Introduction to Optimization for Data Science

Fundamentals of Continuous Optimization - Enhanced Version

Optimization Methods

2025-11-21

Table of contents

Chapter Overview

This chapter introduces the fundamentals of **continuous optimization algorithms** for data science applications. We explore how machine learning, statistics, and data analysis problems can be formulated as optimization challenges.

Key Topics Covered: - Data analysis through optimization lens - Classical optimization problems (Least Squares, LASSO) - Matrix factorization and low-rank problems - Machine learning formulations (SVM, Logistic Regression) - Deep learning optimization challenges

Enhanced Features: - Improved mathematical notation and formatting - Corrected equation rendering - Enhanced readability and structure

1 Introduction

This book focuses on the **fundamentals of algorithms** for solving continuous optimization problems, which involve:

- **Minimizing functions** of multiple real-valued variables
- **Handling constraints** on variable values

- **Emphasizing convex problems** with data science applications
- **Connecting theory to practice** in machine learning, statistics, and data analysis

! Core Focus

Our choice of topics is **motivated by relevance to data science** — the formulations and algorithms discussed are directly useful for solving real-world problems in machine learning, statistics, and data analysis.

This chapter outlines several **key paradigms from data science** and demonstrates how they can be formulated as continuous optimization problems. Understanding the **smoothness properties and structure** of these formulations is crucial for selecting appropriate algorithms.

2 Data Analysis and Optimization

The typical optimization problem in data analysis involves finding a **model that balances** two competing objectives:

1. **Agreement with collected data**
2. **Adherence to structural constraints** reflecting our beliefs about good models

2.1 The Data Science Framework

In a typical analysis problem, our **dataset** consists of m objects:

$$D := \{(a_j, y_j), j = 1, 2, \ldots, m\} \tag{1}$$

where:

Features (a_j) - Vector or matrix of features - Input variables - Predictors

Labels/Observations (y_j)
- Target values - Responses - Outcomes

> **Data Preprocessing**
>
> We assume the data has been **cleaned** so that all pairs (a_j, y_j) have consistent size and shape.

2.2 The Learning Objective

The **data analysis task** consists of discovering a function ϕ such that:

$$\phi(a_j) \approx y_j \quad \text{for most } j = 1, 2, \ldots, m \tag{2}$$

> **Terminology**
>
> The process of discovering the mapping ϕ is often called **"learning"** or **"training"**.

The function ϕ is often defined in terms of a vector or matrix of parameters, which we denote by x or X. With these parametrizations, the problem of identifying ϕ becomes a traditional data-fitting problem: Find the parameters x defining ϕ such that $\phi(a_j) \approx y_j$ for $j = 1, 2, \ldots, m$ in some optimal sense.

Once we define the term "optimal" (and possibly also with restrictions on the parameter values), we have an optimization problem. Frequently, these optimization formulations have objective functions of the **finite-sum type**:

$$L_D(x) := \frac{1}{m} \sum_{j=1}^{m} \ell(a_j, y_j; x) \tag{3}$$

The function $\ell(a, y; x)$ represents a **"loss"** incurred for not properly aligning our prediction $\phi(a)$ with y. Thus, the objective $L_D(x)$ measures the average loss accrued over the entire dataset when the parameter vector equals x.

2.3 Model Learning and Prediction

Once an appropriate value of x (and thus ϕ) has been learned from the data, we can use it to make predictions about other items not in the set D. Given an unseen item $\hat{a}$ of the same type as a_j, we predict the label $\hat{y}$ associated with $\hat{a}$ to be $\phi(\hat{a})$.

The mapping ϕ may also expose other structures and properties in the dataset:

- **Feature selection**: Revealing that only a small fraction of features in a_j are needed to predict y_j
- **Dimensionality reduction**: When x is a matrix, it could reveal low-dimensional subspaces containing most vectors a_j
- **Structure discovery**: Matrices with particular structure (low-rank, sparse) such that observations prompted by feature vectors a_j yield results close to y_j

2.4 Problem Types by Label Structure

The form of the labels y_j determines the nature of the data analysis problem:

Classification of Problems

Regression Problems Each y_j is a real number $\rightarrow$ continuous prediction

Classification Problems Each y_j is a label from set $\{1, 2, \ldots, M\} \rightarrow$ discrete class prediction

- Binary classification: $M = 2$
- Multiclass classification: $M > 2$ (can be thousands in speech/image recognition)

Unsupervised Learning No labels y_j exist $\rightarrow$ discover structure in features a_j - Clustering: Group similar vectors

- Dimensionality reduction: Find low-dimensional representations

2.5 Regularization and Overfitting Prevention

One key consideration is avoiding **overfitting** the model to dataset D. The particular dataset available can be thought of as a finite sample from some underlying larger collection, and we want ϕ to perform well on unobserved data points.

The standard approach is **regularization** — modifying the objective function by adding constraints or penalty terms that limit the "complexity" of function ϕ:

$$\min_{x \in \Omega} L_D(x) + \lambda \cdot \text{pen}(x) \tag{4}$$

where: - Ω is the set of allowable values for x - pen$(\cdot)$ is a regularization function (regularizer) - $\lambda \geq 0$ is the regularization parameter

> **! Regularization Trade-off**
>
> The parameter λ balances: - **Small** λ: Better fit to training data D, higher model complexity - **Large** λ: Lower model complexity, better generalization

The constraint set Ω may exclude irrelevant parameter values. For example, in some applications we may require non-negative components: $\Omega = \{x : x_i \geq 0 \text{ for all } i\}$.

3 Least Squares

The oldest and best-known data analysis problem is **linear least squares**. Here, data points (a_j, y_j) lie in $\mathbb{R}^n \times \mathbb{R}$, and we solve:

$$\min_x \frac{1}{2m} \sum_{j=1}^{m} (a_j^T x - y_j)^2 = \frac{1}{2m} \|Ax - y\|_2^2 \tag{5}$$

where A is the matrix with rows a_j^T for $j = 1, 2, \ldots, m$ and $y = (y_1, y_2, \ldots, y_m)^T$.

The function ϕ is defined by $\phi(a) := a^T x$. We can introduce a nonzero intercept by adding parameter $\beta \in \mathbb{R}$ and defining $\phi(a) := a^T x + \beta$.

> **i Statistical Motivation**
>
> This formulation can be motivated as a maximum-likelihood estimate of x when observations y_j contain independent, identically distributed Gaussian noise.

3.1 Ridge Regression

Adding a squared ℓ_2-norm penalty yields **ridge regression**:

$$\min_x \frac{1}{2m} \|Ax - y\|_2^2 + \lambda \|x\|_2^2 \tag{6}$$

The solution x of this regularized formulation has less sensitivity to perturbations in the data (a_j, y_j).

3.2 LASSO Formulation

The **LASSO** (Least Absolute Shrinkage and Selection Operator) formulation uses ℓ_1 regularization:

$$\min_x \frac{1}{2m}\|Ax - y\|_2^2 + \lambda\|x\|_1 \tag{7}$$

This tends to yield solutions x that are **sparse** — containing relatively few nonzero components (**Tibshirani1996?**).

Feature Selection Benefits

Statistical Appeal: Predictors depending on few features are simpler and more comprehensible

Practical Appeal: For new data $\hat{a}$, we need only the "selected" features for prediction, not all components

The LASSO formulation is an important prototype involving regularization term $\lambda\|x\|_1$ that is **non-smooth and convex** but has relatively simple structure exploitable by algorithms.

4 Matrix Factorization Problems

Many data analysis problems require estimating a **low-rank matrix** from sparse collections of data. These extend least squares to problems where data A_j are matrices rather than vectors.

Changing notation, suppose each A_j is an $n \times p$ matrix, and we seek another $n \times p$ matrix X that solves:

$$\min_X \frac{1}{2m}\sum_{j=1}^{m}(\langle A_j, X\rangle - y_j)^2 \tag{8}$$

where $\langle A, B\rangle := \text{trace}(A^T B)$. We can think of the A_j as "probing" the unknown matrix X.

4.1 Common Observation Types

Random Linear Combinations Elements of A_j selected i.i.d. from some distribution

Single-Element Observations Each A_j has 1 in a single location, zeros elsewhere

4.2 Nuclear Norm Regularization

A regularized version leading to **low-rank solutions**:

$$\min_X \frac{1}{2m} \sum_{j=1}^{m} (\langle A_j, X \rangle - y_j)^2 + \lambda \|X\|_* \tag{9}$$

where $\|X\|_*$ is the **nuclear norm** — the sum of singular values of X (**Recht2010?**).

Nuclear Norm Properties

- Analogous to ℓ_1 norm for vectors
- ℓ_1 norm favors sparse vectors
- Nuclear norm favors low-rank matrices
- Non-smooth but convex → convex formulation

4.3 Factorized Representation

Another regularization approach represents X explicitly as a product of two "thin" matrices $L \in \mathbb{R}^{n \times r}$ and $R \in \mathbb{R}^{p \times r}$, with $r \ll \min(n, p)$:

$$\min_{L,R} \frac{1}{2m} \sum_{j=1}^{m} (\langle A_j, LR^T \rangle - y_j)^2 \tag{10}$$

Setting $X = LR^T$, the rank r is "hard-wired" into the definition, eliminating need for regularization.

Factorization Trade-offs

Advantages: - Much more compact: $(n + p)r \ll np$ parameters
- No explicit regularization needed
Challenges: - Non-convex in (L, R) jointly - Requires careful algorithmic strategy

Recent research (**BurerMonteiro2003?**) shows this non-convexity is often benign under certain data assumptions, yielding good solutions despite non-convexity.

4.4 Nonnegative Matrix Factorization

Applications in computer vision, chemometrics, and document clustering require factors with **nonnegative elements**:

$$\min_{L,R} \|LR^T - Y\|_F^2 \quad \text{subject to } L \geq 0, R \geq 0 \tag{11}$$

when the full matrix $Y \in \mathbb{R}^{n \times p}$ is observed.

5 Support Vector Machines

Classical ML Problem

Classification via Support Vector Machines (SVM) is a classical optimization problem in machine learning, tracing origins to the **1960s**.

Given input data (a_j, y_j) with $a_j \in \mathbb{R}^n$ and $y_j \in \{-1, 1\}$, SVM seeks vector $x \in \mathbb{R}^n$ and scalar $\beta \in \mathbb{R}$ such that:

$$a_j^T x - \beta \geq 1 \quad \text{when } y_j = +1 \tag{12}$$

$$a_j^T x - \beta \leq -1 \quad \text{when } y_j = -1 \tag{13}$$

Any pair (x, β) satisfying these conditions defines a **separating hyperplane** in $\mathbb{R}^n$ that separates positive cases $\{a_j | y_j = +1\}$ from negative cases $\{a_j | y_j = -1\}$.

Among all separating hyperplanes, the one minimizing $\|x\|_2$ maximizes the **margin** between classes — the hyperplane whose distance to the nearest point a_j of either class is greatest.

5.1 SVM Optimization Formulation

We formulate the separating hyperplane problem as:

$$H(x, \beta) = \frac{1}{m} \sum_{j=1}^{m} \max(1 - y_j(a_j^T x - \beta), 0) \tag{14}$$

The j-th term is zero if conditions (Equation 12)-(Equation 13) are satisfied, positive otherwise.

A regularized version includes the margin maximization term:

$$H(x, \beta) = \frac{1}{m} \sum_{j=1}^{m} \max(1 - y_j(a_j^T x - \beta), 0) + \frac{1}{2\lambda} \|x\|_2^2 \tag{15}$$

Key Distinction

Unlike previous examples, SVM has: - **Non-smooth loss function** (hinge loss) - **Smooth regularizer** (ℓ_2 penalty)

5.2 Maximum Margin Property

If λ is sufficiently small and separating hyperplanes exist, the pair (x, β) minimizing (Equation 15) is the **maximum-margin separating hyperplane**.

This property ensures **generalizability and robustness**: if observed data (a_j, y_j) is drawn from underlying "clouds" of positive and negative cases, the maximum-margin solution usually separates other empirical samples from the same clouds better than hyperplanes passing close to training points.

5.3 Kernel Methods

When linear separation is insufficient, we transform data vectors a_j by nonlinear mapping ψ and perform SVM classification on vectors $\psi(a_j)$:

$$\psi(a_j)^T x - \beta \geq 1 \quad \text{when } y_j = +1 \tag{16}$$

$$\psi(a_j)^T x - \beta \leq -1 \quad \text{when } y_j = -1 \tag{17}$$

Leading to the analog of (Equation 15):

$$H(x, \beta) = \frac{1}{m} \sum_{j=1}^{m} \max(1 - y_j(\psi(a_j)^T x - \beta), 0) + \frac{1}{2\lambda} \|x\|_2^2 \tag{18}$$

When transformed back to $\mathbb{R}^n$, the surface $\{a | \psi(a)^T x - \beta = 0\}$ is **nonlinear and possibly disconnected**, often providing much more powerful classification than linear hyperplanes.

5.4 Dual Formulation

SVM can be expressed as a convex quadratic program. Taking the dual yields another convex quadratic program in m variables:

$$\begin{aligned} \min_{\alpha \in \mathbb{R}^m} \quad & \frac{1}{2} \alpha^T Q \alpha - \mathbf{1}^T \alpha \\ \text{subject to} \quad & 0 \leq \alpha \leq \frac{1}{\lambda} \mathbf{1}, \quad y^T \alpha = 0 \end{aligned} \tag{19}$$

where: - $Q_{kl} = y_k y_l \psi(a_k)^T \psi(a_l)$ - $y = (y_1, y_2, \ldots, y_m)^T$
- $\mathbf{1} = (1, 1, \ldots, 1)^T$

5.5 The Kernel Trick

Remarkably, problem (Equation 19) can be solved without explicit knowledge of mapping ψ. We need only a **kernel function** $K : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ where $K(a_k, a_l)$ replaces $\psi(a_k)^T\psi(a_l)$ (**BoserEtAl1992?**; **CortesVapnik1995?**).

A popular choice is the **Gaussian kernel**:

$$K(a_k, a_l) := \exp\left(-\frac{1}{2\sigma^2}\|a_k - a_l\|_2^2\right) \tag{20}$$

where $\sigma > 0$ is a parameter.

6 Logistic Regression

Logistic regression is a **softened form** of binary SVM classification. Rather than giving unqualified class predictions, it returns **estimates of odds** of belonging to one class or another.

We seek an "odds function" p parametrized by vector $x \in \mathbb{R}^n$:

$$p(a; x) := (1 + \exp(-a^T x))^{-1} \tag{21}$$

and aim to choose parameter x so that: - $p(a_j; x) \approx 1$ when $y_j = +1$ - $p(a_j; x) \approx 0$ when $y_j = -1$

6.1 Maximum Likelihood Formulation

The optimal value of x minimizes a **negative-log-likelihood function**:

$$L(x) := -\frac{1}{m}\left[\sum_{j:y_j=-1} \log(1 - p(a_j; x)) + \sum_{j:y_j=+1} \log p(a_j; x)\right] \tag{22}$$

Since definition (Equation 21) ensures $p(a; x) \in (0, 1)$ for all a and x, both log terms are negative. When conditions above are satisfied, these terms are only slightly negative, so minimizing (Equation 22) yields near-optimal x.

6.2 Regularized Logistic Regression

Feature selection using ℓ_1 regularization (similar to LASSO):

$$\min_x L(x) + \lambda\|x\|_1 \tag{23}$$

where $\lambda > 0$ is the regularization parameter. This produces solutions where few components of x are nonzero, enabling evaluation of $p(a; x)$ using only selected features of a.

6.3 Multiclass Logistic Regression

For **multiclass (multinomial) logistic regression** with $M > 2$ classes, we need distinct odds functions p_k for each class $k \in \{1, 2, \dots, M\}$:

$$p_k(a; X) := \frac{\exp(a^T x^{[k]})}{\sum_{l=1}^{M} \exp(a^T x^{[l]})}, \quad k = 1, 2, \dots, M \tag{24}$$

where $X := \{x^{[k]} | k = 1, 2, \dots, M\}$.

Properties: - $p_k(a) \in (0, 1)$ for all a and k
- $\sum_{k=1}^{M} p_k(a) = 1$ - Functions (Equation 24) perform **"softmax"** on quantities $\{a^T x^{[l]} | l = 1, 2, \dots, M\}$

6.4 Multiclass Labels and Objective

Labels y_j are vectors in $\mathbb{R}^M$ with elements:

$$y_{jk} = \begin{cases} 1 & \text{when } a_j \text{ belongs to class } k \\ 0 & \text{otherwise} \end{cases} \tag{25}$$

We seek vectors $x^{[k]}$ such that: - $p_k(a_j; X) \approx 1$ when $y_{jk} = 1$
- $p_k(a_j; X) \approx 0$ when $y_{jk} = 0$

This yields the **negative-log-likelihood** minimization:

$$L(X) := -\frac{1}{m} \sum_{j=1}^{m} \left[\sum_{\ell=1}^{M} y_{j\ell} (x^{[\ell]T} a_j) - \log \sum_{\ell=1}^{M} \exp(x^{[\ell]T} a_j) \right] \tag{26}$$

Group-sparse regularization terms can be included to select features in vectors a_j common to each class that distinguish effectively between classes.

7 Deep Learning

Deep neural networks perform the same function as multiclass logistic regression — classifying data vector a into one of M possible classes, often for large M. The major innovation is that mapping ϕ from data vector to prediction is now a **nonlinear function, explicitly parametrized** by structured transformations.

7.1 Neural Network Architecture

A neural network consists of **layers** — each representing a transformation taking input vector and applying nonlinear transformation to produce output vector. Each layer has its own parameters, and the collection of all parameters comprises our optimization variable.

A typical transformation converting vector a_j^{l-1} (output from layer $l-1$) to vector a_j^l (output from layer l) is:

$$a_j^l = \sigma(W^l a_j^{l-1} + g^l) \tag{27}$$

where: - W^l is a matrix of dimension $|a_j^l| \times |a_j^{l-1}|$ - g^l is a vector of length $|a_j^l|$
- σ is a componentwise nonlinear **activation function**

7.2 Common Activation Functions

The activation function σ acts independently on each component:

Sigmoid

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

Rectified Linear Unit (ReLU)

$$\sigma(t) = \max(t, 0)$$

7.3 Deep Learning Loss Function

Consider a linear graph of D levels where layer $l-1$ output becomes layer l input, with $a_j = a_j^0$ and transformation (Equation 27).

Using notation w for all layer-to-layer transformations:

$$w := (W^1, g^1, W^2, g^2, \ldots, W^D, g^D)$$

The **loss function for deep learning** is:

$$L(w) = -\frac{1}{m}\sum_{j=1}^{m}\left[\sum_{\ell=1}^{M} y_{j\ell} a_{j,\ell}^D(w) - \log\sum_{\ell=1}^{M}\exp a_{j,\ell}^D(w)\right] \tag{28}$$

where $a_{j,\ell}^D(w) \in \mathbb{R}$ is the output of the ℓ-th element in layer D corresponding to input vector a_j^0.

Connection to Logistic Regression

Multiclass logistic regression is a special case of deep learning with $D = 1$:

$$a^1_{j,\ell} = W^1_{\ell,\cdot} a^0_j$$

where $W^1_{\ell,\cdot}$ denotes row ℓ of matrix W^1.

7.4 Practical Neural Networks

Neural networks for real applications include many architectural variants:

Advanced Architecture Features

Structural Constraints: - Restricted connectivity between layers (sparse matrices W^l) - Parameter sharing (subsets of W^l elements have same values)
Complex Arrangements: - Outputs from multiple layers - Connections across non-adjacent layers
- Different activation functions σ at different layers - Highly engineered, application-specific designs

7.5 Deep Learning Characteristics

The loss function (Equation 28) shares the finite-sum form with other applications but has distinctive features:

Key Distinguishing Properties

1. **Non-convex** in parameters w
2. **Very large** number of parameters in w

3. **Requires extensive data and computation**
4. **Uses powerful computing clusters** (multicore processors, GPUs, specialized processing units)

8 Emphasis and Framework Properties

Many problems can be formulated within framework (Equation 4), with properties that may differ significantly. They might be **convex or nonconvex**, **smooth or nonsmooth**. But they share important features:

8.1 Shared Mathematical Properties

Universal Characteristics

Real-valued Variables: Formulated as functions of real variables, typically arranged in vector of length n

Continuity: Functions are continuous. When non-smoothness appears, it has structured form exploitable by algorithms

Local Information: Smoothness properties allow algorithms to make good inferences about function behavior based on knowledge at nearby previously-visited points

Finite-sum Structure: Objective often contains summation of many terms, each depending on single data item

Two-term Decomposition: Objective often sums "loss term" (from maximum likelihood) and "regularization term" (imposing structure and generalizability)

8.2 Algorithmic Focus and Analysis

Our treatment emphasizes **algorithms** for solving these problems, with analysis of **convergence properties**. We pay attention to **complexity guarantees** — bounds on computational effort required to obtain solutions of given accuracy.

These bounds usually depend on fundamental properties of: - The objective function
- The data defining it - Problem dimensions and variable counts

Practical Considerations

Trade-off Evaluation: Any problem presents various trade-offs requiring optimizer evaluation of most appropriate tools

Implementation Factors: Must account for: - Time budget for the task - Type of computer for solving - Required solution guarantees

This emphasis contrasts with much optimization literature, where global convergence results typically don't involve complexity bounds (notable exception: interior-point methods (**NesterovNemirovskii1994?**; **Wright1997?**)).

---

9 Enhanced Document Summary

This enhanced version provides:

Improved Mathematical Notation: All equations properly formatted with LaTeX

Enhanced Structure: Better organization with callouts and grids

Corrected Expressions: Fixed broken mathematical expressions from OCR

Professional Formatting: Consistent styling and enhanced readability

Complete References: Proper citation formatting throughout

Visual Enhancement: Strategic use of callouts, grids, and typography

Next Steps: Compare with original version and integrate improvements