# ML – Overview of Foundations

IS 365

# Bias in Machine Learning

- A more subtle case is the issue of bias.
- One might naively think that since machine learning algorithms are based on mathematical principles, they are somehow objective.
- However, machine learning predictions come from the training data, and the training data comes from society, so any biases in society are reflected in the data and propagated to predictions.
- The issue of bias is a real concern when machine learning is used to decide whether an individual should receive a loan or get a job etc.
- Unfortunately, the problem of fairness and bias is as much of a philosophical one as it is a technical one.

# How do we solve AI tasks?

- How should we actually solve AI tasks?
- The real world is complicated.
- At the end of the day, we need to write some code (and possibly build some hardware, too).
- But there is a huge gap.

# Paradigm

- In this course, we will adopt the **modeling-inference-learning** paradigm to help us navigate the solution space.
- In reality the lines are blurry, but this paradigm serves as an ideal and a useful guiding principle.
- The first pillar is modeling.
- Modeling takes messy real world problems and packages them into neat formal mathematical objects called models, which can be subject to rigorous analysis and can be operated on by computers.

- However, modeling is lossy: not all of the richness of the real world can be captured, and therefore there is an art of modeling:

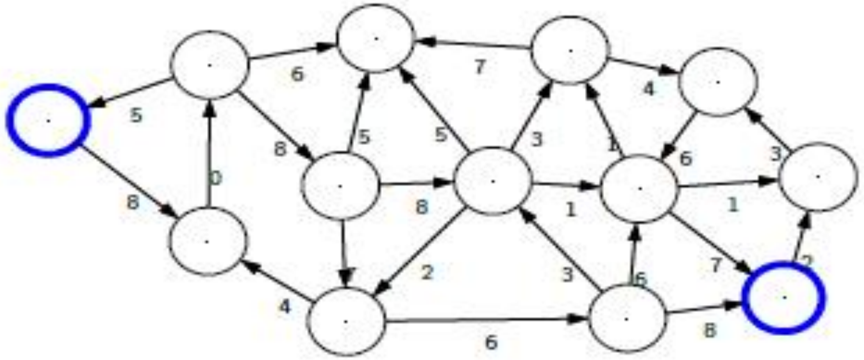
what does one keep versus ignore?

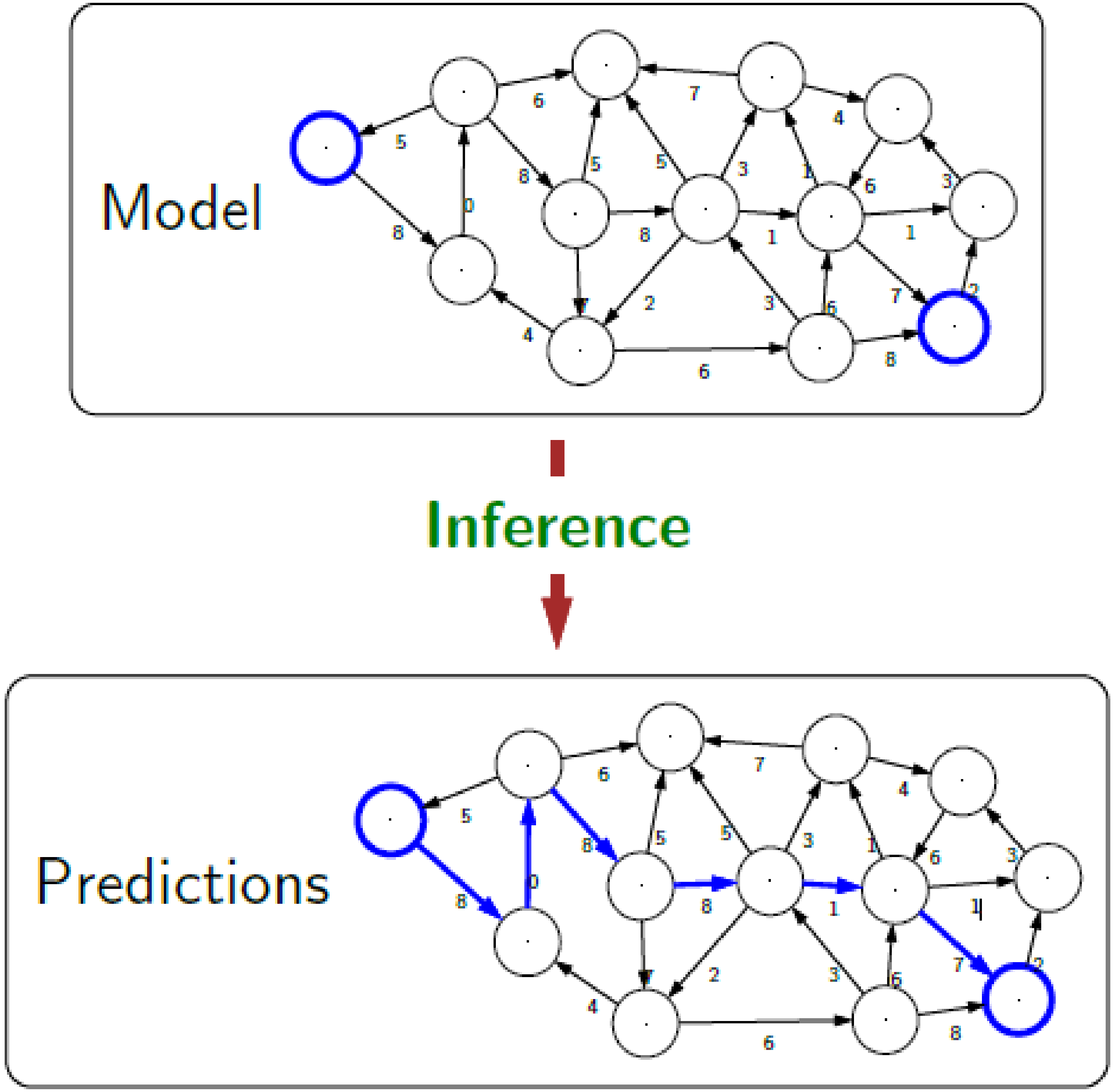
- As an example, suppose we're trying to have an AI that can navigate through a busy city.
- We might formulate this as a graph where nodes represent points in the city, edges represent the roads, and the cost of an edge represents the traffic on that road.

Real world

**Modeling**

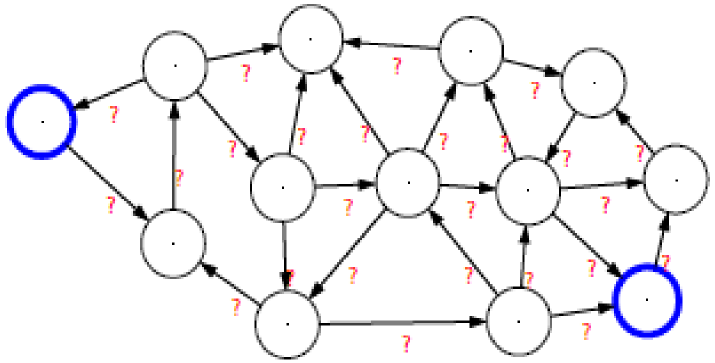
Model




Model
Inference
Predictions

- The second pillar is inference.
- Given a model, the task of inference is to answer questions with respect to the model.
- For example, given the model of the city, one could ask questions such as: what is the shortest path?
- What is the cheapest path?
- The focus of inference is usually on efficient algorithms that can answer these questions.
- For some models, computational complexity can be a concern, and usually approximations are needed.

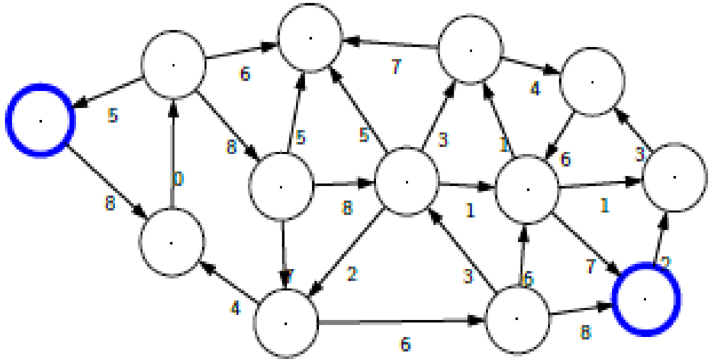
Model without parameters



+data

**Learning**

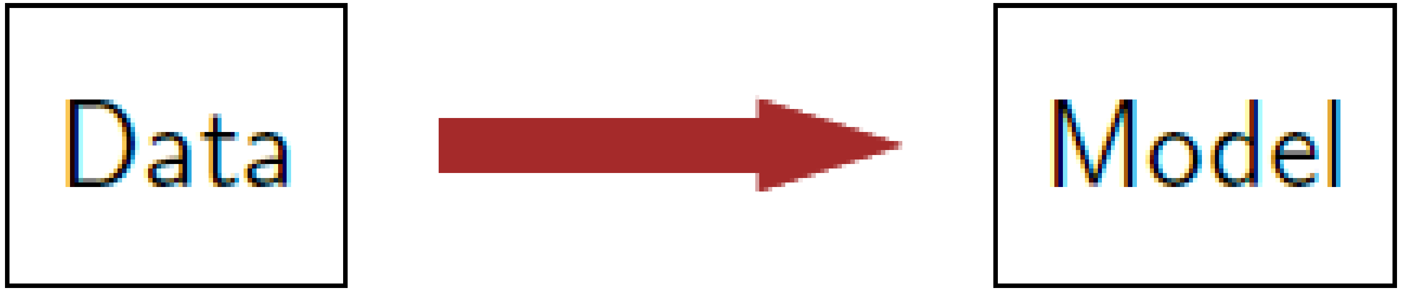
Model with parameters



- But where does the model come from?
- Remember that the real world is rich, so if the model is to be faithful, the model has to be rich as well.
- But we can't possibly write down such a rich model manually.
- The idea behind (machine) learning is to instead get it from data.
- Instead of constructing a model, one constructs a skeleton of a model (more precisely, a model family), which is a model without parameters.

- And then if we have the right type of data, we can run a machine learning algorithm to tune the parameters of the model.
- Note that learning here is not tied to a particular approach (e.g., neural networks), but more of a philosophy.
- This general paradigm will allow us to bridge the gap between logic-based AI and statistical AI.

# Machine Learning




- The main driver of recent successes in AI
- Move from "code" to "data" to manage the information complexity
- Requires a leap of faith: generalization

- Supporting all of these models is **machine learning**, which has been arguably the most crucial ingredient powering recent successes in AI.
- From a systems engineering perspective, machine learning allows us to shift the information complexity of the model from code to data, which is much easier to obtain.
- The main conceptually magical part of learning is that if done properly, the trained model will be able to produce good predictions beyond the set of training examples.
- This leap of faith is called generalization, and is, explicitly or implicitly, at the heart of any machine learning algorithm. This can even be formalized using tools from probability and statistical learning theory.

# Optimization

- We will approach inference and learning from an optimization perspective, which allows us to decouple the mathematical specification of what we want to compute from the algorithms for how to compute it.
- In total generality, optimization problems ask that you find the $x$ that lives in a constraint set $C$ that makes the function $F(x)$ as small as possible.
- There are two types of optimization problems: discrete optimization problems (mostly for inference) and continuous optimization problems (mostly for learning).
- For this course, we will learn: gradient descent.
- For now, we are assuming that the model (optimization problem) is given and only focus on algorithms.

# Example Problem: finding the least squares line

- **Input**: $set\ of\ pairs\ \{(x_1, y_1), \ldots, (x_n, y_n)\}$
- **Output**: $w \epsilon \mathbb{R}\ \ that\ minimizes\ the\ squared\ error$

$$F(w) = \sum_{i=1}^{n} (x_i w - y_i)^2$$

- Examples:

$\{(2, 4)\} \implies 2$

$\{(2, 4), (4, 2)\} \implies ?$

- **The formal task is this**: given a set of $n$ two-dimensional points $(x_i, y_i)$ which defines $F(w)$, compute the $w$ that minimizes $F(w)$.
- **Linear regression** is an important problem in machine learning.
- Here's a motivation for the problem: suppose you're trying to understand how your exam score $(y)$ depends on the number of hours you study $(x)$.
- Let's posit a linear relationship $y = wx$ (not exactly true in practice, but maybe good enough).
- Now we get a set of training examples, each of which is a $(x_i, y_i)$ pair.
- The goal is to find the slope $w$ that best fits the data.

- Back to algorithms for this formal task. We would like an algorithm for optimizing general types of $F(w)$.
- So let's abstract away from the details.
- Start at a guess of $w$ $(say\, w = 0)$, and then iteratively update $w$ based on the derivative (gradient if $w$ is a vector) of $F(w)$.
- The algorithm we will use is called **gradient descent**.
- If the derivative $F'(w) < 0$, then increase $w$; if $F'(w) > 0$, decrease $w$; otherwise, keep $w$ still.

- This motivates the following update rule, which we perform over and over again:

$w \leftarrow w - \alpha\, F'(w)$, where $\alpha > 0$ is a step size that controls how aggressively we change $w$.

- If $\alpha$ is too big, then $w$ might bounce around and not converge. If $\alpha$ is too small, then w might not move very far to the optimum.
- Choosing the right value of $\alpha$ can be rather tricky.
- Empirically, we will just try a few values and see which one works best.
- This will help develop some intuition in the process.

# Exercise:

- Now to specialize to our function, we just need to compute the derivative, which is an elementary calculus exercise:

Find the derivative of $F(w) = \sum_{i=1}^{n}(x_i w - y_i)^2$ with respect to $w$

# Machine Learning - Roadmap

- Linear Predictors
- Loss Minimization
- Gradient Descent => Stochastic Gradient Descent
- Linear Predictors => cover both classification and regression
- 1st => geometric intuition for linear predictors, then learning the weights of a linear predictor by formulating an optimization problem based on the loss minimization framework.
- Finally => SGD, an efficient algorithm for optimizing (i.e. minimizing) the loss that's tailored for ML which is much faster than GD.

# Linear Predictors

- Input: $x = email\ message$
- Output: $y \epsilon \{spam, not\ spam\}$
- Objective: $obtain\ a\ predictor\ f$
- $x \rightarrow f \rightarrow y$
- A predictor is a function $f$ that maps an input $x$ to an output $y$.
- In statistics, $y$ is known as a response, and when $x$ is a real vector, it is known as the covariate.

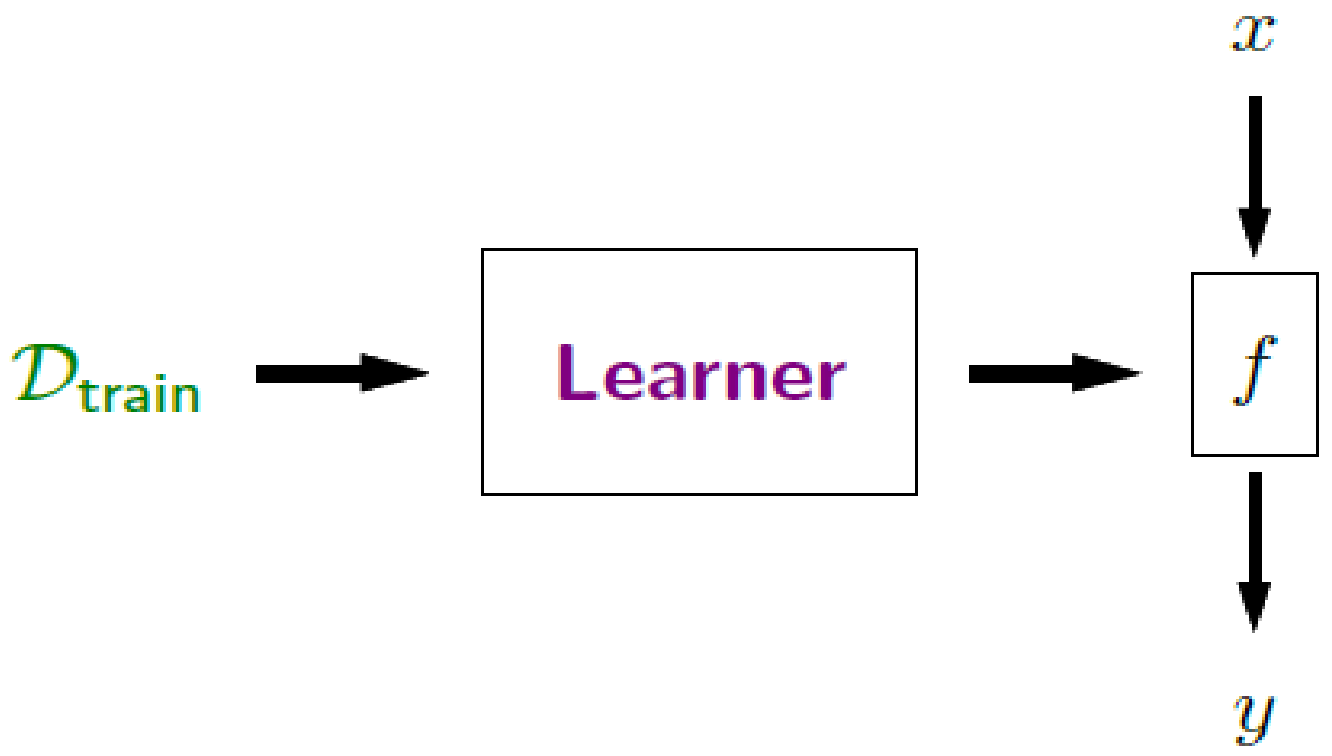
# Types of Prediction Tasks

- Binary classification (e.g., email ) ⇒ spam/not spam):
- $x \rightarrow f \rightarrow y \,\epsilon \{+1, -1\}$
- Regression (e.g., location, year ) ⇒ housing price):
- $x \rightarrow \; f \rightarrow \; y \,\epsilon \mathbb{R}$
- In the context of classification tasks, $f$ is called a classifier and $y$ is called a label (sometimes class, category, or tag).
- The key distinction between binary classification and regression is that the former has discrete outputs (e.g., "yes" or "no"), whereas the latter has continuous outputs.

# Data

- Example: species that $y$ is the ground-truth output for $x$, $(x, y)$
- Training data: list of examples $D_{train} = [(Ali, 3), (Ann, 2), (Austin, 5), \ldots]$
- The starting point of ML is the data.
- For now, we will focus on supervised learning, in which our data provides both inputs and outputs, in contrast to unsupervised learning, which only provides inputs.
- A (supervised) example (a data point or instance) is simply an input-output pair $(x, y)$, which species that $y$ is the ground-truth output for $x$.
- The training data $D_{train}$is a multiset of examples (repeats are allowed, but this is not important), which forms a partial specification of the desired behavior of a predictor.

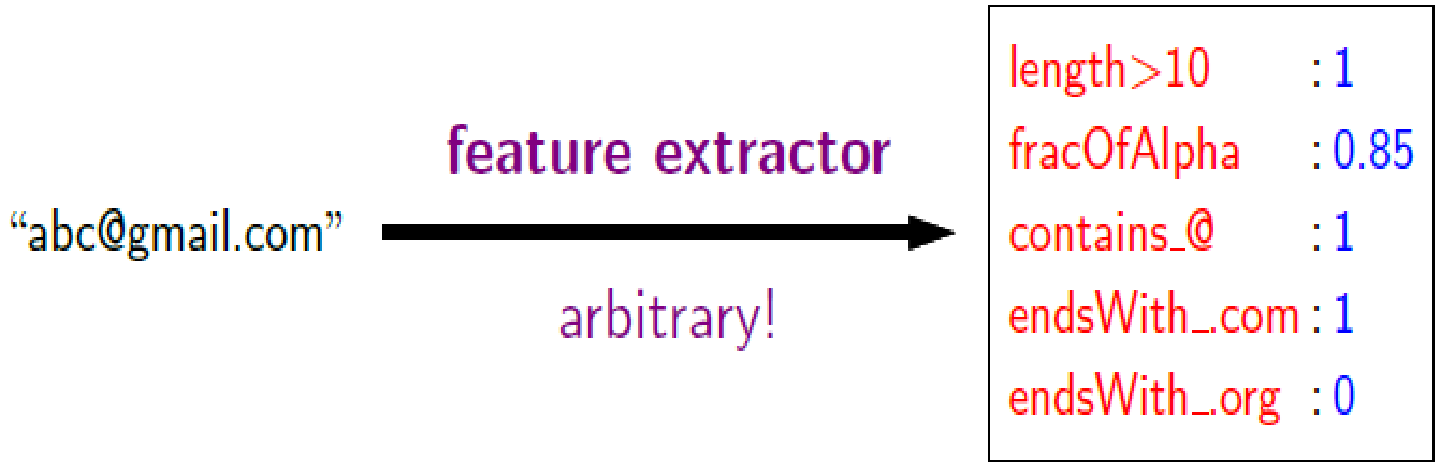
# Framework



- Learning is about taking the training data $D_{train}$ and producing a predictor $f$, which is a function that takes inputs $x$ and tries to map them to outputs $y = f(x)$.
- One thing to keep in mind is that we want $f$ to approximately work even for examples that we have not seen in $D_{train}$.
- This problem is referred as generalization.
- We will first focus on examining what $f$ is, independent of how the learning works.
- Then we will come back to learning $f$ based on data.

# Feature Extraction

- Example task: predict $y$, whether a string $x$ is an email address
- Question: what properties of $x$ might be relevant for predicting $y$?
- Feature extractor: Given input $x$, output a set of (feature name, feature value) pairs.



- We will consider predictors $f$ based on feature extractors.
- Feature extraction is a bit of an art that requires intuition about both the task and also what machine learning algorithms are capable of.
- The general principle is that features should represent properties of $x$ which might be relevant for predicting $y$.
- It is okay to add features which turn out to be irrelevant, since the learning algorithm can sort it out (though it might require more data to do so).

# Feature Vector Notation

- Definition: feature vector
  - For an input $x$, its feature vector is: $\phi(x) = [\phi_1(x), \ldots, \phi_d(x)]$
  - Think of $\phi(x) \epsilon \mathbb{R}^d$ as a point in a high-dimensional space.
- Each input $x$ represented by a feature vector $\phi(x)$, which is computed by the feature extractor.
- When designing features, it is useful to think of the feature vector as being a map from strings (feature names) to doubles (feature values).
- But formally, the feature vector $\phi(x) \epsilon \mathbb{R}^d$ is a real vector $\phi(x) = [\phi_1(x), \ldots, \phi_d(x)]$, where each component $\phi_j(x), for\ j = 1, \ldots, d$ represents a feature.
- This vector-based representation allows us to think about feature vectors as a point in a (high-dimensional) vector space.

# Weight Vector

Weight vector $\mathbf{w} \in \mathbb{R}^d$

| | |
|---|---|
| length>10 | :-1.2 |
| fracOfAlpha | :0.6 |
| contains_@ | :3 |
| endsWith_.com | :2.2 |
| endsWith_.org | :1.4 |

Feature vector $\phi(x) \in \mathbb{R}^d$

| | |
|---|---|
| length>10 | :1 |
| fracOfAlpha | :0.85 |
| contains_@ | :1 |
| endsWith_.com | :1 |
| endsWith_.org | :0 |

**Score**: weighted combination of features

$$\mathbf{w} \cdot \phi(x) = \sum_{j=1}^{d} w_j \phi(x)_j$$

Example: $-1.2(1) + 0.6(0.85) + 3(1) + 2.2(1) + 1.4(0) = 4.51$