Classification

- CLS 3a. Stochastic Gradient Decent
- CLS 3b. Evaluation Metrics

How fast does GE converge?

- Theorem

If ℓ is both convex and differentiable [1]

$$\ell\left(w^{(k)}\right)-\ell\left(w^{*}\right) \leq \begin{cases} \frac{\|w^{(0)}-w^{*}\|_2^2}{2\eta k} = O\left(\frac{1}{k}\right) & \ell \text{ is convex} \\ \frac{c^k L \|w^{(0)}-w^{*}\|_2^2}{2} = O\left(c^k\right) & \ell \text{ is strongly convex} \end{cases} \quad (4)$$

where k is the number of iterations and $c \in (0,1)$.

1: the step size η must be no larger than $\frac{1}{L}$ where L is the Lipschitz constant satisfying
$\|\nabla\ell(a) - \nabla\ell(b)\|_2 \leq L\|a-b\|_2 \;\; \forall a, b$

How fast does GD converge?

- Theorem

If ℓ is both convex and differentiable [1]

$$\ell\left(w^{(k)}\right)-\ell\left(w^{*}\right) \leq \begin{cases} \frac{\|w^{(0)}-w^{*}\|_2^2}{2\eta k} = O\left(\frac{1}{k}\right) & \ell \text{ is convex} \\ \frac{c^k L\|w^{(0)}-w^{*}\|_2^2}{2} = O\left(c^k\right) & \ell \text{ is strongly convex} \end{cases} \quad (4)$$

where k is the number of iterations and $c \in (0,1)$.

  - In general, to achieve $l\left(w^{(k)}\right) - l(w^{*}) \leq \rho$, GD needs $O\left(\frac{1}{\rho}\right)$ iterations;
  - With strong convexity, it takes $O\left(\log\left(\frac{1}{\rho}\right)\right)$ siterations[2]

2: Convex Optimization, S. Boyd & L. Vandenberghe, Ch 9.3

Why not happy with GD?

- Fast convergence ≠ high efficiency

$$w^{(k)} = w^{(k-1)} - \eta_k \nabla \ell\left(w^{(k-1)}\right) \quad (5)$$

$$= w^{(k-1)} - \eta_k \nabla\left[\frac{1}{n}\sum_{i=1}^{n}\ell_i\left(w^{(k-1)}\right)\right] \quad (6)$$

- Per-iteration complexity = $O(n)$ (extremely large); a single iteration may take forever.
- How to make it cheaper? GD ⇒ SGD

Stochastic Gradient Decent

Approximate the full gradient via an unbiased estimator

$$w^{(k)} = w^{(k-1)} - \eta_k \nabla \left(\frac{1}{n} \sum_{i=1}^{n} \ell_i \left(w^{(k-1)} \right) \right) \tag{7}$$

$$\approx \underbrace{w^{(k-1)} - \eta_k \nabla \left(\frac{1}{|B|} \sum_{i \in B} \ell_i \left(w^{(k-1)} \right) \right)}_{\text{mini-batch SGD }^3} \quad B \overset{unif}{\sim} \{1, 2, \dots n\} \tag{8}$$

$$\approx \underbrace{w^{(k-1)} - \eta_k \nabla \ell_i \left(w^{(k-1)} \right)}_{\text{pure SGD}} \quad i \overset{unif}{\sim} \{1, 2, \dots n\} \tag{9}$$

Trade-off: lower computation cost v.s. larger variance

[3] When using GPU, $|B|$ usually depends on the memory budget.

GD vs SGD

For strongly convex $\ell(w)$, according to [Bottou, 2012]

Optimizer	GD	SGD	Winner
Time per-iteration	$O(n)$	$O(1)$	SGD
Iterations for accuracy ρ	$O\left(\log\left(\frac{1}{\rho}\right)\right)$	$\tilde{O}\left(\frac{1}{\rho}\right)$	GD
Time for accuracy ρ	$O\left(n \log \frac{1}{\rho}\right)$	$\tilde{O}\left(\frac{1}{\rho}\right)$	Depends
Time for test-set error ϵ	$O\left(\frac{1}{\epsilon^{1/\alpha}} \log \frac{1}{\epsilon}\right)$	$\tilde{O}\left(\frac{1}{\epsilon}\right)$	SGD

where $\frac{1}{2} \leq \alpha \leq 1$

SVM Solver: Pegasos [Shalev-Shwartz et al., 2011]

$$\ell_i(w) = \max\left(0, 1 - y_i w^\top x_i\right) + \frac{\lambda}{2}\|w\|^2 \tag{10}$$

$$= \begin{cases} \frac{\lambda}{2}\|w\|^2 & y_i w^\top x_i \geq 1 \\ 1 - y_i w^\top x_i + \frac{\lambda}{2}\|w\|^2 & y_i w^\top x_i < 1 \end{cases} \tag{11}$$

Therefore

$$\nabla \ell_i(w) = \begin{cases} \lambda w & y_i w^\top x_i \geq 1 \\ \lambda w - y_i x_i & y_i w^\top x_i < 1 \end{cases} \tag{12}$$

10

SVM Solver in 10 Lines

Algorithm 1: Pegasos: SGD solver for SVMs

Input: n, λ, T;
Initialization: $w \leftarrow 0$;
for $k = 1, 2, \ldots, T$ **do**
  $i \overset{uni}{\sim} \{1, 2, \ldots n\}$;
  $\eta_k \leftarrow \frac{1}{\lambda k}$;
  if $y_i {w^{(k)}}^\top x_i < 1$ **then**
    $w^{(k+1)} \leftarrow w^{(k)} - \eta_k \left(\lambda w^{(k)} - y_i x_i\right)$
  else
    $w^{(k+1)} \leftarrow w^{(k)} - \eta_k \lambda w^{(k)}$
  end
end
Output: $w^{(T+1)}$

11

Empirical Comparison

SGD v.s. batch solvers[4] on RCV1

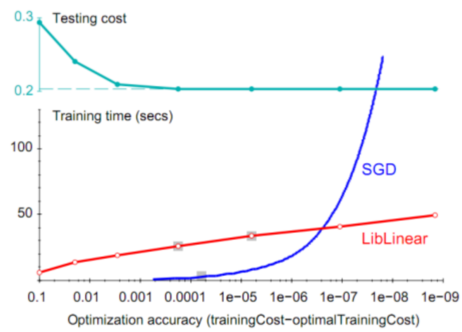
#Features	#Training examples
47, 152	781, 265

Algorithm	Time (secs)	Test Error
SMO (SVM^{light})	$\approx 16,000$	6.02%
Cutting Plane (SVM^{perf})	≈ 45	6.02%
SGD	< 1	6.02%

What is the magic?

[4]http://leon.bottou.org/projects/sgd

The Magic

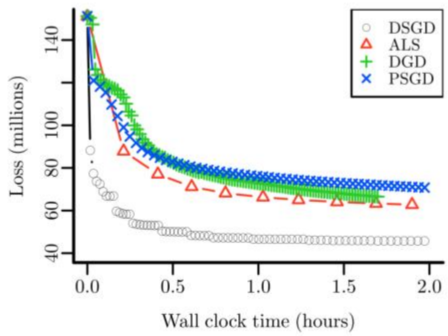


- SGD takes a long time to reach an highly accurate solution on the training data.
- However, for low generalization error (on test data) we may not need the training-set accuracy to be too high.

[4]http://leon.bottou.org/slides/largescale/lstut.pdf

Empirical Results

On Netflix [Gemulla et al., 2011]



DSGD, PSGD: Distributed SGD

ALS: Alternating least squares, one of the state-of-the-art batch solvers

DGD: Distributed GD

Popular SGD Variants

A non-exhaustive list

1. AdaGrad [Duchi et al., 2011]
2. Momentum [Rumelhart et al., 1988]
3. Nesterov's method [Nesterov et al., 1994]
4. AdaDelta: AdaGrad refined [Zeiler, 2012]
5. Rprop & Rmsprop [Tieleman and Hinton, 2012]
6. Stochastic Variance Reduced Gradient [Johnson and Zhang, 2013]
7. ADAM [Kingma and Ba, 2014]

All are empirically found effective in solving nonconvex problems (e.g., deep neural nets).

Demos [6]: Animation 0, 1, 2, 3

[6] https://www.reddit.com/r/MachineLearning/comments/2gopfa/visualizing_gradient_optimization_techniques/cklhott

Summarization Remarks

- SGD is extremely handy for large problems.
- It's only one of many handy tools
  - Alternatives: quasi-Newton (BFGS), Coordinate descent, ADMM, CG, etc.
  - How to choose? Depending on the problem structures.

16

Classification

- CLS 3a. Stochastic Gradient Decent
- CLS 3b. Evaluation Metrics

19

Testing-phase Decision Making

- Binary Classification
  - Using one binary classifier to make a yes/no decision for each test instance.
- Multi-label Classification
  - Using K OVA (one-vs-all) classifiers to make an independent yes/no decision per category for each test instance.
- Multi-class Classification
  - Using a multi-class model (e.g., softmax or kNN) to pick one label (out of K) for each test instance.

Binary Classification Evaluation
(e.g., sentiment classification)

- Given a test set of n instances, the system make n yes/no decisions, as summarized by a two-by-two contingency table

	$y = 1$	$y = 0$	
$\hat{y} = 1$	a	b	$a + b$
$\hat{y} = 0$	c	d	$c + d$
	$a + c$	$b + d$	$\mathrm{n} = a + b + c + d$

- Here a (true positive), b (false positive), c (false negative) and d (true negative) are the counts of test instances in the four categories

Evaluation Metrics for Binary Classifier

- Precision $p = \frac{a}{a+b}$
- Recall $r = \frac{a}{a+c}$
- F_1-measure $F_1 = \frac{2pr}{p+r} = \frac{2a}{2a+b+c}$
- Accuracy $Acc = \frac{a+d}{n}$
- Error $Err = \frac{b+c}{n}$

Why do we prefer F_1 over p and r?

- A trivial classifier always predicting "yes" for every input will have $r = 100\%$ but is totally useless.
- A trivial classifier always predicting the most popular label without checking the input is totally useless, but would have a high precision, especially when the labels are unbalanced in the data collection.
- F_1 is the harmonic average of p and r, which means that the F_1 value is high only if p and r are both high.

Harmonic Average

Def. $f(x_1, \cdots, x_n) = \frac{1}{\frac{1}{n}\left(\frac{1}{x_1}+\cdots+\frac{1}{x_n}\right)}$

$F_1(p,r) = \frac{2pr}{r+p} = \frac{1}{\frac{1}{2}\left(\frac{1}{p}+\frac{1}{r}\right)}$ ← Harmonic average of p and r

← Dominated by the smaller one

$F_\beta(p,r) = \frac{(1+\beta^2)pr}{r+\beta^2 p} = \frac{1}{\frac{1}{1+\beta^2}\frac{1}{p}+\frac{\beta^2}{1+\beta^2}\frac{1}{r}}$ ← Weighted harmonic average

F_β with different values of β

$$F_\beta(p,r) = \frac{1}{\frac{1}{1+\beta^2}\frac{1}{p}+\frac{\beta^2}{1+\beta^2}\frac{1}{r}}$$

$$F_{\beta=0} = \frac{1}{\frac{1}{1+0}\frac{1}{p}+\frac{0}{1+0}\frac{1}{r}} = p$$

$$F_{\beta=\infty} = \frac{1}{\frac{1}{1+\infty}\frac{1}{p}+\frac{\infty}{1+\infty}\frac{1}{r}} = r$$

Does $F_{\beta=0.5}$ favor r or p? What about $F_{\beta=2}$?

Multi-class Classification Evaluation
(e.g., OCR word recognition, Amazon product ID, etc.)

- The system assign one and only one label per test instance, which can be summarized using a K-by-K confusion matrix M.
- A toy example about 3-way movie classification to the right.
- $Acc = \frac{1}{n}\sum_{k=1}^{K} M_{kk} = \frac{1+76+1}{100} = 78\%$
- $Err = 1 - Acc = 22\%$

	$y = \text{good}$	$y = \text{soso}$	$y = \text{bad}$	
$\hat{y} = \text{good}$	1	3	1	5
$\hat{y} = \text{soso}$	20	70	0	90
$\hat{y} = \text{bad}$	3	1	1	5

Confusion Matrix

Multi-label Classification Evaluation
(e.g., wiki page classification by an OVA model per label)

- For $k = 1, 2, \ldots, K$, construct a contingency table per category as

	$y_k = 1$	$y_k = 0$
$\hat{y}_k = 1$	a_k	b_k
$\hat{y}_k = 0$	c_k	d_k

$$a_k + b_k + c_k + d_k = n$$

- Micro-averaging
  - Merge the K tables into one table by summing the corresponding cells
  - Use the merged table to compute p, r and F_1 (but not Acc or Err)
- Macro-averaging
  - Use each of the K tables to compute the category-specific p, r and F_1, then average them over categories as $\bar{p} = \frac{1}{K}\sum_{k=1}^{K} p_k$, and so on.

Micro-averaging vs. Macro-averaging

- Micro-averaging gives each test instance an equal weight
- Macro-average gives each category an equal weight
- Both are informative for method comparison

Issues with Accuracy or Error
(especially for rare categories)

- Consider when the number of labels is large, e.g., $K = 1000$.
- On average, 99.9% of the documents are negative instances for each OVA model.
- Consider a trivial classifier Mr. NO, with the contingency table like

	$y_k = 1$	$y_k = 0$
$\hat{y}_k = 1$	0	0
$\hat{y}_k = 0$	c_k	d_k

$$Acc = \frac{a_k + d_k}{n} = 99.9\%$$

$$Err = \frac{b_k + c_k}{n} = 0.1\%$$

- Take-home message: Focus on F_1 (or F_β) instead.