MapReduce

- Map: "map" each split of the data to a machine
- Reduce: combine results onto one machine
- Launch another task to deal with failures and stragglers
- Hadoop passes through disk after every map and reduce

Spark

- RDD (Resilient Distributed Datasets): distributed lists
- Dataframes: SQL-like structured datasets with query ops
- Datasets: Mix of RDDs and Dataframes
- Transformations: lazily computed on RDDs
- Actions: eagerly computed results based on RDDs

Dimensionality Reduction

- Find embedding to project data onto lower dimension
- Find embedding that minimizes reconstruction error and maximizes variance

PCA: $\mathbf{X}$ is $n \times k$ raw data, $\mathbf{Z} = \mathbf{XP}$ is $n \times r$ PCA scores, $\mathbf{P}$ is $k \times r$ columns with r prinicipal components

Assumes data is linear

Covariance Matrix: $\mathbf{C_X} = \frac{1}{n}\mathbf{X}^T\mathbf{X}$

$\mathbf{P}$ is the top r eigenvectors of $\mathbf{C_X}$

k eigenvectors are orthonormal directions of max variance

1. Center Data
2. Compute Covariance Matrix $\mathbf{C_X} = \frac{1}{n}\mathbf{X}^T\mathbf{X}$ $O(nk^2)$
3. Eigendecomposition ($\mathbf{P} = r$ columns of $\mathbf{U}$)$\mathbf{C_X} = \mathbf{U\Lambda U}^T$ $O(k^3)$
4. Compute PCA Scores $Z = XP$ $O(nkr)$

Choose r to retain some fraction of the variance: $\frac{\sum_{i=1}^{r}\lambda_i}{\sum_{i=1}^{k}\lambda_i}$

Johnson-Lindenstrauss: To provide a random project with error ϵ, set min. dimension of the subspace to: $r > \ln(n)/\epsilon^2$

t-SNE: $O(n^2)$

1. Calculate probability of similarity in high-dim space
   - Focus on similiarities in a local neighborhood
2. Try to recreate probability in low-dim space
   - Minimize KL-divergence

Linear Regression

Closed form: $(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}_k)^{-1}\mathbf{X}^T\mathbf{y}$

L2 (ridge): minimize magnitude of weights

$O(nk^2 + k^3)$ compute, $O(nk + k^2)$ storage

Matrix product as sum of outer products (distribute)

Gradient descent: $O(nk)$ dist. compute, $O(k)$ local storage

$\mathbf{w}_{i+1} = \mathbf{w}_i - \alpha_i \sum_{j=1}^{n}(\mathbf{w}_i^T\mathbf{x}^{(i)} - y^{(j)})\mathbf{x}^{(j)}$

Network is expensive, compute more, communicate less (reduce iterations)

Distributed PCA

Case 1: Big n, small k, $O(k^2)$ local storage, $O(k^3)$ local computation, $O(kr)$ communication

Center, covar matrix, eigendecomposition (locally), scores

Case 2: Big n, big k, $O(k)$ local storage and computation, $O(kr)$ communication

1. Communicate $\mathbf{v}_i \in \mathbb{R}^k$ to all workers
2. Compute $\mathbf{q}_i = \mathbf{X}^T\mathbf{X}\mathbf{v}_i$
3. Use $\mathbf{q}_i$ to update estimate of $\mathbf{P}$

Kernel Approximations

Nonlinear basis function for regression: $\mathbf{w}^T\phi(x)$

$\phi()$ maps the feature vector x to a new M-dim feature vector

Kernel trick: kernel matrix $\mathbf{K} = \phi\phi^T = k(\mathbf{X})$

- Polynomial: $(x_m^T x_n + c)^d$
- Gaussian RBF: $\exp(-||x_m - x_n||_2^2/2\sigma^2)$

Allows us to encode data in high-dim features spaces, but requires $O(n^2)$ kernel matrix

Kernel Ridge Regression: $w^* = \Phi^T(\mathbf{K} + \lambda\mathbf{I})^{-1}y$

$\alpha = (\mathbf{K} + \lambda\mathbf{I})^{-1}y$

Φ is the design matrix of transformed features

$\hat{y}(x) = \sum_{i=1}^{n}\alpha_i\mathbf{K}(x, x_i)$

Random Fourier Features: approximate inner products of kernel matrix with randomized map $\mathbf{z} : \mathbb{R}^k \to \mathbb{R}^R$

$k(x_i, x_j) = \phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j) \sim \mathbf{z}(\mathbf{x})$

Bochner's Theorem: $k(x - y) = \int p(w)\exp(iw^T(x - y))\,dw$

Find inverse Fourier transform $q(w)$ s.t. $\hat{q} = \phi$

Sample d weight vectors $w_\ell \sim q$

and phase shifts $b_i \sim \text{Uniform}(0, 2\pi)$

Compute features $z(x) = \sqrt{\frac{2}{D}}\cos(w^T x + b)$

Low-rank Approx: $\mathbf{K} = \Phi\Phi^T \in \mathbb{R}^{n\times n}$ is an SPSD matrix

SVD: $\mathbf{K} = \mathbf{U\Sigma U}^T$, $\mathbf{K}_r = \mathbf{U}_r\Sigma_r\mathbf{U}_r^T$

Goal: approximate SVD for large-scale kernel methods, minimize reconstruction error $\|\mathbf{K} - \tilde{\mathbf{K}}\|_F$

Column-sampling: sample r columns of $\mathbf{K}$, SVD of $\mathbf{C} : O(nr^2)$ SVD on $\mathbf{C}$ provides good approximation of full SVD

Nyström Method: better on low-rank structures

1. Sample r columns of $\mathbf{K}$
2. Sample the same rows of $\mathbf{C}$
3. SVD / inverse of $\mathbf{W} : O(r^3)$
4. Approximate $\tilde{\mathbf{K}} = \mathbf{CW}^{-1}\mathbf{C}^T : O(nr^2)$

Logistic Regression

Learn mapping ($\mathbf{w}$) that minimizes logistic loss on training data: $\min_{\mathbf{w}}\sum_{i=1}^{n}\ell_{\log}\left(y^{(i)} \cdot \mathbf{w}^T\mathbf{x}^{(i)}\right)$

$\mathbf{w}_{i+1} = \mathbf{w}_i - \alpha_i\sum_{j=1}^{n}(\sigma(y^{(j)\mathbf{w}_i^T\mathbf{x}^{(j)}}) - 1)y^{(j)}\mathbf{x}^{(j)}$

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

$O(nk)$ distributed compute, $O(k)$ local storage

Outputs probability, we can adjust threshold to tradeoff between FPR and TPR, maximize area under curve

One-Hot Encoding

Create a dummy feature for each category e.g. {'ARG', 'FRA', 'USA'} 'ARG' → $[1, 0, 0]$

Doesn't introduce spurious relationships

Always sparse, can be representated as index-value pairs assuming all other entries are zero.

Bag of words: represent documents with vocab of words

OHE increases dimensionality, inefficient learning (need larger n for larger k), more communication/storage

Feature Hashing: map features to buckets, output vectors size number of buckets

hashed features can be computed locally and minimize communication

hash kernels preserve dot products

Count-min Sketch

Hash input into buckets, take minimum across all hash functions

Will never underestimate, overestimate error bounded given $r > \log_2(\frac{1}{\delta})$ hash functions, each with $m > \frac{2}{\epsilon}$ buckets, $P[\hat{c}_s \geq c_s + \epsilon\|\mathbf{c}\|_1] \leq \delta$

Locality-Sensitive Hashing

map feature vector to a bit vector and ensure if $\mathbf{x}, \mathbf{y}$ are similar, then $\mathbf{x}', \mathbf{y}'$ are similar; use random projections

1. repeat b times:
2. pick a random hyperplane $\mathbf{z}$ by picking random weights
3. compute inner product of $\mathbf{z}$ with $\mathbf{x} :< \mathbf{z}, \mathbf{x} >$
4. if $< \mathbf{z}, \mathbf{x} >\geq 0 \to 1$, else 0

Data Curation

Lots of duplicate data on the internet

Data pruning can improve models past power laws

Custom crawlers fix broken formatting, remove irrelevant stuff, task specific crawlers, more geographic diversity

Geometric Curation: Cluster text/embeddings, use to:

- remove redundancies
- select salient examples
- balance cluster sizes

Duplicates have less utility, deduping reduces train time & memorization

Semantic deduplication: cluster into fine mini-clusters, keep centroids

Class balancing: clip to a maximum of each example

Quality Filtering: heuristics or model-based

Heuristics: e.g. stop words, doc size, pagerank, karma

Model-based: predict if data is good or not (CLIP score, perplexity, . . .)

removing 1 bad example is equal to adding 3 good examples

Synthetic Data: train small model with data generated from big models

can use models to rephrase data, filling in gaps in style

synthetic data must be used to add diversity to your data

Data Mixing: web data is heterogenous, mix matters

annealing: upsample high quality data at the end of training

this is when learning rate is annealed to zero