Machine learning Algorithms

蔡苗[1]

2019-04-19

[1]Department of Epidemiology and Biostatistics, College for Public Health and Social Justice, Saint Louis University. Email: miao.cai@slu.edu

感谢我的家人的支持。

Acknowledgement

I want to thank my mentor.

目录

表格

插图

Preface

This book works as a notebook to summarize the algorithms used in Bayesian inference and machine learning.

第一章　Introduction

第二章　Discrete optimization

Most of the concepts are explain in Chapter 4 Numerical Computation from the Deep Learning book https://www.deeplearningbook.org/contents/numerical.html. The other useful resource is CS231m Convolutional Neural Networks[1] for Visual Recognition by Stanford University

The **objective function** allows us to measure how "good" any given solution to the problem is. We seek to maximize or minimize the objective function.

Derivative/gradient based methods keep going "uphill" until they are at the top of the h

2.1　Heuristic and metaheuristic methods

> a **heuristic** is a technique designed for solving a problem more quickly when classic methods are too slow, or for finding an approximate solution when classic methods fail to find any exact solution.
>
> Wikipedia

Heuristic methods do not guarantee to find the global optimal solution (best solution)! Instead, they seek to find **a best available solution, given the resource spent looking for it**. A **heuristic method** is **geared towards a specific problem**.

[1] http://cs231n.github.io/

> a **metaheuristic** is a higher-level procedure or heuristic designed to find, generate, or select a heuristic (partial search algorithm) that may provide a sufficiently good solution to an optimization problem, especially with incomplete or imperfect information or limited computation capacity. Metaheuristics sample a set of solutions which is too large to be completely sampled. Metaheuristics may make few assumptions about the optimization problem being solved, and so they may be usable for a variety of problems
>
> – Wikipedia

A metaheuristic method is like a heuristic, but generalizable to a broad class of problems.

1. Genetic Algorithms (Holland – 1975)

- Natural selection / genetics based. Popular method.

2. Simulated Annealing (Kirpatrick – 1983)

- Metallurgy annealing, find lowest energy level!

3. Particle Swarm Optimization (Eberhart Kennedy - 1995)

- Based on insect behavior, swarming towards optimal location (food). Less common in discrete spaces. originally proposed for continuous spaces.

4. Tabu Search (Al-Sultan – 1999)

- Search for best neighborhood solution, then find new neighborhood. Prior neighborhoods are forbidden (tabu)

General meta-heuristics traits

- Evaluate many potential optimal solutions.
- Evaluate the fitness of each solution based on a cost (objective) function.
- Use some concept of stochastic (random) movement to generate new solutions from the parameter space.
- Use some set of rules to determine where to move next in the parameter space.

- Declare convergence once some set of criteria has been met. Perhaps no improvement for X iterations.

2.2 Genetic algorithm and simulated Annealing as examples

Genetic algorithm: need to explore large portions of the parameter space at random. Concept of "neighbor" is vague.

A nice shiny app[2]

An GA example: Since a new treatment for Hep C has become available, where is the optimal place to locate limited new Hep C resources, considering where our patients live?

The problem become intractable with large number of locations and resources: How many combinations of patients and clinics can I calculate the full feature space for to find a maximum?

- Exact Solution is NP-Hard
- Calculations = $n^{\sqrt{k}}$
- I conveniently stopped my analysis at 6 sites with ~5k patients, requiring 1,149,712,053 distance calculations (I have a big server)
- The "k-center" problem

Simulated Annealing:

- The concept of a 'neighbor' is strong.
- Can be sensitive to parameter choice, or algorithm gets stuck in global minima!
- Generally, you should try both to see what works best. Hard to guess up front.

[2] https://toddwschneider.com/posts/traveling-salesman-with-simulated-annealing-r-and-shiny/

2.3 Constrains

Hard constraints

- If this constraint is violated, we have invalid solution.
- Labor laws, number of nurses available, etc

Soft Constraints

- These are nice to meet if possible (included in cost function somehow), but if they are not met the solution is still valid.
- Nurse prefers to only work X night shifts per month.
- Leave requests.

第三章　Continuous optimization

Points to learn in continuous optimization:

- Understand first and second derivatives and the role they play in optimizing continuous functions.
- Understand general steps in continuous optimization
- Contrast 1st order versus 2nd order derivative optimization methods
- Extend these thoughts to the distributed computing context

Things to consider for smart steps:

- Initialization value: where should I start?
- Direction of the gradient: what direction should I we step towards?
- Step size: how big of a step should we take?

3.1 First and second derivative methods

First derivative/gradient

- Instantaneous slope of a point (rate of change of the function)
- If we have multiple input variables (multiple x's), then we need to know the gradient in the direction of each of the x's (partial derivatives). The matrix of partial first derivatives is called **the Jacobian**.

Second derivative

- Tells me the 'curvature' of a function.

- Rate of change of the first derivative.
- If we have multiple input variables (multiple x's), then the matrix of partial second derivatives is called **the Hessian**.

A comparison of first and second order derivative methods

- Second order derivative methods are generally more accurate and converge in fewer steps
- Second order derivative methods are more resource intensive
- Sometimes it is easy and cheap to calculate the Hessian... (generalized linear models with canonical link), so why not?
- Sometimes it is expensive though.
- There is a tradeoff here that is context dependent.

Why not always second derivative

- It's expensive and takes more time / resources / memory.
- The Jacobian matrix only requires $O(n)$ storage.
- The Hessian matrix requires $O(n^2)$ storage.
- The size of the matrix grows exponentially with the size of the input data (specifically the number of columns).
- But... it can be more efficient if we take fewer steps, as long as the dataset isn't too big.

There is a tradeoff between the accuracy of the next step we take, and the amount of resources is take to calculate the next step.

3.2 First Order Derivative methods

3.2.1 Gradient descent

Sourced from wikipedia[1]

- Start somewhere (initial values for X)

[1] https://en.wikipedia.org/wiki/Gradient_descent

- Calculate the gradient at that point
- Take a step in the correct direction based on the gradient
- Step size is a function of the gradient (larger gradient means larger step size)
- Repeat.
- Stop algorithm once it converges (within tolerance) to a single point.
- This is **expensive**! Requires a full pass over all training data at every step...

3.2.2 Stochastic gradient descent

(https://en.wikipedia.org/wiki/Stochastic_gradient_descent)

- Start somewhere (initial values for X)
- Randomly shuffle the data by row.
- For i=1,2,3...n, calculate the gradient **only for the i'th sample** (not the full dataset).
- Take a step in the correct direction based on the gradient
- Step size is a function of the gradient (larger gradient means larger step size)
- Repeat.
- Stop algorithm once it converges (within tolerance) to a single point.
- This is less costly, since you don't need a full pass over the data for every step. But it is less accurate as well...

3.2.3 Coordinate descent

(https://en.wikipedia.org/wiki/Coordinate_descent

- If we have multiple X values, then we optimize them by only considering a change in a single X value at a time. The step size is based on only changing one X.
- This is useful if it is hard to calculate the gradient for all variables (the Jacobian), but easier to only work on one variable at a time.
- This is the optimal solver for regularized GLM's (elastic net regression).
  - Start somewhere (initial values for X)
  - Choose one of the X's (coordinates), change the value (your step).

- Calculate the objective function. Next round, change a different X.
- Repeat.
- Stop algorithm once it converges (within tolerance) to a single point.

Reference to Regularization Paths for Generalized Linear Models via Coordinate Descent[2]

3.3 Second Order Derivative methods

3.3.1 Iteratively reweighted least squares (IRLS)

- This is the most popular in data science frameworks.
- This is efficient and accurate for generalized linear models (logistic regression, Poisson regression etc. . .)
- The Hessian matrix (second derivative) gives us information about the uncertainty of the solution. This is where our confidence intervals and p-values come from!

3.3.2 Newton-Raphson optimization and Fisher Scoring

- More general cases of IRLS. These are identical when applied to GLM's, so you often see the terms interchanged when talking about GLM's. These are not necessarily identical outside of GLM's.
- Second-order methods are sometimes termed 'Newton methods'

3.3.3 Limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS)

Technically first order since it does not evaluate the Hessian.

[2] https://web.stanford.edu/~hastie/Papers/glmnet.pdf

- However, it does approximate the Hessian by storing the prior gradient evaluations!
- So we get some idea of the rate of change of the gradient by looking at the trend of the prior gradients.
- This is termed 'quasi-Newton' since we approximate the Hessian without actually incurring the full cost

Orthant-Wise Limited-memory Quasi-Newton (OWL-QN) is an extension to this that effectively optimizes regularized regression (L1 or elastic net). This is implemented in Apache Spark.

3.3.4 L-BFGS Versus IRLS for GLM's

- Both can be implemented in parallel by calculating chunks of rows at a time.
- Consider m rows and n columns... the IRLS algorithm requires an NxN matrix be generated no matter how small we make M by chunking by row.
- So if we have a large number of columns, IRLS can underperform (take too long / too much memory), even in distributed environments.
- L-BFGS is more efficient for a large number of columns. But, it is generally less accurate (Takes more steps).

3.4 Close thoughts

- Choice of optimization method is important!
- Depending on how large your data is, or how complex your objective function is, you may have to try different optimization methods.
- If the optimization method you choose does not give you estimates about the uncertainty of the solution (I.E. confidence intervals and p-values), you may be able to get that from a direct Hessian calculation once you have declared the optimal solution to be found.

第四章　Machine learning basics

> Machine learning is essentially a form of applied statistics with increased emphasis on the use of computers to statistically estimate complicated functions and a decreased emphasis on proving confidence intervals around those functions
>
> – Deep Learning Book

- Using data, develop a 'learning algorithm' (our model).
- Often the focus is prediction of an outcome, given inputs.
- Finding patterns in the data versus finding generalizable trends in the data.

4.1 What do we need to develop a learning algorithm?

- Data
- Model
- Cost function
- Optimization function

4.2 Classification, regression, and clustering

1. Classification

- Predicting class membership (or probabilities) among distinct classes.
- Death (Yes / No)
- Risk Strata (Low / Medium / High)

2. Regression

- Predicting a continuous summary statistic (like the mean)
- Hospital cost (Mean, median, 90th percentile)

3. Clustering

- Identifying clusters in our data.
- Project data into smaller dimensionality.
- Clustering can be discrete or continuous.

Central challenge to ML: **generalization**.

The algorithm must perform well on new data it has never seen before

- Next years data
- New healthcare system
- New patients

4.3 Under/overfitting and out of sample data

Given the data we have, how good is our model?

- This is really just optimization.
- *Training error* is how well we fit the training data.
- Increased performance here sometimes decreases performance outside of our sample of data (*overfitting*)

In ML, we target generalization error

- *Generalization error* is how well our algorithm fits data outside our sample.
- But we don't have any data outside our sample...
- Can we pretend we do?

4.4 Validation approaches

- If the new data does not come from the same data-generating distribution as the observed data, full stop.
- If we assume the new data comes from the same data-generating distribution, then we can implement validation approaches.
  - Create multiple random samples from the data we have
  - Call one 'training data' and one 'Validation' data.
  - Actually we usually split into training / validation / testing (3 splits).
- Optimization goal:
  - Minimize training error (high error = underfitting)
  - Minimize gap between training and testing error (big gap = overfitting)

Creative Validation Approaches:

- Splitting by clusters.
  - Split by year, region, etc
- Cross validation

Balancing under and overfitting:

- We can balance under and overfitting by making our model more/less complex. The deep learning book calls this **model capacity**
- Increasing model capacity generally allows the model to fit more nuanced relationships.
  - In linear modeling – add more inputs, consider non-linear terms (polynomials), consider interactions...
- What is the downside of increased model capacity?

Model parsimony

> Among competing hypotheses that explain known observations equally well, choose the simplest one.
>
> - Occam's razor (c. 1287-1347)

4.5 Regularization

- Hard code preferences into the model.
  - I prefer Beta's close to or equal to zero (parsimony)
  - However, if I find enough support for a relationship, it can stay.
  - How to I hard code that into my model?
  - What is an example you have learned of this in ML?

  Regularization is any modification we make to a learning algorithm that is intended to reduce its **generalization error not it's training error**.'

4.6 Hyperparameter tuning

- Hyperparameters are 'knobs' we can use to tune an ML algorithm
- We do not learn these from the training data, because...
  - It is too hard/impossible to optimize them directly OR
  - Their intent is to decrease generalization error (not training error), so it is not appropriate to learn them from the training data. Why is this true?
- We often have multiple hyperparameters, and wish to tune across all of them. This is referred to as the 'hyperparameter grid'.

4.7 Binary classification models in scikit-learn

- Logistic regression
- Elastic net logistic regression (regularized)
- Support Vector Machine
- General SGD estimation (sklearn), it can specify different loss functions
- Nearest neighbor majority vote (non-parametric)
- Random forests
  - Fully grown trees (not weak learners). Low bias, high variance per tree

  - Grow many trees (maybe in parallel?) to reduce variance.
- Gradient boosting machine
  - Forest of weak learner trees (high bias, low variance)
  - Correct bias each new sequence.
  - Sequential method!

第五章　MCMC and variational inference

5.1　MCMC

5.2　Variational inference

https://ermongroup.github.io/cs228-notes/inference/variational/

Variational Inference: A Review for Statisticians[1]

[1] https://arxiv.org/pdf/1601.00670.pdf

第六章　Introduction