MACHINE LEARNING

Overfitting: Good ML "Hygiene"

Remember, you can overfit not just the parameters, but your design choices!

For any given task:

- Have train, dev, val, and test set.
- Train your estimators on the train set.
- Choose hyperparameters based on dev set.
  - Function class
  - Regularization weight
  - Number of iterations to train, etc.
- Keep periodically checking to see if it generalizes to val set.
- Look at the test set only at the "end" of the project.

Learning Components

- Training set of pairs \((x_i, y_i)\)
- Loss function \(L(y, \hat{y})\)
- Hypothesis Space \(H\) to search over for

\[
f = \arg \min_{f \in H} \sum_i L(y_i, f(x_i))
\]

- Basically, algorithm design by trial and error (on training set)
- A better way of solving problems when the problems are ill-posed
- Need to watch out for over-fitting the training set
- The only guarantee that you are not overfitting is empirical verification!

Classification

Consider the case when \(y\) is binary, i.e., \(\mathcal{Y} = \{0, 1\}\).

How do you define the loss function then?

- Ideally, \(L(y, \hat{y})\) is 0 if they are equal, 1 otherwise.

But don’t know how to solve that. What if we solved by regression?

\[
w = \arg \min_w \frac{1}{T} \sum_i (y_i - w^T \hat{x}_i)^2
\]

And at test time, we can output \(y = 1\) if \(w^T \hat{x} > 0.5\) and 0 otherwise.

The problem is the loss function will penalize \(w^T \hat{x}_i > 1\) when \(y_i = 1\). While at test time, this would give us exactly the right answer!
MACHINE LEARNING

Logistic regression

- Learn a function \( f(x) = P(y = 1) \) which regresses to the probability \( y \) is 1.
- We have to choose \( f \) such that the domain of \( f(x) \) lies between [0, 1].

\[
f(x; w) = \sigma(w^T\hat{x}), \quad \sigma(p) = \frac{\exp(p)}{1 + \exp(p)}
\]

- This ensures that the output of \( f \) is between [0, 1]
- \( w^T\hat{x} \) can be interpreted as the log of the odds, or log of ratio between \( P(y = 1) \) to \( P(y = 0) \)
- \( \hat{x} \) is some augmented "feature vector" derived from \( x \).
  - "Linear Classifier" if \( \hat{x} = [x^T; 1]^T \) (log-odds are linear)
  - Could be polynomial \( \hat{x} = [1, x, x^2, x^3] \)
  - Or other arbitrary non-linear functions of \( x \)
  - Can apply even when \( x \) is non-numeric, as long as \( \hat{x} \) is numeric.

MACHINE LEARNING

Logistic Regression

For Binary Classification: \( \mathcal{X} \rightarrow [0, 1] \)

\[
f(x; w) = \sigma(w^T\hat{x}) = \frac{\exp(w^T\hat{x})}{1 + \exp(w^T\hat{x})}
\]

- To classify, \( y = 1 \) if \( P(y = 1) > 0.5 \) or 0 otherwise
- That is, \( y = 1 \) if \( w^T\hat{x} > 0 \) and 0 otherwise.

- Note: Classifier is linear in chosen encoding \( \hat{x} \).
- \( w^T\hat{x} \neq 0 \) defines a "separating hyperplane" between positive and negative part of the space of \( \hat{x} \).

Will correspond to a non-linear boundary in the original \( x \) space

Decision “boundary” Hyperplane
**MACHINE LEARNING**

Logistic regression

\[ P(y = 1) = f(x) = \sigma(w^T \hat{x}) \]

What about the loss?

Cross-Entropy Loss

If true is 1, we want \( f \) to be high, and if it is 0, we want it to be low.

\[
L(y, f(x)) = \begin{cases} 
\log P(y = 1) = \log f(x) & \text{if } y = 1 \\
\log P(y = 1) = \log 1 - f(x) & \text{if } y = 0 
\end{cases}
\]

There’s a minus because this is the loss.

Minimizing \( \sum L(y_i, f(x_i)) \) can be viewed as maximizing the sum of the log-probabilities, or the product of the probabilities of the labels \( y_i \) under our predicted distribution.

Promotes a high probability for the correct label > uniform distribution (low confidence) over both labels > high probability for incorrect label.

But now, how do we minimize this function in terms of \( w \)? No longer least-squares.

---

**GRADIENT DESCENT**

Logistic Regression

\[ f(x; w) = \sigma(w^T \hat{x}) = \frac{\exp(w^T \hat{x})}{1 + \exp(w^T \hat{x})} \]

- Cross-entropy / Negative Likelihood Loss

\[
L(y, f(x; w)) = -y \log f(x; w) - (1 - y) \log(1 - f(x; w))
\]

Putting it all together, given a training set of \( \{(x_i, y_i)\} \):

\[
w = \arg \min_w \frac{1}{T} \sum_{t=1}^{T} y_t \log [1 + \exp(-w^T \hat{x}_t)] + (1 - y_t) \log [1 + \exp(w^T \hat{x}_t)]
\]

You can show that this loss is a convex function of \( w \)
(compute the Hessian matrix and show that it’s eigenvalues are non-negative)

So it has a single global minimum.

But how do we find it?
**Gradient Descent**

Logistic Regression

\[ w = \arg \min_w \frac{1}{T} \sum_{t=1}^{T} y_t \log [1 + \exp(-w^T x_t)] + (1 - y_t) \log [1 + \exp(w^T x_t)] \]

More General Form

\[ w = \arg \min_w C(w) \quad C(w) = \frac{1}{T} \sum_{t} C_t(w) \]

Iterative algorithm

- Given a current estimate of \( w \), approximate \( C(w) \) as a linear function of \( w \)
  - \( C(w) = \alpha^T w \)
- Do this fit by computing the gradient of \( C(w) \) wrt \( w \)
  - \( \alpha = \nabla_w C(w) \) (would be true if \( C(w) = \alpha^T w \))

Think of \([C(w), w]\) as the co-ordinates on a plane. Which direction to move in \( w \)-space to reduce \( C(w) \)?

\[ -\alpha \]

---

**Gradient Descent**

\[ w = \arg \min_w C(w) \quad C(w) = \frac{1}{T} \sum_{t} C_t(w) \]

- Begin with initial guess \( w_0 \)
- At each iteration:
  - \( w_{i+1} \leftarrow w_i - \gamma \nabla_w C(w_i) \)
- At each iteration, we update the parameters \( w \) by "moving", in \( w \)-space, in the opposite direction of the gradient (at that point \( w_i \)).
- \( \gamma \) is the step-size. When running optimization for training, often called the "learning rate".
- In some cases, \( \gamma \) can be set by doing a line-search
  - Check values of \( C(w - \gamma \nabla_w) \) and pick \( \gamma \) which minimizes the cost
- In other cases, we choose a fixed value of \( \gamma \) (or change it in some pre-determined schedule per iteration)
  - Then, we are moving by a distance that is proportional to magnitude of the gradient
If you select optimal step size by doing a “line search” for $\gamma$, you can prove that gradient-descent will converge.

If the function is convex, gradient-descent will converge to the unique global minimum.

Second order variants that consider the Hessian matrix include:

- Newton’s Method
- Quasi-Newton Methods
- Levenberg-Marquardt

But simple gradient descent suffices / our only choice when:

- Function isn’t convex.
- Can’t afford to do line search.
- So many parameters that can’t compute Hessian.

Also, no theoretical guarantees.

Theory still catching up. Meanwhile, we’ll try to understand the “behavior” of the gradients.

---

**Logistic Regression**

What is $\nabla_w C(w)$, the gradient of the loss from a single training example?

$$\nabla_w C(w) = \begin{bmatrix} \frac{\partial}{\partial w_1} C(w) \\ \frac{\partial}{\partial w_2} C(w) \\ \vdots \end{bmatrix}$$

If $C(w) = \frac{1}{T} \sum_t C_t(w)$, then $\nabla_w C(w) = \frac{1}{T} \sum_t \nabla_w C_t(w)$

Logistic Regression

$$C_t(w) = y_t \log \left[ 1 + \exp(-w^T \tilde{x}_t) \right] + (1 - y_t) \log \left[ 1 + \exp(w^T \tilde{x}_t) \right]$$

What is $\nabla_w C_t(w)$, the gradient of the loss from a single training example?

---

$$C_t(w) = y_t \log \left[ 1 + \exp(-w^T \tilde{x}_t) \right] + (1 - y_t) \log \left[ 1 + \exp(w^T \tilde{x}_t) \right]$$

Ok, what is the derivative of

$$C_t(p) = y_t \log \left[ 1 + \exp(-p) \right] + (1 - y_t) \log \left[ 1 + \exp(p) \right]$$

with respect to $p$ (where $p$ is a scalar).

$$\frac{\partial}{\partial p} C_t(p) = \frac{\exp(p) - y_t}{1 + \exp(p)}$$

$$= \frac{\exp(p)}{1 + \exp(p)} - y_t \left[ \frac{\exp(-p)}{1 + \exp(-p)} + \frac{\exp(p)}{1 + \exp(p)} \right]$$

$$= \frac{\exp(p)}{1 + \exp(p)} - y_t \left[ \frac{1}{1 + \exp(p)} + \frac{\exp(p)}{1 + \exp(p)} \right]$$
\[ C_i(w) = y_i \log \left[ 1 + \exp(-\mathbf{w}^T \tilde{x}_i) \right] + (1 - y_i) \log \left[ 1 + \exp(\mathbf{w}^T \tilde{x}_i) \right] \]

\[ C_i(p) = y_i \log \left[ 1 + \exp(-p) \right] + (1 - y_i) \log \left[ 1 + \exp(p) \right] \]

\[ \frac{\partial}{\partial p} C_i(p) = \frac{\exp(p)}{1 + \exp(p)} - y_i \]

Observations

- \( \frac{\exp(p)}{1 + \exp(p)} \) is basically the output \( f(x_i; w) \), predicted probability that \( y_i = 1 \).
- Remember: this is the expression for gradient of \( p \), i.e. logit / log-odds.
- Gradient 0 if \( y_i = 0 \) and probability 0, \( y = 1 \) and probability 1.
- Do nothing if predicting right answer with perfect confidence.
- If we say probability \( > 0 \), and \( y_i = 0 \). Gradient is positive.
- If we say probability \( < 1 \), and \( y_i = 1 \). Gradient is negative.

Remember we move in the opposite direction of gradient.

\[ w = \arg \min_w \frac{1}{T} \sum_{i=1}^{T} y_i \log \left[ 1 + \exp(-\mathbf{w}^T \tilde{x}_i) \right] + (1 - y_i) \log \left[ 1 + \exp(\mathbf{w}^T \tilde{x}_i) \right] \]

Putting it together:

- At each iteration \( t \),
  - Based on current \( w \), compute \( f(x_i, w) = \tilde{y}_i \)
  - Compute derivative of the "output" as \( \tilde{y}_i - y_i \)
  - Multiply by \( x_i \) to get \( \nabla_w \)
  - Change \( w \) by subtracting some \( \gamma \) times this gradient.

\[ \nabla_w C_i(w) = \nabla_w (\mathbf{w}^T \tilde{x}_i) \left[ \frac{\exp(\mathbf{w}^T \tilde{x}_i)}{1 + \exp(\mathbf{w}^T \tilde{x}_i)} - y_i \right] \]

\[ \nabla_w C_i(w) = \nabla_w (\mathbf{w}^T \tilde{x}_i) \frac{\partial C_i(p)}{\partial p} \]
**Gradient Descent**

Putting it together:

- At each iteration $i$,
  - Based on current $w$, compute $f(x_i, w) = \hat{y}_i$ for every training sample
  - Compute derivative of the "output" as $\hat{y}_i - y_i$ for every training sample
  - Multiply by $x_i$ and average across all training samples to get $V_w$
  - Change $w$ by subtracting some $\gamma$ times this gradient.

$C(w) = \frac{1}{T} \sum_{t=1}^{T} C_t(w) \Rightarrow V_w = \frac{1}{T} \sum_{t=1}^{T} V_w C_t$

Expensive when we have a LOT of training data.

**Stochastic Gradient Descent**

In practice:

- Shuffle order of training examples
- Choose a batch size
- Take consecutive groups of $B$ samples as you loop through iterations
  - $[1, B]$ in iteration 1
  - $[B+1, 2B]$ in iteration 2
  - ...
- Once you reach the end of the training set (called one "epoch"), shuffle the order again.