Consolidation and Review #2

Applied Machine Learning
Derek Hoiem

Let's talk about $\boldsymbol{X}$

$$\theta^* = \underset{\theta}{\operatorname{argmin}}\, Loss(f(\boldsymbol{X}; \theta), \boldsymbol{y})$$

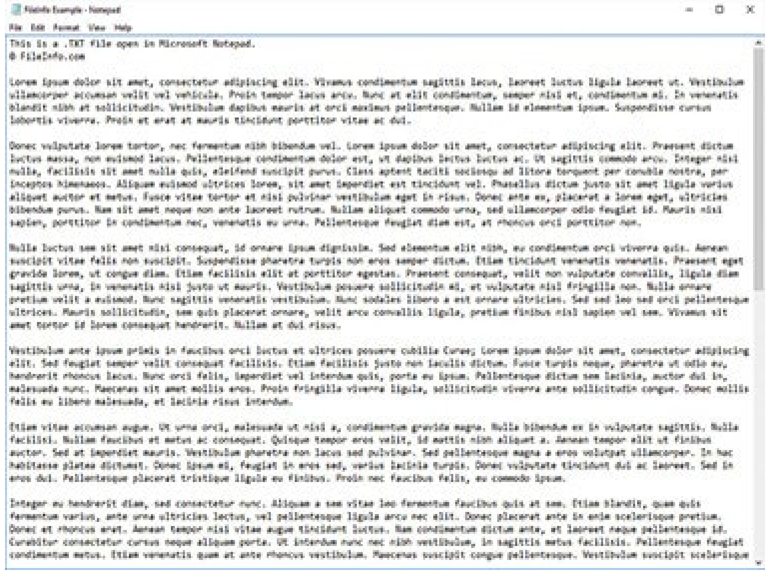
What is data?

- Information that helps us make decisions
- Numbers (bits)

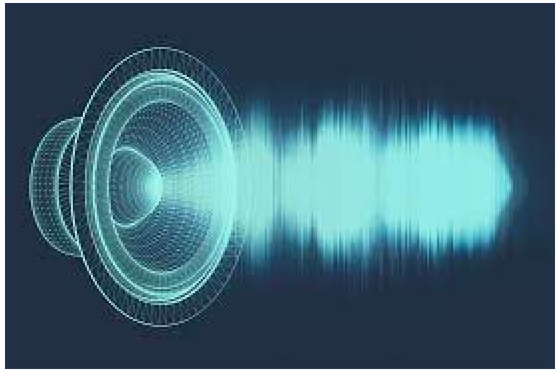
How do we represent data?

- As humans: media we can see, read, and hear
  - Words, imagery, sounds, tables, plots

https://www.rd.com/list/funny-photos/



https://fileinfo.com/extension/txt



https://www.canto.com/blog/audio-file-types/

Sometimes, we can transform the data while preserving much or all of the information

- Resize an image
- Rephrase a paragraph
- 1.5x an audio book

Sometimes, we can even transform the data so that it is more informative

- Perform denoising on an image
- Identify key points and insights in a document
- Remove background noise from audio
- None of these operations add information to the data, but they re-organize and/or remove distracting information

In computers, data are numbers

- The numbers do not “mean” anything by themselves
- The meaning comes from the way the numbers were produced and how they can inform
- The meaning can be contained in each number by itself, or commonly by patterns in groups of numbers

Sometimes, we can transform the data while preserving much or all of the information

- Add or multiply by a constant value
- Represent as a 16-bit or 32-bit float or integer
- Compress a document, or store in a different file format

Sometimes, we can even transform the data so that it is more informative

- Center and rescale images of digits so they are easier to compare to each other

- Normalize (subtract means and divide by standard deviations) cancer cell measurements to make simple similarity measures better reflect malignancy

- Select features or create new ones out of combinations of inputs

Sometimes, we change the structure of data to make it easier to process

Image as matrix

0.92	0.93	0.94	0.97	0.62	0.37	0.85	0.97	0.93	0.92	0.99
0.95	0.89	0.82	0.89	0.56	0.31	0.75	0.92	0.81	0.95	0.91
0.89	0.72	0.51	0.55	0.51	0.42	0.57	0.41	0.49	0.91	0.92
0.96	0.95	0.88	0.94	0.56	0.46	0.91	0.87	0.90	0.97	0.95
0.71	0.81	0.81	0.87	0.57	0.37	0.80	0.88	0.89	0.79	0.85
0.49	0.62	0.60	0.58	0.50	0.60	0.58	0.50	0.61	0.45	0.33
0.86	0.84	0.74	0.58	0.51	0.39	0.73	0.92	0.91	0.49	0.74
0.96	0.67	0.54	0.85	0.48	0.37	0.88	0.90	0.94	0.82	0.93
0.69	0.49	0.56	0.66	0.43	0.42	0.77	0.73	0.71	0.90	0.99
0.79	0.73	0.90	0.67	0.33	0.61	0.69	0.79	0.73	0.93	0.97
0.91	0.94	0.89	0.49	0.41	0.78	0.78	0.77	0.89	0.99	0.93

Convenient for local pattern analysis

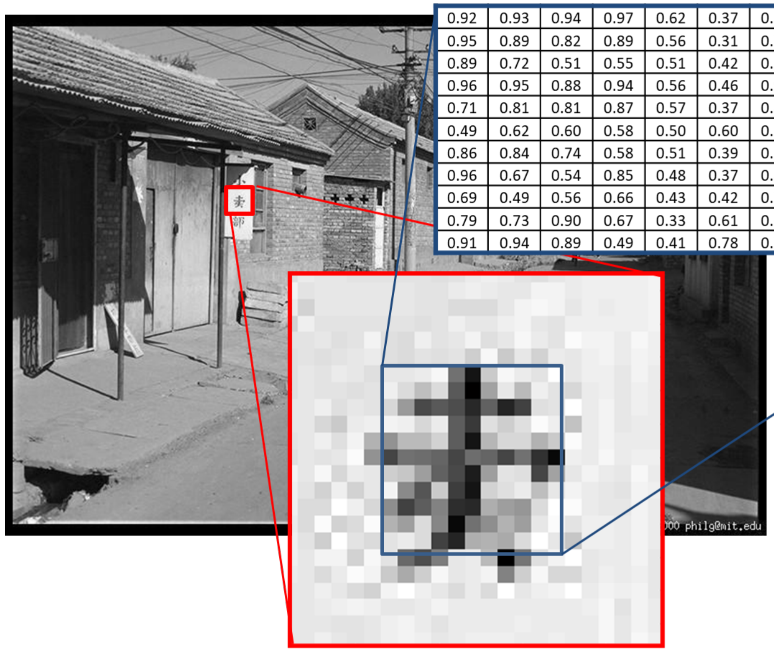
Image as vector

0.92
0.95
0.89
0.96
0.71
0.49
0.86
0.96
0.69
0.79
0.91
0.93
0.89
0.72
0.95
0.81
0.62
0.84
0.67
0.49
0.73
0.94
…
0.93

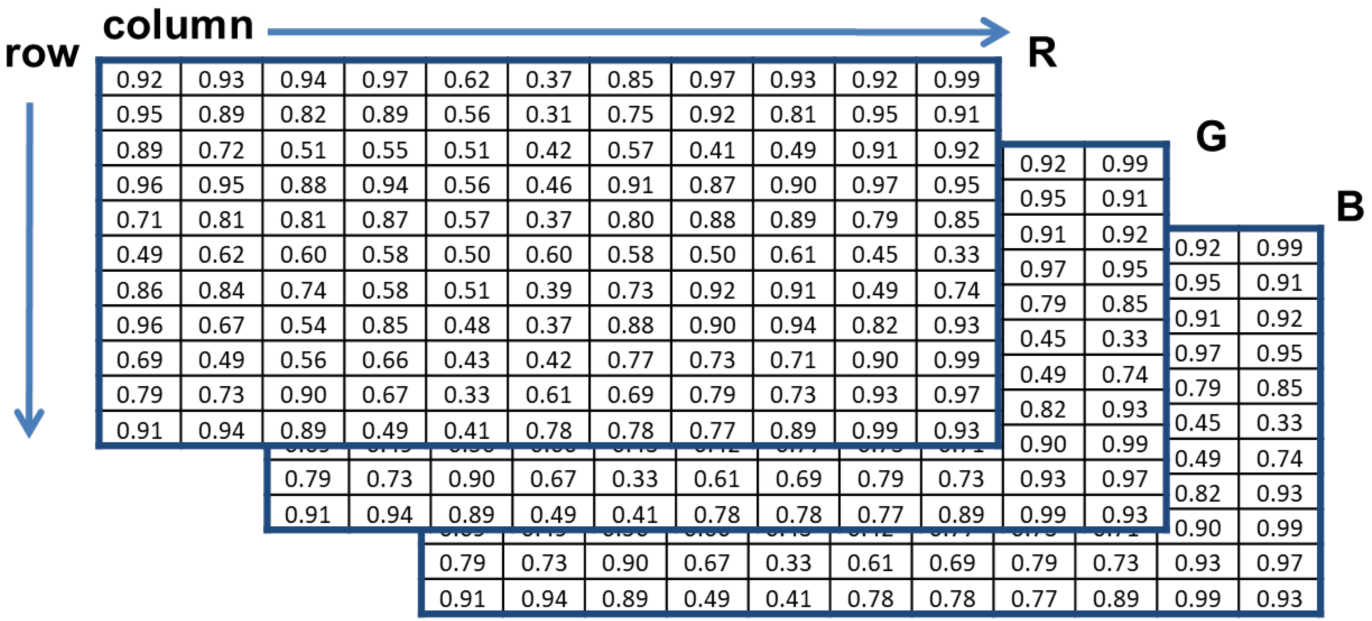
Convenient for linear projection

This does not change the information in the data, but it makes it harder to understand by people and more/less convenient for certain kinds of processing

Images are represented as 3D matrices (row, col, color)



0.92	0.93	0.94	0.97	0.62	0.37	0.85	0.97	0.93	0.92	0.99
0.95	0.89	0.82	0.89	0.56	0.31	0.75	0.92	0.81	0.95	0.91
0.89	0.72	0.51	0.55	0.51	0.42	0.57	0.41	0.49	0.91	0.92
0.96	0.95	0.88	0.94	0.56	0.46	0.91	0.87	0.90	0.97	0.95
0.71	0.81	0.81	0.87	0.57	0.37	0.80	0.88	0.89	0.79	0.85
0.49	0.62	0.60	0.58	0.50	0.60	0.58	0.50	0.61	0.45	0.33
0.86	0.84	0.74	0.58	0.51	0.39	0.73	0.92	0.91	0.49	0.74
0.96	0.67	0.54	0.85	0.48	0.37	0.88	0.90	0.94	0.82	0.93
0.69	0.49	0.56	0.66	0.43	0.42	0.77	0.73	0.71	0.90	0.99
0.79	0.73	0.90	0.67	0.33	0.61	0.69	0.79	0.73	0.93	0.97
0.91	0.94	0.89	0.49	0.41	0.78	0.78	0.77	0.89	0.99	0.93



Text can be represented as a sequence of integers

- Each character can map to a byte value, and then we have a sequence of bytes

  `“Dog ate” → [4 15 7 27 1 20 5]`

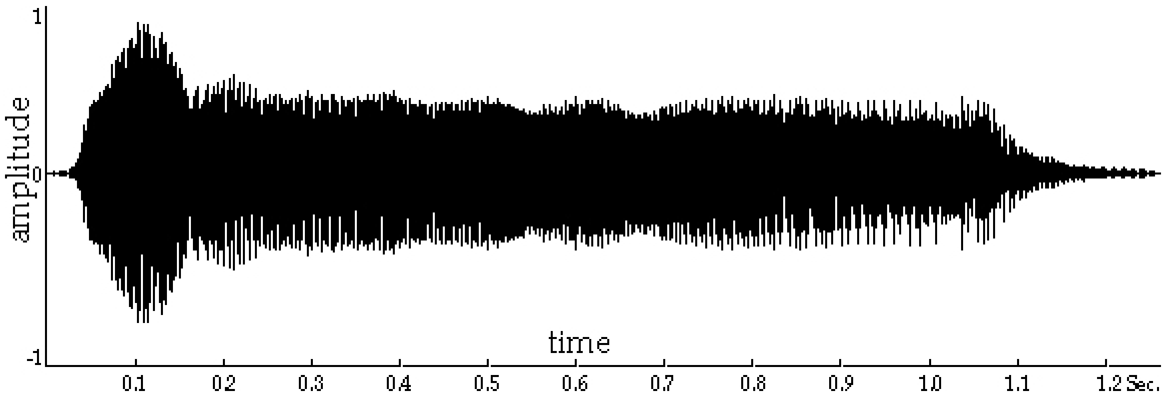
- Each complete word can map to an integer value, and we have a sequence of integers

  `“Dog ate” → [437 1256]`

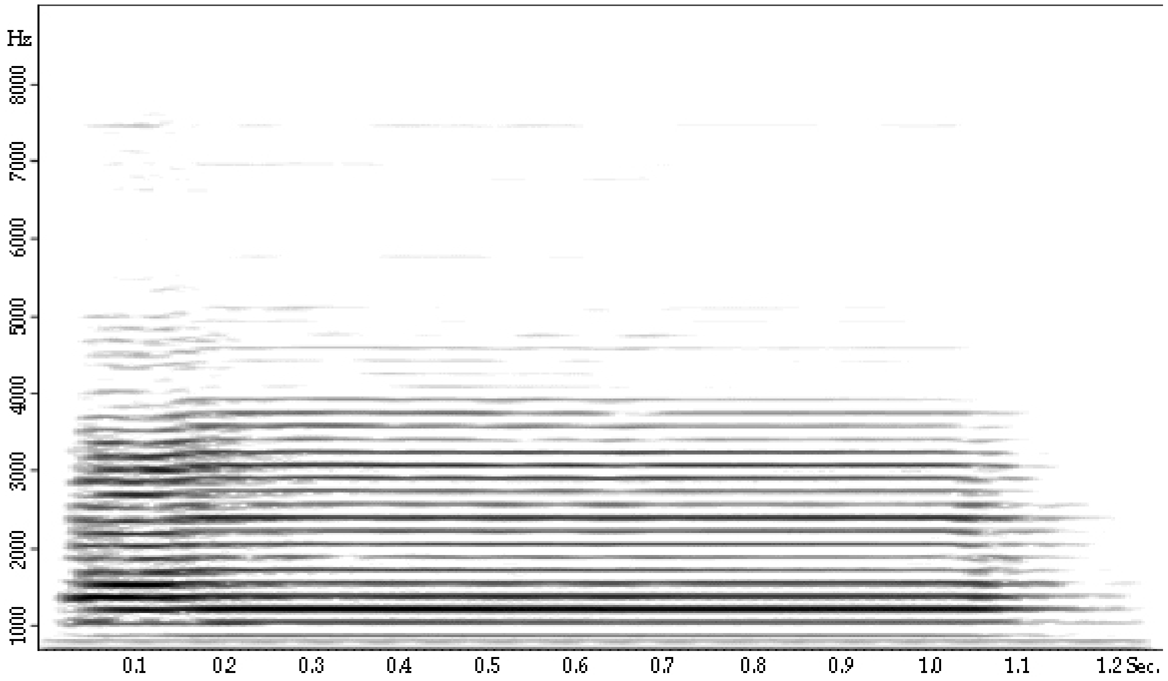
- Common groups of letters can be mapped to subwords and then to integers

  `“Bedroom 1521” → [bed-room- -1-5-2-1]→[125 631 27 28 32 29 27]`

Audio can be represented as a waveform or spectrum



Amplitude vs Time



Frequency-Amplitude vs Time

Fig source

Other kinds of data

- Measurements and continuous values typically represented as floating point numbers
  – Temperature, length, area, dollars

- Categorical values represented as integers
  - Happy/Indifferent/Sad → 0/1/2
  - Red/Green/Blue/Orange → 0/1/2/3/4

- Different kinds of values (text, images, measurements) can be reshaped and concatenated into a long feature vector

The same information content can be represented in many ways. If the original numbers can be recovered, then a change in representation does not change the information content.

All types of data can be stored as 1D vectors/arrays.

Matrices and other data structures make code easier to program and read.

From data point to data set

$\boldsymbol{x} = \{x_0, \ldots x_M\} \sim D$: x is an M-dimensional vector drawn from some distribution D

We can sample many $\boldsymbol{x}$ (e.g. download documents from the Internet, take pictures, take measurements) to get
$\boldsymbol{X} = \{\boldsymbol{x_0}, \ldots, \boldsymbol{x_N})$

We may repeat this collection multiple times, or collect one large dataset and randomly sample it to get
$\boldsymbol{X_{train}}, \boldsymbol{X_{test}}$

Typically, we assume that all of the data samples within $\boldsymbol{X_{train}}$ and $\boldsymbol{X_{test}}$ come from the same distribution and are independent of each other. That means, e.g. that $\boldsymbol{x_0}$ does not tell us anything about $\boldsymbol{x_1}$ if we already know the sampling distribution D

Consider an example from the penguins dataset

	species	island	culmen_length_mm	culmen_depth_mm	flipper_length_mm	body_mass_g	sex
0	Adelie	Torgersen	39.1	18.7	181	3750	MALE
1	Adelie	Torgersen	39.5	17.4	186	3800	FEMALE
2	Adelie	Torgersen	40.3	18.0	195	3250	FEMALE
3	Adelie	Torgersen	36.7	19.3	193	3450	FEMALE
4	Adelie	Torgersen	39.3	20.6	190	3650	MALE
5	Adelie	Torgersen	38.9	17.8	181	3625	FEMALE
6	Adelie	Torgersen	39.2	19.6	195	4675	MALE
7	Adelie	Torgersen	34.1	18.1	193	3475	Unknown
8	Adelie	Torgersen	42.0	20.2	190	4250	Unknown
9	Adelie	Torgersen	37.8	17.1	186	3300	Unknown

Convert the data into numbers

```
df_penguins = pd.read_csv(datadir + 'penguins_size.csv')
df_penguins.head(10)

# convert features with multiple string values to binary features so they can be used by sklearn
def get_penguin_xy(df_penguins):
  data = np.array(df_penguins[['island', 'culmen_length_mm', 'culmen_depth_mm', 'flipper_length_mm’, \
                                   'body_mass_g', 'sex']])
  y = df_penguins['species']
  ui = np.unique(data[:,0]) # unique island
  us = np.unique(data[:,-1]) # unique sex
  X = np.zeros((len(y), 10))
  for i in range(len(y)):
    f = 0
    for j in range(len(ui)): # replace island name with three indicator variables
      if data[i, f]==ui[j]:
        X[i, f+j] = 1
    f = f + len(ui)
    X[i, f:(f+4)] = data[i, 1:5] # copy original measurement features
    f=f+4
    for j in range(len(us)): # replace sex with three indicator variables (male/female/unknown)
      if data[i, 5]==us[j]:
        X[i, f+j] = 1
  feature_names = ['island_biscoe', 'island_dream', 'island_torgersen', 'culmen_length_mm’, \
                 'culmen_depth_mm', 'flipper_length_mm', 'body_mass_g', 'sex_female', 'sex_male', 'sex_unknown'
  X = pd.DataFrame(X, columns=feature_names)
  return(X, y, feature_names, np.unique(y))
```

How do we measure $\boldsymbol{X}$?

- We can check the number of samples and dimensions

```
X.shape
```

```
(341, 10)
```

- We can measure the distribution with statistics

```
X.mean(axis=0)
```

```
island_biscoe              0.486804
island_dream               0.363636
island_torgersen           0.149560
culmen_length_mm          43.920235
culmen_depth_mm           17.155425
flipper_length_mm        200.868035
body_mass_g             4199.780059
sex_female                 0.483871
sex_male                   0.492669
sex_unknown                0.023460
dtype: float64
```

```
X.std(axis=0)
```

```
island_biscoe              0.500560
island_dream               0.481753
island_torgersen           0.357164
culmen_length_mm           5.467516
culmen_depth_mm            1.976124
flipper_length_mm         14.055255
body_mass_g              802.300201
sex_female                 0.500474
sex_male                   0.500681
sex_unknown                0.151583
dtype: float64
```

Different samples will give us different measurements of the distribution

```
X.sample(100, replace=True).mean(axis=0)
```

```
island_biscoe           0.450
island_dream            0.380
island_torgersen        0.170
culmen_length_mm       43.369
culmen_depth_mm        17.543
flipper_length_mm     199.020
body_mass_g          4106.500
sex_female              0.450
sex_male                0.510
sex_unknown             0.040
dtype: float64
```

```
X.sample(100, replace=True).mean(axis=0)
```

```
island_biscoe           0.540
island_dream            0.310
island_torgersen        0.150
culmen_length_mm       43.970
culmen_depth_mm        16.908
flipper_length_mm     201.120
body_mass_g          4211.250
sex_female              0.490
sex_male                0.490
sex_unknown             0.020
dtype: float64
```

```
X.sample(100, replace=True).mean(axis=0)
```

```
island_biscoe           0.440
island_dream            0.340
island_torgersen        0.220
culmen_length_mm       43.412
culmen_depth_mm        17.342
flipper_length_mm     200.780
body_mass_g          4232.250
sex_female              0.420
sex_male                0.540
sex_unknown             0.040
dtype: float64
```

The estimates from larger sample sizes will vary less

```
X.sample(1000, replace=True).mean(axis=0)
```

```
island_biscoe           0.4750
island_dream            0.3680
island_torgersen        0.1570
culmen_length_mm       43.7581
culmen_depth_mm        17.1759
flipper_length_mm     200.4740
body_mass_g          4164.7000
sex_female              0.4770
sex_male                0.5060
sex_unknown             0.0170
dtype: float64
```

```
X.sample(1000, replace=True).mean(axis=0)
```

```
island_biscoe           0.4730
island_dream            0.3800
island_torgersen        0.1470
culmen_length_mm       43.6959
culmen_depth_mm        17.1774
flipper_length_mm     200.1530
body_mass_g          4138.8750
sex_female              0.5070
sex_male                0.4760
sex_unknown             0.0170
dtype: float64
```

```
X.sample(1000, replace=True).mean(axis=0)
```

```
island_biscoe           0.4870
island_dream            0.3740
island_torgersen        0.1390
culmen_length_mm       44.0078
culmen_depth_mm        17.1370
flipper_length_mm     200.8820
body_mass_g          4190.0750
sex_female              0.5010
sex_male                0.4860
sex_unknown             0.0130
dtype: float64
```

How do we measure $\boldsymbol{X}$?

- We can measure the entropy of a particular variable:

$H(x) = -\sum_k [P(x=k) \log P(x=k)]$ (if x is discrete, i.e. finite number of possible values)

```
i=0
print(feature_names[i])
xi = X.iloc[:, i]
print(np.unique(xi))
pxi_0 = np.mean(xi==0)
pxi_1 = np.mean(xi==1)
hxi = -pxi_0 * np.log2(pxi_0) + -pxi_1 * np.log2(pxi_1)
print('P(xi=0)={:0.3f}  P(xi=1)={:0.3f}   H(xi)={:0.3f}'.format(pxi_0, pxi_1, hxi))
```

```
island_biscoe
[0. 1.]
P(xi=0)=0.513  P(xi=1)=0.487   H(xi)=0.999
```

```
i=2
print(feature_names[i])
xi = X.iloc[:, i]
print(np.unique(xi))
pxi_0 = np.mean(xi==0)
pxi_1 = np.mean(xi==1)
hxi = -pxi_0 * np.log2(pxi_0) + -pxi_1 * np.log2(pxi_1)
print('P(xi=0)={:0.3f}  P(xi=1)={:0.3f}   H(xi)={:0.3f}'.format(pxi_0, pxi_1, hxi))
```

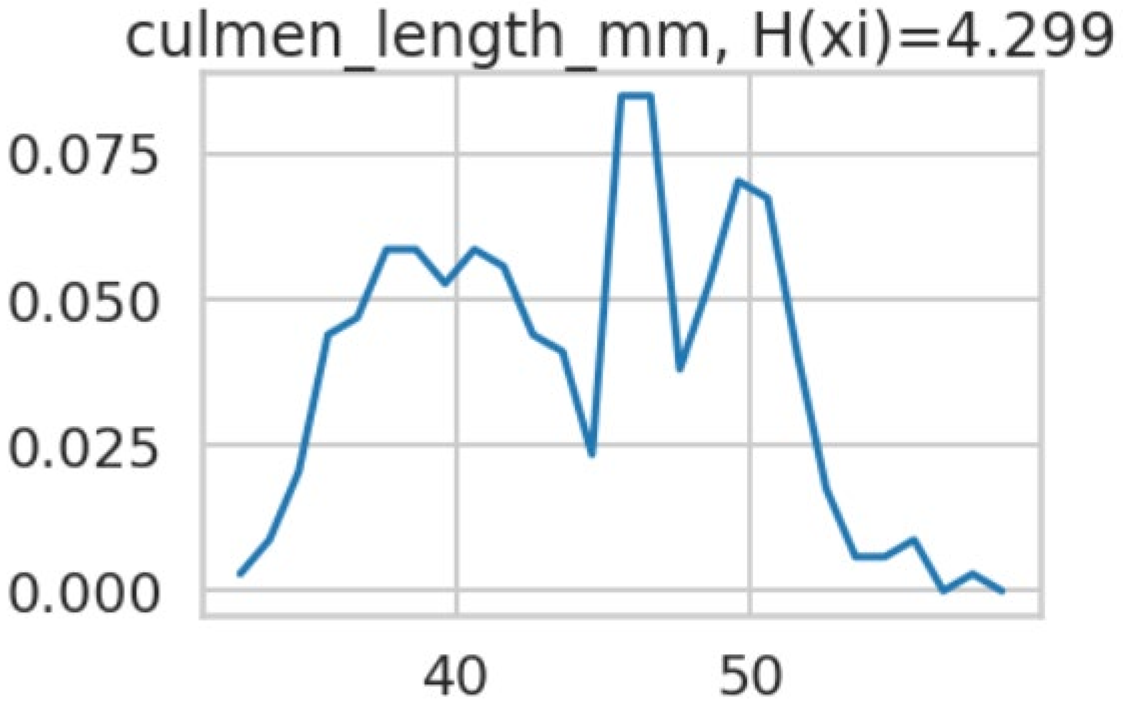
```
island_torgersen
[0. 1.]
P(xi=0)=0.850  P(xi=1)=0.150   H(xi)=0.609
```

How do we measure $\boldsymbol{X}$?

- We can measure the entropy of a particular variable:

$$H(x) = -\int p(x)\log(p(x))\,dx$$ (if x is continuous)

```
i=3
print(feature_names[i])
xi = X.iloc[:, i]
print(len(np.unique(xi)))
xval = []
pxi = []
step = 1
for k in np.arange(xi.min()+step/2, xi.max()-step/2, step):
  xval.append(k)
  pxi.append(np.mean(np.logical_and(xi>=k-step/2,xi<k+step/2)))
pxi = np.array(pxi)/step+1E-20
hxi = np.sum(-pxi*np.log2(pxi)*step)
plt.plot(xval, pxi)
plt.title('{}, H(xi)={:0.3f}'.format(feature_names[i], hxi))
```

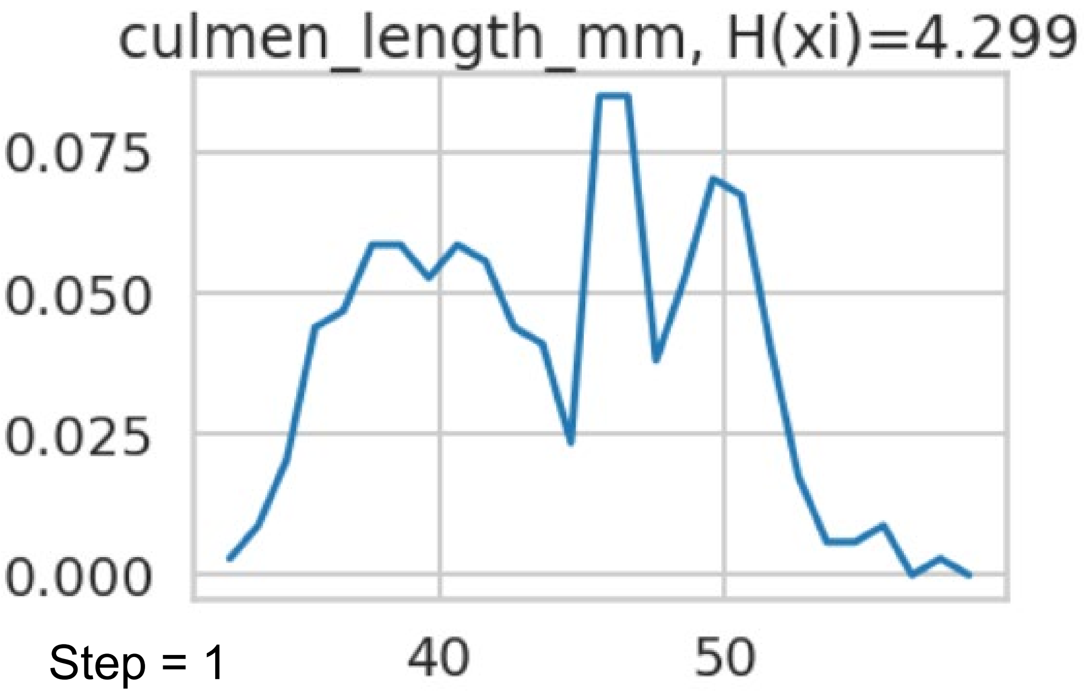


How do we measure $\boldsymbol{X}$?

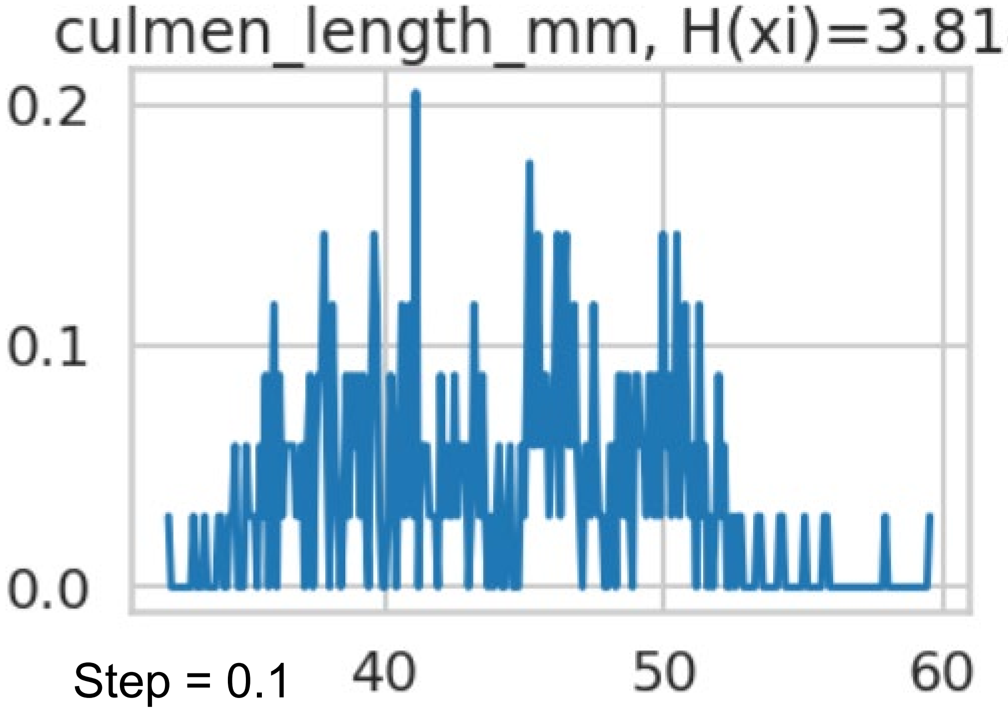
- We can measure the entropy of a particular variable:

$H(x) = -\int p(x)\log(p(x))$ (if x is continuous)

But probability densities and entropy of continuous variables are tricky to estimate



Step = 1



Step = 0.1

Entropy measures how many bits are required to store an element of data

Does this mean that entropy is a measure of information?

Does a random array contain information?

Information gain: IG(y|x) = H(y)-H(y|x)

- Information gain measures how much a variable x reduces the entropy of y when known, i.e. how many fewer bits are needed to encode y given the value of x

$$
\begin{aligned}
H(Y|X) &= \sum_{x\in X} p(x)H(Y|X=x) \\
&= -\sum_{x\in X}\sum_{y\in Y} p(x,y)\log_2 p(y|x)
\end{aligned}
$$

```
# Information gain of X wrt male/female
i=0
print(feature_names[i])
xi = X.iloc[:, i]
y =  X.loc[:,'sex_male']-X.loc[:,'sex_female'] # 1 for male, -1 for female
xi_m = xi[y==1]
xi_f = xi[y==-1]
N = (np.sum(y==1)+np.sum(y==-1))
py = np.sum(y==1) / N # P(y=male)
print(py)
Hy = -py*np.log2(py) - (1-py)*np.log2(1-py) # Entropy(y)
pxi_0 = (np.sum(xi_m==0) + np.sum(xi_f==0))/N
py1_x0 = np.sum(xi_m==0) / (np.sum(xi_m==0) + np.sum(xi_f==0)) # P(male | x=0)
py1_x1 = np.sum(xi_m==1) / (np.sum(xi_m==1) + np.sum(xi_f==1)) # P(male | x=1)
Hyx = pxi_0*(-py1_x0*np.log2(py1_x0) - (1-py1_x0)*np.log2(1-py1_x0)) + \
      (1-pxi_0)*(-py1_x1*np.log2(py1_x1) - (1-py1_x1)*np.log2(1-py1_x1))
IGyx = Hy - Hyx
print('H(y)={:0.4f}   H(y|x)={:0.4f}  IG(y|x)={:0.4f}'.format(Hy, Hyx, IGyx))
```

```
island_biscoe
0.5045045045045045
H(y)=0.9999   H(y|x)=0.9999  IG(y|x)=0.0001
```

Knowing the island is Biscoe tells us very little about whether a penguin is likely to be male or female

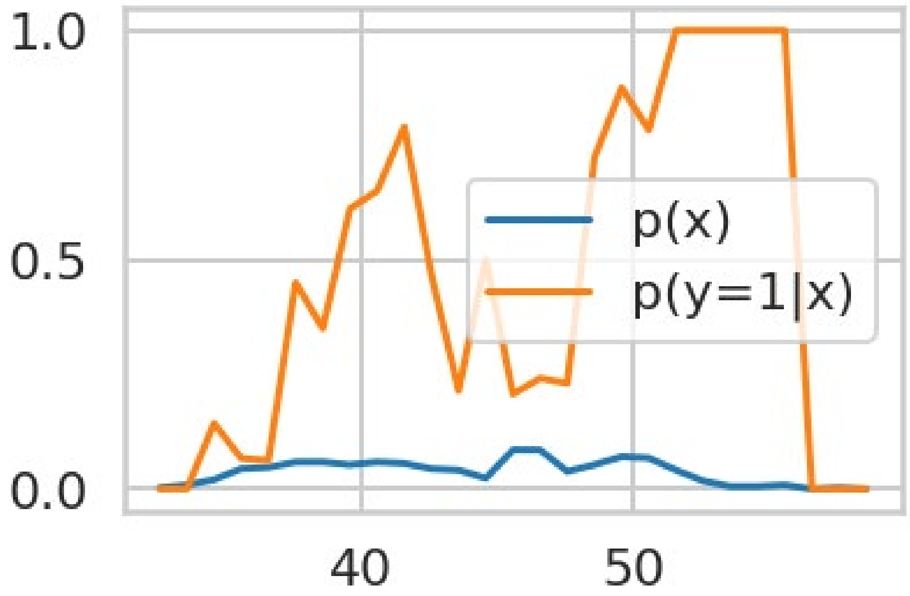
Information gain: IG(y|x) = H(y)-H(y|x)

- Also applies when x is continuous

```
# Information gain of continuous x wrt male/female
i=3
print(feature_names[i])
xi = X.iloc[:, i]
y =  X.loc[:,'sex_male']-X.loc[:,'sex_female'] # 1 for male, -1 for female
N = np.sum(y==1) + np.sum(y==-1)
xi_m = xi[y==1]
xi_f = xi[y==-1]
px = []
py1_x = []
step = 1
xval = np.arange(xi.min()+step/2, xi.max()-step/2, step)
for k in xval:
  px.append(np.mean(np.logical_and(xi>=k-step/2,xi<k+step/2)))
  py1_x.append(np.mean((xi>=k-step/2) & (xi<k+step/2) & (y==1)) / (px[-1]+1E-40))

eps = 1E-40
px = np.array(px)
py1_x = np.array(py1_x)
plt.plot(xval, px/step)
plt.plot(xval, py1_x)
plt.legend(('p(x)', 'p(y=1|x)'))
Hy = -py*np.log2(py+eps) - (1-py)*np.log2(1-py+eps) # Entropy(y)
Hyx = -np.sum(px*py1_x*np.log2(py1_x+eps))-np.sum(px*(1-py1_x)*np.log2(1-py1_x+eps))
IGyx = Hy - Hyx
print('H(y)={:0.4f}   H(y|x)={:0.4f}  IG(y|x)={:0.4f}'.format(Hy, Hyx, IGyx))
```

```
culmen_length_mm
H(y)=0.9999   H(y|x)=0.6959  IG(y|x)=0.3040
```

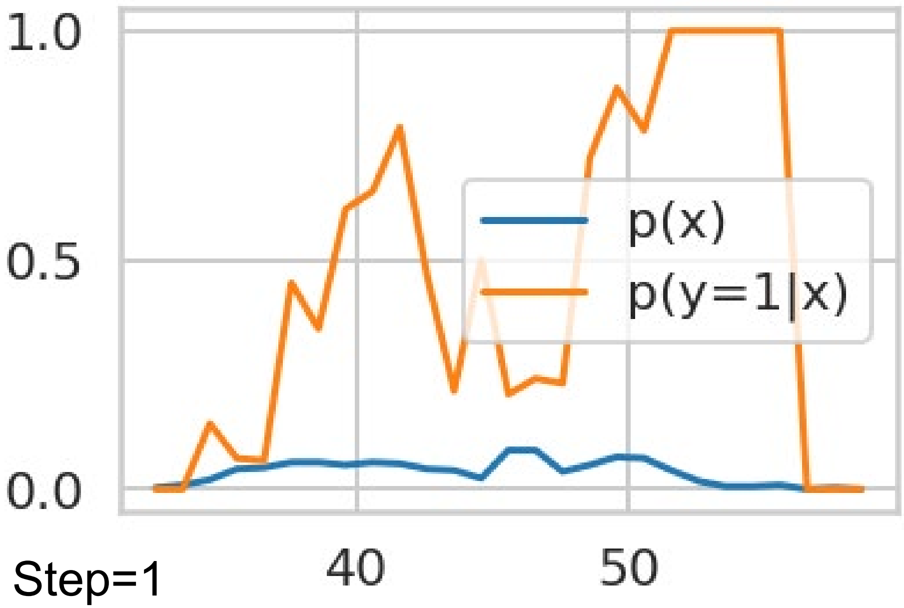


IG(y|x) = 0.304

Knowing the culmen length tells us a lot whether a penguin is likely to be male or female. Large culmens are always male, but smaller ones could be male (maybe young) or female.

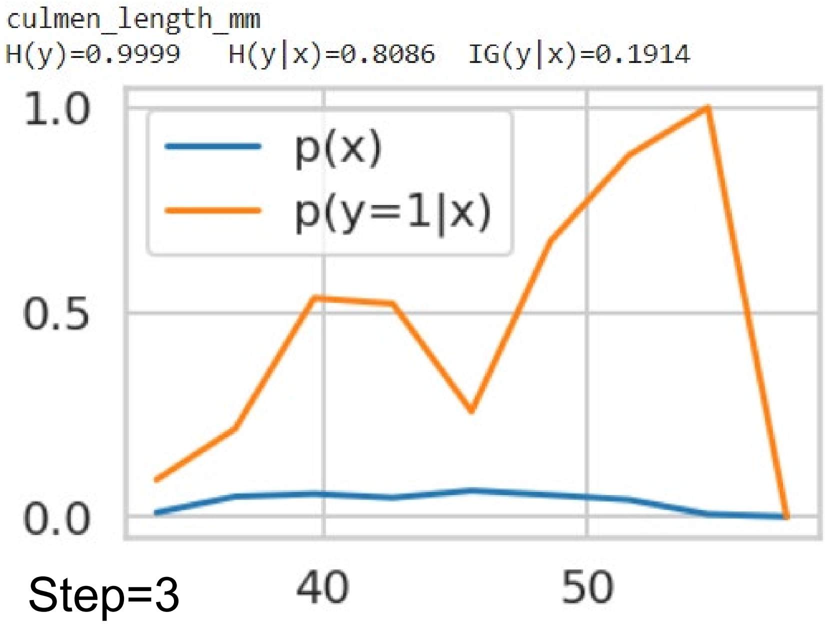
Information gain: $IG(y|x) = H(y)-H(y|x)$

- Again, details on how continuous distribution is estimated can lead to different information gains



Step=1

IG(y|x) = 0.304



Step=3

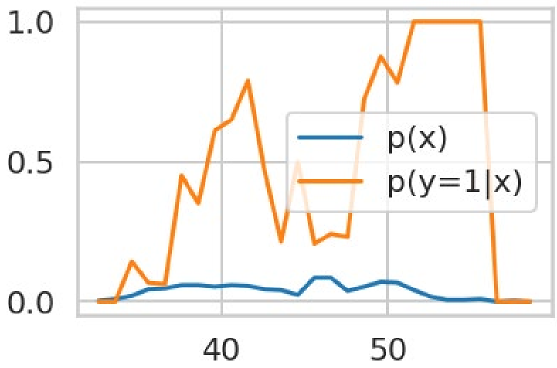
IG(y|x) = 0.1904

How can the information gain be different depending our step size?

- We have only an *empirical estimate* (based on observed samples) of probabilities used to compute information gain
- With more data, we could obtain a better estimate
- With continuous variables, there is a trade-off between over-smoothing or simplifying the distribution and making overly confident predictions based on small data samples
- This is another example of the bias-variance trade-off
  - The step size we choose would likely depend on the amount of data available
- The true probability distributions and information gain cannot be known. We can only try to make our best estimate

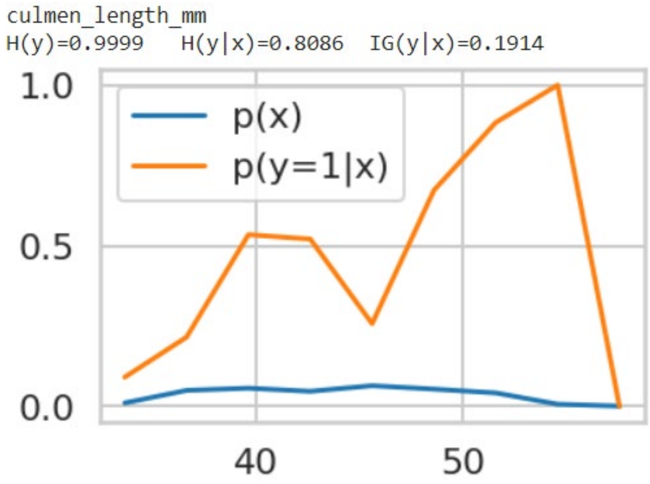
Step=1

IG(y|x) = 0.304



Step=3

IG(y|x) = 0.190



Coming back to

$$\theta^* = \underset{\theta}{\operatorname{argmin}}\, Loss(f(\boldsymbol{X}; \theta), \boldsymbol{y})$$

- The aim is to automatically find a model that predicts y given X
- Probabilistically, this can be viewed as maximizing the information gain of y given X, with constraints/priors to improve robustness to limited data

$$\theta^* = \underset{\theta}{\operatorname{argmin}}[H(y|x;\theta) - H(y) + R(\theta)]$$

$$H(y|x;\theta) = -\int p(x)\log p(y|x)dx \approx \sum_{(x_n,y_n)\in \mathrm{X},\boldsymbol{y}} -\log p(y_n|x_n)$$

- Manually (computer-assisted), we can at most identify how to extract the information from one or two variables for y
- This is why we have machine learning:
  - Encode: automatically transform X into a representation that makes it easier to extract information about y (Often, humans do this part, especially if there is limited data available for learning)
  - Decode: automatically extract information about y from X

The most powerful ML algorithms smoothly combine encoding (feature extraction) with decoding (prediction) and offer controls or protections against overfitting

Random Forests
- Deep trees partition the feature space by optimizing information gain for a subset of features (individually low bias, high variance)
- Vote averaging reduces variance/overfitting

Boosted Trees
- Shallow trees partition the feature space by optimizing information gain (high bias, low variance)
- Each tree is trained on weighted sample to focus on previous mispredictions, so combination reduces bias (but may increase variance if there are many deeper trees, as eventually all the weight will be on the few hardest examples)

Deep Networks
- *End-to-end learning* (gradient flow from prediction to input) enables joint optimization of features and prediction
- Intermediate layers represent transformations of the data that are more easily re-usable than tree partitions
- The structure of the network (max feature width) controls overfitting
- Massive datasets further reduce variance when training “from scratch”

Deep network optimization

- The long-standing challenge in deep (many-layer) neural networks is how to optimize them
- Optimization is by stochastic gradient descent (SGD) and back-propagation
  - where weight updates are computed by summing products of error gradients from input of the weight to the network's output
  - SGD is performed efficiently using back-propagation, a dynamic program that re-uses weight gradient computations at each layer to compute the gradients for the previous layer
- Deep networks are composed of layers and activations
  - Sigmoid activations, traditionally used, have gradients less than 1 everywhere, and often much less than 1, so gradients "vanish" in earlier layers, due to a product of many values less than 1
  - ReLU activations have gradients of 0 or 1 everywhere, so they do not have this problem as much, but you can have "dead" (gradient=0 for input of all/most samples) ReLUs that can hinder optimization
  - Skip connections add the output of one layer to the output of a later layer (gradient=1), enabling error gradients to flow through the entire network
- SGD has many variants and tricks to improve speed and stability of optimization
  - Momentum accelerates the steps when sequential batches produce similar error gradients
  - Gradient path length normalizations prevent focusing too much on a small number of weights
  - Gradient clipping (for example, g = max(min(g, g_max), -gmax)) prevents gradients from "exploding" and improves stability of optimization
  - SGD+momentum and Adam (SGD+momentum+normalization) are most widely used, but more advanced methods are available, such as RANGER (Rectified Adam with gradient centering and look-ahead)

Whose job is ML – human or machine?

$$\theta^* = \underset{\theta}{\operatorname{argmin}}\, Loss(f(\boldsymbol{X};\theta), \boldsymbol{y})$$

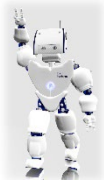
- Problem definition: human

- Objective (*Loss*): human (with automatic validation)

- Data collection/curation (*X*, *y*): mainly human, but less supervised approaches becoming popular to reduce requirements

- Feature encoding (*X*): human or machine, depending on *f*

- Model definition (*f*): human



- Parameters (θ): machine

Shake hands fig

Midterm Exam Logistics

- Mar 9 (exam will be open for most of the day)
- Exam will be 75 minutes long (or longer for those with DRES accommodations)
- Mainly multiple choice / multiple select
  - No coding or complex calculations; mainly tests conceptual understanding
- You take it at home (open book) on PrairieLearn
- **Not cheating**
  - Consult notes, practice questions/answers, slides, internet, etc.
- **Cheating**
  - Talking to a classmate about the exam after one (but not both) of you has taken it
  - Getting help from another person during the exam
- You will not have time to look up all the answers, so do prepare by reviewing slides, lectures, AML book, and practice questions

Midterm Exam Central Topics

- How does train/test error depend on
  - Number of training samples
  - Complexity of model
- Bias-variance trade-off, including meaning of “bias” and “variance” for ML models and “overfitting”
- Basic function/form/assumptions of classification/regression models (KNN, NB, linear/logistic regression, trees, SVMs, boosted trees, random forests, ensembles
- Entropy/Information gain
- SGD and activation layers

Bias-Variance Trade-off

$$\underbrace{E_{\mathbf{x},y,D}\left[\left(h_D(\mathbf{x}) - y\right)^2\right]}_{\text{Expected Test Error}} = \underbrace{E_{\mathbf{x},D}\left[\left(h_D(\mathbf{x}) - \bar{h}(\mathbf{x})\right)^2\right]}_{\text{Variance}} + \underbrace{E_{\mathbf{x},y}\left[\left(\bar{y}(\mathbf{x}) - y\right)^2\right]}_{\text{Noise}} + \underbrace{E_{\mathbf{x}}\left[\left(\bar{h}(\mathbf{x}) - \bar{y}(\mathbf{x})\right)^2\right]}_{\text{Bias}^2}$$

Variance: due to limited data
Different training samples will give different models that vary in predictions for the same test sample

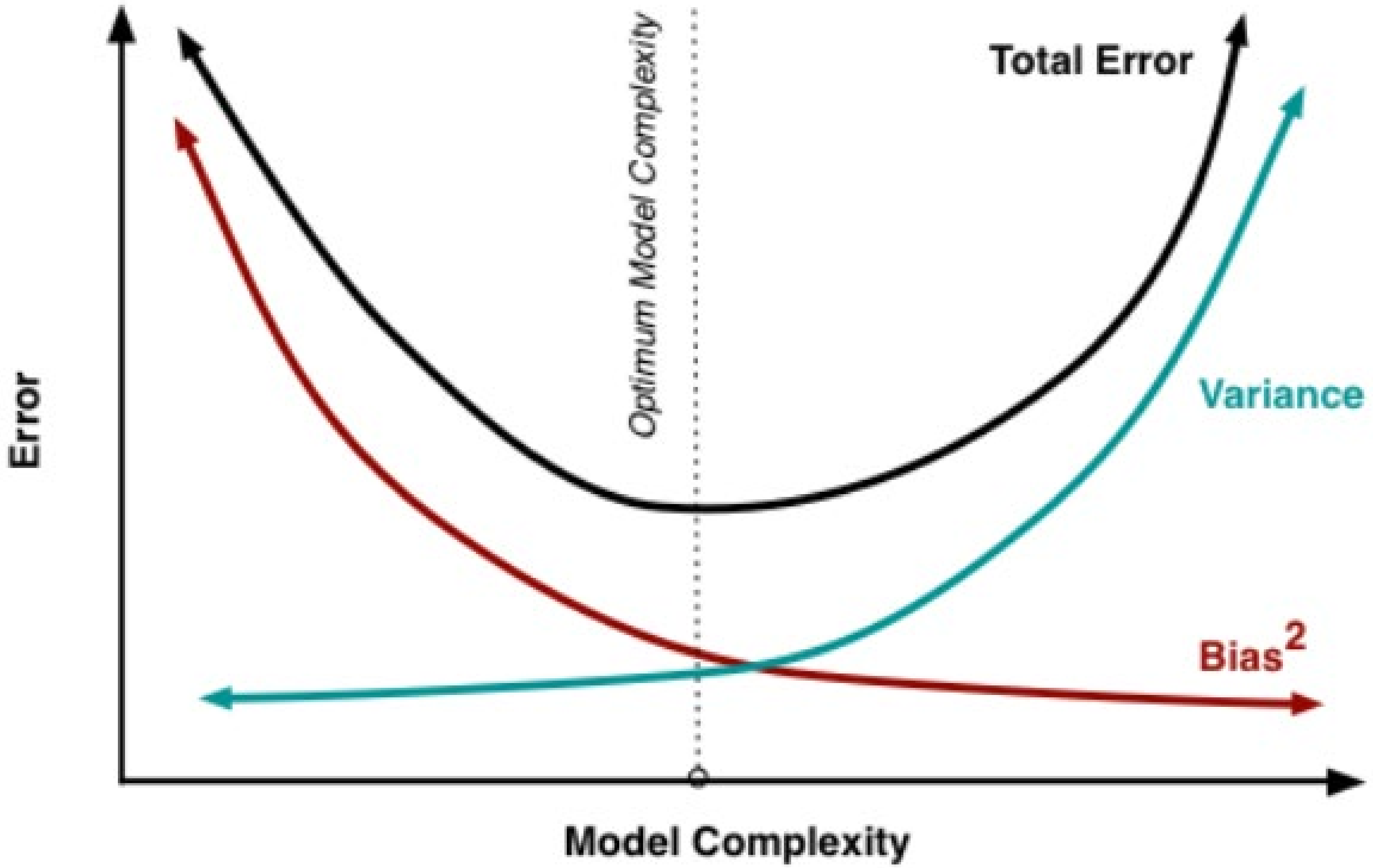
“**Noise**”: irreducible error due to data/problem

Bias: error when optimal model is learned from infinite data

Above is for regression.
But same error = variance + noise + bias^2 holds for classification error and logistic regression.

See this for derivation

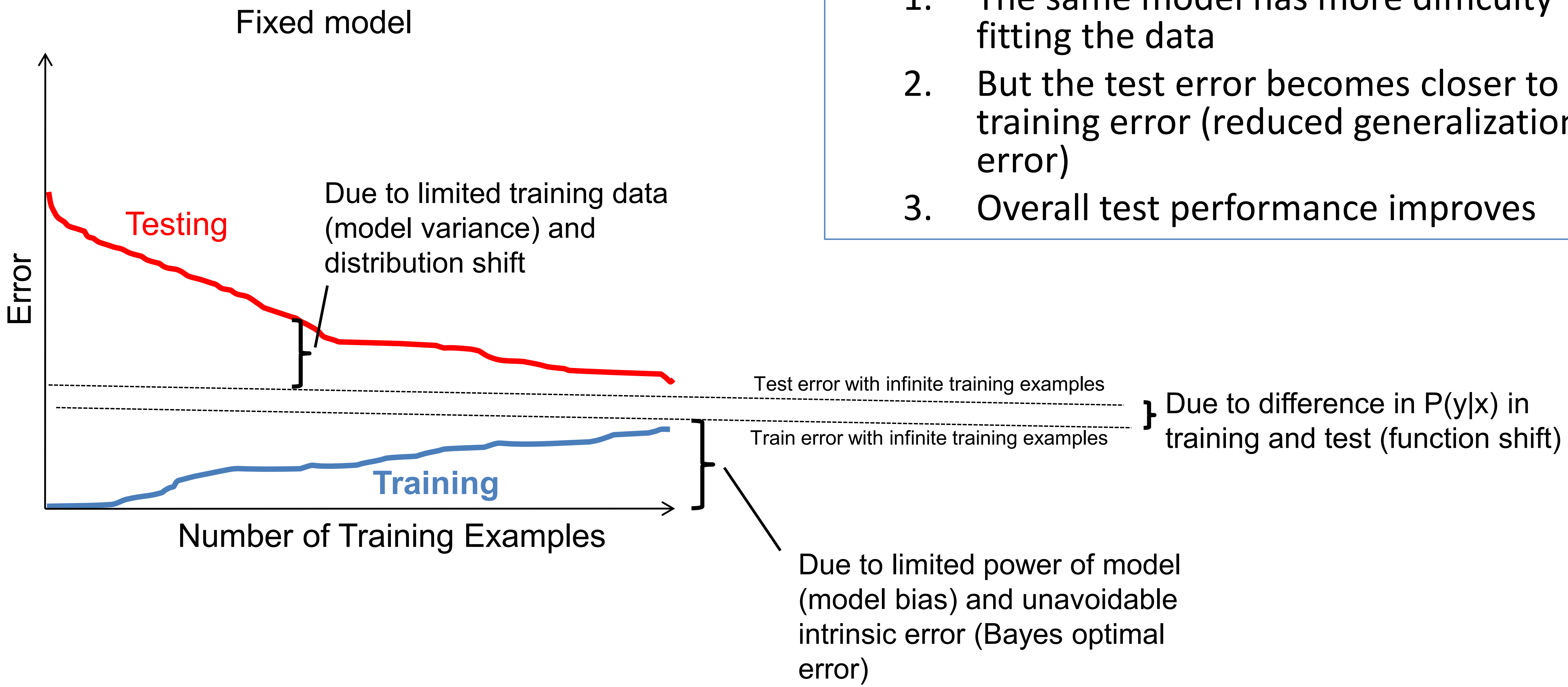
Fig Sources


Total Error
Variance
Bias^2
Optimum Model Complexity
Error
Model Complexity

Performance vs training size

As we get more training data:

1. The same model has more difficulty fitting the data
2. But the test error becomes closer to training error (reduced generalization error)
3. Overall test performance improves



Questions

- What are ways to reduce model bias?
  - More complex model
  - Boosting ensemble
- What are ways to reduce model variance?
  - More training examples
  - Averaging ensemble
  - Simpler model
- Which models are linear (in terms of input features)?
  1. KNN
  2. Linear regression
  3. Linear SVM
  4. SVM with RBF kernel
  5. Decision tree
  6. Random forest
  7. Naïve bayes with Gaussian/multinomial
  8. Perceptron
  9. MLP

  2,3,7,8

Upcoming schedule

- Thursday: CNNs and Vision
- Next week (Feb 27+)
  - HW 2 due, HW3 released
  - Word representations and language models
  - Transformers in vision and language
- Following week (Mar 7+)
  - Foundation models
  - Exam
- Spring break