Working with Data

Applied Machine Learning
Derek Hoiem
Machine learning model maps from features to prediction

\[
f(x) \rightarrow y
\]

**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

\[ f(x; \theta) \rightarrow y \]

Input features \quad \text{Model parameters} \quad \text{Predicted label}

Digit classification example

\begin{align*}
\mathbf{x} & \quad 3 & 6 & 7 & 9 & 9 & 0 & 1 & 1 & 5 & 2 \\
\mathbf{y} & \quad 3 & 8 & 7 & 9 & 9 & 0 & 1 & 1 & 5 & 2 
\end{align*}

Developing a classifier involves training and testing with training, validation, and test data
- \textit{Training} data: used to fit parameters of the model
- \textit{Validation} data: used to select the best model and any parameters that need to be manually set (“hyperparameters”)
- \textit{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
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)
Sometimes, we change the structure of data to make it easier to process. 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.

Image as matrix

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

Convenient for local pattern analysis

Image as vector

<table>
<thead>
<tr>
<th>0.92</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.95</td>
</tr>
<tr>
<td>0.89</td>
</tr>
<tr>
<td>0.96</td>
</tr>
<tr>
<td>0.71</td>
</tr>
<tr>
<td>0.49</td>
</tr>
<tr>
<td>0.86</td>
</tr>
<tr>
<td>0.96</td>
</tr>
<tr>
<td>0.69</td>
</tr>
<tr>
<td>0.79</td>
</tr>
<tr>
<td>0.91</td>
</tr>
<tr>
<td>0.93</td>
</tr>
<tr>
<td>0.89</td>
</tr>
<tr>
<td>0.72</td>
</tr>
<tr>
<td>0.95</td>
</tr>
<tr>
<td>0.81</td>
</tr>
<tr>
<td>0.62</td>
</tr>
<tr>
<td>0.84</td>
</tr>
<tr>
<td>0.67</td>
</tr>
<tr>
<td>0.49</td>
</tr>
<tr>
<td>0.73</td>
</tr>
<tr>
<td>0.94</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>0.93</td>
</tr>
</tbody>
</table>
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.
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 $\rightarrow$ 0/1/2
  - One-hot: Happy $\rightarrow$ [1 0 0]
  - Another example: Red/Green/Blue/Orange/Other $\rightarrow$ 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

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

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

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

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

<table>
<thead>
<tr>
<th>species</th>
<th>island</th>
<th>culmen_length_mm</th>
<th>culmen_depth_mm</th>
<th>flipper_length_mm</th>
<th>body_mass_g</th>
<th>sex</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Adelie</td>
<td>39.1</td>
<td>18.7</td>
<td>181</td>
<td>3750</td>
<td>MALE</td>
</tr>
<tr>
<td>1</td>
<td>Adelie</td>
<td>39.5</td>
<td>17.4</td>
<td>186</td>
<td>3800</td>
<td>FEMALE</td>
</tr>
<tr>
<td>2</td>
<td>Adelie</td>
<td>40.3</td>
<td>18.0</td>
<td>195</td>
<td>3250</td>
<td>FEMALE</td>
</tr>
<tr>
<td>3</td>
<td>Adelie</td>
<td>36.7</td>
<td>19.3</td>
<td>193</td>
<td>3450</td>
<td>FEMALE</td>
</tr>
<tr>
<td>4</td>
<td>Adelie</td>
<td>39.3</td>
<td>20.6</td>
<td>190</td>
<td>3650</td>
<td>MALE</td>
</tr>
<tr>
<td>5</td>
<td>Adelie</td>
<td>38.9</td>
<td>17.8</td>
<td>181</td>
<td>3625</td>
<td>FEMALE</td>
</tr>
<tr>
<td>6</td>
<td>Adelie</td>
<td>39.2</td>
<td>19.6</td>
<td>195</td>
<td>4675</td>
<td>MALE</td>
</tr>
<tr>
<td>7</td>
<td>Adelie</td>
<td>34.1</td>
<td>18.1</td>
<td>193</td>
<td>3475</td>
<td>Unknown</td>
</tr>
<tr>
<td>8</td>
<td>Adelie</td>
<td>42.0</td>
<td>20.2</td>
<td>190</td>
<td>4250</td>
<td>Unknown</td>
</tr>
<tr>
<td>9</td>
<td>Adelie</td>
<td>37.8</td>
<td>17.1</td>
<td>186</td>
<td>3300</td>
<td>Unknown</td>
</tr>
</tbody>
</table>
# 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, 6]==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 $X$?

- We can check the number of samples and dimensions
  
  ```python
  X.shape
  (341, 10)
  ```

- We can measure the distribution with statistics
  
  ```python
  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
  ```

  ```python
  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.

The estimates from larger sample sizes will vary less.
How do we measure $X$?

- We can measure the entropy of a particular variable:

$$H(x) = - \sum_k [P(x = k) \log P(x = k)] \text{ (if } x \text{ is discrete, i.e. finite number of possible values)}$$
How do we measure $X$?

- We can measure the entropy of a particular variable:

$$H(x) = - \int p(x) \log(p(x)) \, dx$$ (if $x$ is continuous)
How do we measure $X$?

- 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.
What is the difference between data and information?

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
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: $\text{IG}(y|x) = \text{H}(y) - \text{H}(y|x)$

- 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 $x$.

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

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
How can the information gain be different depending on 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.

\[
IG(y|x) = 0.304 \\
IG(y|x) = 0.190
\]
Introduction to MNIST, a classification benchmark

- 60,000 training samples
- 10,000 test samples
- Each sample has features $x \in \mathbb{R}^{768}$, 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

```python
# 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

\[ \theta^* = \arg\min_{\theta} \text{Loss}(f(X; \theta), y) \]

- 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

\[ \theta^* = \arg\min_{\theta} [H(y|x; \theta) - H(y) + R(\theta)] \]

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

- 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.

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