Online Learning
Perceptron Algorithm

Online learning:
Fitting models from streaming data
NOTATION WARNING!!!!!!!

• To save a lot of writing, here we assume linear features

\[ h_j(x) = x[j] \]

• Things follow straightforwardly for the more general case by replacing \( x[j] \) with \( h_j(x) \)

---

Batch vs online learning

**Batch learning**
• All data is available at start of training time

```
Data ➔ ML algorithm ➔ Ŵ
```

**Online learning**
• Data arrives (streams in) over time
  – Must train model as data arrives!

```
t=1
Data

Data ➔ ML algorithm ➔ Ŵ

Data ➔ ML algorithm ➔ Ŵ

Data ➔ ML algorithm ➔ Ŵ

Data ➔ ML algorithm ➔ Ŵ
```
Online learning example: Ad targeting

Input: $x_t$
- User info, page text

ML algorithm

$\hat{\mathbf{w}}(t) \rightarrow \hat{\mathbf{w}}(t+1)$

$\hat{y} = \text{Suggested ads}$

User clicked on Ad2

$y_t = \text{Ad2}$

Online learning problem

- Data arrives over each time step $t$:
  - Observe input $x_t$
    - Info of user, text of webpage
    - Note: many assumptions possible, e.g., data i.i.d., adversarially chosen...Details beyond scope of course
  - Make a prediction $\hat{y}_t$
    - Which ad to show
    - Note: many models possible, we focus on linear models
  - Observe true output $y_t$
    - Which ad user clicked on
    - Note: other observation models are possible, e.g., we don’t observe the label directly, but only a noisy version...Details beyond scope of course

Need ML algorithm to update coefficients each time step!
Stochastic gradient ascent can be used for online learning!!!

- init $\mathbf{w}^{(1)} = 0$, $t=1$
- Each time step $t$:
  - Observe input $x_t$
  - Make a prediction $\hat{y}_t$
  - Observe true output $y_t$
  - Update coefficients:

```
for j=0,...,D
    w_j^{(t+1)} \leftarrow w_j^{(t)} + \eta \frac{\partial \ell_t(\mathbf{w})}{\partial w_j}
```

The perceptron algorithm
The perceptron algorithm [Rosenblatt ’58, ’62]

- Classification setting: \( y \) in \{-1, +1\}
- Linear model
  - Prediction:

- Training:
  - Initialize weight vector:
  - At each time step:
    - Observe features:
    - Make prediction:
    - Observe true class:
  - Update model:
    - If prediction is not equal to truth

Intuition

If \( \hat{y} = y_t \),
\[
\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)}
\]
else
\[
\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} + y_t \mathbf{x}_t
\]
\[
\hat{y} = \text{sign}(\mathbf{w}^{(t)} \cdot \mathbf{x}_t)
\]

Why is this a reasonable update rule?
Which weight vector to report?

- Practical problem for all online learning methods
- Suppose you run online learning method and want to sell your learned weight vector... Which one do you sell???

- Last one?
- Random
- Average
- Voting + more advanced

Choice can make a huge difference!!
How many mistakes can a perceptron make?

Mistake bounds

Algorithm “pays” every time it makes a mistake:

How many mistakes is it going to make?
Linear separability: More formally, using margin

Data linearly separable, if there exists
- a vector
- a margin
such that

Perceptron analysis: Linearly separable case

Theorem [Block, Novikoff]:
- Given a sequence of labeled examples:
  - Each feature vector has bounded norm:
    - If dataset is linearly separable:

Then the # mistakes made by the online perceptron on this sequence is bounded by
Perceptron proof for linearly separable case

• Every time we make a mistake, we get $\gamma$ closer to $w^*$:
  - Mistake at time $t$: $w^{(t+1)} \leftarrow w^{(t)} + y_t x_t$
  - Taking dot product with $w^*$:
  - Thus after $m$ mistakes:

• Similarly, norm of $w^{(t+1)}$ doesn’t grow too fast:
  $$||w^{(t+1)}||^2 = ||w^{(t)}||^2 + 2y_t (w^{(t)} \cdot x_t) + ||x_t||^2$$
  - Thus, after $m$ mistakes:

• Putting all together:

Beyond linearly separable case

• Perceptron algorithm is super cool!
  - No assumption about data distribution!
    • Could be generated by an oblivious adversary, no need to be iid
  - Makes a fixed number of mistakes, and it’s done for ever!
    • Even if you see infinite data

• However, real world not linearly separable
  - Can’t expect never to make mistakes again
  - Analysis extends to non-linearly separable case
  - Very similar bound, see Freund & Schapire
  - Converges, but ultimately may not give good accuracy (make many many mistakes)
What is the perceptron optimizing?

What is the perceptron doing???

• When we discussed logistic regression:
  – Started from maximizing conditional log-likelihood

• When we discussed the perceptron:
  – Started from description of an algorithm

• What is the perceptron optimizing????
Perceptron prediction: Margin of confidence

Hinge loss

- Perceptron prediction:
- Makes a mistake when:
  - Hinge loss (same as maximizing the margin used by SVMs)
Minimizing hinge loss in batch setting

- Given a dataset:
- Minimize average hinge loss:
- How do we compute the gradient?

Subgradients of convex functions

- Gradients lower bound convex functions:
- Gradients are unique at $x$ if function differentiable at $x$
- Subgradients: Generalize gradients to non-differentiable points:
  - Any plane that lower bounds function:
Subgradient of hinge

• Hinge loss:

Subgradient of hinge loss:
- If $y_i(w \cdot x_i) > 0$:
- If $y_i(w \cdot x_i) < 0$:
- If $y_i(w \cdot x_i) = 0$:
- In one line:

Subgradient descent for hinge minimization

• Given data: $(x_1, y_1), \ldots, (x_N, y_N)$

• Want to minimize:

$$\frac{1}{N} \sum_{i=1}^{N} \ell(w, x_i) = \frac{1}{N} \sum_{i=1}^{N} (-y_i(w \cdot x_i))^+$$

• Subgradient descent works the same as gradient descent:
  - But if there are multiple subgradients at a point, just pick (any) one:
Perceptron revisited

• Perceptron update:
  \[ w^{(t+1)} \leftarrow w^{(t)} + \mathbb{I} \left[ y_t (w^{(t)} \cdot x_t) \leq 0 \right] y_t x_t \]

• Batch hinge minimization update:
  \[ w^{(t+1)} \leftarrow w^{(t)} + \eta \frac{1}{N} \sum_{i=1}^{N} \left\{ \mathbb{I} \left[ y_i (w^{(t)} \cdot x_i) \leq 0 \right] y_i x_i \right\} \]

• Difference?

What you need to know

• Notion of online learning
• Perceptron algorithm
• Mistake bounds and proof
• In online learning, report averaged weights at the end
• Perceptron is optimizing hinge loss
• Subgradients and hinge loss
• (Sub)gradient decent for hinge objective