CPSC 540: Machine Learning

Coordinate Optimization

Mark Schmidt

University of British Columbia

Winter 2019

Last Time: Structured Regularization

- We discussed total variation regularization,

$$\operatorname*{argmin}_{w} f(w) + \sum_{(i,j)\in E} \lambda_{ij}(w_i - w_j)^2,$$

if we want w_i values to be similar across nodes in a graph.

- We discussed structured sparsity,

$$\operatorname*{argmin}_{w} f(w) + \sum_{g\in\mathcal{G}} \lambda_g \|w_g\|,$$

where overlapping groups can enforce patterns of sparsity.

- These regularizers aren't "simple", but several solvers exist.
  - Gradient descent if smooth, inexact proximal gradient for non-smooth.

$UV^\top$ Parameterization for Matrix Problems

- We discussed nuclear norm regularization problems,

$$\underset{W \in \mathbb{R}^{d \times k}}{\text{argmin}}\ f(W) + \lambda \|W\|_*,$$

which gives a solution with a low rank representation $W = UV^\top$.

- But standard algorithms are too costly in many applications.
    - We often can't store W.

- Many recent approaches directly minimize under $UV^\top$ parameterization,

$$\underset{U \in \mathbb{R}^{d \times R}, V \in \mathbb{R}^{k \times R}}{\text{argmin}}\ f(UV^\top) + \lambda_U \|U\|_F^2 + \lambda_V \|V\|_F^2,$$

and just regularize U and V (here we're using the Frobenius matrix norm).

$UV^\top$ Parameterization for Matrix Problems

- We used this approach in 340 for latent-factor models,

$$f(W,Z) = \frac{1}{2}\|ZW - X\|_F^2 + \frac{\lambda_1}{2}\|Z\|_F^2 + \frac{\lambda_2}{2}\|W\|_F^2.$$

- We can sometimes prove these non-convex re-formulation give a global solution.
    - Includes PCA.
- In other cases, people are working hard on finding assumptions where this is true.
    - These assumptions are typically unrealistically strong.
    - But it works well enough in practice that practitioners don't seem to care.

Transductive Learning

- Our usual supervised learning framework:

$$X = \begin{bmatrix} 0 & 0.7 & 0 & 0.3 & 0 & 0 \\ 0.3 & 0.7 & 0 & 0.6 & 0 & 0.01 \\ 0 & 0 & 0 & 0.8 & 0 & 0 \\ 0.3 & 0.7 & 1.2 & 0 & 0.10 & 0.01 \end{bmatrix}, \quad y = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \end{bmatrix}.$$

- In transductive learning, we also have unlabeled examples,

$$\bar{X} = \begin{bmatrix} 0.3 & 0 & 1.2 & 0.3 & 0.10 & 0.01 \\ 0.6 & 0.7 & 0 & 0.3 & 0 & 0.01 \\ 0 & 0.7 & 0 & 0.6 & 0 & 0 \\ 0.3 & 0.7 & 0 & 0 & 0.20 & 0.01 \end{bmatrix},$$

and our goal is only to label these particular examples.

  - We don't worry about performance on other potential test examples.

Transductive Learning

- Transductive learning framework:
  1. We have n labeled examples (x^i, y^i).
  2. We have t unabeled examples $\bar{x}^i$ that we want to label.

- This arises a lot:
  - Usually getting unlabeled data is easy but getting labeled data is hard $(t >> n)$.
  - Typically situation: small number of labeled and huge number of unlabeled.

- Sometimes classifying the data is an intermediate step:
  - Goal is to ulimately use labeled examples to do something else.
  - "I can label a small number of examples, if it helps labeling them all".

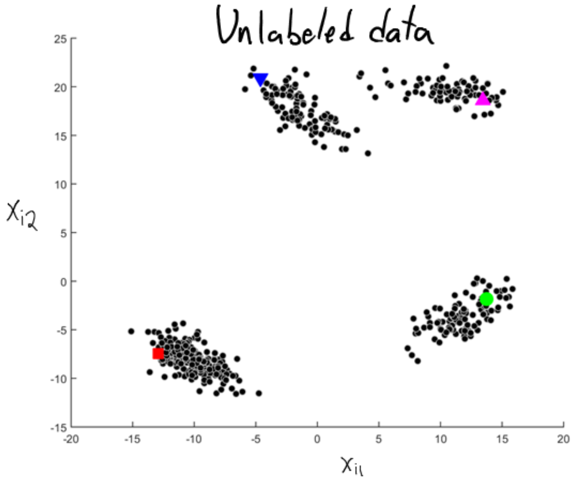
- Sometimes it's not possible to obtain labels for any x^i.
  - Predicting gene functions is limited by what we can measure.

Transductive Learning vs. (Semi-)Supervised Learning

- Transductive learning is a special case semi-supervised learning (SSL).
  - Learning with labeled and unlabeled examples (we'll come back to SSL later).
- But transductive SSL has an unusual measure of performance:
  - We don't worry about "test error" (performance on all possible examples).
  - We only care about error for our "test" examples $\bar{x}^i$.
- Any supervised or semi-supervised method can be used for transduction.
  - Fit model, then apply it to unlabeled examples.
- But in transductive learning, we don't need a model that can predict on new $\tilde{x}^i$.
  - Some methods don't fit a generic model for mapping from x^i to y^i.

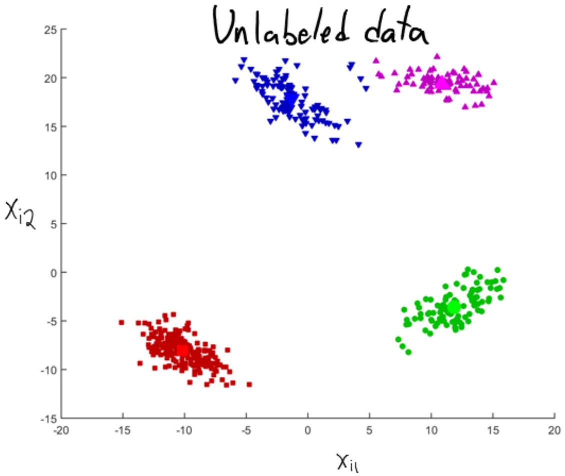
Transductive Learning

- Why should unlabeled data tell us anything about labels?
  - Usually, we assume that similar features $\rightarrow$ similar labels.



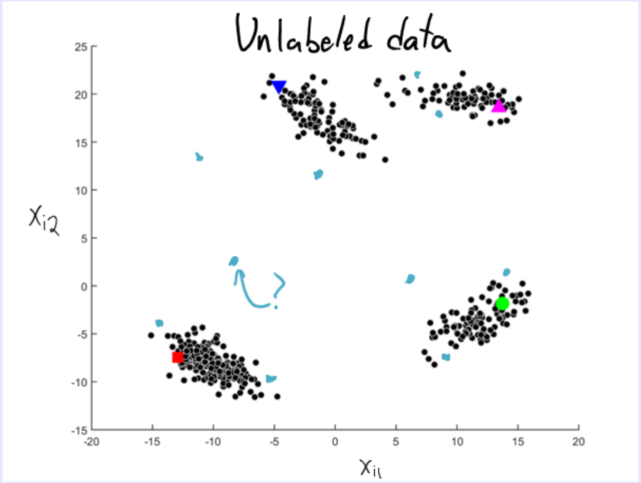
Transductive Learning

- Why should unlabeled data tell us anything about labels?
  - Usually, we assume that similar features → similar labels.



Digression: Transductive vs. Inductive SSL

- In transductive learning we don't need to be able to predict on new examples.
  - In inductive semi-supervised learning goal is to predict well on new examples.



Label Propagation (Graph-Based SSL)

- A weird idea: treat the $\bar{y}^i$ as variables that we can optimize.
  - Now optimize the $\bar{y}^i$ to encourage that "similar features have similar labels".

- Label propagation ("graph-based SSL") method:
  - Define weights w_{ij} saying how similar labeled example i is to unlabled example j.
  - Define weights $\bar{w}_{ij}$ saying how similar unlabeled example i is to unlabeled example j.
  - Find labels $\bar{y}^i$ minimizing a measure of total variation on the label space:

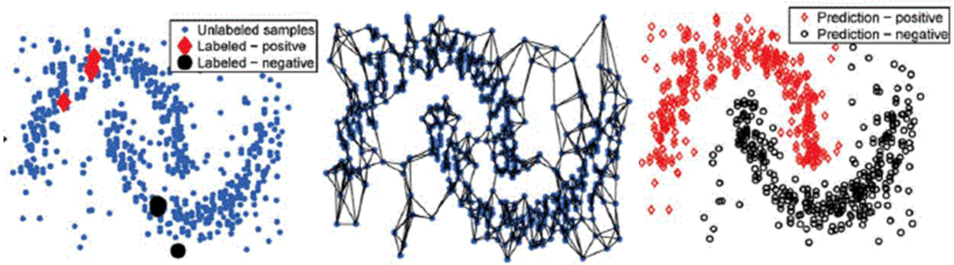
$$\underset{\bar{y}\in\mathbb{R}^t}{\text{argmin}} \sum_{i=1}^{n}\sum_{j=1}^{t} w_{ij}(y^i - \bar{y}^j)^2 + \frac{1}{2}\sum_{i=1}^{t}\sum_{j=1}^{t}\bar{w}_{ij}(\bar{y}^i - \bar{y}^j)^2.$$

  - First term: unlabeled example should get similar labels to "close" labeled examples.
    - "If x^i and $\bar{x}^j$ are similar, then $\bar{y}^j$ should be similar to y^i."
  - Second term: similar unlabeled examples should have similar labels.
    - "Label information 'propagates' through the graph of $\bar{y}^i$ values".

Label Propagation (Graph-Based SSL)

- Label propagation is often surprisingly effective (even with few labeled examples).

- A common choice of the weights (many variations exist):
  - Find the k-nearest neighbours of each example (among labeled and unlabeled).
  - Set w_{ij} and $\bar{w}_{ij}$ to 0 if nodes i and j aren't neighbours.
  - Otherwise, set these to some measure of similarity between features.



http://www.ee.columbia.edu/ln/dvmm/pubs/publications.html

Label Propagation for YouTube Tagging and Bioinformatics

- Label propagation doesn't necessarily need features.
  - Consider assigning "tags" to YouTube vidoes (e.g., "cat").

www.youtube.com

  - Construct a graph based on sequence of videos that people watch.
    - Give high weight if video 'A' is often followed/preceded by video 'B'.
  - Use label propagation to tag all videos.

- Becoming popular in bioinformatics:
  - Label a subset of genes using manual experiments.
  - Find out which genes interact using more manual experiments.
  - Predict function/location/etc. of genes using label propagation.

Label Propagation Variations

- Many variations on label propagation exist:
  - Different ways to choose the graph/weights.
  - Multi-class versions,

$$\underset{\bar{Y} \in \mathbb{R}^{t \times k}}{\text{argmin}} \sum_{i=1}^{n} \sum_{j=1}^{t} w_{ij} \|y^i - \bar{y}^j\|^2 + \frac{1}{2} \sum_{i=1}^{t} \sum_{j=1}^{t} \bar{w}_{ij} \|\bar{y}^i - \bar{y}^j\|^2.$$

  - Other measures of similarity/distance,

$$\underset{\bar{y} \in \mathbb{R}^{t}}{\text{argmin}} \sum_{i=1}^{n} \sum_{j=1}^{t} f_{ij}(y^i, \bar{y}^j) + \sum_{i=1}^{t} \sum_{j=1}^{t} f_{ij}(\bar{y}^i, \bar{y}^j).$$

  - Variants where the given labels y^i are also variables (as they might be wrong).
    - Weight gives how much you trust original label.
  - Variants where the unlabeled $\bar{y}^i$ are regularized towards a default value.
    - Can reflect that example is really far from any labeled examples.

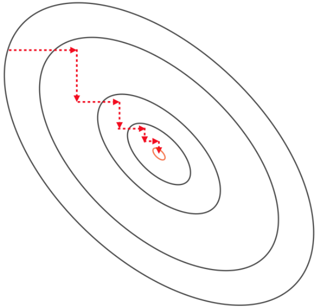
Outline

Beyond Gradient Descent

- For high-dimensional problems we often prefer gradient descent over Newton.
  - Gradient descent requires far more iterations.
  - But iteration cost is only linear in d.
- For very large datasets, even gradient descent iterations can be too slow.
  - If iteration cost is $O(nd)$, we may only be able to do a small number of iterations.
- Two common strategies for yielding even cheaper iterations:
  - Coordinate optimization (today).
  - Stochastic gradient (next time).

Coordinate Optimization

- Each iteration of coordinate optimization only updates on variable:



- For example, on iteration k we select a variable j_k and set

$$w_{j_k}^{k+1} = w_{j_k}^{k} - \alpha_k \nabla_{j_k} f(w^k),$$

  a gradient descent step on coordinate j_k (other w_j stay the same).
  - This variation is called coordinate descent (many variations exist).

Why use Coordinate Descent?

- Theoretically, coordinate descent is a provably bad algorithm:
  - The convergence rate is slower than gradient descent.
  - The iteration cost can be similar to gradient descent.
    - Computing 1 partial derivative may have same cost as computing gradient.
- But it is widely-used in practice:
  - Nothing works better for certain problems.
  - Certain fields think it is the "ultimate" algorithm.
- ???
- Renewed theoretical interest began with a paper by Nesterov in 2010:
  - Showed global convergence rate for randomized coordinate selection.
  - Coordinate descent is faster than gradient descent if iterations are d times cheaper.

Problems Suitable for Coordinate Optimization

- For what functions is coordinate descent d times faster than gradient descent?
- The simplest example is separable functions,

$$f(w) = \sum_{j=1}^{d} f_j(w_j),$$

- Here f is the sum of an f_j applied to each w_j, like $f(x) = \|w - v\|^2 = \sum_{j=1}^{d}(w_j - v_j)^2$.
- Cost of gradient descent vs. coordinate descent:
  - Gradient descent costs $O(d)$ to compute each $f'(w_j^k)$.
  - Coordinate descent costs $O(1)$ to compute the *one* $f'_{j_k}(w_{j_k}^k)$.
- In fact, for separable functions you should only use coordinate optimization.
  - The variables w_j have "separate" effects, so can be minimized independently.

Problems Suitable for Coordinate Optimization

- A more interesting example is pairwise-separable functions,

$$f(w) = \sum_{i=1}^{d}\sum_{j=1}^{d} f_{ij}(w_i, w_j),$$

  which depend on a function of each pair of variables.

- An example is label propagation.
  - Also includes any quadratic function.

- Cost of gradient descent vs. coordinate descent:
  - Gradient descent costs $O(d^2)$ to compute each f'_{ij}.
  - Coordinate descent costs $O(d)$ to compute d values of f'_{ij}.

Problems Suitable for Coordinate Optimization

- Our label propagation example looked a bit more like this:

$$f(w) = \sum_{j=1}^{d} f_j(w_j) + \sum_{(i,j)\in E} f_{ij}(w_i, w_j),$$

  where E is a set of (i,j) pairs ("edges" in a graph).

- Adding a separable function doesn't change costs.
  - We could just combine the f_j with one f_{ij}.

- Restricting (i,j) to E makes gradient descent cheaper:
  - Now costs $O(|E|)$ to compute gradient.
  - Coordinate descent could also cost $O(|E|)$ if degree of j_k is $O(|E|)$.

- Coordinate descent is still d times faster in expectation if you randomly pick j_k.
  - Each f'_{ij} is needed with probability $2/d$.
  - So expected cost of $O(|E|/d)$ to compute one partial derivative.

Label Propagation with Coordinate Optimization

- For the binary label propagation objective,

$$\underset{\bar{y}\in\mathbb{R}^t}{\operatorname{argmin}} \sum_{i=1}^{n}\sum_{j=1}^{t} w_{ij}(y^i - \bar{y}^j)^2 + \frac{1}{2}\sum_{i=1}^{t}\sum_{j=1}^{t}\bar{w}_{ij}(\bar{y}^i - \bar{y}^j)^2,$$

  we can exactly optimize one coordinate given the others.

- Taking the derivative and setting it to 0 gives:

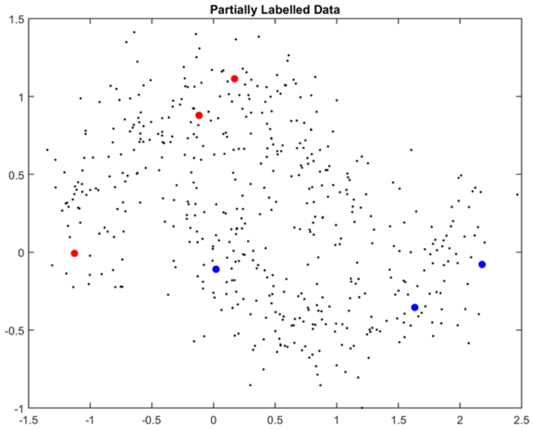
$$\bar{y}^i = \frac{\sum_{j=1}^{n} w_{ij}y^j + \sum_{j\neq i}\bar{w}_{ij}\bar{y}^j}{\sum_{j=1}^{n} w_{ij} + \sum_{j\neq i}\bar{w}_{ij}},$$

  where I'm assuming $\bar{w}_{ij} = \bar{w}_{ji}$ (otherwise, you replace both by their average).

- So coordinate optimization takes weighted average of neighbours.

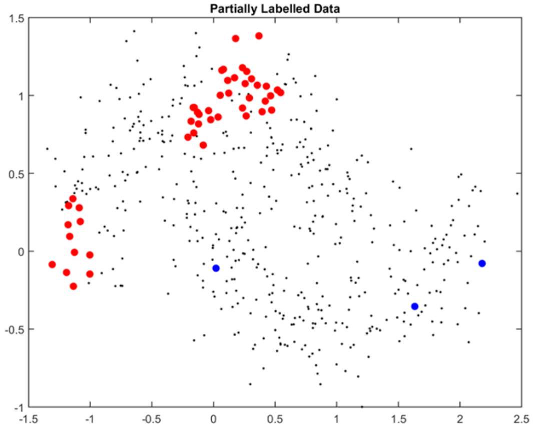
Label Propagation with Coordinate Optimization

- Label propagation with coordinate optimization in action:



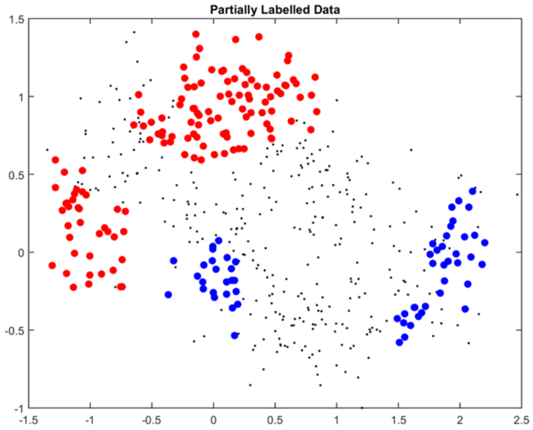
Label Propagation with Coordinate Optimization

- Label propagation with coordinate optimization in action:



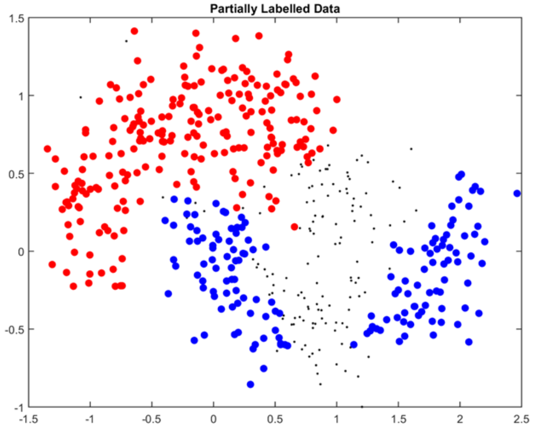
Label Propagation with Coordinate Optimization

- Label propagation with coordinate optimization in action:



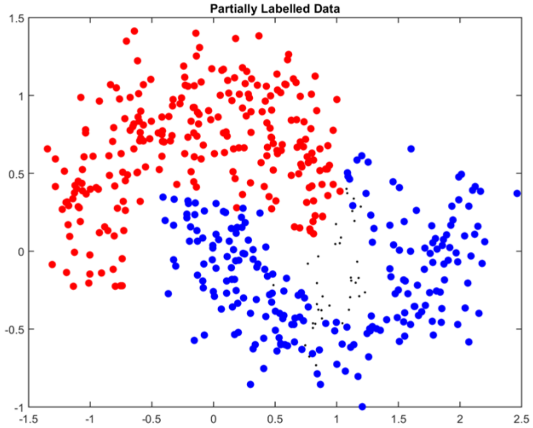
Label Propagation with Coordinate Optimization

- Label propagation with coordinate optimization in action:



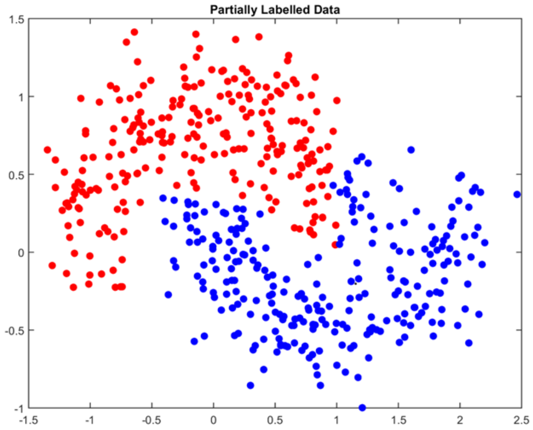
Label Propagation with Coordinate Optimization

- Label propagation with coordinate optimization in action:



Label Propagation with Coordinate Optimization

- Label propagation with coordinate optimization in action:



Analyzing Coordinate Descent

- To analyze coordinate descent, we can write it as

$$w^{k+1} = w^k - \alpha_k e_{j_k} \nabla_{j_k} f(w^k),$$

where "elementary vector" e_j has a zero in every position except j,

$$e_3^\top = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

- We usually assume that each $\nabla_j f$ is L-Lipshitz ("coordinate-wise Lipschitz"),

$$|\nabla_j f(w + \gamma e_j) - \nabla_j f(w)| \leq L|\gamma|,$$

which for $\mathcal{C}^2$ functions is equivalent to $|\nabla^2_{jj} f(w)| \leq L$ for all i.

(diagonals of Hessian are bounded)

- This is not a stronger assumption:
  - If the gradient is L-Lipschitz then it's also coordiante-wise L-Lipschitz.

Convergence Rate of Coordinate Optimization

- Coordinate-wise Lipschitz assumption implies a coordinate-wise descent lemma,

$$f(w^{k+1}) \leq f(w^k) + \nabla_j f(w^k)(w^{k+1} - w^k)_j + \frac{L}{2}(w^{k+1} - w^k)_j^2,$$

  for any w^{k+1} and w^k that only differ in coordinate j.

- With $\alpha_k = 1/L$ (for simplicity), plugging in $(w^{k+1} - w^k) = -(1/L)e_{j_k}\nabla_{j_k} f(w^k)$ gives

$$f(w^{k+1}) \leq f(w^k) - \frac{1}{2L}|\nabla_{j_k} f(w^k)|^2,$$

  a progress bound based on only updating coordinate j_k.

- If we did optimal update (as in label propagation), this bound would still hold.
  - Optimal update decreases f by at least as much as any other update.

Convergence Rate of Randomized Coordinate Optimization

- Our bound for updating coordinate j_k is

$$f(w^{k+1}) \leq f(w^k) - \frac{1}{2L}|\nabla_{j_k} f(w^k)|^2,$$

so progress depends on which j_k that we choose.

- Let's consider expected progress with random selection of j_k,

$$\begin{aligned}
\mathbb{E}[f(w^{k+1})] &\leq \mathbb{E}\left[f(w^k) - \frac{1}{2L}|\nabla_{j_k} f(w^k)|^2\right] && \text{(expectation wrt } j_k \text{ given } w^k\text{)}\\
&= \mathbb{E}[f(w^k)] - \frac{1}{2L}\mathbb{E}[|\nabla_{j_k} f(w^k)|^2] && \text{(linearity of expectation)}\\
&= f(w^k) - \frac{1}{2L}\sum_{j=1}^{d} p(j_k = j)|\nabla_j f(w^k)|^2 && \text{(definition of expectation)}
\end{aligned}$$

Convergence Rate of Randomized Coordinate Optimization

- The bound from the previous slide is

$$E[f(w^{k+1})] \leq f(w^k) - \frac{1}{2L}\sum_{j=1}^{d} p(j_k = j)|\nabla_j f(w^k)|^2.$$

- Let's choose j_k uniformly in this bound, $p(j_k = j) = 1/d$.

$$\begin{aligned}\mathbb{E}[f(w^{k+1})] &\leq f(w^k) - \frac{1}{2L}\sum_{j=1}^{d}\frac{1}{d}|\nabla_j f(w^k)|^2 \\ &= f(w^k) - \frac{1}{2dL}\sum_{j=1}^{d}|\nabla_j f(w^k)|^2 \\ &= f(w^k) - \frac{1}{2dL}\|\nabla f(w^k)\|^2.\end{aligned}$$

Convergence Rate of Randomized Coordinate Optimization

- Our guaranteed progress bound for randomized coordinate optimization,

$$\mathbb{E}[f(w^{k+1}))] \leq f(w^k) - \frac{1}{2dL}\|\nabla f(w^k)\|^2.$$

- If we use strongly convexity or PL and recurse carefully (see bonus) we get

$$\mathbb{E}[f(w^k)] - f^* \leq \left(1 - \frac{\mu}{dL}\right)^k [f(w^0) - f^*].$$

which means we expect to need $O\left(d\frac{L}{\mu}\log(1/\epsilon)\right)$ iterations.

- Remember that gradient descent needs $O\left(\frac{L}{\mu}\log(1/\epsilon)\right)$ iterations.

- So coordinate optimization needs d-times as many iterations?

Randomized Coordinate Optimization vs. Gradient Descent

- If coordinate descent step are d-times cheaper then both algorithms need
$$O\left(\frac{L}{\mu}\log(1/\epsilon)\right),$$
in terms of gradient descent iteration costs.

- So why prefer coordinate optimization?

- The Lipschitz constants L are different.
    - Gradient descent uses L_f and coordinate optimization uses L_c.
    - L_c is maximum gradient changes if you change *one* coordinate.
    - L_f is maximum gradient changes if you change *all* coordinates.

- Since $L_c \leq L_f$, coordinate optimization is faster.
    - By a factor that could be as large as d.
    - The gain is because coordinate descent allows bigger step-sizes.

Summary

- Transductive learning:
  - Given labeled and unlabeled examples, label the unlabeled examples.
- Label propagation:
  - Transductive learning method minimizing variation in the label space.
- Coordinate optimization: updating one variable at a time.
  - Efficient if updates are d-times cheaper than gradient descent.

- Next time: the most important algorithm in machine learning.

Applying Expected Bound Recursively

- Our guaranteed progress bound for randomized coordinate optimization,

$$\mathbb{E}[f(w^{k+1}))] \leq f(w^k) - \frac{1}{2dL}\|\nabla f(w^k)\|^2.$$

- If we subtract f^* and use strong-convexity or PL (as before),

$$\mathbb{E}[f(w^{k+1})] - f^* \leq \left(1 - \frac{\mu}{dL}\right)[f(w^k) - f^*].$$

- By recursing we get linear convergence rate,

$$\begin{aligned}\mathbb{E}[\mathbb{E}[f(w^{k+1})]] - f^* &\leq \mathbb{E}\left[\left(1 - \frac{\mu}{dL}\right)[f(w^k) - f^*]\right] && \text{(expectation wrt } j_{k-1}\text{)}\\ \mathbb{E}[f(w^{k+1})] - f(w^*) &\leq \left(1 - \frac{\mu}{dL}\right)[\mathbb{E}[f(w^k)] - f^*] && \text{(iterated expectations)}\\ &\leq \left(1 - \frac{\mu}{dL}\right)^2[f(w^{k-1}) - f^*]\end{aligned}$$

- You keep alternating between taking an expectation back in time and recursing.