Working with Data

Applied Machine Learning
Derek Hoiem

Machine learning model maps from features to prediction

$$f(x) \to y$$

Features

Prediction

Examples

- Classification: predict label
  - Is this a dog or a cat?
  - Is this email spam or not?

- Regression: predict value
  - What will the stock price be tomorrow?
  - What will be the high temperature tomorrow?

- Structured prediction
  - What is the pose of this person?

Classification problem

$$f(\boldsymbol{x}; \theta) \rightarrow y$$

Input features   Model parameters   Predicted label

Digit classification example

$\boldsymbol{x}$

y 3   8   7   9   9   0   1   1   5   2

Developing a classifier involves training and testing with training, validation, and test data

- *Training* data: used to fit parameters of the model
- *Validation* data: used to select the best model and any parameters that need to be manually set (“hyperparameters”)
- *Test* data: used to evaluate the final version of the model

This lecture

- Representing data points
- Data sets
- Measuring data and information

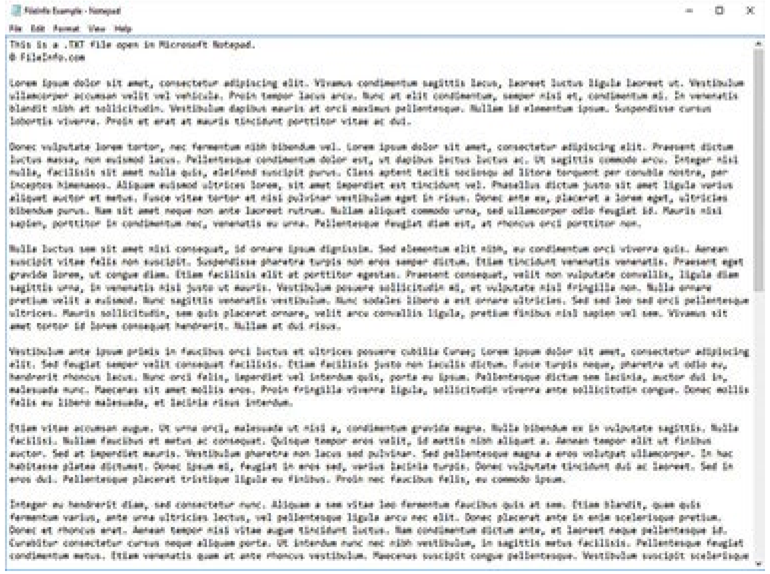
What is data?

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

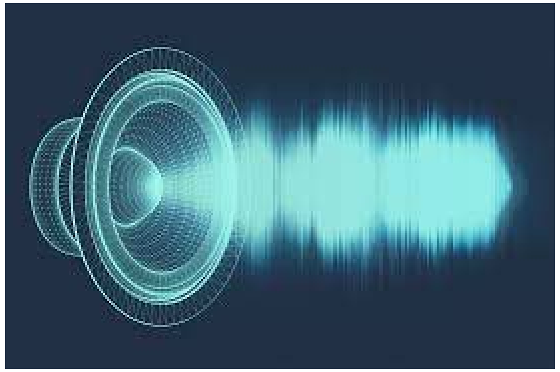
How do we represent data?

- 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

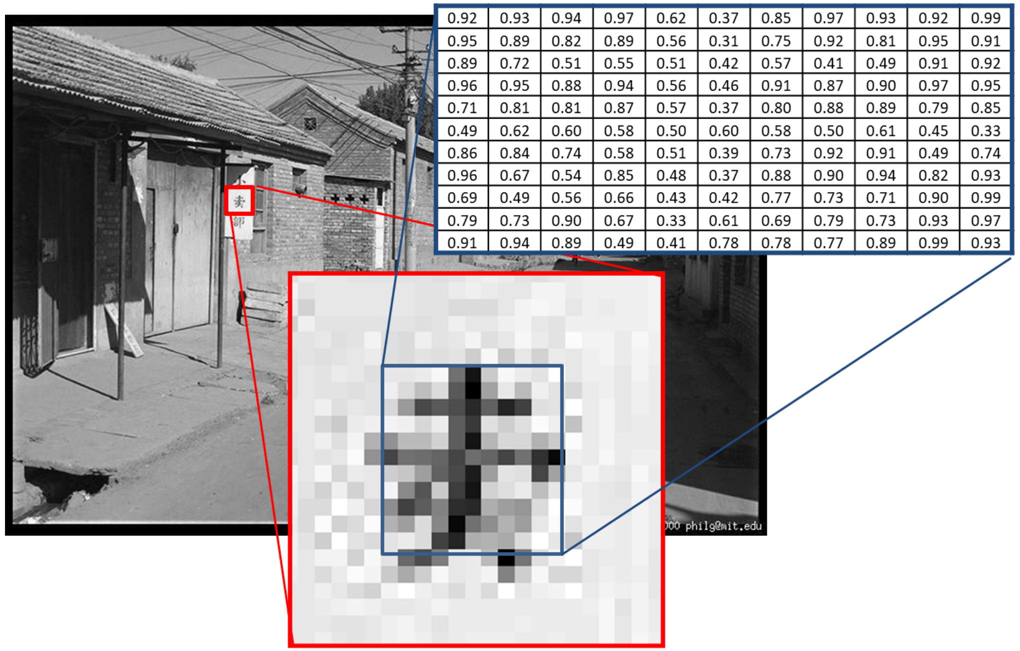
Sometimes, we can even transform the data so that it is more informative

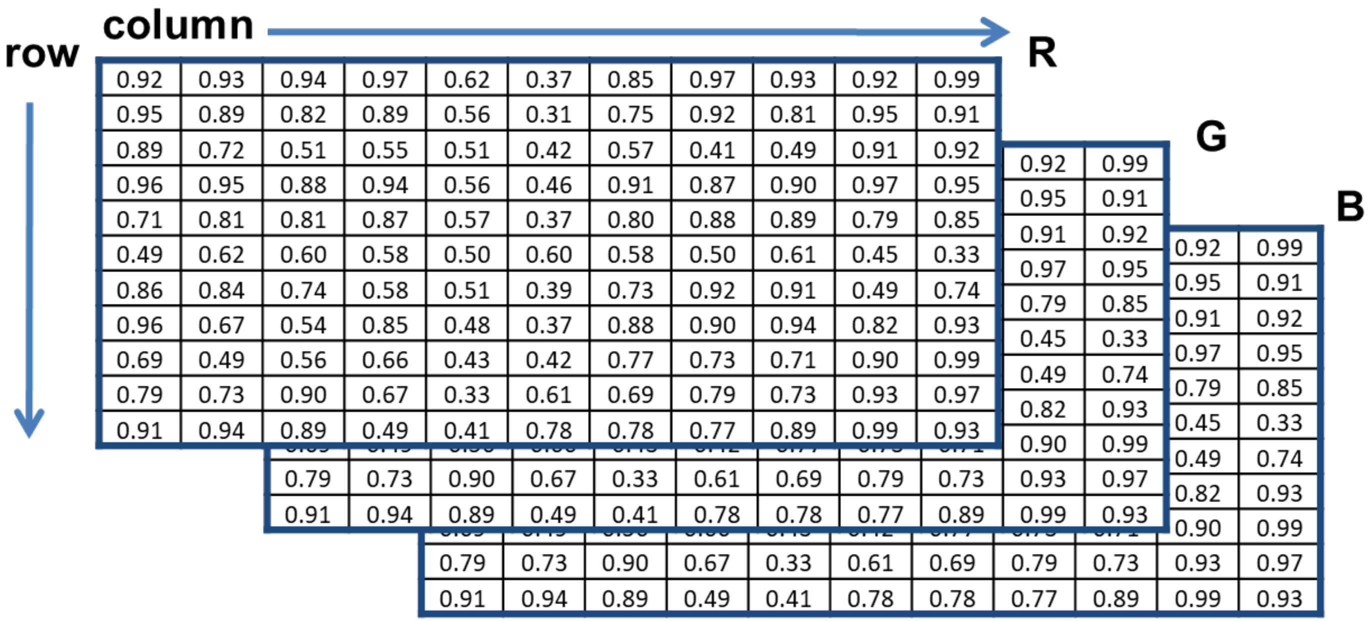
- Center and rescale images of digits so they are easier to compare to each other

- 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

Images can be 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

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

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]`

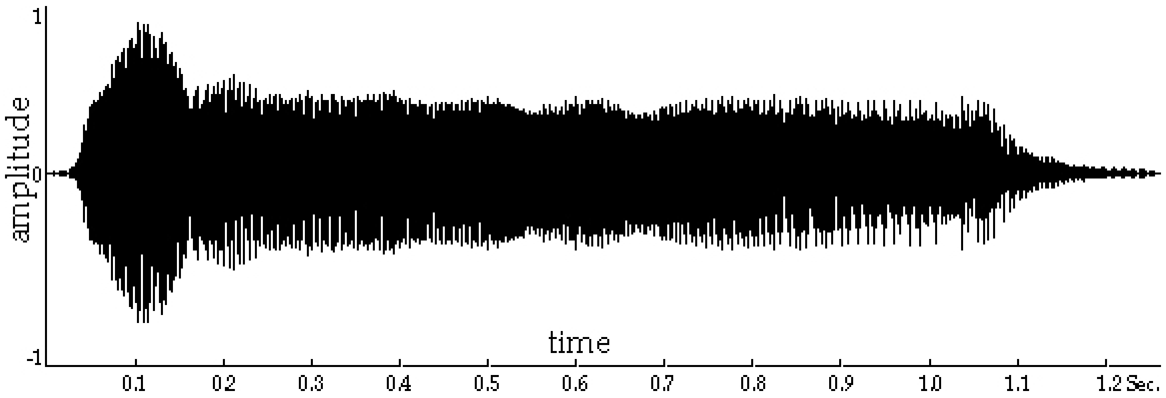
- Each complete word can map to an integer value, and we have a sequence of integers

  `“Dog ate” → [437 1256]`

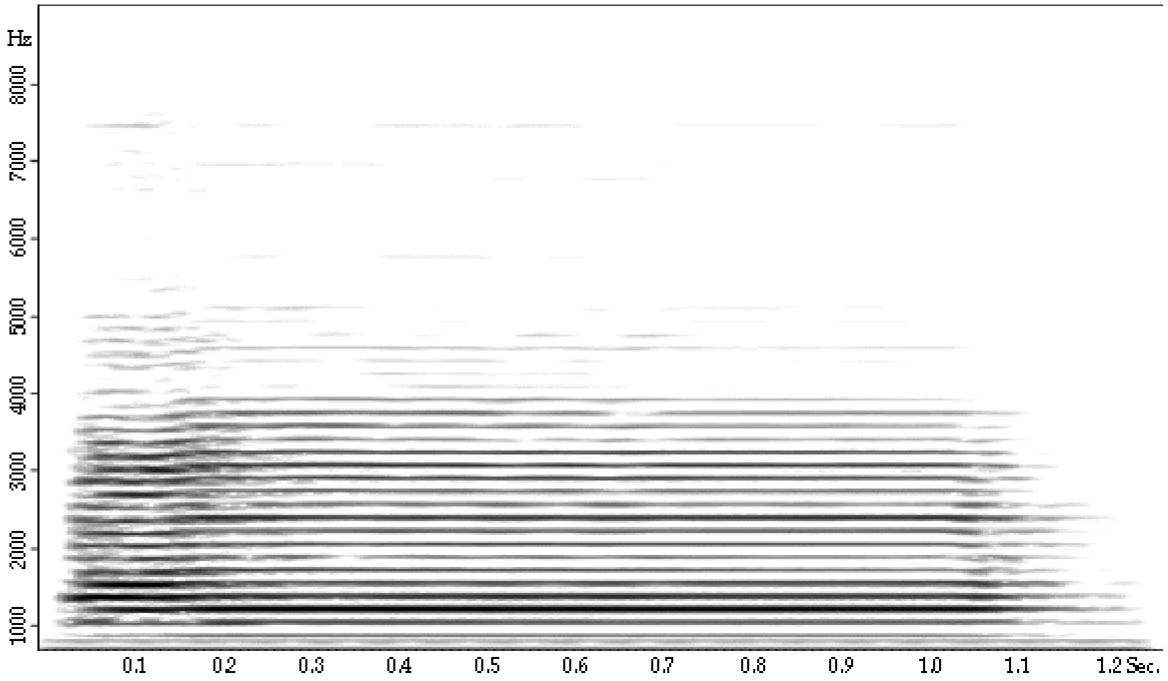
- Common groups of letters can be mapped to subwords and then to integers

  `“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 or one-hot vectors
  - Integer: Happy/Indifferent/Sad → 0/1/2
  - One-hot: Happy → [1 0 0]
  - Another example: Red/Green/Blue/Orange/Other → 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 can make code easier to program and read.

From data point to data set

$\boldsymbol{x} = \{x_0, \dots x_M\} \sim D$: 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
$\boldsymbol{X} = \{\boldsymbol{x_0}, \dots, \boldsymbol{x_N})$

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))
```

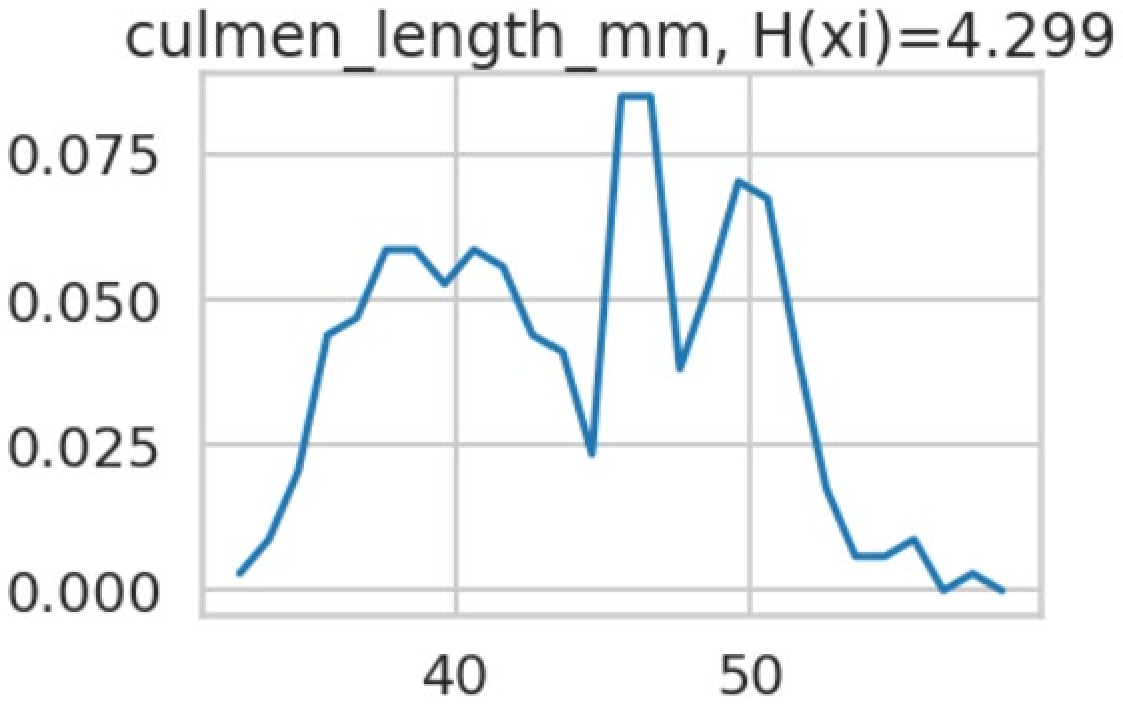
```
island_torgersen
[0. 1.]
P(xi=0)=0.850  P(xi=1)=0.150   H(xi)=0.609
```

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))
```

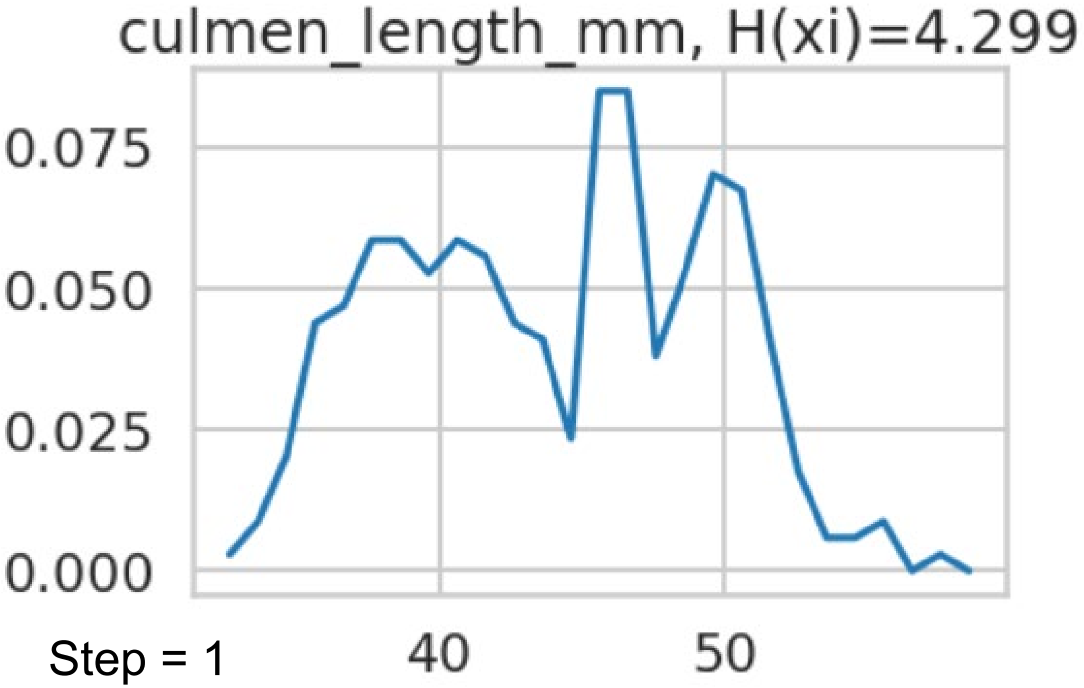


How do we measure $\boldsymbol{X}$?

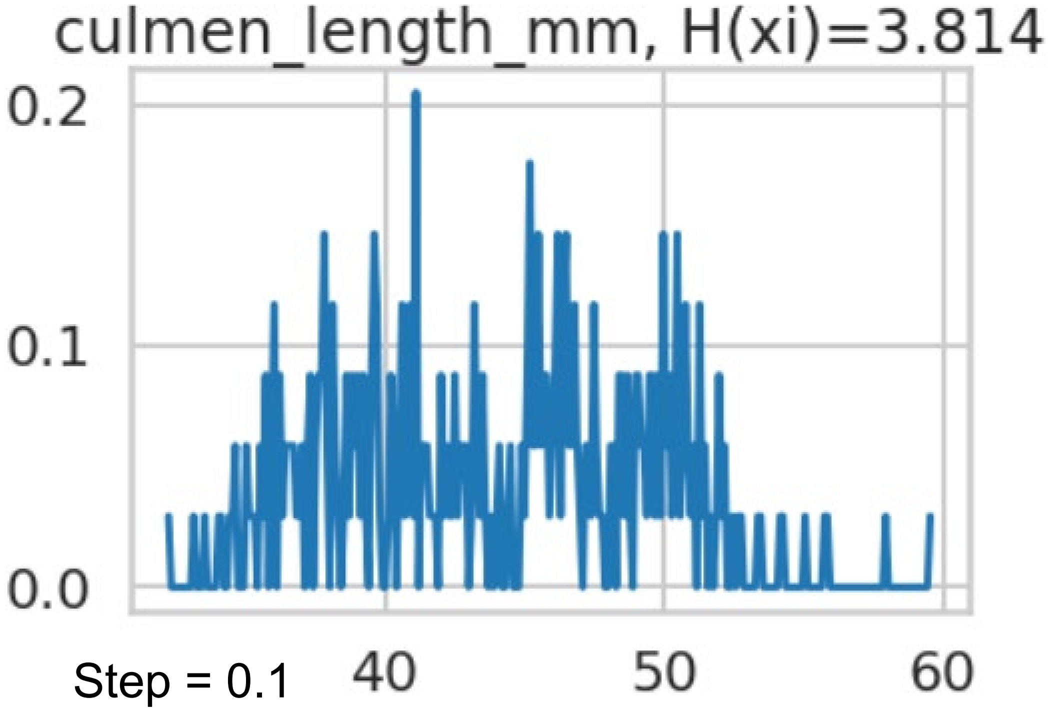
- We can measure the entropy of a particular variable:

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

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



Step = 1



Step = 0.1

3-minute Break

Which of these change the information contained in the data? *

- [ ] Lossless image compression
- [ ] Replacing text with a summary of the text
- [ ] Reshaping a vector or matrix
- [ ] Applying noise filtering to an audio stream
- [ ] Converting uint8 format data to uint16

What is the difference between data and information?

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

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

```
# 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

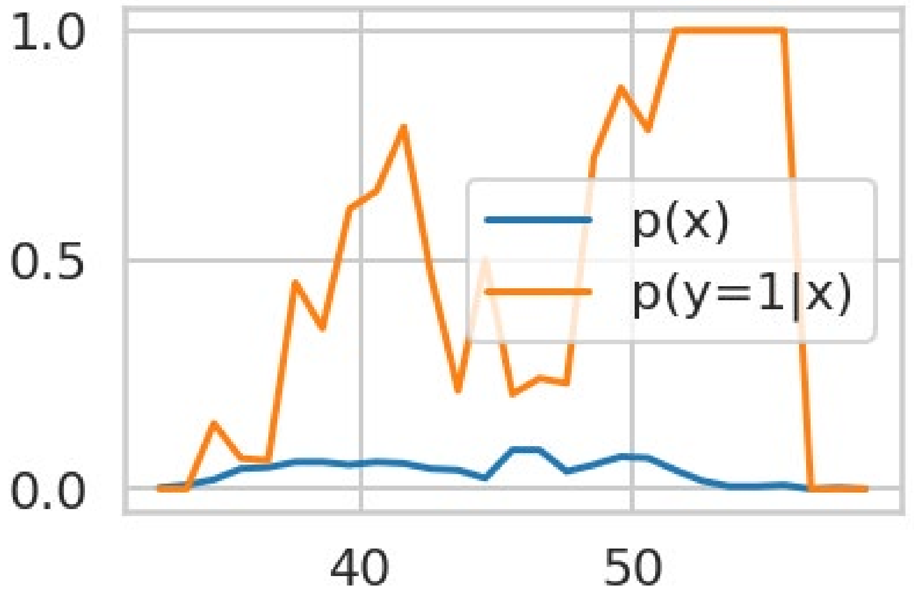
Information gain: IG(y|x) = H(y)-H(y|x)

- 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
```

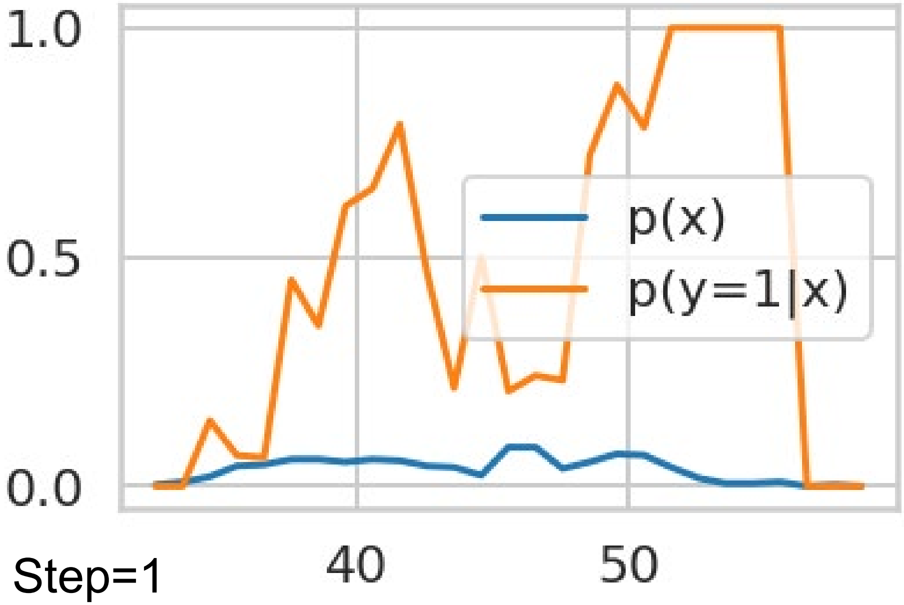


IG(y|x) = 0.304

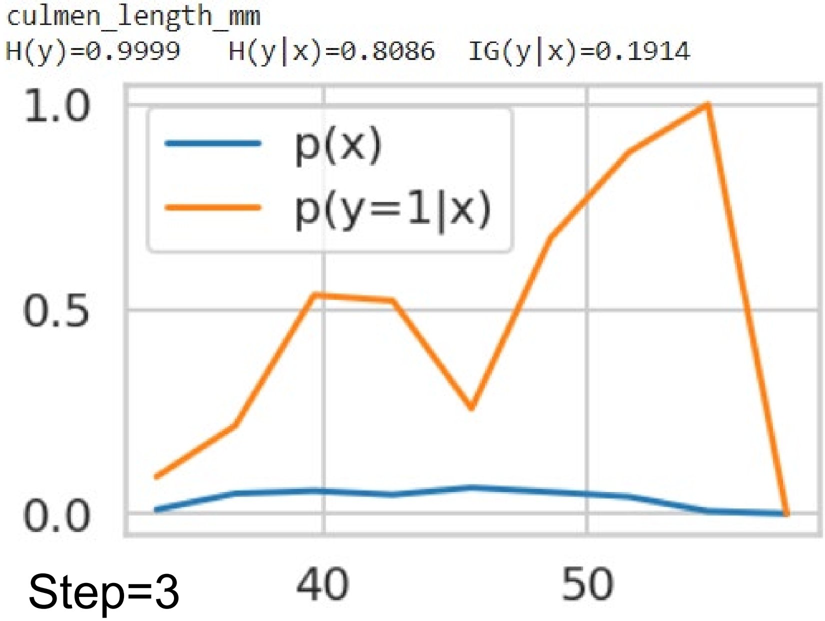
Knowing the culmen length tells us a lot whether a penguin is likely to be male or female. Large culmens are always male, but smaller ones could be male (maybe young) or female.

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

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



IG(y|x) = 0.304



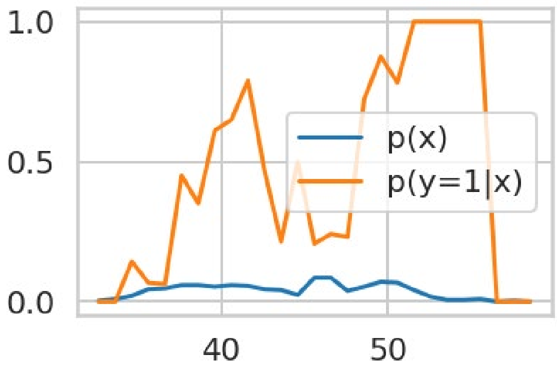
IG(y|x) = 0.1904

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
- The true probability distributions and information gain cannot be known. We can only try to make our best estimate, and it depends on our representation and model

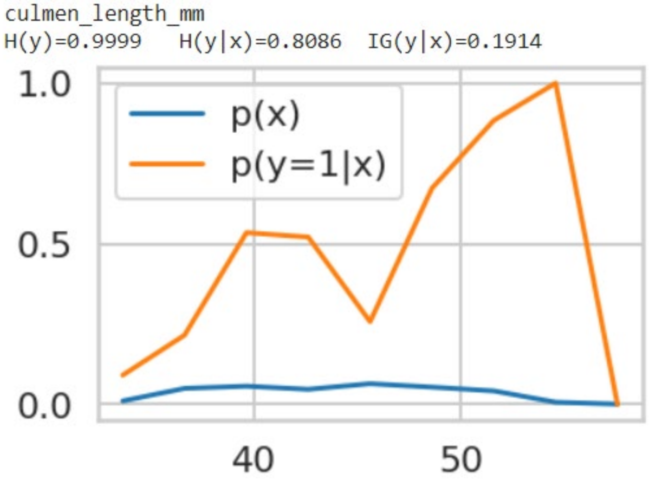
Step=1

IG(y|x) = 0.304



Step=3

IG(y|x) = 0.190



Introduction to MNIST, a classification benchmark

x

y 3 8 7 9 9 0 1 1 5 2

- 60,000 training samples
- 10,000 test samples
- Each sample has features $x \in R^{768}$, each value in the range of 0 to 1, and a label $y \in [0, 1, \ldots, 9]$

MNIST Processing

- x_train[0] is the features of the first training sample
- y_train[0] is the label of the first training sample
- x_train[:1000] is the features of the first 1000 training samples

```
# initialization code
import numpy as np
from keras.datasets import mnist
%matplotlib inline
from matplotlib import pyplot as plt
from scipy import stats


def load_mnist():
  '''
  Loads, reshapes, and normalizes the data
  '''
  (x_train, y_train), (x_test, y_test) = mnist.load_data() # loads MNIST data
  x_train = np.reshape(x_train, (len(x_train), 28*28))  # reformat to 768-d vectors
  x_test = np.reshape(x_test, (len(x_test), 28*28))
  maxval = x_train.max()
  x_train = x_train/maxval  # normalize values to range from 0 to 1
  x_test = x_test/maxval
  return (x_train, y_train), (x_test, y_test)

def display_mnist(x, subplot_rows=1, subplot_cols=1):
  '''
  Displays one or more examples in a row or a grid
  '''
  if subplot_rows>1 or subplot_cols>1:
    fig, ax = plt.subplots(subplot_rows, subplot_cols, figsize=(15,15))
    for i in np.arange(len(x)):
      ax[i].imshow(np.reshape(x[i], (28,28)), cmap='gray')
      ax[i].axis('off')
  else:
      plt.imshow(np.reshape(x, (28,28)), cmap='gray')
      plt.axis('off')
  plt.show()
```

An ML formulation

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

- The aim is to automatically find a model and its parameters that predicts y given X
- Under some losses, 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

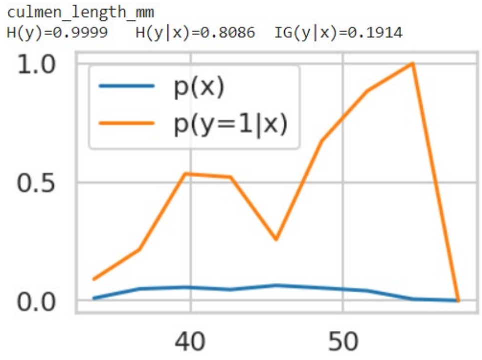
Things to remember

Machine learning is fitting parameters of a model so that you can accurately predict one set of numbers from another set of numbers

$$f(\boldsymbol{x};\theta) \rightarrow y$$

Something can take a lot of data storage but provide little information, or vice versa

The predictiveness or information gain of the features depends on how they are modeled



Next week

- Similarity, clustering, and retrieval
- K-NN classification and regression