Accelerating Newton Optimization for Log-Linear Models through Feature Redundancy

Arpit Mathur
Soumen Chakrabarti
http://www.cse.iitb.ac.in/~soumen/

IIT Bombay

Log-linear models: Ubiquitous in machine learning

- Given training observations x and labels y
- Goal is to learn a weight vector $\beta \in \mathbb{R}^d$ to fit a conditional distribution

$$\Pr(Y = y|x) = \frac{\exp\left(\beta^\top f(x,y)\right)}{Z_\beta(x)} = \frac{\exp\left(\sum_j \beta_j f_j(x,y)\right)}{Z_\beta(x)},$$

- $f(x,y) \in \mathbb{R}^d$ is a feature vector (often $f_j(x,y) \geq 0$)
- $Z_\beta(x)$ is a normalizing constant
- Given instances $\{(x_i, y_i)\}$ find β to maximize $\sum_i \log \Pr(Y = y_i|x_i)$, i.e., to minimize

$$\ell(\beta) = \underbrace{-\sum_i \left(\beta^\top f(x_i, y_i) - \log Z_\beta(x_i)\right)}_{\text{Negative log likelihood of training data}} \underbrace{+\frac{1}{2\sigma^2}\sum_j \beta_j^2}_{\substack{\text{Gaussian prior} \\ \text{“Ridge penalty”}}}$$

Log-linear models: Ubiquitous in machine learning

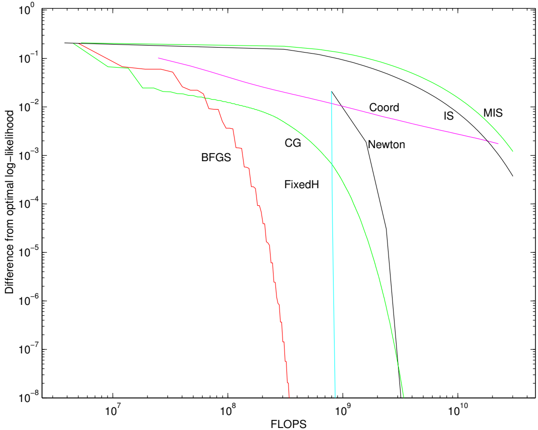
- Given training observations x and labels y
- Goal is to learn a weight vector $\beta \in \mathbb{R}^d$ to fit a conditional distribution

$$\Pr(Y = y|x) = \frac{\exp\left(\beta^\top f(x,y)\right)}{Z_\beta(x)} = \frac{\exp\left(\sum_j \beta_j f_j(x,y)\right)}{Z_\beta(x)},$$

- $f(x,y) \in \mathbb{R}^d$ is a feature vector (often $f_j(x,y) \geq 0$)
- $Z_\beta(x)$ is a normalizing constant
- Given instances $\{(x_i, y_i)\}$ find β to maximize $\sum_i \log \Pr(Y = y_i|x_i)$, i.e., to minimize

$$\ell(\beta) = \underbrace{-\sum_i \left(\beta^\top f(x_i, y_i) - \log Z_\beta(x_i)\right)}_{\text{Negative log likelihood of training data}} \underbrace{+\frac{1}{2\sigma^2}\sum_j \beta_j^2}_{\substack{\text{Gaussian prior} \\ \text{“Ridge penalty”}}}$$

Training performance of optimizers



From [Minka 2003]
x-axis: FLOPS
y-axis: difference from optimal objective

- Iterative scaling no longer in the race
- Newton method costs too many FLOPS per iteration
- Sparse Newton methods (LBFGS) lead the pack

Redundancy in f and β

- Log-linear models allow us to use large number of potentially redundant features f_j
- E.g., `hasDigit`, `isFourDigits`, `hasCap`, `isAllCaps`, `isAbbrev`
  - $\texttt{isFourDigits} \Rightarrow \texttt{hasDigit}$
  - $\texttt{isAllCaps} \Rightarrow \texttt{hasCap}$
  - $\Pr(\texttt{isAllCaps}|\texttt{isAbbrev}) \gg \Pr(\texttt{isAllCaps})$
- Combined with dictionaries, often leads to millions of features in NLP tasks
- Convenient, but training bottleneck
- Optimizer has to deal with objective that is more complicated than really necessary

Ridge penalty leads to redundant models

- For every occurrence of feature j, add new feature j'
- Features f_j and $f_{j'}$ are perfectly correlated
- j' adds no predictive power to any classifier
- LR without Ridge penalty can keep β_j unchanged and set $\beta_{j'} = 0$ to get same training accuracy

$$\ell(\beta) = \underbrace{-\sum_i \left(\sum_j \beta_j f_j(x_i, y_i) - \log Z_\beta(x_i) \right)}_{\text{data log likelihood}} \underbrace{+ \frac{1}{2\sigma^2} \sum_j \beta_j^2}_{\text{Ridge penalty}}$$

- With Ridge penalty, better to "split the evidence"
- Set $\beta_j^{\text{new}} = \beta_{j'}^{\text{new}} = \beta_j^{\text{old}}/2$
- Data log likelihood remains unchanged whenever $\beta_j^{\text{new}} + \beta_{j'}^{\text{new}} = \beta_j^{\text{old}}$
- $2\,(\beta_j^{\text{old}}/2)^2 = (\beta_j^{\text{old}})^2/2 < (\beta_j^{\text{old}})^2 + 0^2$

Ridge penalty leads to redundant models

- For every occurrence of feature j, add new feature j'
- Features f_j and $f_{j'}$ are perfectly correlated
- j' adds no predictive power to any classifier
- LR without Ridge penalty can keep β_j unchanged and set $\beta_{j'} = 0$ to get same training accuracy

$$\ell(\beta) = \underbrace{-\sum_i \left(\sum_j \beta_j f_j(x_i, y_i) - \log Z_\beta(x_i) \right)}_{\text{data log likelihood}} \underbrace{+ \frac{1}{2\sigma^2} \sum_j \beta_j^2}_{\text{Ridge penalty}}$$

- With Ridge penalty, better to "split the evidence"
- Set $\beta_j^{\text{new}} = \beta_{j'}^{\text{new}} = \beta_j^{\text{old}}/2$
- Data log likelihood remains unchanged whenever $\beta_j^{\text{new}} + \beta_{j'}^{\text{new}} = \beta_j^{\text{old}}$
- $2\,(\beta_j^{\text{old}}/2)^2 = (\beta_j^{\text{old}})^2/2 < (\beta_j^{\text{old}})^2 + 0^2$

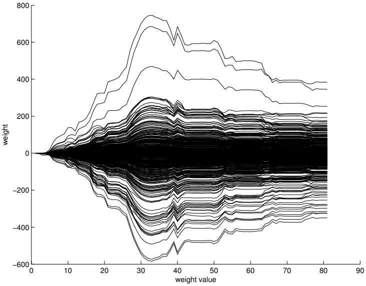
A new form of model parsimony?

- Ridge penalty encourages small $|\beta_j|$
- Lasso penalty reduces $\|\beta\|_1$, makes solution sparse, i.e., encourages $\beta_j = 0$
- In view of (approximately) redundant but informative features, a new form of model parsimony is that $\beta \in \mathbb{R}^d$ has far fewer degrees of freedom than d

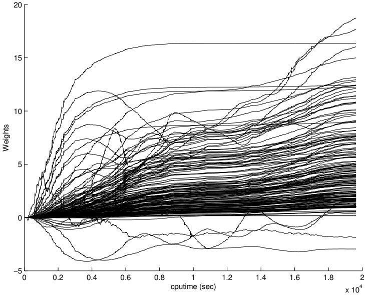
- Let $\gamma \in \mathbb{R}^{d'}$ be the "hidden model" with $d' \ll d$
- In general β and γ can be related in very complex ways
- A very simple starting point is that elements of γ are copied into elements of β

There is no assumption about clusters in the *data*

Behavior of β with iterations



Topic classification (LR)



Named entity tagging (CRF)

- "Big Bang" moment followed by gradual evolution
- Trajectory cross-overs become rare quite quickly
- Can approximate adjacent trajectories with a band for a short time
- However, cannot ignore cross-overs all the time

Idea: Optimize in clustered subspace

Let initial approximation to the solution be $\beta \in \mathbb{R}^d$
for *numRounds* rounds **do**
  for *fineIters* iterations **do**
    $[f, g] \leftarrow$ `ObjAndGrad`(β) {wait until Big Bang over}
    $\beta \leftarrow$ `fineBFGS`(β, f, g)
  $d' \leftarrow d / clusterFactor$
  Feature cluster map *cmap* $\leftarrow$ `Cluster`(β, d')
  $\gamma \leftarrow$ `ProjectDown`$(\beta, cmap)$ $\{\gamma \in \mathbb{R}^{d'}, d' \ll d\}$
  for *coarseIters* iterations **do**
    $[\phi, \delta] \leftarrow$ `ObjAndGrad`$(\gamma, cmap)$
    $\gamma \leftarrow$ `coarseBFGS`(γ, ϕ, δ) {faster than `fineBFGS`}
  $\beta \leftarrow$ `ProjectUp`$(\gamma, cmap)$
while not converged **do**
  $[f, g] \leftarrow$ `ObjAndGrad`(β)
  $\beta \leftarrow$ `fineBFGS`(β, f, g) {patch up remaining problems}

Idea: Optimize in clustered subspace

Let initial approximation to the solution be $\beta \in \mathbb{R}^d$
for *numRounds* rounds **do**
  for *fineIters* iterations **do**
    $[f, g] \leftarrow$ `ObjAndGrad`(β) {wait until Big Bang over}
    $\beta \leftarrow$ `fineBFGS`(β, f, g)
  $d' \leftarrow d/clusterFactor$
  Feature cluster map *cmap* $\leftarrow$ `Cluster`(β, d')
  $\gamma \leftarrow$ `ProjectDown`$(\beta, cmap)$ $\{\gamma \in \mathbb{R}^{d'}, d' \ll d\}$
  for *coarseIters* iterations **do**
    $[\phi, \delta] \leftarrow$ `ObjAndGrad`$(\gamma, cmap)$
    $\gamma \leftarrow$ `coarseBFGS`(γ, ϕ, δ) {faster than `fineBFGS`}
  $\beta \leftarrow$ `ProjectUp`$(\gamma, cmap)$
while not converged **do**
  $[f, g] \leftarrow$ `ObjAndGrad`(β)
  $\beta \leftarrow$ `fineBFGS`(β, f, g) {patch up remaining problems}

Idea: Optimize in clustered subspace

Let initial approximation to the solution be $\beta \in \mathbb{R}^d$
for *numRounds* rounds **do**
  for *fineIters* iterations **do**
    $[f, g] \leftarrow$ `ObjAndGrad`(β) {wait until Big Bang over}
    $\beta \leftarrow$ `fineBFGS`(β, f, g)
  $d' \leftarrow d / clusterFactor$
  Feature cluster map *cmap* $\leftarrow$ `Cluster`(β, d')
  $\gamma \leftarrow$ `ProjectDown`$(\beta,$ *cmap*$)$ $\{\gamma \in \mathbb{R}^{d'}, d' \ll d\}$
  for *coarseIters* iterations **do**
    $[\phi, \delta] \leftarrow$ `ObjAndGrad`$(\gamma,$ *cmap*$)$
    $\gamma \leftarrow$ `coarseBFGS`(γ, ϕ, δ) {faster than `fineBFGS`}
  $\beta \leftarrow$ `ProjectUp`$(\gamma,$ *cmap*$)$
while not converged **do**
  $[f, g] \leftarrow$ `ObjAndGrad`(β)
  $\beta \leftarrow$ `fineBFGS`(β, f, g) {patch up remaining problems}

Idea: Optimize in clustered subspace

Let initial approximation to the solution be $\beta \in \mathbb{R}^d$
for *numRounds* rounds **do**
  for *fineIters* iterations **do**
    $[f, g] \leftarrow$ `ObjAndGrad`(β) {wait until Big Bang over}
    $\beta \leftarrow$ `fineBFGS`(β, f, g)
  $d' \leftarrow d / clusterFactor$
  Feature cluster map *cmap* $\leftarrow$ `Cluster`(β, d')
  $\gamma \leftarrow$ `ProjectDown`$(\beta,$ *cmap*$)$ $\{\gamma \in \mathbb{R}^{d'}, d' \ll d\}$
  for *coarseIters* iterations **do**
    $[\phi, \delta] \leftarrow$ `ObjAndGrad`$(\gamma,$ *cmap*$)$
    $\gamma \leftarrow$ `coarseBFGS`(γ, ϕ, δ) {faster than `fineBFGS`}
  $\beta \leftarrow$ `ProjectUp`$(\gamma,$ *cmap*$)$
while not converged **do**
  $[f, g] \leftarrow$ `ObjAndGrad`(β)
  $\beta \leftarrow$ `fineBFGS`(β, f, g) {patch up remaining problems}

Idea: Optimize in clustered subspace

Let initial approximation to the solution be $\beta \in \mathbb{R}^d$
for *numRounds* rounds **do**
  for *fineIters* iterations **do**
    $[f, g] \leftarrow$ `ObjAndGrad`(β) {wait until Big Bang over}
    $\beta \leftarrow$ `fineBFGS`(β, f, g)
  $d' \leftarrow d/clusterFactor$
  Feature cluster map *cmap* $\leftarrow$ `Cluster`(β, d')
  $\gamma \leftarrow$ `ProjectDown`$(\beta, cmap)$ $\{\gamma \in \mathbb{R}^{d'}, d' \ll d\}$
  for *coarseIters* iterations **do**
    $[\phi, \delta] \leftarrow$ `ObjAndGrad`$(\gamma, cmap)$
    $\gamma \leftarrow$ `coarseBFGS`(γ, ϕ, δ) {faster than `fineBFGS`}
  $\beta \leftarrow$ `ProjectUp`$(\gamma, cmap)$
while not converged **do**
  $[f, g] \leftarrow$ `ObjAndGrad`(β)
  $\beta \leftarrow$ `fineBFGS`(β, f, g) {patch up remaining problems}

Cluster

- Sort β_j, identify d' contiguous blocks to minimize square error within clusters
- For $j = 1, \ldots, d$, $1 \le cmap(j) \le d'$ gives the cluster index of j

ProjectDown

- β always appears in objective as $\beta^\top f$
- Naturally suggests that initial γ_k be the average of β_js where $cmap(j) = k$

ProjectUp

- We just copy γ_k to all β_j where $cmap(j) = k$
- Can perhaps do better

`ObjAndGrad`(β) to `ObjAndGrad`$(\gamma, \textit{cmap})$

- Given code for $\ell(\beta)$ and $\nabla_\beta \ell$, want code for $\ell(\gamma)$ and $\nabla_\gamma \ell$
- Add parameter *cmap* to `ObjAndGrad`(β)
- In $\ell(\beta)$, β always appears as $\beta^\top f$, so replace $\sum_j \beta_j f_j$ by $\sum_j \gamma_{cmap(j)} f_j$
- To compute $\nabla_\gamma \ell$, observe that

$$\frac{\partial \ell}{\partial \gamma_k} = \sum_j \frac{\partial \ell}{\partial \beta_j} \frac{\partial \beta_j}{\partial \gamma_k} = \sum_j \frac{\partial \ell}{\partial \beta_j} \begin{cases} 1 & cmap(j) = k \\ 0 & \text{otherwise} \end{cases}$$

- $\therefore \nabla_\gamma \ell = A \nabla_\beta \ell$, where $A \in \mathbb{N}^{d' \times d}$ is defined as

$$A(k, j) = \begin{cases} 1 & cmap(j) = k \\ 0 & \text{otherwise} \end{cases}$$

- Can efficiently push down computation of $\nabla_\gamma \ell$ by trivial transformations of $\nabla_\beta \ell$ code

Three tuned performance parameters

clusterFactor $= d/d'$

- d' too large $\Rightarrow$ not enough redundancy removal
- d' too small $\Rightarrow$ `coarseBFGS` drifts from true solution

numRounds

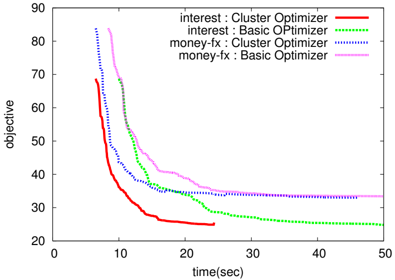
- *numRounds* $= 1$ or 2 almost always optimal

coarseIters

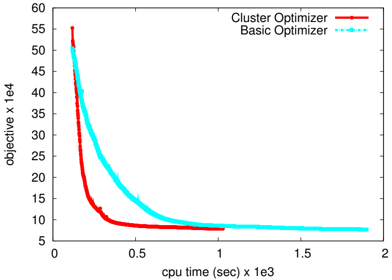
- `coarseBFGS` terminated if last 5 iterations improve objective by less than 0.01%

Training is rarely a one-time job in applications: training data, labels, features, etc. keep changing

Sample results: Objective vs. time



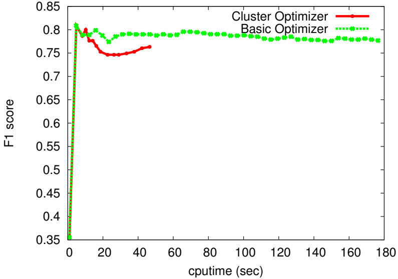
LR, two Reuters topics



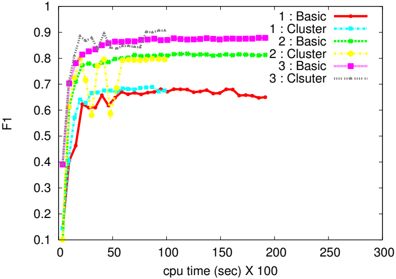
CRF tagger, one entity type

- Clustered optimizer reduces objective much faster
- Careful analysis shows improvement is mainly due to objective in $\mathbb{R}^{d'}$ which is simpler than objective in $\mathbb{R}^{d}$

Sample results: Test accuracy vs. training time



LR, one Reuters topic



CRF tagger, three entity types

- Test accuracy may stabilize before training objective
- Clustered optimizer shows earlier test accuracy saturation
- Sometimes significantly more accurate early on
- Overdoing `coarseBFGS` may damage test F1 momentarily
- Final patch-up fixes matters eventually, but wastes time

Sample results: Effect of *clusterFactor* choice

ClassName	*clusterFactor*	TrainTime (s)
Wheat	10	16.0
	20	**15.7**
	50	17.3
Money-fx	10	64.3
	20	44.5
	50	**39.4**
Interest	10	61.3
	20	**44.3**
	50	87.6

- 10–50 adequate range to explore
- Recommend starting with large *clusterFactor* and reducing it if final patch-up is slow

Summary

- Log-linear models ubiquitous in machine learning
- State-of-the-art optimizer: LBFGS
- Redundant features: Convenient, but training bottleneck
- Very simple idea: Cluster features by β, optimize in $\gamma \in \mathbb{R}^{d'}$ instead of $\beta \in \mathbb{R}^d$, $d' \ll d$, recluster periodically
- Experiments with logistic regression and CRFs
- Speeds up between $2\times$ and $12\times$, typical 3–$5\times$
- No noticeable degradation of accuracy of trained model

Future work

- More elaborate maps between β and γ
- Extend to other model penalty functions
- Auto-tuning of performance parameters

Summary

- Log-linear models ubiquitous in machine learning
- State-of-the-art optimizer: LBFGS
- Redundant features: Convenient, but training bottleneck
- Very simple idea: Cluster features by β, optimize in $\gamma \in \mathbb{R}^{d'}$ instead of $\beta \in \mathbb{R}^{d}$, $d' \ll d$, recluster periodically
- Experiments with logistic regression and CRFs
- Speeds up between $2\times$ and $12\times$, typical 3–$5\times$
- No noticeable degradation of accuracy of trained model

Future work

- More elaborate maps between β and γ
- Extend to other model penalty functions
- Auto-tuning of performance parameters