10-605/805 – ML for Large Datasets Lecture 6: Distributed PCA & Logistic Regression

Henry Chai

9/15/22

Front Matter

- Recitation 3 on 9/16 will go over HW1 solutions
- HW2 released 9/8, due 9/22 at 11:59 PM

Recall: PCA Algorithm

- Input: $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(i)}\right)\right\}_{i=1}^{n}, r$

1. Center the data
   A. Optionally, normalize the data by features so that all features are of the same scale
2. Compute the covariance matrix $C_X = X^T X$
3. Collect the top r eigenvectors (corresponding to the r largest eigenvalues), $P \in \mathbb{R}^{k \times r}$
4. Project the data into the space defined by P, $Z = XP$

Recall: Computational Cost of PCA

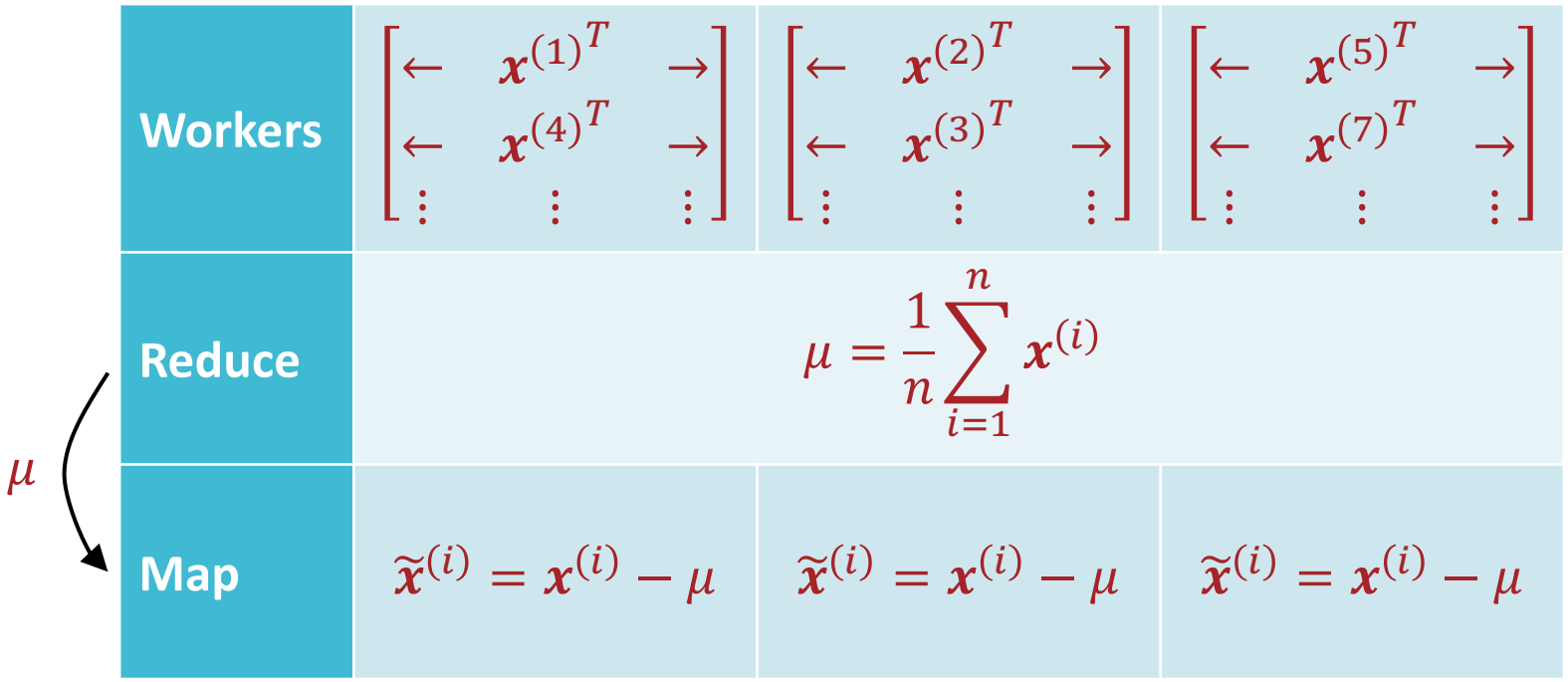
- Input: $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(i)}\right)\right\}_{i=1}^{n}, r$

1. Center the data
   A. Optionally, normalize the data by features so that all features are of the same scale
2. Compute the covariance matrix $C_X = X^T X$ $(O(nk^2))$
3. Collect the top r eigenvectors (corresponding to the r largest eigenvalues), $P \in \mathbb{R}^{k \times r}$ $(O(k^3))$
4. Project the data into the space defined by P, $Z = XP$ $(O(nkr))$

PCA: Large n, Small k

- Assume $O(k^3)$ computation and $O(k^2)$ storage is possible on a single machine
  - ✔ We can store and compute the eigenvalues of $X^T X$
  - We cannot compute $X^T X$
  - We cannot store X

- Approach: basically the same as distributed linear regression
  1. Center the data in a distributed way
  2. Store the rows of X across different machines
  3. Compute $X^T X$ as the sum of outer products



$O(k)$
communication

$O(nk)$ distributed storage (total)

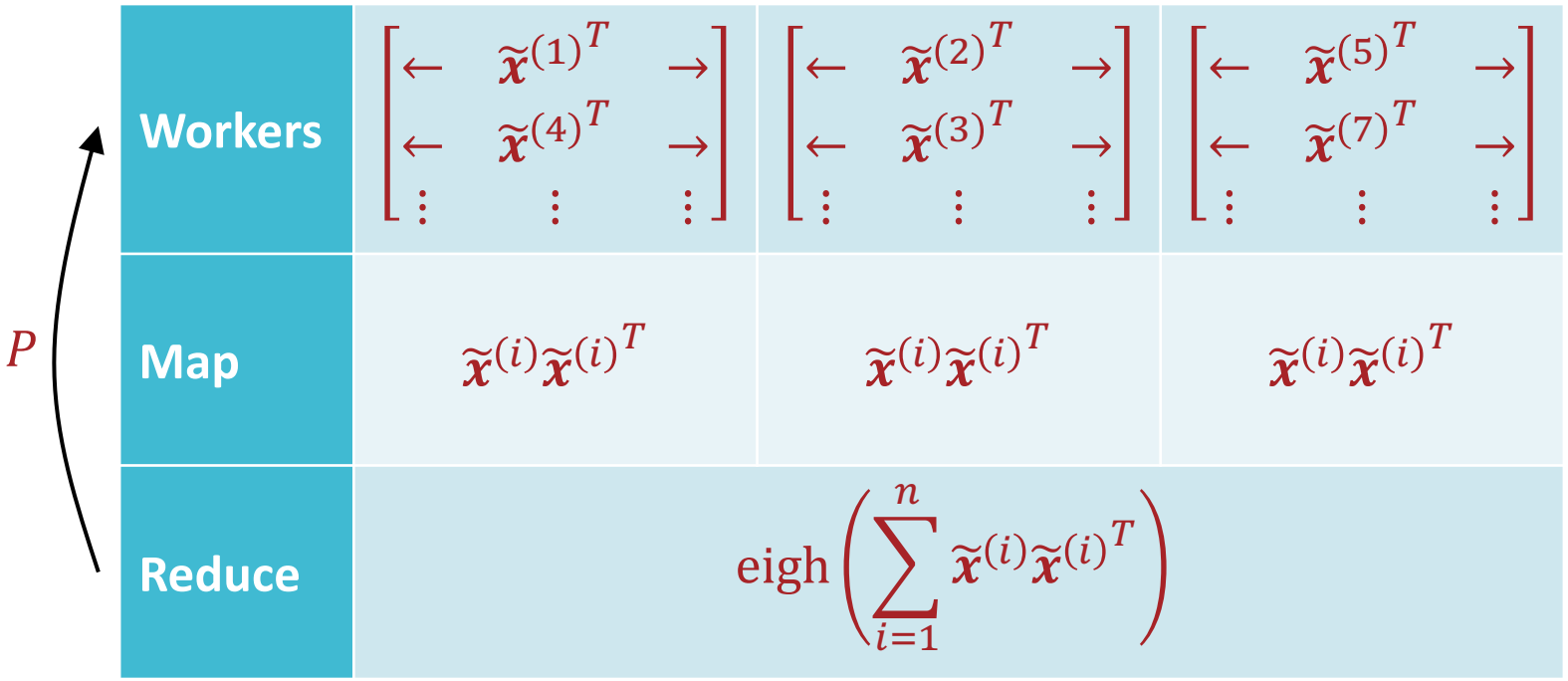
$O(k)$
local work

$O(k)$
local storage

$O(nk)$
distributed work (total)

$O(nk)$
distributed storage (total)

Distributed Centering of the Data



$O(kr)$ communication

$O(nk)$ distributed storage (total)

$O(nk^2)$ distributed work (total)

$O(k^2)$ local storage

$O(k^3)$ local work

$O(k^2)$ local storage

Distributed Eigendecomposition of $X^T X$

Workers	$\begin{bmatrix} \leftarrow & \tilde{\boldsymbol{x}}^{(1)^T} & \rightarrow \\ \leftarrow & \tilde{\boldsymbol{x}}^{(4)^T} & \rightarrow \\ \vdots & \vdots & \vdots \end{bmatrix}$	$\begin{bmatrix} \leftarrow & \tilde{\boldsymbol{x}}^{(2)^T} & \rightarrow \\ \leftarrow & \tilde{\boldsymbol{x}}^{(3)^T} & \rightarrow \\ \vdots & \vdots & \vdots \end{bmatrix}$	$\begin{bmatrix} \leftarrow & \tilde{\boldsymbol{x}}^{(5)^T} & \rightarrow \\ \leftarrow & \tilde{\boldsymbol{x}}^{(7)^T} & \rightarrow \\ \vdots & \vdots & \vdots \end{bmatrix}$
Map	$P\tilde{\boldsymbol{x}}^{(i)}$	$P\tilde{\boldsymbol{x}}^{(i)}$	$P\tilde{\boldsymbol{x}}^{(i)}$

$O(nk)$ distributed storage (total)

$O(nkr)$ distributed work (total)

$O(nr)$ local storage

Distributed Computation of PCA Scores

PCA: Large n, Large k

- Now, $O(k^3)$ computation and $O(k^2)$ storage is *not* possible on a single machine
  - We cannot store and compute the eigenvalues of $X^T X$
  - We cannot compute $X^T X$
  - We cannot store X
- Idea: use a different algorithm!
  - Turn to an iterative method for computing eigenvectors

PCA: Large n, Large k

- Now, $O(k^3)$ computation and $O(k^2)$ storage is *not* possible on a single machine
  - We cannot store and compute the eigenvalues of $X^T X$
  - We cannot compute $X^T X$
  - We cannot store X
- Idea: use a different algorithm!
  - Turn to an iterative method for computing ~~eigenvectors~~ the eigenvector associated with the largest eigenvalue ($r = 1$) → power iteration

Power Iteration

- Fact: $A = X^T X$ is "diagonalizable", i.e., any k-dimensional vector can be written as a linear combination of A's eigenvectors:

$$\boldsymbol{b} = c_1 \boldsymbol{v}_1 + c_2 \boldsymbol{v}_2 \ldots + c_k \boldsymbol{v}_k$$

  - This follows because $X^T X$ is real and symmetric

- Assume $A = X^T X$ has one eigenvalue that is strictly larger than the others:

$$\lambda_1 > \lambda_2 \geq \ \ldots \geq \lambda_k$$

Power Iteration

- Input: $A = X^T X$
- Initialize $\boldsymbol{b}^{(0)}$ to all zeros and set $t = 0$

1. While TERMINATION CRITERION is not satisfied
   a. Update the vector $\boldsymbol{b}$:

$$\boldsymbol{b}^{(t+1)} \leftarrow \frac{A\boldsymbol{b}^{(t)}}{\left\| A\boldsymbol{b}^{(t)} \right\|_2}$$

   b. Increment t: $t \leftarrow t + 1$

- Output: $\boldsymbol{b}^{(t)}$, the eigenvector corresponding to the largest eigenvalue of A!

Power Iteration

$$\boldsymbol{b}^{(0)} = c_1\boldsymbol{v}_1 + c_2\boldsymbol{v}_2 + \cdots + c_k\boldsymbol{v}_k$$

$$A\boldsymbol{b}^{(0)} = c_1A\boldsymbol{v}_1 + c_2A\boldsymbol{v}_2 + \cdots + c_kA\boldsymbol{v}_k$$

$$= c_1\lambda_1\boldsymbol{v}_1 + c_2\lambda_2\boldsymbol{v}_2 + \cdots + c_k\lambda_k\boldsymbol{v}_k$$

$$A\left(A\boldsymbol{b}^{(0)}\right) = (A)^2\boldsymbol{b}^{(0)} = c_1\lambda_1A\boldsymbol{v}_1 + c_2\lambda_2A\boldsymbol{v}_2 + \cdots + c_k\lambda_kA\boldsymbol{v}_k$$

$$= c_1\lambda_1^2\boldsymbol{v}_1 + c_2\lambda_2^2\boldsymbol{v}_2 + \cdots + c_k\lambda_k^2\boldsymbol{v}_k$$

$$(A)^t\boldsymbol{b}^{(0)} = c_1\lambda_1^t\boldsymbol{v}_1 + c_2\lambda_2^t\boldsymbol{v}_2 + \cdots + c_k\lambda_k^t\boldsymbol{v}_k$$

$$= c_1\lambda_1^t\left(\boldsymbol{v}_1 + c_2\left(\frac{\lambda_2}{\lambda_1}\right)^t\boldsymbol{v}_2 \ldots + c_2\left(\frac{\lambda_k}{\lambda_1}\right)^t\boldsymbol{v}_k\right)$$

$$\rightarrow c_1\lambda_1^t(\boldsymbol{v}_1) \text{ as } t \rightarrow \infty$$

Normalize $(A)^t\boldsymbol{b}^{(0)}$ to have unit norm: $\boldsymbol{b}^{(t)} = \frac{(A)^t\boldsymbol{b}^{(0)}}{\left\|(A)^t\boldsymbol{b}^{(0)}\right\|_2}$

Power Iteration

- Input: $A = X^T X$
- Initialize $\boldsymbol{b}^{(0)}$ to all zeros and set $t = 0$

1. While TERMINATION CRITERION is not satisfied
   a. Update the vector $\boldsymbol{b}$:

$$\boldsymbol{b}^{(t+1)} \leftarrow \frac{X^T X \boldsymbol{b}^{(t)}}{\left\| X^T X \boldsymbol{b}^{(t)} \right\|_2}$$

   b. Increment t: $t \leftarrow t + 1$

- Output: $\boldsymbol{b}^{(t)}$, the eigenvector corresponding to the largest eigenvalue of A!

Power Iteration

- Input: $A = X^T X$
- Initialize $\boldsymbol{b}^{(0)}$ to all zeros and set $t = 0$

1. While TERMINATION CRITERION is not satisfied
   a. Update the vector $\boldsymbol{b}$ using some intermediate steps:

$$\boldsymbol{b}^{(t+1)} \leftarrow X^T X \boldsymbol{b}^{(t)} = \left(\sum_{i=1}^{n} \tilde{\boldsymbol{x}}^{(i)} \tilde{\boldsymbol{x}}^{(i)^T} \right) \boldsymbol{b}^{(t)}$$

$$\boldsymbol{b}^{(t+1)} \leftarrow \frac{\boldsymbol{b}^{(t+1)}}{\left\| \boldsymbol{b}^{(t+1)} \right\|_2}$$

   b. Increment t: $t \leftarrow t + 1$

- Output: $\boldsymbol{b}^{(t)}$, the eigenvector corresponding to the largest eigenvalue of A!

Power Iteration

- Input: $A = X^T X$
- Initialize $\boldsymbol{b}^{(0)}$ to all zeros and set $t = 0$

1. While TERMINATION CRITERION is not satisfied
   a. Update the vector $\boldsymbol{b}$ using some intermediate steps:

$$\boldsymbol{b}^{(t+1)} \leftarrow X^T X \boldsymbol{b}^{(t)} = \sum_{i=1}^{n} \tilde{\boldsymbol{x}}^{(i)} \left(\tilde{\boldsymbol{x}}^{(i)^T} \boldsymbol{b}^{(t)} \right)$$

$$\boldsymbol{b}^{(t+1)} \leftarrow \frac{\boldsymbol{b}^{(t+1)}}{\left\| \boldsymbol{b}^{(t+1)} \right\|_2}$$

   b. Increment t: $t \leftarrow t + 1$

- Output: $\boldsymbol{b}^{(t)}$, the eigenvector corresponding to the largest eigenvalue of A!

Power Iteration

- Input: $A = X^T X$
- Initialize $\boldsymbol{b}^{(0)}$ to all zeros and set $t = 0$

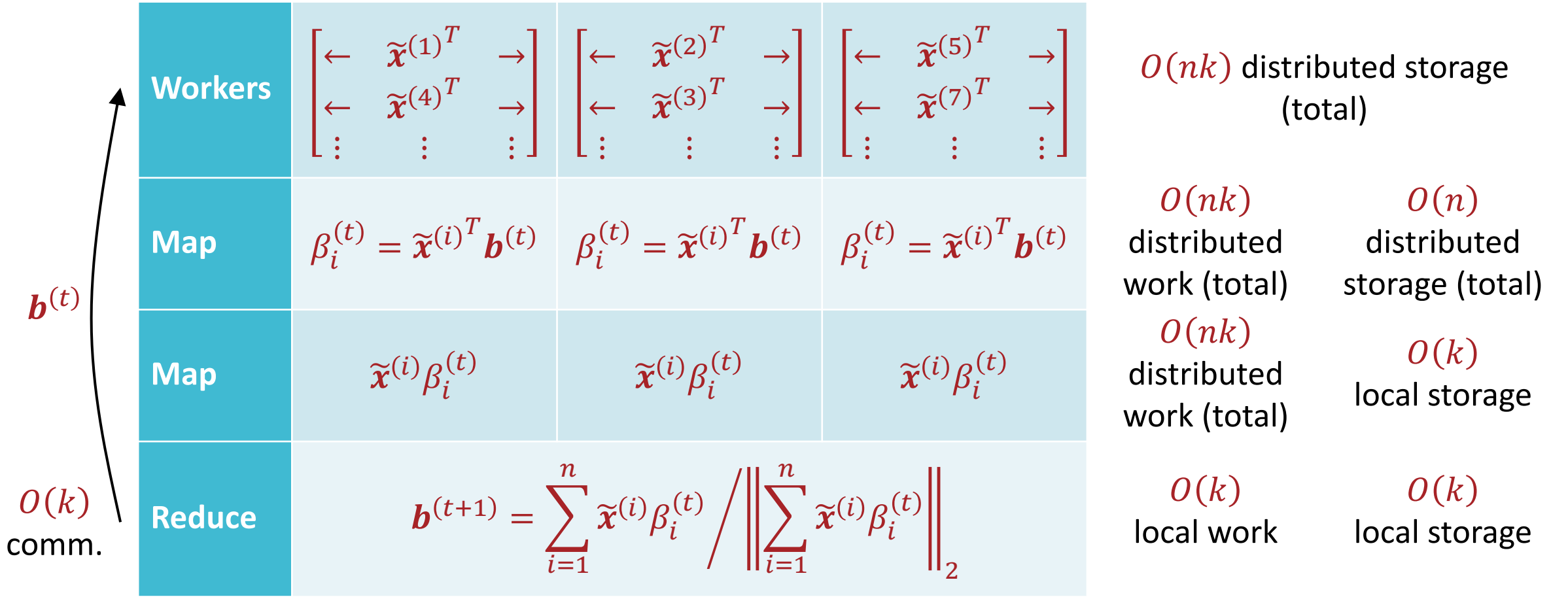
1. While TERMINATION CRITERION is not satisfied
   a. Update the vector $\boldsymbol{b}$ using some intermediate steps:

$$\boldsymbol{b}^{(t+1)} \leftarrow X^T X \boldsymbol{b}^{(t)} = \sum_{i=1}^{n} \tilde{\boldsymbol{x}}^{(i)} \left(\beta_i^{(t)} \right)$$

$$\boldsymbol{b}^{(t+1)} \leftarrow \frac{\boldsymbol{b}^{(t+1)}}{\left\| \boldsymbol{b}^{(t+1)} \right\|_2}$$

   b. Increment t: $t \leftarrow t + 1$

- Output: $\boldsymbol{b}^{(t)}$, the eigenvector corresponding to the largest eigenvalue of A!



$O(nk)$ distributed storage (total)

$O(nk)$ distributed work (total)

$O(n)$ distributed storage (total)

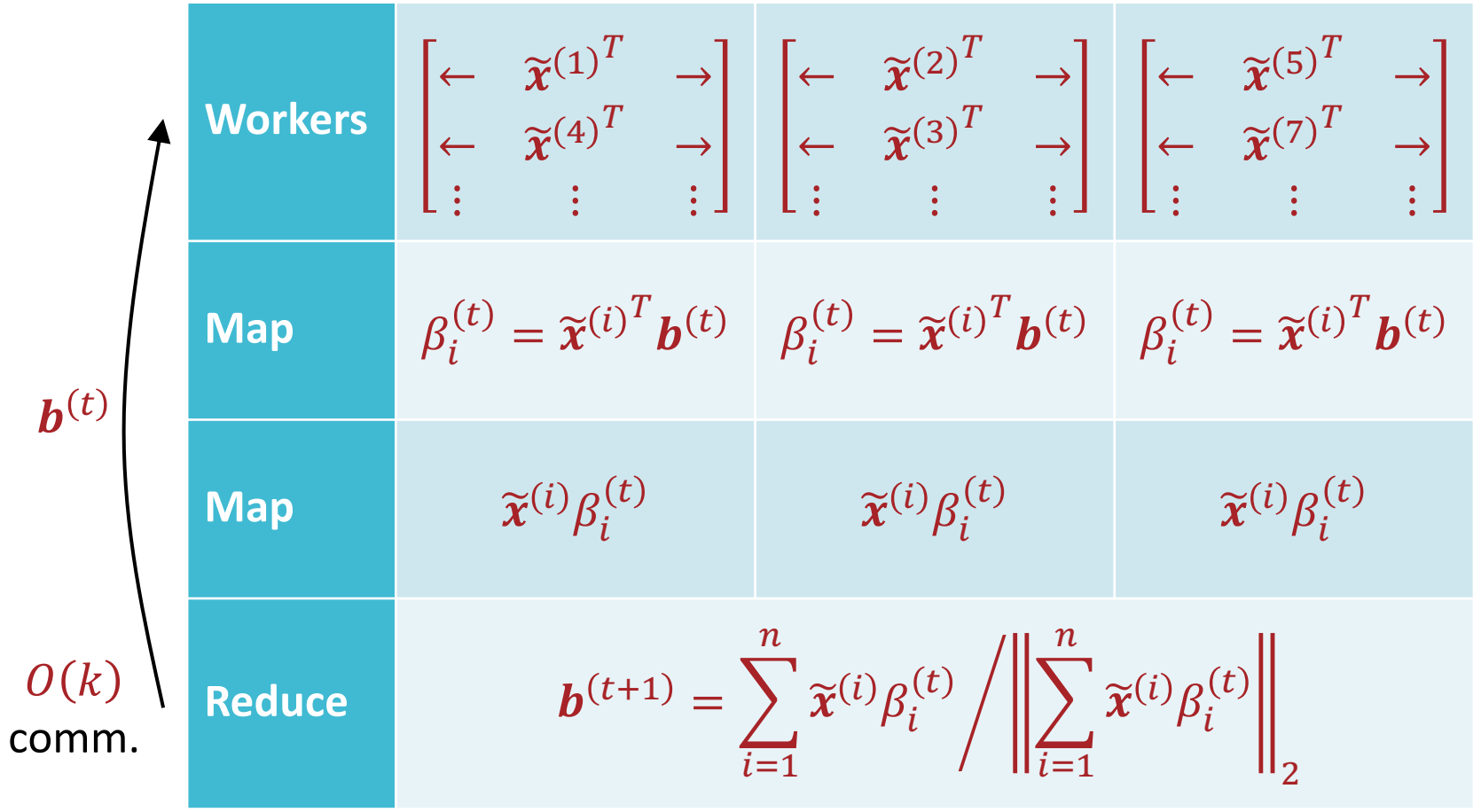
$O(nk)$ distributed work (total)

$O(k)$ local storage

$O(k)$ local work

$O(k)$ local storage

Distributed Power Iteration



$O(nk)$ distributed storage (total)

$O(nk)$ distributed work (total)	$O(n)$ distributed storage (total)
$O(nk)$ distributed work (total)	$O(k)$ local storage
$O(k)$ local work	$O(k)$ local storage

Disclaimer: to find more than one eigenvector, we need to use similar (but slightly more complicated methods), e.g., PySpark's MLlib uses Krylov subspace methods

Background: Empirical Risk Minimization

- A common framework for supervised learning
- Given:
  - some labelled training dataset $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(i)}, y^{(i)}\right)\right\}_{i=1}^{n}$
  - a loss function $\ell: \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$
  - a hypothesis class or set of functions $\mathcal{F}$

the goal is to find

$$\hat{f} = \underset{f \in \mathcal{F}}{\operatorname{argmin}} \sum_{i=1}^{n} \ell\left(f\left(\boldsymbol{x}^{(i)}\right), y^{(i)}\right)$$

with the hope that

$$\mathbb{E}_{p(\boldsymbol{x},y)}[\ell(f(\boldsymbol{x}), y)] \approx \sum_{i=1}^{n} \ell\left(f\left(\boldsymbol{x}^{(i)}\right), y^{(i)}\right)$$

Background: Binary Classification

- A type of supervised learning
- Given:
  - some labelled training dataset $\mathcal{D} = \{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^{n}$
  - a loss function $\ell: \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ where $\mathcal{Y} = \{0,1\}$
  - a hypothesis class or set of functions $\mathcal{F}$

the goal is to find

$$\hat{f} = \underset{f \in \mathcal{F}}{\operatorname{argmin}} \sum_{i=1}^{n} \ell\left(f\left(\boldsymbol{x}^{(i)}\right), y^{(i)}\right)$$

Background: Binary Classification w/ 0-1 Loss

- A type of supervised learning
- Given:
  - some labelled training dataset $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(i)}, y^{(i)}\right)\right\}_{i=1}^{n}$
  - $\ell(y, y') = \mathbb{1}(y \neq y')$ for $y, y' \in \{0,1\}$
  - a hypothesis class or set of functions $\mathcal{F}$

  the goal is to find

$$\hat{f} = \underset{f \in \mathcal{F}}{\operatorname{argmin}} \sum_{i=1}^{n} \ell\left(f\left(\boldsymbol{x}^{(i)}\right), y^{(i)}\right)$$

- This loss function is difficult to optimize (non-convex)…

A Probabilistic Approach to Binary Classification

- Suppose we have binary labels $y \in \{0,1\}$ and D-dimensional inputs $\boldsymbol{x} = [1, x_1, \ldots, x_D]^T \in \mathbb{R}^{D+1}$
- **Assume**

$$P(Y = 1|\boldsymbol{x}) = \sigma(\boldsymbol{w}^T\boldsymbol{x}) = \frac{1}{1 + \exp(-\boldsymbol{w}^T\boldsymbol{x})}$$

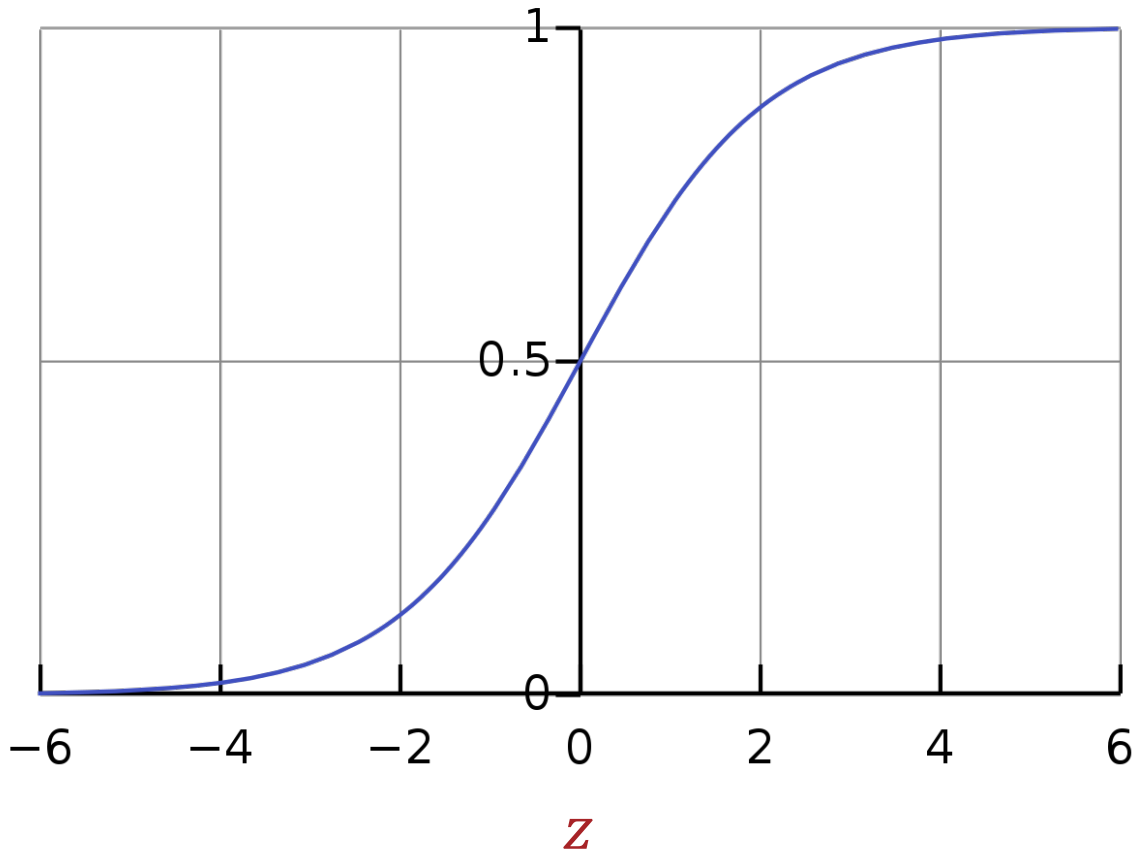
$$= \frac{\exp(\boldsymbol{w}^T\boldsymbol{x})}{\exp(\boldsymbol{w}^T\boldsymbol{x}) + 1}$$

- This implies two useful facts:

1. $P(Y = 0|\boldsymbol{x}) = 1 - P(Y = 1|\boldsymbol{x}) = \frac{1}{\exp(\boldsymbol{w}^T\boldsymbol{x}) + 1}$
2. $\frac{P(Y = 1|\boldsymbol{x})}{P(Y = 0|\boldsymbol{x})} = \exp(\boldsymbol{w}^T\boldsymbol{x}) \rightarrow \log\frac{P(Y = 1|\boldsymbol{x})}{P(Y = 0|\boldsymbol{x})} = \boldsymbol{w}^T\boldsymbol{x}$

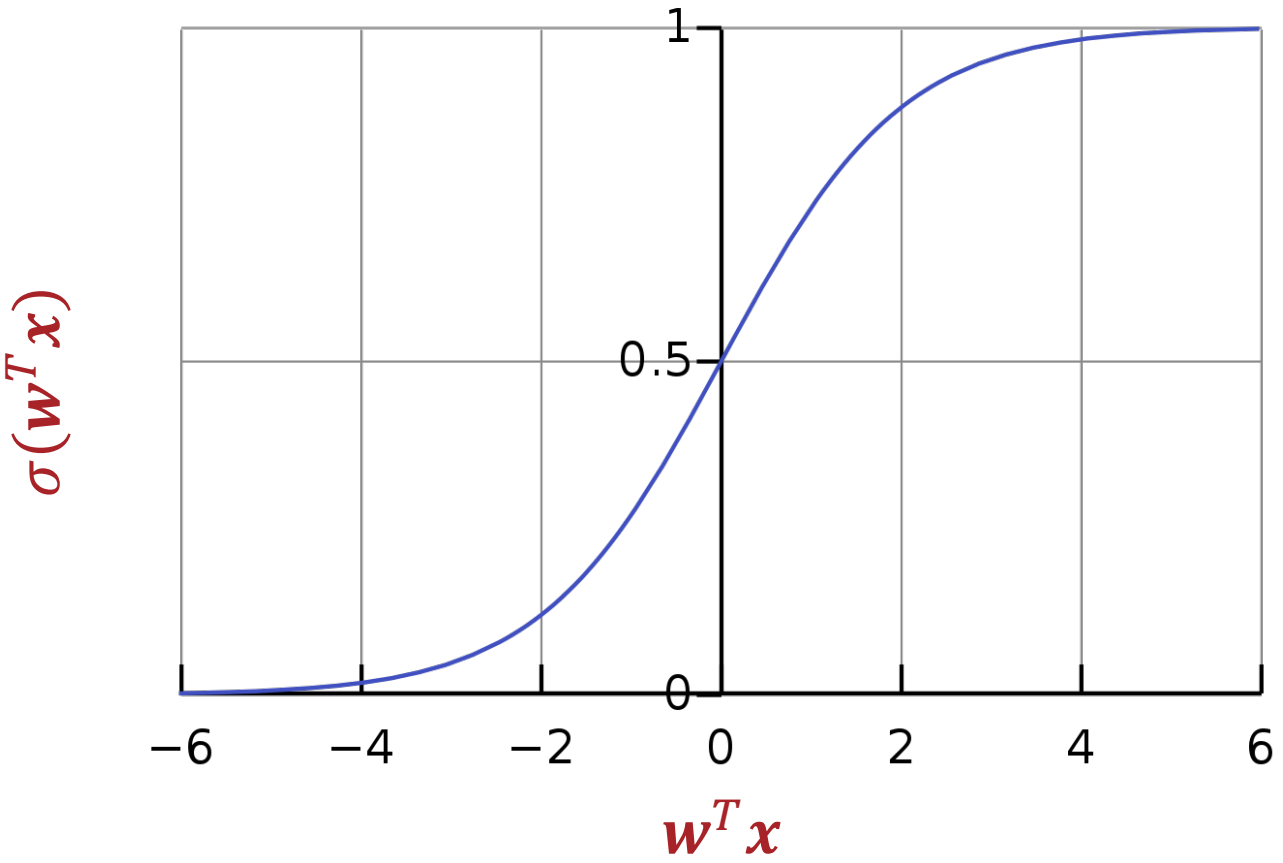
Logistic Function

$$\sigma(z) = \frac{1}{1+e^{-z}}$$



Source: https://en.wikipedia.org/wiki/Logistic_function#/media/File:Logistic-curve.svg

Why use the Logistic Function?



- Differentiable everywhere
- $\sigma: \mathbb{R} \to [0, 1]$
- The decision boundary is linear in $\boldsymbol{x}$!

Source: https://en.wikipedia.org/wiki/Logistic_function#/media/File:Logistic-curve.svg

Logistic Regression Decision Boundary

$$y' = \begin{cases} 1 \text{ if } P(Y = 1|\boldsymbol{x}) \geq \frac{1}{2} \\ 0 \text{ otherwise} \end{cases}$$

$$P(Y = 1|\boldsymbol{x}) = \sigma(\boldsymbol{w}^T\boldsymbol{x}) = \frac{1}{1 + \exp(-\boldsymbol{w}^T\boldsymbol{x})} \geq \frac{1}{2}$$

$$2 \geq 1 + \exp(-\boldsymbol{w}^T\boldsymbol{x})$$

$$1 \geq \exp(-\boldsymbol{w}^T\boldsymbol{x})$$

$$\log(1) \geq -\boldsymbol{w}^T\boldsymbol{x}$$

$$0 \leq \boldsymbol{w}^T\boldsymbol{x}$$

Why 1/2?

$$y' = \begin{cases} 1 \text{ if } P(Y = 1|\boldsymbol{x}) \geq \frac{1}{2} \\ 0 \text{ otherwise} \end{cases}$$

$$P(Y = 1|\boldsymbol{x}) = \sigma(\boldsymbol{w}^T\boldsymbol{x}) = \frac{1}{1 + \exp(-\boldsymbol{w}^T\boldsymbol{x})} \geq \frac{1}{2}$$

$$2 \geq 1 + \exp(-\boldsymbol{w}^T\boldsymbol{x})$$

$$1 \geq \exp(-\boldsymbol{w}^T\boldsymbol{x})$$

$$\log(1) \geq -\boldsymbol{w}^T\boldsymbol{x}$$

$$0 \leq \boldsymbol{w}^T\boldsymbol{x}$$

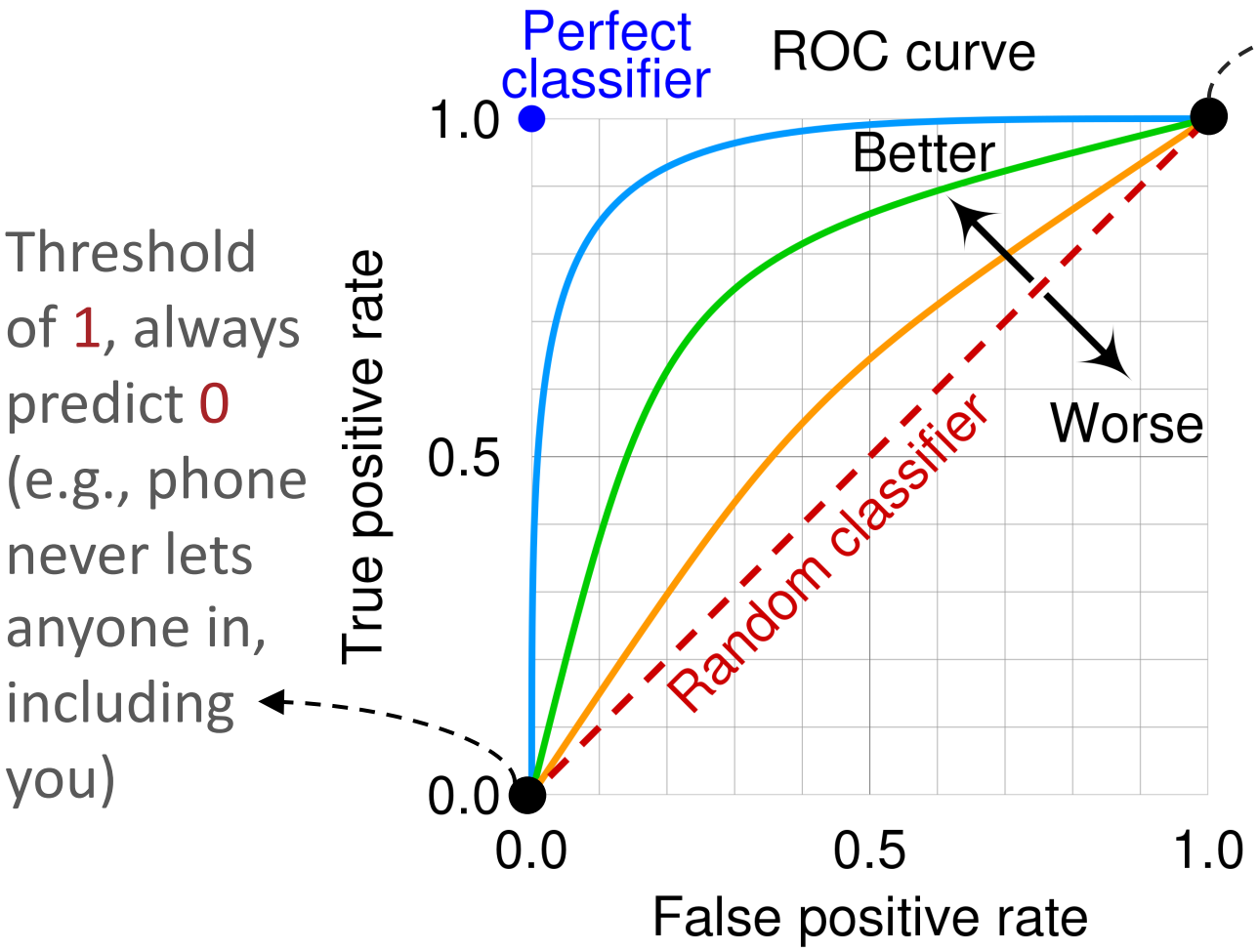
Aside: Different Kinds of Errors

		y	
		0	1
y'	0	True negative (TN)	False negative (FN)
	1	False positive (FP)	True positive (TP)

- Depending on the setting, these kinds of errors can have wildly different impacts/costs, e.g., facial recognition for unlocking phones:
  - False negative = minor nuisance, just try again
  - False positive = major security breach
    - Using a threshold > 1/2 will tend to decrease the number of false positives

Aside: Receiver Operating Characteristic (ROC) Curves

- Plots true positive rate, TPR = TP / (TP + FN), against false positive rate, FPR = FP / (FP + TN)



Source: https://en.wikipedia.org/wiki/Receiver_operating_characteristic#/media/File:Roc_curve.svg

Logistic Regression: Decision Boundary

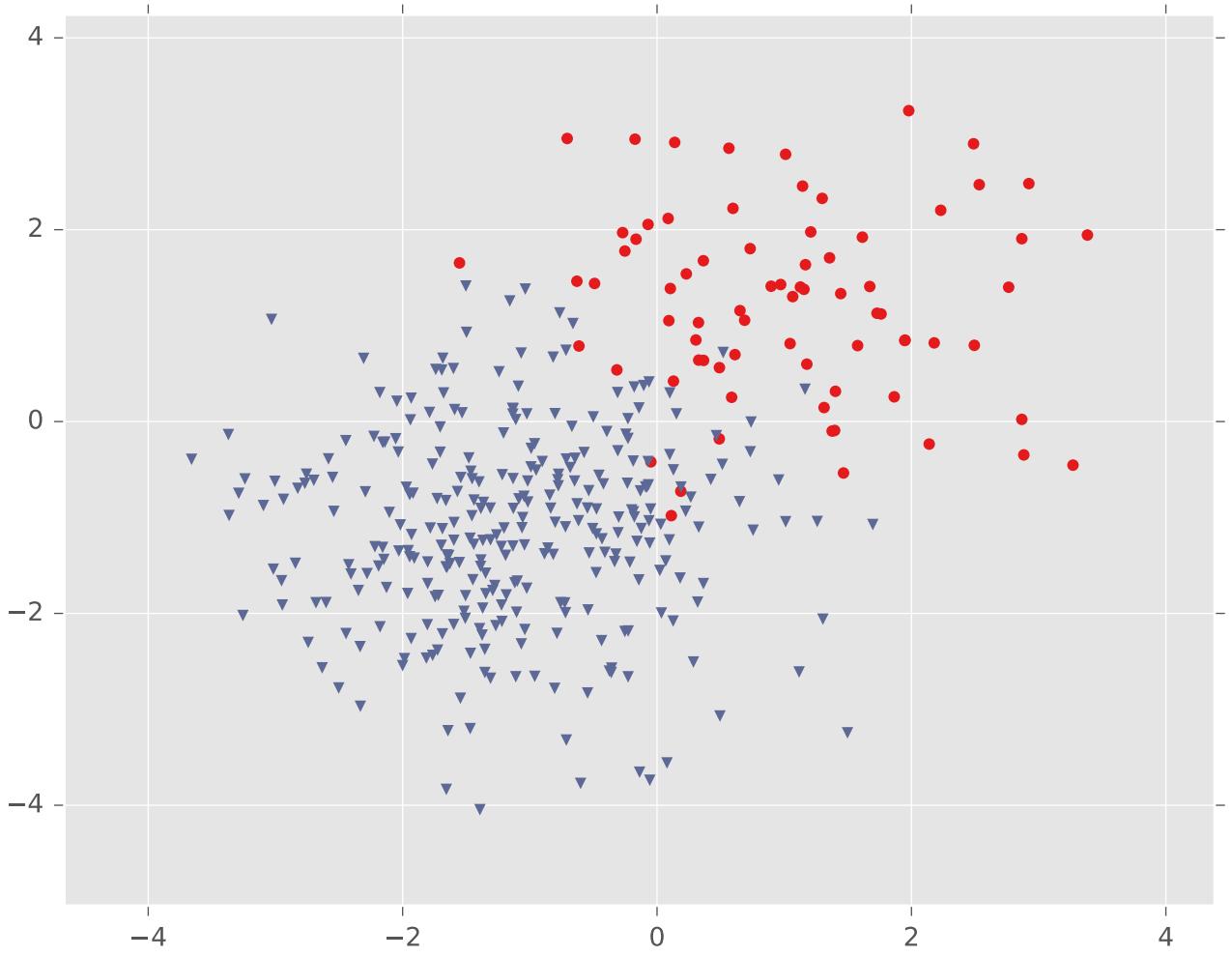


Figure courtesy of Matt Gormley

Logistic Regression: Decision Boundary

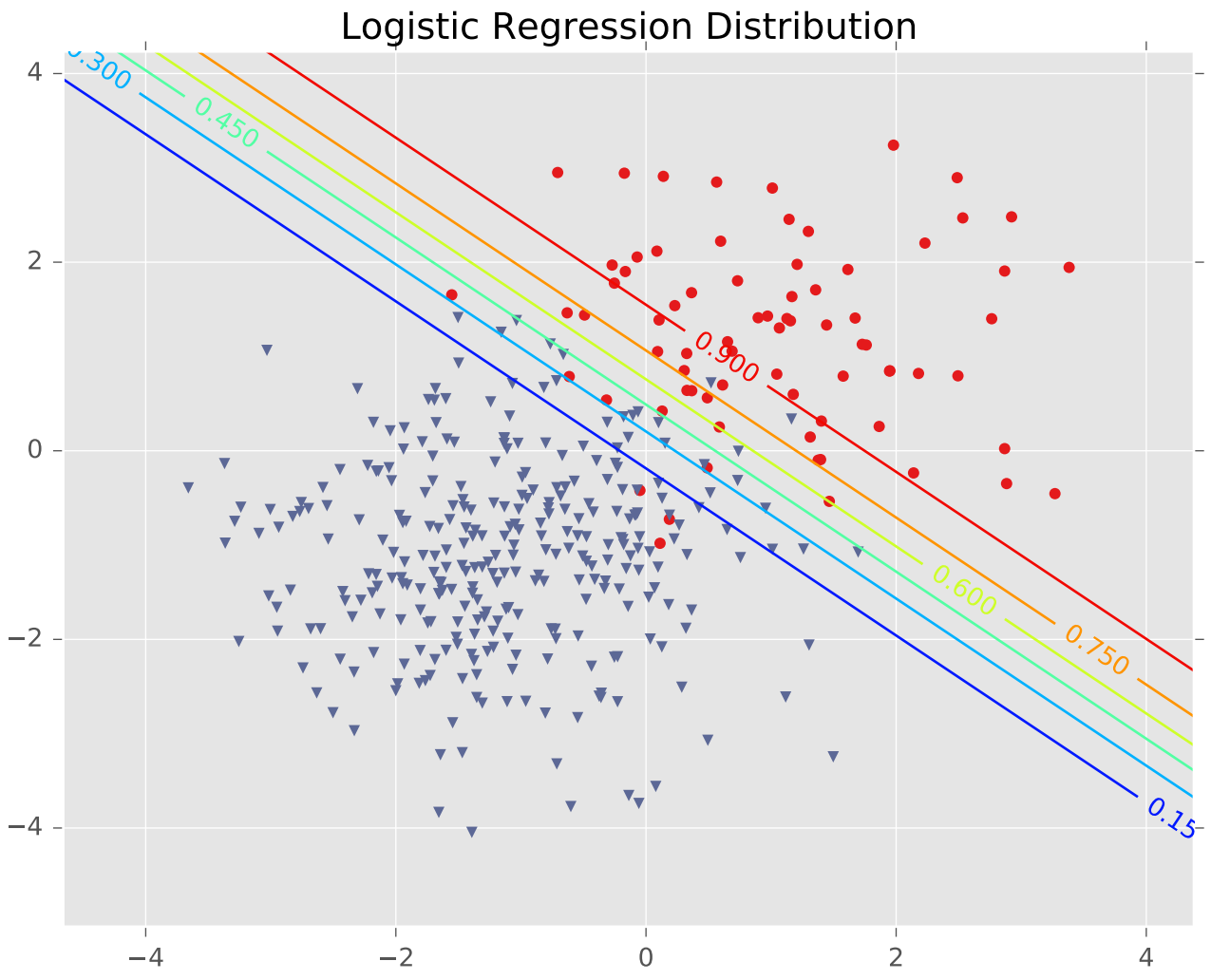


Figure courtesy of Matt Gormley

Logistic Regression: Decision Boundary

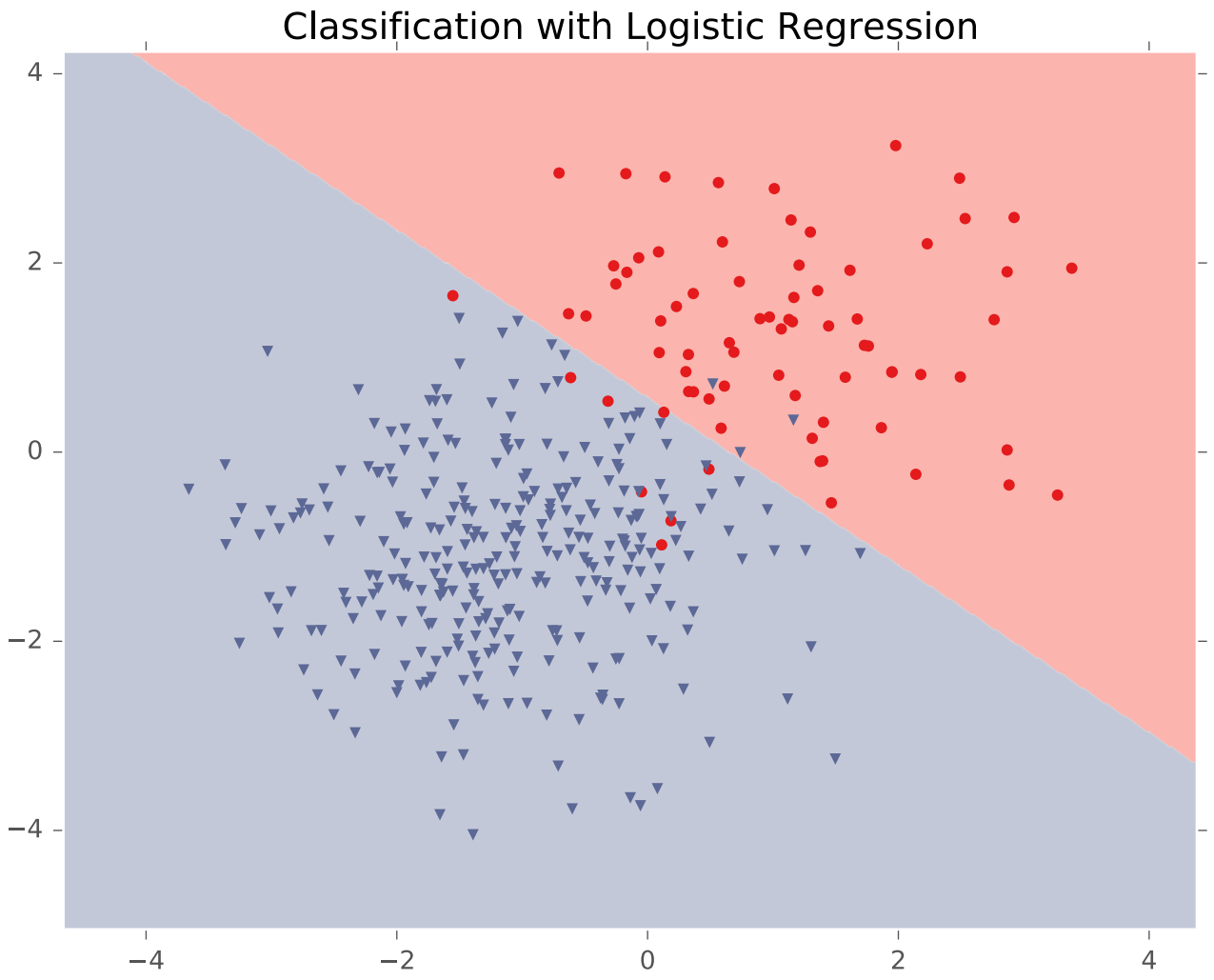


Figure courtesy of Matt Gormley

Logistic Regression: Setting the Parameters

- Goal: find the $\boldsymbol{w}$ that *maximizes* the (conditional) probability of the training dataset:

$$\prod_{i=1}^{n} P\left(y^{(i)} \middle| \boldsymbol{x}^{(i)}, \boldsymbol{w}\right)$$

- This is equivalent to finding the $\boldsymbol{w}$ that *minimizes* the negative log of this probability:

$$L_{\mathcal{D}}(\boldsymbol{w}) = -\log \prod_{i=1}^{n} P\left(y^{(i)} \middle| \boldsymbol{x}^{(i)}, \boldsymbol{w}\right)$$

$$= -\log \prod_{i=1}^{n} P\left(Y = 1 \middle| \boldsymbol{x}^{(i)}, \boldsymbol{w}\right)^{y^{(i)}} \left(P\left(Y = 0 \middle| \boldsymbol{x}^{(i)}, \boldsymbol{w}\right)\right)^{1-y^{(i)}}$$

$$= -\sum_{i=1}^{n} \underbrace{y^{(i)} \log P\left(Y = 1 \middle| \boldsymbol{x}^{(i)}, \boldsymbol{w}\right) + \left(1 - y^{(i)}\right) \log P\left(Y = 0 \middle| \boldsymbol{x}^{(i)}, \boldsymbol{w}\right)}_{\text{log loss or cross-entropy loss}}$$

Logistic Regression as Empirical Risk Minimization

- A type of supervised learning
- Given:
  - some labelled training dataset $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(i)}, y^{(i)}\right)\right\}_{i=1}^{n}$
  - the log loss
  - $\mathcal{F} =$ the set of all linear decision boundaries

the goal is to find

$$\underset{\boldsymbol{w}}{\operatorname{argmin}} - \sum_{i=1}^{n} \left(y^{(i)} \log P\left(Y = 1 \middle| \boldsymbol{x}^{(i)}, \boldsymbol{w}\right)\right) + \left(1 - y^{(i)}\right) \log P\left(Y = 0 \middle| \boldsymbol{x}^{(i)}, \boldsymbol{w}\right)$$

$$= \underset{\boldsymbol{w}}{\operatorname{argmin}} - \sum_{i=1}^{n} y^{(i)} \log \frac{P\left(Y = 1 \middle| \boldsymbol{x}^{(i)}, \boldsymbol{w}\right)}{P\left(Y = 0 \middle| \boldsymbol{x}^{(i)}, \boldsymbol{w}\right)} + \log P\left(Y = 0 \middle| \boldsymbol{x}^{(i)}, \boldsymbol{w}\right)$$

$$= \underset{\boldsymbol{w}}{\operatorname{argmin}} - \sum_{i=1}^{n} y^{(i)} \boldsymbol{w}^T \boldsymbol{x}^{(i)} - \log\left(1 + \exp\left(\boldsymbol{w}^T \boldsymbol{x}^{(i)}\right)\right)$$

Logistic Regression as Empirical Risk Minimization

$$L_{\mathcal{D}}(\boldsymbol{w}) = -\sum_{i=1}^{n} y^{(i)} \boldsymbol{w}^T \boldsymbol{x}^{(i)} - \log\left(1 + \exp\left(\boldsymbol{w}^T \boldsymbol{x}^{(i)}\right)\right)$$

$$\nabla_{\boldsymbol{w}} L_{\mathcal{D}}(\boldsymbol{w}) = -\sum_{i=1}^{n} y^{(i)} \nabla_{\boldsymbol{w}} \boldsymbol{w}^T \boldsymbol{x}^{(i)} - \nabla_{\boldsymbol{w}} \log\left(1 + \exp\left(\boldsymbol{w}^T \boldsymbol{x}^{(i)}\right)\right)$$

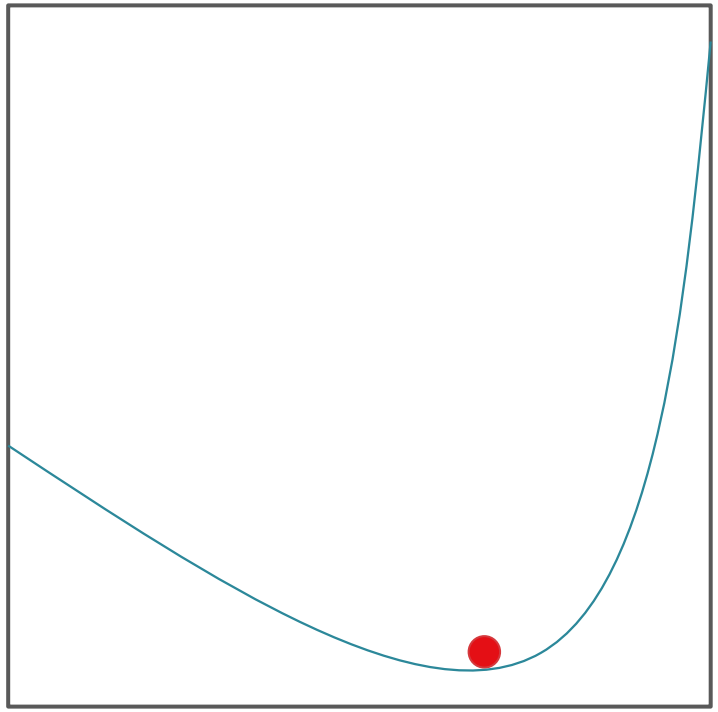
$$= -\sum_{i=1}^{n} y^{(i)} \boldsymbol{x}^{(i)} - \frac{\exp\left(\boldsymbol{w}^T \boldsymbol{x}^{(i)}\right)}{1 + \exp(\boldsymbol{w}^T \boldsymbol{x}^{(i)})} \boldsymbol{x}^{(i)}$$

$$= \sum_{i=1}^{n} \boldsymbol{x}^{(i)} \frac{1}{\exp(-\boldsymbol{w}^T \boldsymbol{x}^{(i)}) + 1} - y^{(i)} \boldsymbol{x}^{(i)}$$

$$= \sum_{i=1}^{n} \boldsymbol{x}^{(i)} \left(\sigma\left(\boldsymbol{w}^T \boldsymbol{x}^{(i)}\right) - y^{(i)}\right)$$

Recall: Gradient Descent

- An iterative method for minimizing functions
- Requires the gradient to exist everywhere



- Good news: the log loss for logistic regression is convex!

Gradient Descent for Logistic Regression

- Input: $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(i)}, y^{(i)}\right)\right\}_{i=1}^{n}, \alpha$

1. Initialize $\boldsymbol{w}^{(0)}$ to all zeros and set $t = 0$
2. While TERMINATION CRITERION is not satisfied
   a. Compute the gradient:

$$\nabla_{\boldsymbol{w}} L_{\mathcal{D}}\left(\boldsymbol{w}^{(t)}\right) = \sum_{i=1}^{n} \boldsymbol{x}^{(i)}\left(\text{logit}\left(\boldsymbol{w}^{(t)^T}\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)$$

   b. Update $\boldsymbol{w}$: $\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} - \alpha \nabla_{\boldsymbol{w}} L_{\mathcal{D}}\left(\boldsymbol{w}^{(t)}\right)$
   c. Increment t: $t \leftarrow t + 1$

- Output: $\boldsymbol{w}^{(t)}$

Idea: distribute $\boldsymbol{x}^{(i)}$ and compute summands in parallel

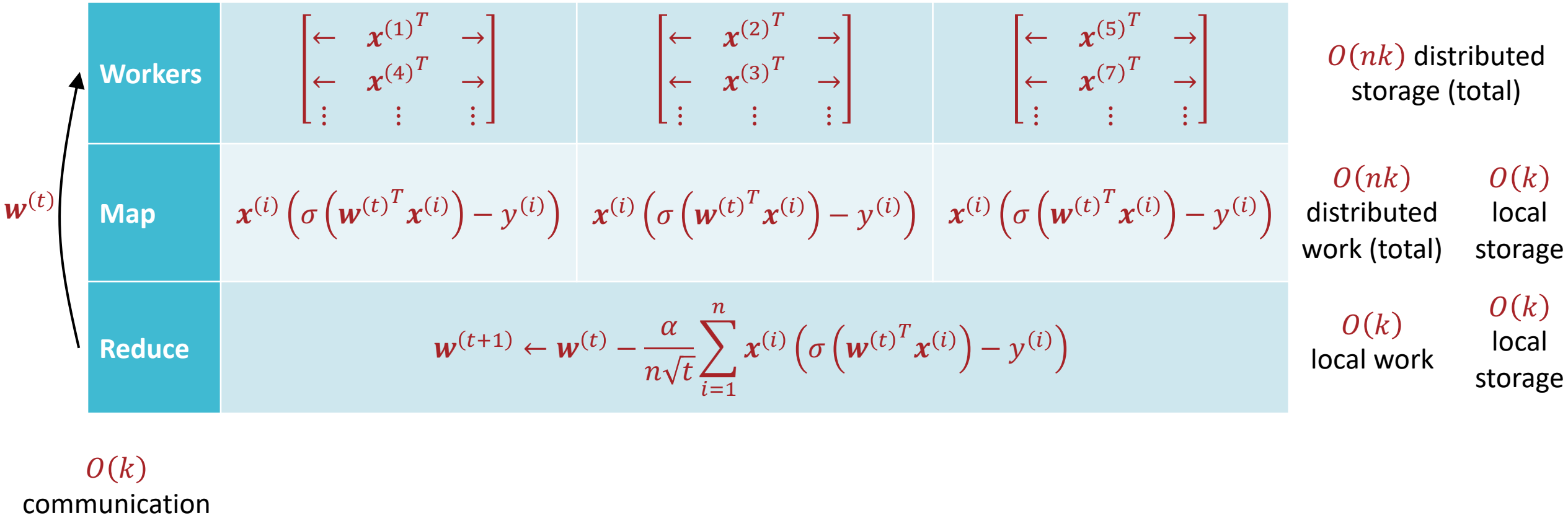
- Input: $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(i)}, y^{(i)}\right)\right\}_{i=1}^{n}, \alpha$

1. Initialize $\boldsymbol{w}^{(0)}$ to all zeros and set $t = 0$
2. While TERMINATION CRITERION is not satisfied
   a. Compute the gradient:

$$\nabla_{\boldsymbol{w}} L_{\mathcal{D}}\left(\boldsymbol{w}^{(t)}\right) = \sum_{i=1}^{n} \boldsymbol{x}^{(i)}\left(\text{logit}\left(\boldsymbol{w}^{(t)^T}\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)$$

   b. Update $\boldsymbol{w}$: $\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} - \alpha \nabla_{\boldsymbol{w}} L_{\mathcal{D}}\left(\boldsymbol{w}^{(t)}\right)$
   c. Increment t: $t \leftarrow t + 1$

- Output: $\boldsymbol{w}^{(t)}$



Distributed Gradient Descent for Logistic Regression

What do linear regression, PCA and logistic regression all have in common?

- They're all linear methods…
- … that can be kernelized to achieve nonlinear models!
  - What is a kernel and how can we apply it at scale?

    Come back on Tuesday and find out!

Key Takeaways

- Distributed PCA
  - Very similar to distributed linear regression:
    - If k is small, simply distribute the storage and computation of $X^T X$
    - If k is large, use iterative methods (e.g., power iteration) to compute eigenvector-eigenvalue pairs
- Distributed logistic regression
  - Distribute the summands of gradient descent and compute in parallel

Positive Semidefinite Matrices

- For a symmetric matrix $A \in \mathbb{R}^{n \times n}$, the following statements are equivalent:

1. A is positive semidefinite
2. $\forall\, \boldsymbol{x} \in \mathbb{R}^n, \boldsymbol{x}^T A \boldsymbol{x} \geq 0$
3. All eigenvalues of A are greater than or equal to 0
4. $\exists\, U$ s.t. $A = U^T U$

Block Matrices

- Sometimes, it may be convenient to express a matrix A as comprised of several smaller matrices:

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$

- The Schur complement of A_{22} is defined as

$$C = A_{11} - A_{12}A_{22}^{-1}A_{21}$$

- Using the Schur complement, we can succinctly express quantities like

$$\det(A) = \det(A_{22}) * \det(C)$$

$$A^{-1} = \begin{bmatrix} I & 0 \\ -A_{22}^{-1}A_{21} & I \end{bmatrix} \begin{bmatrix} C & 0 \\ 0 & A_{22}^{-1} \end{bmatrix} \begin{bmatrix} I & -A_{12}A_{22}^{-1} \\ 0 & I \end{bmatrix}$$

Orthogonal Matrices

- An orthogonal matrix is a real symmetric matrix whose columns and rows are orthonormal vectors.
- If A is an orthogonal matrix, then this implies that

$$A^T A = A A^T = I$$

or equivalently

$$A^T = A^{-1}$$

Singular Value Decomposition

- Any matrix real $A \in \mathbb{R}^{m \times n}$ can be expressed as the product of three matrices:

$$A = U\Sigma V^T$$

where

- $U \in \mathbb{R}^{m \times m}$ is an orthogonal matrix whose columns are the eigenvectors of AA^T
- $V \in \mathbb{R}^{n \times n}$ is also an orthogonal matrix whose columns are the eigenvectors of $A^T A$
- $\Sigma \in \mathbb{R}^{m \times n}$ is a diagonal rectangular matrix whose non-zero elements are the square roots of the eigenvalues of AA^T (equivalently, $A^T A$)