Overview

CS 446 / ECE 449

2022-01-17 04:49:48 -0600 (d748537)

Plan for today

- Course webpage: staff, policies, schedule.
- ML examples.
- ML math/coding background.
- ML setting and meta-algorithms.
- First ML technique: linear regression.

What is machine learning?

Improving algorithms by fitting them to data.

Application 1: image classification

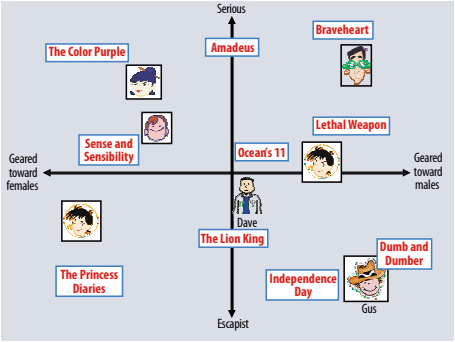
- Birdwatcher takes photos of birds, organized by species.
- **Goal**: automatically recognize bird species in new photos.

Indigo bunting

- **Why ML**: variation in lighting, occlusions, morphology.

Application 2: recommender system

- Netflix users watch movies and provide ratings.
- **Goal**: predict user's rating of unwatched movie.
- (**Real goal**: keep users paying customers.)
- (**Real effect**: reinforce stereotypes found in the data?)

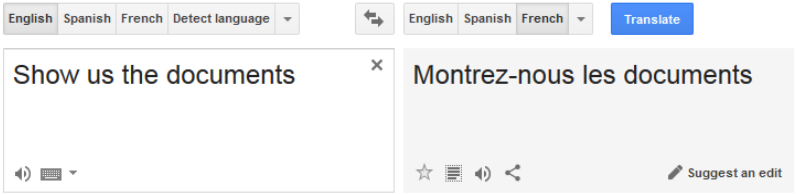


(Image credit: Koren, Bell, and Volinsky, 2009.)

- **Why ML**: easily adapt to and leverage movie attributes, viewer attributes, viewer relationships, etc.

Application 3: machine translation

- Linguists provide translations of all English language books into French, sentence-by-sentence.
- **Goal**: translate any English sentence into French.



Note: the text-to-speech is via ML (~~recurrent network~~transformer).

- **Why ML**? Not only avoid hard-coding many rules, but also capture idiom and other nuances.

Application 4: chess

- Chess enthusiasts construct a large corpus of chess games.
  (**Or:** start with nothing, play random games!)
- **Goal**: Win chess games.

- **Why ML**? Avoid hard-coding evaluation; magically interpolate between observed positions, as humans do.

Math/coding background

Math/coding background

Math.

- Linear algebra (e.g., null spaces; eigendecomposition; SVD...).
- Basic probability and statistics (e.g., variance of a random variable).
- Multivariable calculus (e.g., gradient of $\|\boldsymbol{A}\boldsymbol{w} - \boldsymbol{b}\|^2$ wrt $\boldsymbol{w}$).
- Basic proof writing (e.g., prove $\boldsymbol{A}^\top \boldsymbol{A}$ is positive semi-definite).

Math/coding background

Math.

- Linear algebra (e.g., null spaces; eigendecomposition; SVD...).
- Basic probability and statistics (e.g., variance of a random variable).
- Multivariable calculus (e.g., gradient of $\|\boldsymbol{A}\boldsymbol{w} - \boldsymbol{b}\|^2$ wrt $\boldsymbol{w}$).
- Basic proof writing (e.g., prove $\boldsymbol{A}^\top \boldsymbol{A}$ is positive semi-definite).

Coding.

- `python3`. It's slow, but often computation will be inside fast libraries.
- `numpy`, an easy-to-use numeric library.
- `pytorch`, a numeric library with gpu support, auto-differentiation, and deep learning helpers.

  My opinion. `pytorch` is one of the nicest libraries I've ever used, for anything. I use it for much more than deep learning.

python3, pytorch, numpy

```
>>> import numpy
>>> import torch

>>> 3 / 2
1.5
>>> 3 // 2
1

>>> A = torch.randn(5,5)
>>> b = torch.randn(5,1)
>>> x = torch.gels(b, A)[0]
>>> (A @ x - b).norm()
tensor(3.7985e-06)
```

See also: pytorch tutorial (lecture 9).

pytorch on gpu is easy

```
>>> import torch

>>> A = torch.randn(5, 5)
>>> b = torch.randn(5)
>>> (A @ b).norm()
tensor(4.7746)

>>> device = torch.device("cuda:0")
>>> (A.to(device) @ b.to(device)).norm()
tensor(4.7746, device='cuda:0')

>>> (A.to(device) @ b).norm()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
RuntimeError: Expected object of type
torch.cuda.FloatTensor but found type
torch.FloatTensor for argument #2 'vec'
```

Note. Homeworks will be graded in a gpu-free container!

Simplest setting: supervised learning

Supervised learning.

1. Receive training set $((\boldsymbol{x}_i, y_i))_{i=1}^n$, where each $\boldsymbol{x}_i$ is an input or covariate, and each y_i is a label or target. ("Supervision": something gives us the labels!)
2. Algorithmically choose a predictor f via the training set so that $f(\boldsymbol{x}) \approx y$ on future examples.

Simplest setting: supervised learning

Supervised learning.

1. Receive training set $((\boldsymbol{x}_i, y_i))_{i=1}^n$, where each $\boldsymbol{x}_i$ is an input or covariate, and each y_i is a label or target. ("Supervision": something gives us the labels!)
2. Algorithmically choose a predictor f via the training set so that $f(\boldsymbol{x}) \approx y$ on future examples.

Other settings.

- Unsupervised learning: find structure in $(\boldsymbol{x}_i)_{i=1}^n$ (no labels!).
- Time series label of $\boldsymbol{x}_i$ depends on $(\boldsymbol{x}_{i-1}, \boldsymbol{x}_{i-2}, \dots)$.
- Reinforcement learning: predictions/outputs affect future state (e.g., driving a car).

What are the difficulties?

Consider supervised learning (simplest setting):
learn $f : X \to Y$ from $((\boldsymbol{x}_i, y_i))_{i=1}^n$.

What are the difficulties?

Consider supervised learning (simplest setting):
learn $f : X \to Y$ from $((\boldsymbol{x}_i, y_i))_{i=1}^n$.

- How to encode data for the algorithm?

What are the difficulties?

Consider supervised learning (simplest setting):
learn $f : X \to Y$ from $((\boldsymbol{x}_i, y_i))_{i=1}^n$.

- How to encode data for the algorithm?
- How to clean/improve/augment data?

What are the difficulties?

Consider supervised learning (simplest setting):
learn $f : X \to Y$ from $((\boldsymbol{x}_i, y_i))_{i=1}^n$.

- How to encode data for the algorithm?
- How to clean/improve/augment data?
- How to choose structure/model for f?

What are the difficulties?

Consider supervised learning (simplest setting):
learn $f : X \to Y$ from $((\boldsymbol{x}_i, y_i))_{i=1}^n$.

- How to encode data for the algorithm?
- How to clean/improve/augment data?
- How to choose structure/model for f?
- How to algorithmically fit f to data?

What are the difficulties?

Consider supervised learning (simplest setting):
learn $f : X \to Y$ from $((\boldsymbol{x}_i, y_i))_{i=1}^n$.

- How to encode data for the algorithm?
- How to clean/improve/augment data?
- How to choose structure/model for f?
- How to algorithmically fit f to data?
- How to ensure f does not overfit,
  meaning it is good on future predictions, and not just on $((\boldsymbol{x}_i, y_i))_{i=1}^n$?

"pytorch meta-algorithm"

1. Clean/augment data (lecture 10?).
2. Pick model/architecture (many lectures).
3. Pick a loss function measuring model fit to data (lectures 2-4, 6).
4. Run a gradient descent variant to fit model to data (many lectures).
5. Tweak 1-4 until training error is small.
6. Tweak 1-5, possibly reducing model complexity, until testing error is small (lectures 4, 6, 13).

"pytorch meta-algorithm"

1. Clean/augment data (lecture 10?).
2. Pick model/architecture (many lectures).
3. Pick a loss function measuring model fit to data (lectures 2-4, 6).
4. Run a gradient descent variant to fit model to data (many lectures).
5. Tweak 1-4 until training error is small.
6. Tweak 1-5, possibly reducing model complexity, until testing error is small (lectures 4, 6, 13).

Is that all of ML?

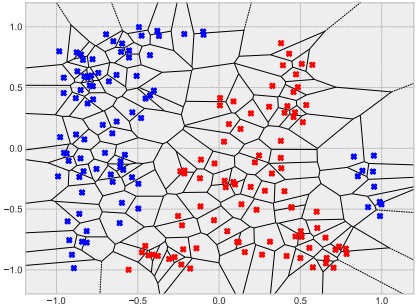
"pytorch meta-algorithm"

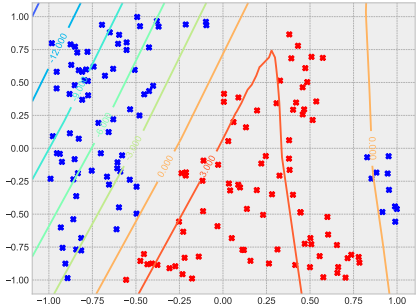
1. Clean/augment data (lecture 10?).
2. Pick model/architecture (many lectures).
3. Pick a loss function measuring model fit to data (lectures 2-4, 6).
4. Run a gradient descent variant to fit model to data (many lectures).
5. Tweak 1-4 until training error is small.
6. Tweak 1-5, possibly reducing model complexity, until testing error is small (lectures 4, 6, 13).

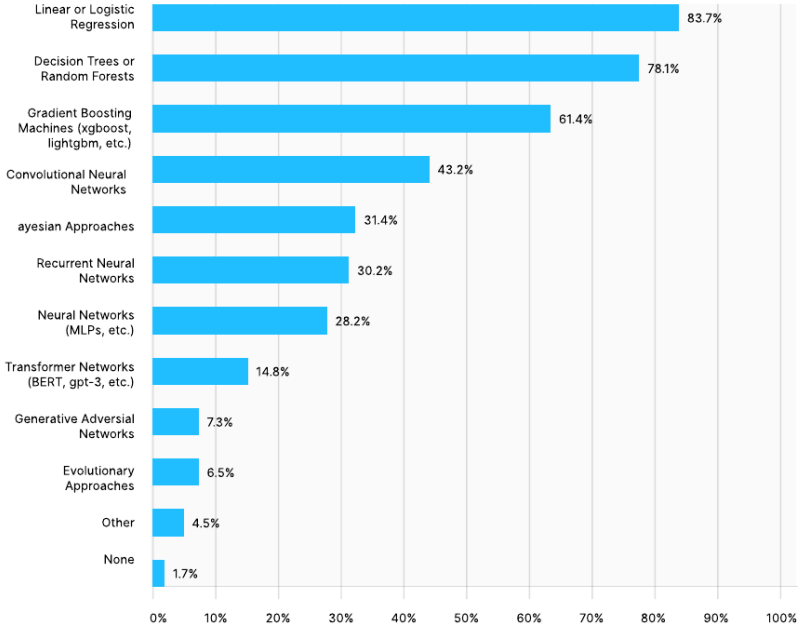
Is that all of ML?
No, but these days it's much of it!

What is "model/architecture choice"?

Should we use 1-nearest-neighbor or a 2-layer ReLU network?







(From kaggle 2020 survey.)

Linear regression — basic setup

1. Start from training data $((\boldsymbol{x}_i, y_i))_{i=1}^n$, with $\boldsymbol{x}_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$.
2. Model is a linear predictor: pick $\boldsymbol{w} \in \mathbb{R}^d$ with

$$\boldsymbol{x}_i \mapsto \boldsymbol{w}^\mathsf{T} \boldsymbol{x}_i =: \hat{y}_i \approx y_i.$$

3. Loss function ℓ is squared loss ℓ_{sq} (standard regression loss):

$$\ell_{\mathrm{sq}}(\boldsymbol{w}^\mathsf{T} \boldsymbol{x}_i, y_i) = \frac{1}{2}(\boldsymbol{w}^\mathsf{T} \boldsymbol{x}_i - y_i)^2.$$

We will minimize the empirical risk (average loss over training examples):

$$\widehat{\mathcal{R}}(\boldsymbol{w}) = \frac{1}{n} \sum_{i=1}^n \ell_{\mathrm{sq}}(\boldsymbol{w}^\mathsf{T} \boldsymbol{x}_i, y_i) = \frac{1}{2n} \sum_{i=1}^n \left(\boldsymbol{w}^\mathsf{T} \boldsymbol{x}_i - y_i \right)^2.$$

Convenient form using matrices:

$$\widehat{\mathcal{R}}(\boldsymbol{w}) = \frac{1}{2n} \|\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}\|^2 \qquad \text{where } \boldsymbol{X} := \begin{bmatrix} \leftarrow & \boldsymbol{x}_1^\mathsf{T} & \rightarrow \\ & \vdots & \\ \leftarrow & \boldsymbol{x}_n^\mathsf{T} & \rightarrow \end{bmatrix} \in \mathbb{R}^{n \times d}.$$

Linear regression — basic solver

3. Minimize the empirical risk

$$\widehat{\mathcal{R}}(\boldsymbol{w}) = \frac{1}{2n}\|\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}\|^2 \qquad \text{where } \boldsymbol{X} := \begin{bmatrix} \leftarrow & \boldsymbol{x}_1^\mathsf{T} & \rightarrow \\ & \vdots & \\ \leftarrow & \boldsymbol{x}_n^\mathsf{T} & \rightarrow \end{bmatrix} \in \mathbb{R}^{n\times d}.$$

4. Basic method: gradient descent. Set $\boldsymbol{w}_0 = 0$, and thereafter

$$\boldsymbol{w}_{i+1} := \boldsymbol{w}_i - \eta\nabla\widehat{\mathcal{R}}(\boldsymbol{w}_i) = \boldsymbol{w}_i - \frac{\eta}{n}\boldsymbol{X}^\mathsf{T}(\boldsymbol{X}\boldsymbol{w}_i - \boldsymbol{y}),$$

where η is a learning rate (step size).

```
w = torch.zeros(d)
for i in range(niters):
  w -= eta * X.T @ (X @ w - y) / n
```

Summary for today

- Course webpage: staff, policies, schedule.
- ML examples.
- ML math/coding background.
- ML setting and meta-algorithms.
- First ML technique: linear regression.

(Appendix.)

Supplemental reading

- Murphy: chapter 1.