CS 615 – Deep Learning

ML Basics

Objectives

- Problems in ML
- Data Types
- Supervised Learning
  - Datasets
  - Cross Validation
  - Over/Under-fitting

Ok, back to Machine Learning

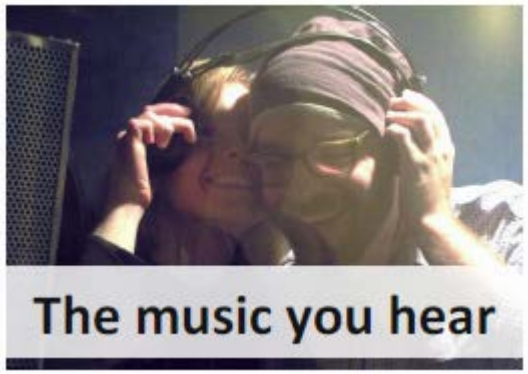
And Deep Learning

What is Machine Learning?

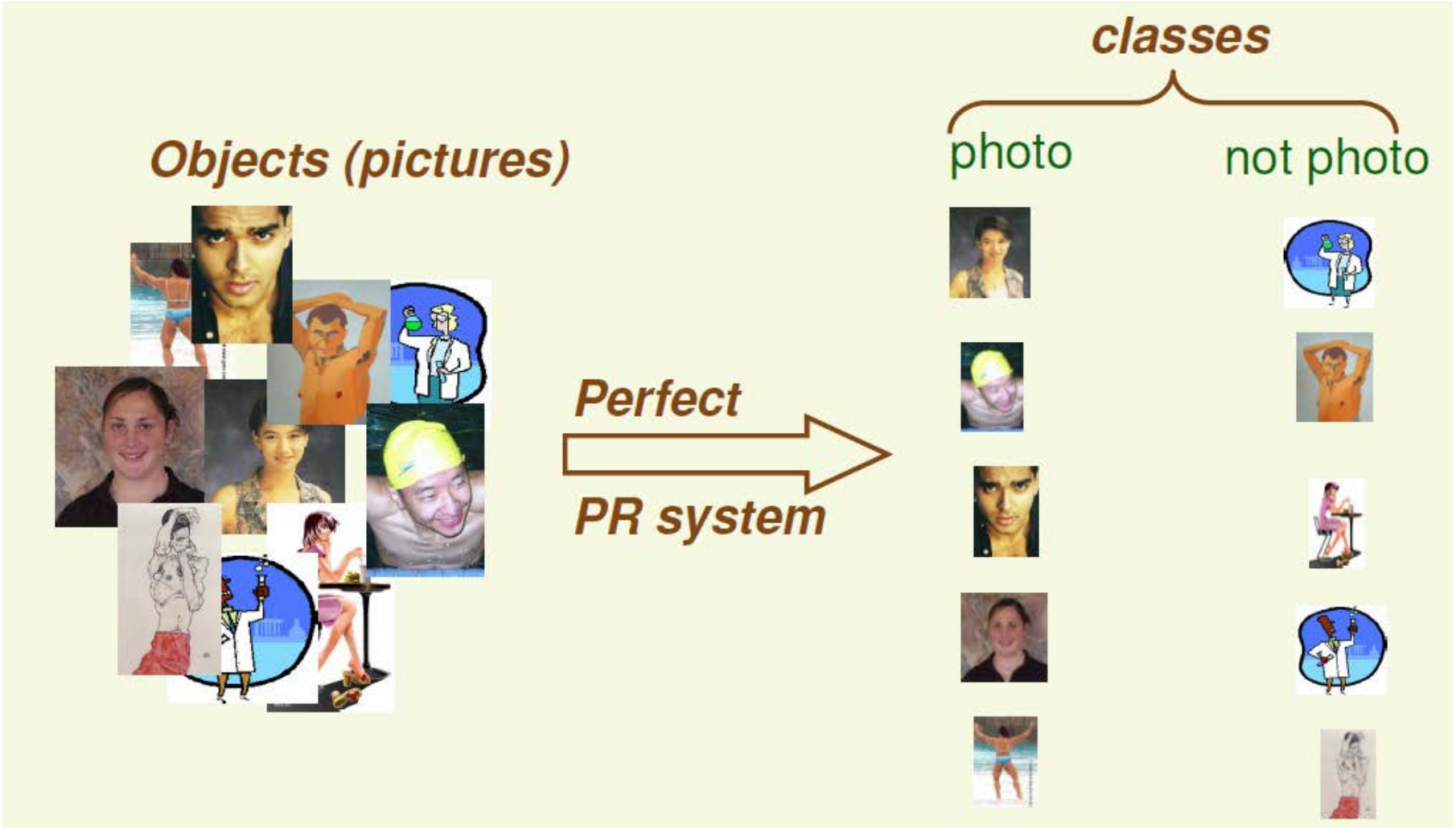
- Definition: "The study of computer algorithms that improve automatically through experience"
- Formally:
  - Improve at task T
  - With respect to performance measure P
  - Based on experience E
- Example: Recognize a Person
  - T: recognize a person
  - P: number of time we recognized a person correctly
  - E: a database of labeled faces

Why do we care?

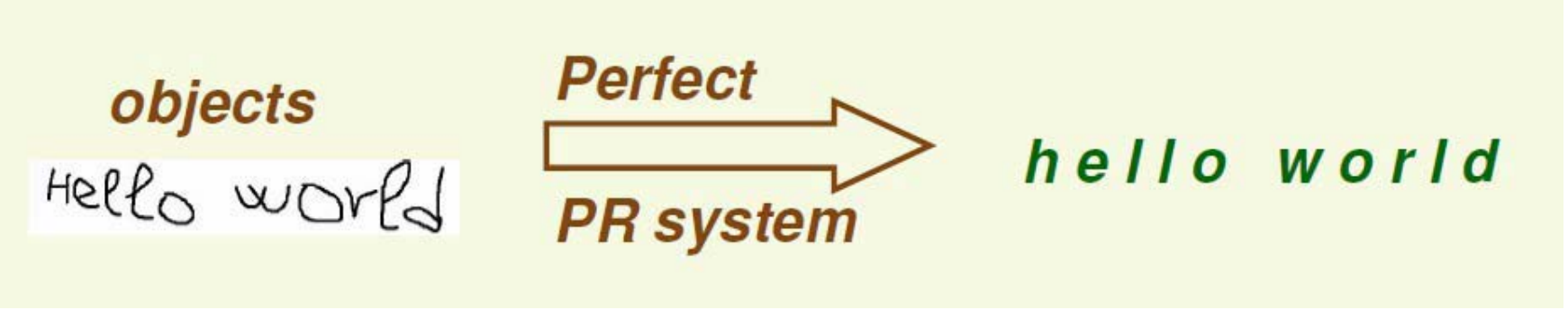
- It’s everywhere!!!



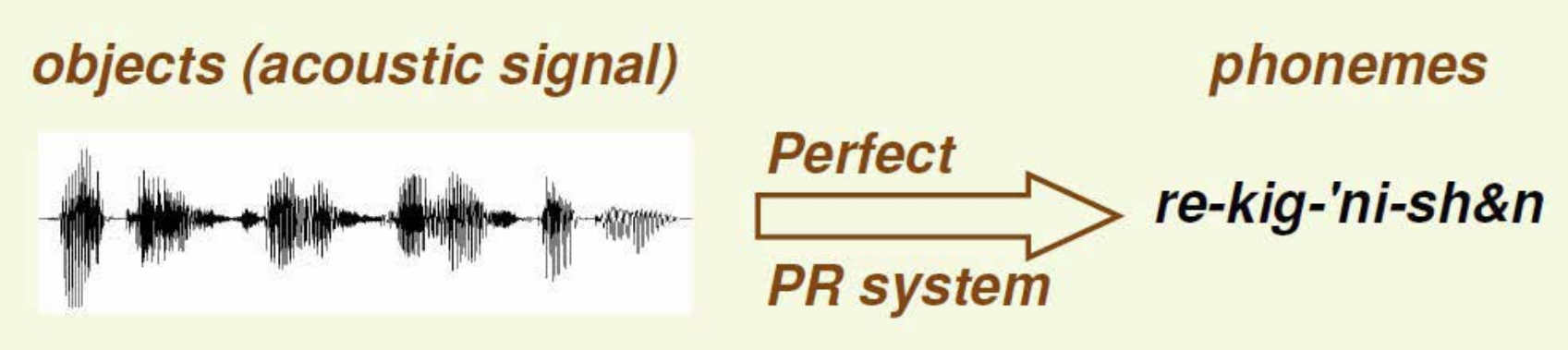
Example: Photograph or Not?



Example: Character Recognition



Example: Speech Understanding



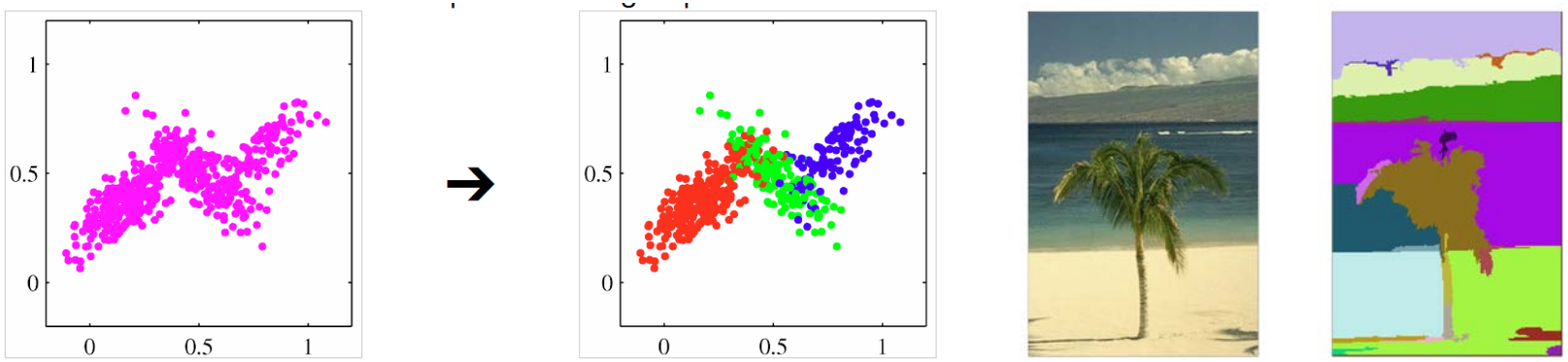
How Do We Do It?

- We need to think about
  1. What needs to be learned?
     - What's our task/goal?
  2. What feedback can we get and in what form?
     - Supervised learning (correct answers for each example)
     - Unsupervised learning (correct answers not given)
  3. What representation should we use (features)?

Problems in ML

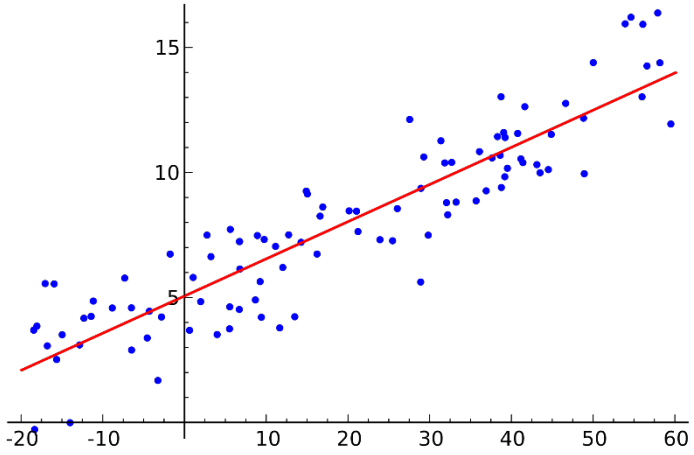
- There are several typically machine learning problems
- But they can generally be broken down into three categories:

1. Clustering
   - Given some data we want to figure out how to group them together.
   - This is an example of *unsupervised learning* where we're trying to discover the groups (instead of being given them)



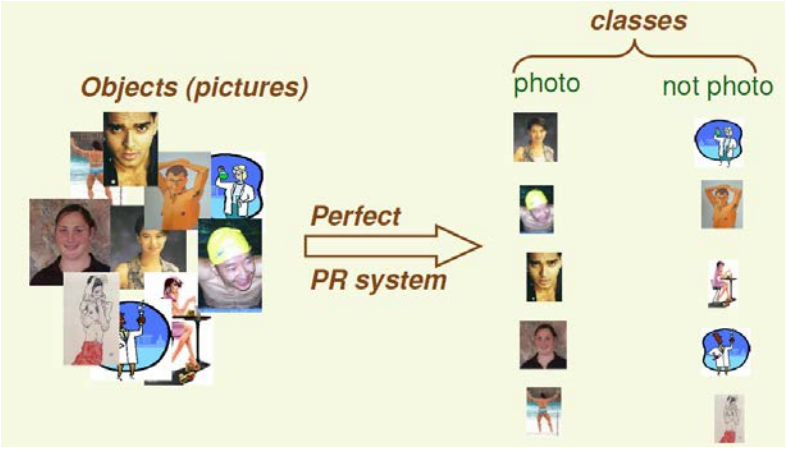
Problems in ML

2. Regression
   - Given some data, can we predict an outcome value?
   - Example: We have a car's brand, year, mpgs and want to figure out its worth
   - This is an example of *supervised* learning
     - To build our prediction system, we have data with labels.

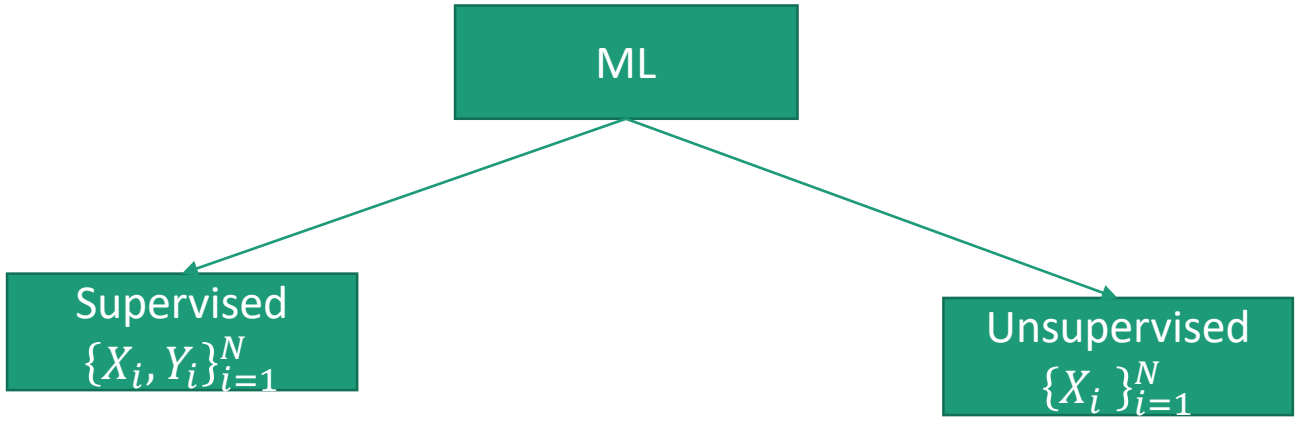


Problems in ML

3. Classification
   - Given data, can we predict which category something belongs to.
   - Typically involves *learning* some rules.
   - This is an example of *supervised* learning
     - To build our prediction system, we have data with labels.

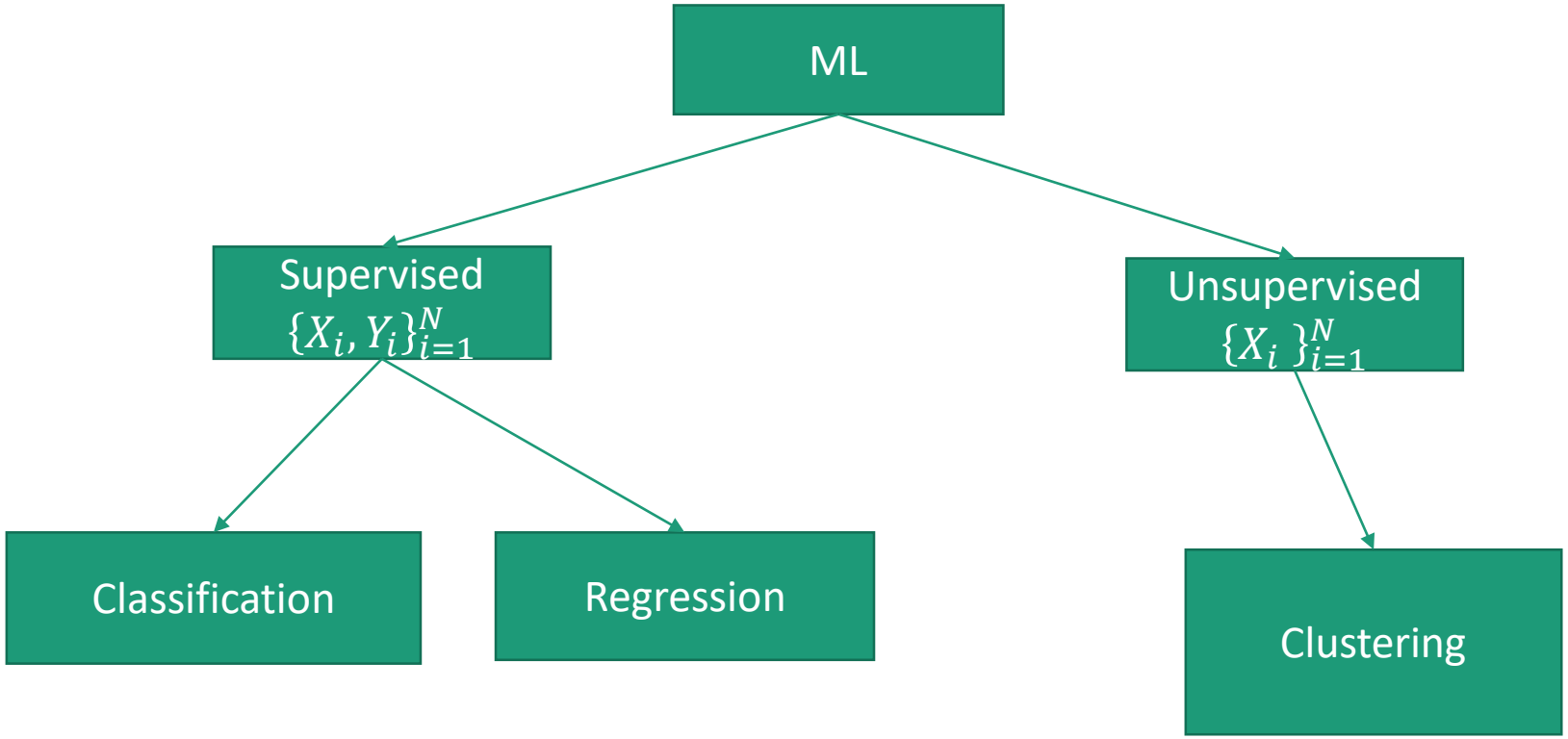


ML Overview

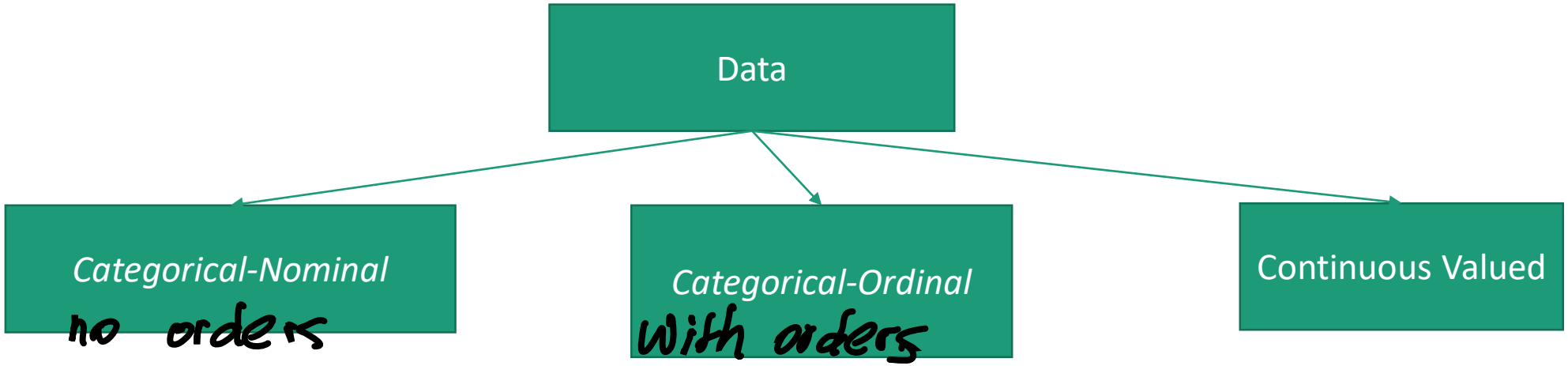


- We can basically break machine learning tasks into two categories
  1. Supervised Learning
  2. Unsupervised Learning
- *Supervised learning*
  - Data X_i and correct answer (label) Y_i given for each example $i \in \{1, \ldots, N\}$
- *Unsupervised learning*
  - Only data given for each example, X_i

Types of Problems



Types of Data



- Typically we have some information for each of our observations, which we call **feature vectors, or descriptors.**
- Each piece of information pertaining to a feature vector can fall into one of three categories:
  - *Categorical-Nominal (unordered)*
    - Examples: Car Model, School
  - *Categorical-Ordinal (can be ordered)*
    - Examples: Colors, small < medium < large
  - *Continuous Valued*
    - Examples: Blood Pressure, Height

Feature Types

- Let's assume that we have our data in our system.
- The next thing to consider is the nature of our data.
- As previously mentioned, we categorize each feature as one of three types:
  1. Continuous valued
  2. Categorical-Nominal
  3. Categorical-Ordinal
- Depending on the algorithm being used we may need to
  - Convert categorical features into a set of binary features
  - "Standardize" features so that they have the same range

Prepping Data

- Categorical-Normal → Binary
  - If our algorithm computes some floating value/location of a feature then it wouldn't make sense to have a floating point value of a categorical feature
    - Imagine if a feature is car make $\mathrm{x}_1 \in \{Honda, Ford, Toyota\} \rightarrow \{0,1,2\}$
    - What would the meaning of 0.7 be?
  - Instead we'll want to create a set of binary features from our enumerations
    - $isHonda \in [0,1], isFord \in [0,1], isToyota \in [0,1]$
- Standardizing Features
  - If our algorithm incrementally updates parameters, then we may want all features to have the same variance.
  - Otherwise ones with greater range will have greater influence than ones with a smaller range

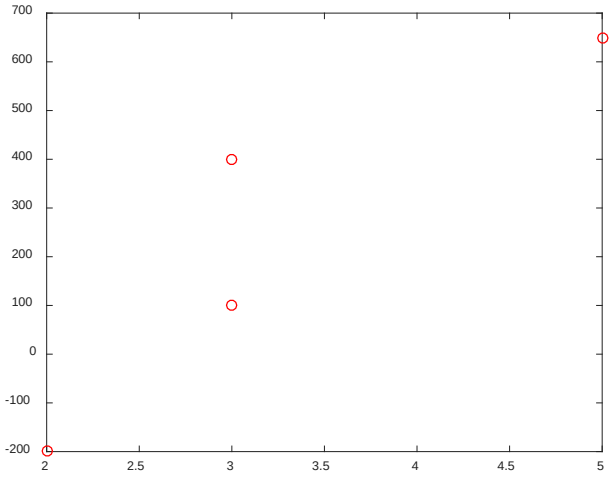
Standardizing Data

- Example: Imagine two features for a person
  - Height
  - Weight
- These are both on very different scales
  - Height (inches): Maybe 12 → 80
  - Weight (lbs): Maybe 5→400
- If we used the data as-is, then one feature may have more influence than the other
  - Depends on the algorithm

Standardizing Data

- Standardized data has
  - Zero mean
  - Unit deviation
- We treat each feature independently and
  1. Center it (subtract the mean from all samples)
  2. Make them all have the same span (divide by standard deviation).
- Example

```
X = [3, 400;
     2, -200;
     3, 100;
     5, 650];
figure(1);
plot(X(:,1),X(:,2),'or');
```

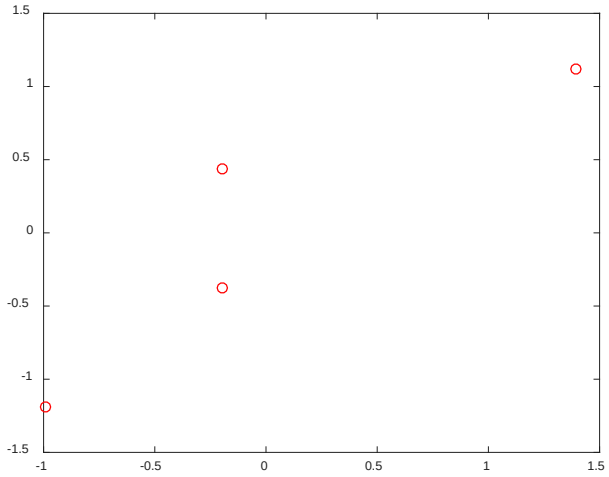


Standardizing Data

- We can compute the mean and standard deviation of each feature easily and then subtract the means from each observation and divide each (centered) observation by the standard deviation of each feature.
- Would it make sense to do this with categorical (nominal and/or ordinal) data?
- How about if it was made into a binary feature set?

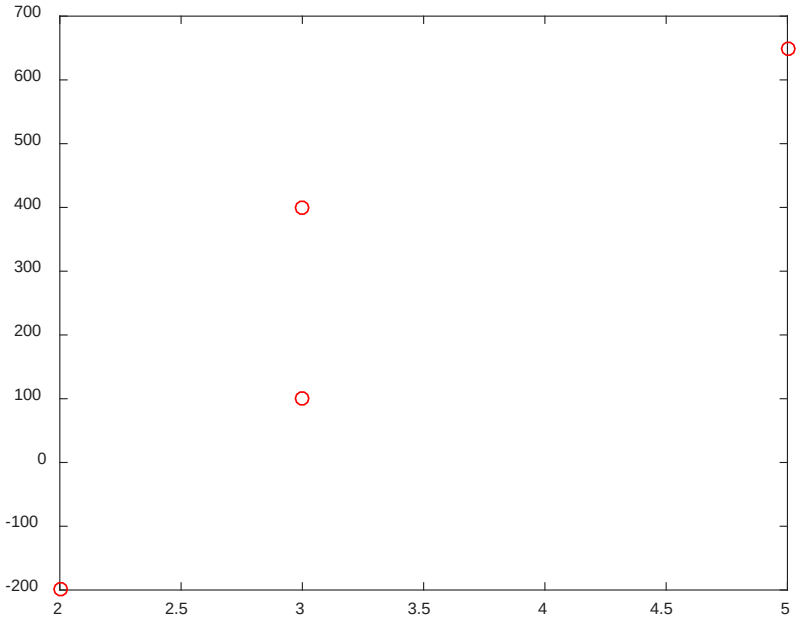
```
m = mean(X);  %get mean of each feature
s = std(X);   %get std of each feature
%m = 3.2500  237.5000
%s = 1.2583  368.2730

X = X - repmat(m,size(X,1),1);
X = X./repmat(s,size(X,1),1);
figure(2);
plot(X(:,1),X(:,2),'or');
```

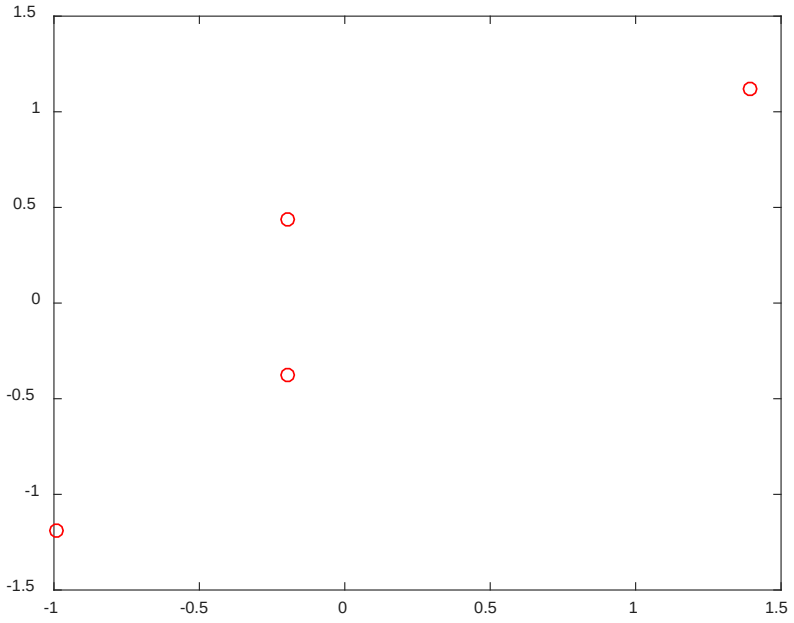


Standardizing Data

Original



Standardized



Note on Standard Deviation

- There are two commonly used, slightly different computations for standard deviation:

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}$$

known mean
Or true mean

$$\sigma = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu)^2}$$

calculated from the sample

- Typically the first version is to be used if we got the mean, μ, somehow other than computing it from the data itself.
- The second version adjusts for this lack of independence of σ and , u if in fact μ was computed as $\mu = \frac{1}{N}\sum_{i=1}^{N} x_i$
- For this course, we will typically want to use the version that divides by $N - 1$

Note: Python's std defaults to using N instead of $N - 1$!

Supervised Learning

- With regression we're trying to build a system in order to predict a (continuous) value given some data
- These systems are built using prior *labeled* data
  - That is, for each data sample X_t there is an associated value Y_t
  - Together this gives us our dataset: $\{X_t, Y_t\}_{t=1}^{N}$
- Since we have labeled data, this is our first example of *supervised learning.*
  - And there's several things/caveats about supervised learning that we must first discuss

Data Sets

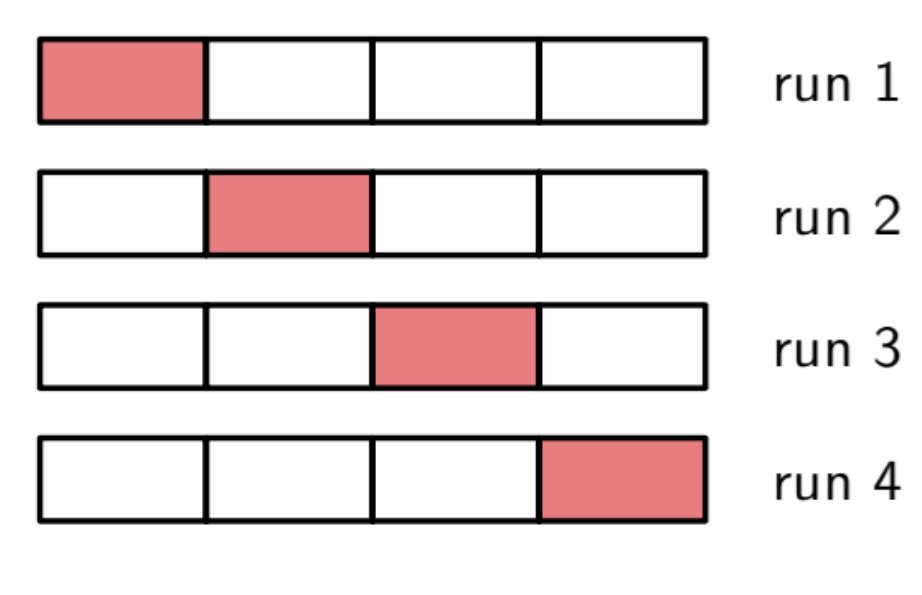
- For constructing our final model/system we will want to use all of our data
- However, often we need to figure out which model is best
- Therefore we will split our data into two groups:
  1. **Training Data**
  2. **Testing Data**
- Typically this is done as a 2/3 training, 1/3 testing split
- We then build/train our system using the training data and test our system using the testing data
- Now we can compare this system against others!
- Once we've chosen our model then we can use all the data and know what the upper-bound on the error should be
- The key is to have the training data and testing data pulled from the same distribution

Cross Validation

- What if we don't have that much data?
- Then we can do something called *cross-validation*
- Here we do several training(and validation, if necessary)/testing runs
  - Keeping track of all the errors
- We can then compute statistics for our classifier based on the list of errors.
- Again, in the end our final system will be built using all the data.

S-Folds Cross Validation

- There are a few types of cross-validation
  - S-Folds: Here we'll divide our data up into S parts, train on $S-1$ of them and test the remaining part. Do this S times
  - Leave-one-out: If our data set is really small we may want to built our system on $N-1$ samples and test on just one sample. And do this N times (so it's basically N-folds)



Learning Function

- In general with supervised learning, with the *absence* of *noise* and with complete data in Z, we can say there is some function $f(z)$ such that

$$y = f(z)$$

- However, in reality, we observe a limited set of features and data
  - And some of it can be noisy

$$X \subset Z + \epsilon$$

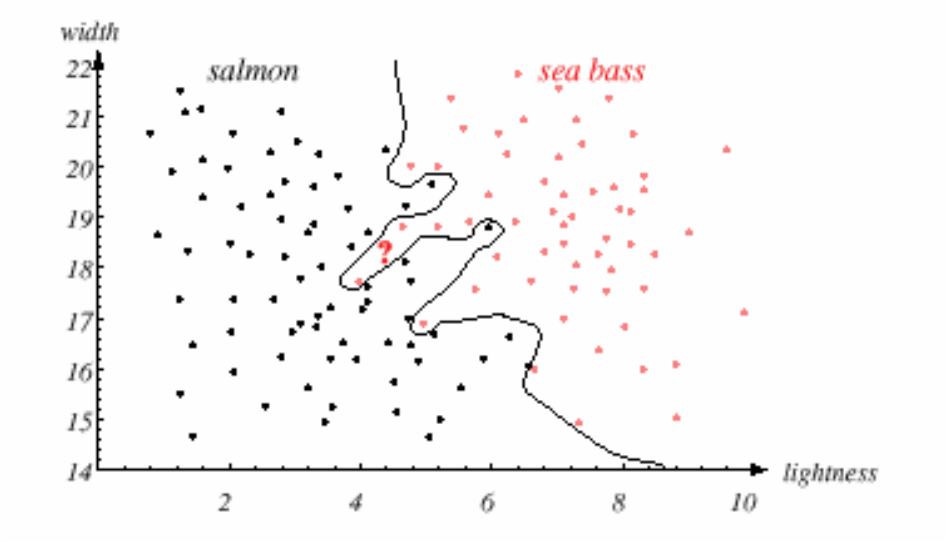
- So we want to do is to learn a function $g(x, \theta)$ that is an approximation of the true underlying $f(z)$

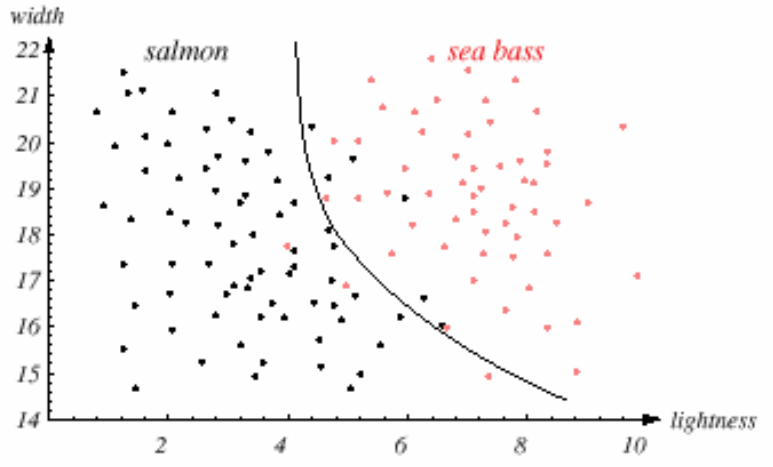
$$g(x, \theta) \approx f(z)$$

  - Where θ are parameters of our model

Over/Underfitting

- The biggest problem with supervised learning is over or under-fitting
- If we over-fit our training data, then we're finding a function for the training data, not the function of the entire system.
  - Therefore may not fit the general data
- If we under-fit our data then our model $g(x, \theta)$ is not specific enough to be an approximation to $f(z)$.

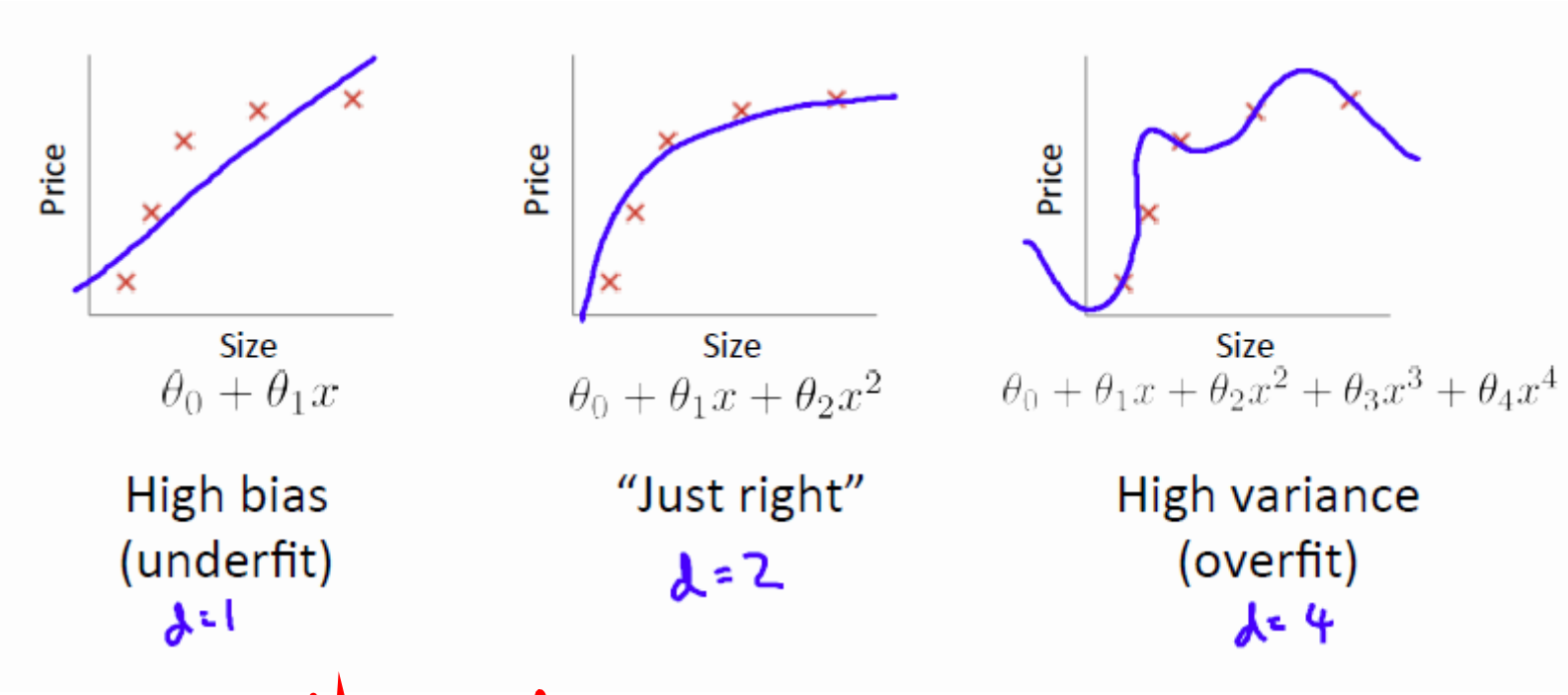




Bias vs Variance

- Sometimes instead of talking about over-fitting and under-fitting we talk about *bias* and *variance*
- If our trained model is very dependent on the training data set (that is it fits it very well), then there will be a large ***variance*** in the models given different training sets.
  - Therefore we say variance and overfitting are related
- Conversely, if our model barely changes at all when it sees different training data, then we say it is **biased**.
  - Which is not necessarily due to underfitting. But it could be…

Over/Under Fitting



Not complicated enough

Detecting Over/Under-Fitting

- How can we detect under-fitting?
  - If we don’t do well on either the training or the testing sets
- How can we detect over-fitting?
  - If we do well on the training set but poorly on the testing set.

Dealing with Over/Under-Fitting

- How can we deal with under-fitting?
  - Make a more complex model
    - May involve need more features
  - Trying a different algorithm
- How can we deal with over-fitting?
  - Use a less complex model
    - May involve using less features
  - Try a different algorithm.
  - Get more data
  - Use a third set to choose between hypothesis (called a *validation set)*.
  - Add a penalization (or regularization) term to the equations to penalize model complexity.