

Fairness, bias, and transparency in Machine Learning

Module 1: regression and classification

This module

- 1.1: Introduction to machine learning and regression
- 1.2: Linear regression
- 1.3: Feature engineering (briefly)
- 1.4: Gradient descent (very briefly)
- 1.5: Linear classification (logistic regression)
- 1.6: Exploring “interpretable” classifiers
- 1.7: Classifier evaluation
- 1.8: The learning pipeline (briefly)
- Case study: gender shades

(approx. 2 weeks)

Regression and classification

1.1: Introduction to machine learning and regression

This section

- Informally introduce **supervised learning** via a few representative tasks
- Informally think about what potential fairness issues those tasks might encounter

But I already saw this in XYZ class?

- I realize *most* people might have seen *most* of this content in another class, but still have to cover the basics for those who haven't
- I'll try to cover it *fairly* quickly – faster than I do in my other classes
- The examples I give will relate specifically *to fairness and bias issues in regression and classification*, so there should be some new content here even for people who have seen regression and classification before
- Will also cover e.g. different classifiers and compare them in terms of interpretability etc.
- That being said **anyone is welcome to skip this content** and come back later, or watch the podcasts brush up on anything they missed

What is supervised learning?

Supervised learning is the process of trying to infer from **labeled data** the underlying function that produced the labels associated with the data

What is supervised learning?

Given labeled training data of the form

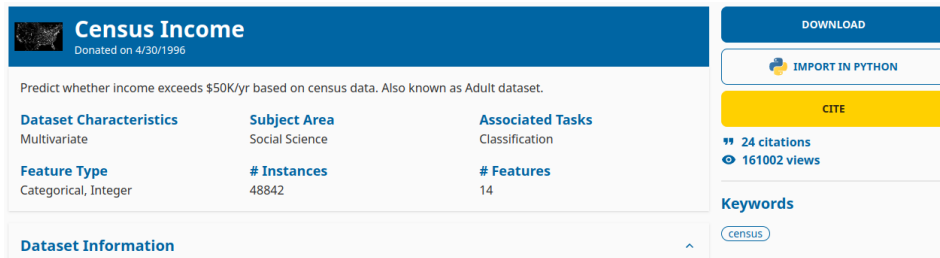
$$\{(data_1, label_1) \dots (data_n, label_n)\}$$

Infer the function

$$f(data) \rightarrow label$$

What is supervised learning?

example task: income prediction



Census Income
Donated on 4/30/1996

Predict whether income exceeds \$50K/yr based on census data. Also known as Adult dataset.

Dataset Characteristics	Subject Area	Associated Tasks
Multivariate	Social Science	Classification
Feature Type	# Instances	# Features
Categorical, Integer	48842	14

Dataset Information

[DOWNLOAD](#)

[IMPORT IN PYTHON](#)

[CITE](#)

24 citations
161002 views

Keywords
census

data_i = [gender, location ...]

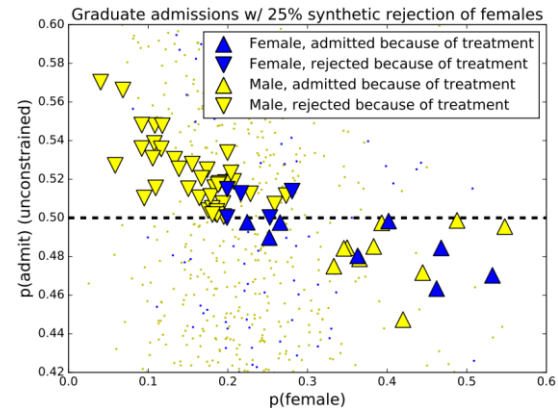
label = $\begin{cases} 1 & \text{if income} > \$50k \\ 0 & \text{otherwise} \end{cases}$

What is supervised learning?

example task: admissions outcomes

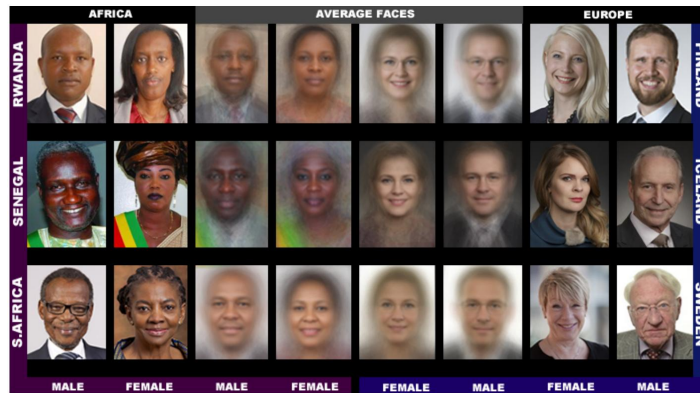
data (feature): [GPA, school, ...]

label: $\begin{cases} 1 & \text{if admitted} \\ 0 & \text{otherwise} \end{cases}$



What is supervised learning?

example task: face recognition



data = (features from) face image

label = male / female

What is supervised learning?

(self study) How does supervised learning differ from other types of machine learning (unsupervised / semi-supervised)?

What does it mean to use “machine learning” to solve these tasks, as opposed to using solutions based on heuristics, business logic, etc.

What are potential fairness issues in these tasks?

Income prediction:

- Predictions may be **less accurate for certain groups** that are less represented in the data
- If used for downstream tasks (e.g. to approve home loans), **predictions may (indirectly) reflect biases**
- In some contexts, use of the input variables (race, gender, etc.) may present **legal issues**
- Predictions may reflect **historical biases** that are inconsistent with recent trends
- Certain features of the data (e.g. outliers) may lead to predictions that harm or benefit certain groups (vague, but we'll explore this example in this module!)

What are potential fairness issues in these tasks?

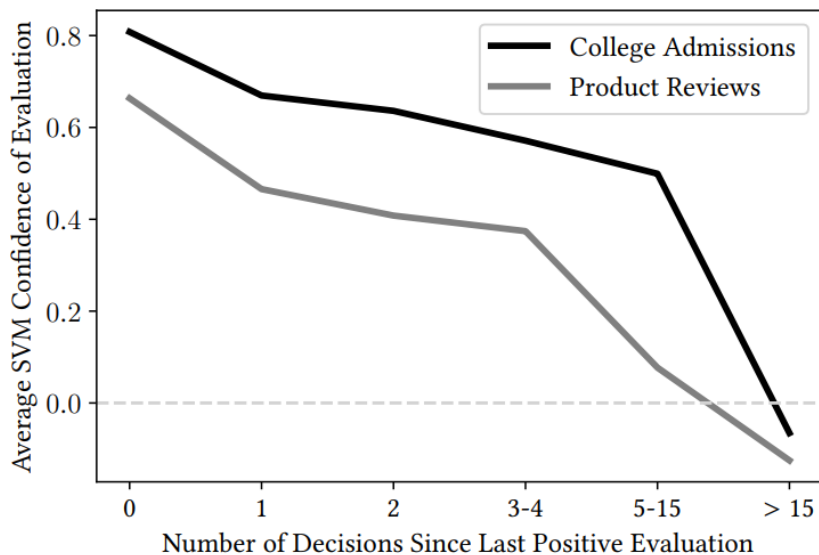
Admissions outcomes (or outcome prediction):

- Again, predictions may be less accurate for some groups based on historical proportions, e.g. consider the model's prediction for a popular school versus one that has never been seen during training
- Some features may reflect cultural differences, e.g. wording or percentiles in reference letters
- Reviewers may historically have had certain biases that are reflected in the data
- etc.

What are potential fairness issues in these tasks?

Admissions outcomes (or outcome prediction):

- More exotic: reviewers decisions may be influenced by their recent decisions!



(case study for later)

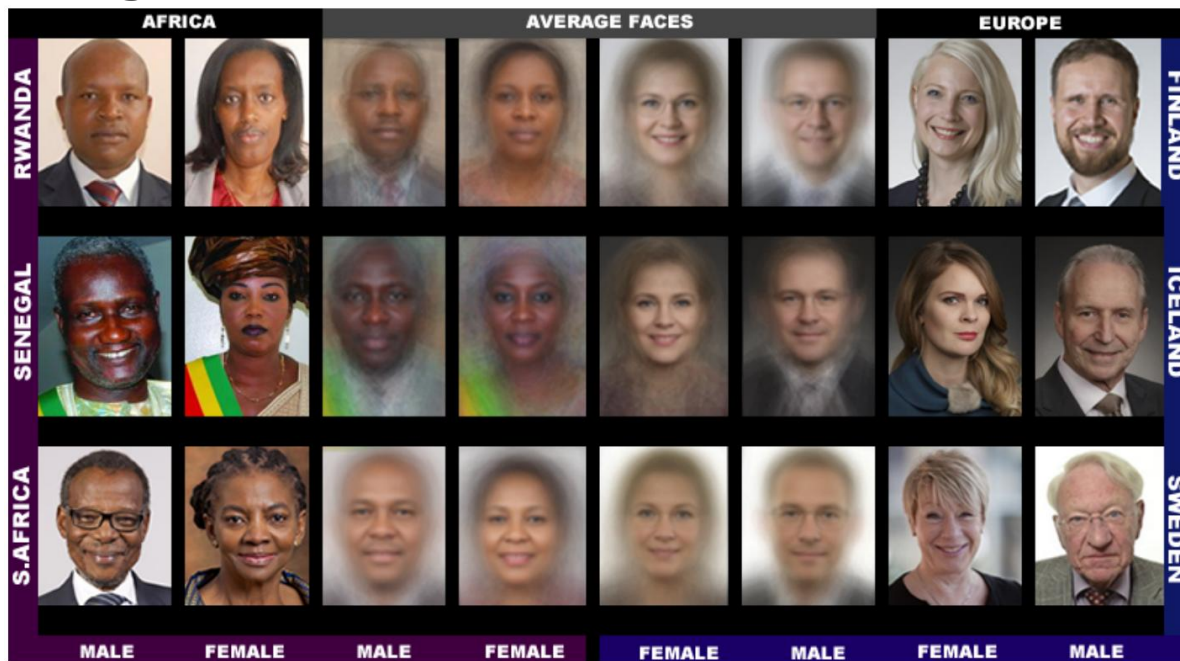
What are potential fairness issues in these tasks?

Face recognition:

- does it work as well for underrepresented groups?
- are errors "equally" spread?
- etc.

What are potential fairness issues in these tasks?

Face recognition:



(case study
for later)

How will “fair” ML differ from regular ML?

In addition to “data” and “labels”, we will (usually) also have some notion of a **sensitive attribute** (or equivalently, a protected group/class)

- This attribute might be e.g. age / gender / skin tone / nationality, etc.
- Such an attribute is *usually* one (or more) of the features in the data and/or a characteristic against which we want to measure outcomes
- We will usually consider this to be a binary attribute (e.g. “female” versus “not female”)

How will “fair” ML differ from regular ML?

- Sometimes, sensitive attributes have **restrictions** around how they can be used, e.g. colleges (in California) may not use race as a factor in admissions decisions.
- This means we could not deploy a classifier that used race as a feature
- We could (maybe?) use the sensitive attribute when training our model and selecting data, *so long as that feature is not available at inference time*
- In other applications (e.g. medicine), using the sensitive attribute is allowable

Sensitive attributes

Income prediction: age / gender / other demographic information

Admissions outcomes: age / gender / other demographic information

Face recognition: age / gender / other demographic information (but in this case these are not features in the data, but rather *prediction outcomes*)

Sensitive attributes

We will use these attributes in various ways:

- To **measure** performance differences between groups (e.g. is a classifier less accurate for females?)
- To **modify** the dataset or algorithm (e.g. to make data or outcomes more “balanced” with respect to a particular group)
- To **restrict** what is available to an algorithm (e.g. in the event of legal barriers to use of certain characteristics)

Study points & take-homes

- I'll include these at the end of each module, use them to help recap the main points!
- For now, just think about how lots of everyday ML problems can have potential fairness considerations!

Regression and classification

1.2: Linear regression

This section

- Introduce **linear regression** algorithms, and solutions to finding a line-of-best fit
- Introduce the **Mean Squared Error** and related error metrics
- Explore some cases where fitting a model can result in a (qualitatively) suboptimal solution
- (Informally) explore strategies to intervene, i.e., “fix” the model to get better outcomes

Notation

To formalize the ideas we've introduced so far, let's start with the following problem variables:

X : features (matrix)
 y : labels (vector)
 z : sensitive attributes \rightarrow binary
(e.g. which instances are females)

Linear regression

Perhaps the simplest association we could assume between our features X and our labels y would be a linear relationship, i.e., that the relationship between X and y is defined as:

$$y = X\theta \quad (\text{or } Ax = b)$$
$$\Rightarrow y_i = x_i \cdot \theta$$

Linear regression

E.g. predict how long a user will play a video game for based on the length of their review (this will demonstrate an example of bias!)

$$\text{time_played} = \theta_0 + \theta_1 \times \text{review_length}$$

$$[1, \text{review_length}] \cdot [\theta_0, \theta_1]$$

Linear regression

Code for this example: **workbook1.ipynb** (see course webpage)

Linear regression

- In this (pretty trivial) problem, our model nearly always *overpredicts* and only *underpredicts* for occasional outlying instances
- Why is this bad?

- not useful for majority of users

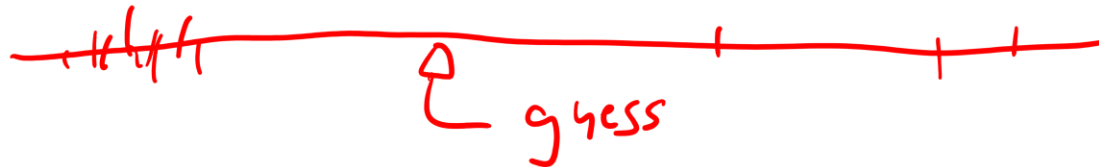
- not useful for anyone!

- if we just made lower guesses, wouldn't that be better?

Linear regression

- In this (pretty trivial) problem, our model nearly always *overpredicts* and only *underpredicts* for occasional outlying instances
- Why does this happen?

- we're minimizing a squared error $(y_i - x_i \cdot \theta)^2$
- so, large errors are heavily penalized (see later)
- so, guess in the middle



What is actually being optimized?

When we fit our regressor we actually solved a problem of this form:

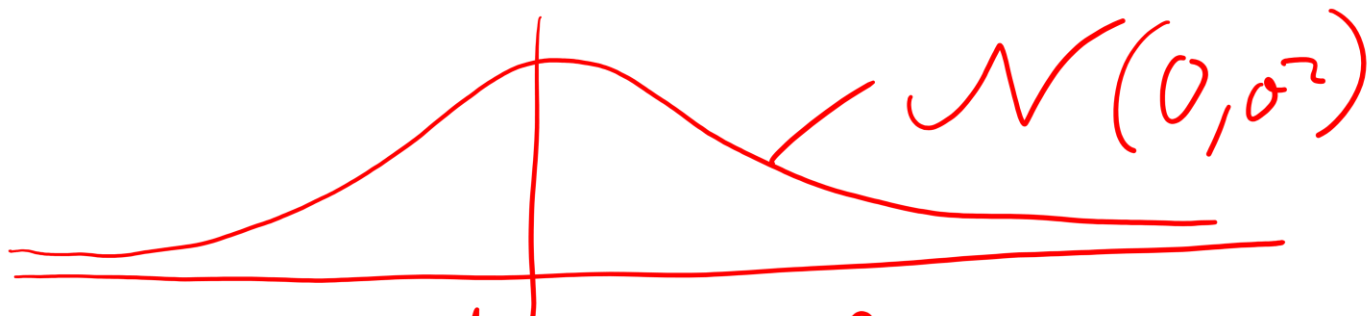
arg Min _{θ}

$$\sum_i \underbrace{(y_i - x_i \cdot \theta)^2}_{\text{error}}$$

Sum of squared errors (SSE)

What is actually being optimized?

proof:



$$d_i = y_i - x_i \cdot \theta$$

$$y_i = x_i \cdot \theta + \mathcal{N}(0, \sigma^2)$$

$$P_{\theta}(y|X) = \prod_i \frac{1}{\sqrt{2\pi}} e^{-\frac{(y_i - x_i \cdot \theta)^2}{2\sigma^2}}$$

$$\max_{\theta} P_{\theta}(y|X) = \max_{\theta} - \sum_i (y_i - x_i \cdot \theta)^2 = \min_{\theta} \sum_i (y_i - x_i \cdot \theta)^2$$

What is actually being optimized?

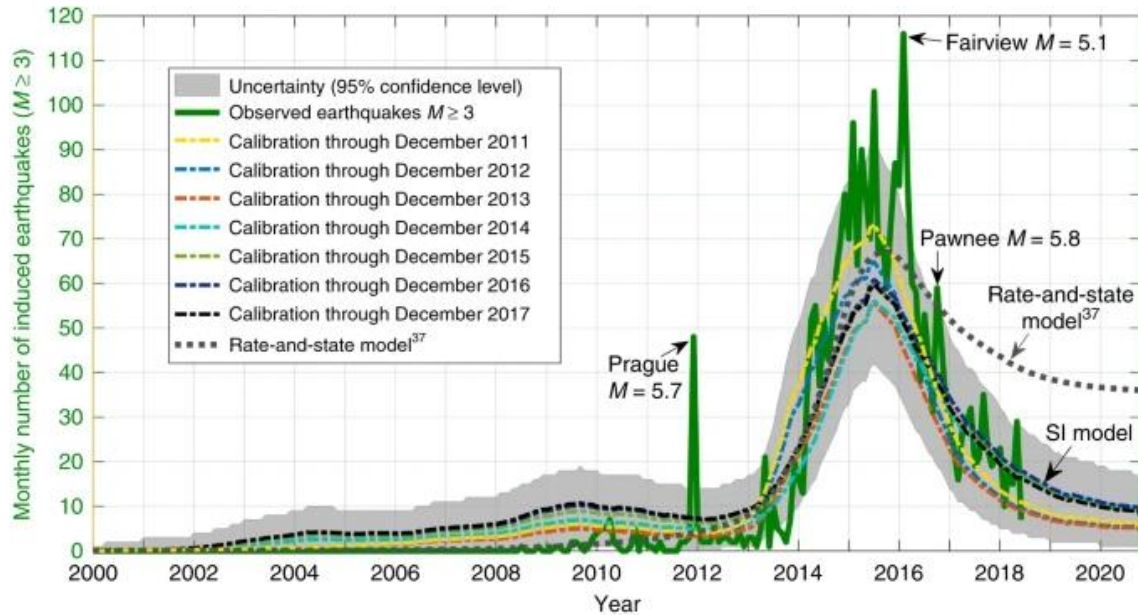
Why does such a model nearly always overpredict on our dataset?

- errors are heavily penalized (by squaring!)
- "cheaper" to make many small/medium errors than a few big ones

Q: are normally distributed errors "right" for this data?

What is actually being optimized?

(e.g. seismicity prediction)



What is actually being optimized?

(e.g. seismicity prediction)

- lots of small values
- a few big ones
- lowest MSE \rightarrow
always slightly overpredict

What is actually being optimized?

- So, the model we fit is the *best we could possibly do* in terms of the MSE
- Put differently, interventions to “fix” this model will cause its MSE to be *worse*
- But it is a “bad” model!

What sort of interventions might we apply to make this model “better”, and on what basis would we argue that it’s better?

Note: statistical bias versus "bias"

- But I learned in my stats class that linear regression is an “unbiased estimator”?!?
- This just means that, if model assumptions are satisfied, our procedure for fitting the model parameters will estimate the true parameter values (in expectation) (Q: can anyone word this better than me)

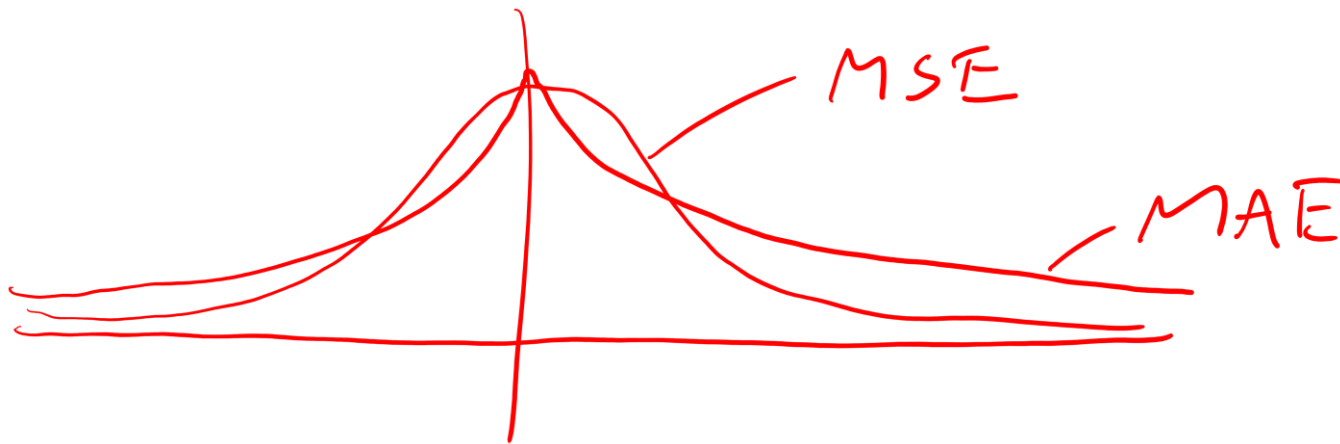
Not related to colloquial definitions
of bias!

Some model interventions

Designing appropriate “fairness interventions” will be the goal of following modules, but for now let’s think about the problem informally. E.g. we could:

1. Change what is being optimized: MSE vs MAE

(see workbook1 code)



Some model interventions

2. Remove outliers from the dataset

— errors will be more
"normal" w/o outliers

Some model interventions

3. Transform the features (or labels) to better correspond to the error distribution

e.g. fit $\log(y)$
rather than fitting y

Some model interventions

4. Collect better features

- Maybe outliers are actually predictable!
(if we have the right features)
- Different features could change the error distribution

Food for thought

What did we just do?

We just made our model work better for *most of the samples* by making it worse for a *minority of the samples* (gamers!)

- Think of how this type intervention could be problematic in some scenarios!
- **But:** as outliers, they were getting fairly useless predictions anyway, so were they really “harmed” by this intervention anyway?

Ultimately, these “quick and dirty” model interventions **can be helpful but highlight why the situation is more nuanced** – and why we’ll need more sophisticated interventions (and metrics)

Interpretability

What about model *interpretability*?

We'll spend a large fraction of the course discussing interpretability, but for now, let's just think about the parameters of our linear model

Interpretability

$$\text{time} = \theta_0 + \theta_1 \times \text{length}$$

θ_0 = offset - what would time be if length = 0 (is this possible?)

θ_1 = slope - how much does a unit change in length change the prediction of time
- If no other feature changes!

Interpretability

$$GPA = 3.2 + 0.4 \times [\text{lives off campus}]$$

- commuting is good for GPA?
- campus life is bad for GPA?

$$GPA = 3.2 - 0.1 \times [\text{off campus}] + 0.03 \times [\text{age}]$$

- living off campus is bad
- but being older is good, and older students live off campus

Food for thought

Simple models (like linear models) might seem “interpretable” since we can easily reason about their parameters

But this can easily be misleading due to the way parameters interact; is their greater interpretability just an illusion?

If the model is not accurate, does it really matter whether it’s “interpretable” or not?

Won’t really discuss now, but these will be the types of questions we examine in the second half of the course

Anscombe's quartet

(see `workbook1.ipynb` code)

Anscombe's quartet

What's the significance of all this?

- Models are **approximations** of data
- If we draw conclusions from models, those conclusions will probably be at least as wrong as our models are!
- Specific models have certain assumptions and will poorly fit datasets that violate those assumptions
- But that can be very hard to tell without *looking at the data!*

Study points & take-homes

- Discussed how particular modeling choices, such as the use of the MSE, are due to underlying modeling assumptions
- In the example we showed, the “assumption” was about the shape of the error distribution
- Data that violate these assumptions can lead to undesirable models (even when using (statistically) “unbiased” estimators!); though this is just the first of many potential issues
- Saw some heuristic intervention strategies, which (roughly) tried to better align the data with model assumptions

Regression and classification

1.3: Feature engineering

This section

- Explore strategies for feature engineering
- I mostly don't want to spend too much time on this in this class, as it will slow us down – so just want to give some pointers for self-revision (mostly, see textbook: https://cseweb.ucsd.edu/~jmcauley/pml/pml_book.pdf)

Feature engineering – main points

1. Feature **transformations**
2. Binary and categorical features: **one-hot encodings**
3. **Missing values**
4. **Temporal** features
5. Transformation of **output** variables

These are all covered in the textbook, Section 2.3.1-2.3.5 (respectively)

Feature engineering – main points

1. Feature transformations

Linear models take the form:

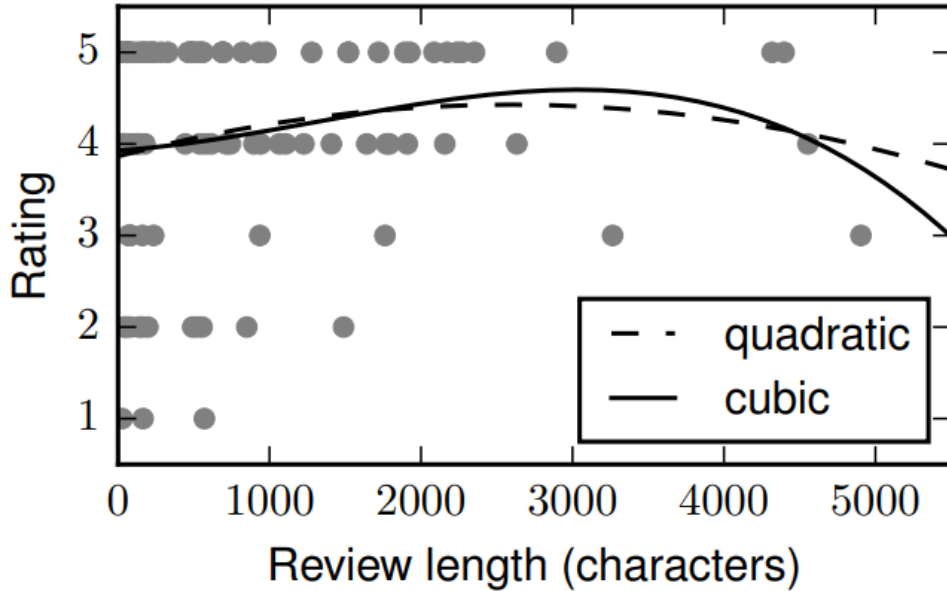
$$y = X \cdot \theta; \quad y_i = x_i \cdot \theta$$

That is they are linear in *theta* – we can still apply arbitrary transformations to the *features*!

$$\text{height} = \theta_0 + \theta_1 \times \text{age} + \theta_2 \times \text{age}^2 + \theta_3 \times \text{Min}(\text{age}, 18) \\ \text{(etc.)}$$

Feature engineering – main points

Rating vs. review length



E.g. fitting a polynomial function
(from textbook, see also textbook
code samples)

Feature engineering – main points

2. Binary and categorical features: **one-hot encodings**

Suppose we want to encode e.g. job qualification as a function of gender. We might want a model that looks something like:

$$\text{height} = \theta_0 + \theta_1 \times \text{gender}$$

How would we *encode* gender as a number?

Feature engineering – main points

2. Binary and categorical features: **one-hot encodings**

Naive encoding:

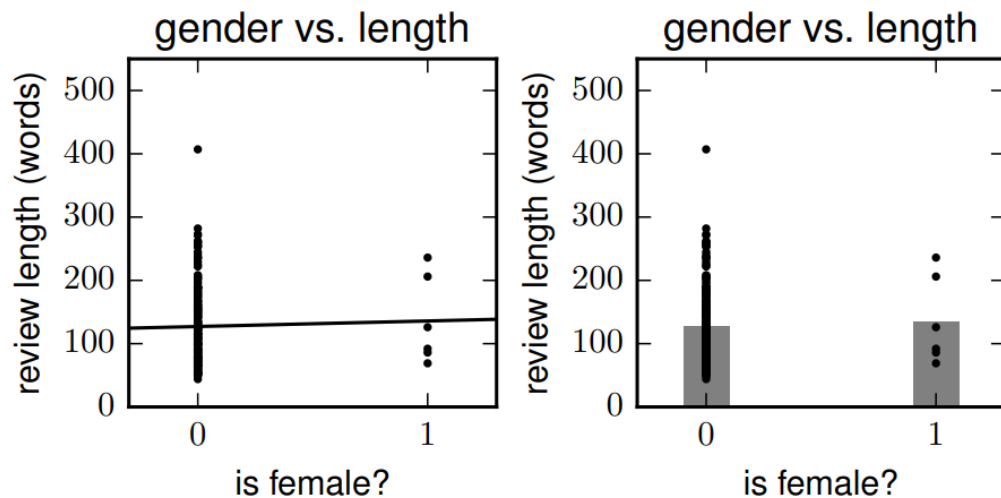
$$\text{length} = \theta_0 + \theta_1 \times \text{gender}$$
$$\text{gender} = \begin{cases} 0 & \text{if male} \\ 1 & \text{if female} \end{cases}$$

Feature engineering – main points

2. Binary and categorical features: **one-hot encodings**

Our naive encoding works pretty much out-of-the-box

Note that what we actually fit is more like a bar plot than a “line”



Feature engineering – main points

2. Binary and categorical features: **one-hot encodings**

What if our features are not binary?

Naive encoding:

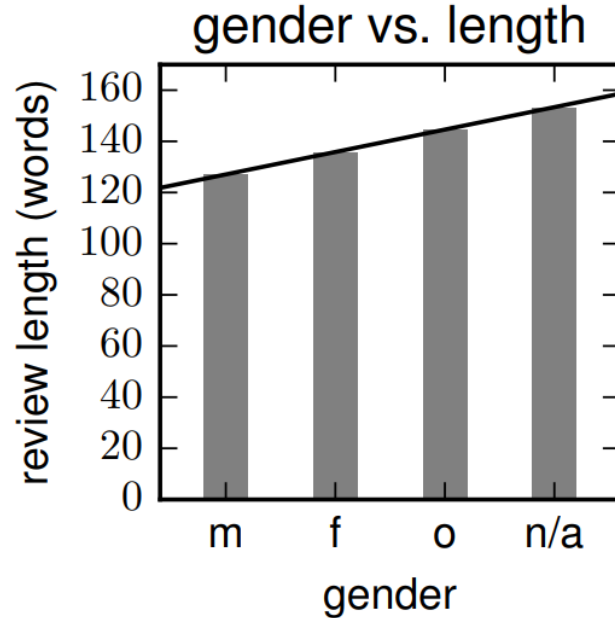
$$\theta_0 + \theta_1 \times \text{gender}$$

gender = 0 if female
1 if male
2 if other etc.

Feature engineering – main points

2. Binary and categorical features:
one-hot encodings

Naive encoding implies
relationships among the feature
values that *don't necessarily exist*



Feature engineering – main points

2. Binary and categorical features: **one-hot encodings**

It sort of makes sense that this didn't work: we want to fit four possible values, but our model only has two parameters

Try a different encoding:

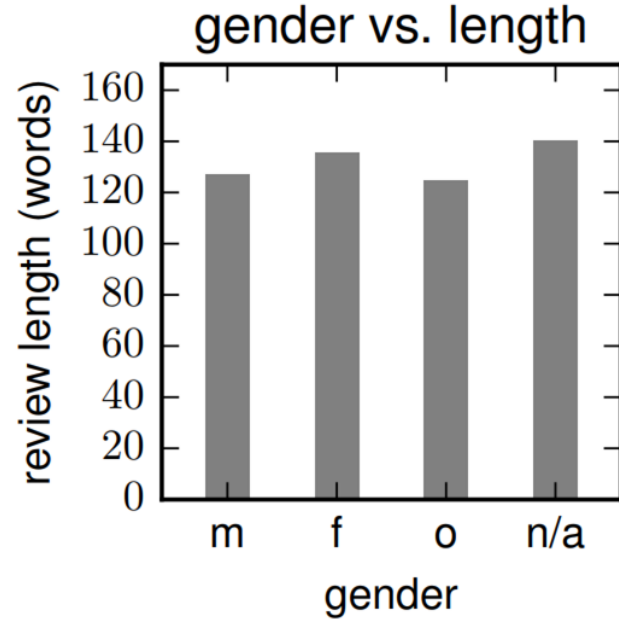
$$x = [1, g = \text{male}, g = \text{female}, g = \text{other}, \dots]$$

$$\text{female} \rightarrow [1, 0, 1, 0, \dots]$$

Feature engineering – main points

2. Binary and categorical features:
one-hot encodings

Model now has as many
parameters as there are feature
values



Feature engineering – main points

2. Binary and categorical features: **one-hot encodings**

- The above is called a **one-hot encoding**, and will show up when building models based on all sorts of categorical features (though we'll usually hide these details)
- Can also have “multi-hot” encodings when multiple categories can be active simultaneously, e.g. imagine a feature based on “nationality”
- Be careful with the details! E.g. to make sure feature dimensions aren't redundantly encoded – see textbook for details

Feature engineering – main points

3. Missing features

What should we do if certain features are (sometimes) missing?

With (e.g.) gender this was pretty easy – just make a dimension in our encoding for “n/a”

But what about a numerical feature like income?

Feature engineering – main points

3. Missing features

Two commonly used options:

- Missing value **imputation**: replace any missing instances by a specific value, e.g. by a population average or mode

Feature engineering – main points

3. Missing features

Two commonly used options:

- Missing value **indicator**: replace our feature with two features:

$$x' = \begin{cases} [x, 0] & \text{if } x \text{ available} \\ [0, 1] & \text{otherwise} \end{cases}$$

↳ missing value indicator

Feature engineering – main points

3. Missing features

Two commonly used options:

- Missing value **indicator**: replace our feature with two features, so that the parameterized model becomes:

$$\begin{aligned} y &= \theta_0 + \theta_1 x && \text{(if available)} \\ y &= \theta_0 + \theta_2 && \text{(if missing)} \end{aligned}$$

So θ_2 learns what to predict when features are missing

Feature engineering – main points

4. Temporal features

Suppose we want to predict a regression outcome (such as a rating) based on the day of the week

Once again let's try a naive encoding:

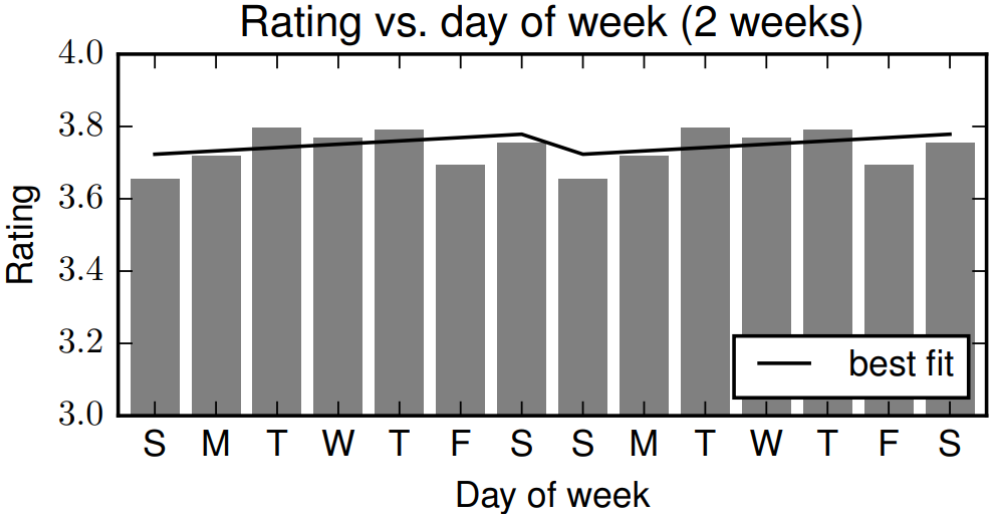
$$\text{rating} = \theta_0 + \theta_1 \times \text{day}$$

mon=1, tue=2, etc.

Feature engineering – main points

4. Temporal features

line of best fit with naive encoding:



Feature engineering – main points

4. Temporal features

Can just try a one-hot encoding again! E.g:

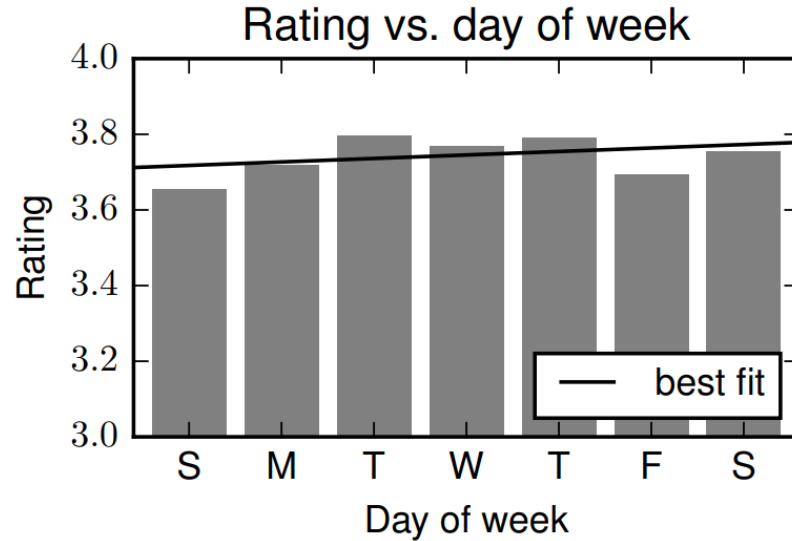
$$\text{Mon} = [1, 0, 0, 0, 0, 0]$$

$$\text{Tue} = [0, 1, 0, 0, 0, 0]$$

Feature engineering – main points

4. Temporal features

line of best fit with one-hot encoding:



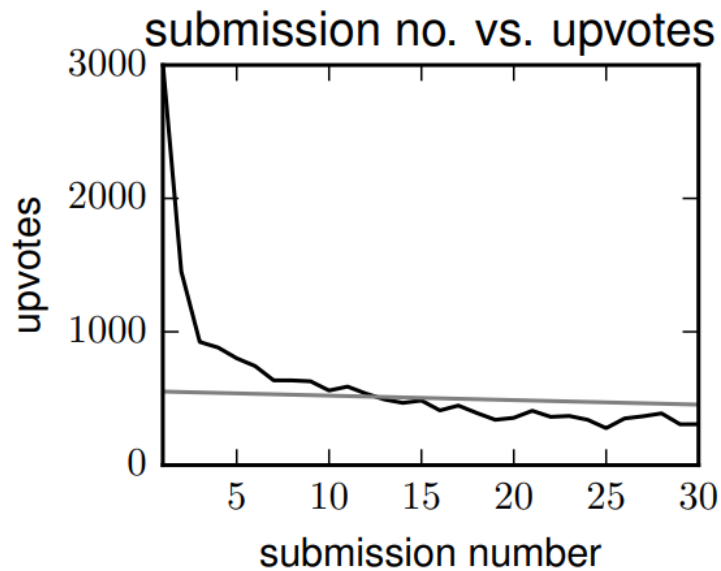
Feature engineering – main points

5. Transformation of output variables

Consider for example trying to predict upvotes on reddit with a linear model

$$\text{upvotes} = \theta_0 + \theta_1 \times [\text{\# of times submitted}]$$

Feature engineering – main points



Feature engineering – main points

5. Transformation of output variables

Details of the dataset don't matter too much: point is that the relationship between features and the output variable is **not linear** – so at first glance we might think it can't be fit by a linear model

Looks a bit more like an exponential function, e.g.:

$$\text{updates} = \exp(\theta_0 + \theta_1 [\# \text{ subs}])$$

Feature engineering – main points

5. Transformation of output variables

Although the exponential function is not linear (in theta), the equation is equivalent to:

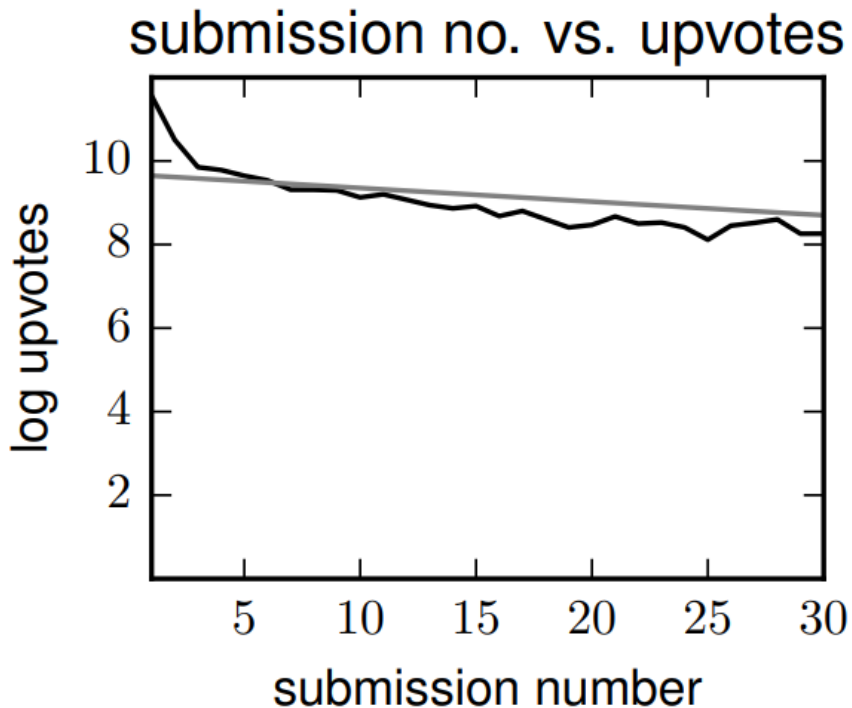
$$\log(\text{upvotes}) = \theta_0 + \theta_1 [\# \text{ subs}]$$

Which is linear in theta!

Feature engineering – main points

Let's fit this new model:

Looks much better, and (after we transform the predictions back to their original scale) has lower MSE than the original model



Feature engineering – main points

Feature engineering is certainly not the main point of this course (hence why I only spent ~10 slides on it), but we'll apply these types of simple operations to various models throughout the course

So, revise this material on your own (above examples are from the textbook: https://cseweb.ucsd.edu/~jmcauley/pml/pml_book.pdf) if you're behind on any of this

Feature engineering

Code examples in textbook Chapter 2:

<https://cseweb.ucsd.edu/~jmcauley/pml/code/chap2.html>

<https://cseweb.ucsd.edu/~jmcauley/pml/code/Chapter%202.ipynb>

Study points & take-homes

Five feature engineering strategies to cover most situations:

1. *Features* can be arbitrarily transformed in linear models (even though parameters cannot be)
2. One-hot encodings
3. Dealing with missing attributes
4. Dealing with temporally-evolving data
5. Transformation of output variables

Regression and classification

1.4: Gradient descent

This section

- Define the *gradient descent* process
- Show how this process can be used to fit complex models

Fitting models with gradient descent

So far, when solving regression problems, we've looked for *closed form* solutions. That is, we've set up a system of equations and attempted to solve them for θ .

As we begin to fit more complex models, a closed form solution may no longer be available.

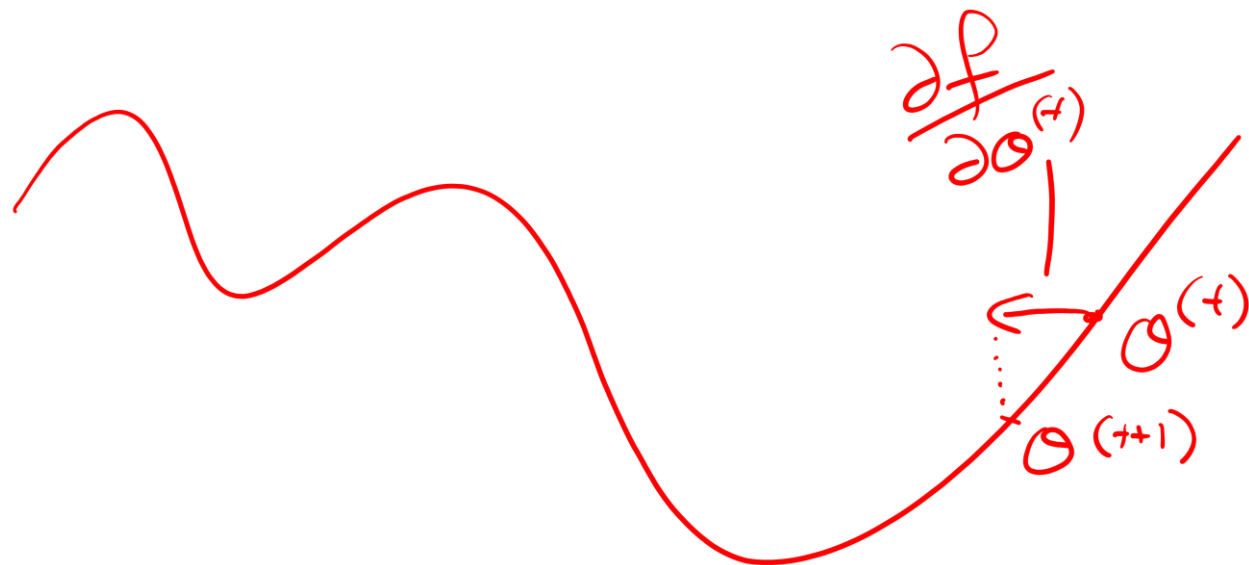
Fitting models with gradient descent

Gradient descent is an approach to search for the minimum value of a function, by iteratively finding better solutions based on an initial starting point.

1. Start with an initial guess for θ
2. Compute the derivative $\partial / \partial \theta f(\theta)$. Here $f(\theta)$ is the MSE (or whatever criterion we are optimizing) under our model θ
3. Update our estimate of $\theta := \theta - \alpha \cdot f'(\theta)$
4. Repeat Steps (2) and (3) until convergence

$$\theta^{(t+1)} \rightarrow \theta^{(t)} - \alpha \frac{\partial f}{\partial \theta}(\theta^{(t)})$$

Fitting models with gradient descent



Fitting models with gradient descent

If actually trying to implement such a model “from scratch”, main issues include:

- Given a particular starting point, the model may only achieve a *local* rather than a *global* optimum. It might be important to come up with a good initial “guess” for θ , or to investigate variants less susceptible to local minima.
- The step size must be chosen carefully. If α is too large, the procedure may overshoot the minimum and obtain a *worse* solution during the next iteration
- “Convergence” is not well-defined. We might define it based on the change in θ (or $f(\theta)$) between iterations, or we might use our validation set (see later)

Linear regression via gradient descent

Let's consider the example of minimizing the Mean Squared Error of a linear model:

$$\operatorname{argmin}_{\theta} \sum_i (x_i \cdot \theta - y_i)^2$$

Linear regression via gradient descent

The derivative can be computed as follows:

$$\frac{\partial}{\partial \theta_k} \sum_i (x_i \cdot \theta - y_i)^2$$
$$= \sum_i 2 x_{ik} (x_i \cdot \theta - y_i)$$

Study points & take-homes

- Not much, this section is mostly here for completeness
- We will occasionally look at gradient-based methods in later modules, so try to at least become comfortable with (the idea of) computing gradients of objective functions with respect to particular parameters

Regression and classification

1.5: Linear classification

This section

- Introduce classification
- Show how linear regression can be adapted to develop logistic regression
- Nothing on fairness issues yet (save this for a following section on classifier evaluation)

Intro to classification

To develop *regressors*, we wanted to build a model f_{θ} whose estimates $f_{\theta}(x_i)$ were as close as possible to the (real-valued) labels y_i

To develop *classifiers*, we might instead seek to associate positive values of $x_i \cdot \theta$ with positive labels ($y_i = 1$), and negative values of $x_i \cdot \theta$ with negative labels ($y_i = 0$).

Intro to classification

So our classifier would look like:

$$y_i = \begin{cases} 1 & \text{if } X_i \cdot \theta > 0 \\ 0 & \text{otherwise} \end{cases}$$

Intro to classification

We could then write down the *accuracy* associated with such a model:

$$\frac{1}{|y|} \left(\sum_{y_i=1} \delta(x_i \cdot \theta > 0) + \sum_{y_i=0} \delta(x_i \cdot \theta \leq 0) \right)$$

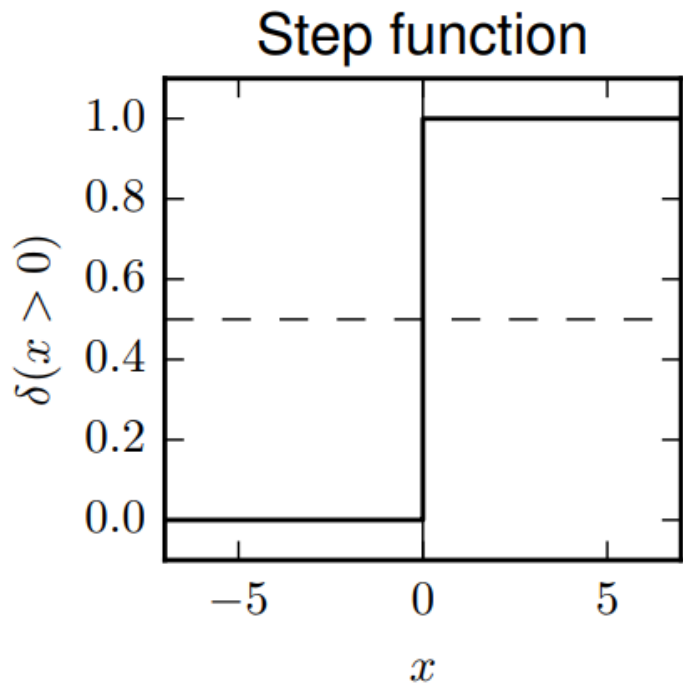
↘ 1 if argument is true

Intro to classification

Sounds great! All we have to do is select the value of θ that maximizes the accuracy of the classifier

Let's try doing gradient *ascent*...

Intro to classification



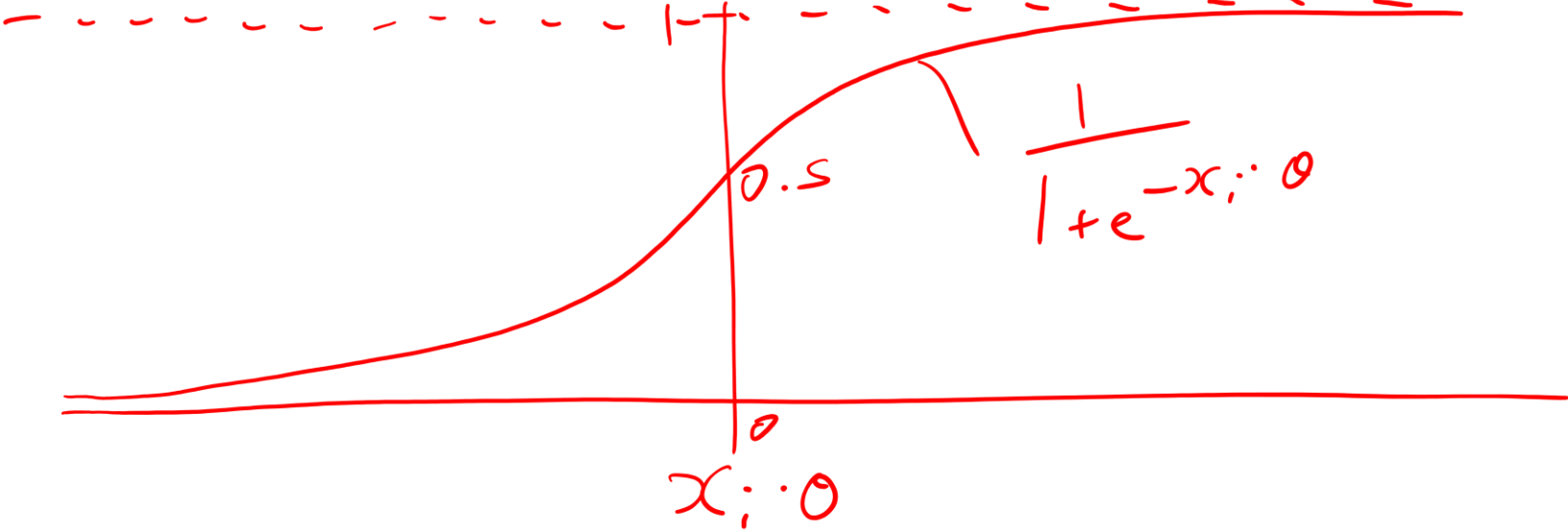
Problem: the derivative is zero everywhere!

Unlike regressors,
no notion of being
"a bit" right/wrong

Intro to classification

Solution: sigmoid function

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$



Logistic regression

Logistic regression treats the sigmoid function as a measure of the **probability** that a sample has a particular label, i.e.,

$$P_{\theta}(y_i=1 | x_i) = \sigma(x_i \cdot \theta)$$

Logistic regression

Assuming our samples are independent, we can then write down the probability of our entire dataset (this is called the “likelihood”):

$$p_{\theta}(y|X) = \prod_{y_i=1} \sigma(x_i \cdot \theta) \prod_{y_i=0} (1 - \sigma(x_i \cdot \theta))$$

+ive examples have high $x_i \cdot \theta$

-ive examples have low $x_i \cdot \theta$ (large negative)

Logistic regression

We can now do gradient ascent! Take the log of the likelihood:

$$\begin{aligned} \ell_{\theta}(y/X) &= \sum_{y_i=1} \log \left(\frac{1}{1+e^{-x_i \cdot \theta}} \right) + \sum_{y_i=0} \log \left(\frac{e^{-x_i \cdot \theta}}{1+e^{-x_i \cdot \theta}} \right) \\ &= \sum_i -\log(1+e^{-x_i \cdot \theta}) - \sum_{y_i=0} x_i \cdot \theta \end{aligned}$$

Logistic regression

And compute its derivative:

$$\text{obj.} \quad \sum_i -\log(1 + e^{x_i \cdot \theta}) + \sum_{y_i=0} -x_i \cdot \theta$$

$$\frac{\partial \text{obj.}}{\partial \theta} = \sum_i \frac{x_{i:k} e^{-x_i \cdot \theta}}{1 + e^{-x_i \cdot \theta}} + \sum_{y_i=0} -x_{i:k}$$
$$\sum_i x_{i:k} (1 - \sigma(x_i \cdot \theta))$$

Logistic regression

Lots of other little details:

- Getting step-sizes etc. right
- Setting termination criteria correctly
- Initialization
- Dealing with local vs. global optima (more relevant as we explore more complex models)

But in practice we'll just let our library function do this for us!

Logistic regression

Once trained, we classify points according to:

$$P_{\theta}(y_i = 1 | x_i) > 0.5 \quad ?$$

$$\sigma(x_i \cdot \theta) > 0.5$$

$$x_i \cdot \theta > 0$$

Logistic regression

Code examples in textbook Chapter 3:

<https://cseweb.ucsd.edu/~jmcauley/pml/code/chap3.html>

<https://cseweb.ucsd.edu/~jmcauley/pml/code/Chapter%203.ipynb>

Study points & take-homes

- Likewise, mostly here for completeness: just want to make sure everyone knows how at least one classifier works!
- Later, when we explore interventions (Module 3), we will explore how to modify the objective functions of classifiers, so worth understanding at least at a surface level

Regression and classification

1.6: Exploring “interpretable” classifiers

This section

- Exploration of alternative classification techniques
 - Decision trees
 - Nearest neighbors
 - SVMs (maybe, probably skip if we're more than two weeks in by now!)
 - Multilayer perceptrons (maybe)
- Generally, just exploring different ways a classifier can be built
- Informally, exploring whether these classifiers are “more interpretable” than linear classification techniques

A few words about other classification techniques...

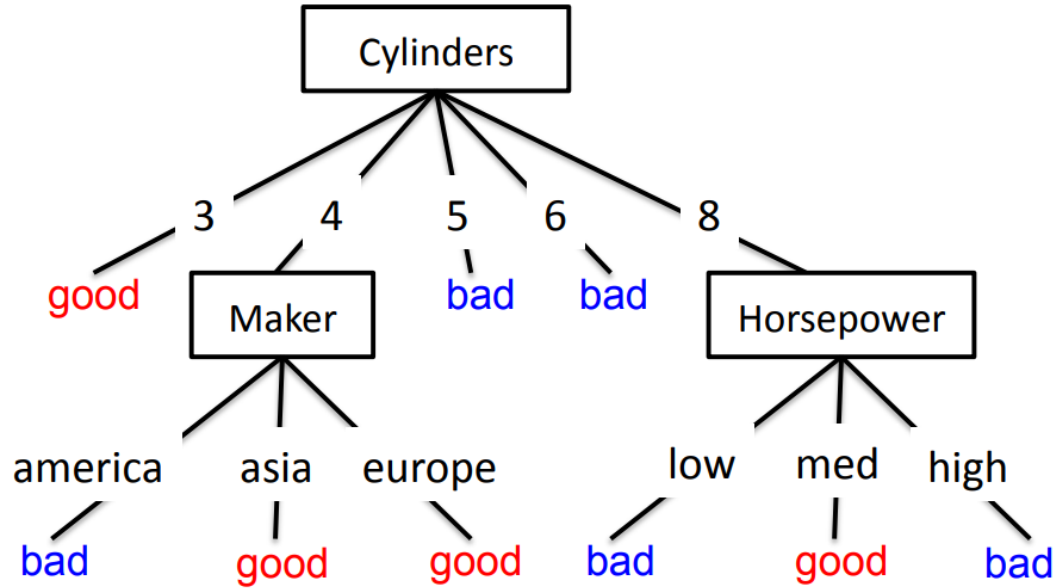
- Decision trees
- Nearest neighbors
- SVMs (maybe)
- Multilayer perceptrons (maybe)

Decision trees and nearest neighbors are intended to demonstrate that different types of classifiers can have quite a different notion of “interpretability” compared to a linear classifier;

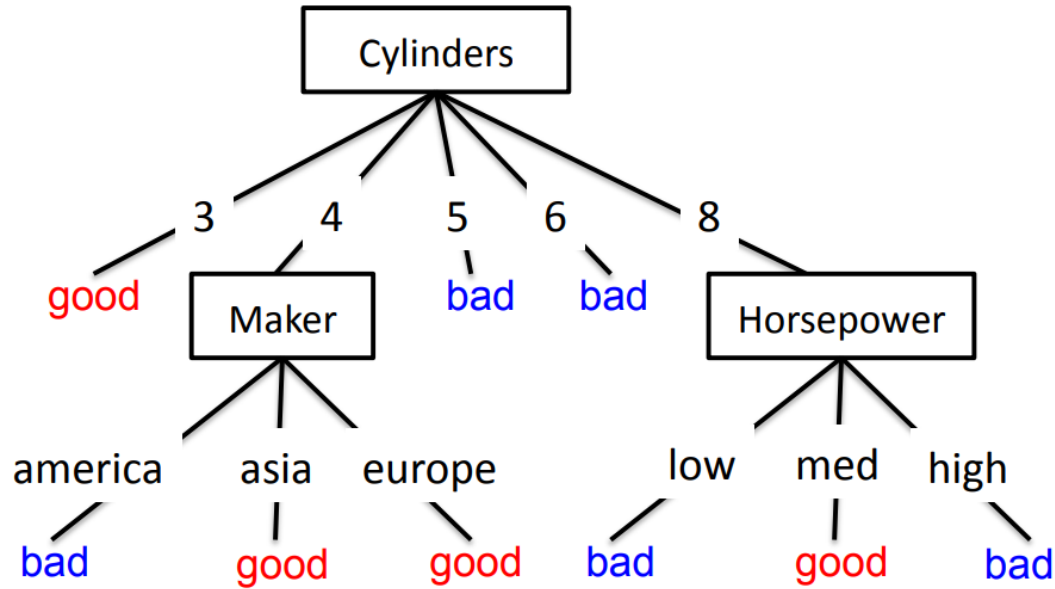
SVMs are intended to demonstrate that “good” classifiers depend on specific modeling assumptions, which determine the classification boundary;

Multilayer perceptrons are included to introduce “neural” models and (later) to explore the notion that neural models are “less interpretable” than other models

Decision trees



Decision trees

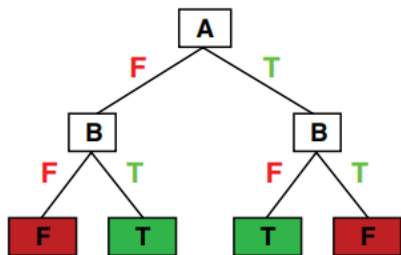


- Internal nodes test attributes
- Leaf nodes assign a class
- Branches for each attribute value
- Inputs are classified by traversing the tree

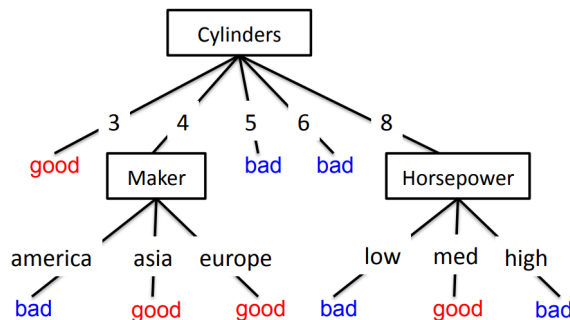
Decision trees

- Decision trees can represent any function of the input attributes (though could require exponentially many nodes to do so)

A	B	A xor B
F	F	F
F	T	T
T	F	T
T	T	F



(Figure from Stuart Russell)

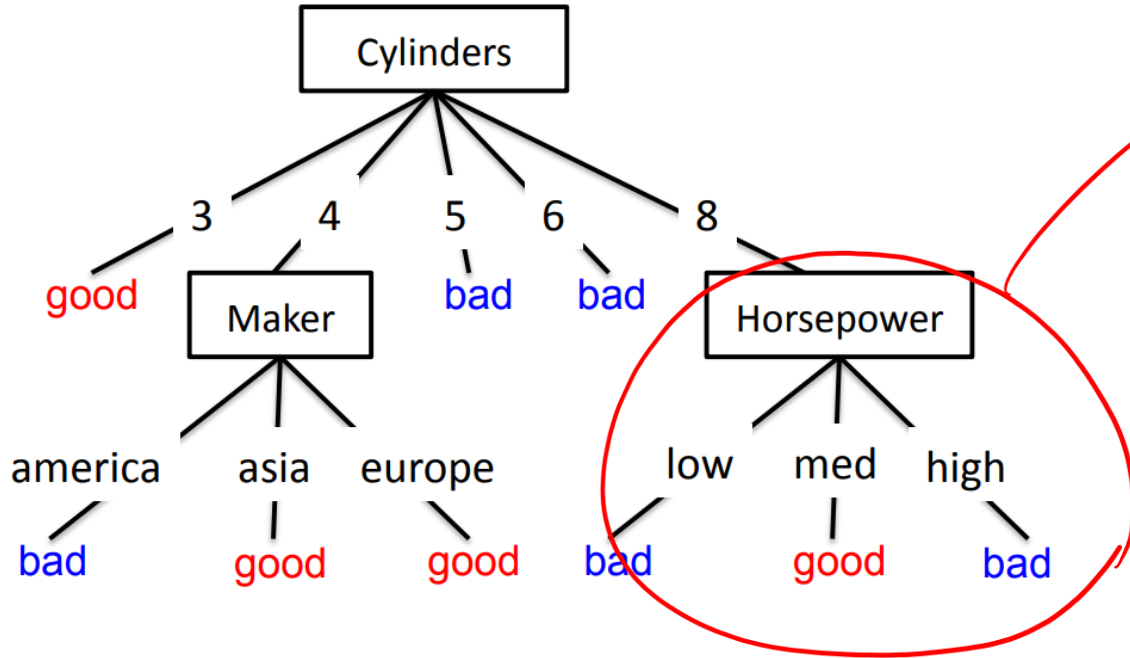


$\text{cyl}=3 \vee (\text{cyl}=4 \wedge (\text{maker}=\text{asia} \vee \text{maker}=\text{europe})) \vee \dots$

Decision trees

- Learning the simplest (smallest) decision tree is NP-complete (Hyafil & Rivest '76)
- Instead will use greedy heuristics to create:
 - Start with an empty tree
 - Split on next best attribute
 - Recurse

Decision trees



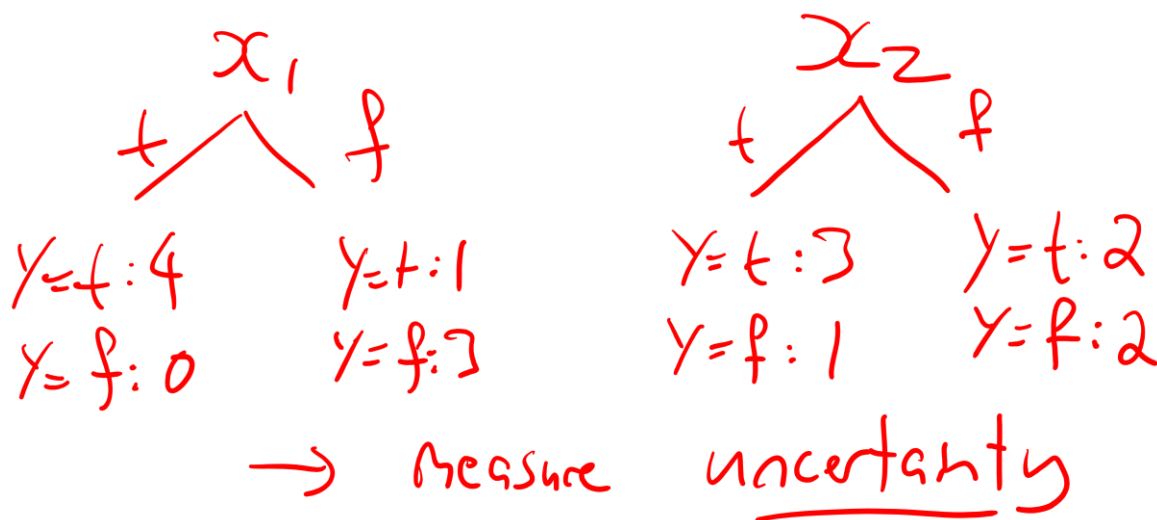
(sub) tree

- fit using a subset of the data

Decision trees

Q: How should we choose a good attribute to split?

E.g. for this subtree, should we split on x_1 or x_2 ?



x_1	x_2	y
T	T	T
T	F	T
T	T	T
T	F	T
F	T	T
F	F	F
F	T	F
F	F	F

Decision trees

Q: How should we choose a good attribute to split?

E.g. for this subtree, should we split on x_1 or x_2 ?

A: treat counts at leaves as probability distributions so that we can measure uncertainty

x_1	x_2	y
T	T	T
T	F	T
T	T	T
T	F	T
F	T	T
F	F	F
F	T	F
F	F	F

Decision trees

Q: How should we choose a good attribute to split?

A: treat counts at leaves as probability distributions so that we can measure uncertainty

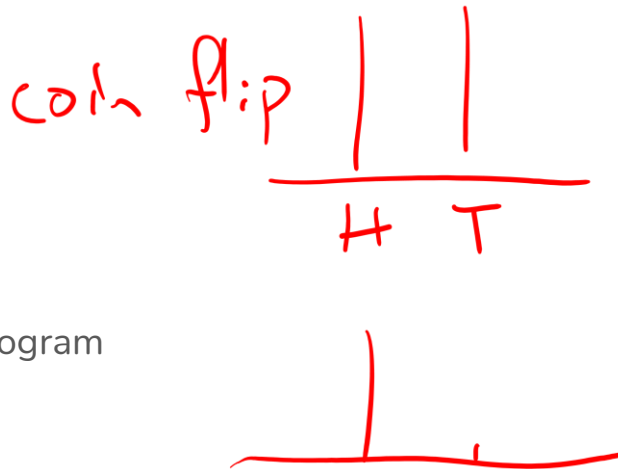
A “good” split is one that makes us more certain about our classification:

- Deterministic (all true or all false) is good
- Uniform is bad
- What about something in between?

Decision trees

Entropy is a statistical measure of the level of *uncertainty* in an event:

- High entropy (entropy close to 1):
 - From a uniform-like distribution
 - Flat histogram
 - Sampled values are less predictable
- Low entropy (entropy close to 0):
 - Highly variable (peaks and valleys) distribution/histogram
 - Sampled values are more predictable



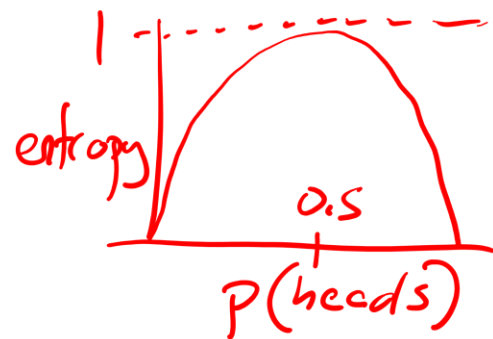
Decision trees

Entropy formula:

$$H(Y) = - \sum_{i=1}^k P(Y=y_i) \log_2 P(Y=y_i)$$

"expected #bits needed to encode randomly drawn value of Y "

$$\begin{aligned} \text{coin flip: } & -2 \times 0.5 \log_2(0.5) \\ \text{(fair)} & = 1 \end{aligned}$$



Decision trees

Entropy example:

$$H(Y) = -\sum_i P(Y=y_i) \log_2 P(Y=y_i)$$

$$P(Y=t) = \frac{5}{6} \quad P(Y=f) = \frac{1}{6}$$

$$\begin{aligned} H(Y) &= -\frac{5}{6} \log_2 \frac{5}{6} - \frac{1}{6} \log_2 \frac{1}{6} \\ &= 0.65 \end{aligned}$$

x_1	x_2	y
T	T	T
T	F	T
T	T	T
T	F	T
F	T	T
F	F	F

Decision trees

To split, we want to measure what the entropy will be if we split based on a particular attribute. For this we measure the **conditional** entropy $H(Y|X)$ of a variable Y conditioned on X :

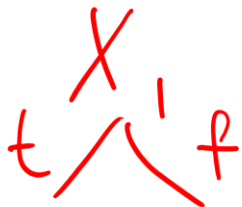
$$H(Y|X) = \sum_{j=1}^u P(X=x_j) \sum_{i=1}^k P(Y=y_i|X=x_j) \log_2 P(Y=y_i|X=x_j)$$

Decision trees

Conditional entropy example:

$$P(X_1 = t) = \frac{4}{6}$$

$$P(X_1 = f) = \frac{2}{6}$$



$$Y = t : 4$$

$$Y = f : 0$$

$$Y = t : 1$$

$$Y = f : 1$$

$$\begin{aligned} H(Y|X_1) &= -\frac{4}{6} (1 \log_2 1 + 0 \log_2 0) \\ &\quad - \frac{2}{6} (\frac{1}{2} \log_2 \frac{1}{2} + \frac{1}{2} \log_2 \frac{1}{2}) \\ &= \frac{2}{6} \end{aligned}$$

x_1	x_2	y
T	T	T
T	F	T
T	T	T
T	F	T
F	T	T
F	F	F

Decision trees

Want to measure the **information gain**, i.e., the decrease in entropy (uncertainty) after splitting:

$$IG = H(Y) - H(Y|X)$$

e.g. $IG(X_1) = H(Y) - H(Y|X_1)$
 $= 0.65 - 0.33$

Decision trees

Overall algorithm:

- Start with an empty tree
- Split on **next best attribute (feature)** (e.g. information gain)

$$\operatorname{argmax}_i \text{IG}(X_i) = \operatorname{argmax}_i H(Y) - H(Y|X_i)$$

- Recurse

Decision trees

When should we stop?

- If all samples in a subset have the same label or attribute values, can stop
- What to do otherwise? What about e.g. stopping based on information gain?

a	b	y
0	0	0
0	1	1
1	0	1
1	1	0

(xor function)

$$H(Y|A) = 2 \left(-\frac{1}{2} \left(\frac{1}{2} \log \frac{1}{2} \times 2 \right) \right) = 1$$
$$H(Y|B) = \quad \quad \quad "$$
$$H(Y) = -2 \left(\frac{1}{2} \log \frac{1}{2} \right) = 1$$

$$\hookrightarrow H(Y) - H(Y|A) = 0 \quad (\text{why?})$$

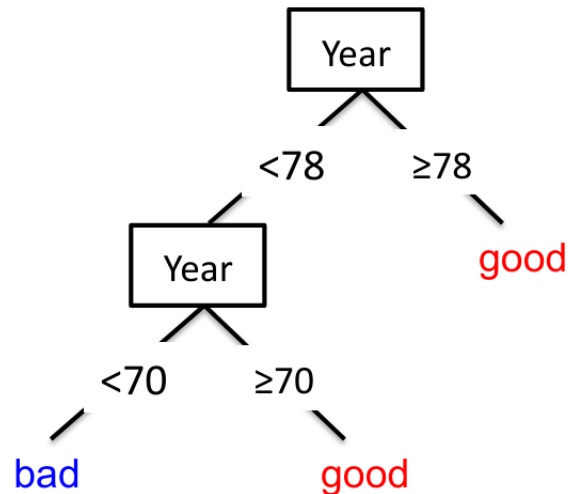
Decision trees

- Instead we'll generally fit a full tree
- Such a model will always get zero error on the training set (which is to say, it will overfit!)
- So, we need some strategy for selecting “simple” trees, e.g. a fixed depth, or a minimum number of samples per leaf

Decision trees

Much more to say about decision trees for those interested (though not so relevant to this class!):

- E.g. fitting based on a threshold (i.e., dealing with real-valued attributes)
- Random forest classifiers
- Can also be used for regression! (**note:** cannot actually fit a linear function!)



Decision trees

Is a decision tree “more interpretable” than other types of classifier?

- **Yes:** following a path in a decision tree tells us exactly why an instance was classified in a certain way
- **Yes:** decisions can be “decomposed” into multiple steps
- **No** (or, at least not more than any other classifier...): a small change in even a single feature value may mean traversing a different part of the tree with very different characteristics
- **Is it more “human like” than a linear model?**

(put differently: this is an example of a classifier with “built in” explainability, though doesn’t exhibit the property that similar instances will be treated similarly)

Decision trees: food for thought

Does it have other desirable characteristics?

- Is it more “**human like**” than a linear model? If a doctor diagnoses a condition, is their underlying thinking *actually* similar to a decision tree? Or is it more like a linear (additive) model? Or neither?
- Does it give a user a means of “**recourse**” (see later)? E.g. if a user has their loan denied, they might be able to see that this decision was based on an income threshold, and know what income they’d need for the loan to be approved; is this useful?

Nearest neighbors

Nearest neighbor classification:

classify a point according to:

$$f(x) = Y_{\text{argmin}_{x'}} d(\phi(x), \phi(x'))$$

dist. between features

Issues:

- Is a single neighbor reliable? (see: k nearest neighbors)
- Is each feature equally important? (see: weighted nearest neighbors)

The image displays six real estate listings arranged in a 3x2 grid. Each listing features a photograph of a townhouse interior or exterior, a price tag, and a brief description of the property. The listings are as follows:

- Top Left:** \$929,000. 3 bds | 3 ba | 1,380 sqft - Townhouse for sale. 3821 Camino Lindo, San Diego, CA 92122. KELLER WILLIAMS REALTY. Open: Sat. 1-4pm.
- Top Right:** \$1,350,000. 3 bds | 3 ba | 1,721 sqft - Townhouse for sale. 4083 Porte De Palmas UNIT 107, San Diego, CA... TEAM Z REALTY. Open: Sat. 12-3pm.
- Middle Left:** \$849,000. 3 bds | 2 ba | 1,053 sqft - Townhouse for sale. 4924 Via Lapiz, San Diego, CA 92122. BIG BLOCK PLATINUM. 30 days on Zillow.
- Middle Right:** \$1,249,000. 3 bds | 3 ba | 1,500 sqft - Townhouse for sale. 3252 Caminito Ameca, La Jolla, CA 92037. PRIME BROKER. 4 days on Zillow.
- Bottom Left:** \$789,000. 3 bds | 2 ba | 1,273 sqft - Townhouse for sale. 5754 Ferber St #125, San Diego, CA 92122. REGENT PROPERTIES OF SAN DIEGO. 23 hours ago.
- Bottom Right:** \$1,088,000. 3 bds | 3 ba | 1,342 sqft - Townhouse for sale. 7952 Playmor Ter, San Diego, CA 92122. SHEIVA POUSTI. 30 days on Zillow.

example: "comps" in real estate

Nearest neighbors

Is a nearest neighbor classifier “more interpretable” than other types of classifier?

- **Yes:** unlike regression or a decision tree, it provides “real-world” exemplars as evidence for a decision.
- **No:** provides no attribution of individual features to its decisions

Support Vector Machines

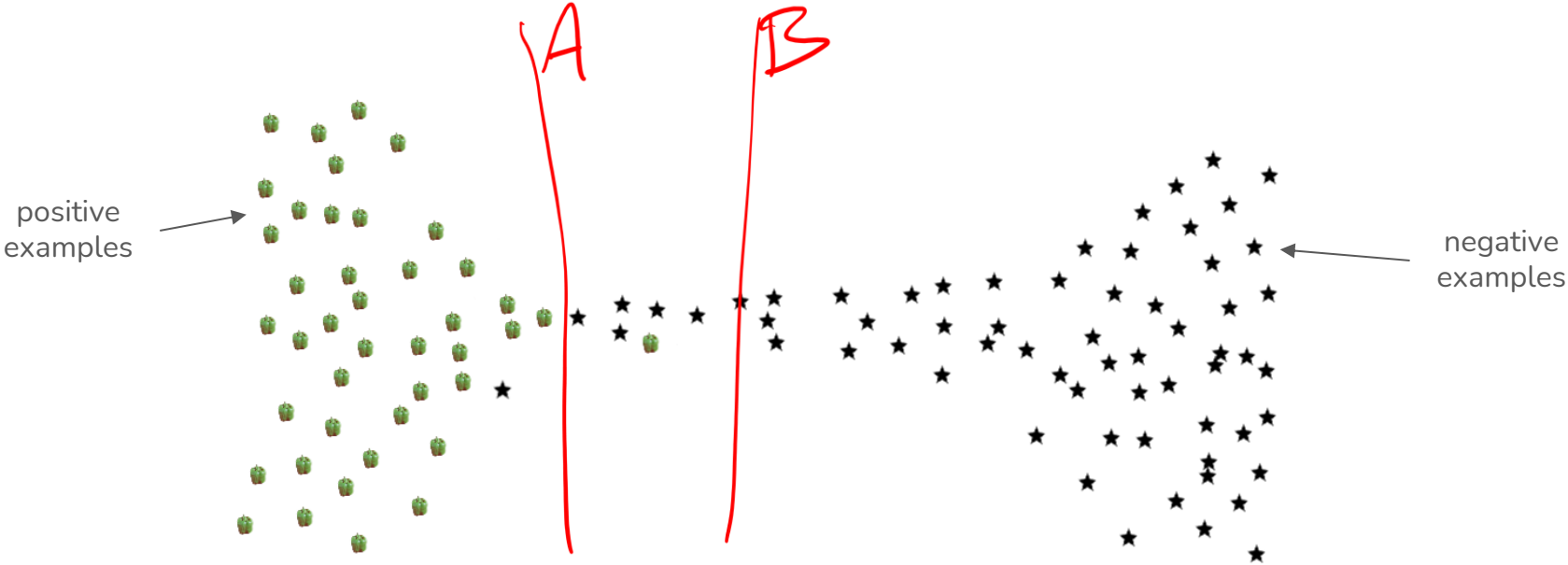
When we studied logistic regression, we fit a classification model of the form:

$$y_i = \begin{cases} 1 & \text{if } X_i \cdot \theta > 0 \\ 0 & \text{otherwise} \end{cases}$$

Like linear regression, the specific value of theta we fit depends on certain **modeling assumptions**: different assumptions would lead to different models!

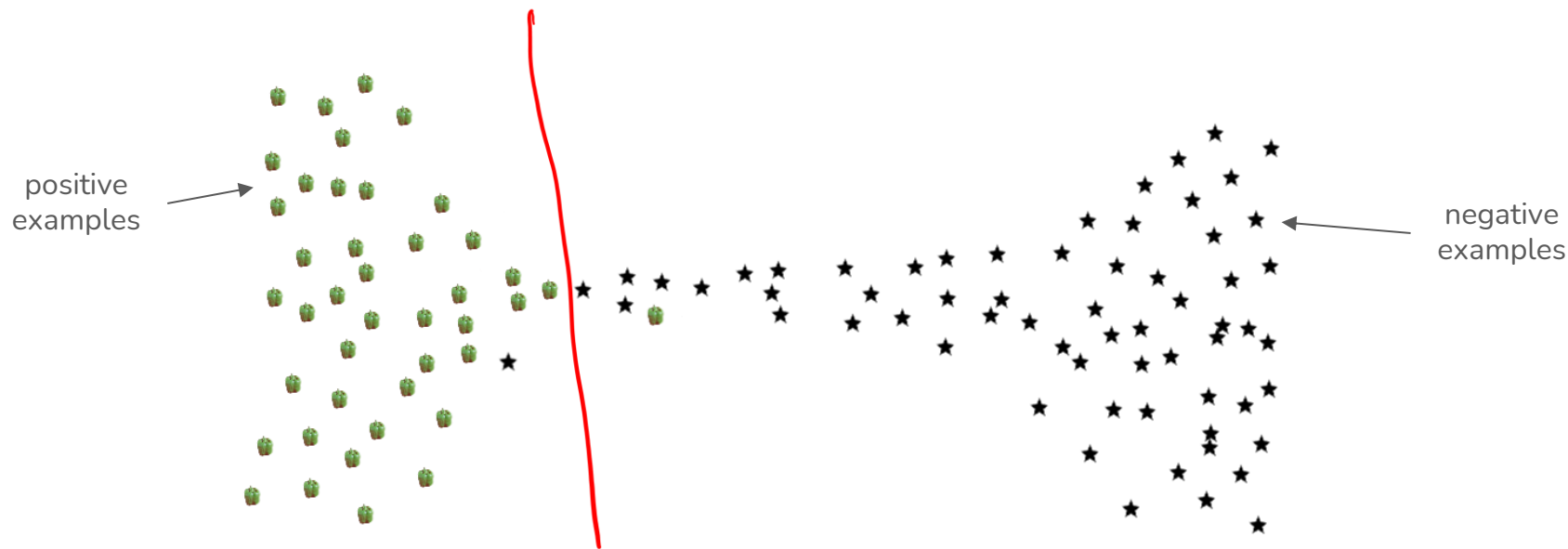
Support Vector Machines

E.g. where would logistic regression place the decision boundary for this dataset?



Support Vector Machines

Where **should** it place the decision boundary?



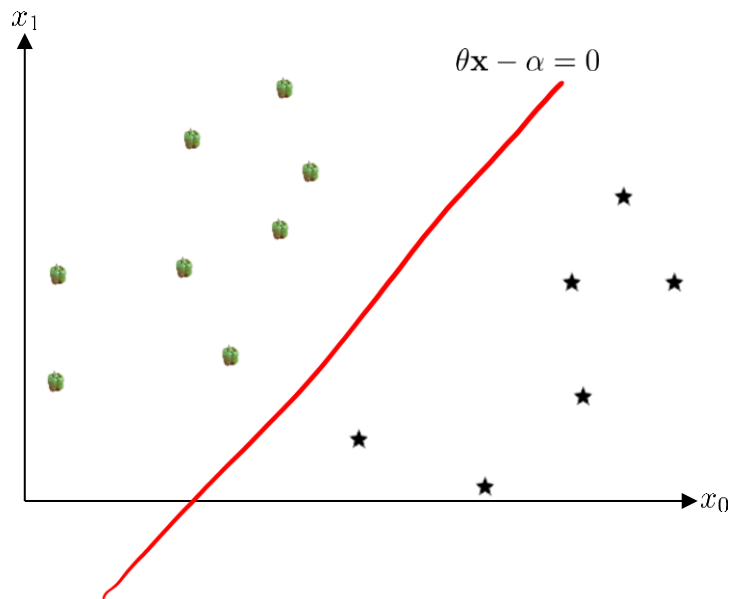
Support Vector Machines

The dataset is:

- Well-separated (for the most part)
- Has only a few ambiguous points in the middle
- Changing the predictions for those points in the middle means moving the decision boundary a lot
- **So:** we might want a decision function that specifically focuses on the “difficulty to classify” examples (which logistic regression does not)
- **SVMs:** directly optimize the **number of misclassifications**

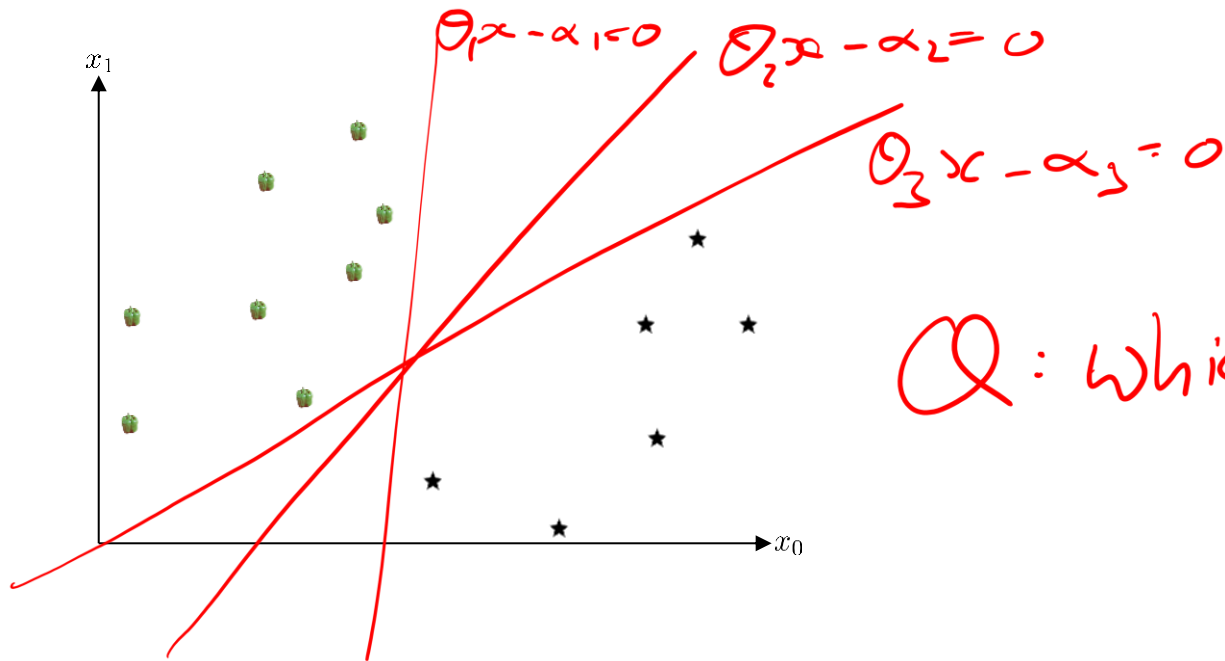
Support Vector Machines

A classifier can be defined by a hyperplane (line): $\theta \mathbf{x} - \alpha = 0$



Support Vector Machines

A classifier can be defined by a hyperplane (line): $\theta \mathbf{x} - \alpha = 0$



Q: Which is best?

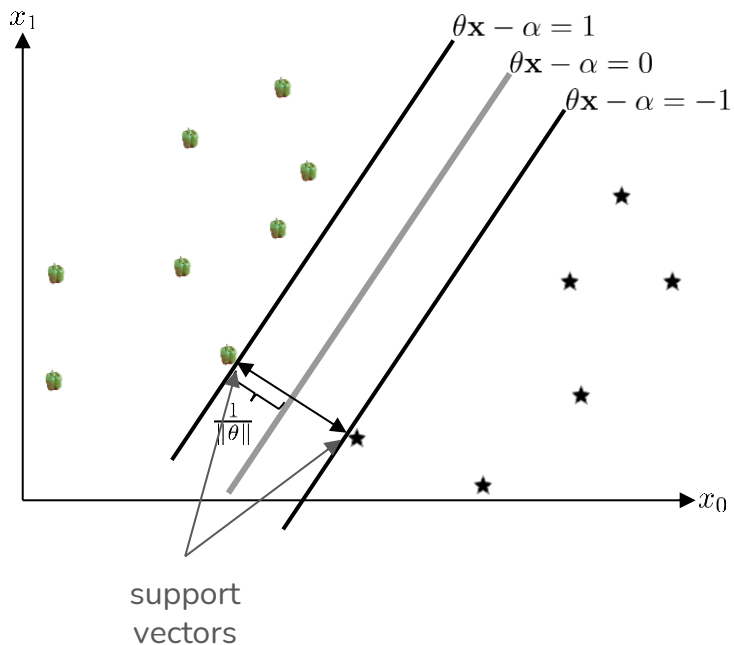
Support Vector Machines

Not all lines are equally good



A: choose the one
that maximizes
distance to
the nearest point

Support Vector Machines

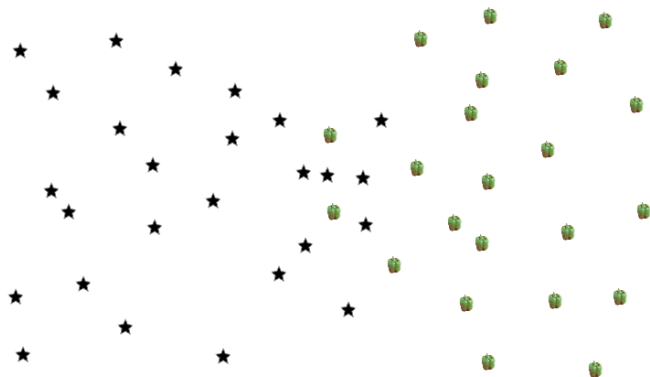


- An SVM seeks the classifier (in this case a line) that is **furthest from the nearest points**
- This can be written in terms of a specific optimization problem:

$$\begin{aligned} & \arg \min_{\theta, \alpha} \frac{1}{2} \|\theta\|_2^2 \\ & \text{such that} \\ & \forall_i y_i (\theta \cdot X_i - \alpha) \geq 1 \end{aligned}$$

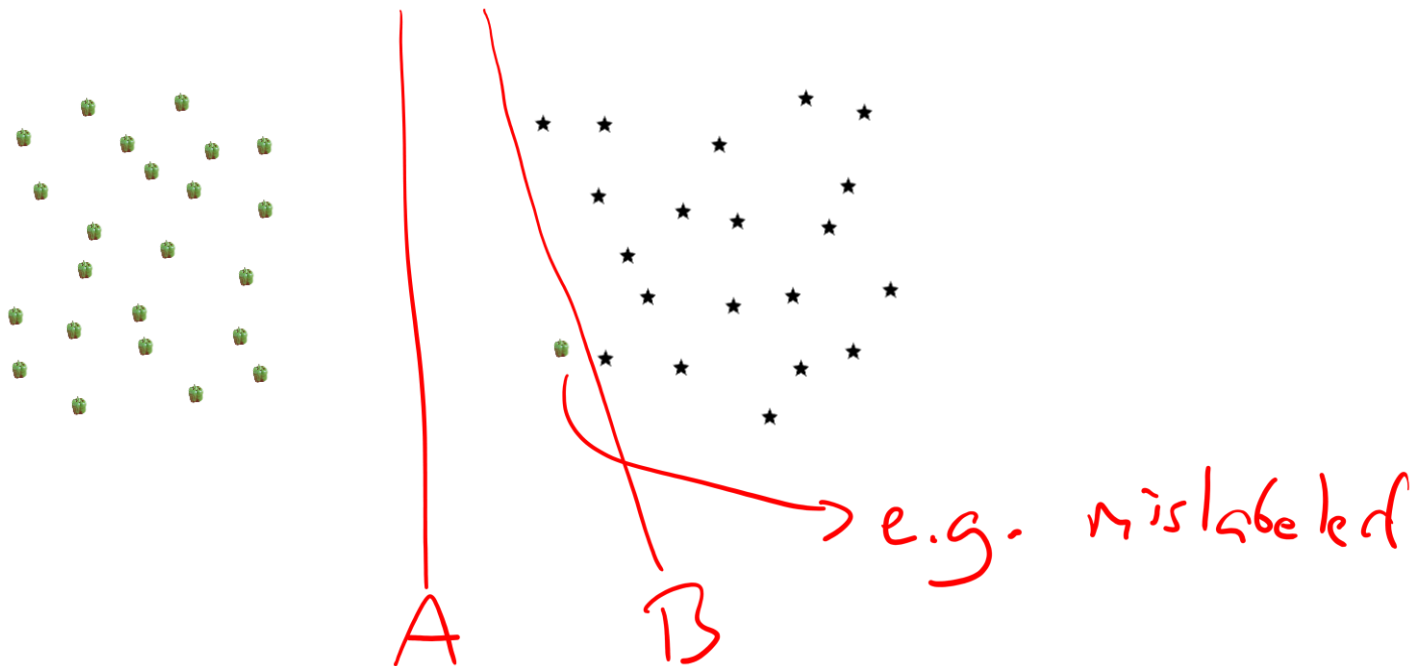
Support Vector Machines

But: is finding such a separating hyperplane even possible?



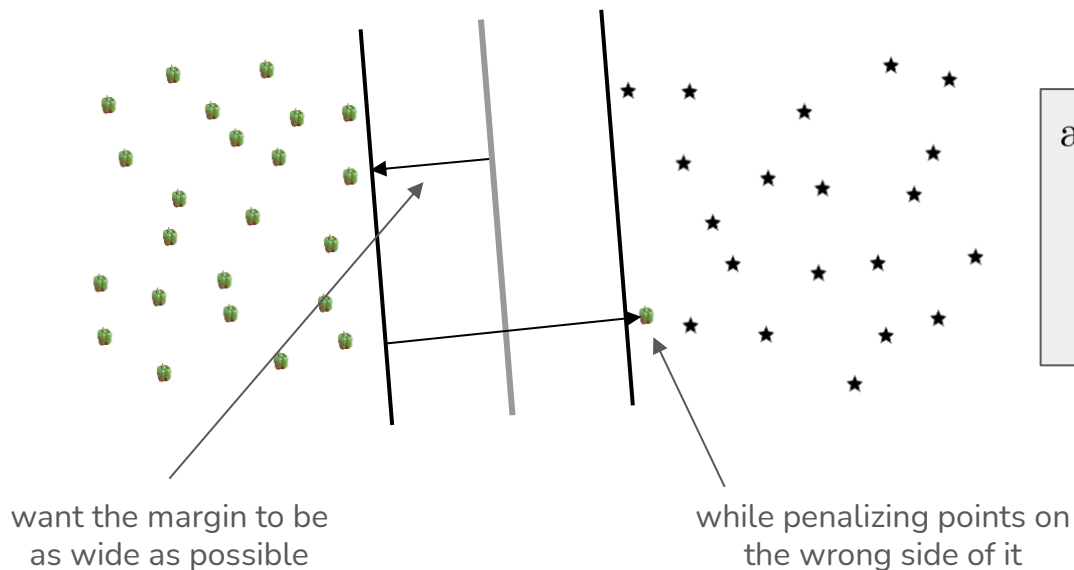
Support Vector Machines

Or: is it actually a good idea?



Support Vector Machines

“Soft-margin” formulation:



$$\arg \min_{\theta, \alpha, \xi > 0} \frac{1}{2} \|\theta\|_2^2 + C \sum_i \xi_i$$

such that

$$\forall_i y_i (\theta \cdot X_i - \alpha) \geq 1 - \xi_i$$

Support Vector Machines

- SVMs seek to find a hyperplane (in two dimensions, a line) that optimally separates two classes of points
- The “best” classifier is the one that classifies all points correctly, such that the nearest points are **as far as possible** from the boundary
- If not all points can be correctly classified, a penalty is incurred that is proportional to **how badly the points are misclassified** (i.e., their distance from this hyperplane)

Support Vector Machines

Is an SVM “more interpretable” than other types of classifier?

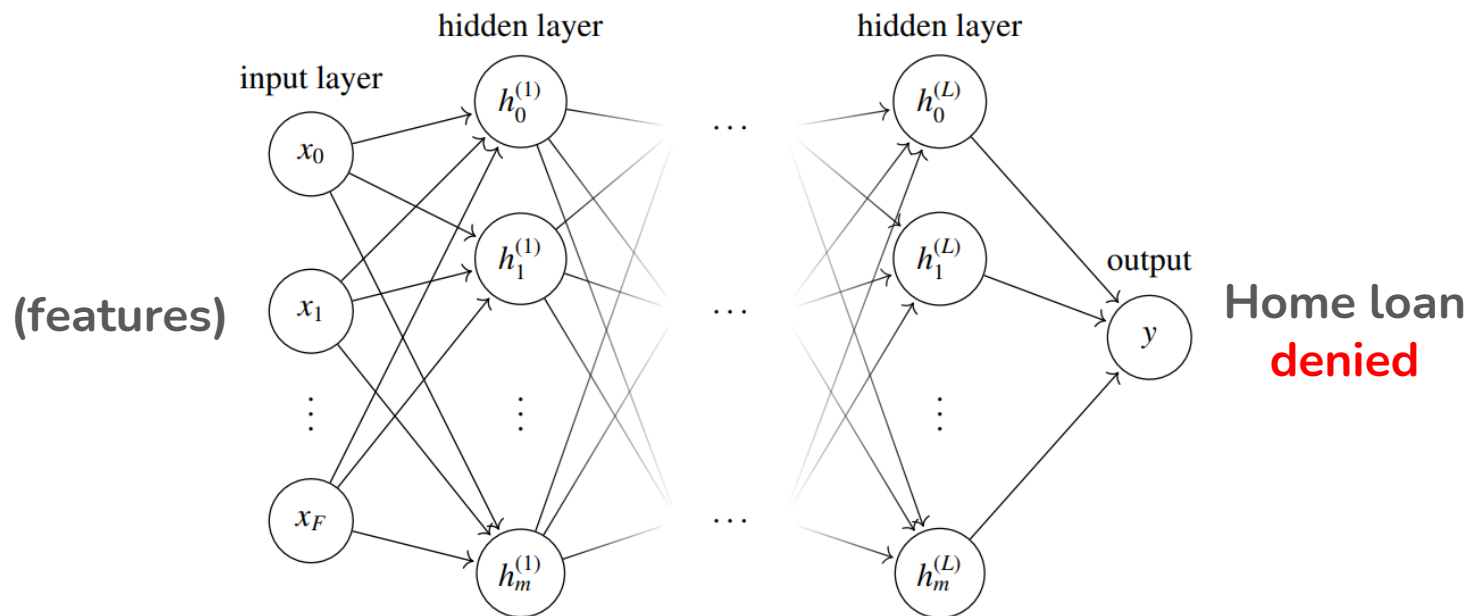
(probably not?)

It is **fairer** than logistic regression?

- **Yes:** optimizes the number of errors, so will tend to make more reasonable predictions for the most marginal cases
- **No:** “confidence” score of an SVM (distance from boundary) doesn’t correspond to a particularly meaningful concept, compared to a probability score

Multilayer perceptron

The methods we've seen so far have **interpretability** at the cost of **complexity**.
But in practice classifiers may be more complex **black box** models:



Multilayer perceptron

Multilayer perceptrons (MLPs) are a staple of neural networks, offering a way to learn non-linear transformations and interactions among features.

Roughly, a 'layer' of an MLP transforms a vector of inputs to a (possibly lower dimensional) vector of outputs; typically, outputs are related to inputs via a linear transformation followed by a non-linear activation, e.g.:

$$f(x) = \sigma(Mx)$$

Here x is a vector of input variables, $f(x)$ is a vector of output variables, and M is a learned matrix, such that each term in Mx is a weighted combination of the original features in x .

Multilayer perceptron

While the above is just one *layer* of a multilayer perceptron, several such layers can be 'stacked' in order for the network to learn complex non-linear functions.

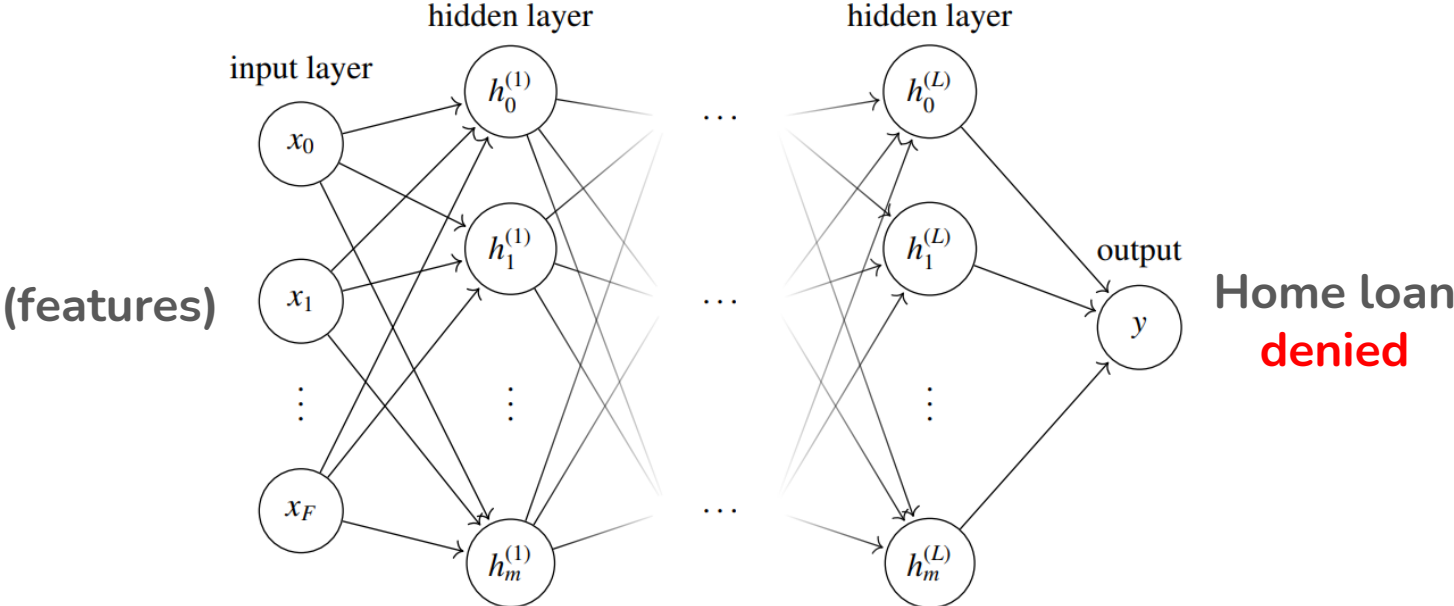
Eventually, the final layer predicts some desired output, e.g. a regression or classification objective. E.g. the final layer might take a weighted combination of features from the previous layer:

$$f(x) = \sigma(\theta \cdot x)$$

i.e., similar to the output of a logistic regressor (for a classification task).

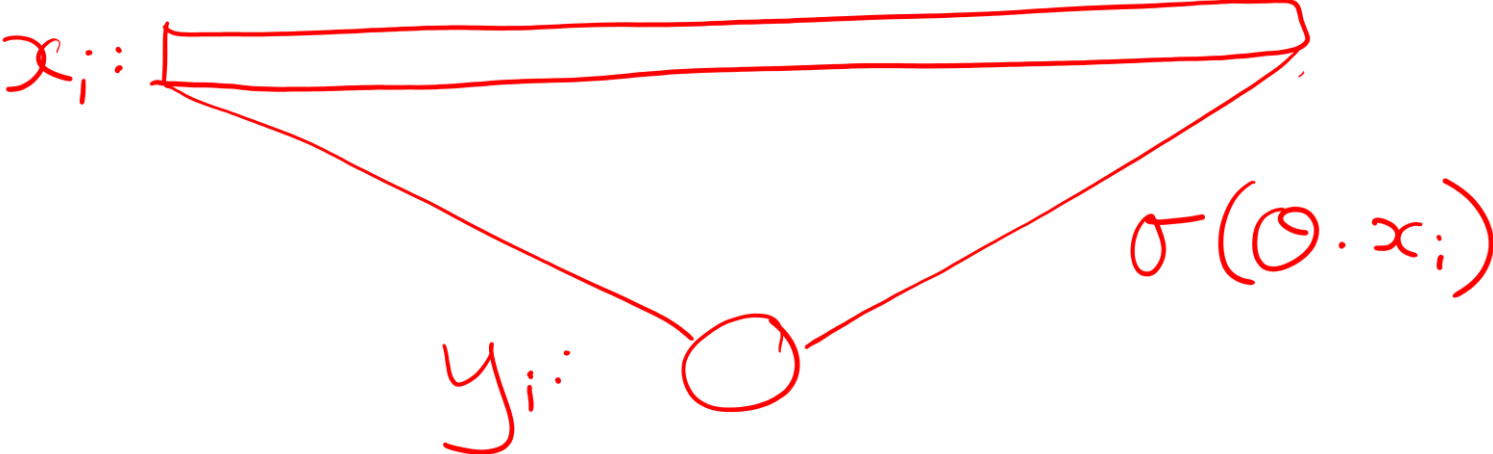
Multilayer perceptron

The multilayer perceptron might be visualized as follows:



Multilayer perceptron

Q: How would our linear model look if we tried to visualize it this way?



Multilayer perceptron

Compared to a decision tree or a linear model, a multilayer perceptron (or other “deep network”) might seem *less interpretable*

What criteria might support (or refute) that categorization?

Study points & take-homes

- For the purpose of this class, no need to study any of the above classifiers in detail
- Instead, think generally about:
 - Different assumptions lead to different classifiers (and therefore, different decisions) given the same data; thus, even our choice of classification function may be a source of **bias or unfairness**;
 - How would you justify the choice of one classifier over another in the event that they led to consequentially different decisions for some subgroup of people?
 - By what criteria would you call any of these classifiers “more **interpretable**” than others?

Regression and classification

1.7: Classifier evaluation

Some definitions – mathematical notation

- Label:

y_i

- Prediction:

\hat{y}_i

$$= \begin{cases} 1 & \text{if } \theta \cdot x_i > 0 \\ 0 & \text{otherwise} \end{cases}$$

(or whatever)

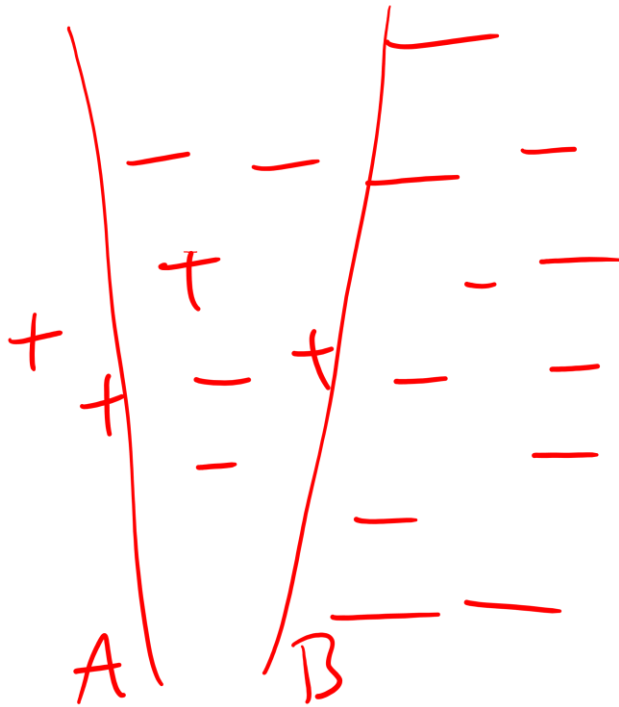
Evaluating classification models

Accuracy: $\frac{1}{N} \sum_i \delta(y_i = \hat{y}_i)$

Error: $1 - \text{acc}$

Evaluating classification models

Example: accuracy and error are not enough



A: 2 errors

B: 4 errors

True and false positives (and negatives)

- TP labeled as T, predicted as T
 $= \sum_i (y_i \wedge \hat{y}_i)$
- FP $= \sum_i (\neg y_i \wedge \hat{y}_i)$
- TN $= \sum_i (\neg y_i \wedge \neg \hat{y}_i)$
- FN $= \sum_i (y_i \wedge \neg \hat{y}_i)$

True and false positives (and negatives) – rates

- TPR

$$TP / \# \text{positives} = \frac{TP}{(TP + FN)}$$

- FPR

$$= FP / (TP + FN)$$

- TNR

$$TN / \# \text{negative} = \frac{TN}{(TN + FP)}$$

- FNR

$$= FN / (TN + FP)$$

Balanced error rate

$$\begin{aligned} \text{BER} &= \frac{1}{2} (\text{FPR} + \text{FNR}) \\ &= 1 - \frac{1}{2} (\text{TPR} + \text{TNR}) \end{aligned}$$

Optimizing the balanced error rate

$$l_{\theta}(y|X) = \sum_{y_i=1} \log \sigma(x_i \cdot \theta) + \sum_{y_i=0} \log (1 - \sigma(x_i \cdot \theta))$$

↙

$$\frac{1}{|y_i=1|} \sum_{y_i=1} \log \sigma(x_i \cdot \theta) + \frac{1}{|y_i=0|} \sum_{y_i=0} \log (1 - \sigma(x_i \cdot \theta))$$

+ive

TPR (sort of)

Looking ahead: fairness interventions

This was (sort of...) a fairness intervention!

We improved the performance of the classifier with respect to a particular group (individuals with a particular label) by modifying the training objective

More complex fairness interventions that we'll see later will generally have the same flavor (though will be more complex...)

Looking ahead: fairness interventions

This is (roughly speaking) what we'll later call an “in-processing” intervention:

- **Pre-processing:** try to get fairer outcomes by modifying the dataset (*before* training)
- **In-processing:** try to get fairer outcomes by modifying the training objective (*during* training)
- **Post-processing:** try to get fairer outcomes by modifying the model's predictions (*after* training, or more simply during *inference*)

Looking ahead: fairness interventions

Note: we could have achieved a similar outcome via **pre-processing:**

- If we have N examples from the positive class and M examples from the negative class (and suppose $N > M$)
- Build a new dataset that includes all samples from the larger (in this case, positive) class, and randomly samples N instances from the smaller class (i.e., so that some instances will be repeated)
- **This would still optimize a balanced error rate**

Looking ahead: fairness interventions

Exercise: what might a **post-processing** intervention look like?

$$\text{Classifier: } f_{\theta}(x) = \begin{cases} 1 & \text{if } \theta \cdot x > 0 \\ 0 & \text{otherwise} \end{cases}$$

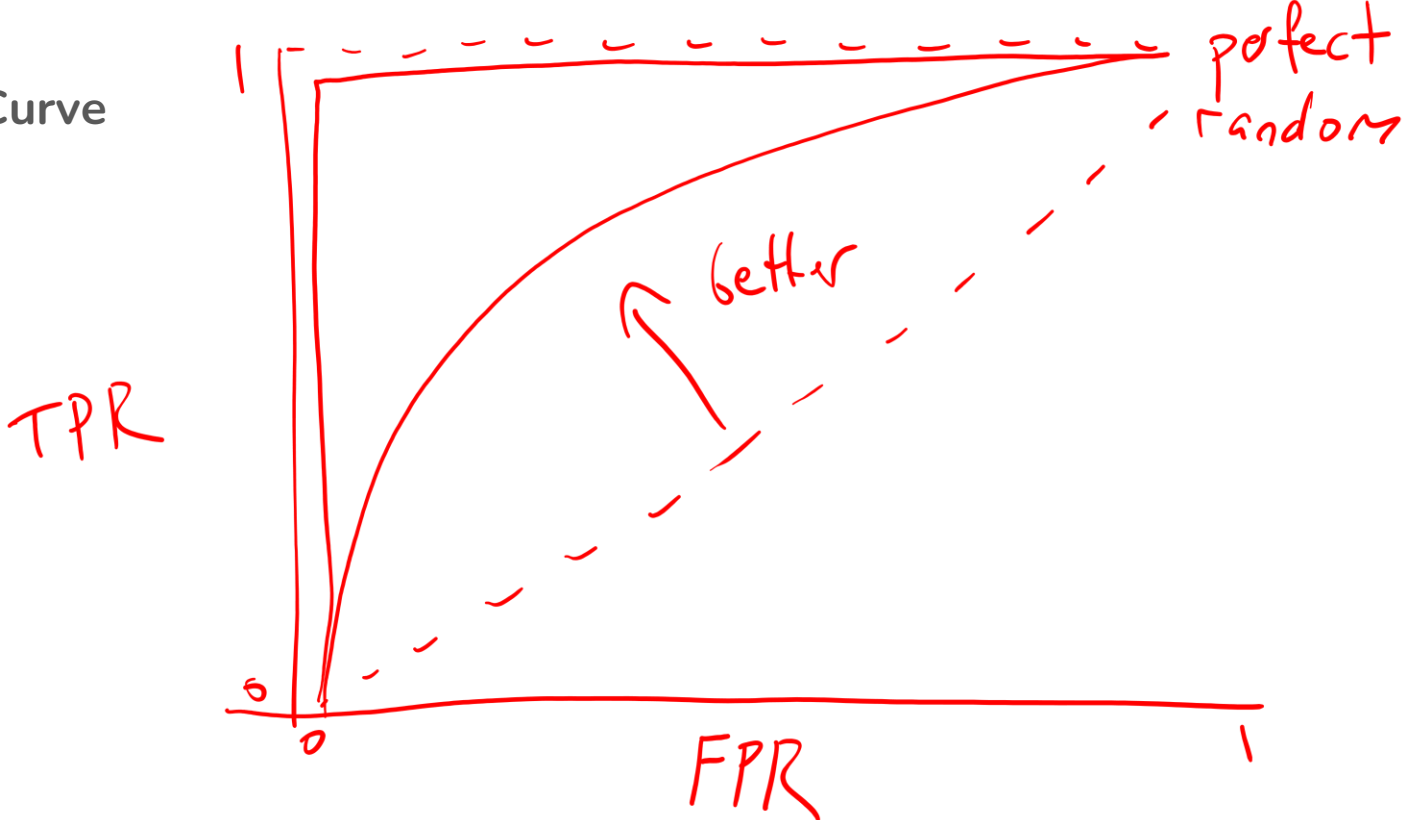
$$\theta \cdot x > 0 \implies \theta \cdot x > \alpha$$

higher α : lower FPR, higher FNR

lower α : higher FPR, lower FNR

Other metrics

ROC Curve



see e.g. <https://afraenkel.github.io/fairness-book/content/07-score-functions.html>

Other metrics

Precision / recall / F1 / etc.

— See textbook

Looking ahead: fairness interventions

(we'll come back to this later, and look at all sorts of interventions for different settings and their relative advantages / disadvantages)

Other metrics

Precision / recall / F1 / etc.

These don't show up much in this class, so I won't spend time covering them.

For self study see https://cseweb.ucsd.edu/~jmcauley/pml/pml_book.pdf Section 3.3.3

What's to come?

We'll look at *lots* of metrics in the upcoming modules. Generally speaking, we'll want to compute metrics with respect to particular *groups* (e.g. female versus not female) that exhibit a particular feature:

e.g. TPR high overall,
but low for females

Example

Let's build a **synthetic** datasets to explore error rates, and in particular explore error rates **by group** (gender, in this example)

These type of “per group” metrics will form the basis of most of our fairness metrics and interventions in the next module

Example – synthetic data generation

```
for sample in range(1000):
    female = 1
    if random.random() > 0.5: female = 0
    qualified = 0
    years_of_exp = random.random() * 10 # 0 to 10 years of
experience
    if female:
        if random.random() > 0.6 + years_of_exp*0.05: qualified = 1
    else:
        if random.random() > 0.5 - years_of_exp*0.05: qualified = 1
    X.append([1, years_of_exp, female])
    y.append(qualified)
```

(synthetic) bias!

Example

(walk through code example from workbook1)

Example

Need to address two issues when training a model:

- How do classifiers reproduce (or amplify!) biases from data?
- How can biases arise as a function of different features (e.g. if female applicants generally have more/less experience than males), or as a function of differences in group sizes (e.g. if 90% of applicants are male)?

Study points & take-homes

- Understand the difference between error types (FP vs FN)
- Understand how to express errors and evaluate classifiers in terms of rates (TPR, TNR, etc.)
- Understand the rationale (and ideally, the definition) behind different error metrics, including BER, AUC

Regression and classification

1.8: The learning pipeline

This section

- Introduce generalization, overfitting, and underfitting
- Introduce the concept of model “complexity” and regularization
- Introduce validation sets and the idea of a machine learning “pipeline”
- Discuss the difference between the l_1 and l_2 norms
- Describe general guidelines implementing a model pipeline

Generalization, overfitting and underfitting

So far, when evaluating models, we considered training a model to predict labels \mathbf{y} from a dataset \mathbf{X} ; we then evaluated the model by comparing the predictions $\mathbf{f}(\mathbf{x}_i)$ to the labels \mathbf{y}_i .

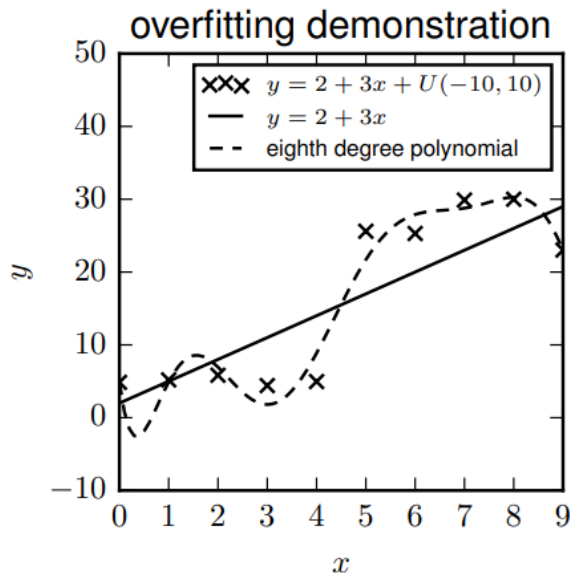
Critically, we're using the *same data* to train the model as we're using to evaluate it.

The risk in doing so is that our model may not generalize well to new data. For example, fitting a dataset with an increasingly high-degree polynomial would continue to lower the errors of the predictor.

Such models could fit the data very closely, but it is unclear whether they would capture meaningful trends in the data.

Generalization, overfitting and underfitting

E.g. which function fits the data better?



- Should I use more dims. in my embedding?
- How large should the dictionary be in my language model?
- etc.

Generalization, overfitting and underfitting

Need to address two issues when training a model:

1. We should not evaluate a model on the same data that was used to train it. Rather we should use a held-out dataset (i.e., a test set).
2. Features that improve performance on the training data will not necessarily improve performance on the held-out data.

Evaluating a model on held-out data gives us a sense of how well we can expect that model to work 'in the wild.'

This held-out data, known as a **test set**, measures how well our model can be expected to **generalize** to new data.

Generalization, overfitting and underfitting

Overfitting occurs when our model works well on our training data, but does not generalize well to held-out (test) data

Underfitting occurs when our model is insufficiently complex to capture the underlying dynamics in a dataset (meaning that both its training error and test error are high)

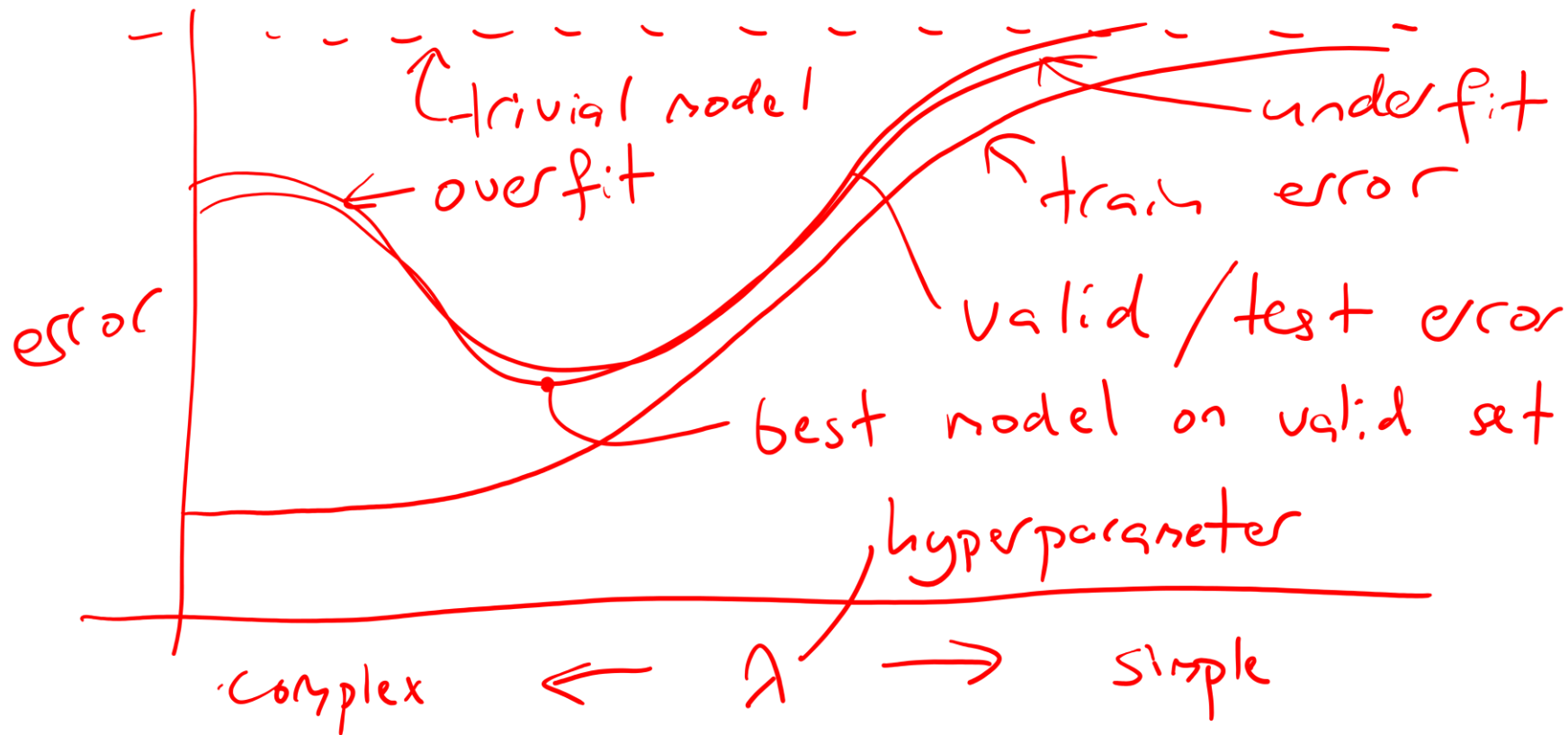
Generalization, overfitting and underfitting

Q: But how can we **detect** overfitting or underfitting?

- (Just about) *all* models will work better on seen data compared unseen data, so *having worse performance on the training set doesn't necessarily mean we've overfit* (i.e., there may be no model that generalizes better)
- Likewise, poor performance on the training set doesn't necessarily mean we've underfit: possibly *no model is capable of fitting the data* (based on the available features)

A: We'll need to try several models of varying complexity to find which one generalizes best

Generalization, overfitting and underfitting



Study points & take-homes

- Try to reproduce the above plot! Concepts like underfitting and overfitting should make intuitive sense, rather than being things you try to memorize
- Understand the functions of **training**, **validation**, and **test sets**
- Be able to implement a **model fitting / regularization pipeline**: if this is totally new to you, I'd suggest going through the textbook (Personalized Machine Learning) to brush up

Regression and classification

Case study: Gender Shades

Goals

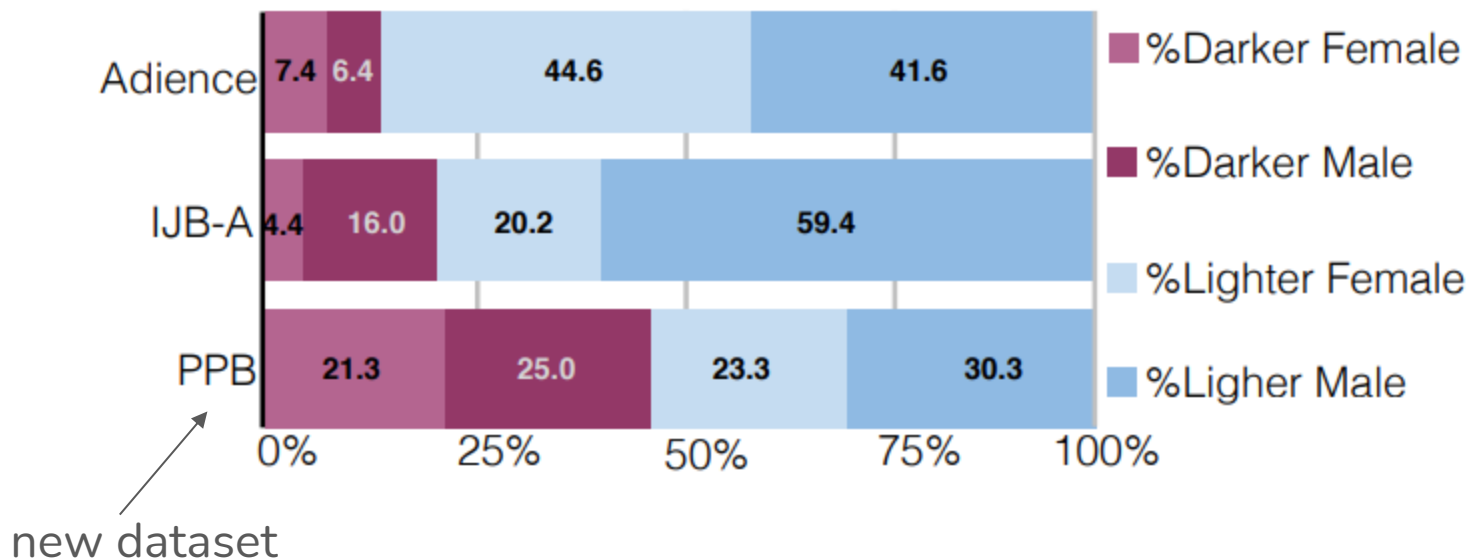
Face detection systems anecdotally seem to work poorly on faces of users from underrepresented groups, either failing to detect a face entirely, or being more likely to misgender the face of a user from an underrepresented group

How well do existing commercial solutions to face/gender classification systems work, and how much do results differ as a function of gender or skin tone?

Classifiers are evaluated from **IBM, Microsoft, and Face++** (Chinese dominant)

Dataset

To assess bias, the authors create a new dataset that is more balanced with respect to characteristics of interest



Dataset




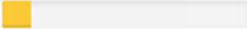







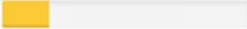
Gender and skin tone are labeled manually, into 6 skin tone classes

Note that these datasets are *small* – as such they are suitable mostly for evaluation but not for model training

Dataset	Lighter (I,II,III)		Darker (IV, V, VI)		Total
PPB	53.6%	681	46.4%	589	1270
IJB-A	79.6%	398	20.4%	102	500
Adience	86.2%	1892	13.8%	302	2194



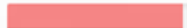















Key findings:

- All systems (significantly!) less accurate for females

Gender Classifier	Darker Subjects Accuracy	Lighter Subjects Accuracy	Error Rate Diff.
 Microsoft	87.1% 	99.3% 	12.2% 
 FACE++	83.5% 	95.3% 	11.8% 
 IBM	77.6% 	96.8% 	19.2% 

Key findings:

- Separating across skin tone, much of the performance degradation is for darker-skinned females

Gender Classifier	Darker Male	Darker Female	Lighter Male	Lighter Female	Largest Gap
 Microsoft	94.0% 	79.2% 	100% 	98.3% 	20.8% 
 FACE++	99.3% 	65.5% 	99.2% 	94.0% 	33.8% 
 IBM	88.0% 	65.3% 	99.7% 	92.9% 	34.4% 

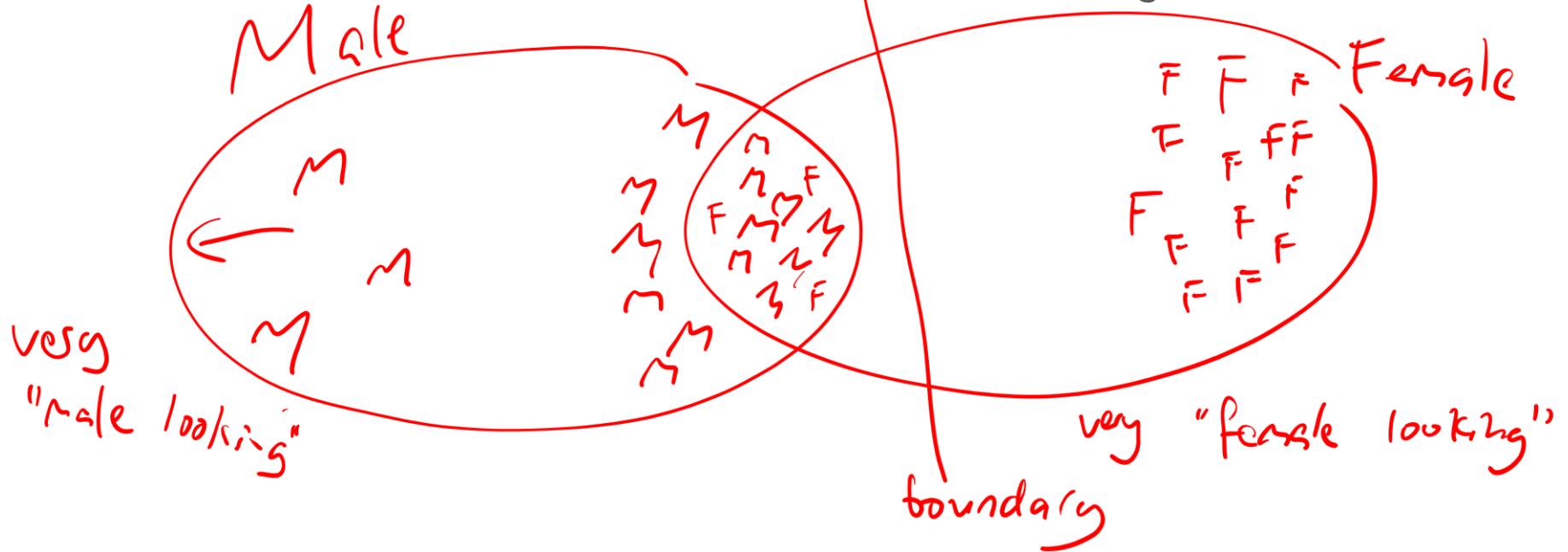
Key findings:

- All classifiers perform better on male faces than female faces (8.1% – 20.6% difference in error rate)
- All classifiers perform better on lighter faces than darker faces (11.8% – 19.2% difference in error rate)
- All classifiers perform worst on darker female faces (20.8% – 34.7% error rate)
- Microsoft and IBM classifiers perform best on lighter male faces (error rates of 0.0% and 0.3% respectively)
- Face++ classifiers perform best on darker male faces (0.7% error rate)
- The maximum difference in error rate between the best and worst classified groups is 34.4%

Key findings:

- 95.9% of the faces misgendered by Face++ were those of female subjects

what does this look like in terms of error categories?



Key findings:

- 95.9% of the faces misgendered by Face++ were those of female subjects

& why should this happen?

- Part of the data is hard to classify
- Most hard to classify samples are M

↳ label every ambiguous case as M

Does this explain everything?

Food for thought

- To what extent are findings aligned with dataset characteristics (i.e., performance outcomes are proportional to representation)?
- The authors wrote the paper to highlight an area where “companies should do better”.
 - What would be a better outcome and ***what would be required to achieve that outcome?***
 - What incentives do companies have to make these algorithms better?

Food for thought

- Face classification is not equivalent to (e.g.) recidivism prediction – mistakes have much lower consequences
- In the case of recidivism prediction, we might make an argument that lowering the overall accuracy of a classifier is “worth it” if it makes the classifier substantially fairer with respect to certain subgroups
- Could we make a similar argument for face classification? What is a fair trade in terms of the number of mistakes we would tolerate for male users in order to increase the accuracy for female users?

References for Module 1

- Fairness & Algorithmic Decision Making: <https://afraenkel.github.io/fairness-book/>
- A Survey on Bias and Fairness in Machine Learning: <https://dl.acm.org/doi/pdf/10.1145/3457607>
- Fairness in Machine Learning: <https://arxiv.org/pdf/2010.04053>
- Gender Shades: Intersectional Accuracy Disparities in Commercial Gender Classification. Buolamwini and Gebru, 2018
- Introduction to Machine Learning: <https://people.csail.mit.edu/dsontag/courses/ml16/>