Data, and how to get it
ML_2022: Machine Learning

https://people.sc.fsu.edu/~jburkardt/classes/ml_2022/data_lecture/data_lecture.pdf

"Data! Data! Data!" he exclaimed. "I can't make bricks without clay!"
Sherlock Holmes, in "The Adventure of the Copper Beeches", by Sir Arthur Conan Doyle.

Data

Data is the "food" that machine learning consumes.

- *what is the "logical" format of data?*
- *what are common computer file formats for storing data?*
- *where can we find data?*
- *how can we read data from a file into our program?*
- *how can we summarize or characterize the data we have just read?*

.

1 The logical structure of data

A mathematician might think of data as simply a table $T_{i,j}$, where row `i` contains values associated with object `j`.
In machine learning, things can have a little more structure. To begin with, suppose I am a realtor observing the sale of houses. I might be interested in the living space, number of rooms, number of bedrooms, number of bathrooms, age, lot size, taxes, asking and selling price. My records of this information might look as follows:

```
2800  10  5  3   60  0.28   3167  142000  160000
1800   8  4  1   12  0.43   4033  175000  180000
1300   6  3  1   41  0.33   1471  129000  132000
```

Each row is associated with a particular house sale, or `item` or *instance* or `record`. Each column is reporting the value of a particular `feature`.

The data will be easier to share with others if I label the columns with a header line of `feature names`. This very useful feature complicates the structure, because our data now includes an initial line of text.

```
Living(SqFt)  #Rooms  #Beds  #Baths  Age(Yr)  Lot(Acre)   $Tax    $Ask   $Sell
        2800      10      5       3      60        0.28   3167  142000  160000
        1800       8      4       1      12        0.43   4033  175000  180000
        1300       6      3       1      41        0.33   1471  129000  132000
```

Especially if the data file is large, it may be important to identify each item with an index, identifier or name. The index might simply go 1, 2, 3... or there might be some jumps in the value. The important thing would be that each record would have a unique index. We may or may not include in the header line a new feature name for the index:

```
 ID  Living(SqFt)  #Rooms  #Beds  #Baths  Age(Yr)  Lot(Acre)   $Tax    $Ask   $Sell
155          2800      10      5       3      60        0.28   3167  142000  160000
158          1800       8      4       1      12        0.43   4033  175000  180000
176          1300       6      3       1      41        0.33   1471  129000  132000
```

or, perhaps, with no header for the index variable:

```
     Living(SqFt)  #Rooms  #Beds  #Baths  Age(Yr)  Lot(Acre)   $Tax    $Ask   $Sell
155          2800      10      5       3      60        0.28   3167  142000  160000
158          1800       8      4       1      12        0.43   4033  175000  180000
176          1300       6      3       1      41        0.33   1471  129000  132000
```

Thus, if we are interested in the house that sold for $180,000, we could ask for the data associated with record #2 (or #1 in Python!) or the record with `ID=158`.

Now obviously we want to put this kind of information on a computer, where it can be safely stored, retrieved, modified, or processed. To do so, we need to come up with an appropriate file format.

One possibility is to use a simple text format, that is, essentially to try to store the data exactly as it is printed above. A file containing such data is usually given a file extension of `.txt`, so our example here might be called `house_data.txt`. A program could easily read this data, if it is told to expect 10 numbers per line. For instance, MATLAB's `load()` function would read our data into an array as long as we omit the header line. Python has a similar `loadtxt()` function which reads text files, and can be instructed to skip one or more initial lines if necessary. Although it's not obvious in this example, we will soon encounter data in which some of the feature values are also text. In that case, our simple text format won't work.

A very common format for data, which can handle the presence or absence of header lines, index values, and even feature values that are text, is known as *Comma Separated Values* or `CSV` format. In a typical CSV file, a single comma separates successive feature values, and any text values are delimited by quotes. So, except possibly inside a text string, there will be no space characters in the file. Here's what our house price data might look like:

```
"Living Area (SqFt)","#Rooms","#Beds","#Baths","Age (Years)","Lot (Acres)","$Tax","$Asking","$Selling"
155,2800,10,5,3,60,0.28,3167,142000,160000
158,1800,8,4,1,12,0.43,4033,175000,180000
176,1300,6,3,1,41,0.33,1471,129000,132000
```

Of course, this format might seem just as tricky for a computer to read as the plain text format. However, there are clever CSV reader programs available in MATLAB (`csvread()`), Python (`csv.reader`), pandas (`read_csv()`), and R (`read.csv()`), so that all you have to do is name the file, call the function, and prepare to store the resulting data. Because a CSV file can contain text information and a few other odd items (such as NaN or NA), the output from the CSV reader may require a little careful treatment before it can be used. But it's all there for you.

Similarly, there are handy functions available to let you write data to a CSV file. We will get a chance to read and write data files in the next lab session.

2 Data varieties

Our housing price example involved simple numeric data, although even there, there were two types of data, real numbers for lot size, and integers for the others. Strictly speaking, price data is actually a special subset of the real numbers with just two decimal places expected.

But we may encounter other data sets with some new characteristics that require special handling.

- `logical` data, which might show up as `0/1`, `Yes/No`, `Y/N`, `T/F`, `True/False`;
- `categorical` data, which might have a specific range of numeric values (1, 2, or 3) or text values `'Married'`, `'Single'`;
- `text` data: 'Joe Friday', 'University of Phoenix', '77 Sunset Strip';
- `date` data: 7 January 2001, 28/2/2002, 2/28/2003, 2004/2/28, Mar 3 2003, April 13th 2004;
- `time` data: 1pm, 2 o'clock, 3:15, 4:25UTM, noon, midnight;
- `missing` data: (blank), ?, 'unknown', NMI, 'missing', nonsense values of -1 or 0;
- `exceptional` data: NA, NaN;
- `monetary` data: dollars $, euros €, yen ¥, yuan ¥, bitcoin Ƀ;
- `length` data: 5 ft 2 in, 6'2", 2 1/2 inches;
- `number and unit` data: 4 meters, 2 fluid ounces, 5.3 miles;

Another version of "text" data arises when our data is a substantial piece of writing, such as the *Gettysburg Address*, or *Lord of the Flies*, and we want to analyze the frequency of words, or the most likely word to follow a given word, or the typical length of a sentence. This is an interesting but advanced topic for a later discussion.

3 Characterizing your data

Especially when the data set is large or unfamiliar, it is useful to be able to produce some simple summary values that suggest the overall range and behavior of the data. In general, we need to apply these measures to a single, specific feature of the data:

- `mean`, the average value;
- `median`, half the data is above this value, half below ;
- `mode`, the most common value;
- `maximum, minimum`, highest and lowest;
- `RMS: root mean square`, square root of average of sum of squares;
- `variance and standard deviation`, RMS of difference between mean and values;
- `quantiles`, AKA percentiles AKA `quartiles`, counts the data over subintervals of the range ;

All of these measures are appropriate for numeric data, that is, for reals and integers. The mode is usually not appropriate for real values, or for widely scattered values, or names; however, for categorical and logical data, the mode is the only useful measurement As long as data can be ordered, (dates, alphabetic strings, times) then the mode, maximum, minimum, percentile and quartile characteristics can be applied.

Also, for any data that is ordered, it is possible to create a histogram, a kind of bar chart, in which the data range is split into subintervals, a count is made of how many items fall occur in each subinterval, and a plot is then made suggesting how the data varies.

Categorical and logical data can also be plotted using a bar chart. The difference is that with this data, each bar corresponds to a single category or True/False value, whereas with a histogram, each bar represents a range of values of the ordered data.

Conveniently, `numpy()` provides functions, such as np.mean(), to compute all the statistical measurements we have discussed, while `matplotlib()` provides the plt.hist() and plt.bar() commands for making histograms and bar plots. We will look at these computational tools during the lab session.

4 Normalizing or Standardizing

Generally, a structured dataset will have many columns, each one reporting the measurement of some specific feature. Let's assume for the moment that all measurements are returned as real numbers. We are familiar with the fact that some quantities, when measured, tend to be small (prices at the Dollar Store, distance in miles to the nearest Starbucks) or large (salaries of administrators, distance in miles between cities in the US).

When the scale for different data features varies too much, it can become difficult to plot the data correctly, to measure the variation in the values of each feature, or to decide whether a new item of data is "close" to the data we already have. There are two common methods for adjusting the data so that some of these scale problems are reduced.

For **normalization**, we use a linear transformation on the data to create new values that run from 0 to 1. Let us represent the original data as x. Compute $xmax$ and $xmin$, and create the transformed data values by

$$y = \frac{x - xmin}{xmax - xmin} \qquad \text{(Normalize)}$$

No x value is smaller than $xmin$, so $x - xmin$ is always at least 0. No x value is bigger than $xmax$, so $x - xmin$ is always no more than $xmax - xmin$. Therefore, y will always be between 0 and 1.

If we need to, we can convert a y value back to its original x scale by

$$x = xmin + (xmax - xmin) * y \qquad \text{(Denormalize)}$$

For **standardization**, we use a linear transformation on the data to create new values which have mean 0 and standard deviation 1. Starting with our data x of n values, we compute

$$\mu = \frac{\sum x_i}{n} \qquad \text{(Mean)}$$

and

$$\sigma^2 = \frac{\sum (x_i - \mu)^2}{n} \qquad \text{(Variance)}$$

and create the transformed data values by

$$y = \frac{x - \mu}{\sigma} \qquad \text{(Standardize)}$$

After standardization, the values are centered at 0, and some values may exceed 1 in size.

If we need to, we can convert a y value back to its original x scale by

$$x = \mu + \sigma * y \qquad \text{(Destandardize)}$$

In the next lab, we will practice both methods of rescaling data.

Note that, if you take a statistics class, you will find a slightly different formula for the variance of a set of data, as:

$$\sigma^2 = \frac{\sum (x_i - \mu)^2}{n - 1} \qquad \text{(Sample variance)}$$

The difference in the two definitions is tiny, the divisor of $(n - 1)$ is hard to remember, and is really of no importance to us!

5 Data Resources and Data Transfer

As we pick up new machine learning algorithms, we will want to test them out. To do this, we need data. We will expect the data to be in the structure format we have discussed, that is:

- a table of values;
- possibly stored as a txt or csv file;
- rows being records or cases;
- columns being values of a given feature;
- an optional first row giving names for the features;
- an optional first column assigning indexes to the records;
- feature values being real, integer, logical, categorical, or text;

Although machine learning algorithms expect to deal with large datasets of thousands or millions of records, we're just beginners, and we will be interested in "toy" datasets, or at least relatively small ones, typically with record counts not much more than 100. But we'd certainly be interested in sets of data that have some practical meaning, or come from some real-world application.

There a few places where we can easily browse for datasets, including

- **kaggle:** https://www.kaggle.com/ which offers code, competitions, data, and "micro courses" for machine learning. You have to create an account to use it;
- **keras:** https://keras.io/ a Python based application that provides a simple interface for using many machine learning algorithms, including neural networks; https://keras.io/api/datasets/ provides 8 datasets
- **scikit-learn**, https://scikit-learn.org/, documentation and program examples on the web, datasets available after installing the Python based software, and also at https://scikit-learn.org/stable/datasets.html including "toy datasets", "real world datasets" and "generated datasets";
- **UCI:**, https://archive.ics.uci.edu/ University of California at Irvine Machine Learning Repository, with 622 datasets;
- **Wikipedia:** lists the locations of hundreds of datasets for for image, text, sound, signals, physics, biology, anomalies, question data, finance, weather, census, transit, internet and games, https://en.wikipedia.org/wiki/List_of_datasets_for_machine-learning_research;
- **ml_2022**: https://people.sc.fsu.edu/~jburkardt/classes/ml_2022/datasets, our local collection of some simple, small datasets.

As an example, if we use our browser to get to the UCI site, one of the datasets that is listed is called "wine". Choosing that dataset, we see a page that includes the following information:

Data Set Characteristics:	Multivariate	**Number of Instances:**	178	**Area:**	Physical
Attribute Characteristics:	Integer, Real	**Number of Attributes:**	13	**Date Donated**	1991-07-01
Associated Tasks:	Classification	**Missing Values?**	No	**Number of Web Hits:**	1829263

which lets us know that this is a relatively small dataset containing real numbers and integers. On the same page, there is a link to an information page that describes the data in more detail, (it's a table measuring the concentrations of 13 chemicals in 178 wine samples) and a link that lets us download the data.

The first five lines of this file are:

```
1,14.23,1.71,2.43,15.6,127,2.8,3.06,.28,2.29,5.64,1.04,3.92,1065
1,13.2,1.78,2.14,11.2,100,2.65,2.76,.26,1.28,4.38,1.05,3.4,1050
1,13.16,2.36,2.67,18.6,101,2.8,3.24,.3,2.81,5.68,1.03,3.17,1185
1,14.37,1.95,2.5,16.8,113,3.85,3.49,.24,2.18,7.8,.86,3.45,1480
1,13.24,2.59,2.87,21,118,2.8,2.69,.39,1.82,4.32,1.04,2.93,735
```

which is a little more readable if we line things up:

```
    1   2     3     4      5      6  7     8     9   10    11    12    13     14
(1) 1, 14.23, 1.71, 2.43, 15.6, 127, 2.8,  3.06, .28, 2.29, 5.64, 1.04, 3.92, 1065
(2) 1, 13.2,  1.78, 2.14, 11.2, 100, 2.65, 2.76, .26, 1.28, 4.38, 1.05, 3.4,  1050
(3) 1, 13.16, 2.36, 2.67, 18.6, 101, 2.8,  3.24, .3,  2.81, 5.68, 1.03, 3.17, 1185
(4) 1, 14.37, 1.95, 2.5,  16.8, 113, 3.85, 3.49, .24, 2.18, 7.8,   .86, 3.45, 1480
(5) 1, 13.24, 2.59, 2.87, 21,   118, 2.8,  2.69, .39, 1.82, 4.32, 1.04, 2.93,  735
```

at which point we realize several things:

- there is no header;
- there are no indexes;
- they "lied" to us, there are 14 items in each row;
- the features in each column have very different ranges;

And after looking over the data some more, we realize a few more facts:

- We need to read the data write up and found out what the extra feature is (it's the first item, which identifies the vineyard);
- the file format is clearly csv, so we will be able to call a built-in data reader to extract the data;
- we may want to normalize or standardize the data, which will be simple since it's all numeric;

In the next lab, we will practice retrieving datasets, peeking inside to get an initial look, reading them into a Python program, applying normalization or standardization, and computing the statistical characteristics we have talked about.