The nature of data unit:

- Objects and vectors (**today**)
- K nearest neighbors classification (Friday)
- (Start linear regression next week)

Today:

- Follow-up: Libraries
- Ways to think about data
- Variable naming conventions
- Meet the datasets
- Feature selection
- The curse of dimensionality
- Term project (and other assignments)

- I want to follow up on a few things about the recent lab & demo.

15

- lab: numpy - mastering this would take some effort & it's not n
  just note slicing, fancy & boolean indexing, vectorized
  matplotlib - by the end of the semester, use this in your final prese
  pandas - data sets etc. - not so critical to us but sometimes usefu
  Scikit learn - ~~Nice[illegible] data sets built-in~~
  - has some data sets like iris
  - classifiers
  - etc.

2

The nature of data

- But today I want to lay the ground work for thinking about data

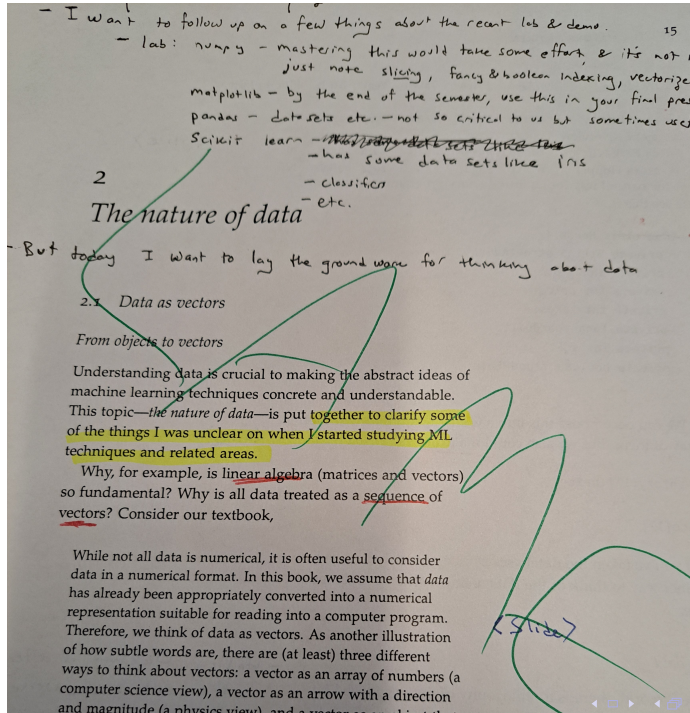
2.1 Data as vectors

From objects to vectors

Understanding data is crucial to making the abstract ideas of machine learning techniques concrete and understandable. This topic—*the nature of data*—is put together to clarify some of the things I was unclear on when I started studying ML techniques and related areas.

Why, for example, is linear algebra (matrices and vectors) so fundamental? Why is all data treated as a sequence of vectors? Consider our textbook,

> While not all data is numerical, it is often useful to consider data in a numerical format. In this book, we assume that *data* has already been appropriately converted into a numerical representation suitable for reading into a computer program. Therefore, we think of data as vectors. As another illustration of how subtle words are, there are (at least) three different ways to think about vectors: a vector as an array of numbers (a computer science view), a vector as an arrow with a direction and magnitude (a physics view), and a vector as [illegible]



Why Python library would you use for each purpose?

Efficient arrays. `numpy`

Data visualization. `pandas`

Managing large datasets. `matplotlib`

Machine learning tools. Scikit-learn, `sklearn`

Address	ZIP	Price	Taxes	Year	Sq ft	Flood plain
214 Hickory	27148	$417K	$12K	1987	2120	N
17 W Riverview	30921	$295K	$7K	1927	3015	Y
55 Park NE	81066	$528K	$22K	2012	3525	N
2s912 Hopper	67811	$238K	$4K	1971	1800	N

```
public class House {
    private String streetAddr;
    private int zip;
    private int price;
    private int taxes;
    private long yearBuilt;
    private int sqFt;
    private boolean floodPlain;
}

House[] inventory = new House[100];
```

Address	ZIP	Price	Taxes	Year	Sq ft	Flood plain
214 Hickory	27148	$417K	$12K	1987	2120	N
17 W Riverview	30921	$295K	$7K	1927	3015	Y
55 Park NE	81066	$528K	$22K	2012	3525	N
2s912 Hopper	67811	$238K	$4K	1971	1800	N

```
struct house {
    char* street_addr;
    int zip;
    int price;
    int taxes;
    time_t year_built;
    int sq_ft;
    int flood_plain;
};

struct house inventory[100];
```

Address	ZIP	Price	Taxes	Year	Sq ft	Flood plain
214 Hickory	27148	$417K	$12K	1987	2120	N
17 W Riverview	30921	$295	$7K	1927	3015	Y
55 Park NE	81066	$528K	$22K	2012	3525	N
2s912 Hopper	67811	$238	$4k	1971	1800	N

```
char* street_addr[100];
int zip[100];
int price[100];
int taxes[100];
time_t year_built[100];
int sq_ft[100];
int flood_plain[100];
```

From Hadley Wickham, "Tidy Data", *Journal of Statistical Software*, Aug 2014, 59:10.

> *A dataset is a collection of values, usually either numbers (if quantitative) or strings (if qualitative). Values are organized in two ways. Every value belongs to a variable and an observation. A variable contains all values that measure the same underlying attribute (like height, temperature, duration) across units. An observation contains all values measured on the same unit (like a person, a day, or a race) across attributes.* *Wickham, pg 3*

> *Tidy data is a standard way of mapping the meaning of a dataset to its structure. A dataset is messy or tidy depending on how rows, columns, and tables are matched up with observations, variables, and types. In tidy data,*
>
> 1. *Each variable forms a column.*
> 2. *Each observation forms a row.*
> 3. *Each type of observational unit forms a table.*
>
> *Wickham, pg 4*

While not all data is numerical, it is often useful to consider data in a numerical format. [It is common to] assume that data has already been appropriately converted into a numerical representation suitable for reading into a computer program. Therefore, we think of data as vectors. As another illustration of how subtle words are, there are (at least) three different ways to think about vectors: a vector as an array of numbers (a computer science view), a vector as an arrow with a direction and magnitude (a physics view), and a vector as an object that obeys addition and scaling (a mathematical view).

Deisenroth et al, Mathematics for Machine Learning, pg 4

- N is the number of observations
- D is the number of variables/attributes, or *dimensionality* of the data
- $\mathbf{X}$ is the $N \times D$ data set as a matrix
- $\vec{y}$ is the N-length vector of target values
- n is used as an index into the data set, for example $\mathbf{X}_n$ and $\vec{y}_n$
- i and j are used as indices into features, for example $\vec{x}_i$ where $\vec{x} = \mathbf{X}_n$

"Hello, Data."

Many machine learning problems become exceedingly difficult when the number of dimension in the data is high. This phenomenon is known as the **curse of dimensionality**. *Of particular concern is that the number of possible distinct configurations of a set of variables increases exponentially as the number of variables increases.* *Goodfellow et al, Deep Learning, pg 151*

The origin of the problem [the curse of dimensionality] is illustrated in that if we divide a region of a space into regular cells, then the number of such cells grows exponentially with the dimensionality of the space. The problem with an exponentially large number of cells is that we would need an exponentially large quantity of training data in order to ensure that the cells are not empty.
Bishop, Pattern Recognition and Machine Learning, pg 35.

We are so used to living in three dimensions that our intuition fails us when we try to imagine a high-dimensional space. Even a basic 4D hypercube is incredibly hard to picture in our mind, let alone a 200-dimensional ellipsoid bent in a 1000-dimensional space.

It turns out that many things behave very differently in high-dimensional space. For example, if you pick a random point in a unit square, it will have only about a 0.4% *chance of being located less than 0.001 from a border (in other words, it is very unlikely that a random point will be "extreme" along any dimension.) But in a 10000-dimensional unit hypercube, this probability is greater than* 99.999999%. *Most points in a high-dimensional hypercube are very close to the border.*

In theory, one solution to the curse of dimensionality could be to increase the size of the training set to reach a sufficient density of training instances. Unfortunately, in practice, the number of training instances required to reach a given density grows exponentially with the number of dimensions. With just 100 features, you would need more training instances than atoms in the observable universe in order for the training instances to be within 0.1 of each other on average, assuming they were spread out uniformly across all dimensions.

Geron, Hand-On Machine Learning, pg 208–209

Coming up:

Due today, Wed, Jan 22:
Read Hadley Wickam, "Tidy Data", Sections 1-3.
(Find pdf on Canvas.)

Due Thurs, Jan 23:
Take "objects and vectors" quiz

Due Fri, Jan 24:
Do numpy programming assignment

Due Wed, Jan 29:
Read and respond to article about the Boston Housing Dataset
(Find pdf on Canvas.)

Due Mon, Feb 3:
Propose project topic