Machine Learning Intro with a Bias-Variance discussion

James G. Shanahan[2]

[1]Church and Duncan Group, [2]iSchool UC Berkeley, CA,

EMAIL: James_DOT_Shanahan_AT_gmail_DOT_com

Lecture #2

June 13, 2016

Live Session Outline

- **Machine Learning Introduction**
- **Equation of a line**
- **Linear Regresion**
- **Bias-Variance**
  - Expected Value, Variance
  - Bias-Variance
  - Coding it up
    - For a simulated world (lab setting)
    - For a real world problems
  - Example using bagging

Traditionally we coded up systems

- **Code for storing data, reading data**
- **Making decisions (e.g., give a loan or not)**
- **Nested IF Statements**
  - Written many years ago
  - Tweaked every so often
- **Prefer a data driven approach**

Machine Learning in one slide

- **Machine learning, a branch of artificial intelligence, is a scientific discipline that is concerned with the design and development of algorithms that allow computers to evolve behaviors based on empirical data, such as from sensor data or databases.**
- **A learner can take advantage of examples (data) to capture characteristics of interest of their unknown underlying probability distribution. Data can be seen as examples that illustrate relations between observed variables.**
- **A major focus of machine learning research is to automatically learn to recognize complex patterns and make intelligent decisions based on data; the difficulty lies in the fact that the set of all possible behaviors given all possible inputs is too large to be covered by the set of observed examples (training data).**
- **Hence the learner must generalize from the given examples, so as to be able to produce a useful output in new cases. Machine learning, like all subjects in artificial intelligence, require cross-disciplinary proficiency in several areas, such as probability theory, statistics, pattern recognition, cognitive science, data mining, adaptive control, computational neuroscience and theoretical computer science.**

[WikiPedia]

What is the Learning Problem?

Learning = Improving with experience at some task

- **Improve over Task T**
- **with respect to performance measure P**
- **based on experience E**

Types of Learning

- Supervised learning - Generates a function that maps inputs to desired outputs. For example, in a classification problem, the learner approximates a function mapping a vector into classes by looking at input-output examples of the function.
- Unsupervised learning - Models a set of inputs: like clustering
- Semi-supervised learning - Combines both labeled and unlabeled examples to generate an appropriate function or classifier.
- Reinforcement learning - Learns how to act given an observation of the world. Every action has some impact in the environment, and the environment provides feedback in the form of rewards that guides the learning algorithm.
  Transduction - Tries to predict new outputs based on training inputs, training outputs, and test inputs.

Supervised Learning :Regression

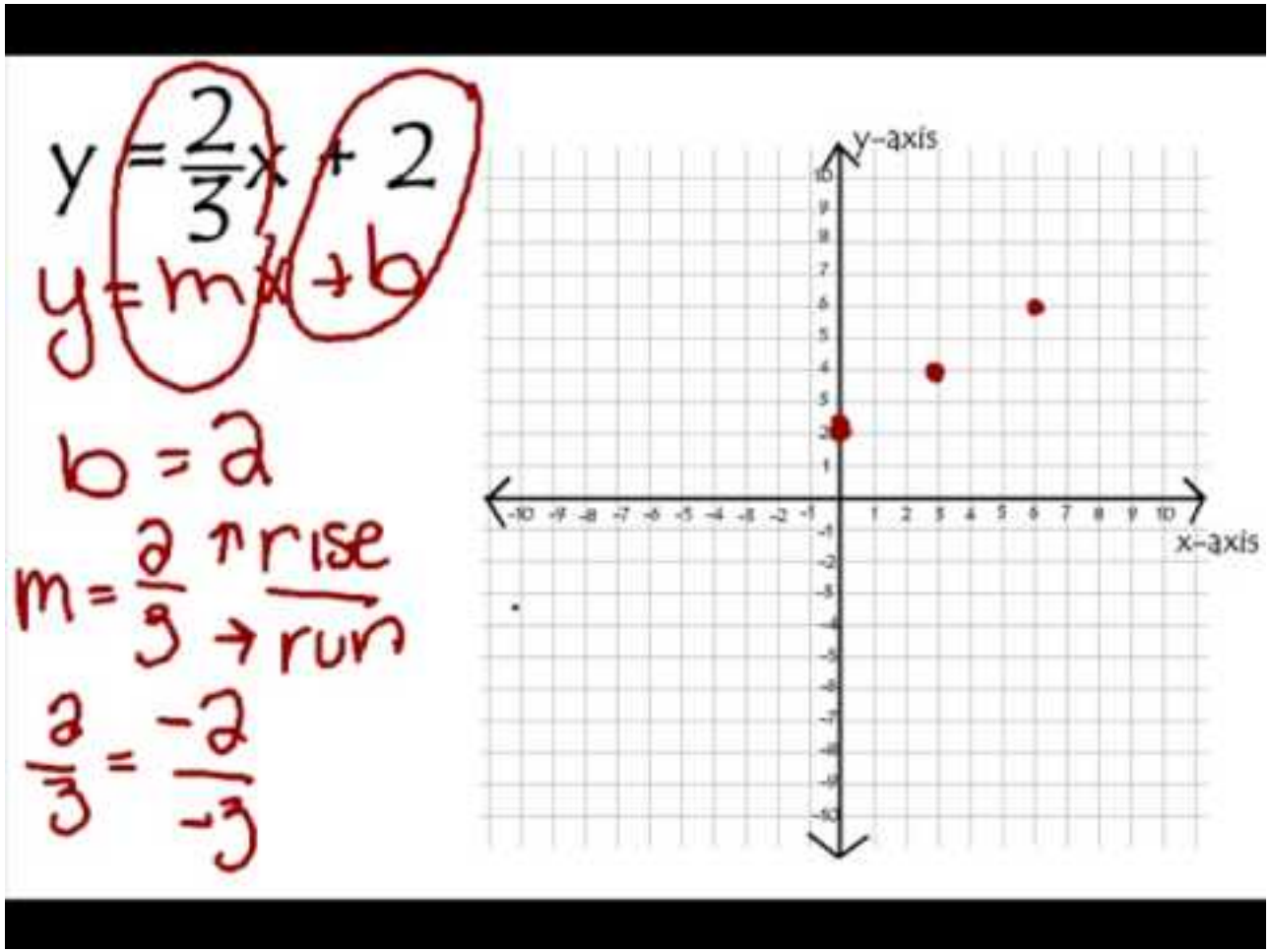
- **Regression**
  - Linear Regression
- **Classification**
  - Logistic Regression
- **Generalized Linear Models (GLMs)**
  - Broader family of models (that subsume Linear Regression and logistic regress and more
  - In R checkout ?glm()

Parametric Approaches vs. Non-parametric
Convex/Concave
Discriminative versus generative

Live Session Outline

- **Machine Learning Introduction**
- **Equation of a line**
- **Linear Regresion**
- **Bias-Variance**
  - Expected Value, Variance
  - Bias-Variance
  - Coding it up
    - For a simulated world (lab setting)
    - For a real world problems
  - Example using bagging

- ..


y = 2/3x + 2
y = mx + b
b = 2
m = 2/3 ↑ rise
3 → run
2/3 = -2/-3
y-axis
x-axis

Live Session Outline

- **Machine Learning Introduction**
- **Equation of a line**
- **Linear Regresion**
- **Bias-Variance**
  - Expected Value, Variance
  - Bias-Variance
  - Coding it up
    - For a simulated world (lab setting)
    - For a real world problems
  - Example using bagging

Terminology: linear regression

W_i are the model coefficients

$$y = W_0 + W_1X_1 + W_2X_2 + \ldots + W_nX_n$$

y

Predicted

Response variable

Outcome variable

Dependent

X_i's

Predictor variables

Explanatory variables

Covariables

Independent variables

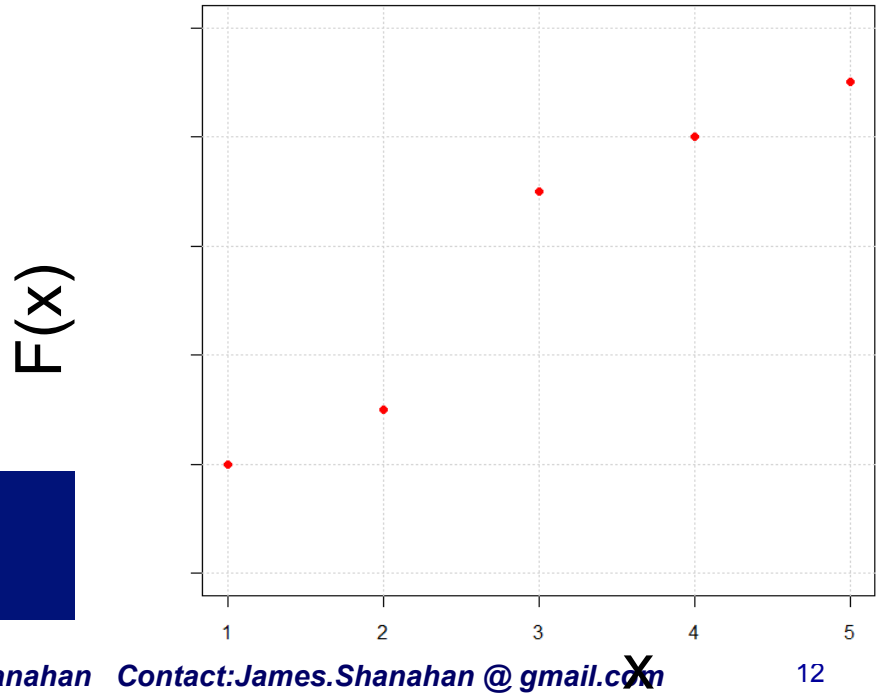
Y-intercept/threshold

Pr(Click): Advertising Problem

- **Predict Pr(Click|dwellTimeOnWebpage)**
  - at the times 1, 2, 3, 4, and 5 seconds after loading the page.
- **Graph each data point with time on the *x*-axis and CTR on the *y*-axis. Your data should follow a straight line.**
- **Use locator() to input data**
- **Find the equation of this line.**

#	x	y%
1	1	2
.	2	3
.	3	7
.	4	8
m	5	9

Temperature Dataset



X are features, aka variables, continuous, discrete, ordinal ($X \varepsilon \Re^n$)

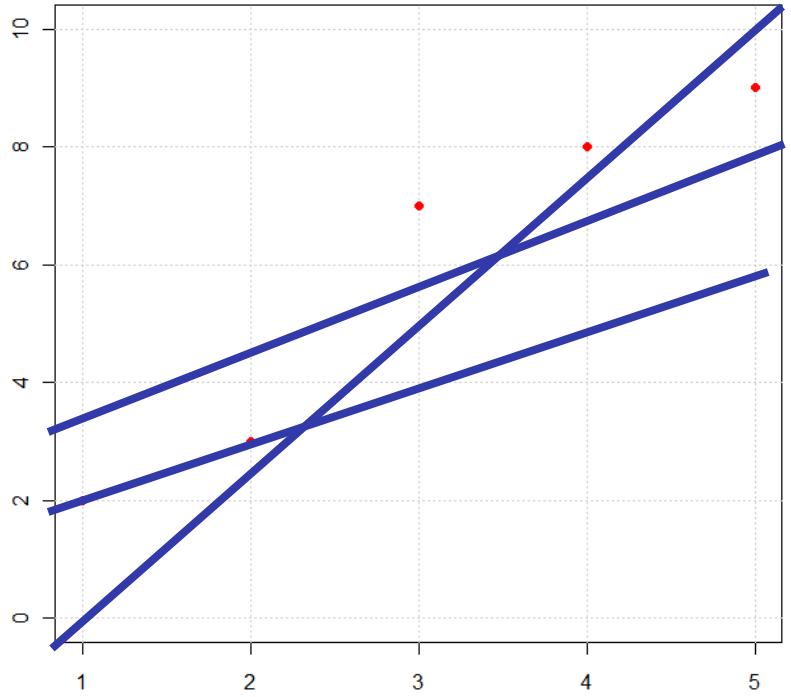
Generate Your Own Data

```
# You can generate data by clicking on a plot.
# Create data that illustrates the effect of varying 'f' and 'iter' in 'lowess'.
example.generateYourOwnData = function(){

  #change range of X and Y and experiment
  plot( c(0,1), c(0,1), type = 'n')
  xy <- locator(type = "p")      # create your own data set by clicking on the
                                 # left mouse button, then with the right mouse button
                                 # to finish. With a Macintosh cntrl-click outside the
                                 # plot to finish.
data1=data.frame(x=xy$x, y=xy$y)   #PLOT LINE
abline(coef(lm(y~x, data=data1)), col="red")

lines(lowess(xy, f = 2/3, iter = 3))  # here I've used the defaults
                                      # for f and iter,
                                      # experiment with other values
```

Least Square Fit Approximations

Suppose we want to fit the data set.

#	x	y
1	1	2
.	2	3
.	3	7
.	4	8
m	5	9

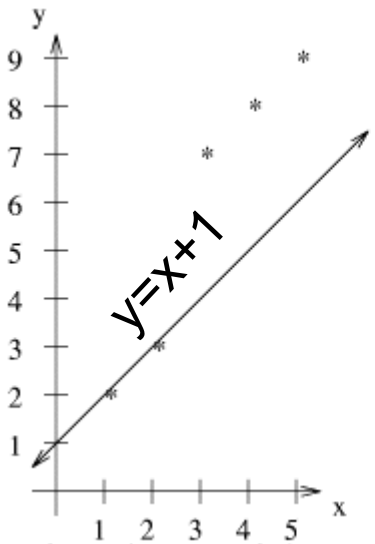


We would like to find the best straight line to fit the data?

Fit a line based on…

- **If we assume that the first two points are correct and choose the line that goes through them, we get the line *y* = 1 + *x*.**
- **If we substitute our points (x-values) into this equation, we get the following chart.**
- **How good is this line?**
  - The sum of the squares of the errors is 27.

$$y = mx + b$$
$$m = \frac{y_2 - y_1}{x_2 - x_1}.$$



x	y	predicted y	error	$(\text{error})^2$
1	2	2	0	0
2	3	3	0	0
3	7	4	3	9
4	8	5	3	9
5	9	6	3	9

SSE = 27

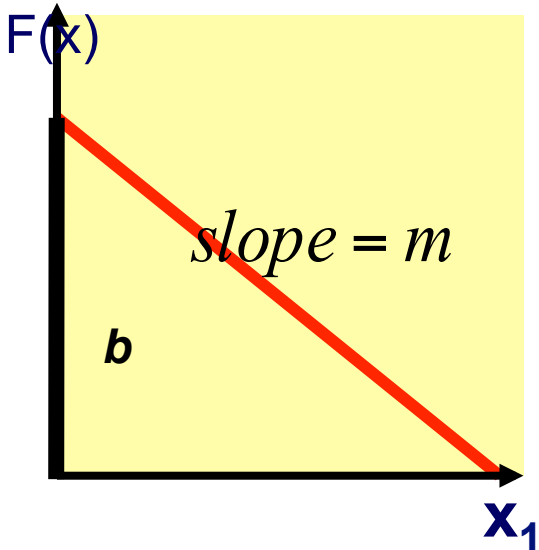
Do you think that we can do better than this?

Linear Model More Generally

- **E.g., y=mx+b can be more generally seen a function of the form**
- **Here the W's are the parameters (also called weights) parametering the space of linear function mapping from X → Y=F(x)**

$$y = f(x_0, x_1) = w_0 x_0 + w_1 x_1$$

$$= \sum_{i=1}^{n} w_i x_i = W^T X$$



Sometimes use θ instead of W

$$y = f(x_0, x_1) = \sum_{i=1}^{n} \theta_i x_i = \theta^T X$$

#	X0	x1	y
1	1	1	2
.	1	2	3
.	1	3	7
.	1	4	8
m	1	5	9

Linear Model: Ordinary Least Squares

Measuring Quality

- **How do we pick, or learn, the parameters W (aka θ)?**
- **One reasonable method seems to be to make f(x) close to y, at least for the training examples.**
- **To formalize, let's define a function that measures, for each possible model/hypothesis, W, how close $f_\theta(x^i)$'s are to the corresponding y^i 's:**

$$J(W) = \sum_{i=1}^{m} \left| WX^i - y^i \right|$$

This error minimization is going to have problems?

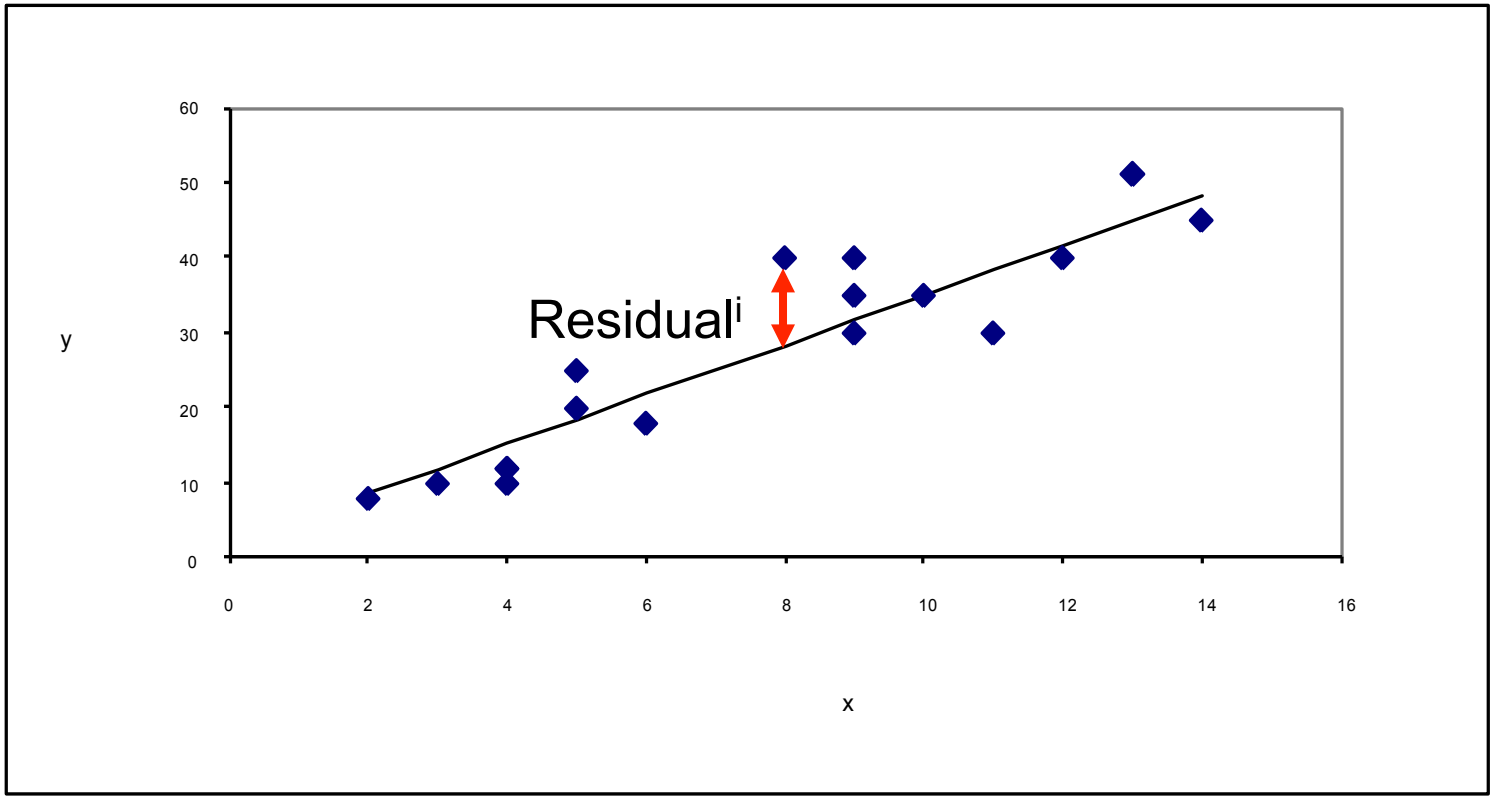
$$J(W) = \frac{1}{2} \sum_{i=1}^{m} \left(WX^i - y^i \right)^2$$

Residual sum of squares

- **Sum of squared error**
- **AKA Residual Sum of Squares (Residual squared)**

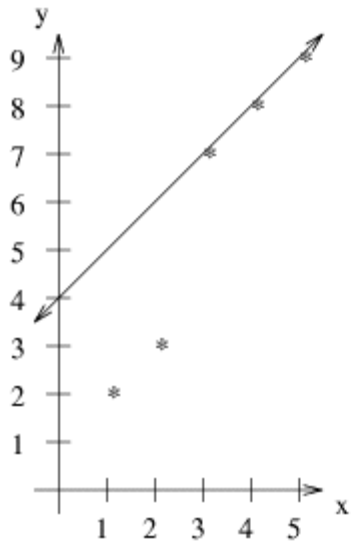
Residual

$$\text{Residual}^{\text{i}} = \left(WX^{i} - y^{i}\right)$$



Which Line is it anyway?

- **Select another two points and build a line**
- **If we choose the line that goes through the points when $x = 3$ and 4, we get the line $y = 4 + x$. Will we get a better fit? Let's look at it.**



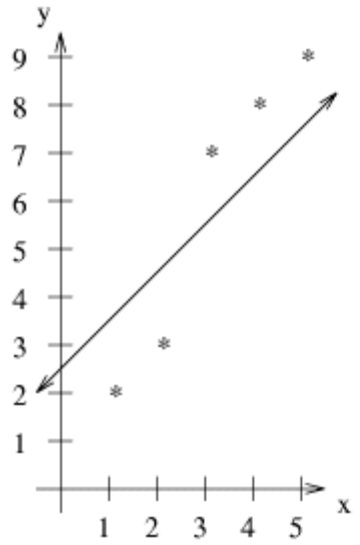
x	y	predicted y	error	$(\text{error})^2$
1	2	5	−3	9
2	3	6	−3	9
3	7	7	0	0
4	8	8	0	0
5	9	9	0	0

SSE = 18. Getting better but can we do better?

Can we do better than guesswork?

- **Let's try the line that is half way between these two lines. The equation would be *y* = 2.5 + *x*.**

- **Is there a more scientific or efficient way than guessing at which line would give the best fit.**
  - Surely there is a methodical way to determine the best fit line. Let's think about what we want.



x	y	predicted y	error	$(\text{error})^2$
1	2	3.5	−1.5	2.25
2	3	4.5	−1.5	2.25
3	7	5.5	1.5	2.25
4	8	6.5	1.5	2.25
5	9	7.5	1.5	2.25

SSE = 11.25. Getting better but can we do better?

Hypothesis Space of Linear Models

- **Here the W's are the parameters (also called weights) parameterizing the space of linear function mapping from X → Y = f(X)**
- **Augment Training Data with dummy intercept variable (simplifies notation and modeling)**

#	X0	x1	y
1	1	1	2
.	1	2	3
.	1	3	7
.	1	4	8
m	1	5	9

$$y = f(x_0, x_1) = w_0 x_0 + w_1 x_1$$

$$= \sum_{i=1}^{n} w_i x_i = W^T X$$

Sometimes use θ instead of W

$$y = f(x_0, x_1) = \sum_{i=1}^{n} \theta_i x_i = \theta^T X$$

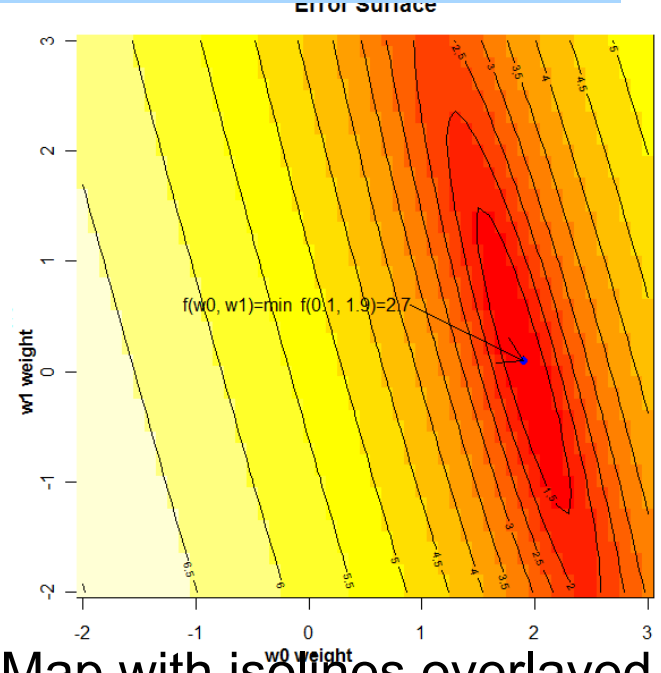
Space of Hypotheses: Weights

- Each model is in our case a coefficient for the y-intercept (bias) and a coefficient for the feature-variable (time)
- Plot weight-space in 2D where the third dimesion is the error

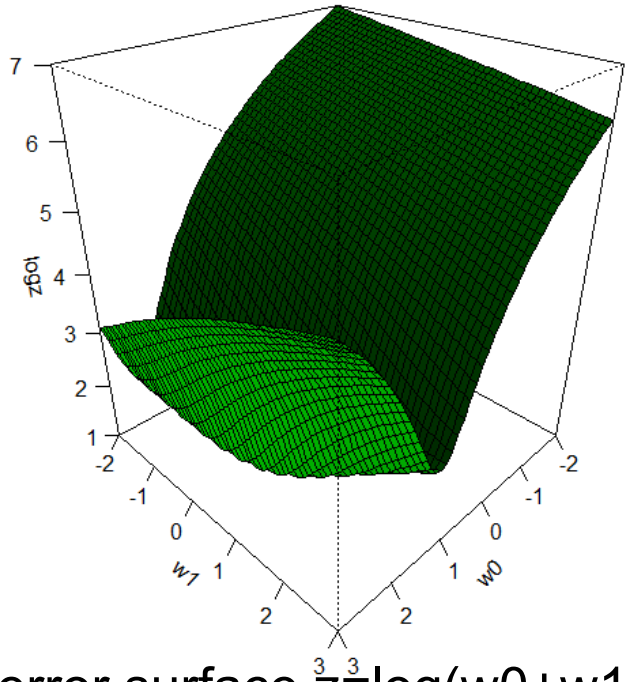
$$J(W) = \frac{1}{2}\sum_{i=1}^{m}\left(W^i X^i - y^i\right)^2$$

- Select combination that minimizes the sum of square error

example.OLS_Heatmap()



HeatMap with isolines overlayed



3D error surface z=log(w0+w1*x)

Find a line that fits all datapoints?

- Recall, a line in slope-intercept form looks like y= $w_0 + w_1 x$ where w_0 is the y -intercept and w_1 is the slope.
- We want to find w_0 and w_1 such that $w_0 + w_1 x_i = y_i$ is true for all our data points:

$$
\begin{aligned}
w_0 &+ 1w_1 = 2 \\
w_0 &+ 2w_1 = 3 \\
w_0 &+ 3w_1 = 7 \\
w_0 &+ 4w_1 = 8 \\
w_0 &+ 5w_1 = 9
\end{aligned}
$$

- We know that there may not exist w_0 and w_1 that fit all these equations, so we try to find the best fit.

Find the best line: Several Approaches

- **Determine w_0, $w_1 \ldots w_n$**

$$y = f(x_0, x_1) = w_0 x_0 + w_1 x_1$$

$$= \sum_{i=1}^{n} w_i x_i = W^T X$$

- **Several Approaches to finding the best-fit line**
  - Select a couple of data points and solve analytically (high variance)
  - Brute-force Search
  - Iterative approaches (via the gradient)
  - Closed Form
  - Probabilistic interpretation/justification via maximum likelihood
  - Bayesian modeling [will be covered in Lecture 4]

Hypothesis Space of Linear Models

- **Here the W's are the parameters (also called weights) parameterizing the space of linear function mapping from X → YF(x)**
- **Augment Training Data with dummy intercept variable (simplifies notation and modeling)**

#	X0	x1	y
1	1	1	2
.	1	2	3
.	1	3	7
.	1	4	8
m	1	5	9

$$y = f(x_0, x_1) = w_0 x_0 + w_1 x_1$$

$$= \sum_{i=1}^{n} w_i x_i = W^T X$$

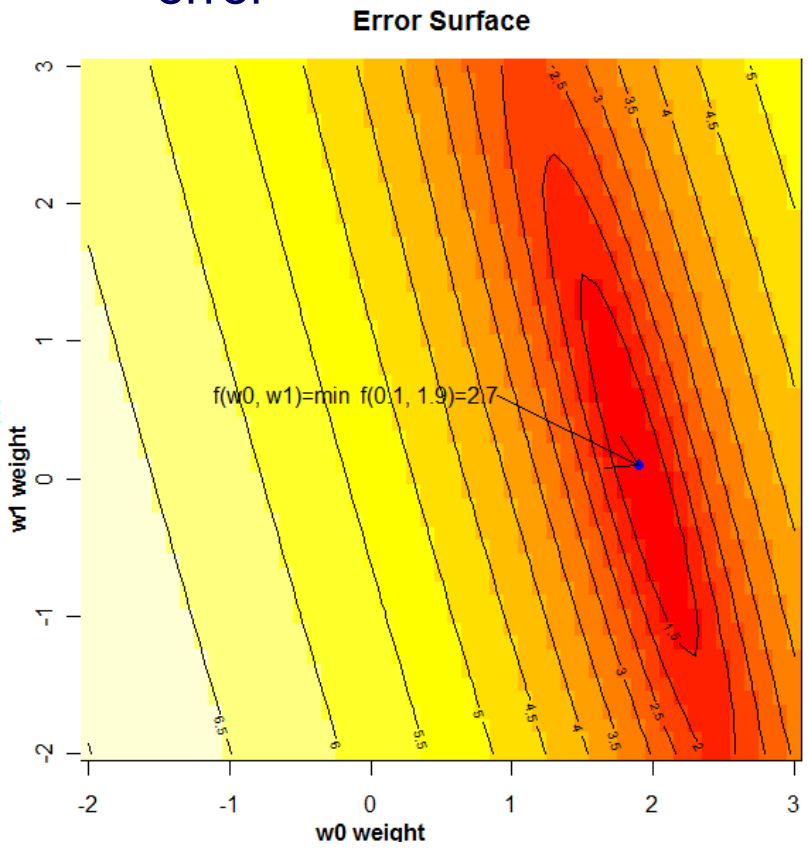
Sometimes use θ instead of W

$$y = f(x_0, x_1) = \sum_{i=1}^{n} \theta_i x_i = \theta^T X$$

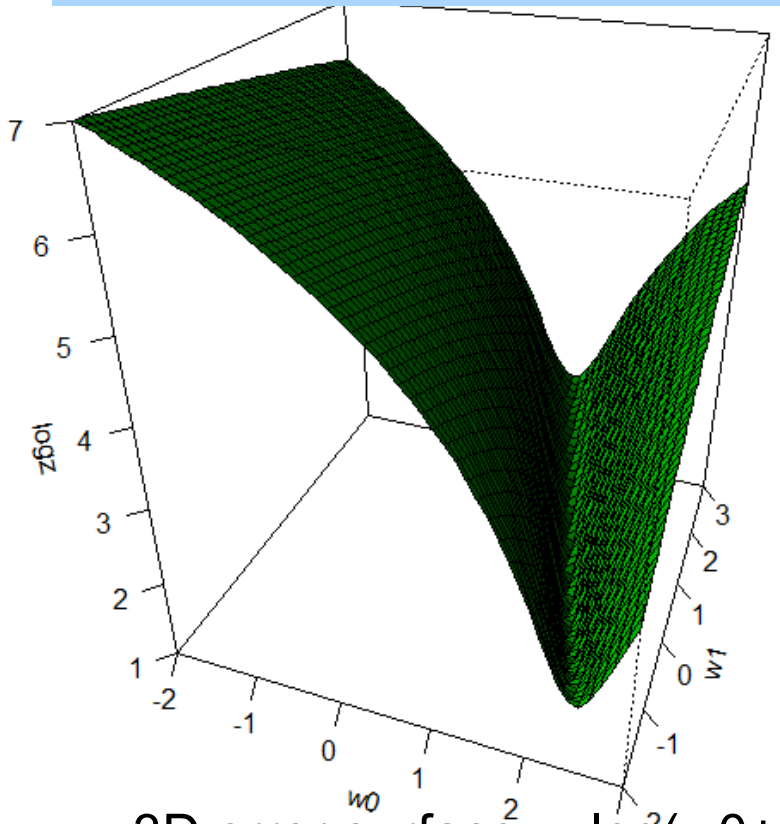
Brute Force Search of Weights

- Enumerate all possible coefficient combinations (in our case coefficient for the y-intercept (bias) and for the c-variable (time)
- Select the weight combination that minimizes the sum of square error

example.OLS_Heatmap()



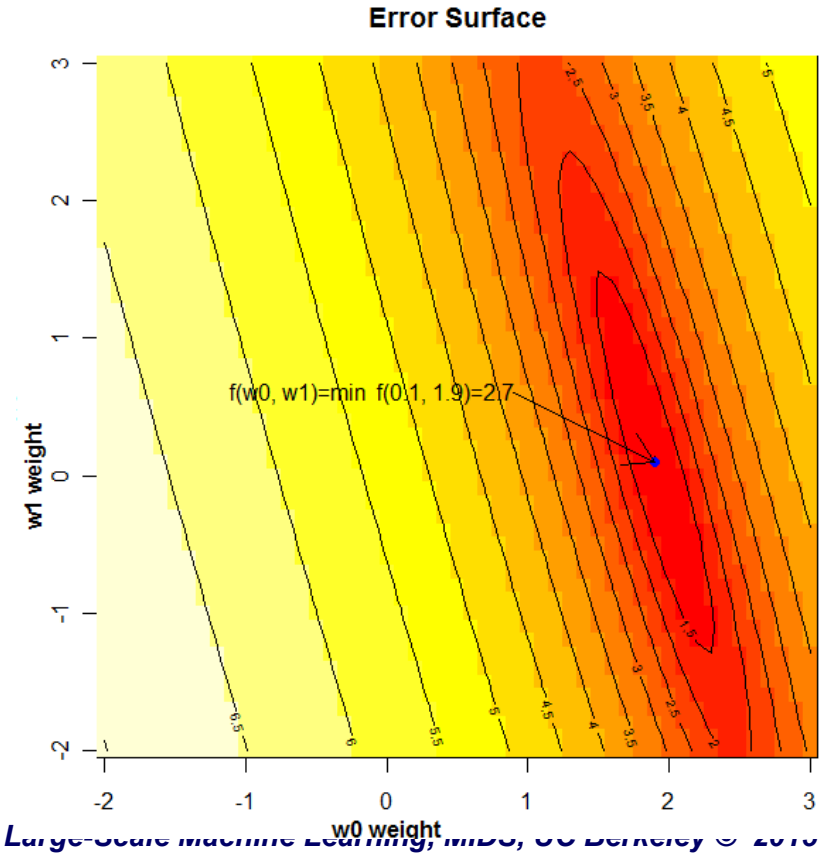
HeatMap with isolines overlayed

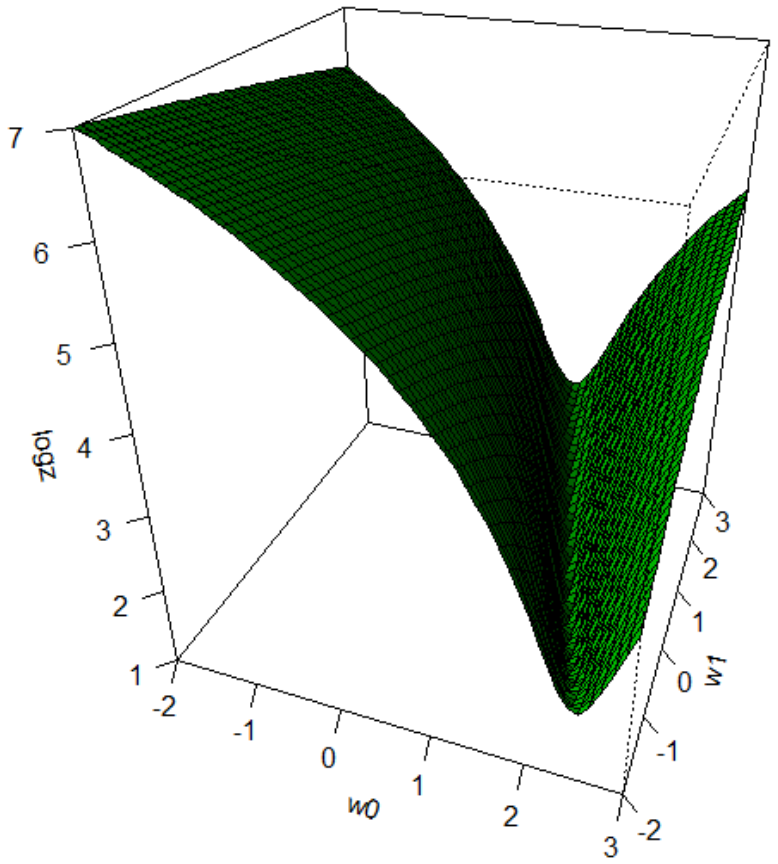


3D error surface z=log(w0+w1*x)

Brute Force Search of Weights

- Very inefficient; at best we can only approximate the surface
- Not scaleable
- Avoid this approach…



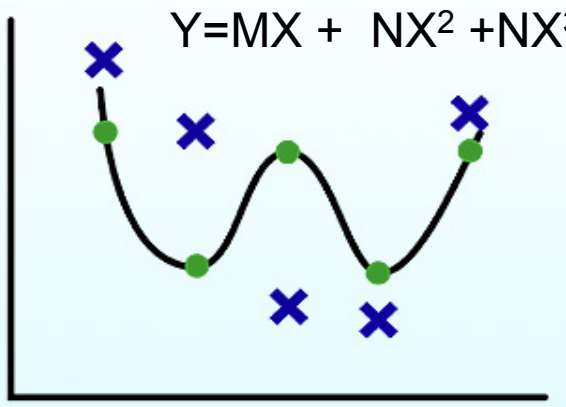


Iterative approach to Learning the Line

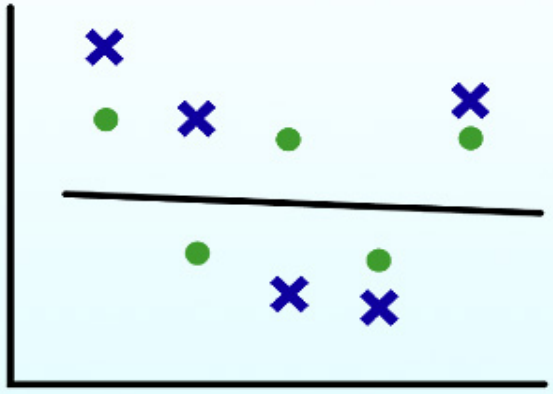
- **Can we navigate the error surface in an efficient manner in the hope of getting to minimum?**
- **Can we leverage other properties of the function? (Hint convexity)**

- **Yes we can!**
  - We can navigate this surface using the gradient (slope)
  - OLS is convex so what [well-behaved function! More about this later this lecture and next lecture]

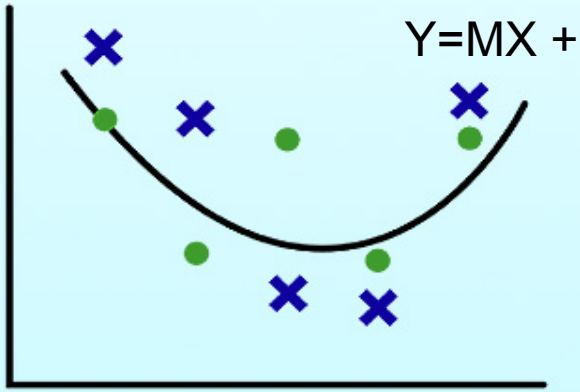
Polynomial regression



(a) High variance/low bias.
4th-order polynomial ($p = 5$).



(b) Low variance/high bias.
1st-order polynomial ($p = 2$).



(c) Balanced variance & bias.
Minimum MSE.
2nd-order polynomial ($p = 3$).

- Data points for fitting
- Typical new data points

Linear (Polynomial) Regression Notebooks

- **R**
  - https://www.dropbox.com/s/v705amixlr1mkvy/LinearRegression.Rmd?dl=0
- **Python**
  - https://github.com/justmarkham/DAT4/blob/master/notebooks/08_linear_regression.ipynb

~/Dropbox/Projects/Target-2016-04/Notebooks/Unit1-Linear-Regression/LinearRegression - RStudio

LinearRegression.Rmd

```
# plot the data and the model
plot(x,y, col=rgb(0.2,0.4,0.6,0.4), main='Linear regression by gradient descent')
abline(res, col='blue')
```

polynomial regression
http://www.r-bloggers.com/fitting-polynomial-regression-in-r/

```{r polynomial, include=TRUE}
fit2 <- lm(y ~ x + I(x^2) + I(x^3))
print(fit2)
# OR
#model <- lm(noisy.y ~ poly(q,3))

pol2 <- function(fit2, x) {
  return(fit2$coefficient[4]*x^3 + fit2$coefficient[3]*x^2 + fit2$coefficient[2]*x + fit2$coefficient[1])
}

lines(x, pol2(fit2, x), col="red", lwd=2)

summary(model.poly)
```

R Markdown

50:1 Chunk 4: polynomial — R Markdown

Console | R Markdown

.../LinearRegression/LinearRegression.Rmd — Output | Issues

```
  |.............................                          |  46%
label: plotmodel (with options)
List of 1
 $ include: logi TRUE

  |.................................                      |  54%
  ordinary text without R code

  |.....................................                  |  62%
label: polynomial (with options)
List of 1
 $ include: logi TRUE


Quitting from lines 46-59 (LinearRegression.Rmd)
Error in plot.xy(xy.coords(x, y), type = type, ...) :
  plot.new has not been called yet
Calls: <Anonymous> ... withVisible -> eval -> eval -> lines -> lines.default -> plot.xy
Execution halted
```

Environment | History

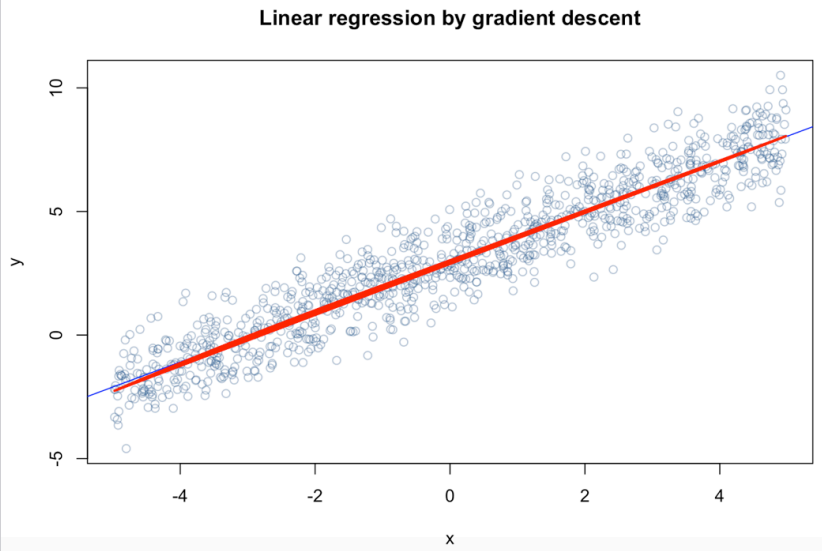
Import Dataset — List

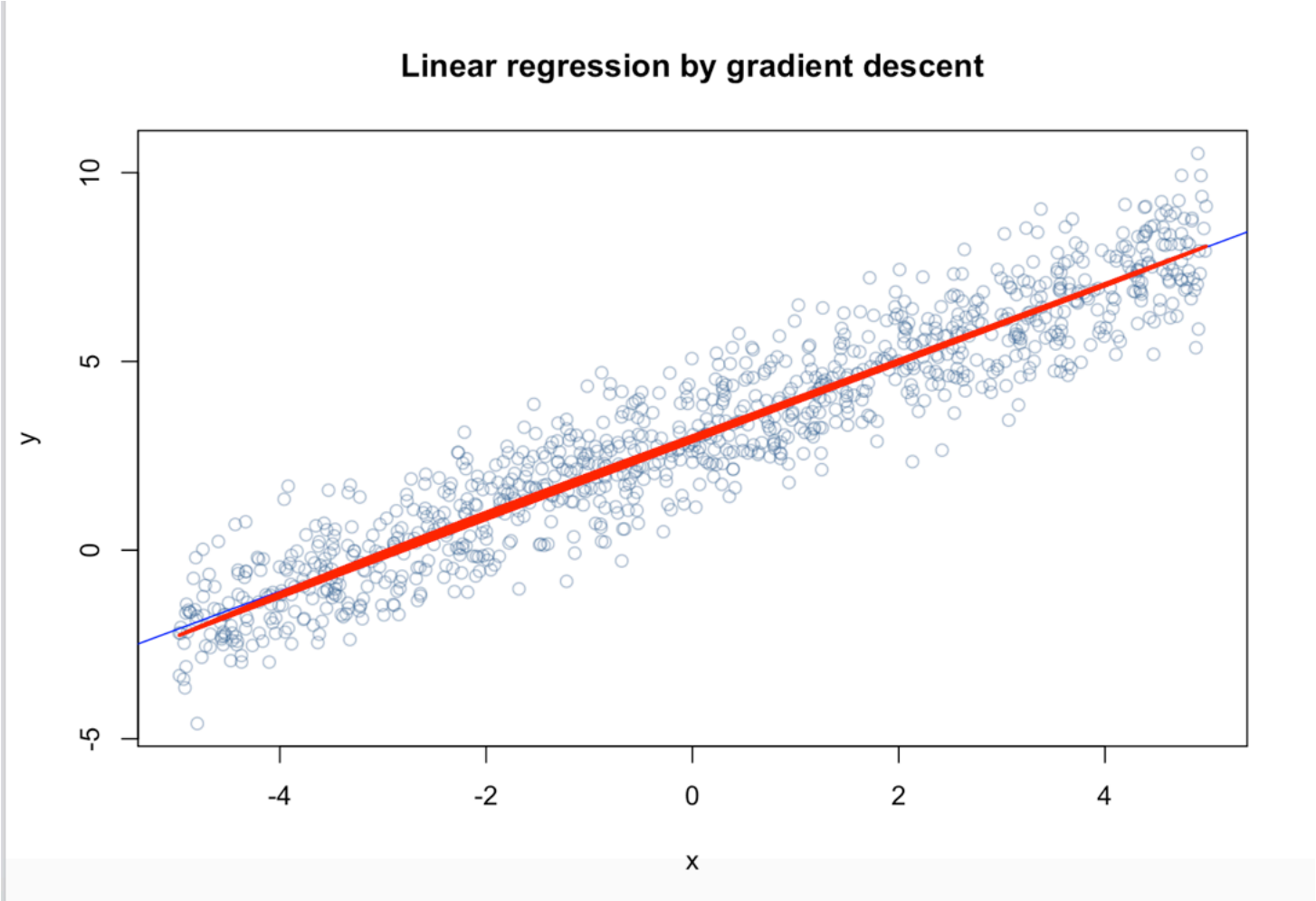
Global Environment

Values	
fit2	List of 12
model.poly	List of 12
res	List of 12
x	num [1:1000] -0.6001 3.9925 0.2543 4.8577 0.0987 ...
y	num [1:1000] 1.9 5.67 4.92 7.2 3.04 ...
Functions	
pol2	function (fit2, x)

Files | Plots | Packages | Help | Viewer

Zoom | Export | Publish




Linear regression by gradient descent
y
x
10
5
0
-5
-4
-2
0
2
4

R

- The S statistical programming language and computing environment has become the de-facto standard among machine learners, statisticians, operation research (kitchen sink, gateway).
- The S language has two implementations: the commercial product S-PLUS, and the free, open-source R.
- Both are available for Windows and Unix/Linux systems; R, in addition, runs on Macintoshes.
- This lecture series will use R.

R: A History (from 1997 –

- In computing, R is a programming language and software environment for general purpose statistical and analytics computing and graphics.
- It is an implementation of the S programming language with lexical scoping semantics inspired by Scheme.
- R was created by Ross Ihaka and Robert Gentleman[2] at the University of Auckland, New Zealand, and is now developed by the *R Development Core Team*.
- It is named partly after the first names of the first two R authors (Robert Gentleman and Ross Ihaka), and partly as a play on the name of S.[3]
- The R language has become a de facto standard among statisticians/engineers for the development of statistical and engineering software, and is widely used for statistical software development and data analysis.

[Wikipedia]

Scripting languages

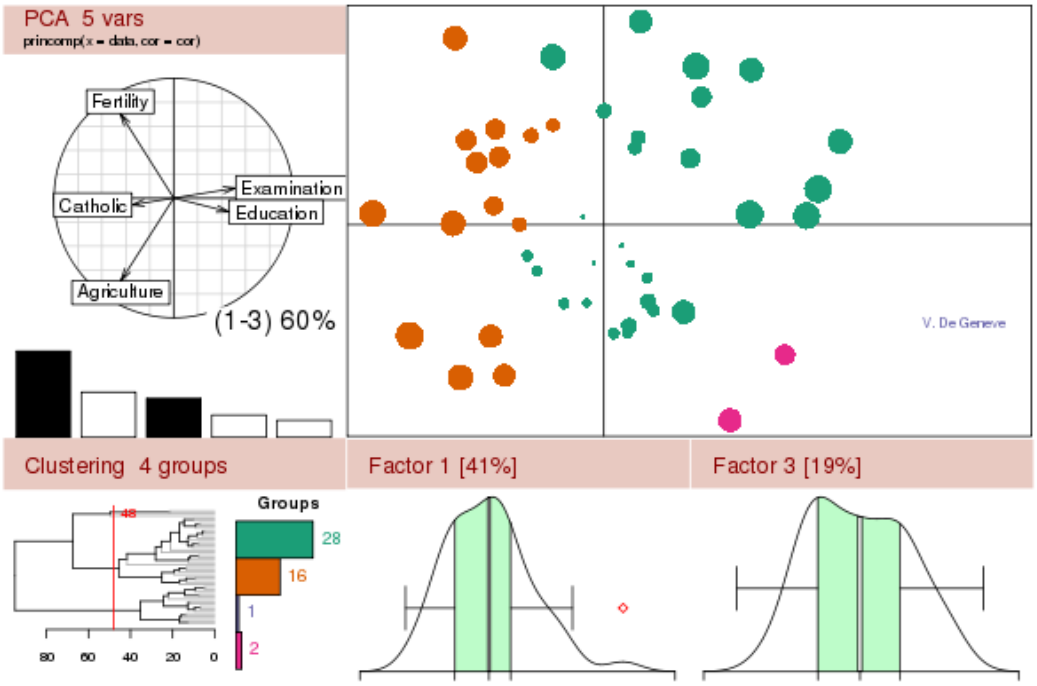
- **R has its own language**
  - R functionality has been made accessible from several scripting languages. E.g.,
    - Python (by the RPy[17] interface package)
    - Perl (by the Statistics::R[18] module).
- **Packages:**
  - Optimization packages are available
  - It can also be used as a general matrix calculation toolbox with comparable benchmark results to GNU Octave and its proprietary counterpart, MATLAB
  - An RWeka[9] interface has been added to the popular data mining software Weka which allows the capability to read/ write into the arff data format thus allowing the usage of data mining capabilities in Weka and statistical in R.

Software and Licenses

- **Available on Windows/Linux/Mac**
- **R is part of the GNU project.**
  - Its source code is freely available under the GNU General Public License, and pre-compiled binary versions are provided for various operating systems.
- **R uses a command line interface, though several graphical user interfaces are available.**

R

- **Intro R website**
  - http://cran.r-project.org/doc/manuals/R-intro.html#Graphics
- **Nice examples**
  - http://www.mayin.org/ajayshah/KB/R/index.html



Online Resources

- **R Site with examples (French, Naïve Bayes)**
  - http://zoonek2.free.fr/UNIX/48_R/12.html#2
- **Intro R website**
  - http://cran.r-project.org/doc/manuals/R-intro.html#Graphics
- **Nice examples**
  - http://www.mayin.org/ajayshah/KB/R/index.html
- **Steward Book website**
  - http://www.stewartcalculus.com/media/9_inside_chapters.php?subaction=showfull&id=1090822711&archive=&start_from=&ucat=2&show_cat=2
- **Taylor's page at Stanford**
  - http://www-stat.stanford.edu/~jtaylo//
- **Contour plots**
  - http://online.redwoods.cc.ca.us/instruct/darnold/MULTCALC/grad/grad.pdf
- **Fox's Book**
  - 2009, http://socserv.socsci.mcmaster.ca/jfox/Courses/R-programming/index.html
  - 2008, http://socserv.socsci.mcmaster.ca/jfox/Courses/R-course/index.html

Online Resources

- **R** http://www.r-project.org/
- **R books online**
  - http://www.math.ccu.edu.tw/~yshih/Rrefs/Rlecturenotes.pdf
  - http://www.cran.r-project.org/doc/contrib/Faraway-PRA.pdf
- **STATISTICS: AN INTRODUCTION USING R (Crawley)**
  - http://www.bio.ic.ac.uk/research/crawley/statistics/exercises.htm
- **Resources at Stanford**
  - http://www-stat.stanford.edu/~jtaylo/courses/stats191/R/logistic/flu.R
  - http://www-stat.stanford.edu/~jtaylo/courses/stats191/R/logistic/fluout.html

Installing R and an Editor

- **Rstudio http://www.rstudio.org/ (self contained environment)**
- **OR do the following:**
- **Installing an editor: EditPlus (for Windows)**
  - Useful Editor on Windows (30 temporary license)
  - http://www.brothersoft.com/download-editplus-16751.html
  - Eclipse
- **Installing R (Windows, also on Linux and Mac)**
  - Click here to download an installer EXE: http://cran.r-project.org/bin/windows/base/R-2.10.0-win32.exe

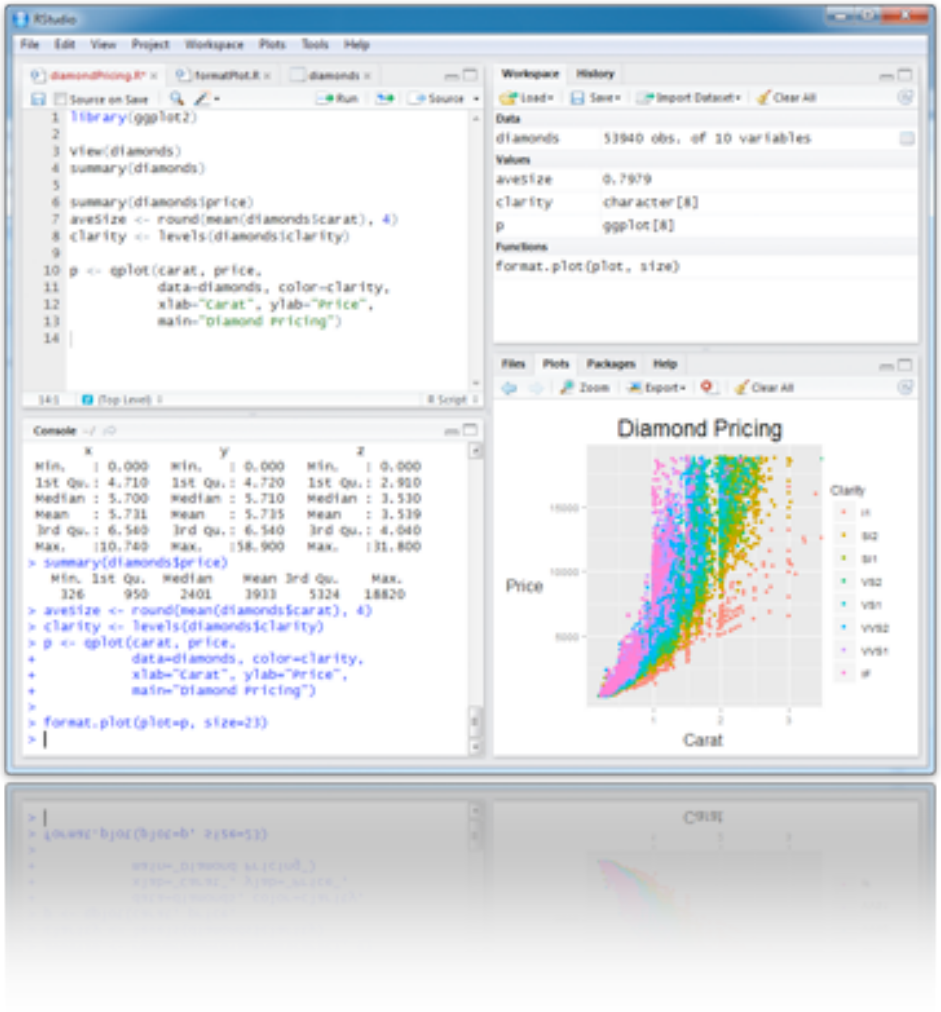
    The distribution is distributed as a 30Mb installer R-2.10.0-win32.exe.

    Just run this for a Windows-XP style installer. It contains all the R components, and you can select what want installed.

    For more details, including command-line options for the installer and how to uninstall, see the rw-FAQ (http://cran.r-project.org/bin/windows/base/rw-FAQ.html).

RStudio

http://www.rstudio.org/

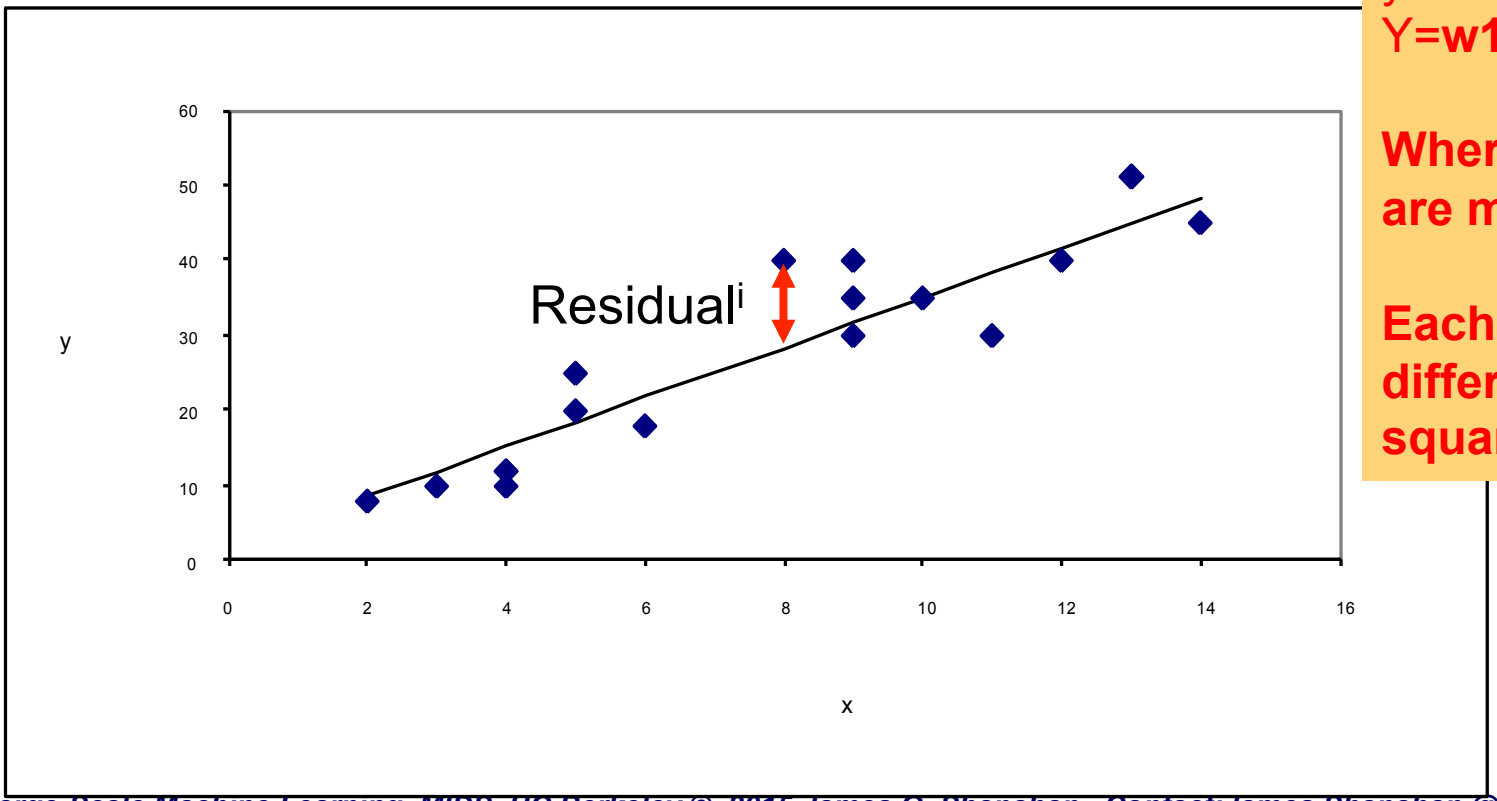


Equation of a line

$$J_q(W, X_1^m) = Minimize \sum_{i=1}^{m} (W^T X_i - y_i)^2$$

$$\text{Residual}^{\text{i}} = \left(WX^i - y^i \right)$$

$$\text{Residual}^{\text{i}} = \left(WX^i - y^i \right)^2$$



y=mx + b
Y=**w1** * x + **w0**

Where w1, w0 are model parameters

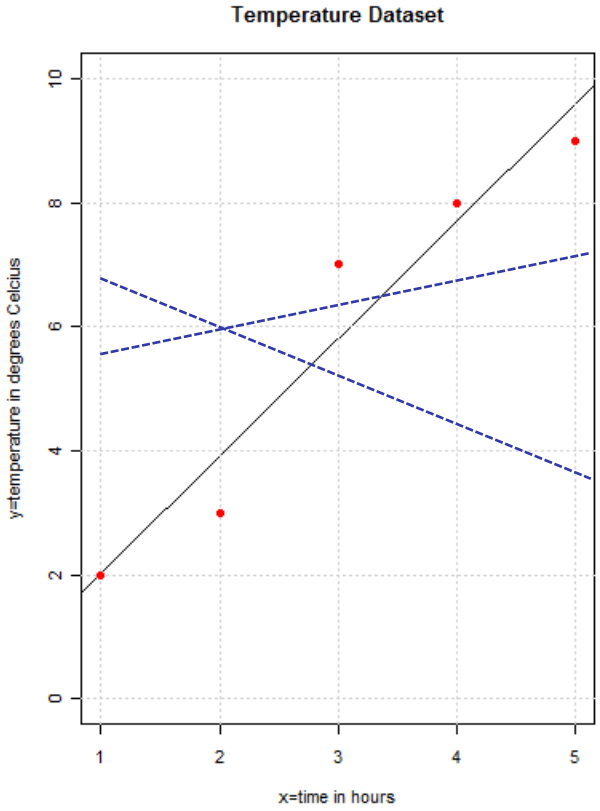
Each pair yields a different sum of squares error

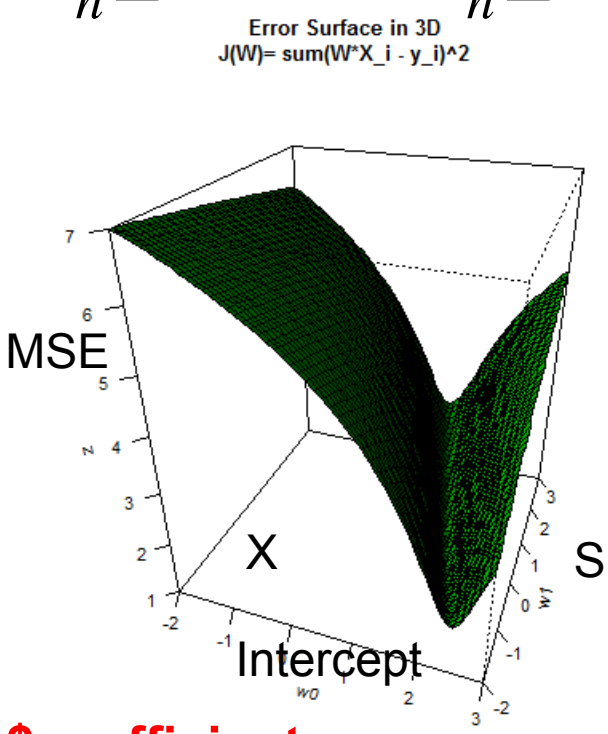
Version Space, Error surface

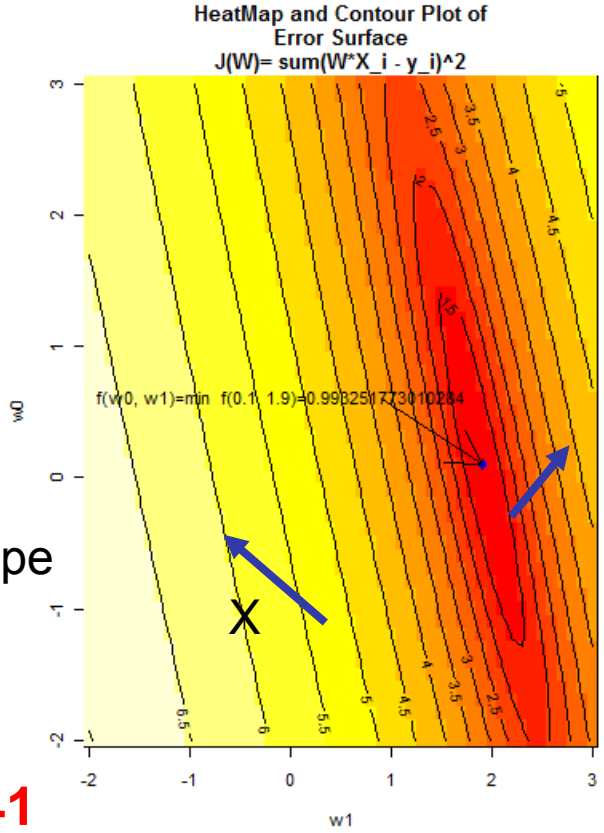
$Y=mx + b$

$Y=w_1x + w_0$

$$MSE = \frac{1}{n}\sum \text{Residual}^i = \frac{1}{n}\sum \left(WX^i - y^i\right)^2$$



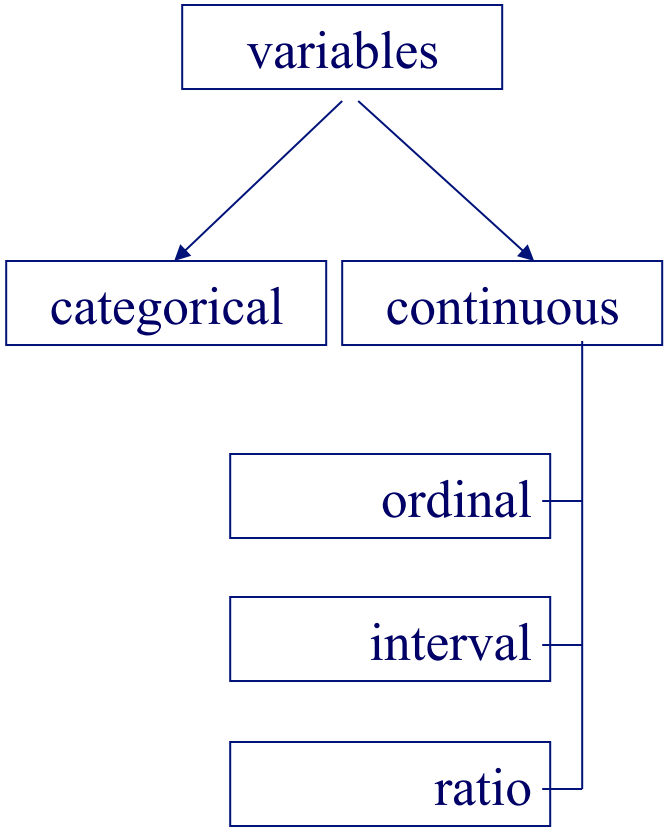




$coefficients
[1] 0.09644596, 1.90098441
$iterations
[1] 658
$SSE
[1] 2.700011

Scales of Measurement

- All measurement in science was conducted using four different types of scales that he called "nominal", "ordinal", "interval" and "ratio"
- In general, many unobservable psychological qualities (e.g., extraversion), are measured on interval scales
- We will mostly concern ourselves with the simple categorical (nominal) versus continuous distinction (ordinal, interval, ratio)
- Check out
  - http://en.wikipedia.org/wiki/Level_of_measurement



Ordinal Measurement

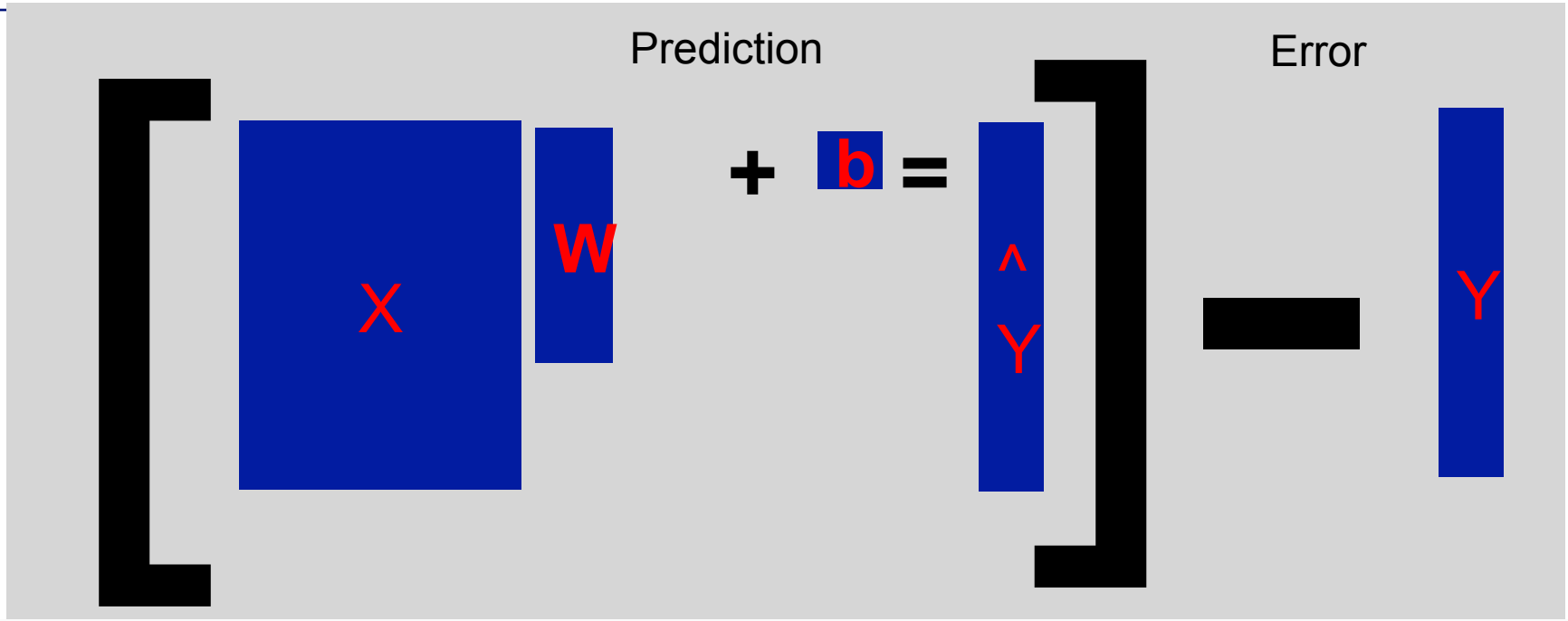
- **Ordinal: Designates an ordering; quasi-ranking**
  - Does not assume that the intervals between numbers are equal.
  - finishing place in a race (first place, second place)

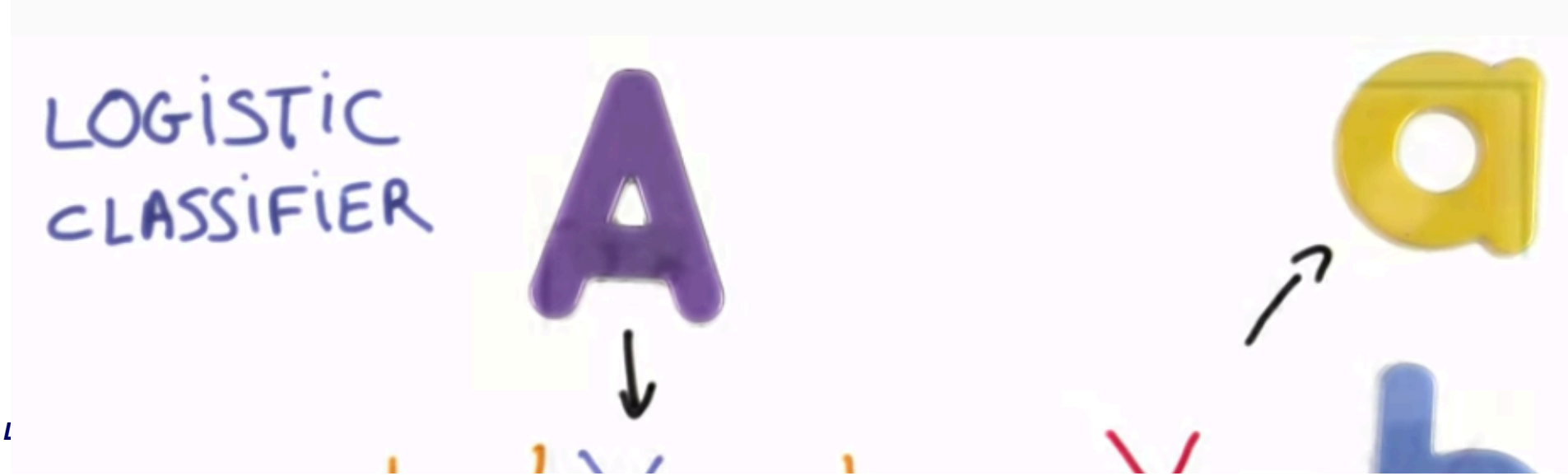
Interval and Ratio Measurement

- **Interval: designates an equal-interval ordering**
  - The distance between, for example, a 1 and a 2 is the same as the distance between a 4 and a 5
  - Example: Common IQ tests are assumed to use an interval metric

- **Ratio: designates an equal-interval ordering with a true zero point (i.e., the zero implies an absence of the thing being measured)**
  - Example: number of intimate relationships a person has had
    - 0 quite literally means *none*
    - a person who has had 4 relationships has had twice as many as someone who has had 2

XW + b = Y





Third generation of Data Analytics: K-Sharing

Ad Tech, Web Search, Social media

Manufacturing, Transportation, Smart cities/ homes, Healthcare, FinTech,and more

WWW/RDF

Social Web

Semantic Web/K-Graph

Gradient Boosted Decision trees

Big Data

Deep Learning

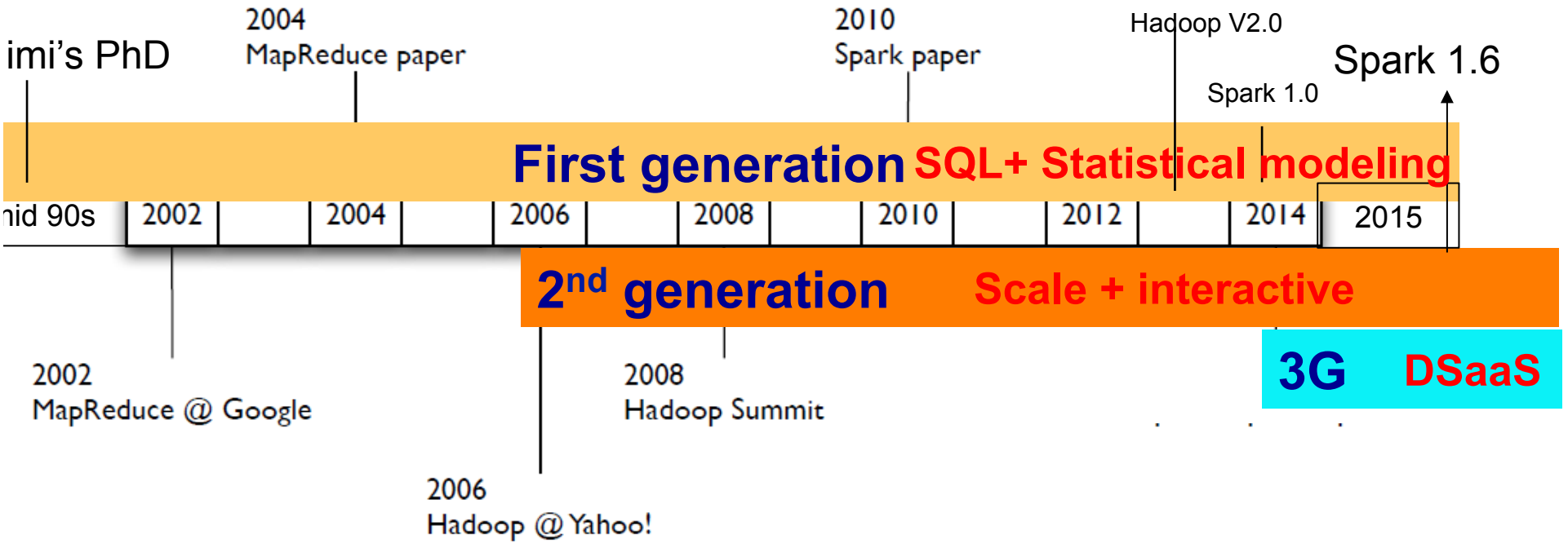
Transfer Learning

Relational DBs

NoSQL DBs

Cloud

In-Memory DBs

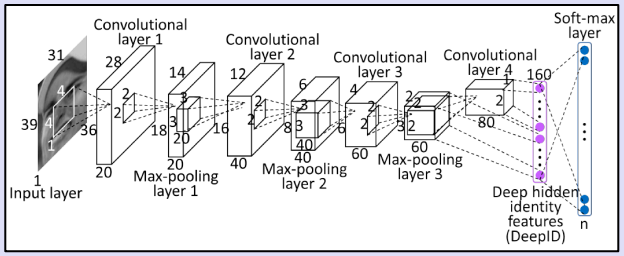


DSaaS:Create workflows + models and SHARE

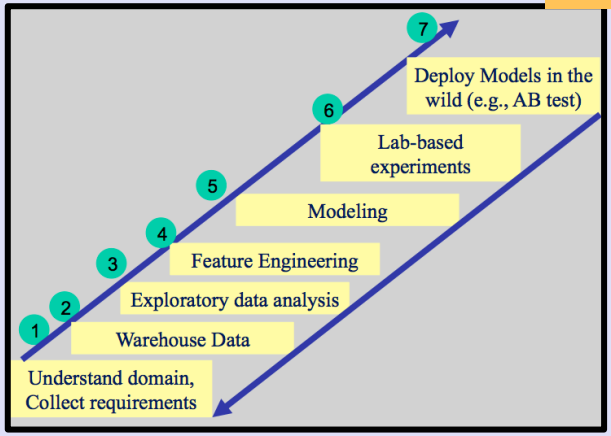
Applications

Dsaas: Opensource distributed DS for the JVM

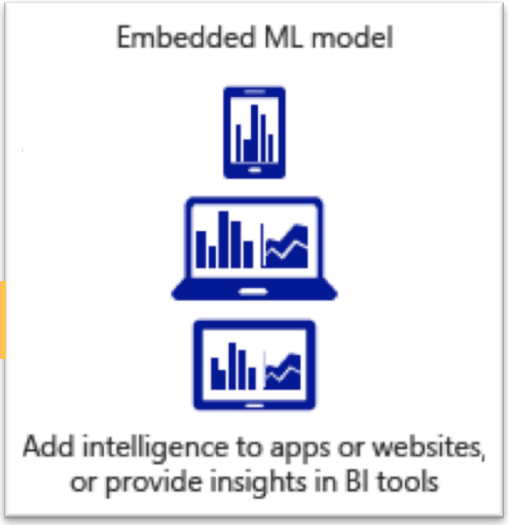
Models



Workflows



Share



Spark/Python/Scala/Java/JVM/SQL/Hive

Data Mgt

Resource Management (Meso/Yarn)

DSaaS: R&D 1/2

- **Systems**
  - Patterns for engineering DSaaS at scale (Map-Reduce)
  - CPU-> GPU-> DSP (tensor programming)
  - Scaling Machine Learning
    - Deep learning; Hierarchical clustering; nonlinear SVMs
  - Graph processing
  - Matrix calculations at scale
- **Theory**
  - Knowledge representation and visualization
    - Workflow management
  - Cooperative frameworks for shared learning
  - Transfer learning
  - Active learning
  - Better Machine Learning
  - Reward framework

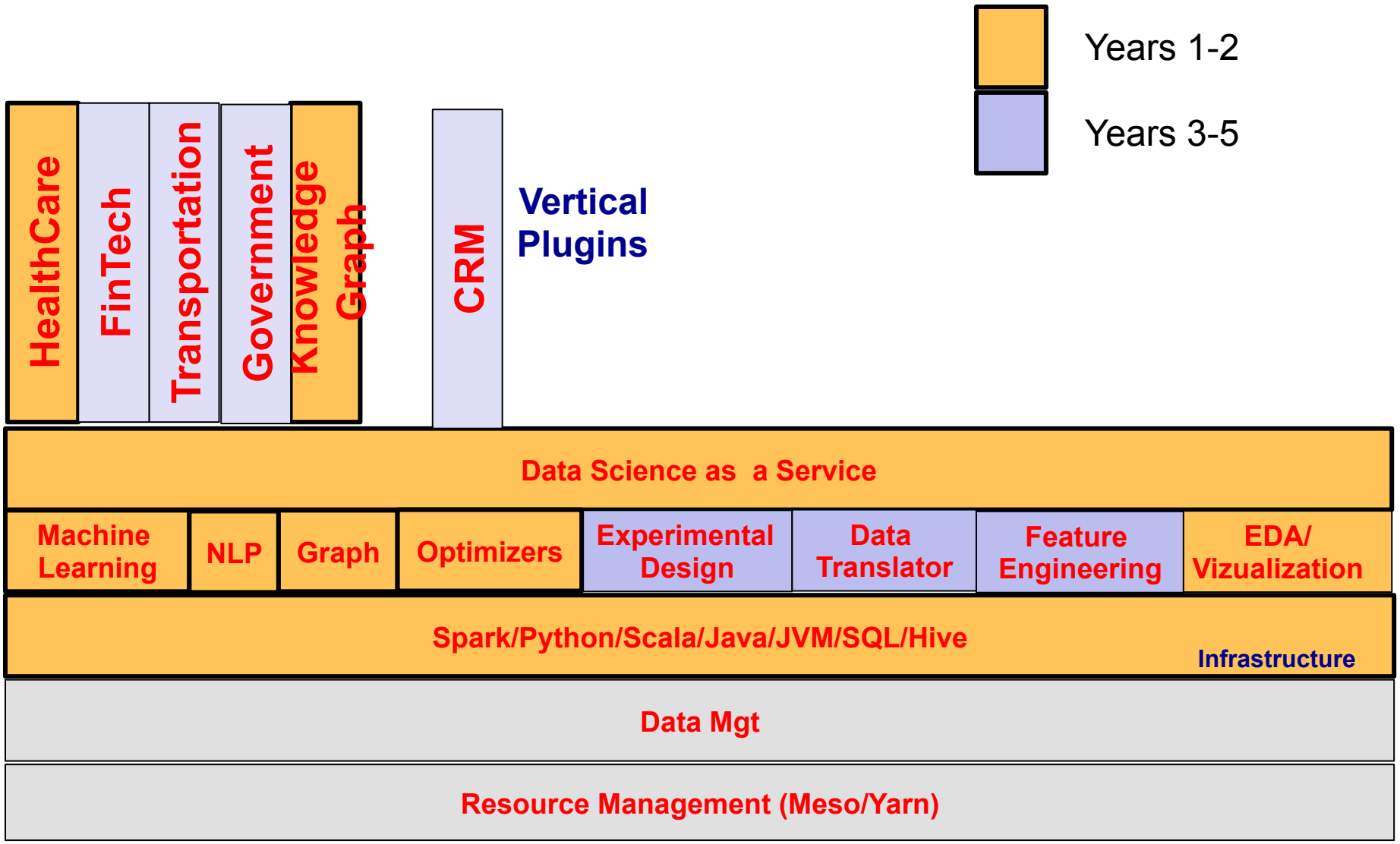
DSaaS: R&D 1/2

- **Verticals**
  - FinTech
  - Government
  - Healthcare
  - Sports and Fitness
  - Advertising and marketing
  - Education
- **Applications: Building example case studies for each usecase/vertical**

DSaaS: 5 year Roadmap

- **Year 1-2:**
  - Assemble Team
  - Build infrastructure to support DSaaS (core functionality)
  - Knowledge representation and visualization for transfer learning
  - CPU-> GPU-> DSP (tensor programming)
  - Build out some 2-3 case studies
  - Adapt for teaching
- **Years 3-5**
  - Get DSaaS to be an Apache project
  - Develop incentive mechanisms
  - Expand infrastructure for supporting DSaaS (Active learning/ experimentation)
  - Build out verticals and integrate

DSaaS: Horizontal framework with vertical plugins



Live Session Outline

- **Machine Learning Introduction**
- **Equation of a line**
- **Linear Regresion**
- **Bias-Variance**
  - Expected Value, Variance
  - Bias-Variance
  - Coding it up
    - For a simulated world (lab setting)
    - For a real world problems
  - Example using bagging

- **A popular question today in the advent of big data**

- **More data scientists versus more data (that another way to ask about do you want a model bias-variance)**

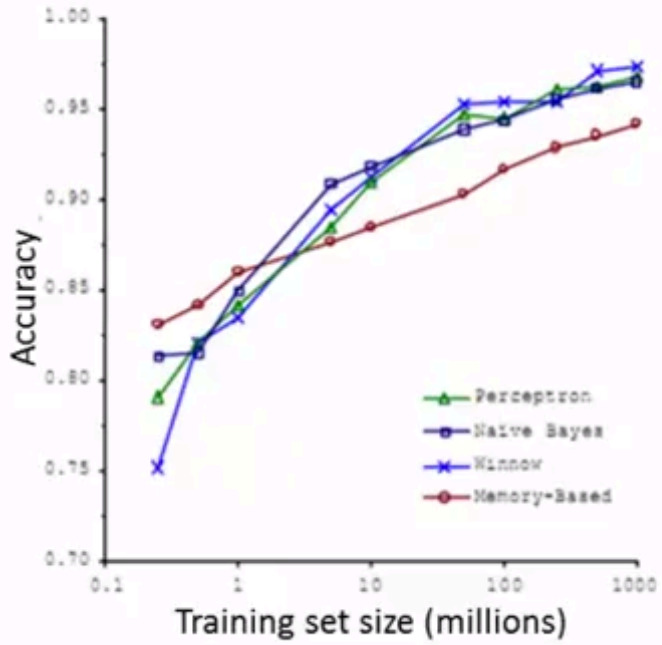
- **Empirical studies suggest more data….But…..**

More data or more data science?

Machine learning and data

Classify between confusable words.
E.g., {to, two, too}, {then, than}.

For breakfast I ate _____ eggs.



"It's not who has the best algorithm that wins.
It's who has the most data."

[Figure from Banko and Brill, 2001]

Andrew Ng

Supervised classification in a nutshell

Given $D = \{(\mathbf{x}_i, y_i)\}_i^n$

label

(sparse) feature vector

Induce $f : X \rightarrow Y$ s.t. loss is minimized

empirical loss = $\frac{1}{n} \sum_{i=0}^{n} \ell(f(\mathbf{x}_i), y_i)$

loss function

Consider functions of a parametric form:

$$\arg\min_{\theta} \frac{1}{n} \sum_{i=0}^{n} \ell(f(\mathbf{x}_i; \theta), y_i)$$

model parameters

Key insight: machine learning as an optimization problem!

(closed form solutions generally not possible)

Gradient Descent

$$w^{(t+1)} = w^{(t)} + \gamma^{(t)} \frac{1}{n} \sum_{i=0}^{n} \nabla l \left(f(\mathbf{x}_i; \theta^{(t)}), y_i \right)$$

"batch" learning: update model after considering all **training instances**

Stochastic Gradient Descent (SGD)

$$w^{(t+1)} = w^{(t)} + \gamma^{(t)} \nabla l \left(f(\mathbf{x}; \theta^{(t)}), y \right)$$

"online" learning: update model after considering ***each*** **(randomly-selected) training instance**

In practice… just as good!

Solves the iteration problem!

What about the single reducer problem?

Ensembles

- Classifier committees are one of the best performing types of learners
- Some of these algorithms are sequential (not very MR friendly)
  - Boosting
- But others rely mostly on randomization
  - Each learner is trained over a different split (features and/or instances) of the data

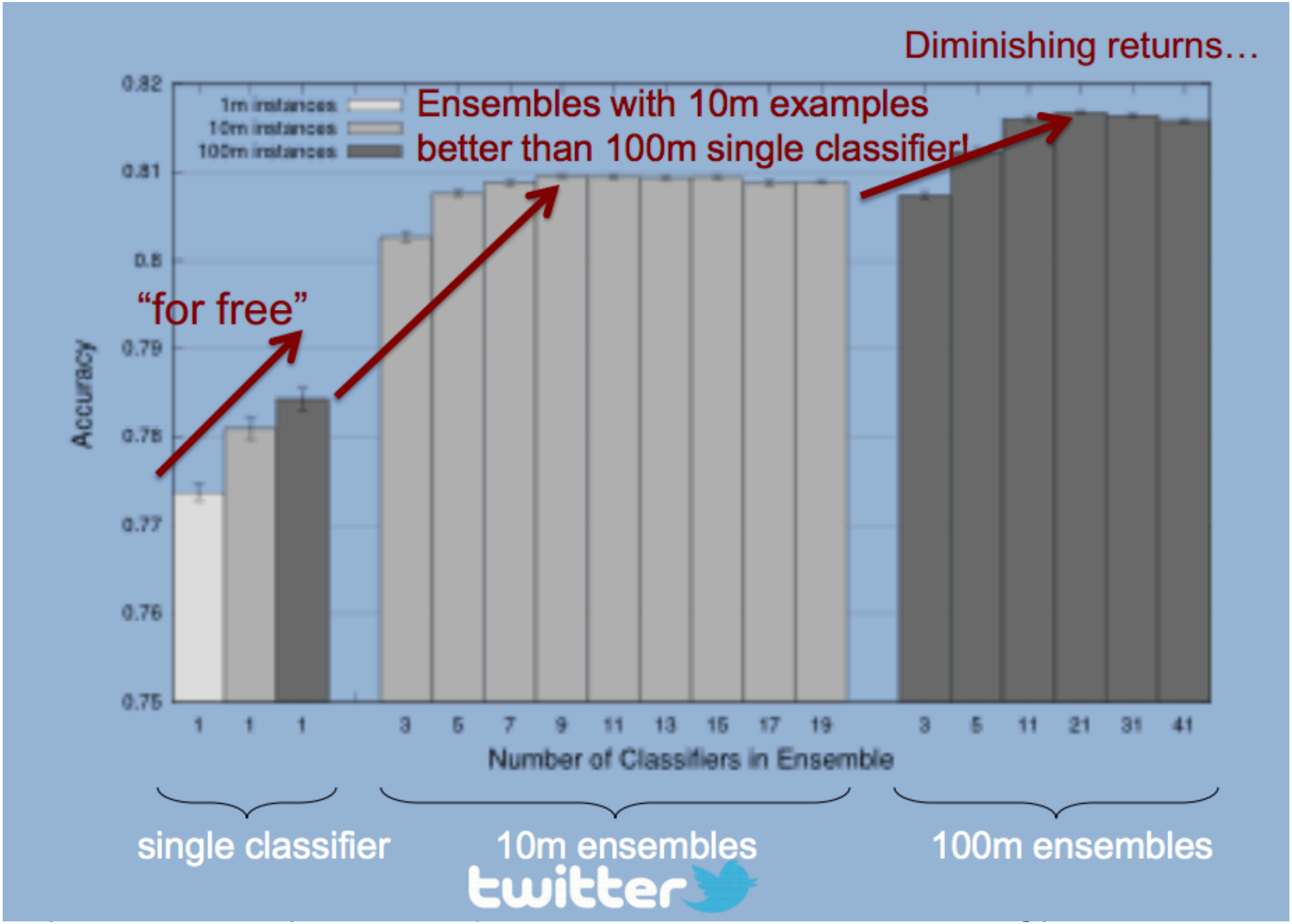
Ensembles: continued

- Ensembles of linear classifiers
  - E.g., each trained using SGD over different subset of features
- Ensembles of decision trees (random forest)
  - Each tree is seeing only a subset of the dataset and node split variables are randomized at each split

twitter

Further advantages of parallelism

- Although stream based learners look at all of the data, a number of them can be executed in parallel
  - Effective for tuning hyper-parameters
- Generative models such as Naïve Bayes are **naturally well suited to distributed learning**
  - Just counting


Diminishing returns…
Ensembles with 10m examples
better than 100m single classifier!
"for free"
1m instances
10m instances
100m instances
Accuracy
0.82
0.81
0.8
0.79
0.78
0.77
0.76
0.75
1 1 1 3 5 7 9 11 13 15 17 19 3 5 11 21 31 41
Number of Classifiers in Ensemble
single classifier
10m ensembles
100m ensembles
twitter

More data or more data scientists

- **Bias (more scientists → increased variance (overfit))**
  - Manage bias via models and features
  - Linear model on nonlinear data?
  - Polynomial model on linear data
- **Variance (more data → reduce variance)**
  - Manage variance with data

- **Look at that characterization next**

Bias Variance Tradeoff

- https://theclevermachine.wordpress.com/2013/04/21/model-selection-underfitting-overfitting-and-the-bias-variance-tradeoff/
- http://insidebigdata.com/2014/10/22/ask-data-scientist-bias-vs-variance-tradeoff/
- **Excellent articles with code in matlab for bias-variance analysis of squared error loss**

Objective of machine learning

- **Objective of machine learning:**
  - Minimize an error/loss function + model complexity
  - leading to better generalization
  - E.g., Mean squared error for regression
- **Decompose loss:**
- **The bias-variance tradeoff is an important aspect way of looking at the loss component of an ML objective function, in particular for big data problems.**

Loss: Irreducible and reducible error

- **To approximate reality, learning algorithms use mathematical or statistical models whose "loss/ error" can be split into two main components:**
  - Reducible
  - And irreducible error.

- **Irreducible error or inherent uncertainty is associated with a natural variability (e.g., noisy sensors) in a system and un-captured aspects of the domain.**

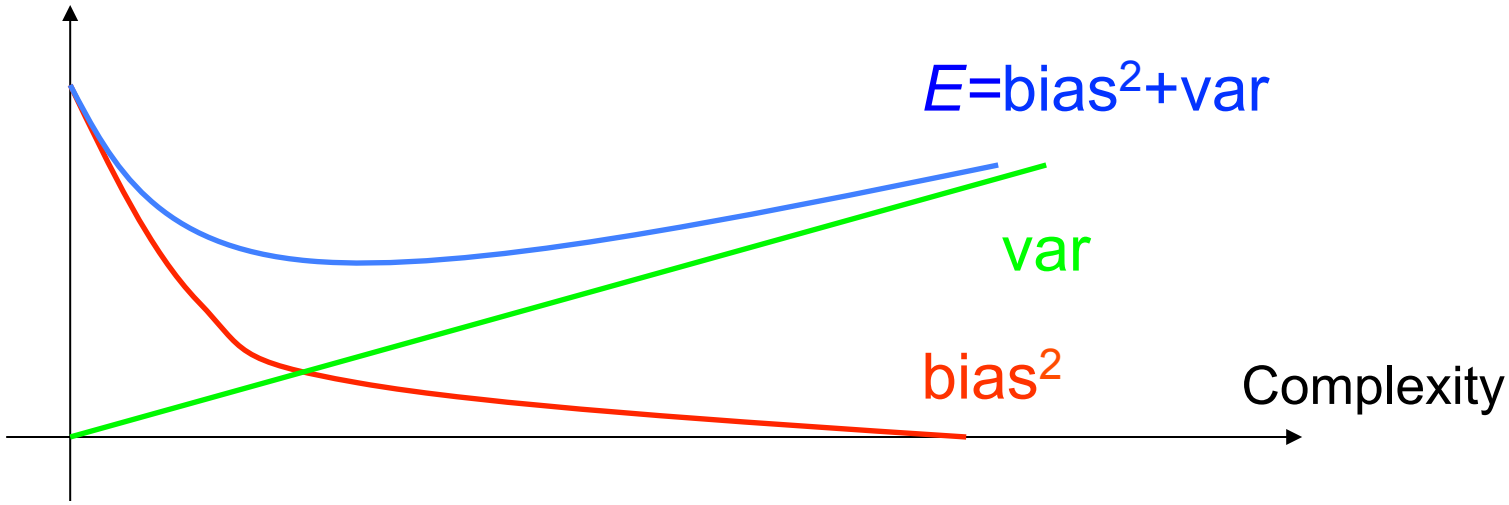
- **On the other hand, reducible error, as the name suggests, can be and should be minimized further to maximize accuracy.**

Reducible error has a bias and variance component

- Reducible error can be further decomposed into "error due to squared bias" and "error due to variance."
- The data scientist's goal is to simultaneously reduce bias and variance as much as possible in order to obtain as accurate model as is feasible.

- However, there is a tradeoff to be made when selecting models of different flexibility or complexity and in selecting appropriate training sets to minimize these sources of error!

Complexity of the model



Usually, the bias is a decreasing function of the complexity, while variance is an increasing function of the complexity.

Live Session Outline

- **Machine Learning Introduction**
- **Equation of a line**
- **Linear Regresion**
- **Bias-Variance**
  - Expected Value, Variance
  - Bias-Variance
  - Coding it up
    - For a simulated world (lab setting)
    - For a real world problems
  - Example using bagging

Expected Value (weighted average)

- **Definition (informal)**
  - The expected value of a random variable X is the weighted average of the values that X can take on, where each possible value is weighted by its respective probability.
  - The expected value of a random variable X is denoted by E(X) and it is often called the expectation of or the mean of X. **We should write $E_{P(x)}[X]$**
- **In probability theory, the expected value (or expectation, or mathematical expectation, or mean, or the first moment) of a random variable is the weighted average of all possible values that this random variable can take on.**
  - The weights used in computing this average correspond to the probabilities in case of a discrete random variable, or densities in case of a continuous random variable.
  - From a rigorous theoretical standpoint, the expected value is the integral of the random variable with respect to its probability measure.

Expected Value for Discrete Variable

When X is a discrete random variable having support R_X and probability mass function $p_X(x)$, the formula for computing its expected value is a straightforward implementation of the informal definition given above: the expected value of X is the weighted average of the values that X can take on (the elements of R_X), where each possible value $x \in R_X$ is weighted by its respective probability $p_X(x)$.

> **Definition** Let X be a discrete random variable with support R_X and probability mass function $p_X(x)$. The expected value of X is:
>
> $$\mathrm{E}[X] = \sum_{x \in R_X} x p_X(x)$$
>
> provided that:
>
> $$\sum_{x \in R_X} |x| p_X(x) < \infty$$

$E_{P(x)}[X]$ where p(x) is uniform

Good

E [X] Bad notation

The symbol

$$\sum_{x \in R_X}$$

indicates summation over all the elements of the support R_X. So, for example, if

$$R_X = \{1, 2, 3\}$$

then:

$$\sum_{x \in R_X} x p_X(x) = 1 \cdot p_X(1) + 2 \cdot p_X(2) + 3 \cdot p_X(3)$$

Expected Value wrt

Suppose random variable X can take value x_1 with probability p_1, value x_2 with probability p_2, and so on, up to value x_k with probability p_k. Then the **expectation** of this random variable X is defined as

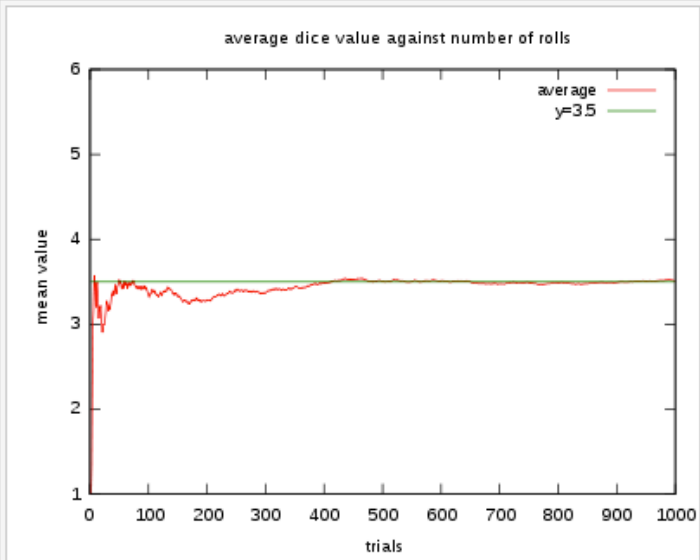
$$\mathrm{E}[X] = x_1p_1 + x_2p_2 + \ldots + x_kp_k \ .$$

Since all probabilities p_i add up to one: $p_1 + p_2 + ... + p_k = 1$, the expected value can be viewed as the weighted average, with p_i's being the weights:

$$\mathrm{E}[X] = \frac{x_1p_1 + x_2p_2 + \ldots + x_kp_k}{p_1 + p_2 + \ldots + p_k} \ .$$

If all outcomes x_i are equally likely (that is, $p_1 = p_2 = ... = p_k$), then the weighted average turns into the simple average. This is intuitive: the expected value of a random variable is the average of all values it can take; thus the expected value is what you expect to happen *on average*. If the outcomes x_i are not equiprobable, then the simple average ought to be replaced with the weighted average, which takes into account the fact that some outcomes are more likely than the others. The intuition however remains the same: the expected value of X is what you expect to happen *on average*.

Example 1. Let X represent the outcome of a roll of a six-sided die. More specifically, X will be the number of pips showing on the top face of the die after the toss. The possible values for X are 1, 2, 3, 4, 5, 6, all equally likely (each having the probability of $\frac{1}{6}$). The expectation of X is



An illustration of the convergence of sequence averages of rolls of a die to the expected value of 3.5 as the number of rolls (trials) grows.

$$\mathrm{E}[X] = 1 \cdot \frac{1}{6} + 2 \cdot \frac{1}{6} + 3 \cdot \frac{1}{6} + 4 \cdot \frac{1}{6} + 5 \cdot \frac{1}{6} + 6 \cdot \frac{1}{6} = 3.5$$

More Generally..

When X is an absolutely continuous random variable with probability density function $f_X(x)$, the formula for computing its expected value involves an integral, which can be thought of as the limiting case of the summation $\sum_{x \in R_X} x p_X(x)$ found in the discrete case above.

Definition Let X be an absolutely continuous random variable with probability density function $f_X(x)$. The expected value of X is:

$$\mathrm{E}[X] = \int_{-\infty}^{\infty} x f_X(x) dx$$

In general, if X is a random variable defined on a probability space (Ω, Σ, P), then the expected value of X, denoted by $\mathrm{E}[X]$, $\langle X \rangle$, $\overline{X}$ or $\mathbf{E}[X]$, is defined as Lebesgue integral

$$\mathrm{E}[X] = \int_{\Omega} X \, \mathrm{d}\, P = \int_{\Omega} X(\omega)\, P(\mathrm{d}\,\omega)$$

When this integral exists, it is defined as the expectation of X. Note that not all random variables have a finite expected value, since the integral may not converge absolutely; furthermore, for some it is not defined at all (e.g., Cauchy distribution). Two variables with the same probability distribution will have the same expected value, if it is defined.

It follows directly from the discrete case definition that if X is a constant random variable, i.e. $X = b$ for some fixed real number b, then the expected value of X is also b.

The expected value of an arbitrary function of X, $g(X)$, with respect to the probability density function $f(x)$ is given by the inner product of f and g:

Variance

- **In probability theory and statistics, the variance is a measure of how far a set of numbers are spread out from each other. It is one of several descriptors of a probability distribution, describing how far the numbers lie from the mean (expected value).**

If a random variable X has the expected value (mean) $\mu = \mathrm{E}[X]$, then the variance of X is given by:

$$\operatorname{Var}(X) = \mathrm{E}[(X - \mu)^2] \,.$$

$E_{P(x)}[(X- Mu)^2]$

That is, the variance is the expected value of the squared difference be
mean. This definition encompasses random variables that are discrete, continuous, or neither (or mixed). It can be expanded as follows:

$$\begin{aligned}
\operatorname{Var}(X) &= \mathrm{E}[(X - \mu)^2] \\
&= \mathrm{E}[X^2 - 2\mu X + \mu^2] \\
&= \mathrm{E}[X^2] - 2\mu\ \mathrm{E}[X] + \mu^2 \\
&= \mathrm{E}[X^2] - 2\mu^2 + \mu^2 \\
&= \mathrm{E}[X^2] - \mu^2 \\
&= \mathrm{E}[X^2] - (\mathrm{E}[X])^2.
\end{aligned}$$

Mean of the square MINUS square of the mean

A mnemonic for the above expression is "mean of square minus square of mean". The variance of random variable X is typically designated as $\operatorname{Var}(X)$, σ_X^2, or simply σ^2 (pronounced "sigma squared").

Variance of a Fair Dice

A six-sided fair die can be modelled with a discrete random variable with outcomes 1 through 6, each with equal probability $\frac{1}{6}$. The expected value is (1 + 2 + 3 + 4 + 5 + 6)/6 = 3.5. Therefore the variance can be computed to be:

$$\sum_{i=1}^{6} \tfrac{1}{6}(i-3.5)^2 = \tfrac{1}{6}\sum_{i=1}^{6}(i-3.5)^2 = \tfrac{1}{6}\left((-2.5)^2+(-1.5)^2+(-0.5)^2+0.5^2+1.5^2+2.5^2\right)$$

$$= \tfrac{1}{6}\cdot 17.50 = \tfrac{35}{12} \approx 2.92.$$

Standard deviation = ~1.5

Standard Deviation

- Standard deviation is a widely used measure of variability or diversity used in statistics and probability theory. It shows how much variation or "dispersion" there is from the average (mean, or expected value). A low standard deviation indicates that the data points tend to be very close to the mean, whereas high standard deviation indicates that the data points are spread out over a large range of values.
- The standard deviation of a statistical population, data set, or probability distribution is the square root of its variance. It is algebraically simpler though practically less robust than the average absolute deviation.[1][2]
- A useful property of standard deviation is that, unlike variance, it is expressed in the same units as the data.

Implications of the mean and SD

- ***"In the Vietnamese population aged 30+ years, the average of weight was 55.0 kg, with the SD being 8.2 kg."***
- **What does this mean?**

- 68% individuals will have height between 55 +/- 8.2*1 = 46.8 to 63.2 kg

- 95% individuals will have height between 55 +/- 8.2*1.96 = 38.9 to 71.1 kg

Live Session Outline

- **Machine Learning Introduction**
- **Equation of a line**
- **Linear Regresion**
- **Bias-Variance**
  - Expected Value, Variance
  - Bias-Variance
  - Coding it up
    - For a simulated world (lab setting)
    - For a real world problems
  - Example using bagging

How good is your learnt model?

- **To assess the effectiveness of a machine learnt model, we want to know the expectation of the MSE (assume a regression problem domain) if we test the model on arbitrarily many test points drawn from the unknown function.**

Given:

- the true function we want to approximate

$$f = f(\mathbf{x})$$

In Simulated world we know f(X)

- the data set for training

$$D = \{(\mathbf{x}_1, t_1), (\mathbf{x}_2, t_2), \ldots, (\mathbf{x}_N, t_N)\} \text{ where } t = f + \varepsilon \text{ and } E\{\varepsilon\} = 0$$

or Linear regression

- given D, we train an arbitrary neural network^to approximate the function f by

$$y = g(\mathbf{x}, \mathbf{w})$$

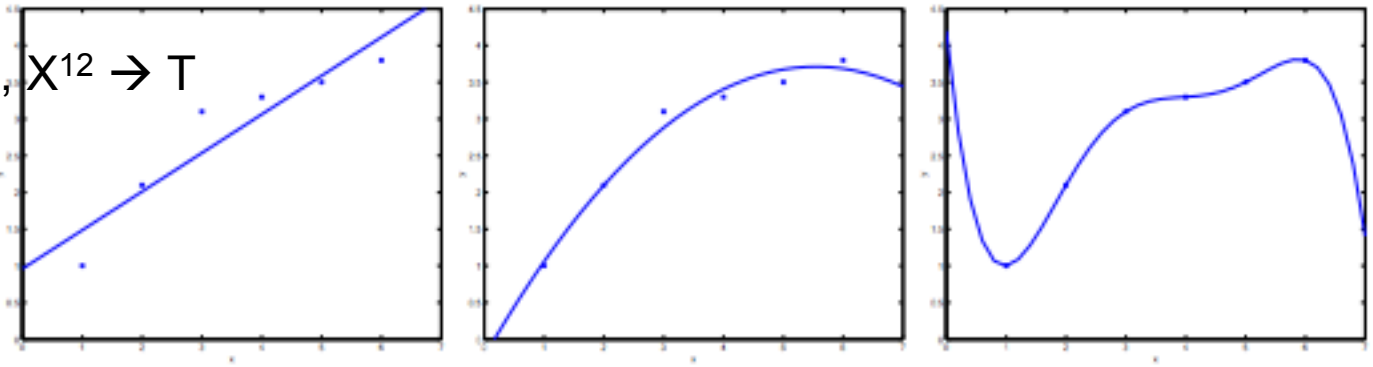
The mean-squared error of this networks is:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(t_i - y_i)^2$$

1 Bias/variance tradeoff

When talking about linear regression, we discussed the problem of whether to fit a "simple" model such as the linear "$y = \theta_0 + \theta_1 x$," or a more "complex" model such as the polynomial "$y = \theta_0 + \theta_1 x + \cdots \theta_5 x^5$." We saw the following example:

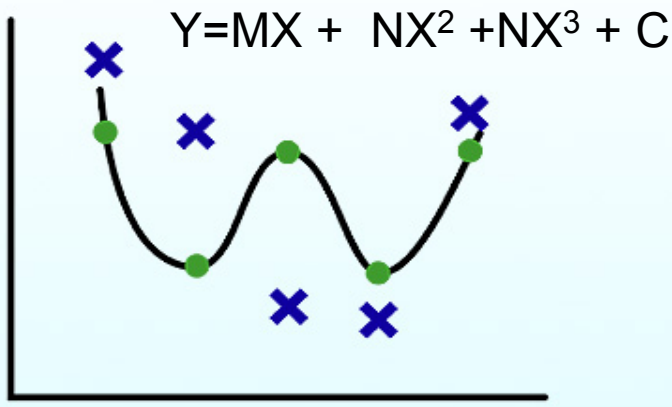
- $X, X^2, X^3 \ldots, X^{12} \rightarrow T$



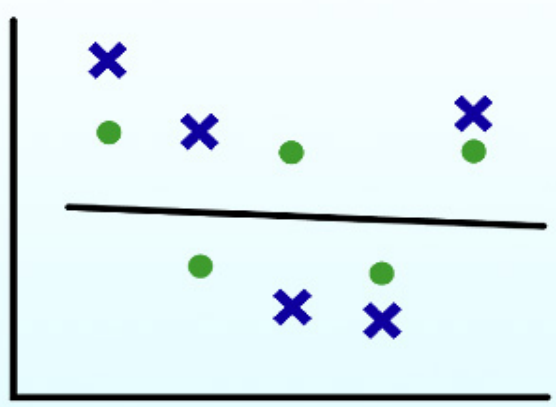
Fitting a 5th order polynomial to the data (rightmost figure) did not result in a good model. Specifically, even though the 5th order polynomial did a very good job predicting y (say, prices of houses) from x (say, living area) for the examples in the training set, we do not expect the model shown to be a good one for predicting the prices of houses not in the training set. In other words, what's has been learned from the training set does not *generalize* well to other houses. The **generalization error** (which will be made formal shortly) of a hypothesis is its expected error on examples not necessarily in the training set.

Both the models in the leftmost and the rightmost figures above have large generalization error. However, the problems that the two models suffer from are very different. If the relationship between y and x is not linear,

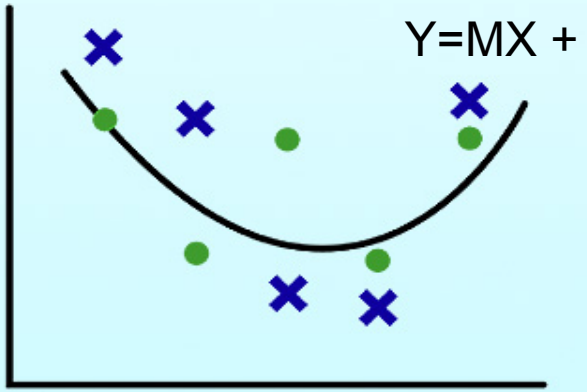
Bias-Variance Tradeoff in Model Selection in Simple Problem



(a) High variance/low bias.
4th-order polynomial ($p = 5$).



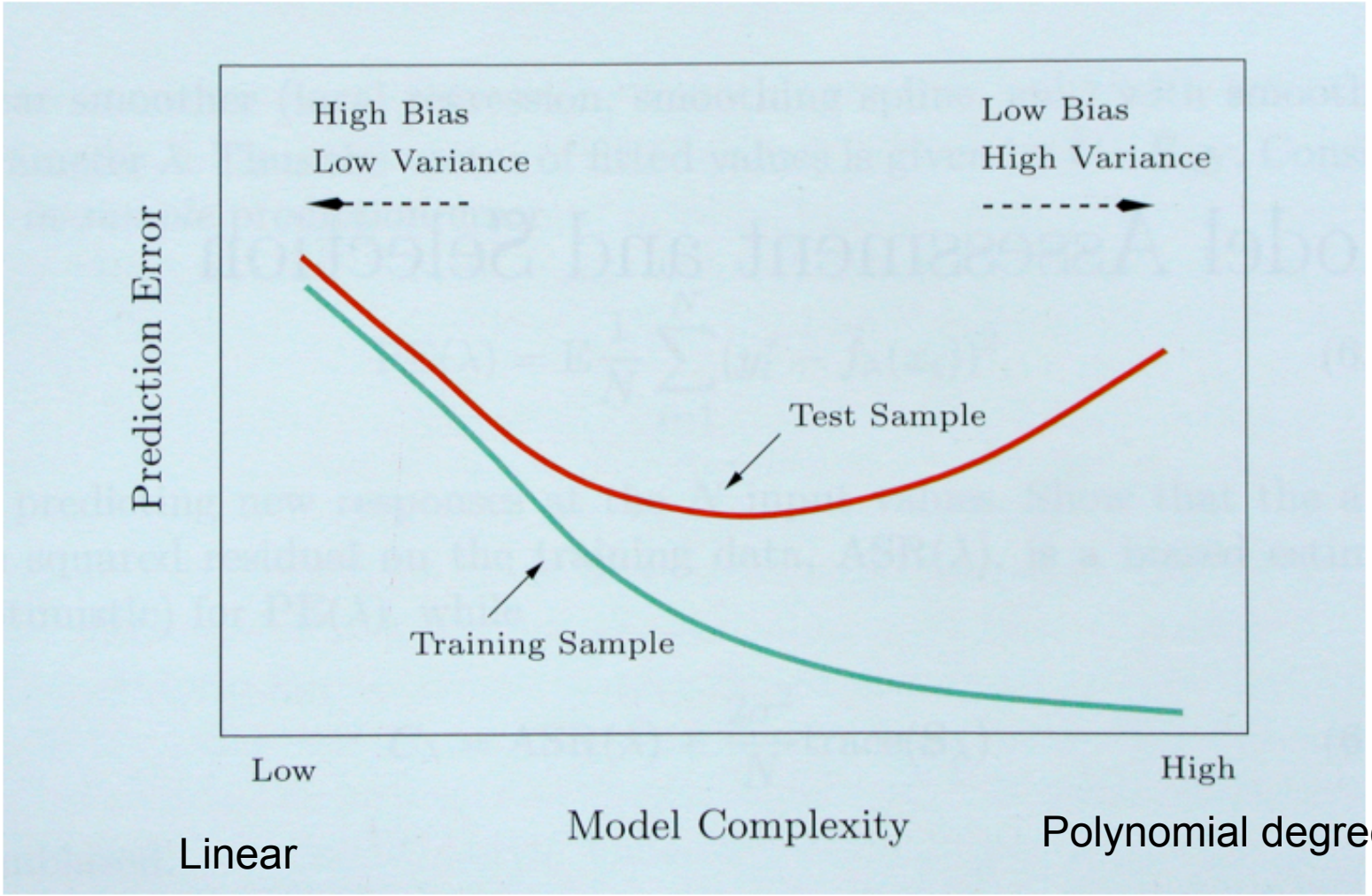
(b) Low variance/high bias.
1st-order polynomial ($p = 2$).



(c) Balanced variance & bias.
Minimum MSE.
2nd-order polynomial ($p = 3$).

- Data points for fitting
- Typical new data points

Bias/Variance Tradeoff

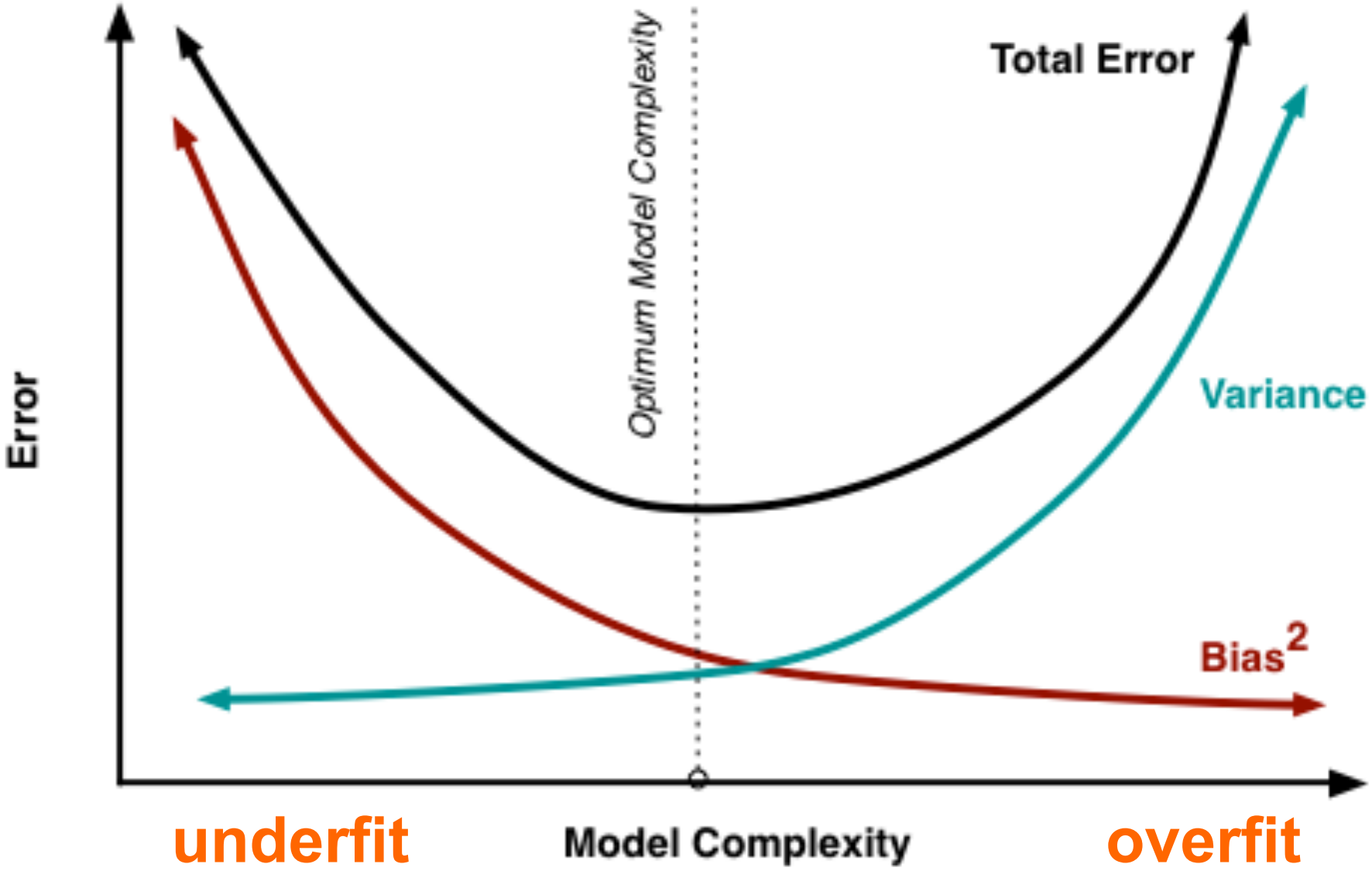


Linear

Polynomial degree 12

Hastie, Tibshirani, Friedman "Elements of Statistical Learning" 2001

Optimum Model



Bias and Variance: Conceptually

- **Error due to Bias:**
  - The error due to bias is taken as the difference between the expected (or average) prediction of our model and the correct value which we are trying to predict.
  - Of course you only have one model so talking about expected or average prediction values might seem a little strange.
  - However, imagine you could repeat the whole model building process more than once: each time you gather new data and run a new analysis creating a new model.
  - Due to randomness in the underlying data sets, the resulting models will have a range of predictions. Bias measures how far off in general these models' predictions are from the correct value.
- **Error due to Variance:**
  - …….

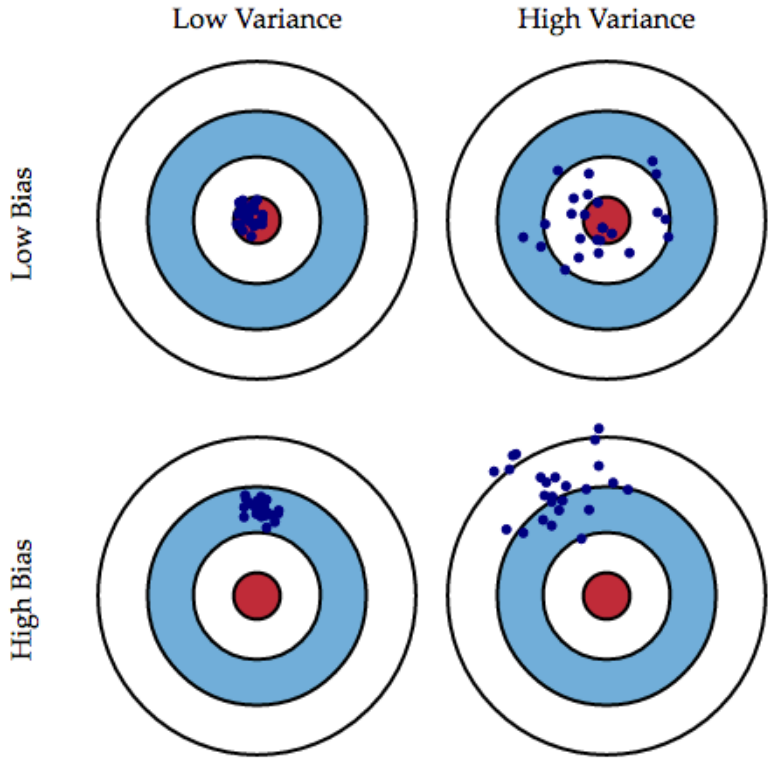
Bias and Variance: Conceptually

- **Error due to Bias:**
  - ……
- **Error due to Variance:**
  - The error due to variance is taken as the variability of a model prediction for a given data point.
  - Again, imagine you can repeat the entire model building process multiple times.
  - The variance is how much the predictions for a given point vary between different realizations of the model.

Bias and Variance: Graphically

We can create a graphical visualization of bias and variance using a bulls-eye diagram. Imagine that the center of the target is a model that perfectly predicts the correct values. As we move away from the bulls-eye, our predictions get worse and worse. Imagine we can repeat our entire model building process to get a number of separate hits on the target. Each hit represents an individual realization of our model, given the chance variability in the training data we gather. Sometimes we will get a good distribution of training data so we predict very well and we are close to the bulls-eye, while sometimes our training data might be full of outliers or non-standard values resulting in poorer predictions. These different realizations result in a scatter of hits on the target.

We can plot four different cases representing combinations of both high and low bias and variance.



BV: Mathematical Definition

If we denote the variable we are trying to predict as Y and our covariates as X, we may assume that there is a relationship relating one to the other such as $Y = f(X) + \epsilon$ where the error term ϵ is normally distributed with a mean of zero like so $\epsilon \sim \mathcal{N}(0, \sigma_\epsilon)$.

We may estimate a model $\hat{f}(X)$ of $f(X)$ using linear regressions or another modeling technique. In this case, the expected squared prediction error at a point x is:

$$Err(x) = E\left[(Y - \hat{f}(x))^2\right]$$

This error may then be decomposed into bias and variance components:

$$Err(x) = \left(E[\hat{f}(x)] - f(x)\right)^2 + E\left[\left(\hat{f}(x) - E[\hat{f}(x)]\right)^2\right] + \sigma_e^2$$

$$Err(x) = \text{Bias}^2 + \text{Variance} + \text{Irreducible Error}$$

That third term, irreducible error, is the noise term in the true relationship that cannot fundamentally be reduced by any model. Given the true model and infinite data to calibrate it, we should be able to reduce both the bias and variance terms to 0. However, in a world with imperfect models and finite data, there is a tradeoff between minimizing the bias and minimizing the variance.

Bias–variance decomposition of squared error [edit]

Suppose that we have a training set consisting of a set of points $x_1, \ldots, x_n$ and real values y_i associated with each point x_i. We assume that there is a functional, but noisy relation $y_i = f(x_i) + \epsilon$, where the noise, ϵ, has zero mean and variance σ^2.

We want to find a function $\hat{f}(x)$, that approximates the true function $y = f(x)$ as well as possible, by means of some learning algorithm. We make "as well as possible" precise by measuring the mean squared error between y and $\hat{f}(x)$: we want $(y - \hat{f}(x))^2$ to be minimal, both for $x_1, \ldots, x_n$ *and for points outside of our sample*. Of course, we cannot hope to do so perfectly, since the y_i contain noise ϵ; this means we must be prepared to accept an *irreducible error* in any function we come up with.

Finding an $\hat{f}$ that generalizes to points outside of the training set can be done with any of the countless algorithms used for supervised learning. It turns out that whichever function $\hat{f}$ we select, we can decompose its expected error on an unseen sample x as follows:[3]:34[4]:223

$$\mathrm{E}\left[(y - \hat{f}(x))^2\right] = \mathrm{Bias}\left[\hat{f}(x)\right]^2 + \mathrm{Var}\left[\hat{f}(x)\right] + \sigma^2$$

Where:

$$\mathrm{Bias}\left[\hat{f}(x)\right] = \mathrm{E}\left[\hat{f}(x)\right] - f(x)$$

and

$$\mathrm{Var}\left[\hat{f}(x)\right] = \mathrm{E}\left[\left(\hat{f}(x) - \mathrm{E}[\hat{f}(x)]\right)^2\right]$$

The expectation ranges over different choices of the training set $x_1, \ldots, x_n, y_1, \ldots, y_n$, all sampled from the same distribution. The three terms represent:

- the square of the *bias* of the learning method, which can be thought of the error caused by the simplifying assumptions built into the method. E.g., when approximating a non-linear function $f(x)$ using a learning method for linear models, there will be error in the estimates $\hat{f}(x)$ due to this assumption;
- the *variance* of the learning method, or, intuitively, how much the learning method $\hat{f}(x)$ will move around its mean;
- the irreducible error σ^2. Since all three terms are non-negative, this forms a lower bound on the expected error on unseen samples.[3]:34

The more complex the model $\hat{f}(x)$ is, the more data points it will capture, and the lower the bias will be. However, complexity will make the model "move" more to capture the data points, and hence its variance will be larger.

https://en.wikipedia.org/wiki/Bias%E2%80%93variance_tradeoff

Bias-Variance Analysis

- Given a new data point **x**, what is the **expected prediction error?**
- Assume that the data points are drawn i.i.d. from *a unique underlying probability distribution P*
- The goal of the analysis is to compute, for an arbitrary new point **x**,

$$E_P\left[(y - h(\mathbf{x}))^2\right]$$

  where y is the value of **x** that could be present in a data set, and the expectation is over *all training sets* drawn according to P.
- We will decompose this expectation into three components:

  bias, variance and noise

Explore the expectation of the MSE

Given:

- the true function we want to approximate

$$f = f(\mathbf{x})$$

Perfect world

- the data set for training

Observed

$$D = \{(\mathbf{x}_1, t_1), (\mathbf{x}_2, t_2), \ldots, (\mathbf{x}_N, t_N)\} \text{ where } t = f + \varepsilon \text{ and } E\{\varepsilon\} = 0$$

- given D, we train an arbitrary neural network to approximate the function f by

t is the target (or observed) and y is prediction

$$y = g(\mathbf{x}, \mathbf{w})$$

The mean-squared error of this networks is:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(t_i - y_i)^2$$

To assess the effectiveness of the network, we want to know the expectation of the MSE if we test the network on arbitrarily many test points drawn from the unknown function.

$$E\{MSE\} = E\left\{\frac{1}{N}\sum_{i=1}^{N}(t_i - y_i)^2\right\} = \frac{1}{N}\sum_{i=1}^{N}E\left\{(t_i - y_i)^2\right\}$$

Error = E(observed – prediction) can be decomposed

Let's investigate the expectation inside the sum, with a little "augmentation trick":

t is the target and y is prediction

$$E\{(t_i - y_i)^2\} = E\{(t_i - f_i + f_i - y_i)^2\}$$

$$= E\{(t_i - f_i)^2\} + E\{(f_i - y_i)^2\} + 2E\{(f_i - y_i)(t_i - f_i)\}$$

$$= E\{\varepsilon^2\} + E\{(f_i - y_i)^2\} + 2\left(E\{f_i t_i\} - E\{f_i^2\} - E\{y_i t_i\} + E\{y_i f_i\}\right)$$

Note : $E\{f_i t_i\} = f_i^2$ since f is deterministic and $E\{t_i\} = f_i$

: $E\{f_i^2\} = f_i^2$ since f is deterministic

: $E\{y_i t_i\} = E\{y_i(f_i + \varepsilon)\} = E\{y_i f_i + y_i \varepsilon\} = E\{y_i f_i\} + 0$

: (the last term is zero because the noise in the infinite test set over which

: we take the expectation is probabilistically independent of the NN

: prediction). Thus the last term in the expectation above cancels to zero.

$$E\{(t_i - y_i)^2\} = E\{\varepsilon^2\} + E\{(f_i - y_i)^2\}$$

noise = $E(t_i - f_i)^2$

$E(t_i - y_i)^2 = E(t_i - f_i)^2 + E(f_i - y_i)$

Thus the MSE can be decomposed in expectation into the variance of the noise and the MSE between the true function and the predicted values. This term can be further composed with the same augmentation trick as above

t is the target and y is prediction

$$E\left\{(f_i - y_i)^2\right\} = E\left\{(f_i - E\{y_i\} + E\{y_i\}_i - y_i)^2\right\}$$

$$= E\left\{(f_i - E\{y_i\})^2\right\} + E\left\{(E\{y_i\} - y_i)^2\right\} + 2E\left\{(E\{y_i\} - y_i)(f_i - E\{y_i\})\right\}$$

$$= bias^2 + Var\{y_i\} + 2\left(E\{f_i E\{y_i\}\} - E\left\{E\{y_i\}^2\right\} - E\{y_i f\}_i + E\{y_i E\{y_i\}\}\right)$$

Note : $E\{f_i E\{y_i\}\} = f_i E\{y_i\}$ since f is deterministic and $E\{E\{z\}\} = z$

: $E\left\{E\{y_i\}^2\right\} = E\{y_i\}^2$ since $E\{E\{z\}\} = z$

: $E\{y_i f_i\} = f_i E\{y_i\}$

: $E\{y_i E\{y_i\}\} = E\{y_i\}^2$

: Thus the last term in the expectation above cancels to zero.

$$E\left\{(f_i - y_i)^2\right\} = bias^2 + Var\{y_i\}$$

Thus the decomposition of the MSE in expectation becomes:

$$E\left\{(t_i - y_i)^2\right\} = Var\{noise\} + bias^2 + Var\{y_i\}$$

The Bias-Variance Tradeoff

t is the target and y is prediction

Bias-Variance

Given:

- the true function we want to approximate

$$f = f(\mathbf{x})$$

- the data set for training

$$D = \{(\mathbf{x}_1, t_1), (\mathbf{x}_2, t_2), \ldots, (\mathbf{x}_N, t_N)\} \text{ where } t = f + \varepsilon \text{ and } E\{\varepsilon\} = 0$$

- given D, we train an arbitrary neural network to approximate the function f by

$$y = g(\mathbf{x}, \mathbf{w})$$

The mean-squared error of this networks is:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(t_i - y_i)^2$$

To assess the effectiveness of the network, we want to know the expectation of the MSE if we test the network on arbitrarily many test points drawn from the unknown function.

$$E\{MSE\} = E\left\{\frac{1}{N}\sum_{i=1}^{N}(t_i - y_i)^2\right\} = \frac{1}{N}\sum_{i=1}^{N}E\left\{(t_i - y_i)^2\right\}$$

Let's investigate the expectation inside the sum, with a little "augmentation trick":

$$E\left\{(t_i - y_i)^2\right\} = E\left\{(t_i - f_i + f_i - y_i)^2\right\}$$

$$= E\left\{(t_i - f_i)^2\right\} + E\left\{(f_i - y_i)^2\right\} + 2E\left\{(f_i - y_i)(t_i - f_i)\right\}$$

$$= E\{\varepsilon^2\} + E\left\{(f_i - y_i)^2\right\} + 2\left(E\{f_i t_i\} - E\{f_i^2\} - E\{y_i t_i\} + E\{y_i f_i\}\right)$$

Note: $E\{f_i t_i\} = f_i^2$ since f is deterministic and $E\{t_i\} = f_i$

: $E\{f_i^2\} = f_i^2$ since f is deterministic

: $E\{y_i t_i\} = E\{y_i(f_i + \varepsilon)\} = E\{y_i f_i + y_i \varepsilon\} = E\{y_i f_i\} + 0$

: (the last term is zero because the noise in the infinite test set over which

: we take the expectation is probabilistically independent of the NN

: prediction). Thus the last term in the expectation above cancels to zero.

$$E\left\{(t_i - y_i)^2\right\} = E\{\varepsilon^2\} + E\left\{(f_i - y_i)^2\right\}$$

Thus the MSE can be decomposed in expectation into the variance of the noise and the MSE between the true function and the predicted values. This term can be further composed with the same augmentation trick as above.

$$E\left\{(f_i - y_i)^2\right\} = E\left\{(f_i - E\{y_i\} + E\{y_i\}_i - y_i)^2\right\}$$

$$= E\left\{(f_i - E\{y_i\})^2\right\} + E\left\{(E\{y_i\} - y_i)^2\right\} + 2E\left\{(E\{y_i\} - y_i)(f_i - E\{y_i\})\right\}$$

$$= bias^2 + Var\{y_i\} + 2\left(E\{f_i E\{y_i\}\} - E\left\{E\{y_i\}^2\right\} - E\{y_i f\}_i + E\{y_i E\{y_i\}\}\right)$$

Note: $E\{f_i E\{y_i\}\} = f_i E\{y_i\}$ since f is deterministic and $E\{E\{z\}\} = z$

: $E\left\{E\{y_i\}^2\right\} = E\{y_i\}^2$ since $E\{E\{z\}\} = z$

: $E\{y_i f_i\} = f_i E\{y_i\}$

: $E\{y_i E\{y_i\}\} = E\{y_i\}^2$

: Thus the last term in the expectation above cancels to zero.

$$E\left\{(f_i - y_i)^2\right\} = bias^2 + Var\{y_i\}$$

Thus the decomposition of the MSE in expectation becomes:

$$E\left\{(t_i - y_i)^2\right\} = Var\{noise\} + bias^2 + Var\{y_i\}$$

Note that the variance of the noise can not be minimized; it is independent of the neural network. Thus in order to minimize the MSE, we need to minimize both the bias and the variance. However, this is not trivial to do this. For instance, just neglecting the input data and predicting the output somehow (e.g., just a constant), would definitely minimize the variance of our predictions: they would be always the same, thus the variance would be zero—but the bias of our estimate (i.e., the amount we are off the real function) would be tremendously large. On the other hand, the neural network could perfectly interpolate the training data, i.e., it predict y=t for every data point. This will make the bias term vanish entirely, since the E(y)=f (insert this above into the squared bias term to verify this), but the variance term will become equal to the variance of the noise, which may be significant (see also Bishop Chapter 9 and the Geman et al. Paper). In general, finding an optimal bias-variance tradeoff is hard, but acceptable solutions can be found, e.g., by means of cross validation or regularization.

http://www.inf.ed.ac.uk/teaching/courses/mlsc/Notes/Lecture4/BiasVariance.pdf

Bias-variance decomposition (2)

- Putting everything together, we have:

$$
\begin{aligned}
E_P\left[(y - h(\mathbf{x}))^2\right] &= E_P\left[(h(\mathbf{x}) - \bar{h}(\mathbf{x}))^2\right] + \\
&\quad \bar{h}(\mathbf{x})^2 - 2f(\mathbf{x})\bar{h}(\mathbf{x}) + f(\mathbf{x})^2 + \\
&\quad E_P\left[(y - f(\mathbf{x}))^2\right] \\
&= E_P\left[(h(\mathbf{x}) - \bar{h}(\mathbf{x}))^2\right] + \qquad \text{(variance)} \\
&\quad (h(\mathbf{x}) - f(\mathbf{x}))^2 + \qquad \text{(bias)}^2 \\
&\quad E_P\left[(y - f(\mathbf{x}))^2\right] \qquad \text{(noise)} \\
&= \text{Var}[h(\mathbf{x})] + \text{Bias}[h(\mathbf{x})]^2 + E_P[\varepsilon^2] \\
&= \text{Var}[h(\mathbf{x})] + \text{Bias}[h(\mathbf{x})]^2 + \sigma^2
\end{aligned}
$$

$h_D(x^*)$ model prediction (assume 20 training datesets)
h(x*) Average model prediction
f(x*) TRUE (Actual function value)
Y* Observed target data (noisy)

- Expected prediction error = Variance + Bias^2 + Noise^2

Estimating Bias and Variance (continued)

- For each data point **x**, we will now have the observed corresponding value y and several predictions $y_1, \ldots, y_K$.
- Compute the average prediction h.
- Estimate **bias** as $(h - y)$
- Estimate **variance** as $\Sigma_k (y_k - h)^2/(K - 1)$
- Assume noise is 0

http://www-scf.usc.edu/~csci567/17-18-bias-variance.pdf

NOTATION: Bias, Variance, and Noise

- **Using a test data set with 20 data points**
  - Denoted as X*, y*
- **For each data point x* in the test data set compute variance over the variance predictions (50 models give 50 predictions for each data point x*).**
- **For each data point x* calculate**
  - **Variance**: $E[(h(x^*) - \underline{h(x^*)})^2]$ ## $\Sigma(h(x^*) - \underline{h(x^*)})^2/50$
    - Describes how much h(x*) varies from one training set S to another
  - **Bias**: $[\underline{h(x^*)} - f(x^*)]$
    - Describes the <u>average</u> error of h(x*).
  - **Noise**: $E[(y^* - f(x^*))^2] = E[\varepsilon^2] = \sigma^2$
    - Describes how much y* varies from f(x*)

$h_D(x^*)$ model prediction (assume 20 training datesets)
<u>h(x*) Average model prediction</u>
f(x*) TRUE (Actual function value)
Y* Observed target data (noisy)

$\underline{h(x^*)} - f(x^*)]$
ce: $E[(h(x^*) - \underline{h(x^*)})^2]$
: $E[(y^* - f(x^*))^2] = E[\varepsilon^2] = \sigma^2$

Bias-Variance written more formally for a single test point x*, using say 20 models

Using a SINGLE **test** data set with 20 data points sum $(h(x^*) - y^*)^2$ over each of the 20 **observed points** and take the average (train multiple models)

$$E_D[(h(x^*) - y^*)^2] =$$

Expected MSE wrt different models that are learned from different datasets D. E.g., 20 models yield 20 predictions $h_D(x^*)$

$$= E_D[(h_D(x^*) - \underline{h(x^*)})^2] + \textit{Variance} +$$

$$(\underline{h(x^*)} - f(x^*))^2 + \quad \textit{Bias}^2 +$$

$$E[(y^* - f(x^*))^2] \quad \textit{Noise}^2$$

$$= Var(h(x^*)) + Bias(h(x^*))^2 + E[\varepsilon^2]$$

$$= Var(h(x^*)) + Bias(h(x^*))^2 + \sigma^2$$

Expected prediction error = Variance + Bias^2 + Noise^2

Estimating Bias and Variance (continued)

- For each data point **x**, we will now have the observed corresponding value y and several predictions $y_1, \ldots, y_K$.
- Compute the average prediction $\underline{h}$.
- Estimate **bias** as $(\underline{h} - y)$
- Estimate **variance** as $\Sigma_k (y_k - \underline{h})^2/(K-1)$
- Assume noise is 0

$h_D(x^*)$ model prediction (assume 20 training datesets

$\underline{h(x^*)}$ Average model prediction

$f(x^*)$ TRUE (Actual function value)

Y^* Observed target data (noisy)

http://www-scf.usc.edu/~csci567/17-18-bias-variance.pdf

Excellent Slides from Sofus

Bias-variance trade-off

- Consider fitting a logistic regression LTU to a data set vs. fitting a large neural net.
- Which one do you expect to have higher bias? Higher variance?
- Typically, *bias* comes from not having good hypotheses in the considered class
- *Variance* results from the hypothesis class containing too many hypotheses
- Hence, we are faced with a *trade-off:* choose a more expressive class of hypotheses, which will generate higher variance, or a less expressive class, which will generate higher bias.

Source of bias

- Inability to represent certain decision boundaries
  - E.g., linear threshold units, naïve Bayes, decision trees
- Incorrect assumptions
  - E.g, failure of independence assumption in naïve Bayes
- Classifiers that are "too global" (or, sometimes, too smooth)
  - E.g., a single linear separator, a small decision tree.

If the bias is high, the model is *underfitting* the data.

Source of variance

- Statistical sources
  - Classifiers that are "too local" and can easily fit the data
    - E.g., nearest neighbor, large decision trees
- Computational sources
  - Making decision based on small subsets of the data
    - E.g., decision tree splits near the leaves
  - Randomization in the learning algorithm
    - E.g., neural nets with random initial weights
  - Learning algorithms that make sharp decisions can be unstable (e.g. the decision boundary can change if one training example changes)

If the variance is high, the model is *overfitting* the data

Live Session Outline

- **Machine Learning Introduction**
- **Equation of a line**
- **Linear Regresion**
- **Bias-Variance**
  - Expected Value, Variance
  - Bias-Variance
  - Coding it up
    - For a simulated world (lab setting)
    - For a real world problems
  - Example using bagging

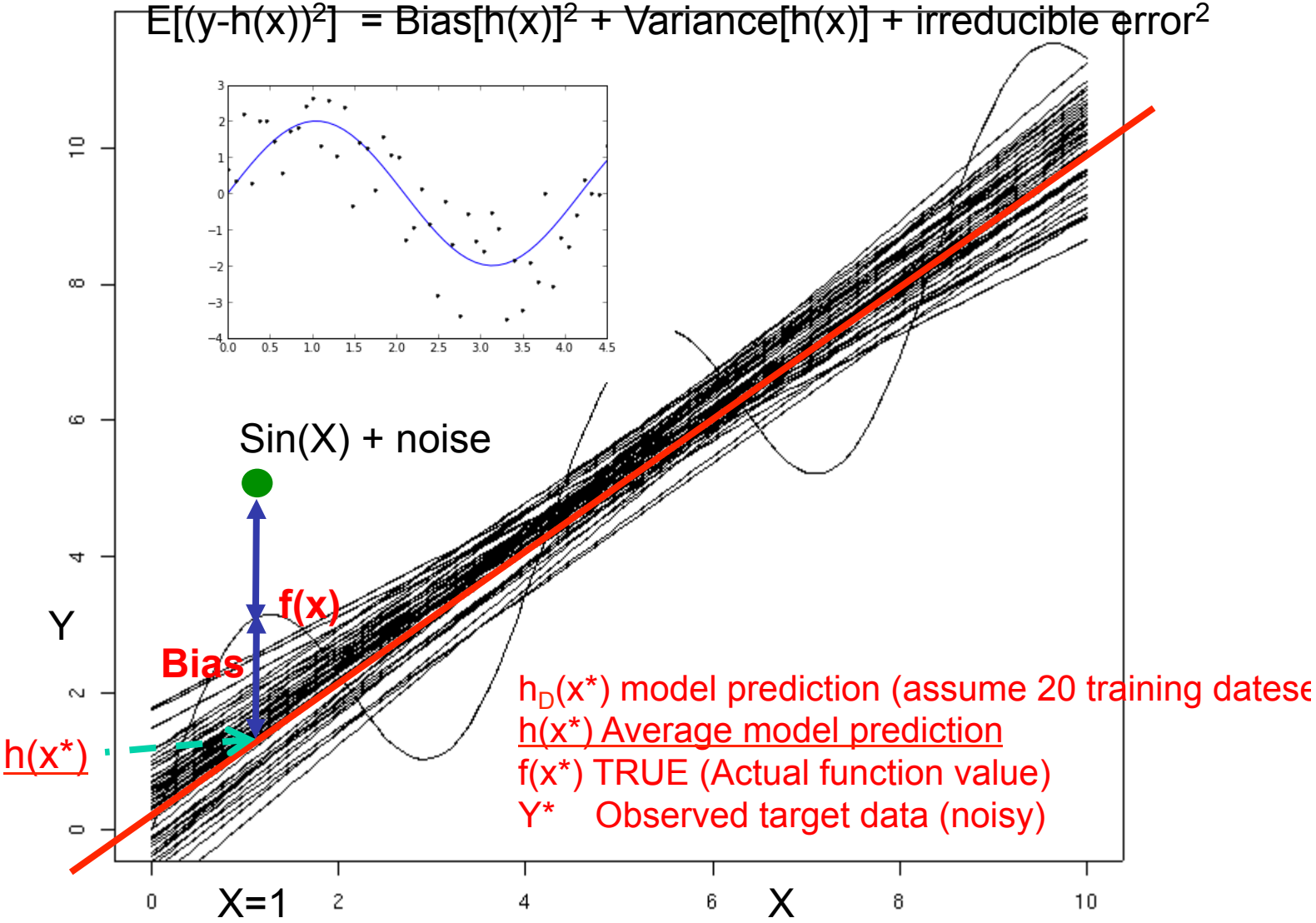
Estimating Bias-Variance: 2 Cases

- **Case 1: Simulated world**
  - Estimating Bias and Variance in a simulated world where we know the target function
- **Case 2: Real world**
  - Estimating Bias and Variance in the real world where we do NOT know the target function

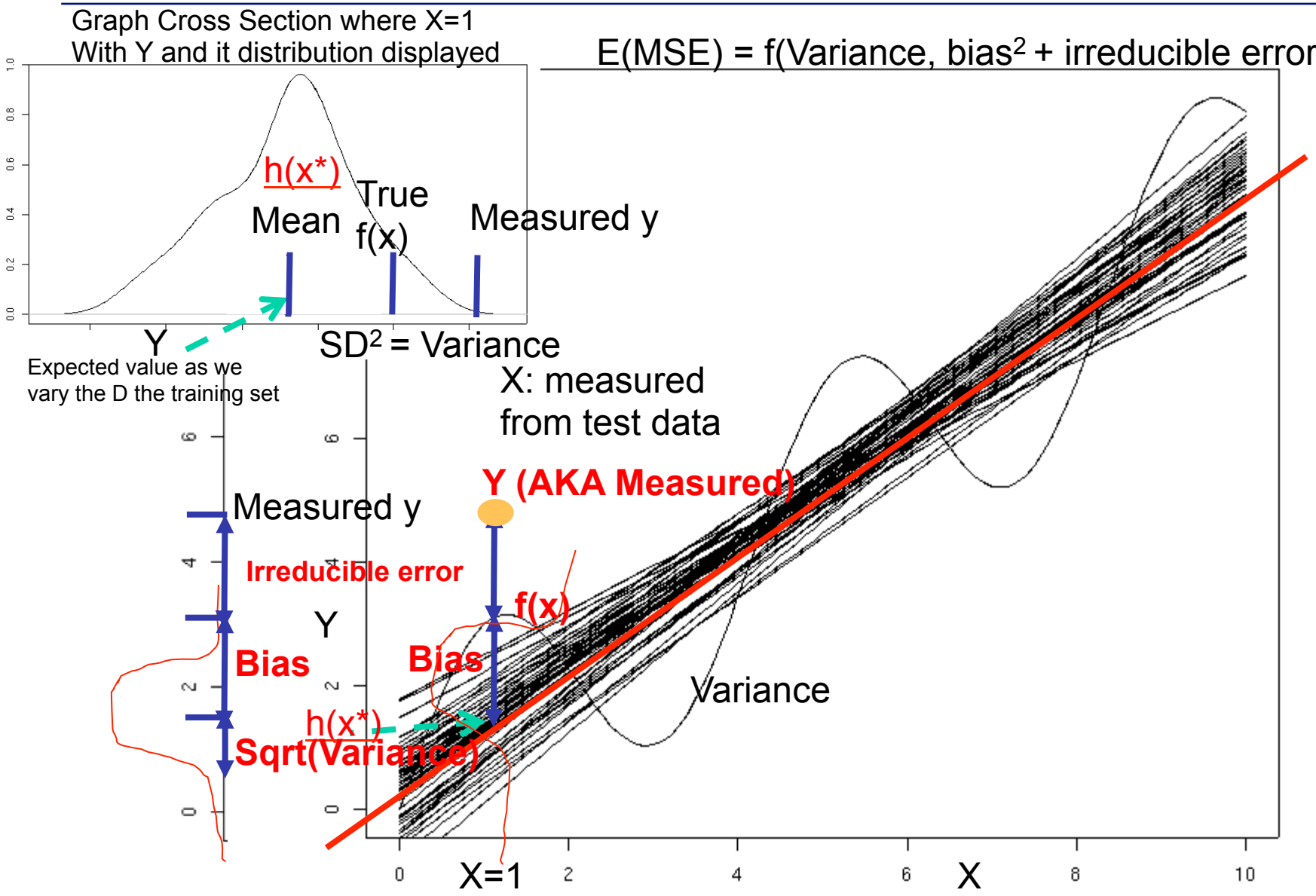
50 linear models (using different samples)

E(SE) = f(Variance, bias^2 + irreducible error^2)

E[(y-h(x))^2] = Bias[h(x)]^2 + Variance[h(x)] + irreducible error^2

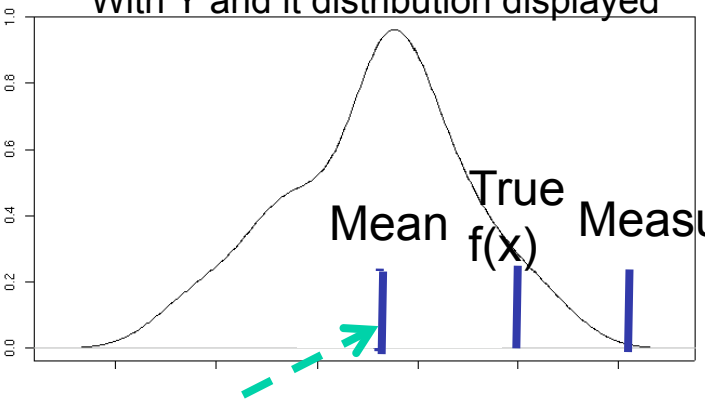


50 linear models (using different samples)



50 linear models (using different samples)

Graph Cross Section where X=1
With Y and it distribution displayed



E(MSE) = f(Variance, bias^2 + irreducible error)

~~$E[(y-h(x))^2] = Bias[h(x)]^2 + Variance[h(x)] + irreducible\ er$~~

Expected value
as we vary the D
the training set

X: measured
from test data

$h_D(x^*)$ model prediction (assume 20 training datese
h(x*) Average model prediction
f(x*) TRUE (Actual function value)
Y* Observed target data (noisy)

bias as $(h - y)$

variance as $\Sigma_k (y_k - h)^2/(K-1)$

Case 1: Simulated world

For each model config
For 1:n

- Generate a train set
- Build a model

Observed TEST Dataset			prediction for X given the model generated from sample i					
X	Yobs=f(X)+noise	Ytrue=f(x)	h_star1	…	h_star_50	var	bias^2	Noise^2
1.2								
3.4								
N								

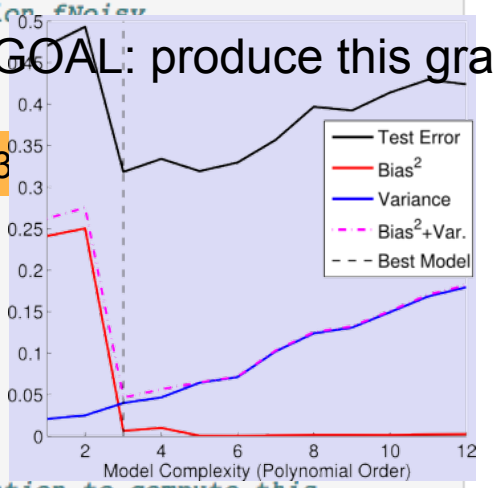
Estimating Bias and Variance in a simulated world where we know the target function

Bias: [h(x*) – f(x*)]
Variance: E[(h(x*) – h(x*))^2]
Noise : E[(y* – f(x*))^2] = E[ε^2] = σ^2
h_D(x*) model prediction (assume 20 training datesets)
h(x*) Average model prediction
f(x*) TRUE (Actual function value)
Y* Observed target data (noisy)

```
# CASE 1 in a simulated world where we know the ground truth
#HW1.0.1 Pseudocode
# Given
# A training data samples of the form (X, yObserved) that is generated from a noisy function fNoisy
# and a test set has the form X, y_true in our artificial world that is generated from
# Running our simulation generates a new column of predictions for each row of the form
#     X, y_true, h_star1, ...h_star_n,
#
for model in models:
    #this is the bagging step needed to calculate variance
    #where n is some constant (like 50)
    for iteration from 1:n
        generate a sample of data from our noisy function
        Train model using train_data
        h_star=predict results for test_data
    h_bar=calculate average prediction across all iterations
    #y_true is the vector of true classes in the test_data
    bias=h_bar-y_true
    variance=sum((h_bar-h_star)^2)/n #in practice, one would need to go through each iteration to compute this
    noise=mean((y_true-h_star)^2) #As with variance, this needs to be calculated across all iterations

choose model that minimizes (bias^2+variance)
```

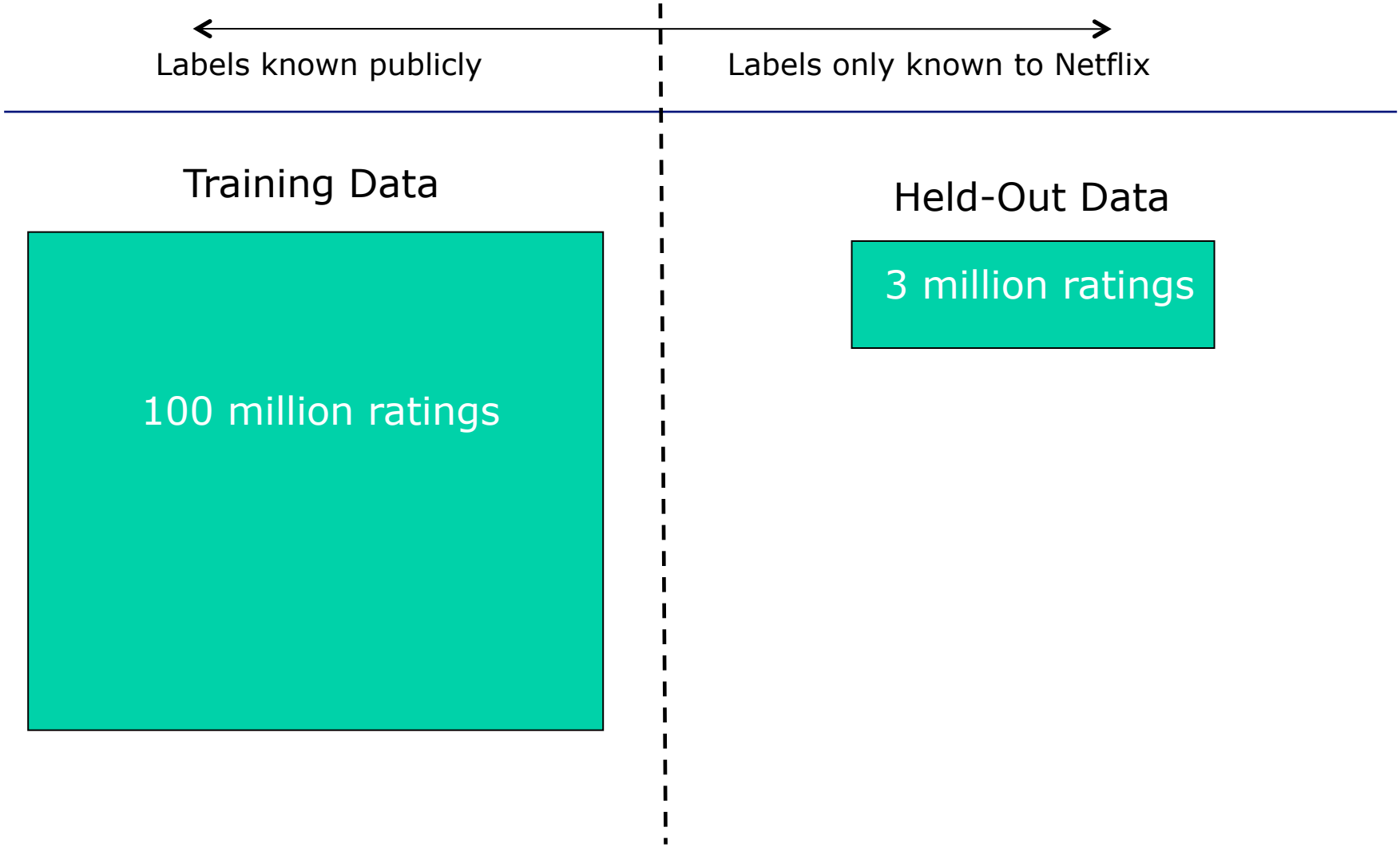
Models E.g., linear regression, polynomial regression degree 2, 3

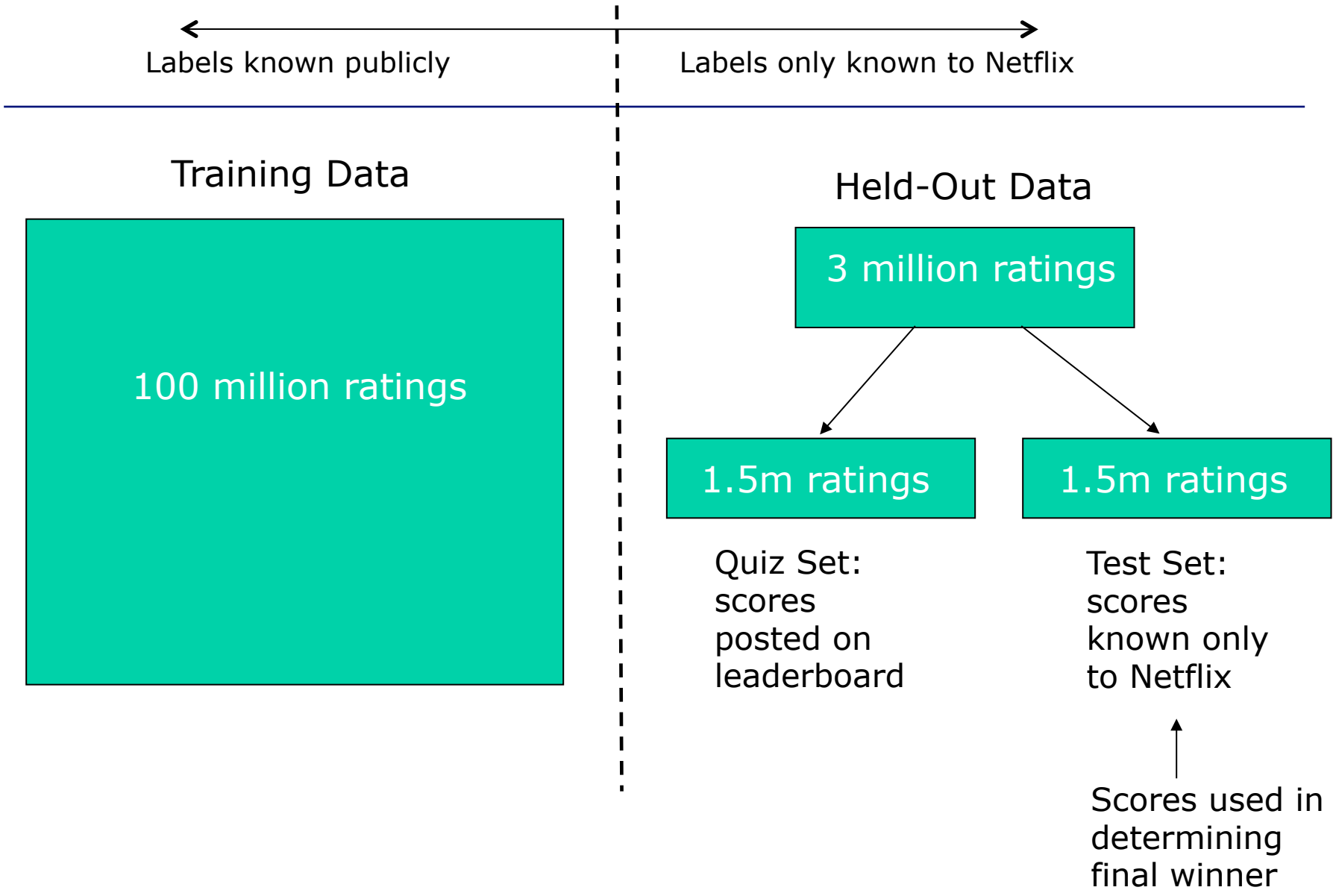
GOAL: produce this gra



Live Session Outline

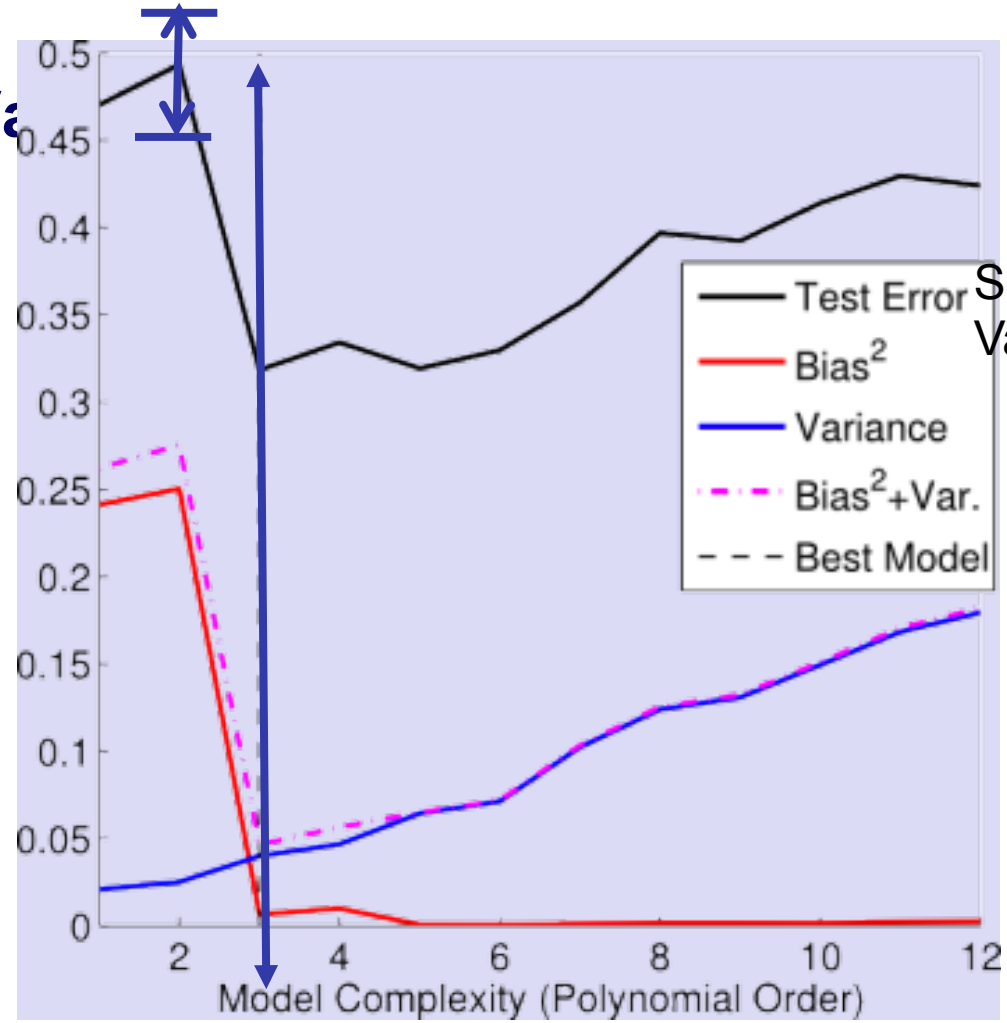
- **Machine Learning Introduction**
- **Equation of a line**
- **Linear Regresion**
- **Bias-Variance**
  - Expected Value, Variance
  - Bias-Variance
  - Coding it up
    - For a simulated world (lab setting)
    - For a real world problems
  - Example using bagging


Labels known publicly
Labels only known to Netflix
Training Data
100 million ratings
Held-Out Data
3 million ratings


Labels known publicly
Labels only known to Netflix
Training Data
100 million ratings
Held-Out Data
3 million ratings
1.5m ratings
1.5m ratings
Quiz Set: scores posted on leaderboard
Test Set: scores known only to Netflix
Scores used in determining final winner

Real World

- .. Va



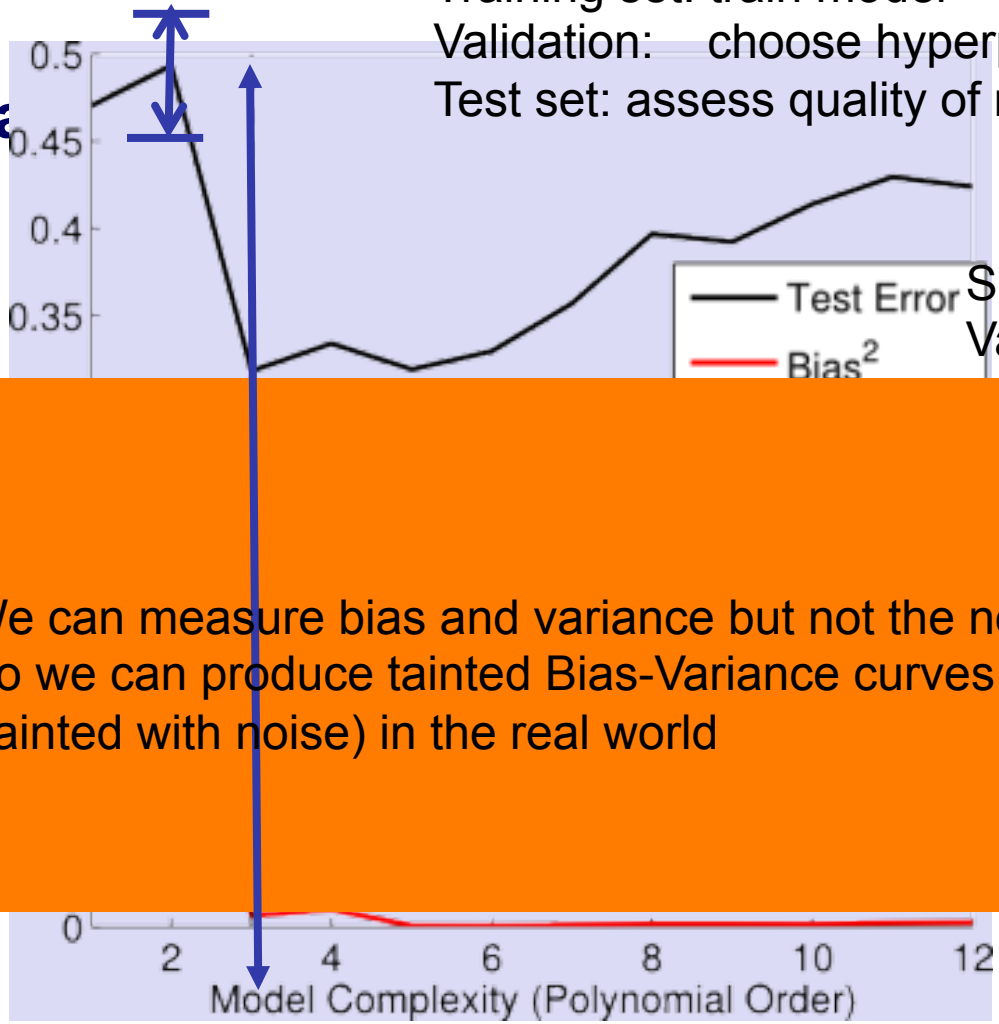
Should be Validation error

Use Cross fold validation for model setup to get better estimate of the MSE
We NEVER use the TEST set for decision making

Real World

Training set: train model
Validation: choose hyperparameters/model
Test set: assess quality of model

- .. Va

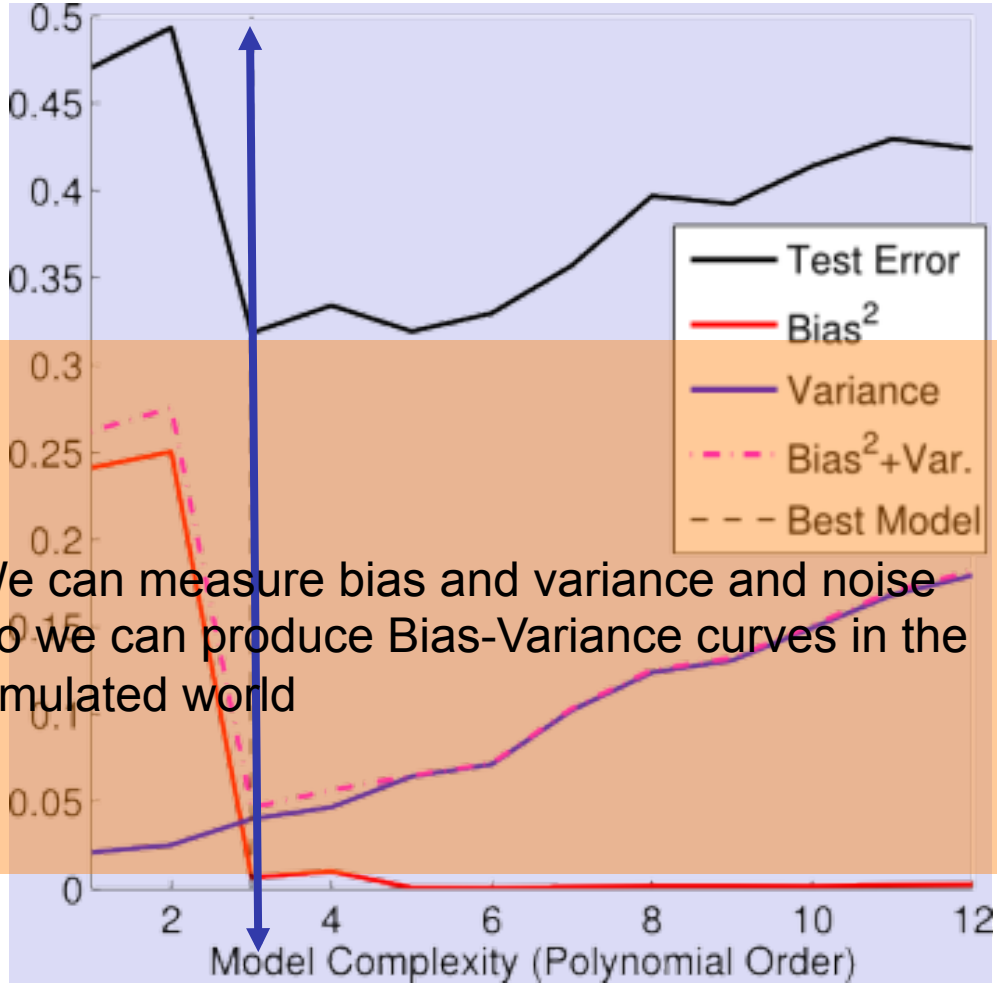


Should be Validation error

We can measure bias and variance but not the noise
So we can produce tainted Bias-Variance curves (tainted with noise) in the real world

Simulated world: we know f(x)

- ..



We can measure bias and variance and noise
So we can produce Bias-Variance curves in the simulated world

Measuring Bias and Variance in Practice

- **Measuring Bias and Variance in Practice**
- **There is no true function available so improvise and estimate bias and variance as follows:**

- **Bootstrapping**
  - Get multiple bootstrap replicates (see next slides for details)
  - Assume 1000 datapoints; sample with replacement 1000 time to generate a bootstrap sample

- **Alternative to boostrapping**
  - If multiple values of y for the same X value
  - Or bucket X values

Measuring Bias and Variance

- In practice (unlike in theory), we have only ONE training set S.
- We can simulate multiple training sets by bootstrap replicates
  - $S' = \{\mathbf{x} \mid \mathbf{x}$ is drawn at random with replacement from $S\}$ and $|S'| = |S|$.

Procedure for Measuring Bias and Variance

- Construct B bootstrap replicates of S (e.g., $B = 200$): $S_1, \ldots, S_B$
- Apply learning algorithm to each replicate S_b to obtain hypothesis h_b
- Let $T_b = S \setminus S_b$ be the data points that do not appear in S_b (out of bag points)
- Compute predicted value $h_b(\mathbf{x})$ for $\mathbf{x} \in T_b$

K times X is in the test set T_b

Estimating Bias and Variance (continued)

- For each data point **x**, we will now have the observed corresponding value y and several predictions $y_1, \ldots, y_K$.
- Compute the average prediction h.
- Estimate **bias** as $(h - y)$
- Estimate **variance** as $\Sigma_k (y_k - h)^2/(K - 1)$
- Assume noise is 0

K times X is in the test set T_b
Assume y the observed value is the true valu
I.e., assume Zero noise

Case 2: Bias Variance calculation in practice

Observed TEST Dataset			prediction for X given the model generated from sample i					
X	Yobs=f(X)+noise	Ytrue=f(x) yObserved	h_star1	…	h_star_50	var	bias^2	Noise^2
1.2								0
3.4								0
								0
N								0

```
#CASE 2 in the real world when we do NOT know the true target function

#HW1.0.1 Pseudocode
# Given training data of the form (X, yObserved)
# y_true is true value but in most real world problems we do NOT know the ground truth
# So in practice we set y_true =  yObserved and set noise =0
for model in models:
    #this is the bagging step needed to calculate variance
    #where n is some constant (like 50)
    for iteration from 1:n
        Split training data randomly into train_data and test_data
        Train model using train_data
        h_star=predict results for test_data
    h_bar=calculate average prediction across all iterations
    #y_true is the vector of true classes in the test_data
    bias=h_bar-y_true
    variance=sum((h_bar-h_star)^2)/n #in practice, one would need to go through each iteration to compute this
    #noise=mean((y_true-h_star)^2) #As with variance, this needs to be calculated across all iterations
    noise=0 #set the noise =0 since we can NOT estimate it
choose model that minimizes (bias^2+variance)
```

Yobservered, yPredictions(multiple predictions)

Expected prediction error* = *estimator variance* + *squared estimator bias* + *noise

Thus the expected prediction error on new data can be used as a quantitative criterion for selecting the best model from a candidate set of estimators! It turns out that, given N new data points $(\mathbf{x}^*, \mathbf{y}^*)$, the expected prediction error can be easily approximated as the mean squared error over data pairs:

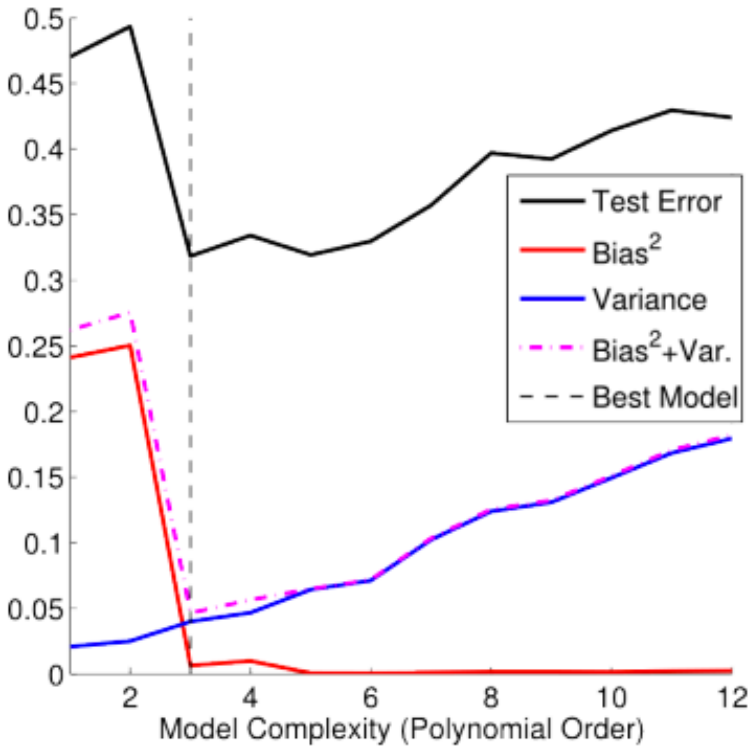
$$\mathbb{E}[(g(\mathbf{x}^*) - \mathbf{y}^*)^2] \approx \frac{1}{N}\sum_{i=1}^{N}(g(x_i^*) - y_i^*)^2$$

Below we demonstrate these findings with another set of simulations. We simulate 100 independent datasets, each with 25 xy pairs. We then partition each dataset into two non-overlapping sets: a training set using for fitting model parameters, and a testing set used to calculate prediction error. We then fit the parameters for estimators of varying complexity. Complexity is varied by using polynomial functions that range in model order from 1 (least complex) to 12 (most complex). We then calculate and display the squared bias, variance, and error on testing set for each of the estimators:

```
+ expand source
```

Real world

Simulated world



Case 2: Bias Variance calculation in practice

```
#CASE 2 in the real world when we do NOT know the true target function

#HW1.0.1 Pseudocode
# Given training data of the form (X, yObserved)
# y_true is true value but in most real world problems we do NOT know the ground truth
# So in practice we set y_true =  yObserved and set noise =0
for model in models:
    #this is the bagging step needed to calculate variance
    #where n is some constant (like 50)
    for iteration from 1:n
        Split training data randomly into train_data and test_data
        Train model using train_data
        h_star=predict results for test_data
    h_bar=calculate average prediction across all iterations
    #y_true is the vector of true classes in the test_data
    bias=h_bar-y_true
    variance=sum((h_bar-h_star)^2)/n #in practice, one would need to go through each iteration to compute this
    #noise=mean((y_true-h_star)^2) #As with variance, this needs to be calculated across all iterations
    noise=0 #set the noise =0 since we can NOT estimate it
choose model that minimizes (bias^2+variance)
```

Assume noise is zero

Yobservered, yPredictions(multiple predictions)

Expected prediction error = ***estimator variance*** + ***squared estimator bias*** + ***noise***

Thus the expected prediction error on new data can be used as a quantitative criterion for selecting the best model from a candidate set of estimators! It turns out that, given N new data points $(\mathbf{x}^*, \mathbf{y}^*)$, the expected prediction error can be easily approximated as the mean squared error over data pairs:

$$\mathbb{E}[(g(\mathbf{x}^*) - \mathbf{y}^*)^2] \approx \frac{1}{N}\sum_{i=1}^{N}(g(x_i^*) - y_i^*)^2$$

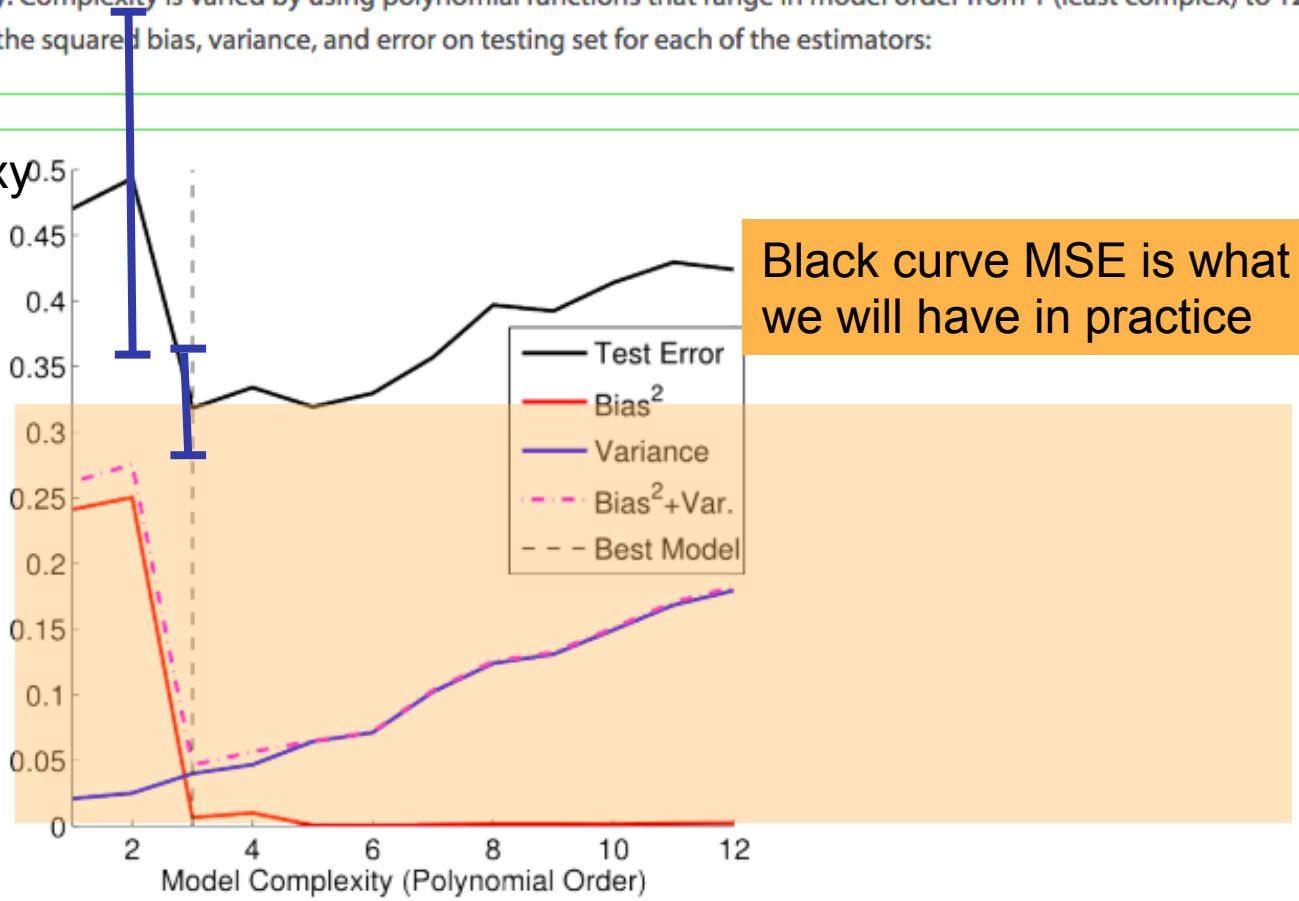
Below we demonstrate these findings with another set of simulations. We simulate 100 independent datasets, each with 25 xy pairs. We then partition each dataset into two non-overlapping sets: a training set using for fitting model parameters, and a testing set used to calculate prediction error. We then fit the parameters for estimators of varying complexity. Complexity is varied by using polynomial functions that range in model order from 1 (least complex) to 12 (most complex). We then calculate and display the squared bias, variance, and error on testing set for each of the estimators:

```
+ expand source
```

Black curve is a good proxy
Bias-variance decomp.

Cross fold validation
Gives us mean and std

Assume TRUE function is available



Black curve MSE is what we will have in practice

Expected prediction error = ***estimator variance*** + ***squared estimator bias*** + ***noise***

Thus the expected prediction error on new data can be used as a quantitative criterion for selecting the best model from a candidate set of estimators! It turns out that, given N new data points $(\mathbf{x}^*, \mathbf{y}^*)$, the expected prediction error can be easily approximated as the mean squared error over data pairs:

$$\mathbb{E}[(g(\mathbf{x}^*) - \mathbf{y}^*)^2] \approx \frac{1}{N}\sum_{i=1}^{N}(g(x_i^*) - y_i^*)^2$$

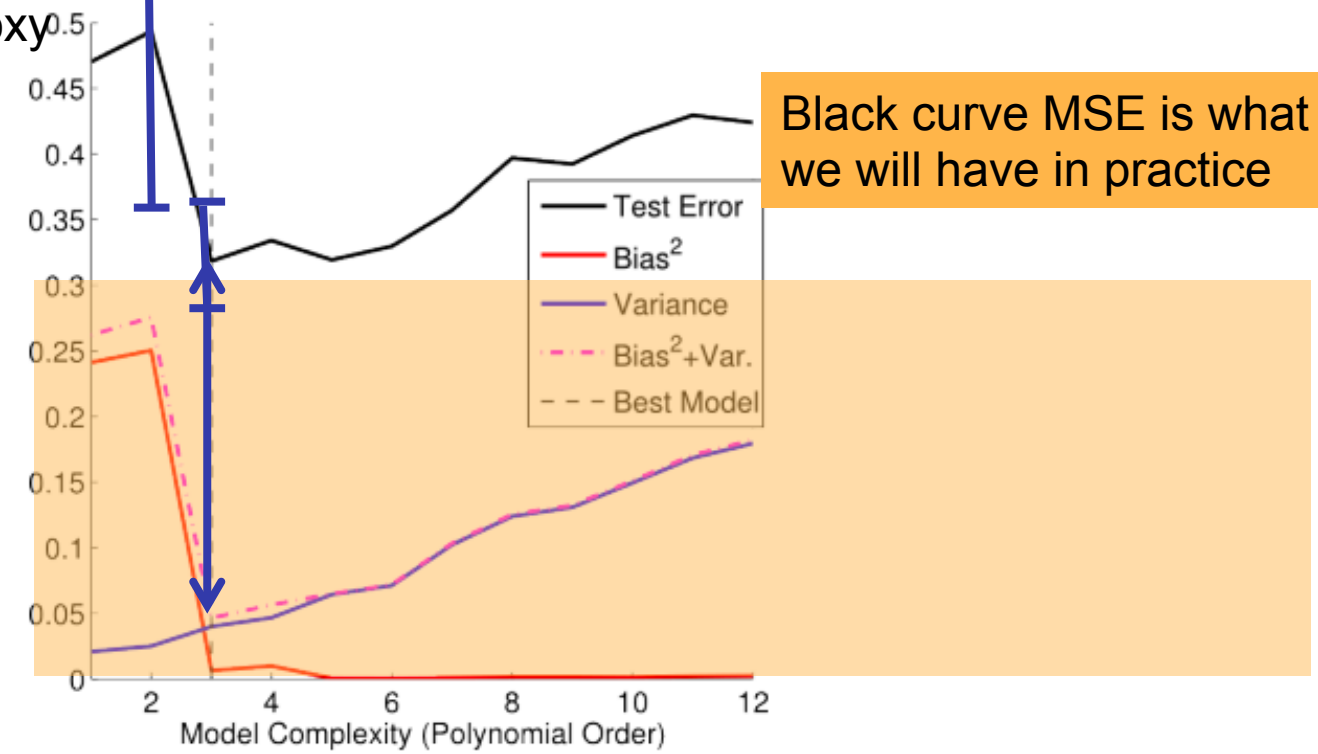
Below we demonstrate these findings with another set of simulations. We simulate 100 independent datasets, each with 25 xy pairs. We then partition each dataset into two non-overlapping sets: a training set using for fitting model parameters, and a testing set used to calculate prediction error. We then fit the parameters for estimators of varying complexity. Complexity is varied by using polynomial functions that range in model order from 1 (least complex) to 12 (most complex). We then calculate and display the squared bias, variance, and error on testing set for each of the estimators:

```
+ expand source
```

Black curve is a good proxy
Bias-variance decomp.

Cross fold validation
Gives us mean and std

Assume TRUE function is available



Black curve MSE is what we will have in practice

In this example, we highlight the best estimator in terms of prediction error on the testing set (black curve) with a dashed black vertical line. The best estimator corresponds to a polynomial model of order of N=3.

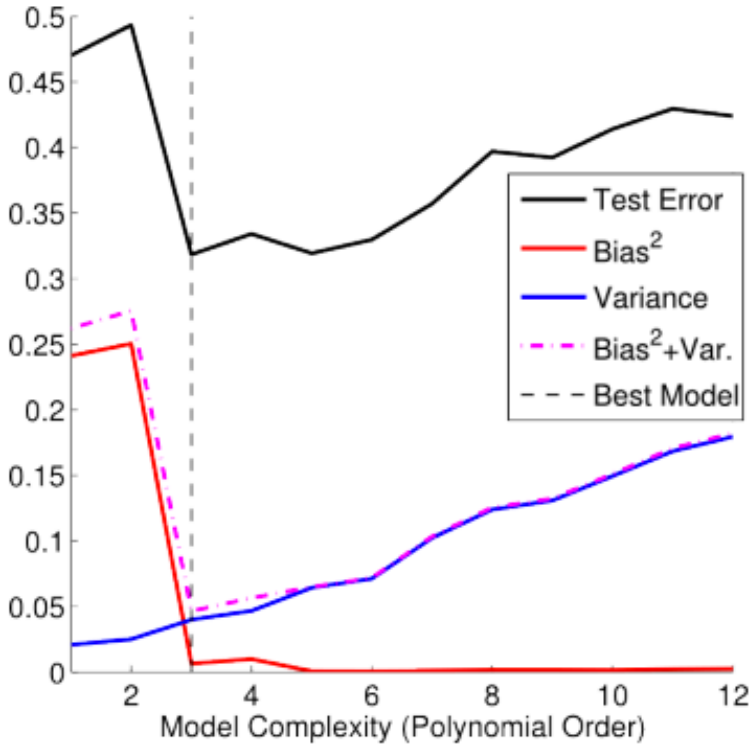
(most complex). We then calculate and display the squared bias, variance, and error on testing set for each of the estimators:

```
+ expand source
```

In Practice

It turns out that, given N new data points, the expected prediction error E(MSE) can be easily approximated as the mean squared error over data pairs (novel test data), i.e., Black curve MSE is what we will have in practice

$$\mathbb{E}[(g(\mathbf{x}^*) - \mathbf{y}^*)^2] \approx \frac{1}{N}\sum_{i=1}^{N}(g(x_i^*) - y_i^*)^2$$



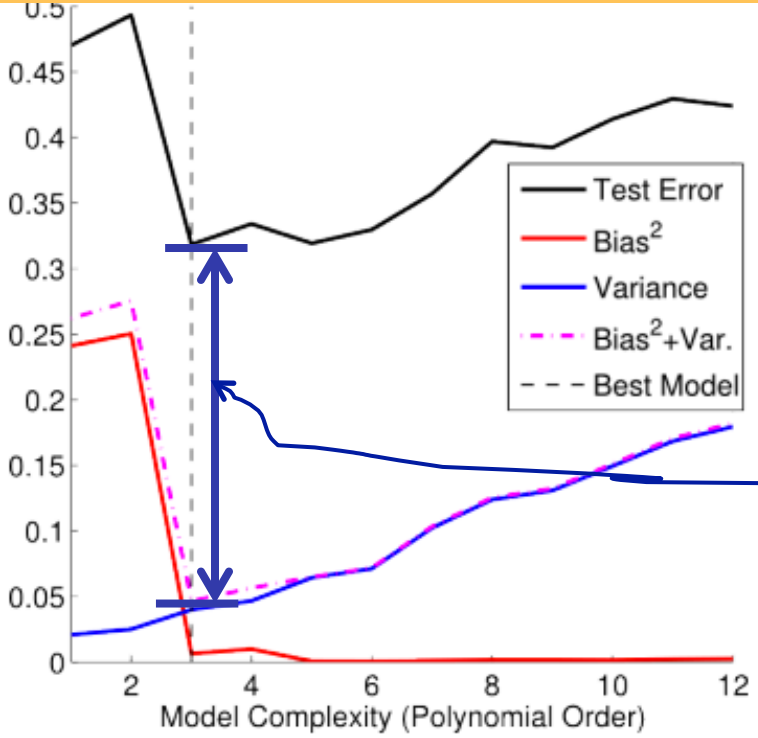
In this example, we highlight the best estimator in terms of prediction error on the testing set (black curve) with a dashed black vertical line. The best estimator corresponds to a polynomial model of order of N=3.
Notice that the vertical black line is located where function defined by the sum of the squared bias and variance (dashed magenta curve) is also at a minimum.
Notice also how the sum of the squared bias and variance also has the same shape as curve defined by the prediction error on the testing set. This exemplifies how the error on novel data can be used as a proxy for determining the best estimator from a candidate set based on squared bias and variance. The noise term in Bias-Variance equation is also represented in the figure by the vertical displacement between the **black curve** and **dashed magenta curve**.

It is very important to point out that all of these results are based on evaluating prediction error on novel data, not used to estimate model parameters.

In Practice

It turns out that, given N new data points, the expected prediction error E(MSE) can be easily approximated as the mean squared error over data pairs (novel test data):

$$\mathbb{E}[(g(\mathbf{x}^*) - \mathbf{y}^*)^2] \approx \frac{1}{N}\sum_{i=1}^{N}(g(x_i^*) - y_i^*)^2$$



noise term (
Variance equ
the vertical d
between the
and **dashe**
magenta

Approximate expected prediction error E(MSE) with mean squared error over data pairs (novel test data)

Expected prediction error = ***estimator variance*** + ***squared estimator bias*** + ***noise***

Thus the expected prediction error on new data can be used as a quantitative criterion for selecting the best model from a candidate set of estimators! It turns out that, given N new data points $(\mathbf{x}^*, \mathbf{y}^*)$, the expected prediction error can be easily approximated as the mean squared error over data pairs:

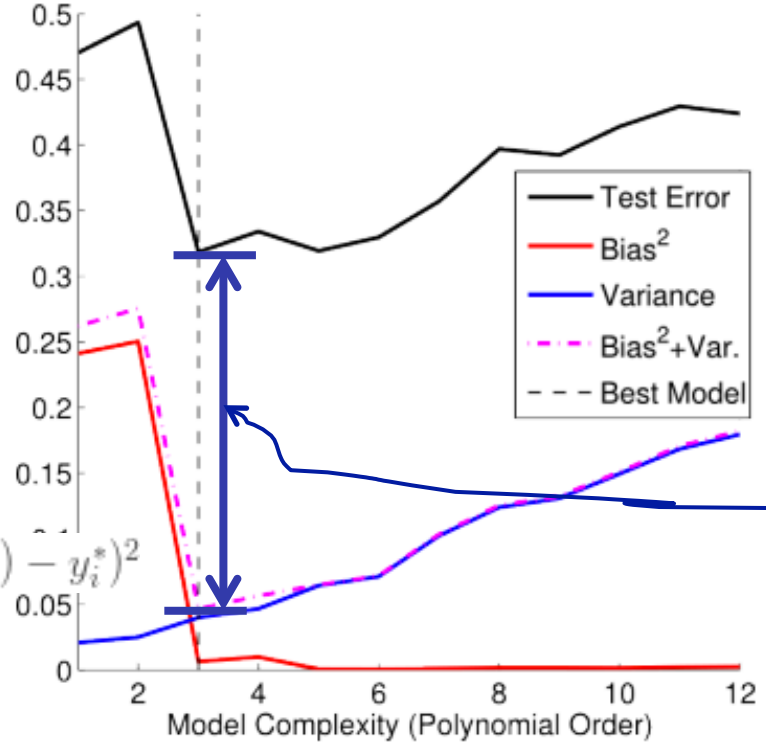
- $\mathbb{E}[(g(\mathbf{x}^*) - \mathbf{y}^*)^2] \approx \frac{1}{N}\sum_{i=1}^{N}(g(x_i^*) - y_i^*)^2$

Below we demonstrate these findings with another set of simulations. We simulate 100 independent datasets, each with 25 xy pairs. We then partition each dataset into two non-overlapping sets: a training set using for fitting model parameters, and a testing set used to calculate prediction error. We then fit the parameters for estimators of varying complexity. Complexity is varied by using polynomial functions that range in model order from 1 (least complex) to 12 (most complex). We then calculate and display the squared bias, variance, and error on testing set for each of the estimators:

```
+ expand source
```

In Practice It turns out that, given N new data points, the expected prediction error E(MSE) can be easily approximated as the mean squared error over data pairs (novel test data):

$\mathbb{E}[(g(\mathbf{x}^*) - \mathbf{y}^*)^2] \approx \frac{1}{N}\sum_{i=1}^{N}(g(x_i^*) - y_i^*)^2$



noise term in Bias-Variance equation the vertical displacement between the **black curve** and **dashed magenta curve**.

Approximations in this Procedure

- Bootstrap replicates are not real data
- We ignore the noise
  - If we have multiple data points with the same **x** value, then we can estimate the noise
  - We can also estimate noise by pooling y values from nearby **x** values

Python: Notebook for Bias Variance

- http://nbviewer.jupyter.org/urls/dl.dropbox.com/s/e391g01ruhj4kai/Bias%20and%20Variance.ipynb

- https://www.dropbox.com/s/e391g01ruhj4kai/Bias%20and%20Variance.ipynb?dl=0

Bias and Variance

Machine learning class support material, Universidad Nacional de Colombia, 2013

The purpose of this notebook is to illustrate the bias-variance trade-off when learning regression models from data. We will use and example based on non-linear regression presented in Chapter 4 of [Alpaydin10].

Training data generation

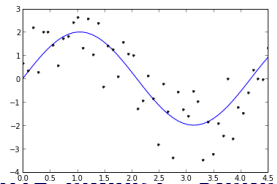
First we will write a function to generate a random sample. The data generation model is the following:

$$r(x) = f(x) + \epsilon$$

with $\epsilon \sim \mathcal{N}(0, 1)$

```
import numpy as np
import pylab as pl

def f(size):
    '''
    Returns a sample with 'size' instances without noise.
    '''
    x = np.linspace(0, 4.5, size)
    y = 2 * np.sin(x * 1.5)
    return (x,y)

def sample(size):
    '''
    Returns a sample with 'size' instances.
    '''
    x = np.linspace(0, 4.5, size)
    y = 2 * np.sin(x * 1.5) + pl.randn(x.size)
    return (x,y)

pl.clf()
f_x, f_y = f(50)
pl.plot(f_x, f_y)
x, y = sample(50)
pl.plot(x, y, 'k.')
```

```
[<matplotlib.lines.Line2D at 0x10731a5d0>]
```



Bias and Variance

Machine learning class support material, Universidad Nacional de Colombia, 2013

The purpose of this notebook is to illustrate the bias-variance trade-off when learning regression models from data. We will use and example based on non-linear regression presented in Chapter 4 of [Alpaydin10].

Training data generation

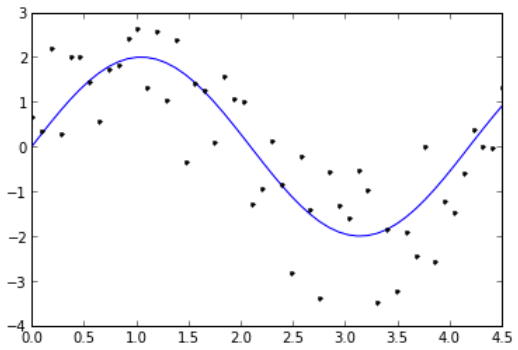
First we will write a function to generate a random sample. The data generation model is the following:

$$r(x) = f(x) + \epsilon$$

with $\epsilon \sim \mathcal{N}(0, 1)$

In [5]:
```
import numpy as np
import pylab as pl

def f(size):
    '''
    Returns a sample with 'size' instances without noise.
    '''
    x = np.linspace(0, 4.5, size)
    y = 2 * np.sin(x * 1.5)
    return (x,y)

def sample(size):
    '''
    Returns a sample with 'size' instances.
    '''
    x = np.linspace(0, 4.5, size)
    y = 2 * np.sin(x * 1.5) + pl.randn(x.size)
    return (x,y)

pl.clf()
f_x, f_y = f(50)
pl.plot(f_x, f_y)
x, y = sample(50)
pl.plot(x, y, 'k.')
```

Out[5]:
```
[<matplotlib.lines.Line2D at 0x10731a5d0>]
```



Model fitting

We will use least square regression (LSR) to fit a polynomial to the data. Actually, we will use multivariate linear regression, over a dataset built in the following way:

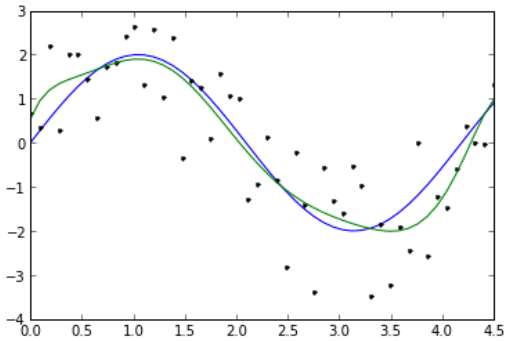
For each sample x_i we build a vector $(1, x_i, x_i^2, \ldots, x_i^n)$ and we use LSR to fit a function $g : \mathbb{R}^{n+1} \to \mathbb{R}$ to the training data.

```
# This illustrates how vander function works:
x1 = np.array([1,2,3])
print np.vander(x1, 4)
```

```
[[ 1  1  1  1]
 [ 8  4  2  1]
 [27  9  3  1]]
```

```
from sklearn.linear_model import LinearRegression

def fit_polynomial(x, y, degree):
    '''
    Fits a polynomial to the input sample.
    (x,y): input sample
    degree: polynomial degree
    '''
    model = LinearRegression()
    model.fit(np.vander(x, degree + 1), y)
    return model

def apply_polynomial(model, x):
    '''
    Evaluates a linear regression model in an input sample
    model: linear regression model
    x: input sample
    '''
    degree = model.coef_.size - 1
    y = model.predict(np.vander(x, degree + 1))
    return y

model = fit_polynomial(x, y, 8)
p_y = apply_polynomial(model, x)
pl.plot(f_x, f_y)
pl.plot(x, y, 'k.')
pl.plot(x, p_y)
```

```
[<matplotlib.lines.Line2D at 0x1075d9bd0>]
```

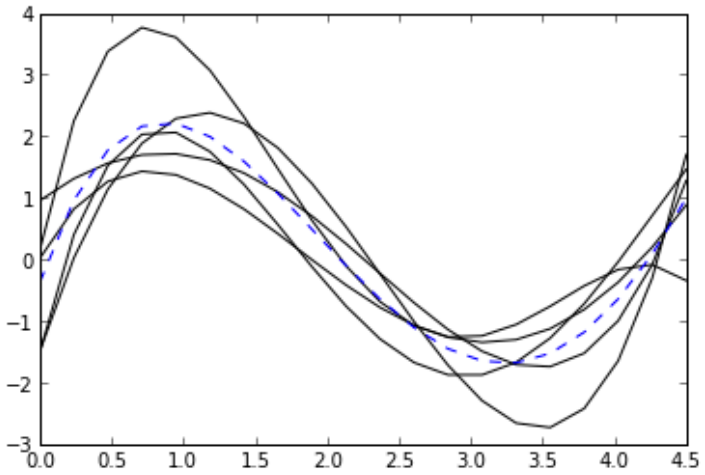


Model averaging

The following code generates a set of samples of the same size and fits a poynomial to each sample. Then the average model is calculated. All the models, including the average model, are plotted.

```
degree = 4
n_samples = 20
n_models = 5
avg_y = np.zeros(n_samples)
for i in xrange(n_models):
    (x,y) = sample(n_samples)
    model = fit_polynomial(x, y, degree)
    p_y = apply_polynomial(model, x)
    avg_y = avg_y + p_y
    pl.plot(x, p_y, 'k-')
avg_y = avg_y / n_models
pl.plot(x, avg_y, 'b--')
```

```
[<matplotlib.lines.Line2D at 0x1079cfbd0>]
```

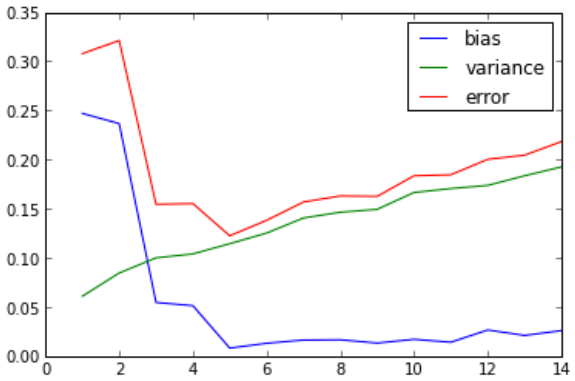


Calculating bias and variance

Same as previous example, we generate several samples and fit a polynomial to each one. We calculate bias an variance among models for different polynomial degrees. Bias, variance and error are plotted against different degree values.

```
from numpy.linalg import norm
n_samples = 20
f_x, f_y = f(n_samples)
n_models = 100
max_degree = 15
var_vals =[]
bias_vals = []
error_vals = []
for degree in xrange(1, max_degree):
    avg_y = np.zeros(n_samples)
    models = []
    for i in xrange(n_models):
        (x,y) = sample(n_samples)
        model = fit_polynomial(x, y, degree)
        p_y = apply_polynomial(model, x)
        avg_y = avg_y + p_y
        models.append(p_y)
    avg_y = avg_y / n_models
    bias_2 = norm(avg_y - f_y)/f_y.size
    bias_vals.append(bias_2)
    variance = 0
    for p_y in models:
        variance += norm(avg_y - p_y)
    variance /= f_y.size * n_models
    var_vals.append(variance)
    error_vals.append(variance + bias_2)
pl.plot(range(1, max_degree), bias_vals, label='bias')
pl.plot(range(1, max_degree), var_vals, label='variance')
pl.plot(range(1, max_degree), error_vals, label='error')
pl.legend()
```

```
<matplotlib.legend.Legend at 0x107b64f50>
```

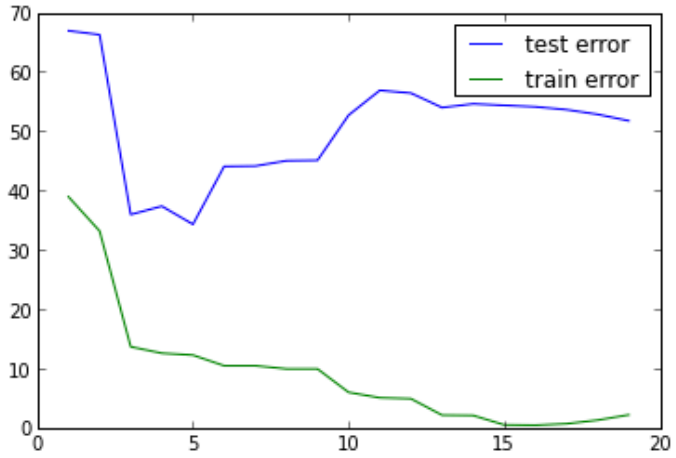


Cross Validation

Since in a real setup we don't have access to the real f function. We cannot exactly calculate the error, hoevere we can approximate it using cross validation. We generate to samples, a training sample and a validation sample. The validation sample is use to calculate an estimation of the error.

```
n_samples = 20
# train sample
train_x, train_y = sample(n_samples)
# validation sample
test_x, test_y = sample(n_samples)
max_degree = 20
test_error_vals = []
train_error_vals = []
for degree in xrange(1, max_degree):
    model = fit_polynomial(train_x, train_y, degree)
    p_y = apply_polynomial(model, train_x)
    train_error_vals.append(pl.norm(train_y - p_y)**2)
    p_y = apply_polynomial(model, test_x)
    test_error_vals.append(pl.norm(test_y - p_y)**2)
pl.plot(range(1, max_degree), test_error_vals, label='test error')
pl.plot(range(1, max_degree), train_error_vals, label='train error')
pl.legend()
```

```
<matplotlib.legend.Legend at 0x107b98250>
```

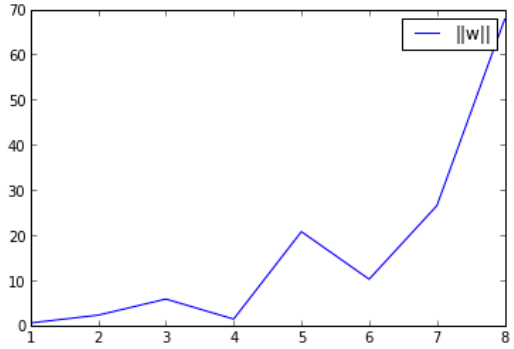


Regularization

Another way to deal with the model complexity is using regularization. A regularizer is a term that penalizes the model complexity and is part of the loss function. the next portion of code shows how the norm of the coefficients of the linear regression model increased when the complexity of the model (polynomial degree) increases.

```
n_samples = 20
train_x, train_y = sample(n_samples)
max_degree = 9
w_norm = []
for degree in xrange(1, max_degree):
    model = fit_polynomial(train_x, train_y, degree)
    w_norm.append(pl.norm(model.coef_))
pl.plot(range(1, max_degree), w_norm, label='||w||')
pl.legend()
```

```
<matplotlib.legend.Legend at 0x1079d6b10>
```



The above result suggests that we can control the complexity by penalizing the norm of the model's weights, $||w||$. This idea is implemented by the *Ridge Regression* method.

Ridge regression

Ridge regression finds a regression model by minimizing the following loss function:

$$\min_W ||WX - Y||^2 + \alpha||W||^2$$

where X and Y are the input matrix and the output vector respectively. The parameter α controls the amount of regularization. You can find more information in the documentation of scikit-learn ridge regresion implementation.

Exercise

Repeat the cross validation experiment using ridge regression. Use a fixed polynomial degree (e.g. 10) and vary the α parameter.

References

- [Alpaydin10] Alpaydin, E. 2010 Introduction to Machine Learning, 2Ed. The MIT Press.

The Bias-Variance Tradeoff

Given:

- the true function we want to approximate

$$f = f(\mathbf{x})$$

- the data set for training

$$D = \{(\mathbf{x}_1, t_1), (\mathbf{x}_2, t_2), \ldots, (\mathbf{x}_N, t_N)\} \text{ where } t = f + \varepsilon \text{ and } E\{\varepsilon\} = 0$$

- given D, we train an arbitrary neural network to approximate the function f by

$$y = g(\mathbf{x}, \mathbf{w})$$

The mean-squared error of this networks is:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(t_i - y_i)^2$$

To assess the effectiveness of the network, we want to know the expectation of the MSE if we test the network on arbitrarily many test points drawn from the unknown function.

$$E\{MSE\} = E\left\{\frac{1}{N}\sum_{i=1}^{N}(t_i - y_i)^2\right\} = \frac{1}{N}\sum_{i=1}^{N}E\left\{(t_i - y_i)^2\right\}$$

Let's investigate the expectation inside the sum, with a little "augmentation trick":

$$E\left\{(t_i - y_i)^2\right\} = E\left\{(t_i - f_i + f_i - y_i)^2\right\}$$

$$= E\left\{(t_i - f_i)^2\right\} + E\left\{(f_i - y_i)^2\right\} + 2E\left\{(f_i - y_i)(t_i - f_i)\right\}$$

$$= E\{\varepsilon^2\} + E\left\{(f_i - y_i)^2\right\} + 2\left(E\{f_i t_i\} - E\{f_i^2\} - E\{y_i t_i\} + E\{y_i f_i\}\right)$$

Note: $E\{f_i t_i\} = f_i^2$ since f is deterministic and $E\{t_i\} = f_i$

: $E\{f_i^2\} = f_i^2$ since f is deterministic

: $E\{y_i t_i\} = E\{y_i(f_i + \varepsilon)\} = E\{y_i f_i + y_i \varepsilon\} = E\{y_i f_i\} + 0$

: (the last term is zero because the noise in the infinite test set over which

: we take the expectation is probabilistically independent of the NN

: prediction). Thus the last term in the expectation above cancels to zero.

$$E\left\{(t_i - y_i)^2\right\} = E\{\varepsilon^2\} + E\left\{(f_i - y_i)^2\right\}$$

Bias-Variance

Thus the MSE can be decomposed in expectation into the variance of the noise and the MSE between the true function and the predicted values. This term can be further composed with the same augmentation trick as above.

$$E\left\{(f_i - y_i)^2\right\} = E\left\{\left(f_i - E\{y_i\} + E\{y_i\}_i - y_i\right)^2\right\}$$

$$= E\left\{(f_i - E\{y_i\})^2\right\} + E\left\{(E\{y_i\} - y_i)^2\right\} + 2E\left\{(E\{y_i\} - y_i)(f_i - E\{y_i\})\right\}$$

$$= bias^2 + Var\{y_i\} + 2\left(E\{f_i E\{y_i\}\} - E\left\{E\{y_i\}^2\right\} - E\{y_i f\}_i + E\{y_i E\{y_i\}\}\right)$$

Note: $E\{f_i E\{y_i\}\} = f_i E\{y_i\}$ since f is deterministic and $E\{E\{z\}\} = z$

: $E\left\{E\{y_i\}^2\right\} = E\{y_i\}^2$ since $E\{E\{z\}\} = z$

: $E\{y_i f_i\} = f_i E\{y_i\}$

: $E\{y_i E\{y_i\}\} = E\{y_i\}^2$

: Thus the last term in the expectation above cancels to zero.

$$E\left\{(f_i - y_i)^2\right\} = bias^2 + Var\{y_i\}$$

Thus the decomposition of the MSE in expectation becomes:

$$E\left\{(t_i - y_i)^2\right\} = Var\{noise\} + bias^2 + Var\{y_i\}$$

Note that the variance of the noise can not be minimized; it is independent of the neural network. Thus in order to minimize the MSE, we need to minimize both the bias and the variance. However, this is not trivial to do this. For instance, just neglecting the input data and predicting the output somehow (e.g., just a constant), would definitely minimize the variance of our predictions: they would be always the same, thus the variance would be zero—but the bias of our estimate (i.e., the amount we are off the real function) would be tremendously large. On the other hand, the neural network could perfectly interpolate the training data, i.e., it predict y=t for every data point. This will make the bias term vanish entirely, since the E(y)=f (insert this above into the squared bias term to verify this), but the variance term will become equal to the variance of the noise, which may be significant (see also Bishop Chapter 9 and the Geman et al. Paper). In general, finding an optimal bias-variance tradeoff is hard, but acceptable solutions can be found, e.g., by means of cross validation or regularization.

http://www.inf.ed.ac.uk/teaching/courses/mlsc/Notes/Lecture4/BiasVariance.pdf

Bias-Variance Tradeoff

using radial basis functions (blue). Several trials are shown in each graph. For each trial, a few noisy data points are provided as training set (top). For a wide spread (image 2) the bias is high: the RBFs cannot fully approximate the function (especially the central dip), but the variance between different trials is low. As spread decreases (image 3 and 4) the bias decreases: the blue curves more closely approximate the red. However, depending on the noise in different trials the variance between trials increases. In the lowermost image the approximated values for x=0 varies wildly depending on where the data points were located.

Bias–variance decomposition of squared error [edit]

Suppose that we have a training set consisting of a set of points $x_1, \ldots, x_n$ and real values y_i associated with each point x_i. We assume that there is a functional, but noisy relation $y_i = f(x_i) + \epsilon$, where the noise, ϵ, has zero mean and variance σ^2.

We want to find a function $\hat{f}(x)$, that approximates the true function $y = f(x)$ as well as possible, by means of some learning algorithm. We make "as well as possible" precise by measuring the mean squared error between y and $\hat{f}(x)$: we want $(y - \hat{f}(x))^2$ to be minimal, both for $x_1, \ldots, x_n$ *and for points outside of our sample*. Of course, we cannot hope to do so perfectly, since the y_i contain noise ϵ; this means we must be prepared to accept an *irreducible error* in any function we come up with.

Finding an $\hat{f}$ that generalizes to points outside of the training set can be done with any of the countless algorithms used for supervised learning. It turns out that whichever function $\hat{f}$ we select, we can decompose its expected error on an unseen sample x as follows:[3]:34[4]:223

$$\mathrm{E}\left[(y - \hat{f}(x))^2\right] = \mathrm{Bias}\left[\hat{f}(x)\right]^2 + \mathrm{Var}\left[\hat{f}(x)\right] + \sigma^2$$

Where:

$$\mathrm{Bias}\left[\hat{f}(x)\right] = \mathrm{E}\left[\hat{f}(x)\right] - f(x)$$

and

$$\mathrm{Var}\left[\hat{f}(x)\right] = \mathrm{E}\left[\left(\hat{f}(x) - \mathrm{E}[\hat{f}(x)]\right)^2\right]$$

The expectation ranges over different choices of the training set $x_1, \ldots, x_n, y_1, \ldots, y_n$, all sampled from the same distribution. The three terms represent:

- the square of the *bias* of the learning method, which can be thought of the error caused by the simplifying assumptions built into the method. E.g., when approximating a non-linear function $f(x)$ using a learning method for linear models, there will be error in the estimates $\hat{f}(x)$ due to this assumption;
- the *variance* of the learning method, or, intuitively, how much the learning method $\hat{f}(x)$ will move around its mean;
- the irreducible error σ^2. Since all three terms are non-negative, this forms a lower bound on the expected error on unseen samples.[3]:34

The more complex the model $\hat{f}(x)$ is, the more data points it will capture, and the lower the bias will be. However, complexity will make the model "move" more to capture the data points, and hence its variance will be larger.

https://en.wikipedia.org/wiki/Bias%E2%80%93variance_tradeoff

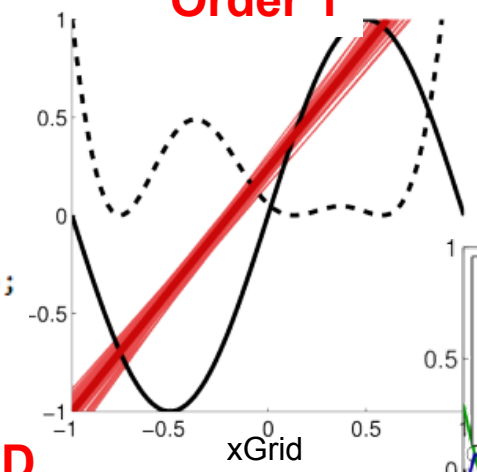
https://theclevermachine.wordpress.com/2013/04/21/model-selection-underfitting-overfitting-and-the-bias-variance-tradeoff/

Bias-Variance
Order 1, 2,3 polynomial

xGrid = linspace(-1,1,100);

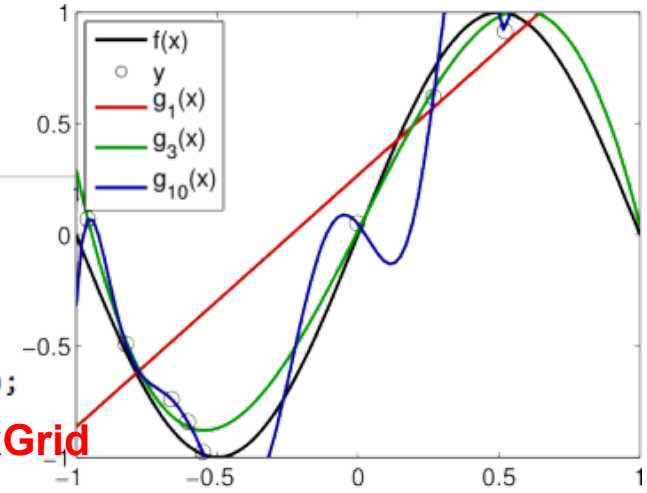
```
% FIT MODELS TO K INDEPENDENT DATASETS
K = 50;
for iS = 1:K
    ySim = f(x) + noiseSTD*randn(size(x));
    for jD = 1:numel(degree)
        % FIT THE MODEL USING polyfit.m
        thetaTmp = polyfit(x,ySim,degree(jD));
        % EVALUATE THE MODEL FIT USING polyval.m
        simFit{jD}(iS,:) = polyval(thetaTmp,xGrid);
    end
end

% DISPLAY ALL THE MODEL FITS
h = [];
for iD = 1:numel(degree)
    figure(iD+1)
    hold on
    % PLOT THE FUNCTION FIT TO EACH DATASET
    for iS = 1:K
        h(1) = plot(xGrid,simFit{iD}(iS,:),'color',brighten(cols(iD,:),.6));
    end
    % PLOT THE AVERAGE FUNCTION ACROSS ALL FITS
    h(2) = plot(xGrid,mean(simFit{iD}),'color',cols(iD,:),'Linewidth',5);
    % PLOT THE UNDERLYING FUNCTION f(x)
    h(3) = plot(xGrid,f(xGrid),'color','k','Linewidth',3);
    % CALCULATE THE SQUARED ERROR AT EACH POINT, AVERAGED ACROSS ALL DATASETS
    squaredError = (mean(simFit{iD})-f(xGrid)).^2;
    % PLOT THE SQUARED ERROR
    h(4) = plot(xGrid,squaredError,'k--','Linewidth',3);
    uistack(h(2),'top')
    hold off
    axis square
    xlim([-1 1])
    ylim([-1 1])
    legend(h,{sprintf('Individual g_{%d}(x)',degree(iD)),'Mean of All Fits','f(x)','Squared Error'},'Location','WestOu
    title(sprintf('Model Order=%d',degree(iD)))
end
```

Order 1



f(x) = sin(pi X x)

Order 1, 2, 3 with data



For polynomial =iD

iS Dataset Sample

Predictions for sample set iS using model iD

Avg predictions of xGrid

BIAS

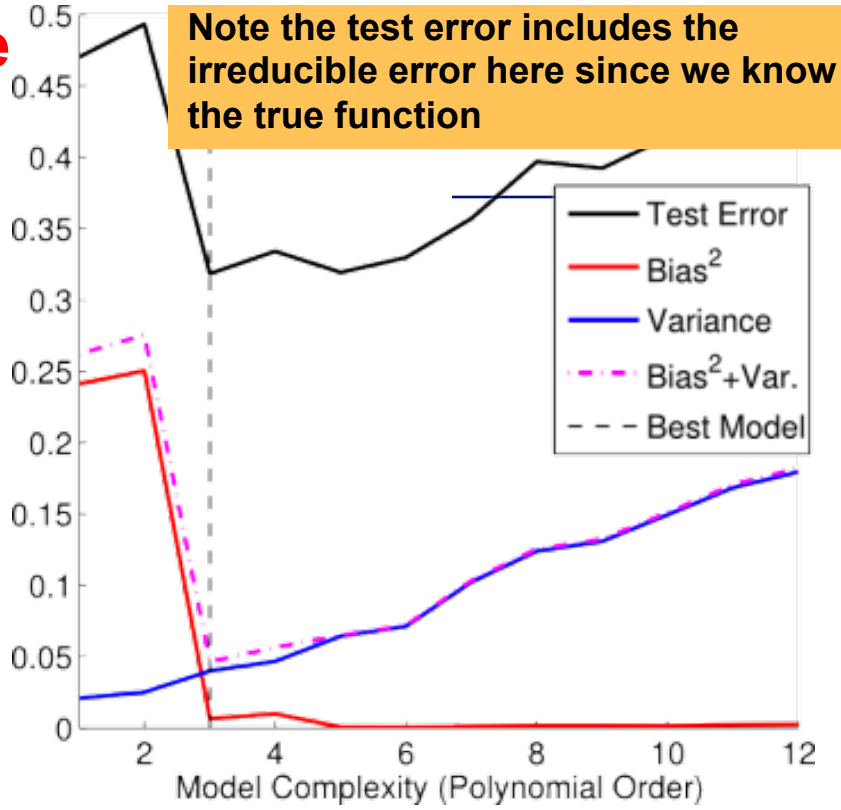
True error

f(x) = true functi
ySim = noisy dat
simFit matrix of p

Our original goal was to approximate f(x), not the data points per se.

Bias-Variance Order [1-12] polynomials

```
% # INITIALIZE SOME VARIABLES
xGrid = linspace(-1,1,N);
meanPrediction = zeros(K,N);
thetaHat = {};
x = linspace(-1,1,N);
x = x(randperm(N));
for iS = 1:K % LOOP OVER DATASETS
    % CREATE OBSERVED DATA, y
    y = f(x) + noiseSTD*randn(size(x));

    % CREATE TRAINING SET
    xTrain = x(1:nTrain);
    yTrain = y(1:nTrain);

    % CREATE TESTING SET
    xTest = x(nTrain+1:end);
    yTest = y(nTrain+1:end);

    % FIT MODELS
    for jD = 1:nPolyMax

        % MODEL PARAMETER ESTIMATES
        thetaHat{jD}(iS,:) = polyfit(xTrain,yTrain,jD);

        % PREDICTIONS
        yHatTrain{jD}(iS,:) = polyval([thetaHat{jD}(iS,:)],xTrain); TRAINING SET
        yHatTest{jD}(iS,:) = polyval([thetaHat{jD}(iS,:)],xTest);% TESTING SET

        % MEAN SQUARED ERROR
        trainErrors{jD}(iS) = mean((yHatTrain{jD}(iS,:) - yTrain).^2); % TRAINING
        testErrors{jD}(iS) = mean((yHatTest{jD}(iS,:) - yTest).^2); % TESTING
    end
end

% CALCULATE AVERAGE PREDICTION ERROR, BIAS, AND VARIANCE
for iD = 1:nPolyMax
    trainError(iD) = mean(trainErrors{iD});
    testError(iD) = mean(testErrors{iD});
    biasSquared(iD) = mean((mean(yHatTest{iD})-f(xTest)).^2);
    variance(iD) = mean(var(yHatTest{iD},1));
end
[~,bestModel] = min(testError);
```



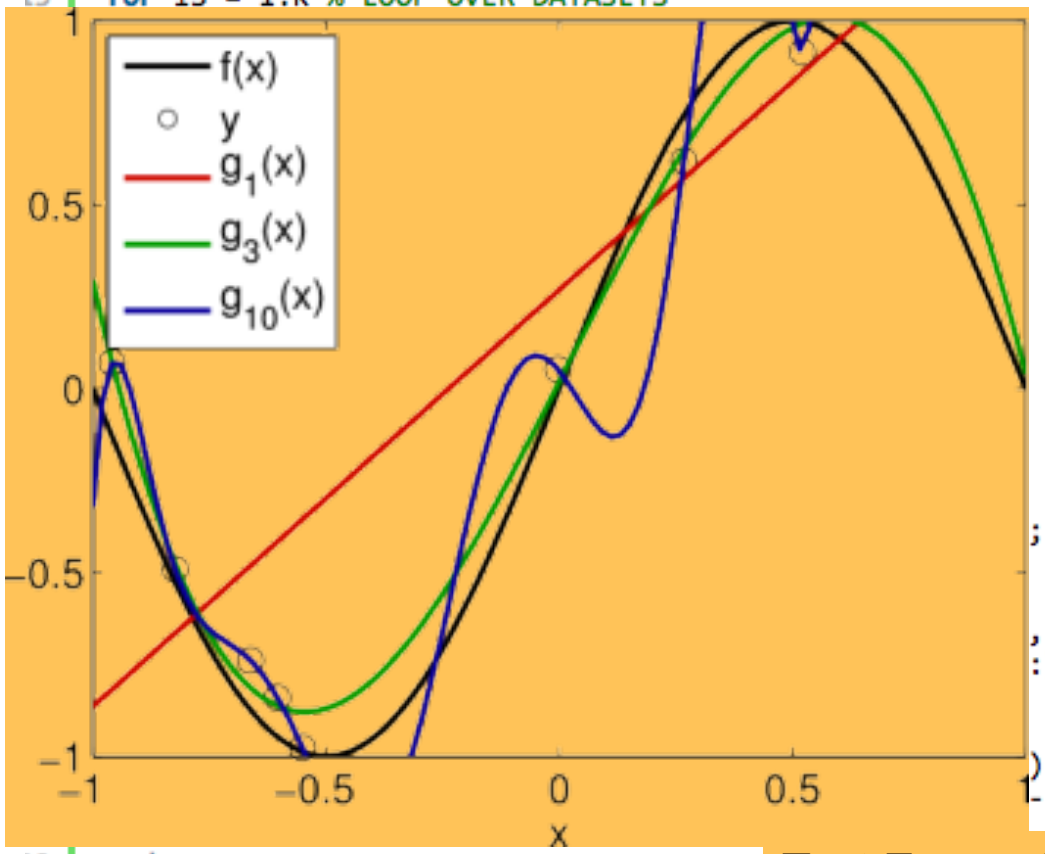
Note the test error includes the irreducible error here since we know the true function

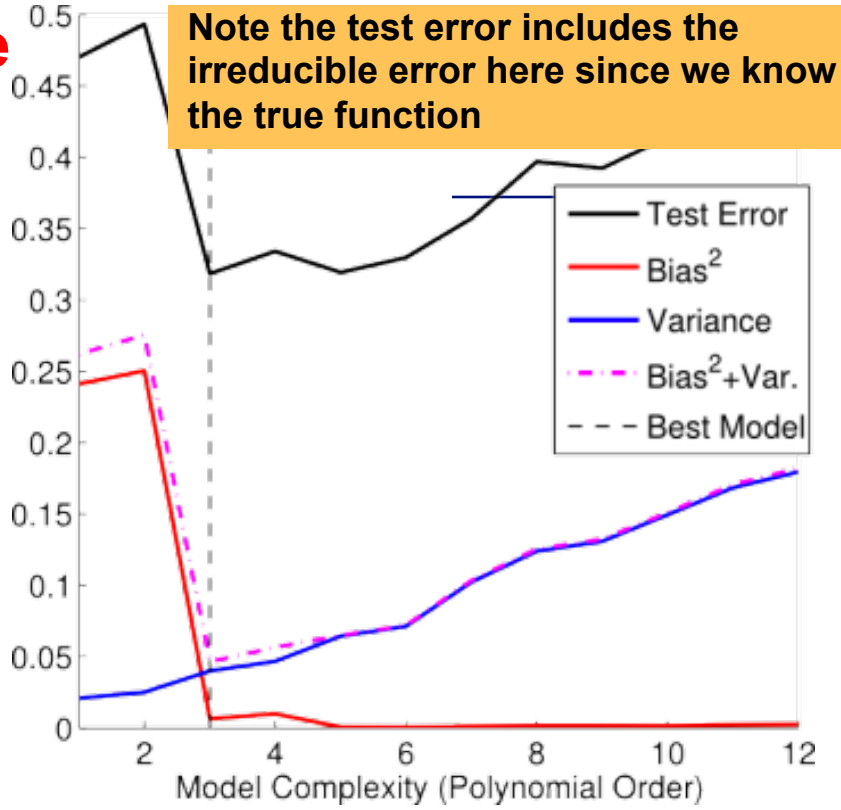
Test Error = Variance(x_i)+ Bias(x_i)+irreducibleError

Avg(Bias (x_i))
Avg(Variance (x_i))

Bias-Variance Order [1-12] polynomials

```
% # INITIALIZE SOME VARIABLES
xGrid = linspace(-1,1,N);
meanPrediction = zeros(K,N);
thetaHat = {};
x = linspace(-1,1,N);
x = x(randperm(N));
for iS = 1:K % LOOP OVER DATASETS
```





Note the test error includes the irreducible error here since we know the true function

```
,:)],xTrain); TRAINING SET
:)],xTest);% TESTING SET

) - yTrain).^2); % TRAINING
- yTest).^2); % TESTING
```

Test Error = Variance(x_i)+ Bias(x_i)+irreducibleError

```
end

% CALCULATE AVERAGE PREDICTION ERROR, BIAS, AND VARIANCE
for iD = 1:nPolyMax
    trainError(iD) = mean(trainErrors{iD});
    testError(iD) = mean(testErrors{iD});
    biasSquared(iD) = mean((mean(yHatTest{iD})-f(xTest)).^2);
    variance(iD) = mean(var(yHatTest{iD},1));
end
[~,bestModel] = min(testError);
```

Avg(Bias (x_i))
Avg(Variance (x_i))

Best least squares fit for monomials of degree 1 to n.

- p = polyfit(x,y,n) returns the coefficients for a polynomial p(x) of degree n that is a best fit (in a least-squares sense) for the data in y. The coefficients in p are in descending powers, and the length of p is n+1

polyfit

Polynomial curve

Syntax

```
p = polyfit(x,y,n)
[p,S] = polyfit(x,y,n)
[p,S,mu] = polyfit(x,y,n)
```

http://www.mathworks.com/help/matlab/ref/polyfit.html

Description

`p = polyfit(x,y,n)` returns the coefficients for a polynomial `p(x)` of degree n that is a best fit (in a least-squares sense) for the data in y. The coefficients in p are in descending powers, and the length of p is n+1

$$p(x) = p_1 x^n + p_2 x^{n-1} + \ldots + p_n x + p_{n+1}.$$

`[p,S] = polyfit(x,y,n)` also returns a structure S that can be used as an input to `polyval` to obtain error estimates.

`[p,S,mu] = polyfit(x,y,n)` also returns `mu`, which is a two-element vector with centering and scaling values. `mu(1)` is `mean(x)`, and `mu(2)` is `std(x)`. Using these values, `polyfit` centers x at zero and scales it to have unit standard deviation

$$\hat{x} = \frac{x - \bar{x}}{\sigma_x}.$$

This centering and scaling transformation improves the numerical properties of both the polynomial and the fitting algorithm.

Examples

▼ Fit Polynomial to Trigonometric Function

Generate 10 points equally spaced along a sine curve in the interval `[0,4*pi]`.

```
x = linspace(0,4*pi,10);
y = sin(x);
```

Use `polyfit` to fit a 7th-degree polynomial to the po

```
p = polyfit(x,y,7);
```

Evaluate the polynomial on a finer grid and plot the re

```
x1 = linspace(0,4*pi);
y1 = polyval(p,x1);
figure
plot(x,y,'o')
hold on
plot(x1,y1)
hold off
```

```
> x1=0.1; x=c(x1^6, x1^5, x1^4,x1^3, x1^2, x1^1, 1)
> t(x) %*% ((c(0.0084,  -0.0983,    0.4217,   -0.7435,    0.1471,    1.1064,
0.00044117)))
          [,1]
[1,] 0.1118499
>
```

Fit a polynomial of degree 6

Determine the coefficients of the approximating polynomial of degree 6.

```
p = polyfit(x,y,6)
```

```
p =

    0.0084   -0.0983    0.4217   -0.7435    0.1471    1.1064    0.0004
```

To see how good the fit is, evaluate the polynomial at the data points and generate a table showing the data, fit, and error.

```
f = polyval(p,x);
T = table(x,y,f,y-f,'VariableNames',{'X','Y','Fit','FitError'})
```

```
T =

     X        Y         Fit          FitError
    ___    _______    _________    __________

      0          0    0.00044117    -0.00044117
    0.1    0.11246       0.11185     0.00060836
    0.2     0.2227       0.22231     0.00039189
    0.3    0.32863       0.32872    -9.7429e-05
    0.4    0.42839        0.4288    -0.0004066
    0.5     0.5205       0.52093    -0.0004256
    0.6    0.60386       0.60408    -0.0002282
    0.7     0.6778       0.67775     4.6383e-0
    0.8     0.7421       0.74183     0.0002699
    0.9    0.79691       0.79654     0.0003651
      1     0.8427       0.84238      0.000316
    1.1    0.88021       0.88005     0.0001594
    1.2    0.91031       0.91035    -3.9919e-0
    1.3    0.93401       0.93422      -0.00021
    1.4    0.95229       0.95258    -0.0002993
    1.5    0.96611       0.96639    -0.0002809
    1.6    0.97635       0.97652    -0.00016704
    1.7    0.98379       0.98379     8.3306e-07
    1.8    0.98909       0.98893     0.00016278
    1.9    0.99279       0.99253     0.00025791
      2    0.99532       0.99508     0.00024347
    2.1    0.99702       0.99691      0.0001131
    2.2    0.99814       0.99823    -8.8548e-05
    2.3    0.99886       0.99911    -0.00025673
    2.4    0.99931       0.99954    -0.00022451
    2.5    0.99959       0.99936     0.00023151
```

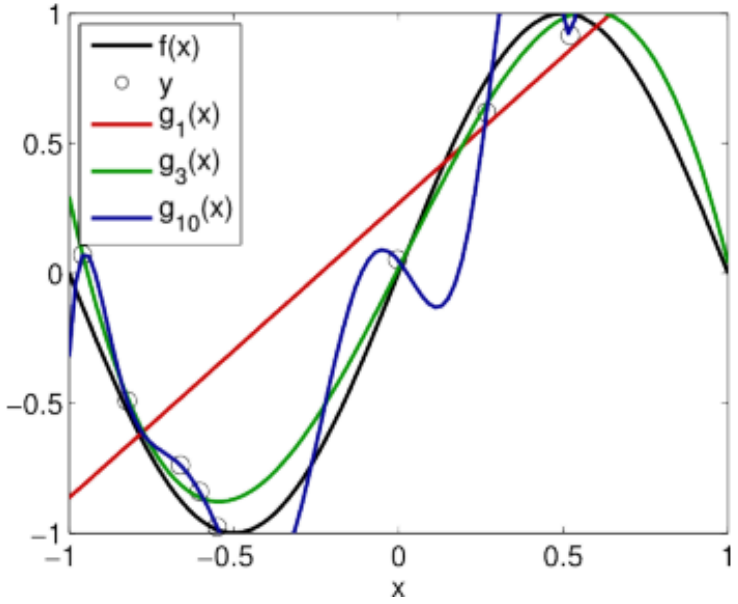
http://www.mathworks.com/help/matlab/ref/polyfit.html

```
> x1=0.1; x=c(x1^6, x1^5, x1^4,x1^3, x1^2, x1^1, 1)
> t(x) %*% ((c(0.0084,  -0.0983,    0.4217,   -0.7435,    0.1471,    1.1
0.00044117)))
         [,1]
[1,] 0.1118499
>
```

Fit a polynomial of degree 1, 3, 10

Below we estimate the parameters of three polynomial model functions of increasing complexity (using Matlab's `polyfit.m`) to the sampled data displayed above. Specifically, we estimate the functions $g_1(x)$, $g_3(x)$ and $g_{10}(x)$.

```
% FIT POLYNOMIAL MODELS & DISPLAY
% (ASSUMING PREVIOUS PLOT ABOVE STILL AVAILABLE)
degree = [1,3,10];
theta = {};
cols = [.8 .05 0.05; 0.05 .6 0.05; 0.05 0.05 .6];
for iD = 1:numel(degree)
    figure(1)
    theta{iD} = polyfit(x,y,degree(iD));
    fit{iD} = polyval(theta{iD},xGrid);
    h(end+1) = plot(xGrid,fit{iD},'color',cols(iD,:),'Linewidth',2);
    xlim([-1 1])
    ylim([-1 1])
end
legend(h,'f(x)','y','g_1(x)','g_3(x)','g_{10}(x)','Location','Northwest')
```



```
% FIT MODELS TO K INDEPENDENT DATASETS
K = 50;
for iS = 1:K
    ySim = f(x) + noiseSTD*randn(size(x));
    for jD = 1:numel(degree)
        % FIT THE MODEL USING polyfit.m
        thetaTmp = polyfit(x,ySim,degree(jD));
        % EVALUATE THE MODEL FIT USING polyval.m
        simFit{jD}(iS,:) = polyval(thetaTmp,xGrid);
    end
end

% DISPLAY ALL THE MODEL FITS
h = [];
for iD = 1:numel(degree)
    figure(iD+1)
    hold on
    % PLOT THE FUNCTION FIT TO EACH DATASET
    for iS = 1:K
        h(1) = plot(xGrid,simFit{iD}(iS,:),'color',brighten(cols(iD,:),.6));
    end
    % PLOT THE AVERAGE FUNCTION ACROSS ALL FITS
    h(2) = plot(xGrid,mean(simFit{iD}),'color',cols(iD,:),'Linewidth',5);
    % PLOT THE UNDERLYING FUNCTION f(x)
    h(3) = plot(xGrid,f(xGrid),'color','k','Linewidth',3);
    % CALCULATE THE SQUARED ERROR AT EACH POINT, AVERAGED ACROSS ALL DATASETS
    squaredError = (mean(simFit{iD})-f(xGrid)).^2;
    % PLOT THE SQUARED ERROR
    h(4) = plot(xGrid,squaredError,'k--','Linewidth',3);
    uistack(h(2),'top')
    hold off
    axis square
    xlim([-1 1])
    ylim([-1 1])
    legend(h,{sprintf('Individual g_{%d}(x)',degree(iD)),'Mean of All Fits','f(x)','Squared Error'},'Location','West
    title(sprintf('Model Order=%d',degree(iD)))
end
```

In Practice: Managing Bias and Variance

- **What to think about when trying to manage bias and variance?**
- **Fight Your Instincts**
  - A gut feeling many people have is that they should minimize bias even at the expense of variance. WRONG
- **Bagging and Resampling**
  - To make a prediction, all of the models in the ensemble are polled and their results are averaged. One powerful modeling algorithm that makes good use of bagging is Random Forests.
  - In Random Forests the bias of the full model is equivalent to the bias of a single decision tree (which itself has high variance). BUT By creating many of these trees, in effect a "forest", and then averaging them the variance of the final model can be greatly reduced over that of a single tree.

http://scott.fortmann-roe.com/docs/BiasVariance.html

- **Asymptotic Properties of Algorithms**
  - Infinite data means zero bias and zero variance. WRONG
  - When working with real data, it is best to leave aside theoretical properties of algorithms and to instead focus on their actual accuracy in a given scenario.
- **Understanding Over- and Under-Fitting**
  - As more and more parameters are added to a model, the complexity of the model rises and variance becomes our primary concern while bias steadily falls.

In Practice: Managing Bias and Variance

- **There are some key things to think about when trying to manage bias and variance.**
- Fight Your Instincts
  - A gut feeling many people have is that they should minimize bias even at the expense of variance. Their thinking goes that the presence of bias indicates something basically wrong with their model and algorithm. Yes, they acknowledge, variance is also bad but a model with high variance could at least predict well on average, at least it is not *fundamentally wrong*.
  - This is mistaken logic. It is true that a high variance and low bias model can preform well in some sort of long-run average sense. However, in practice modelers are always dealing with a single realization of the data set. In these cases, long run averages are irrelevant, what is important is the performance of the model on the data you actually have and in this case bias and variance are equally important and one should not be improved at an excessive expense to the other.

http://scott.fortmann-roe.com/docs/BiasVariance.html

Bagging and Resampling

- Bagging and other resampling techniques can be used to reduce the variance in model predictions. In bagging (Bootstrap Aggregating), numerous replicates of the original data set are created using random selection with replacement. Each derivative data set is then used to construct a new model and the models are gathered together into an ensemble. To make a prediction, all of the models in the ensemble are polled and their results are averaged.
- One powerful modeling algorithm that makes good use of bagging is Random Forests. Random Forests works by training numerous decision trees each based on a different resampling of the original training data. In Random Forests the bias of the full model is equivalent to the bias of a single decision tree (which itself has high variance). By creating many of these trees, in effect a "forest", and then averaging them the variance of the final model can be greatly reduced over that of a single tree. In practice the only limitation on the size of the forest is computing time as an infinite number of trees could be trained without ever increasing bias and with a continual (if asymptotically declining) decrease in the variance.

Asymptotic Properties of Algorithms

- Academic statistical articles discussing prediction algorithms often bring up the ideas of asymptotic consistency and asymptotic efficiency. In practice what these imply is that as your training sample size grows towards infinity, your model's bias will fall to 0 (asymptotic consistency) and your model will have a variance that is no worse than any other potential model you could have used (asymptotic efficiency).
- Both these are properties that we would like a model algorithm to have. We, however, do not live in a world of infinite sample sizes so asymptotic properties generally have very little practical use. An algorithm that may have close to no bias when you have a million points, may have very significant bias when you only have a few hundred data points. More important, an asymptotically consistent and efficient algorithm may actually perform worse on small sample size data sets than an algorithm that is neither asymptotically consistent nor efficient. When working with real data, it is best to leave aside theoretical properties of algorithms and to instead focus on their actual accuracy in a given scenario.

At its root, dealing with bias and variance is really about dealing with over- and under-fitting. Bias is reduced and variance is increased in relation to model complexity. As more and more parameters are added to a model, the complexity of the model rises and variance becomes our primary concern while bias steadily falls. For example, as more polynomial terms are added to a linear regression, the greater the resulting model's complexity will be [3]. In other words, bias has a negative first-order derivative in response to model complexity [4] while variance has a positive slope.

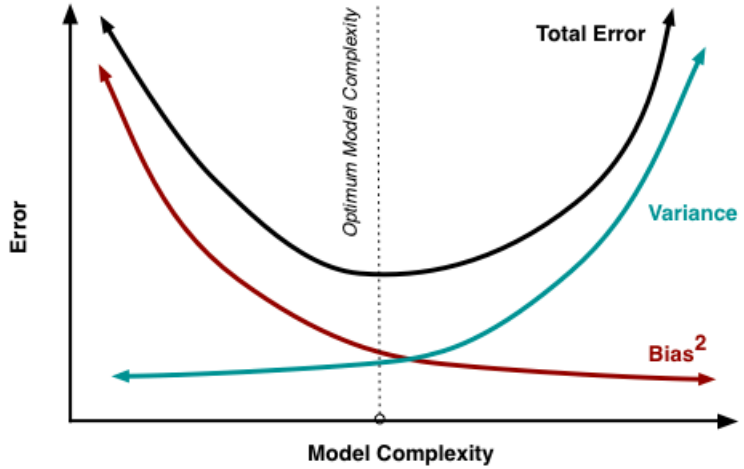


Fig. 6 Bias and variance contributing to total error.

Understanding bias and variance is critical for understanding the behavior of prediction models, but in general what you really care about is overall error, not the specific decomposition. The sweet spot for any model is the level of complexity at which the increase in bias is equivalent to the reduction in variance. Mathematically:

$$\frac{dBias}{dComplexity} = -\frac{dVariance}{dComplexity}$$

If our model complexity exceeds this sweet spot, we are in effect over-fitting our model; while if our complexity falls short of the sweet spot, we are under-fitting the model. In practice, there is not an analytical way to find this location. Instead we must use an accurate measure of prediction error and explore differing levels of model complexity and then choose the complexity level that minimizes the overall error. A key to this process is the selection of an *accurate* error measure as often grossly inaccurate measures are used which can be deceptive. The topic of accuracy measures is discussed here but generally resampling based measures such as cross-validation should be preferred over theoretical measures such as Aikake's Information Criteria.

Live Session Outline

- **Machine Learning Introduction**
- **Equation of a line**
- **Linear Regresion**
- **Bias-Variance**
  - Expected Value, Variance
  - Bias-Variance
  - Coding it up
    - For a simulated world (lab setting)
    - For a real world problems
  - Example using bagging

High bias low variance: Decision Trees How deep are the decision trees

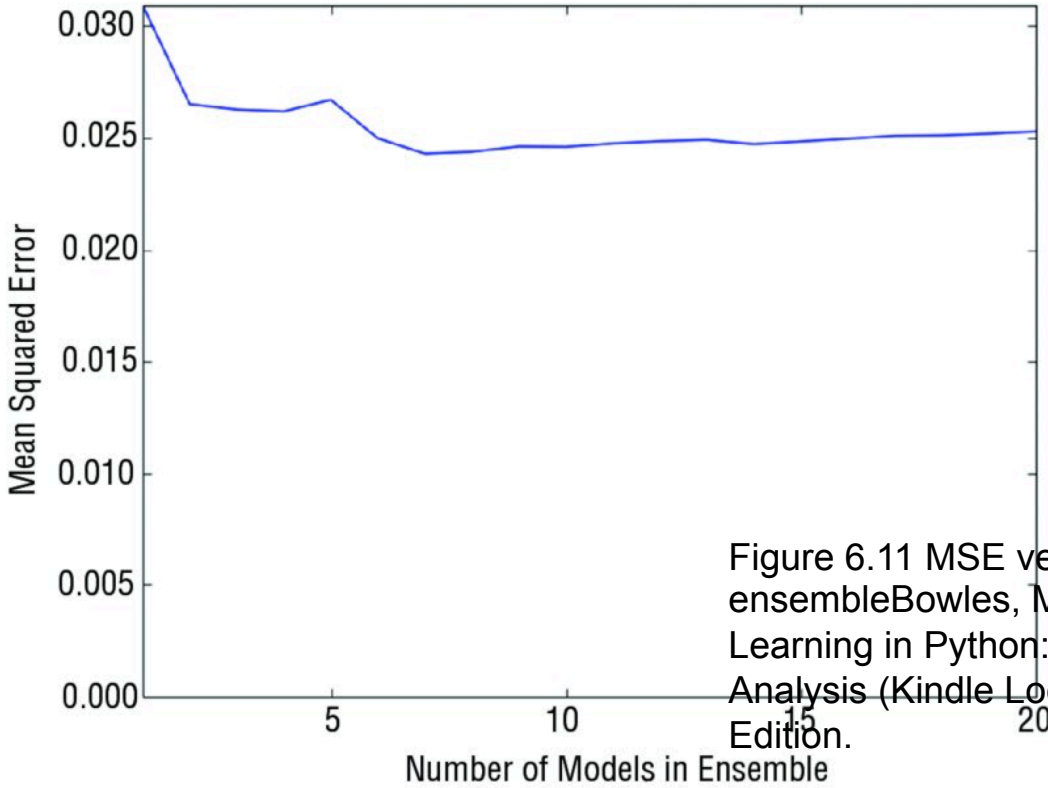


Figure 6.11 MSE versus number of trees in Bagging ensembleBowles, Michael (2015-03-31). Machine Learning in Python: Essential Techniques for Predictive Analysis (Kindle Locations 5229-5230). Wiley. Kindle Edition.

Figure 6.11 shows how the MSE varies as the number of trees is increased. The error more or less levels out at around 0.025. This isn't really very good. The noise that was added had a standard deviation of 0.1. The very best MSE a predictive algorithm could generate is the square of that standard deviation or 0.01. The single binary tree that was trained earlier in the chapter was getting close to 0.01. Why is this more sophisticated algorithm underperforming?Bowles, Michael (2015-03-31). Machine Learning in Python: Essential Techniques for Predictive Analysis (Kindle Locations 5224-5227). Wiley. Kindle Edition.

Bagging Performance— Bias versus Variance
single tree prediction, the 10-tree prediction, and the 20-tree prediction

HINT

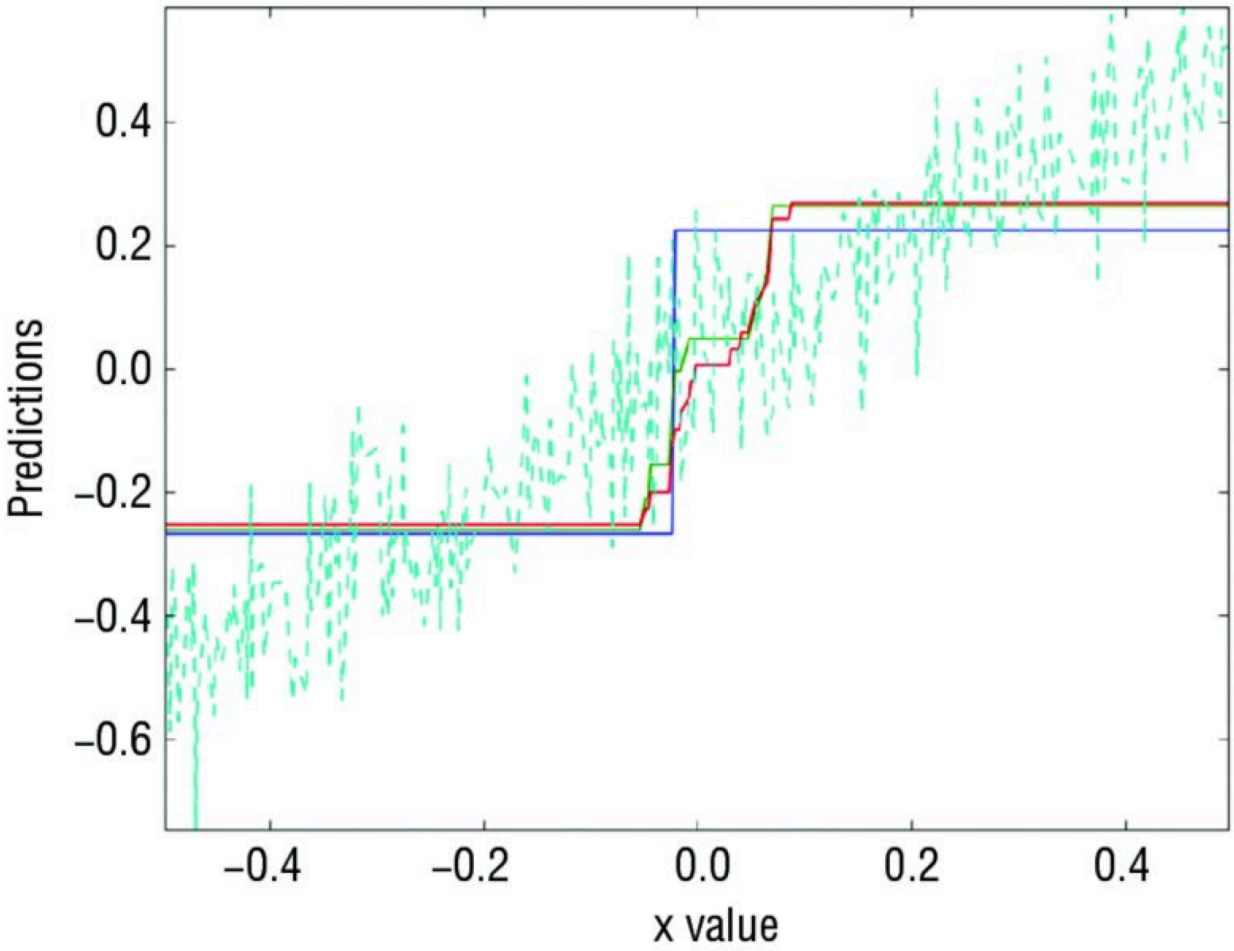
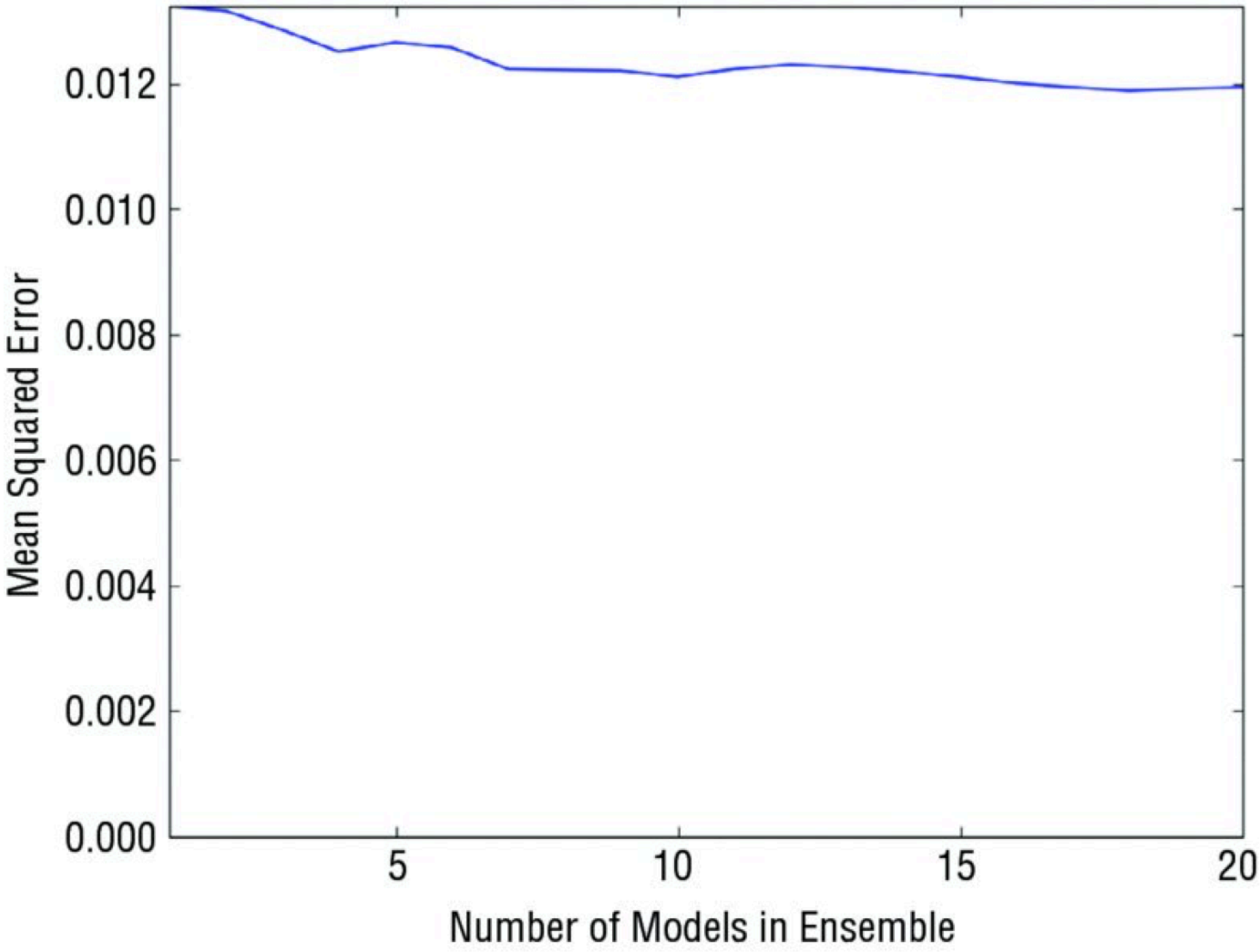


Figure 6.12 Comparison of prediction and actual label as functions of attributeBowles, Michael (2015-03-31). Machine Learning in Python: Essential Techniques for Predictive Analysis (Kindle Location 5242). Wiley. Kindle Edition.

Huge bias (DT depth 1) and low variance (Ensembles via Bagging)

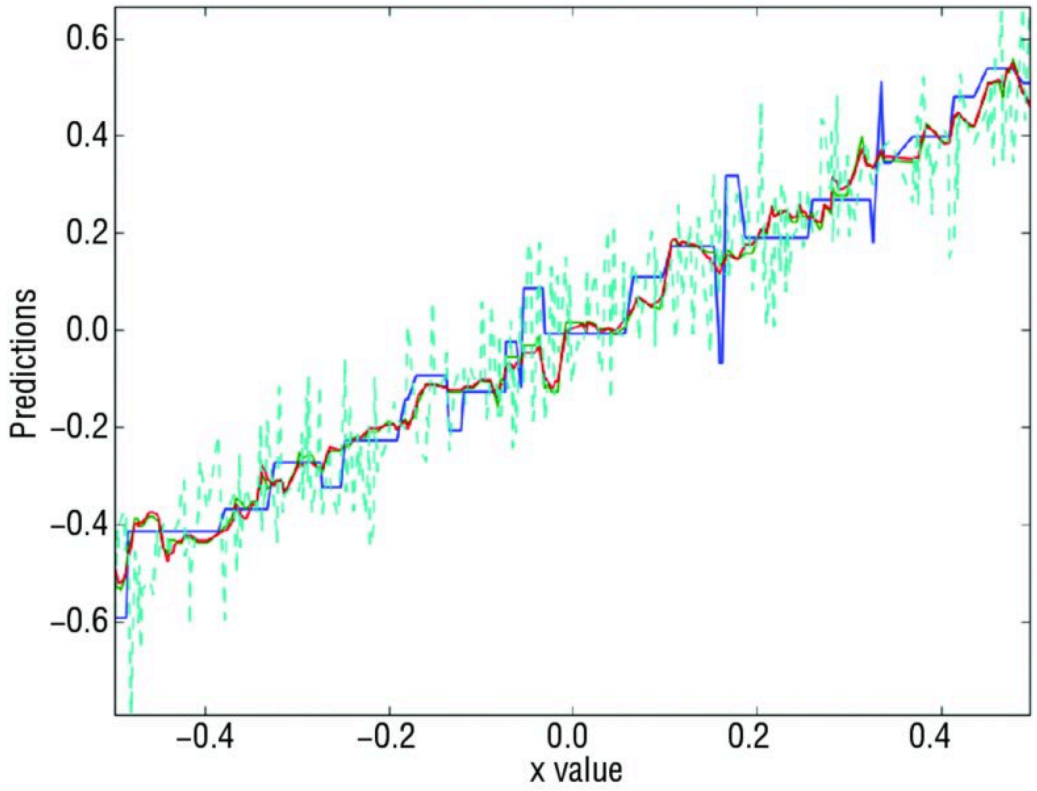
- **Consider trying to fit a wiggly curve with a straight line.**
- **Getting more data can reduce the effect of noise in the data being used for fitting, but more data will not make a straight line into a wiggly curve.**
  - Errors that do not get smaller as more data points are added are called bias errors.
  - Fitting depth-1 trees to the synthetic problem suffers from a bias error.
  - All the split points are chosen near the center of the data, and the model accuracy suffers at the edges of the data.
  - The bias error with depth 1 trees comes from the basic model being too simple and sharing a common limitation.
  - Bagging reduces variance between models.
  - But with depth 1 trees, it gets a bias error, which can't be averaged. The way to overcome this problem is to use trees with more depth.
  - Figure 6.13 shows the curve of MSE versus number of trees in the ensemble for depth 5 trees.
  - The MSE with depth 5 trees is somewhat smaller than 0.01 (probably due to randomness in the noise data), clearly much better performance than with depth 1 trees

Depth 5 DT Bagging Ensemble



gure 6.13 MSE versus number of trees with depth 5 trees

Depth 5 (first tree, the first 10 trees, and the first 20 trees)

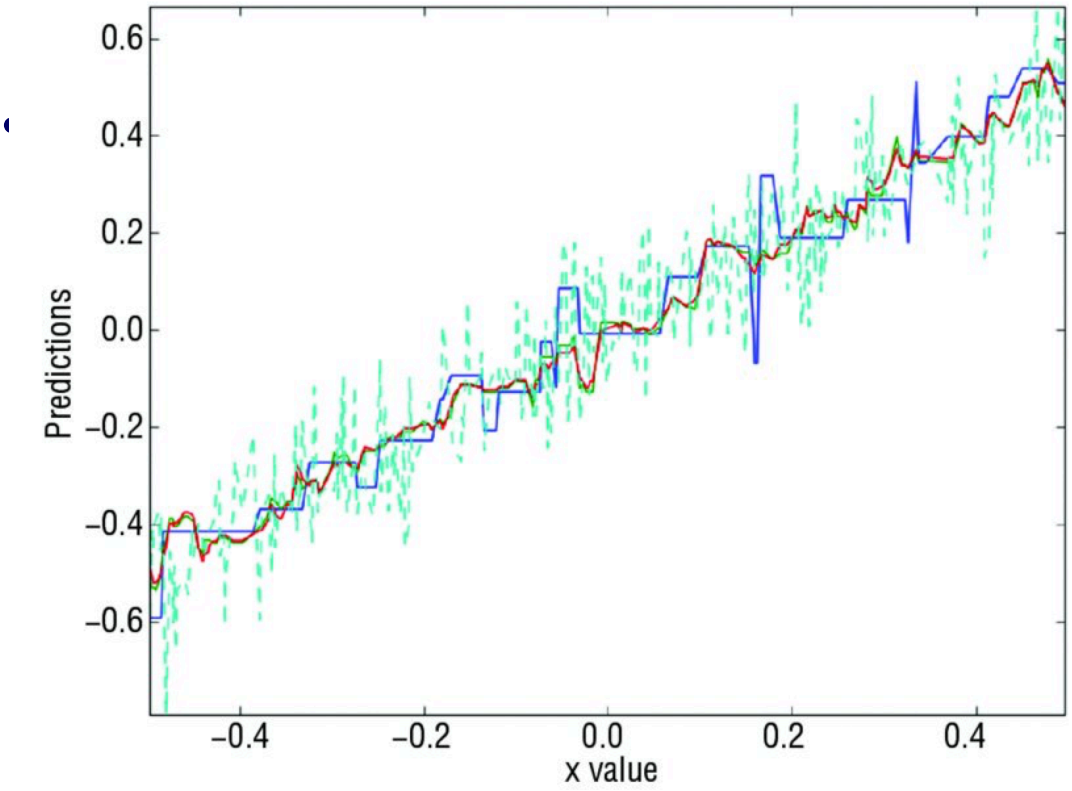


e 6.14 Comparison of prediction and actual labels with depth 5 trees

Which model is which? 1 versu 10 versus 20 trees (of depth 5).

Explain

Depth 5 (first tree, the first 10 trees, and the first 20 trees)



e 6.14 Comparison of prediction and actual labels with depth 5 trees

The single tree prediction stands out from the others because it has a number of sharp spikes where it's making severe errors. In other words, it has a high variance. The other single trees undoubtedly show similar performance. But when they're average, the variance is reduced; the curve representing the prediction from the bagging algorithm is much smoother and closer to the true answer.

BV example for classification

- http://scott.fortmann-roe.com/docs/BiasVariance.html

An Applied Example: Voter Party Registration

Let's look at a bit more realistic example. Assume we have a training data set of voters each tagged with three properties: voter party registration, voter wealth, and a quantitative measure of voter religiousness. These simulated data are plotted below [2]. The x-axis shows increasing wealth, the y-axis increasing religiousness and the red circles represent Republican voters while the blue circles represent Democratic votes. We want to predict voter registration using wealth and religiousness as predictors.

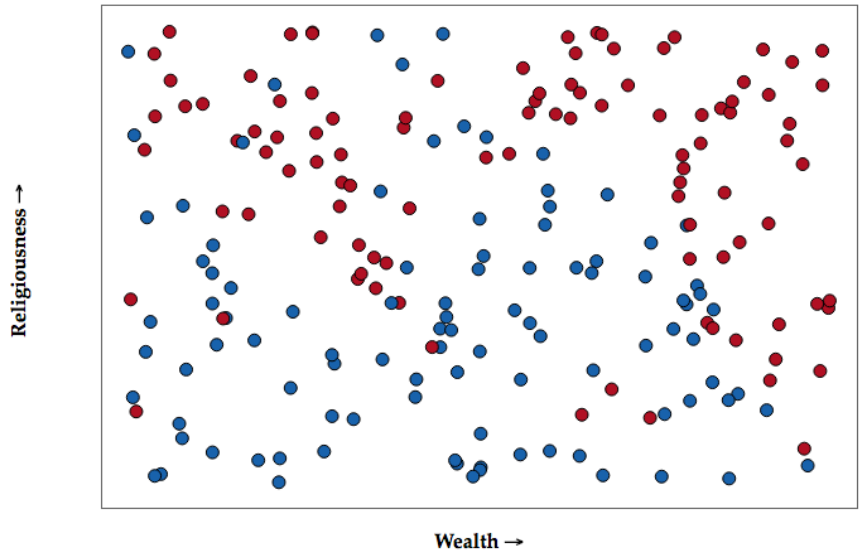


Fig. 2 Hypothetical party registration. Plotted on religiousness (y-axis) versus wealth (x-axis).

Summary of Bagging

- Now you have seen a first example of an ensemble method.
- Bagging clearly demonstrates the two-level hierarchy common to ensemble methods. Properly speaking, bagging is the higher-level algorithm defining a series of subproblems to be solved by base learners and then averaging their predictions.
- The individual problems making up a bagging ensemble are derived by taking random bootstrap samples of the original training data. Bagging reduces the variance exhibited by individual binary trees.
- For bagging to work properly, the trees in a bagging ensemble need to be grown to sufficient depth (Bias needs to be minimized).

When would you use Bias-variance?

- To understand the power of your different learning algorithms/ features
- Do you have enough data?
- Do you have enough data scientists?! (variance increases)

Bias Variance Tradeoff

- **For more details:**
  - Lecture 1 from w261
  - http://www-scf.usc.edu/~csci567/17-18-bias-variance.pdf
  - http://www.eecs.berkeley.edu/~jegonzal/talks/linear_regression.pdf

K-Fold Cross Fold Validations

https://en.wikipedia.org/wiki/Cross-validation_(statistics)

Q 1

Suppose you are facing a supervised learning problem and have a very large data set (m=100,000,000). How can you tell if using all the data is likely to perform much better than using a small subset of the data say, (m=1,000)? As a data scientist, what would you do?

Quiz

Select one:

Quiz

a. There is no need to verify this; using a larger dataset always gives much better performance.

b. Plot $J_{train}(\theta)$ as a function of the number of iterations of the optimization algorithm (such as gradient descent). Note: Where $J_{train}(\theta)$ denotes the error over the training data.

c. Plot a learning curve ($J_{train}(\theta)$ and $J_{CV}(\theta)$ plotted as a a function of m) for some range of values of m (say up to m=1,000) and verify that the algorithm has bias when m is small. Note: Where $J_{train}(\theta)$ and $J_{CV}(\theta)$ denotes the error over the training data and cross validation set respectively. ✗

d. Plot a learning curve for a range of values of m and verify that the algorithm has high variance when m is small.

(D)

Q 1

Suppose you are facing a supervised learning problem and have a very large data set (m=100,000,000). How can you tell if using all the data is likely to perform much better than using a small subset of the data say, (m=1,000)? As a data scientist, what would you do?

Quiz

Select one:

Quiz

a. There is no need to verify this; using a larger dataset always gives much better performance.

b. Plot $J_{train}(\theta)$ as a function of the number of iterations of the optimization algorithm (such as gradient descent). Note: Where $J_{train}(\theta)$ denotes the error over the training data.

c. Plot a learning curve ($J_{train}(\theta)$ and $J_{CV}(\theta)$ plotted as a a function of m) for some range of values of m (say up to m=1,000) and verify that the algorithm has bias when m is small. Note: Where $J_{train}(\theta)$ and $J_{CV}(\theta)$ denotes the error over the training data and cross validation set respectively. ✗

(D)

d. Plot a learning curve for a range of values of m and verify that the algorithm has high variance when m is small.

The correct answer is:
Plot a learning curve for a range of values of m and verify that the algorithm has high variance when m is small.

.

- **End of Lecture**