorflow engine (neural math operations)
- Works with GPU or CPUs

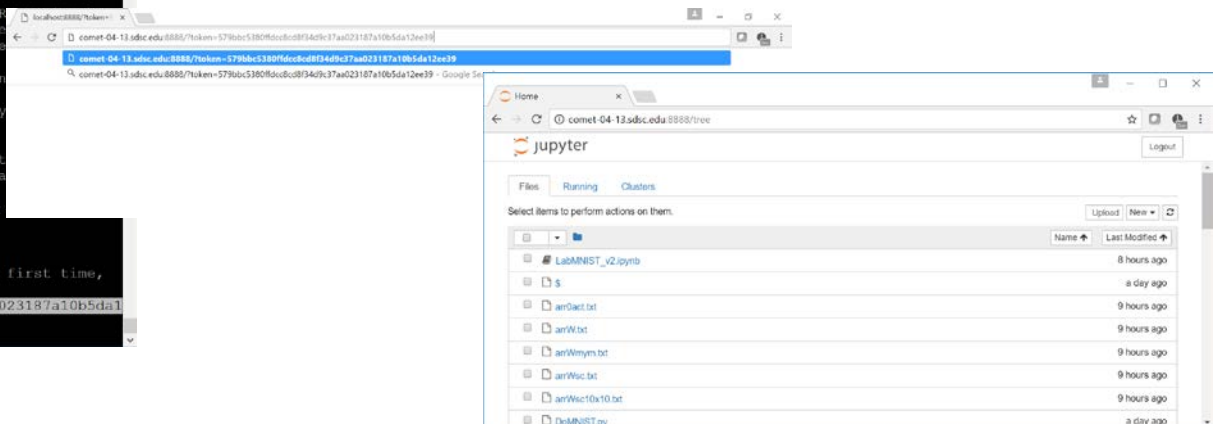


1. Login to comet
2. Access compute node: `srun --partition=debug --pty --nodes=1 --ntasks-per-node=24 -t 00:30:00 --wait=0 --export=ALL -A your-account /bin/bash`
3. Start singularity shell
 1. `module load singularity`
 2. `IMAGE=/oasis/scratch/comet/zonca/temp_project/datascience-notebook-e1677043235c_fixjulia_keras_tf.img`
 3. `singularity exec $IMAGE jupyter notebook --ip=*`
4. on local machine, in browser url edit box, enter the http string shown, but replace localhost with comet-XX-XX.sdsc.edu
5. Open R-introHPC.ipynb or LabMNIST_Final.ipynb
6. After logging out in browser shutdown notebook on Comet with Ctrl-C

```

[Screen 0: bash] p4rodrig@comet-04-13/oasis/scratch/comet/p4rodrig/temp_project
Singularity.sdsd ubuntu tfl1.keras.img> $ ipython notebook --no-browser --ip="*"
# 6
Singularity.sdsd ubuntu tfl1.keras.img> $ [TerminalPythonApp] WARNING: The notebook server is listening on all IP addresses and not using encryption. This is not recommended.
[TerminalPythonApp] WARNING: You likely want to use "jupyter notebook" instead of "ipython notebook".
[W 04:31:57.821 NotebookApp] WARNING: The notebook server is listening on all IP addresses and not using encryption. This is not recommended.
[I 04:31:57.831 NotebookApp] Serving notebooks from local directory: /oasis/scratch/comet/p4rodrig/temp_project/S12017/KerasTest
[I 04:31:57.831 NotebookApp] 0 active kernels
[I 04:31:57.831 NotebookApp] The Jupyter Notebook is running at: http://localhost:8888/?token=579bbc5380ffdc8cd8f34d9c37aa023187a10b5da12ee39
[Copy/paste this URL into your browser when you connect for the first time, to login with a token: http://localhost:8888/?token=579bbc5380ffdc8cd8f34d9c37aa023187a10b5da12ee39]
[C 04:31:57.851 NotebookApp]

```



Home x LabMNIST_Final x

comet-18-14.sdsc.edu:8888/notebooks/LabMNIST_Final.ipynb

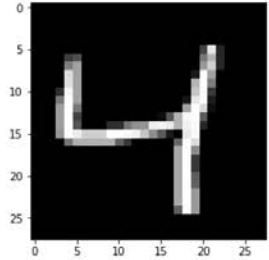
jupyter LabMNIST_Final Last Checkpoint: 7 minutes ago (autosaved) Logout

File Edit View Insert Cell Kernel Help Not Trusted Python 3

```
for i in range(0,3):  
    im = Image.fromarray(X_train[i,:,:])  
    im.save("Xtrain_num"+str(i)+"_cat_"+str(Y_train[i])+".jpeg")  
  
plt.figure()  
plt.imshow(im,'gray')  
plt.show()  
  
print('img load done')  
print (time.strftime("%H:%M:%S"))
```

(5000, 28, 28)

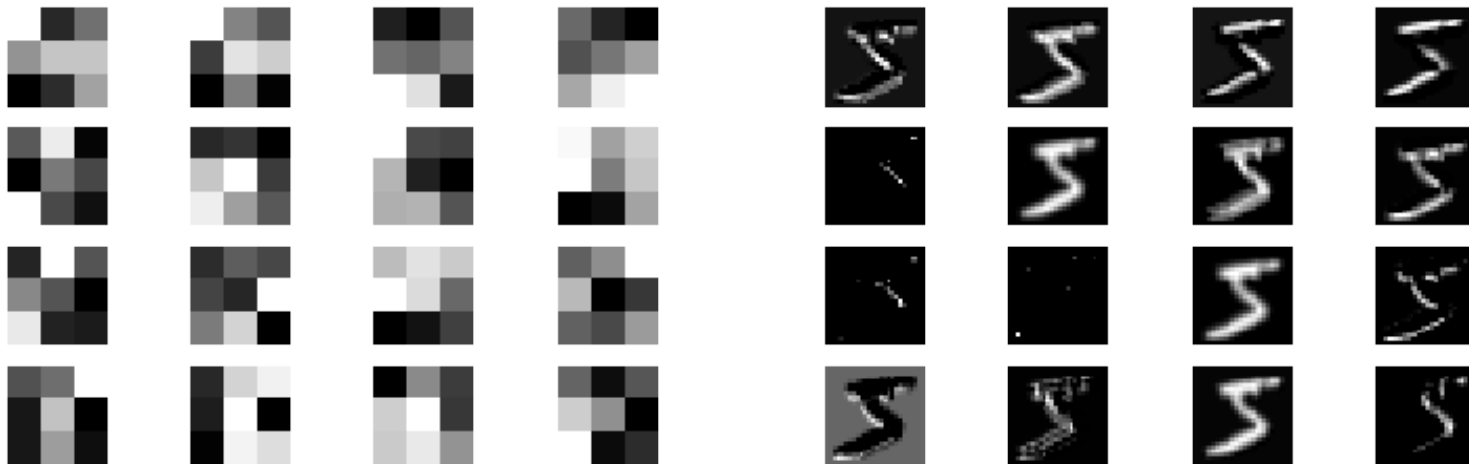
<matplotlib.figure.Figure at 0x2ad45d04f2e8>



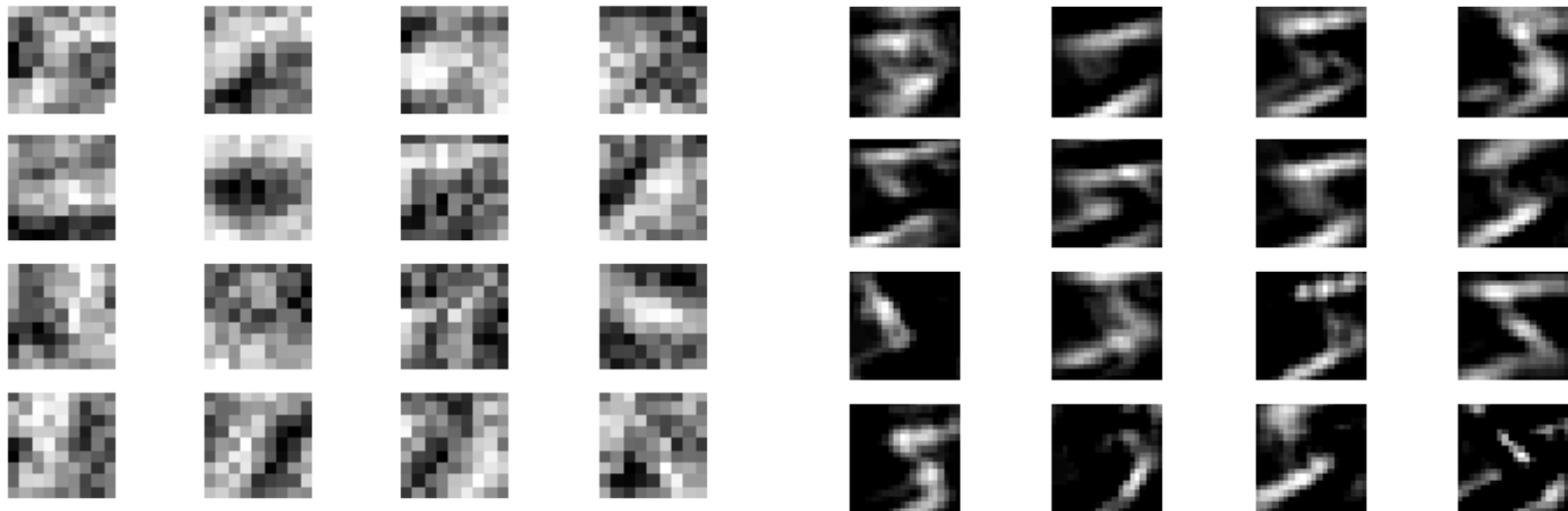
img load done

Windows taskbar: 10:09 PM 7/31/2017

3x3 first convolution layer filter and activation



9x9 first convolution layer filter and activation



Check out: <http://scs.ryerson.ca/~aharley/vis/conv/flat.html>

Some other CNN practices

- Google Vision api – object recognition network



groundbreaking.jpg

White	94%
Black And White	93%
Photograph	93%
Black	93%
People	88%
Monochrome Photography	78%
Monochrome	78%
Musician	75%

Faces

Labels

Web

Properties

Safe Search

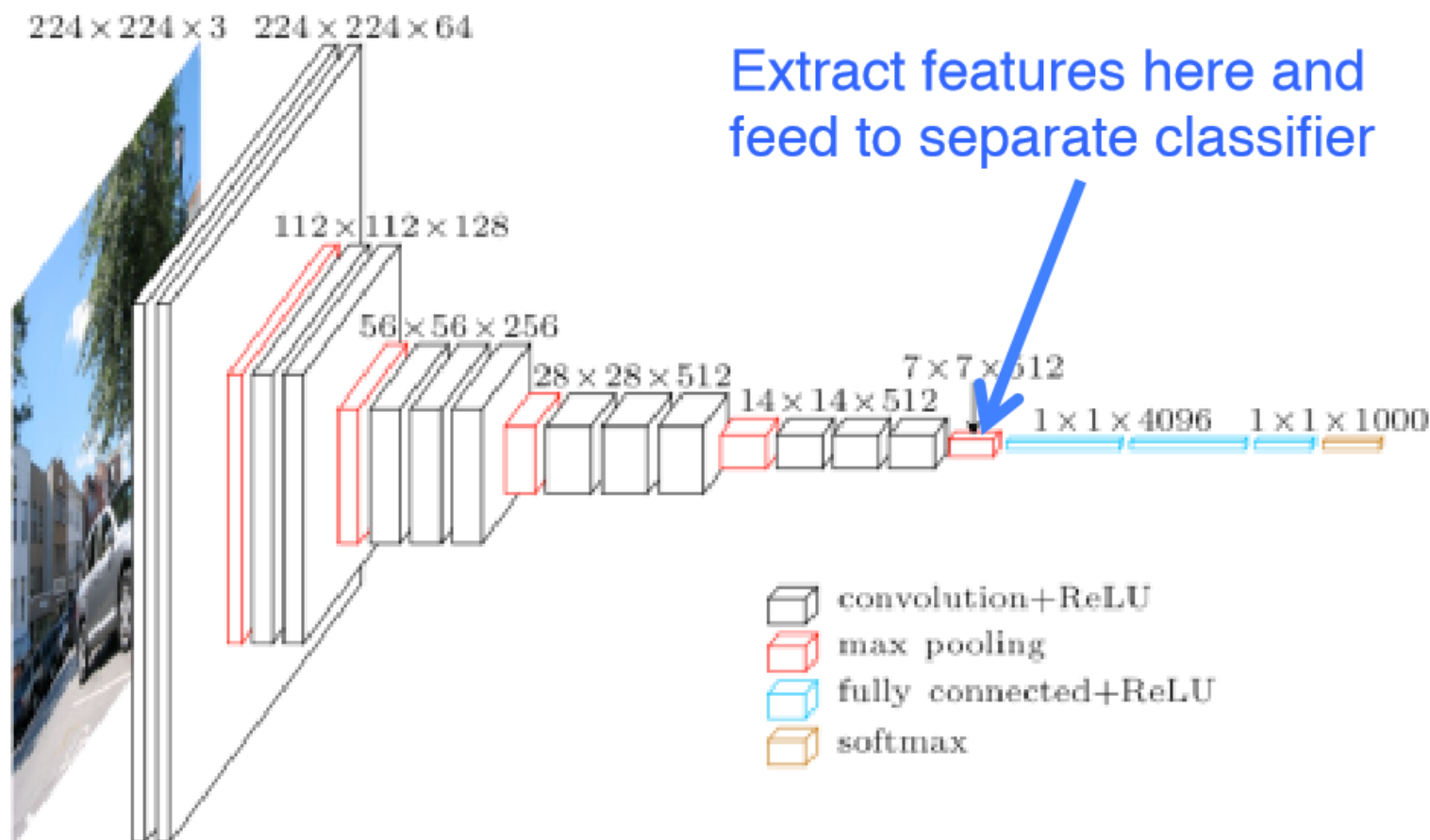
JSON



fsa1997023652#soldier_81;military_62;recreation_59.jpg

Black And White	93%
Soldier	86%
Monochrome Photography	84%
Photography	82%
Monochrome	72%
Military	69%
Recreation	58%
Stock Photography	58%
Grass	52%

Transfer Learning – Feature Extraction



Source: <https://www.cs.toronto.edu/~frossard/post/vgg16/>

Learning Segmentation (deconvolve)

