

Optimization in Data Science

Stephen Wright

University of Wisconsin-Madison

UC-Irvine, October, 2018

Outline

Data Analysis, Machine Learning, Data Science

- Context: Data Science
- Relating Data Science and Optimization
- Formulating 14 specific data science problems as continuous optimization problems

Optimization and Data Science

Optimization is being revolutionized by its interactions with machine learning and data analysis.

- New algorithms, and new interest in *old* algorithms;
- Challenging formulations and new paradigms;
- Renewed emphasis on certain topics: convex optimization algorithms, complexity, structured nonsmoothness, now *nonconvex optimization*.
- Large research community now working on the machine learning / optimization spectrum. The optimization / ML interface is a key component of many top conferences (ISMP, SIOPT, NIPS, ICML, COLT, ICLR, AISTATS, ...) and journals (Math Programming, SIOPT, ...).

Data Science

Related Terms: AI, Data Analysis, Machine Learning, Statistical Inference, Data Mining.

- Extract meaning from data: Understand statistical properties, learn important features and fundamental structures in the data.
- Use this knowledge to make predictions about other, similar data.

Highly multidisciplinary area!

- Foundations in Statistics;
- Computer Science: AI, Machine Learning, Databases, Parallel Systems, Architectures (GPUs);
- **Optimization** provides a toolkit of modeling / formulation and algorithmic techniques.

Modeling and domain-specific knowledge is vital: “80% of data analysis is spent on the process of cleaning and preparing the data.”
[Dasu and Johnson, 2003].

(Most academic research deals with the other 20%.)

Typical Setup

After cleaning and formatting, obtain a data set of m objects:

- Vectors of features: $a_j, j = 1, 2, \dots, m$.
- Outcome / observation / label y_j for each feature vector.

The outcomes y_j could be:

- a **real number**: **regression**
- a **label** indicating that a_j lies in one of M classes (for $M \geq 2$):
classification
- **no labels** (y_j is null):
 - ▶ **subspace identification**: Locate low-dimensional subspaces that approximately contain the (high-dimensional) vectors a_j ;
 - ▶ **clustering**: Partition the a_j into a few clusters.

(Structure may reveal which features in the a_j are important / distinctive, or enable predictions to be made about new vectors a .)

Fundamental Data Analysis Task

Seek a function ϕ that:

- approximately maps a_j to y_j for each j : $\phi(a_j) \approx y_j, j = 1, 2, \dots, m$.
- if there are no labels y_j , or if some labels are missing, seek ϕ that does something useful with the data $\{a_j\}$, e.g. assigns each a_j to an appropriate cluster or subspace.
- satisfies some additional properties — simplicity, structure — that make it “plausible” for the application, robust to perturbations in the data, **generalizable** to data instances beyond the training set.

Can usually **define ϕ in terms of some parameter vector x** — thus identification of ϕ becomes a **data-fitting problem**: Find the best x .

Objective function in this problem often built up of m terms that capture mismatch between predictions and observations for data item (a_j, y_j) .

The process of finding ϕ is called **learning** or **training**.

What's the use of the mapping ϕ ?

- **Analysis:** ϕ — especially the parameter x that defines it — reveals structure in the data. Examples:
 - ▶ Feature selection: reveal the components of vectors a_j that are most important in determining the outputs y_j ;
 - ▶ Uncovers some hidden structure, e.g.
 - ★ reveals low-dimensional subspaces that contain the a_j ;
 - ★ find clusters that contain the a_j ;
 - ★ defines a decision tree that defines the mapping $a_j \rightarrow y_j$.
- **Prediction:** Given new data vectors a_k , predict outputs $y_k \leftarrow \phi(a_k)$.

Complications

The data items (a_j, y_j) available for training and testing are viewed as an **empirical sample** drawn from some **underlying reality**. Want our conclusions to generalize to the unknown underlying set.

- **noise or errors** in a_j and y_j . Would like ϕ (and x) to be robust to such errors. **Regularized** formulations are used.
- **avoid overfitting** to the training data. Again, **generalization / regularization** can be used.
- **missing data**: Vectors a_j may be missing elements (but may still contain useful information).
- **missing labels**: Some or all y_j may be missing or null — semi-supervised or unsupervised learning.
- **online learning**: Data (a_j, y_j) arrives in a stream.

Continuous Optimization and Data Analysis

Optimization is a major source of algorithms for machine learning and data analysis.

- **Optimization Formulations** translate statistical principles (e.g. risk, likelihood, significance, generalizability) into measures and functions that can be solved algorithmically.
- **Optimization Algorithms** provide practical means to solve these problems, but they must be tailored to the structure and context.
- **Duality** is valuable in several cases (e.g. kernel learning).
- **Nonsmoothness** appears often as a regularization device, but often in a highly structured way that can be exploited by algorithms.

ML's Influence on (Continuous) Optimization

The needs of ML (including summation form, nonsmooth regularization) has caused revival, reexamination, and development of known approaches, particularly first-order and “zero order” methods.

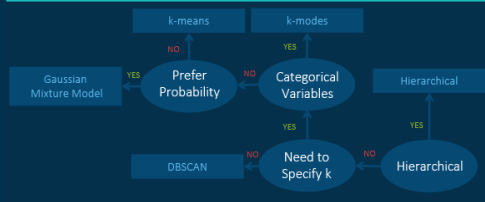
- stochastic gradient
- accelerated gradient
- coordinate descent
- conditional gradient (Frank-Wolfe)
- sparse and regularized optimization e.g. forward-backward.
- augmented Lagrangian, ADMM
- (sampled Newton and quasi-Newton)

This trend continues in nonconvex formulations:

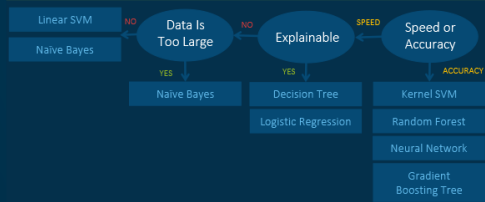
- stochastic gradient
- steepest descent (+ noise)
- trust-region methods
- Nonlinear conjugate gradient, L-BFGS, Newton-conjugate gradient.

Machine Learning Algorithms Cheat Sheet

Unsupervised Learning: Clustering

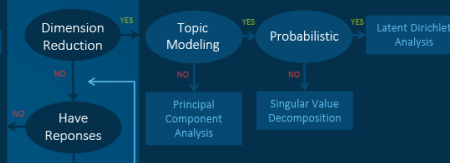


Supervised Learning: Classification

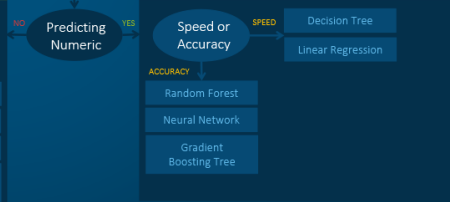


Unsupervised Learning: Dimension Reduction

START



Supervised Learning: Regression



¹<https://blogs.sas.com/content/subconsciousmusings/2017/04/12/machine-learning-algorithm-use/>

Machine Learning Algorithms Cheat Sheet



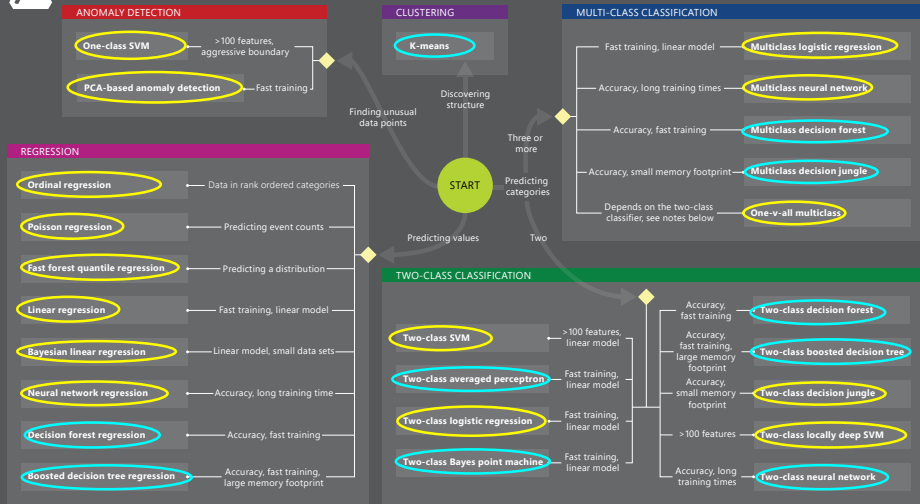
There's a lot of continuous optimization here (yellow)!

Microsoft Azure cheat sheet (optimization in yellow)



Microsoft Azure Machine Learning: Algorithm Cheat Sheet

This cheat sheet helps you choose the best Azure Machine Learning Studio algorithm for your predictive analytics solution. Your decision is driven by both the nature of your data and the question you're trying to answer.



Application I: (Linear) Least Squares

$$\min_x f(x) := \frac{1}{2} \sum_{j=1}^m (a_j^T x - y_j)^2 = \frac{1}{2} \|Ax - y\|_2^2.$$

[Gauss, 1799], [Legendre, 1805]; see [Stigler, 1981].

Here the function mapping data to output is linear: $\phi(a_j) = a_j^T x$.

- ℓ_2 regularization reduces sensitivity of the solution x to **noise in y** .

$$\min_x \frac{1}{2} \|Ax - y\|_2^2 + \lambda \|x\|_2^2.$$

- ℓ_1 regularization yields solutions x with few nonzeros:

$$\min_x \frac{1}{2} \|Ax - y\|_2^2 + \lambda \|x\|_1.$$

Feature selection: Nonzero locations in x indicate important components of a_j .

Application II: Robust Linear Regression

Least squares motivated statistically by assumption of Gaussian errors in observations y_j . What if the errors are distributed otherwise, or contain “outliers”?

Use statistics to write down a likelihood function for x given y , then find the maximum likelihood estimate — optimization! General form is

$$\min_x \frac{1}{m} \sum_{j=1}^m \ell(a_j^T x - y_j) + \lambda R(x)$$

where ℓ is loss function and R is regularizer.

ℓ and R could be convex or nonconvex.

- Tukey biweight: $\ell(\theta) = \theta^2 / (1 + \theta^2)$. Nonconvex: outliers don't affect solution much.
- Nonconvex separable regularizers R such as SCAD and MCP behave like $\|\cdot\|_1$ at zero, but flatten out for larger x .

Application III: Matrix Completion

Regression over a structured matrix: Observe a matrix X by probing it with linear operators $\mathcal{A}_j(X)$, giving observations y_j , $j = 1, 2, \dots, m$. Solve a regression problem:

$$\min_X \frac{1}{2m} \sum_{j=1}^m (\mathcal{A}_j(X) - y_j)^2 = \frac{1}{2m} \|\mathcal{A}(X) - y\|_2^2.$$

Each \mathcal{A}_j may observe a single element of X , or a linear combination of elements. Can be represented as a matrix A_j , so that $\mathcal{A}_j(X) = \langle A_j, X \rangle$.

Seek the “simplest” X that satisfies the observations. Nuclear-norm (sum-of-singular-values) regularization term induces low rank on X :

$$\min_X \frac{1}{2m} \|\mathcal{A}(X) - y\|_2^2 + \lambda \|X\|_*, \quad \text{for some } \lambda > 0.$$

[Recht et al., 2010]

Explicit Low-Rank Parametrization

Compact, nonconvex formulation is obtained by parametrizing X directly:

$$X = LR^T, \quad \text{where } L \in \mathbb{R}^{m \times r}, R \in \mathbb{R}^{n \times r},$$

where r is known (or suspected) rank.

$$\min_{L,R} \frac{1}{2m} \sum_{j=1}^m (\mathcal{A}_j(LR^T) - y_j)^2.$$

(No need for regularizer — **rank is hard-wired** into the formulation.)

Despite the nonconvexity, near-global minima can be found when \mathcal{A}_j are **incoherent**. Use appropriate initialization [Candès et al., 2014], [Zheng and Lafferty, 2015] or the observation that all local minima are near-global [Bhojanapalli et al., 2016].

Application IV: Nonnegative Matrix Factorization

Given $m \times n$ matrix Y , seek factors L ($m \times r$) and R ($n \times r$) that are element-wise positive, such that $LR^T \approx Y$.

$$\min_{L,R} \frac{1}{2} \|LR^T - Y\|_F^2 \text{ subject to } L \geq 0, R \geq 0.$$

Applications in computer vision, document clustering, chemometrics, ...

Could combine with matrix completion, when not all elements of Y are known, if it makes sense on the application to have nonnegative factors.

If positivity constraint were not present, could solve this in closed form with an SVD, since Y is observed completely.

Application V: Sparse Inverse Covariance

Let $Z \in \mathbb{R}^p$ be a (vector) random variable with zero mean. Let z_1, z_2, \dots, z_N be samples of Z . Sample covariance matrix (estimates covariance between components of Z):

$$S := \frac{1}{N-1} \sum_{\ell=1}^N z_\ell z_\ell^T.$$

Seek a **sparse inverse covariance matrix**: $X \approx S^{-1}$.

X reveals dependencies between components of Z : $X_{ij} = 0$ if the i and j components of Z are conditionally independent.

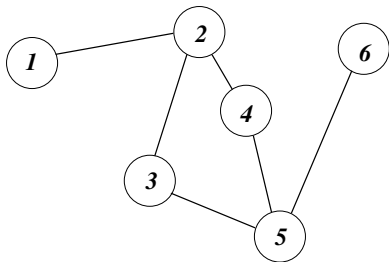
Do nodes i and j influence each other directly, or only indirectly via other nodes?

Obtain X from the regularized formulation:

$$\min_X \langle S, X \rangle - \log \det(X) + \lambda \|X\|_1, \quad \text{where } \|X\|_1 = \sum_{i,j} |X_{ij}|.$$

[d'Aspremont et al., 2008, Friedman et al., 2008].

Reveals Network Structure. Example with $p = 6$.



$$X = \begin{bmatrix} * & * & 0 & 0 & 0 & 0 \\ * & * & * & * & 0 & 0 \\ 0 & * & * & 0 & * & 0 \\ 0 & * & 0 & * & * & 0 \\ 0 & 0 & * & * & * & * \\ 0 & 0 & 0 & 0 & * & * \end{bmatrix}$$

Application VI: Sparse Principal Components (PCA)

Seek **sparse** approximations to the leading eigenvectors of the sample covariance matrix S .

For the leading sparse principal component, solve

$$\max_{v \in \mathbb{R}^n} v^T S v = \langle S, v v^T \rangle \quad \text{s.t. } \|v\|_2 = 1, \|v\|_0 \leq k,$$

for some given $k \in \{1, 2, \dots, n\}$. Convex relaxation replaces $v v^T$ by an $n \times n$ positive semidefinite proxy M :

$$\max_{M \in \mathbb{S}^{n \times n}} \langle S, M \rangle \quad \text{s.t. } M \succeq 0, \langle I, M \rangle = 1, \|M\|_1 \leq R,$$

where $\|\cdot\|_1$ is the sum of absolute values [d'Aspremont et al., 2007].

Adjust the parameter R to obtain desired sparsity.

Could also get a nonconvex relaxation by replacing $\|v\|_0$ by $\|v\|_1$:

$$\max_{v \in \mathbb{R}^n} v^T S v \quad \text{s.t. } \|v\|_2 = 1, \|v\|_1 \leq R.$$

Application VII: Sparse + Low-Rank

Given $Y \in \mathbb{R}^{m \times n}$, seek low-rank M and sparse S such that $M + S \approx Y$.

Applications:

- Robust PCA: Sparse S represents “outlier” observations.
- Foreground-Background separation in video processing.
 - ▶ Each column of Y is one frame of video, each row is a single pixel evolving in time.
 - ▶ Low-rank part M represents background, sparse part S represents foreground.

Convex formulation:

$$\min_{M,S} \|M\|_* + \lambda \|S\|_1 \quad \text{s.t. } Y = M + S.$$

[Candès et al., 2011, Chandrasekaran et al., 2011]

Sparse + Low-Rank: Compact Formulation

Compact formulation: Variables $L \in \mathbb{R}^{n \times r}$, $R \in \mathbb{R}^{m \times r}$, $S \in \mathbb{R}^{m \times n}$ sparse.

$$\min_{L,R,S} \frac{1}{2} \|LR^T + S - Y\|_F^2 + \lambda \|S\|_1 \quad (\text{fully observed})$$

$$\min_{L,R,S} \frac{1}{2} \|P_\Phi(LR^T + S - Y)\|_F^2 + \lambda \|S\|_1 \quad (\text{partially observed}),$$

where Φ represents the locations of the observed entries.

[Chen and Wainwright, 2015, Yi et al., 2016].

Application VIII: Subspace Identification

Given vectors $a_j \in \mathbb{R}^n$ with **missing entries**, find a subspace of \mathbb{R}^n such that all “completed” vectors a_j lie approximately in this subspace.

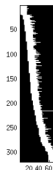
If $\Omega_j \subset \{1, 2, \dots, n\}$ is the set of observed elements in a_j , seek $X \in \mathbb{R}^{n \times d}$ such that

$$[a_j - Xs_j]_{\Omega_j} \approx 0,$$

for some $s_j \in \mathbb{R}^d$ and all $j = 1, 2, \dots$

[Balzano et al., 2010, Balzano and Wright, 2014].

Application: Structure from motion. Reconstruct opaque object from planar projections of surface reference points.



Application IX: Linear Support Vector Machines

Each item of data belongs to one of two classes: $y_j = +1$ and $y_j = -1$.

Seek (x, β) such that

$$a_j^T x - \beta \geq 1 \quad \text{when } y_j = +1;$$

$$a_j^T x - \beta \leq -1 \quad \text{when } y_j = -1.$$

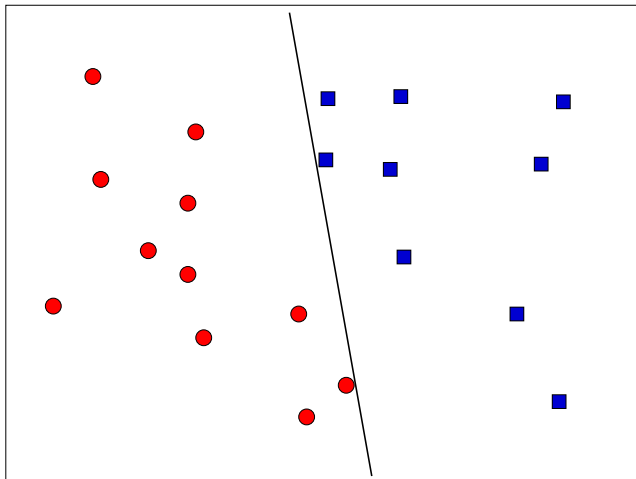
The mapping is $\phi(a_j) = \text{sign}(a_j^T x - \beta)$.

Design an objective so that the j th loss term is zero when $\phi(a_j) = y_j$, positive otherwise. A popular one is **hinge loss**:

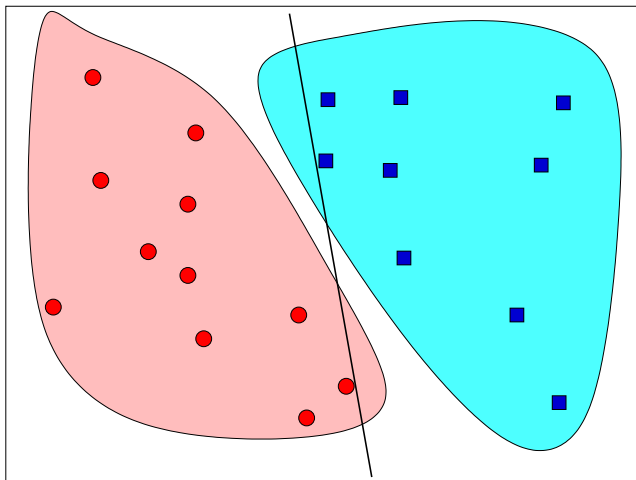
$$H(x, \beta) = \frac{1}{m} \sum_{j=1}^m \max(1 - y_j(a_j^T x - \beta), 0).$$

Add a **regularization term** $(\lambda/2)\|x\|_2^2$ for some $\lambda > 0$ to maximize the margin between the classes.

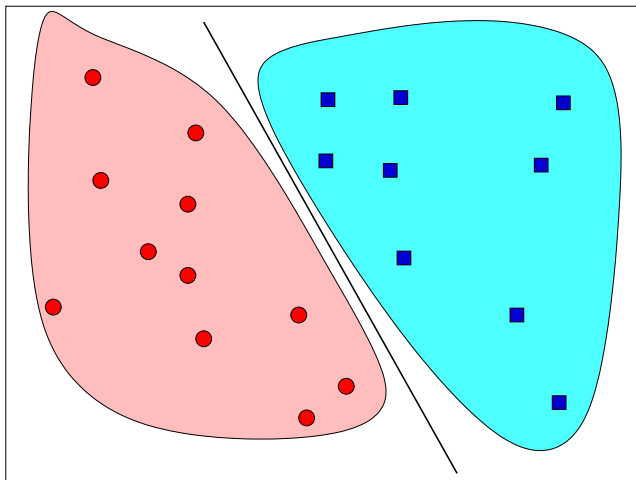
Regularize for Generalizability



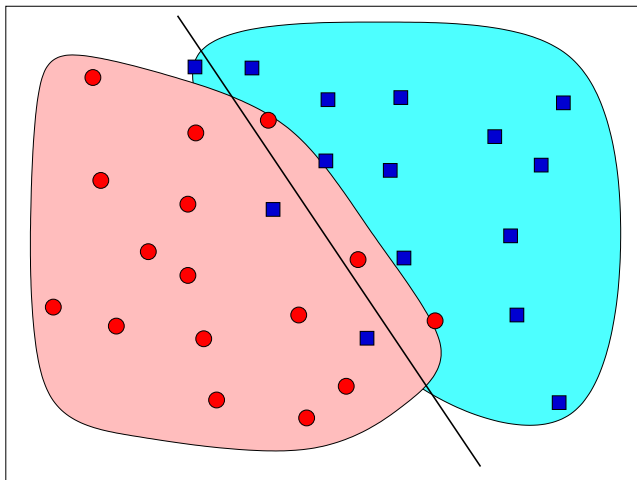
Regularize for Generalizability



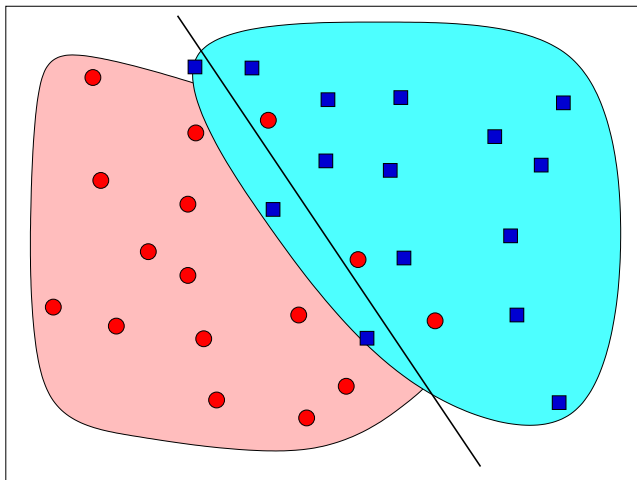
Regularize for Generalizability



Regularize for Generalizability



Regularize for Generalizability



Application X: Nonlinear SVM

Data a_j , $j = 1, 2, \dots, m$ may not be *separable* neatly into two classes $y_j = +1$ and $y_j = -1$. Apply a nonlinear transformation $a_j \rightarrow \psi(a_j)$ (“lifting”) to make separation more effective. Seek (x, β) such that

$$\psi(a_j)^T x - \beta \geq 1 \quad \text{when } y_j = +1;$$

$$\psi(a_j)^T x - \beta \leq -1 \quad \text{when } y_j = -1.$$

Leads to the formulation:

$$\min_x \frac{1}{m} \sum_{j=1}^m \max(1 - y_j(\psi(a_j)^T x - \beta), 0) + \frac{1}{2} \lambda \|x\|_2^2.$$

Can avoid defining ψ explicitly by using instead the **dual** of this QP.

Nonlinear SVM: Dual

Dual is a quadratic program in m variables, with simple constraints:

$$\min_{\alpha \in \mathbb{R}^m} \frac{1}{2} \alpha^T Q \alpha - e^T \alpha \quad \text{s.t.} \quad 0 \leq \alpha \leq (1/\lambda)e, \quad y^T \alpha = 0.$$

where $Q_{k\ell} = y_k y_\ell \psi(a_k)^T \psi(a_\ell)$, $y = (y_1, y_2, \dots, y_m)^T$, $e = (1, 1, \dots, 1)^T$.

No need to choose $\psi(\cdot)$ explicitly. Instead choose a kernel K , such that

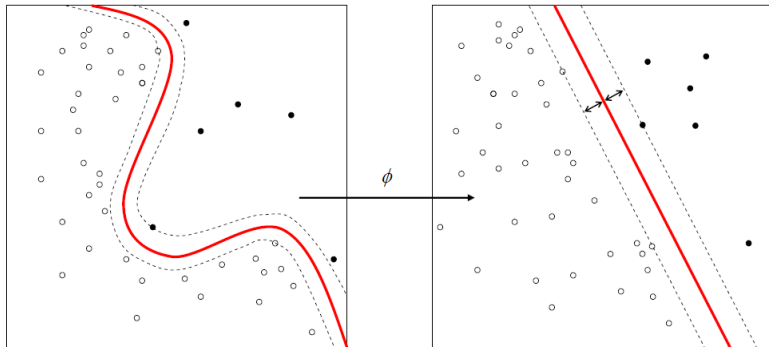
$$K(a_k, a_\ell) \sim \psi(a_k)^T \psi(a_\ell).$$

[Boser et al., 1992, Cortes and Vapnik, 1995]. “Kernel trick.”

Gaussian kernels are popular:

$$K(a_k, a_\ell) = \exp(-\|a_k - a_\ell\|^2 / (2\sigma)), \quad \text{for some } \sigma > 0.$$

Nonlinear SVM



Application XI: Logistic Regression

Binary logistic regression is similar to binary SVM, except that we seek a function p that gives **odds** of data vector a being in class 1 or class -1 , rather than making a simple prediction.

Seek odds function p parametrized by $x \in \mathbb{R}^n$:

$$p(a; x) := (1 + e^{a^T x})^{-1}.$$

Choose x so that $p(a_j; x) \approx 1$ when $y_j = 1$ and $p(a_j; x) \approx 0$ when $y_j = -1$.

Choose x to minimize a negative log likelihood function:

$$\mathcal{L}(x) = -\frac{1}{m} \left[\sum_{y_j=-1} \log(1 - p(a_j; x)) + \sum_{y_j=1} \log p(a_j; x) \right]$$

Sparse solutions x are interesting because they indicate which components of a_j are critical to classification. Can solve: $\min_z \mathcal{L}(z) + \lambda \|z\|_1$.

Multiclass Logistic Regression

Have M classes instead of just 2. M can be large e.g. identify phonemes in speech, identify line outages in a power grid.

Labels $y_{j\ell} = 1$ if data point j is in class ℓ ; $y_{j\ell} = 0$ otherwise; $\ell = 1, \dots, M$.

Find subvectors $x_{[\ell]}$, $\ell = 1, 2, \dots, M$ such that if a_j is in class k we have

$$a_j^T x_{[k]} \gg a_j^T x_{[\ell]} \quad \text{for all } \ell \neq k.$$

Find $x_{[\ell]}$, $\ell = 1, 2, \dots, M$ by minimizing a negative log-likelihood function:

$$f(x) = -\frac{1}{m} \sum_{j=1}^m \left[\sum_{\ell=1}^M y_{j\ell} (a_j^T x_{[\ell]}) - \log \left(\sum_{\ell=1}^M \exp(a_j^T x_{[\ell]}) \right) \right]$$

Can use **group LASSO** regularization terms to select important features from the vectors a_j , by imposing a common sparsity pattern on all $x_{[\ell]}$.

Application XII: Atomic-Norm Regularization

Seek an approx minimizer of $f(x)$ such that x combines a small number of fundamental elements — **atoms**. Define the **atomic norm** of x via its “minimal” representation in terms of the set \mathcal{A} of atoms (possibly infinite):

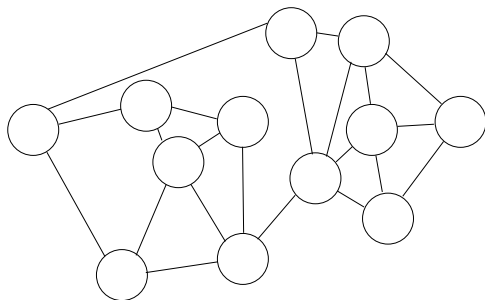
$$\|x\|_{\mathcal{A}} := \inf \left\{ \sum_{a \in \mathcal{A}} c_a : x = \sum_{a \in \mathcal{A}} c_a a, c_a \geq 0 \right\}.$$

- Compressed Sensing: $x \in \mathbb{R}^n$: Atoms are $\pm e_i$, where $e_i = (0, 0, \dots, 0, 1, 0, \dots, 0)^T$; then $\|x\|_{\mathcal{A}} = \|x\|_1$.
- Low-rank Matrix Problems: $x \in \mathbb{R}^{m \times n}$: Atoms are the rank-one matrices (infinite); then $\|\cdot\|_{\mathcal{A}}$ is the nuclear norm.
- Signal processing with few frequencies: $x(t) = \sum_{j=1}^k c_j \exp(2\pi i f_j t)$: signal with k frequencies. Atoms are $a_f := \exp(2\pi i f t)$ for any f .
- Image processing: Atoms are subtrees of wavelet coefficients.

Can solve with Frank-Wolfe, gradient projection [Rao et al., 2015].

Application XIII: Community Detection in Graphs

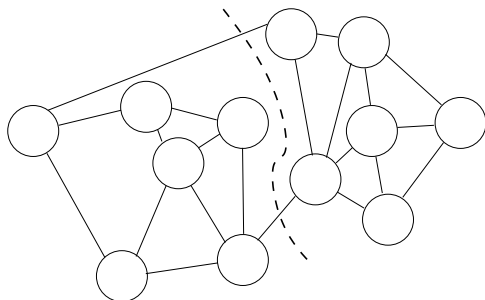
Given an undirected graph, find “communities” (subsets of nodes) such that nodes inside a given community are more likely to be connected to each other than to nodes outside that community.



Probability $p \in (0, 1)$ of being connected to a node *within* your community, and $q \in (0, p)$ of being connected to a node *outside* your community.

Application XIII: Community Detection in Graphs

Given an undirected graph, find “communities” (subsets of nodes) such that nodes inside a given community are more likely to be connected to each other than to nodes outside that community.



Probability $p \in (0, 1)$ of being connected to a node *within* your community, and $q \in (0, p)$ of being connected to a node *outside* your community.

Community Detection: Formulation

Given adjacency matrix A such that

$$A_{ij} = \begin{cases} 1 & \text{if nodes } i \text{ and } j \text{ are connected} \\ 0 & \text{if nodes } i \text{ and } j \text{ are } \textit{not} \text{ connected,} \end{cases}$$

together with probabilities p and q (both assumed known), find a **partition matrix** X that defines the communities:

$$X_{ij} = \begin{cases} 1 & \text{if nodes } i \text{ and } j \text{ are in same community} \\ 0 & \text{if nodes } i \text{ and } j \text{ are } \textit{not} \text{ in same community.} \end{cases}$$

Likelihood function is

$$p(A_{ij} | X_{ij}, p, q) = \begin{cases} p & \text{if } A_{ij} = 1 \text{ and } X_{ij} = 1 \\ 1 - p & \text{if } A_{ij} = 0 \text{ and } X_{ij} = 1 \\ q & \text{if } A_{ij} = 1 \text{ and } X_{ij} = 0 \\ 1 - q & \text{if } A_{ij} = 0 \text{ and } X_{ij} = 0. \end{cases}$$

Community Detection: Formulation

The max-log-likelihood problem has the form

$$\max_{\text{partition matrix } X} \langle X, A - \lambda J_n \rangle,$$

where J_n is the $n \times n$ matrix of all ones and

$$\lambda = \frac{\log(1 - q) - \log(1 - p)}{\log p - \log q + \log(1 - q) - \log(1 - p)}.$$

[Li et al., 2018] This is not tractable, so seek relaxations.

Community Detection: Relaxations

Convex relaxation:

$$\max_X \langle X, A - \lambda J_n \rangle, \quad \text{s.t. } X \succeq 0, X \geq 0, X_{ii} = 1, i = 1, 2, \dots, n.$$

When there are just two communities of similar size, then under some (strong) conditions on p, q , this relaxation will recover the correct X [Li et al., 2018, Theorem 2.1].

Nonconvex relaxation: Write $X = \frac{1}{2}(xx^T + J_n)$, where ideally $x \in \mathbb{R}^n$ has

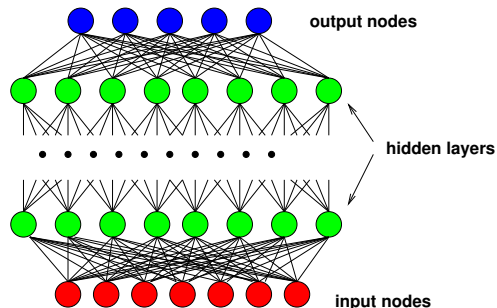
$$x_i = \begin{cases} 1 & \text{if } i \text{ is in community 1} \\ -1 & \text{if } i \text{ is in community 2.} \end{cases}$$

Then write

$$\max_x \langle xx^T + J_n, A - \lambda J_n \rangle, \quad \text{s.t. } \|x\|_2 \leq 1,$$

and replace $x_i \leftarrow \text{sign}(x_i)$ for all i to get a prediction.

Application XIV: Deep Learning



Inputs are the vectors a_j , outputs are a prediction about a_j belonging to each class (e.g. multiclass logistic regression).

At each layer, inputs are converted to outputs by a **linear transformation** composed with an **element-wise function** σ :

$$a^{\ell+1} = \sigma(W^\ell a^\ell + g^\ell),$$

where a^ℓ is node values at layer ℓ , (W^ℓ, g^ℓ) are *parameters* in the network. Nowadays, the ReLU function $\sigma(t) = \max(t, 0)$ is popular.

Training Deep Learning Networks

The network contains many **parameters** — (W^ℓ, g^ℓ) , $\ell = 1, 2, \dots, L$ — that are selected by **training** on the data (a_j, y_j) , $j = 1, 2, \dots, m$.

Objective has the form:

$$\frac{1}{m} \sum_{j=1}^m h(x; a_j, y_j)$$

where $x = (W^1, g^1, W^2, g^2, \dots)$ are the parameters in the model and h measures the mismatch between observed output y_j and the outputs produced by the model.

Nonlinear, Nonconvex, Nonsmooth.

Many software packages available for training: Caffe, PyTorch, Tensor Flow, Theano,... Many run on GPUs.

Overparametrization in Neural Networks

Number of parameters (elements in x) is often vastly greater than the number of data points — sometimes by 1-2 orders of magnitude — yet recent experience shows that “overfitting” is not necessarily a problem!

Training such networks can often achieve “zero loss,” that is, all items in the training data set are correctly classified. Two big questions arise.

1. Isn't this **overfitting**? Yet such models often generalize well, flouting conventional wisdom.
2. Why is Stochastic Gradient (SGD) reliably finding the global minimum of a nonsmooth, nonlinear, nonconvex problem?

We have only started to get some intuition on these issues. See for example [Li and Liang, 2018].

Overparametrization

In an overparametrized network, we suspect that:

- There are many solutions;
- A randomly chosen initial point for the weights will be close to one of the solutions;
- You have to change few (if any) ReLU activations to get from the initial point to the solution;
- Gradient descent (or stochastic gradient descent with big enough batches) will get us from the initial point to the solution efficiently.

This remains an active area of investigation, with important consequences for the understanding of why neural networks are so effective in some applications.

Adversarial Machine Learning

Deep learning classifiers can be fooled with a carefully chosen attack.

(Szegedy et al, Dec 2013): Train digits in the MNIST set, apply **carefully chosen** perturbation. NN misclassifies, even though it's visually obvious what the digit should be.



(a) Even columns: adversarial examples for a linear (softmax) classifier (stddev=0.06)



(b) Even columns: adversarial examples for a 200-200-10 sigmoid network (stddev=0.063)



(c) Randomly distorted samples by Gaussian noise with stddev=1. Accuracy: 51%.

Note that a random perturbation is OK, even when large. But a small, carefully crafted perturbation causes misclassification.

Adversarial ML: The Issues

1. Can we generate efficiently the “carefully chosen perturbations” that break the classifier?
2. Can we **train** the network to be **robust** to perturbations of a certain size?
3. Can we **verify** that a given network will continue to give the same classification when we perturb a given training example x by any perturbation of a given size $\epsilon > 0$?

For 1, various optimization formulations have been proposed, depending on the type of classification done. Usually **constrained nonlinear**.

To implement 2, we can use **robust optimization** techniques (but these are expensive) or selectively generate perturbed data examples and re-train.

Mixed-integer programming formulations have been devised for 3. Very expensive even for small networks and data sets (e.g. MNIST, CIFAR).

Training for Robustness

Instead of incurring a loss $h(x; a_j, y_j)$ for parameters x and data item (a_j, y_j) as above, define the loss to be the **worst possible loss for all a within a ball of radius ϵ centered at a_j** . That is,

$$\max_{v_j: \|v_j\| \leq \epsilon} h(x; a_j + v_j, y_j).$$

Thus, the training problem becomes the following min-max problem:

$$\min_x \frac{1}{m} \sum_{j=1}^m \max_{v_j: \|v_j\| \leq \epsilon} h(x; a_j + v_j, y_j).$$

The inner “max” problems can at least be solved in parallel and sometimes in closed form. We can also often generate a generalized gradient w.r.t. x , and so implement a first-order method for the outer loop.

But this is expensive in general!

Summary

Continuous optimization provides powerful frameworks for formulating and solving problems in data analysis and machine learning.

I haven't talked about the contributions of discrete optimization, which are being promoted more and more.

BUT it's usually not enough to just formulate these problems and use off-the-shelf optimization technology to solve them. The algorithms need to be customized to the problem structure (in particular, large amount of data) and the context.

Research in this area has exploded over the past decade and is still going strong, with a great many unanswered questions. (Many of them in deep learning.)

References I



Balzano, L., Nowak, R., and Recht, B. (2010).
Online identification and tracking of subspaces from highly incomplete information.
In *48th Annual Allerton Conference on Communication, Control, and Computing*, pages 704–711.
<http://arxiv.org/abs/1006.4046>.



Balzano, L. and Wright, S. J. (2014).
Local convergence of an algorithm for subspace identification from partial data.
Foundations of Computational Mathematics, 14:1–36.
DOI: 10.1007/s10208-014-9227-7.



Bhojanapalli, S., Neyshabur, B., and Srebro, N. (2016).
Global optimality of local search for low-rank matrix recovery.
Technical Report arXiv:1605.07221, Toyota Technological Institute.





Boser, B. E., Guyon, I. M., and Vapnik, V. N. (1992).
A training algorithm for optimal margin classifiers.
In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, pages 144–152.





Candès, E., Li, X., and Soltanolkotabi, M. (2014).
Phase retrieval via a Wirtinger flow.
Technical Report arXiv:1407.1065, Stanford University.


References II

 Candès, E. J., Li, X., Ma, Y., and Wright, J. (2011).
Robust principal component analysis?
Journal of the ACM, 58.3:11.

 Chandrasekaran, V., Sanghavi, S., Parrilo, P. A., and Willsky, A. S. (2011).
Rank-sparsity incoherence for matrix decomposition.
SIAM Journal on Optimization, 21(2):572–596.

 Chen, Y. and Wainwright, M. J. (2015).
Fast low-rank estimation by projected gradient descent: General statistical and algorithmic guarantees.
Technical Report arXiv:1509.03025, University of California-Berkeley.

 Cortes, C. and Vapnik, V. N. (1995).
Support-vector networks.
Machine Learning, 20:273–297.

 d'Aspremont, A., Banerjee, O., and El Ghaoui, L. (2008).
First-order methods for sparse covariance selection.
SIAM Journal on Matrix Analysis and Applications, 30:56–66.

 d'Aspremont, A., El Ghaoui, L., Jordan, M. I., and Lanckriet, G. (2007).
A direct formulation for sparse PCA using semidefinite programming.
SIAM Review, 49(3):434–448.

References III



Dasu, T. and Johnson, T. (2003).
Exploratory Data Mining and Data Cleaning.
John Wiley & Sons.



Friedman, J., Hastie, T., and Tibshirani, R. (2008).
Sparse inverse covariance estimation with the graphical lasso.
Biostatistics, 9(3):432–441.



Li, X., Chen, Y., and Xu, J. (2018).
Convex relaxation methods for community detection.
Technical Report arXiv:1810.00315v1, University of California-Davis.



Li, Y. and Liang, Y. (2018).
Learning overparametrized neural networks via stochastic gradient descent on structured data.
Technical Report arXiv:1808.01204, University of Wisconsin-Madison.



Rao, N., Shah, P., and Wright, S. J. (2015).
Forward-backward greedy algorithms for atomic-norm regularization.
IEEE Transactions on Signal Processing, 63:5798–5811.
<http://arxiv.org/abs/1404.5692>.

References IV



Recht, B., Fazel, M., and Parrilo, P. (2010).

Guaranteed minimum-rank solutions to linear matrix equations via nuclear norm minimization.

SIAM Review, 52(3):471–501.



Stigler, S. M. (1981).

Gauss and the invention of least squares.

Annals of Statistics, 9(3):465–474.



Yi, X., Park, D., Chen, Y., and Caramanis, C. (2016).

Fast algorithms for robust pca via gradient descent.

Technical Report arXiv:1605.07784, University of Texas-Austin.



Zheng, Q. and Lafferty, J. (2015).

A convergent gradient descent algorithm for rank minimization and semidefinite programming from random linear measurements.

Technical Report arXiv:1506.06081, Statistics Department, University of Chicago.