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.

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

![Indigo bunting](image.png)
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.

![Image of translation tool]

**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

▶ 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 $\|Aw - b\|_2^2$ with respect to $w$).
▶ Basic proof writing (e.g., prove $A^TA$ 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.
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 \( \|A w - b\|^2 \) wrt \( w \)).
- Basic proof writing (e.g., prove \( A^T A \) 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.
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 $\|A w - b\|^2$ wrt $w$).
- Basic proof writing (e.g., prove $A^T A$ 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.
```python
>>> 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).
```
>>> 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 \(((x_i, y_i))_{i=1}^{n}\), where each \(x_i\) is an input or covariate, and each \(y_i\) is a label or target. (“Supervision”: something determines the labels!)

2. Algorithmically choose a predictor \(f\) via the training set so that \(f(x) \approx y\) on future examples.
Simplest setting: supervised learning

**Supervised learning.**

1. Receive training set \(((x_i, y_i))_{i=1}^{n}\), where each \(x_i\) is an input or covariate, and each \(y_i\) is a label or target. ("Supervision": something determines the labels!)

2. Algorithmically choose a predictor \(f\) via the training set so that \(f(x) \approx y\) on future examples.

**Other settings.**

- **Unsupervised learning**: find structure in \((x_i)_{i=1}^{n}\) (no labels!).
- **Time series** label of \(x_i\) depends on \((x_{i-1}, x_{i-2}, \ldots)\).
- **Reinforcement learning**: predictions/outputs affect future state (e.g., driving a car).
What are the difficulties?

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

▶ 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 $((x_i, y_i))_{i=1}^n$?
What are the difficulties?

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

- How to encode data for the algorithm?
What are the difficulties?

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

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

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

- 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 \( (x_i, y_i)_{i=1}^n \).

- 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 \( f : X \rightarrow Y \) from \( ((x_i, y_i))_{i=1}^n \).

▶ 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 \( ((x_i, y_i))_{i=1}^n \)?
"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).
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?
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!
Model choice?

Should we use **1-nearest-neighbor** or a **2-layer ReLU network**?
Model/algorithm choice?

- Linear or Logistic Regression: 83.7%
- Decision Trees or Random Forests: 78.1%
- Gradient Boosting Machines (xgboost, lightgbm, etc.): 61.4%
- Convolutional Neural Networks: 43.2%
- Bayesian Approaches: 31.4%
- Recurrent Neural Networks: 30.2%
- Neural Networks (MLPs, etc.): 28.2%
- Transformer Networks (BERT, gpt-3, etc.): 14.8%
- Generative Adversial Networks: 7.3%
- Evolutionary Approaches: 6.5%
- Other: 4.5%
- None: 1.7%

(From kaggle 2020 survey.)
Linear regression — basic setup

1. Start from training data \(((x_i, y_i))_{i=1}^n\), with \(x_i \in \mathbb{R}^d\) and \(y_i \in \mathbb{R}\).
2. Model is a linear predictor: pick \(w \in \mathbb{R}^d\) with
   \[x_i \mapsto w^T x_i =: \hat{y}_i \approx y_i.\]
3. Loss function \(\ell\) is squared loss \(\ell_{sq}\) (standard regression loss):
   \[\ell_{sq}(w^T x_i, y_i) = \frac{1}{2}(w^T x_i - y_i)^2.\]

We will minimize the empirical risk (average loss over training examples):
\[\hat{R}(w) = \frac{1}{n} \sum_{i=1}^n \ell_{sq}(w^T x_i, y_i) = \frac{1}{2n} \sum_{i=1}^n (w^T x_i - y_i)^2.\]

Convenient form using matrices:
\[\hat{R}(w) = \frac{1}{2n} \|Xw - y\|^2 \quad \text{where} \quad X := \begin{bmatrix} x_1^T & \cdots & x_n^T \end{bmatrix} \in \mathbb{R}^{n \times d}.\]
3. Minimize the empirical risk

\[ \hat{R}(w) = \frac{1}{2n} \|Xw - y\|^2 \]

where \( X := \begin{bmatrix}
\leftarrow x_1^T \rightarrow \\
\vdots \\
\leftarrow x_n^T \rightarrow 
\end{bmatrix} \in \mathbb{R}^{n \times d}. \)

4. Basic method: gradient descent. Set \( w_0 = 0 \), and thereafter

\[ w_{i+1} := w_i - \eta \nabla \hat{R}(w_i) = w_i - \frac{\eta}{n} X^T (Xw_i - y), \]

where \( \eta \) is a learning rate (step size).

```python
w = torch.zeros(d)
for _ in range(niters):
    w -= X.T @ (X @ w - y) / 100 / 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.