CS777 Module 3: Data Modeling and Optimization Problems

Topics:

Data Modeling and Optimization Problems

- Introduction to modeling: numerical vs. probabilistic vs. Bayesian
- Introduction to Optimization Problems
- Batch and stochastic Gradient Descent
- Newton's Method
- Expectation Maximization,
- Markov Chain Monte Carlo (MCMC)

3.1. Module Overview

In this module, we step back from the Big Data processing topic and learn the basics of data modeling in data science. We learn how we learn model parameters using different approaches including optimization based and probabilistic methods. Regarding optimization methods, we learn how we use an approach named gradient descent to find optimal model parameters. Furthermore, we learn how we can find model parameters using probabilistic methods like maximum likelihood estimate and Baysian approaches.

3.2. Data Models

We can find many different definitions for data model in data science, like the following examples.

Traditional statistical definition:

"*A set of assumptions regarding the stochastic process that generated the data*"

In this definition, we assume that a stochastic process generated the data. We would like to learn from data and find out what the stochastic process was which generated the data. A stochastic or random process is a mathematical object usually defined as a family of random variables, and it changes randomly.

Examples of more modern definition:

- "*An algorithm that can be used to generate an artifact explaining the data*"
- "*A mathematical object that helps us to understand better past and present, and be able to use it for predicting the future*"

What are the differences between the above definitions? Each of them have different focuses to define the data model, for example explaining artifacts, understanding the data better to use for prediction.

3.2.1. Why Do We Need Models?

Real-world data sets are enormous, complex, difficult to understand. A model is mostly very compact, simple, and comprehensible so it can help us understand the relations inside data and be able to build inferences or predictions. By using a model we want to make some assumptions and do problem simplification. We build models to make predictions of future events and to understand the data patterns and relations.

3.2.2. Statistical Modeling

Many data models rely on the idea of probability. The extent to which an event is likely to occur, measured by the ratio of the favorable cases to the whole number of cases possible. When the number of events are very large and goes to infinitely, then the probability tends to zero.

Some Examples:

- The chance that someone jump exactly 4 feet. It is clear that we cannot measure if the jump was exactly 4 feet.
- The chance that a class ends at exactly 12:00 am. It is clear that we cannot calculate if it was exactly at 12:00 am.
- The chance it takes 5 hours to complete an assignment. We do not if it was exactly 5 hours or not.

3.3. Probability Density

Probability density is around the above problem. It measures the relative likelihood of an event – not based on the absolute probability values.

Example:

The probability of that an assignment takes exactly 5 hours to complete is nonsensical. However, the likelihood that the assignment takes 5 hours is 5 times than it takes 1 hour makes more sense and so that we can calculate it.

Probability Density Function (PDF)

A Probability Density Function (PDF) is a function that computes the relative probability of an event.

For example Normal Distribution (Gaussian Distribution):

$$f_{Normal}(x|\mu,\sigma) = \sigma^{-1}(2\pi)^{-\frac{1}{2}}e^{-\frac{1}{2}(x-e)^2\sigma^{-2}}$$

- A PDF can calculate the probability of a range of events.
- $\int_a^b f(x)dx$ is the probability we see a value in range a to b (assume that a and b are continuous).

3.4. Choosing a Data Model

We can say that all models are wrong but they are useful to describe the reality.

Remember that a data model is compact, simple and comprehensible so that it helps us to understand relationships in the data and inference new knowledge from it. We choose data models to reduce, simplify, and comprehend the data.

Hopefully, without incurring inaccuracy!!

As a data scientist, we have the following four main tasks:

1. **Choosing the model** – Choose main concepts, family, complexity, and hyperparameters.
2. **Learning the model** – Fit the model to data by learning the hyperparameters from data.
3. **Validating the model** – Check if the model match the requirements.
4. **Applying the model** – Actual usage of the data model to understand and explain the data, or predict data (Future data or past data).

3.4.1. Example Model

Example 1: Predicting Grade In Class

Assume that a student has completed 5 out of 10 assignments.

To do: We want to predict his/her grade in the class. Consider that the class has 10 assignments.

First Step, **choose a model**:

- For example, we can assume that all of his grades are sampled from a normal distribution $X_i \sim Normal(\mu, \sigma)$ (Why normal?).
- The index i is the identity of the assignment (i_{th} assignment).
- Note: X_i is a random variable controlling grade.
- $f_{X_i}(x)$ gives relative likelihood X_i takes value x, or probability if X_i is discrete.
- So that $f_{X_i}(x) = f_{Normal}(x|\mu, \sigma)$.

Let's assume that a student got the following scores of in the first 5 assignments: {89, 92, 78, 94, 88}

- We estimate mean $\mu = 88.2$, and variance $\sigma^2 = 30.56$.
- Thus $\sum_{i=6\ldots10} X_i \sim Normal\,(88.2 \times 5, (30.56 \times 5)^{0.5})$
- 95% confidence on sum: $882 \pm 2 \times 12.36$
- Or, 95% confidence on average: 88.2 ± 2.47

Example 2: Assignment Turn In

In this example, we assume that 5 out of 10 students have completed the assignment one. Students in this class have one week time to turn in the assignment one (Turn in deadline is 168 hours (one week) to complete the assignment).

To do: We want to predict how many have completed by 1 hour before due date. How should we model this?

We have a probability and two possible outcomes (Turned in by deadline or Not turned in by deadline). This looks like **Binomial distribution** which is about the distribution of two outcomes. Binomial distribution is a discrete distribution.

Properties of Binomial Distribution

- It has 2 parameters: 1) n = number of independent experiments, and 2) p = probability of success.
- The probability of getting exactly k successes in n trials is given by the Probability Mass Function.

  Probability Mass Function (PMF) = $\binom{n}{k} p^k (1-p)^{n-k}$

- Mean: np
- Variance: $np(1-p)$
- It is a good distribution for modeling Yes/No choices, n times.
- It is important that all of the trials must be independent.
- Degenerative form is the Bernoulli distribution, when $n = 1$.

We want to predict how many have completed by 1 hour before due date.

- X_i: number of hours after assignment student i turns in
- Assume $X_i \sim Exponential(\lambda)$
- Exponential: $f_{Exp}(x|\lambda) = \lambda e^{-\lambda x}$
- Memoryless property!
- It means if we waited unites so far …
- $f_{Exp}(x|\lambda, x \geq t) = f_{Exp}(x - t|\lambda)$

Times so far at time tick 100 are : $\{18, 22, 45, 49, 86\}$

- We know mean of exponential is λ^{-1}.
- In our data case mean is $41 = \lambda^{-1}$, so $\lambda \approx 0.0227$
- Look up the Cumulative Distribution Function (CDF): $1 - e^{-\lambda x}$ is 0.878 at $167 - 100$.
- So prob of each remaining person turning in by deadline is 0.781.
- What about number of people?

Five people, each with 0.781 chance of turning in at deadline -1.

How to model this?

- $N \sim Binomial(0.878, 5)$
- N is the number turning in.
- $Pr(N = 5) = 0.291$ = prob all 10 turn in
- $Pr(N = 4) = 0.698$ = prob 9+ turn in
- $Pr(N = 3) = 0.926$ = prob 8+ turn in
- $Pr(N < 3) = 0.074$ = prob < 8 turn in

3.5. Data Modeling

In data modeling, we have three main big tasks:

1. Choosing the model
2. Learning the model
3. Validating the model

3.5.1. Models Are Parameterized

You saw that we select a model and try to learn the parameters.

For Normal Distribution we have two parameters: μ, σ.

$$f_{Normal}(x|\mu,\sigma) = \sigma^{-1}(2\pi)^{-\frac{1}{2}}\, e^{-\frac{1}{2}(x-e)^2\sigma^{-2}}$$

For Exponential Distribution we have λ

$$f_{Exp^{(x|\lambda)}} = \lambda e^{-\lambda x}$$

Key question is how to choose parameters?

- Typically chosen to "**fit**" the model to example data.
- Item that is, to make the model a good explanation for the data.
- This is also called "**Learning**" in Machine Learning.

3.6. Learning a Model

There are many different approaches to Learning a data model from data, including:

- Optimization based (Minimizing a cost function, for example least squares)
- Probabilistic: Maximum Likelihood Estimation (MLE)
- Probabilistic: Bayesian

3.7. Learning a Model: Optimization Based

The goal is to reduce some error metric on example/training data. We do not have any direct probabilistic motivation in this case and we have a cost function that we want to find the minimum of it.

Example 3: Simple Linear Regression. Least Squares Regression

To do: Consider that we observed the following sequence of integer numbers 18, 22, 45, 49, 86. Predict next item?

You can consider this data also as tuples of (1, 18), (2, 22), (3, 45), (4, 49), (5, 86).

- We might want to fit a straight line to the given data.
- Assume to fit a line with the equation $f(t) = m \times t$.

We need to choose m.

- Might choose least-squares fit.
- For example, choose m to min $l(m) = \sum_i (f(t_i) - x_i)^2$
- $l(m)$ often referred to as a "**Loss function**".

Computing Least-Squares Fit:

$$\begin{aligned} l(m) &= \sum_i (f(t_i) - x_i)^2 \\ &= \sum_i (m \times t - x_i)^2 \\ l'(m) &= \sum_i 2(m \times t_i^2 - x_i \times t_i) \\ &= 2m(1 + 4 + 9 + 16 + 25) \\ &\quad - 2(18 + 44 + 135 + 196 + 430) \\ &= 110m - 1646 = 0 \end{aligned}$$

- We know that the cost function that we want to minimize is a "**Convex**" function.
- So just choose unique m where the derivative is zero $l'(m) = 0$.

So loss minimized at m = 14.96; Next value should be 89.8.

3.7.1. Other Loss Functions

View the list of prediction errors $(f(t_i) - x_i)$ as a vector. We can have many loss functions, corresponding to norms, like l_1 norm (mean absolute error) and l_2 norm (mean squared error).

Given a vector of errors $\langle \epsilon_1, \epsilon_2, \cdots, \epsilon_n, \rangle$, l_p norm defined as:

$$\left(\sum_{i=1}^{n} |\epsilon_i| \right)^{\frac{1}{p}}$$

Common loss functions correspond to various norms:

- l_1 corresponds to mean absolute error.
- l_2 corresponds to mean squared error/least squares.
- l_∞ corresponds to minimax.

3.7.2. Optimization Basics

At the center of all model "learning" frameworks is **optimization**!

- It is more explicit in case of a cost function.
- Implicitly in case of Bayesian.

Solving optimization problems is a fundamental question in data science.

3.7.3. Desired Properties for Optimization Framework

To be useful for data science, optimization framework should be:

- **easy to apply** to many types of optimization problems
- **scalable** and easy to implement in map reduce paradigm
- **fast**

3.7.4. Terminology of Convex and Concave Functions

Convex Function: A function $f(x)$ is defined to be convex when

$$\forall x_1, x_2 \in X, \forall t \in [0,1] : f(tx_1 + (1-t)x_2) \leq tf(x_1) + (1-t)f(x_2) \qquad (3.1)$$

Figure 3.1: Convex Function

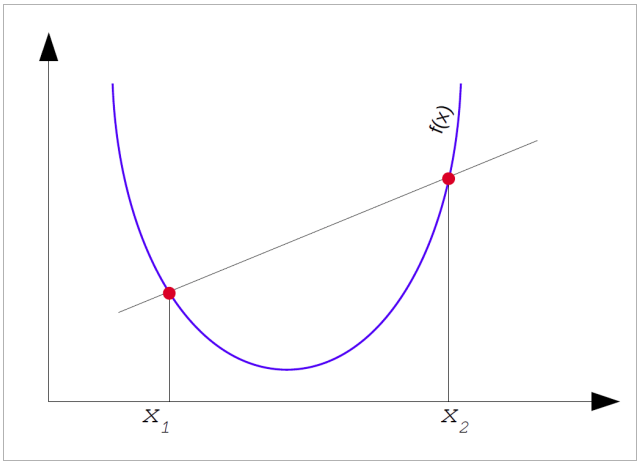
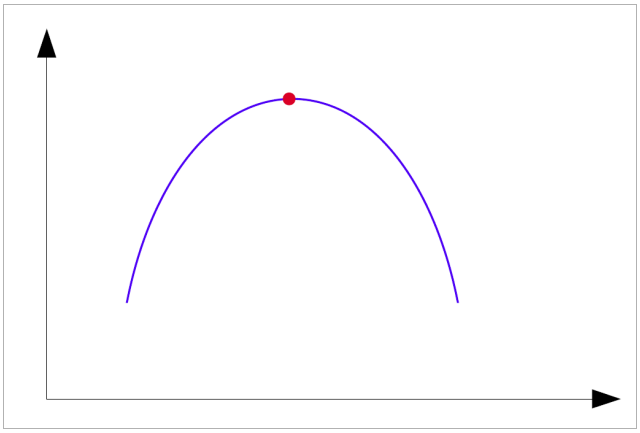


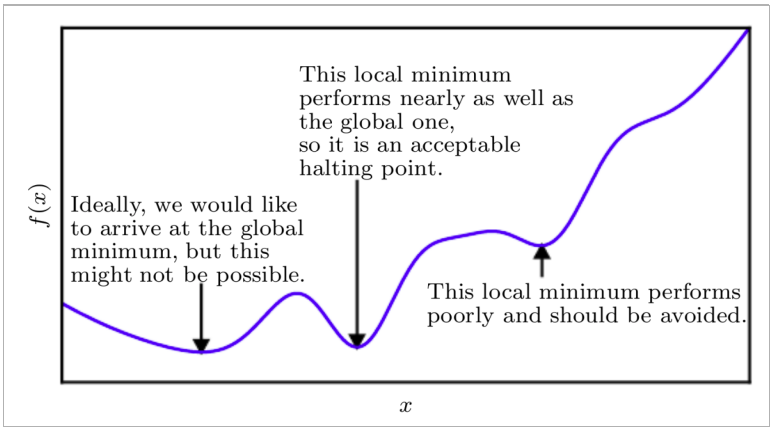
Figure 3.2: Concave Function



For maximum of a Concave function, we calculate minimum of $-f(x)$.

None Convex Functions – Global and Local Minimum

Figure 3.3: Global and Local Minimums



Source: Image from Book: Ian Goodfellow, Yoshua Bengio, Aaron Courville – Deep Learning – The MIT Press (2016)

Example 4: Doing Regression Computation can be Expensive

We have a set of training data like

- Data Matrix of $X \in \mathbb{R}^{n \times d}$, with Labels $Y \in \mathbb{R}^{n \times 1}$
- Linear Regression Model:

$$y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_d x_d + e$$

The same in matrix form: $Y_{n\times 1} = X_{n\times d}\Theta_{d\times 1} + e_{n\times 1}$

- We are looking for a vector of parameters (set of weights) $\Theta \in \mathbb{R}^{d\times 1}$ to minimize

$$MSE = L(\Theta) = \frac{1}{N} e^T e = \frac{1}{N}(y^T y - 2\Theta^T X^T y + \Theta^T X^T X \Theta)$$

Finding exact solution:

- This function is a convex function.
- Take the derivative and set it to zero $\frac{d}{d\Theta} L(\Theta) = -2X^T(Y - X\Theta) = 0$

We set the above to zero and will get $\Theta = (X^T X)^{-1} X^T Y$

In computing exact solution:

- We know that n **number of data rows can be large**.
- If dimension d is small, then $X^T X \in \mathbb{R}^{n\times d}$ and $X^T Y \in \mathbb{R}^{d\times 1}$ are in good size to compute in parallel on a single machine.

This can be one line code in R Program: `solve(t(X) \%*\% X) \%*\% t(X) \%*\% y`.

But what if the **dimension d is large**?

We would need a large size of memory for computing matrix operations (Matrix-Matrix, Matrix-Vector).

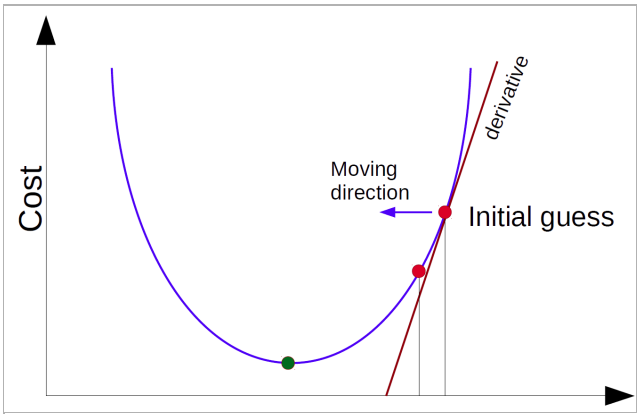
3.7.5. Gradient Descent

Most widely used optimization framework for, at least, **"Big Data" science is gradient descent**.

What is the idea?

- Gradient Descent is an iterative algorithm.
- Goal: choose θ^* to minimize cost function $L(\theta)$.
- Start from an initial guess and try to incrementally improve current solution.
- At iteration step $\theta^{(iter)}$ is the current guess for θ^*.

Figure 3.4: Gradient Descent Algorithm – Initial Guess



3.7.6. What is a Gradient?

Gradient is the multi-dimensional analog to a **derivative**.

- $\nabla L(\theta)$ is a vector-valued function.
- It is a vector whose ith entry is ith partial derivative evaluated at θ_i.

$$\nabla L(\theta) = \begin{bmatrix} \frac{\partial L(\theta)}{\partial \theta_0} \\ \frac{\partial L(\theta)}{\partial \theta_1} \\ \cdot \\ \cdot \\ \cdot \\ \frac{\partial L(\theta)}{\partial \theta_d} \end{bmatrix} \qquad (3.2)$$

- Negative of gradient indicates direction of steepest descent.
- We use the gradient to find out the direction of steepest descent in multidimensional space.

Algorithm 1 Gradient Descent – Basic algorithm

$\theta^{(iter)} \leftarrow$ an initial guess for θ^*
$iter \leftarrow 1$
Repeat
  $\theta^{(iter+1)} \leftarrow \theta^{(iter)} - \lambda \nabla L(\theta^{(iter)})$
  $iter \leftarrow iter + 1$
until (Stop Condition)

- Here λ is the "learning rate" and controls speed of convergence.
- $\nabla L(\theta^{(iter)})$ is the gradient of L evaluated at iteration "iter" with parameter of $\theta^{(iter)}$.
- Stop conditions can be different.

3.7.7. When to Stop?

- Maximum number of iteration is reached $(iter < MaxIteration)$
- Gradient $\nabla L(\theta^{(iter)})$ or parameters are not changing $(||\theta^{(iter+1)} - \theta^{(iter)}|| < precisionValue)$
- Cost is not decreasing $(||L(\theta^{(iter+1)}) - L(\theta^{(iter)})|| < precisionValue)$
- Combination of the above

Mostly we stop it based on number of iterations.

Note: Early stop to avoid "Over-fitting".

Example 5: Learning – Linear Regression Model with Gradient Descent

Returning to linear regression …

- We want to find a line that fits well to points $\langle 118, 122, 145, 149, 186 \rangle$
- At time ticks t in $\langle 1, 2, 3, 4, 5 \rangle$
- Prediction $f(t|c, m) = c + m \times t$
- Loss function $L(c, m) = \sum_i (f(t_i|c, m) - x_i)^2$

Our model would be the following simple line:

- Prediction $f(t|c, m) = c + m \times t$
- The cost function is Mean Squared Error $L(c, m) = \frac{1}{N} \sum_i (f(t_i|c, m) - x_i)^2$.

First we deal with c and m:

$$\frac{\partial F}{\partial c} = \sum_i 2(f(t_i|c,m) - x_i)$$
$$\frac{\partial F}{\partial m} = \sum_i 2t_i(f(t_i|c,m) - x_i)$$

So that

$$\nabla L(c,m) = \langle \sum_i 2(f(t_i|c,m) - x_i), \sum_i 2t_i(f(t_i|c,m) - x_i) \rangle$$

We can divide the loss by 2 to make derivative calculations simpler.

Gradient of the following form is very common:

$$\nabla L(c,m) = \langle \sum_i 2(f(t_i|c,m) - x_i), \sum_i 2t_i(f(t_i|c,m) - x_i) \rangle$$

Big Data Notes: Why is this so good for "Big Data"? MapReduce?

$$y = \theta_0 + \theta_1 x_1 + \ldots + \theta_d x_d$$
$$MSE = L(\Theta) = \frac{1}{2N} \sum_{i=1}^{n} (y^i - (\theta_0 + \theta_1 x_1 + \ldots + \theta_d x_d))^2$$

How to compute the gradient with many dimensions?

Compute partial derivatives:

$$\frac{\partial L}{\partial \theta_1} = \frac{-1}{N} \sum_{i=1}^{n} x_1^{(i)} (y^{(i)} - (\theta_0 + \theta_1 x_1^{(i)} + \cdots + \theta_d x_d^{(i)}))$$
$$\frac{\partial L}{\partial \theta_2} = \frac{-1}{N} \sum_{i=1}^{n} x_2^{(i)} (y^{(i)} - (\theta_0 + \theta_1 x_1^{(i)} + \cdots + \theta_d x_d^{(i)}))$$
$$\ldots$$

Compute components of the gradients (**map operation**) and then sum them up and update weights in the next iteration (reduce operation).

3.7.8. Map Reduce Implementation

Listing 3.1: Gradient Descent Implementation in Map and Reduce Paradigm

```
// initialize parameters
iter= 0
learningRate= 0.01
numIteration= 400
theta = np.zeros(noParameters)

while (iter < maxNumIteration):
  reduceData = myData.map(
          // Calculate the gradients
      )
```

```
            .reduce(
                // update model parameters
                // lambda theta: - learningRate * theta
            )
        iter = iter+1
```

In Spark you can reduceByKey() or better treeAggregate().

3.7.9. Learning Rate

The learning rate in gradient descent is a very important parameter.

Figure 3.5: Gradient Decent with a Very Small Learning Rate

If the learning rate of gradient descent algorithm is set to too small value, gradient descent will do many, many passes through the data to converge.

Figure 3.5 illustrates an example of this case.

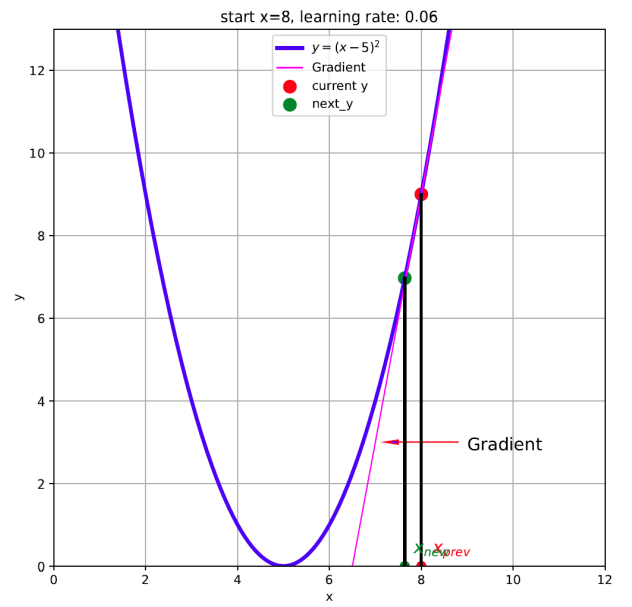
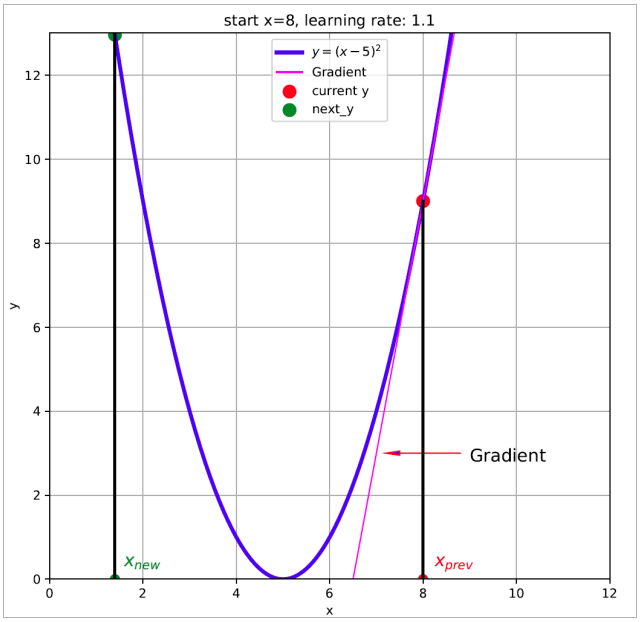


Figure 3.6: Gradient Decent with a Very Large Learning Rate

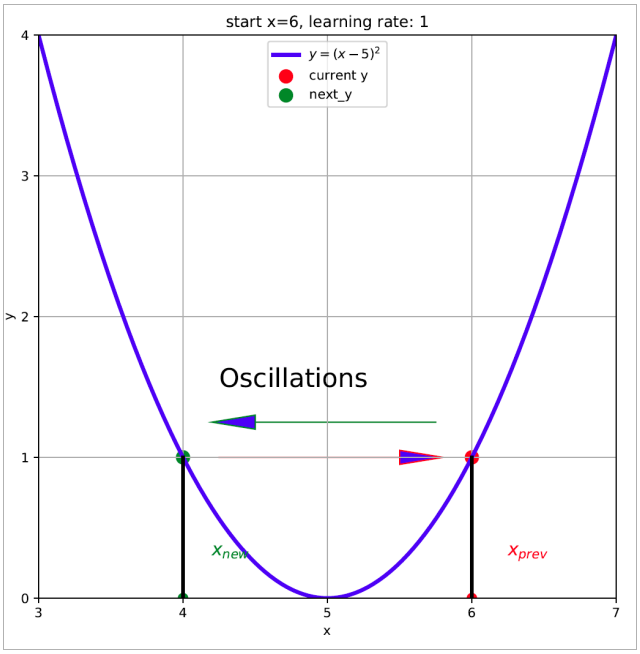
If the learning rate of gradient descent algorithm is set to too large value, then position with change more frequently to the other side and is jumping around.

Figure 3.6 illustrates an example of this case.



If the learning rate is too large, your gradient descent algorithm will oscillate into oblivion and never terminate. Figure 3.7 illustrates an example of this case.

Figure 3.7: Oscillations Case in Gradient Decent



3.7.10. Line Search for Learning Rate

Line search for learning rate is the best option (in terms of results) but **most expensive**:

- Solve another mini-optimization problem
- Select λ so as to minimize $L(\theta^{(iter)})$
- It's a 1-dimensional optimization problem!
- Called a line search.

Algorithm 2 Line Search Algorithm for Finding a Good Learning Rate

$l \leftarrow 0$

$h \leftarrow 999999$

While $(h - l > \epsilon)$ **do**

  $h' \leftarrow l + \frac{1}{c}(h - l)$

  $l' \leftarrow h - \frac{1}{c}(h - l)$

  $goodness_h \leftarrow L(\theta^{(i)} - h' \nabla L(\theta^{(iter)}))$

  $goodness_l \leftarrow L(\theta^{(i)} - l' \nabla L(\theta^{(iter)}))$

  If $(goodness_h < goodness_l)$ **then**

    $l \leftarrow l'$

  else

    $h \leftarrow h'$

  end if

end while

"Golden Section Search" $c = \frac{1}{2}(1 + \sqrt{5}) = 1.618.$

Bold Driver

As we described, the Line Search approach is a very computation intensive and costly approach. One other standard method to choose the learning rate is called "**Bold Driver**".

- Make a very conservative initial guess for λ. At each iteration, compute the cost $L(\Theta^{(iter)})$
- Better than last time? $\lambda \leftarrow 1.05 \times \lambda$
  If cost decreases, increase learning rate.
- Worse than last time? $\lambda \leftarrow 0.5 \times \lambda$
  If cost increases, decrease rate.

This would be then just one evaluation of loss function per iteration!

It is important to monitor the cost function values while running the gradient descent algorithm.

Figure 3.8 illustrates an example of gradient descent algorithm convergence with Bold Driver vs. normal constant learning rate.

Figure 3.8: Gradient Decent Convergence with "Bold Driver" vs. Constant Learning Rates

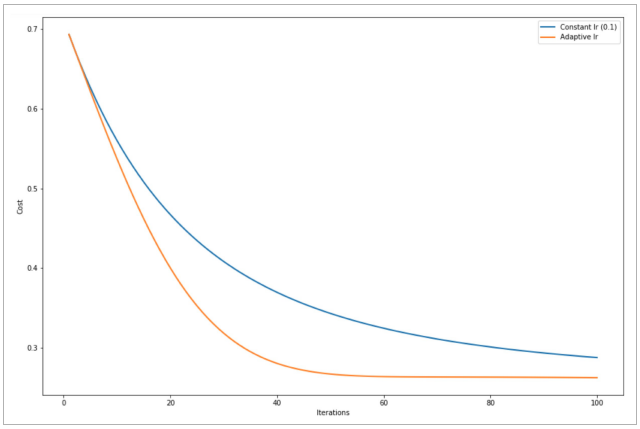


Figure 3.8 is a text classification using Logistic regression (20k Dimensions Term Frequencies). We will learn more about the classification algorithms in the next chapters.

- Horizontal axis is number of iterations.
- Vertical axis shows value of negative log-likelihood.

Figure 3.9 is a text classification using Logistic Regression with different gradient descent learning rates.

Figure 3.9: Gradient Decent Convergence with Different Constant Learning Rates

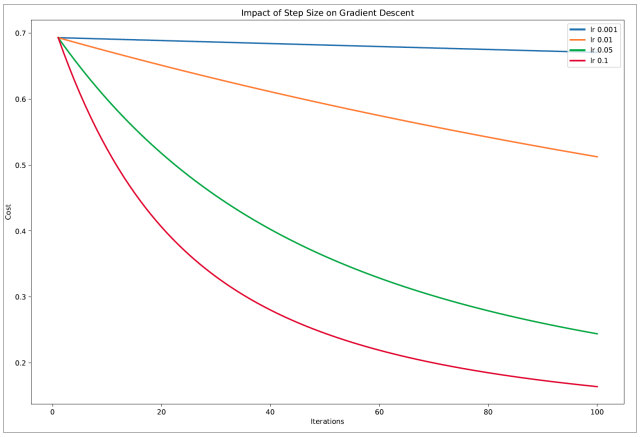


Figure 3.10 is an example Gradient Descent implementation with very large learning rate.

Figure 3.10: Gradient Decent Convergence with Very Large and Very Small Learning Rates

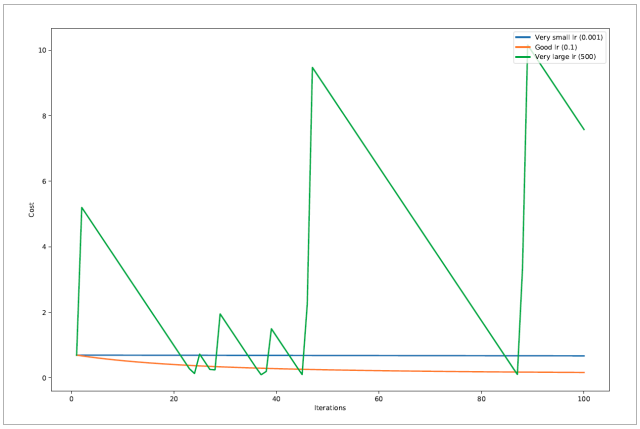


Figure 3.10 includes three different learning rate:

- Very small learning rate of 0.001
- Very Large learning rate
- Good working learning rate

Figure 3.11 is another example of different learning rates.

Figure 3.11: Gradient Decent Convergence with Very Large and Very Small Learning Rates

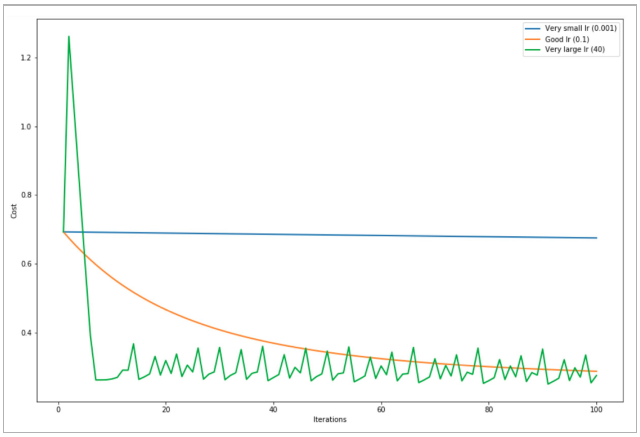


Figure 3.11 includes:

- Very small learning rate of 0.001
- A large learning rate
- A good working learning rate toward

3.7.11. Variations of Gradient Descent

Depending on Size of data that we use in each iteration:

- ***Full Batch Gradient Descent*** (Using the whole data set (size n))

- ***Stochastic Gradient Descent (SGD)*** (Using one sample per iteration (size 1))
- ***Mini Batch Gradient Descent*** (Using a mini batch of data (size $m < n$))

Sometimes people refer to SGD as mini batch.

Stochastic Gradient Descent

Calculate the gradient of a single sample and update parameters in each iteration.

- Pick up samples for update by **sequential read** (**pass over the data** – do not take random sample from a big data set in each iteration – computationally expensive).
- It is noisy and sometimes will do lots of iterations.
- Do not be afraid of **non-convex functions**, SGD can help to get out of local minimums.
- It is a widely applied approach.
- SGD is often faster in producing good results.

Mini-Batch Gradient Descent

Mini-batch gradient descent is something between Stochastic gradient descent and full-batch gradient descent.

- Pick up mini batch samples for update by sequential pass over the data and use it to update.
- Batch gradient descent has better convergence rates than stochastic gradient descent in theory.
- But we may not to pursue to converge fast (presumably results in overfitting).
- It is a widely applied approach.
- Depending on size, can help to out of local minimums.

Implementation Tips:

- Use mini-batch data size of 64, 256, 512, … (power of 2)).
- Make sure that mini-batch data fits in CPU/GPU memory.
- Monitor the costs in each iteration to check if it decreases.
- **Map** to calculate gradients. **Reduce** to update the weights.
- Use vectorization of computations (run bulk operations, e.g., use Numpy inside Spark RDDs or Spark Dataframes).

Some Issues of Gradient Descent

- You can find local min/max and not Global.
- There is no grantee that you can find the global minimum.
- All depends on starting point, step-size, and function type.

The main problem is: **Oscillate into oblivion**.

Gradient Descent with Momentum

A **momentum** is a moving average of our gradients.

$$V^{(iter)} = \beta V^{iter-1} + (1-\beta)\nabla L(W, X, y)$$
$$W^{(iter+1)} = W^{(iter)} - \lambda V^{(iter)}$$

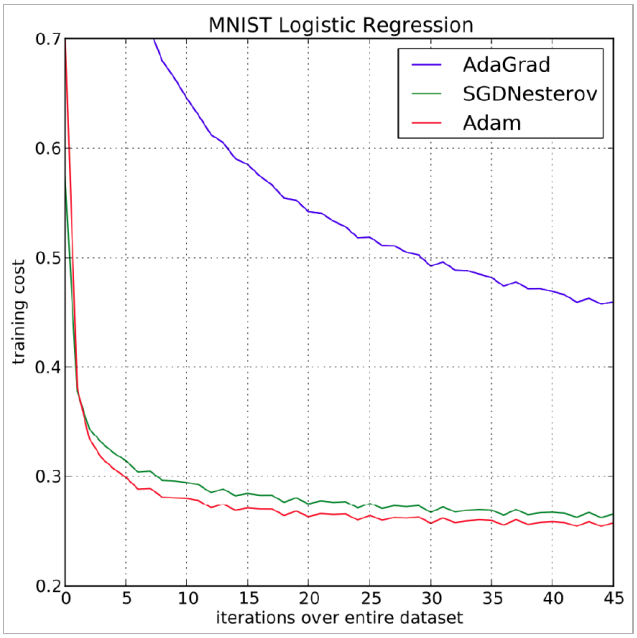
- β is a positive number.
- λ is learning rate.
- $L(W, X, y)$ is cost function.
- W is vector of weights (model parameters).

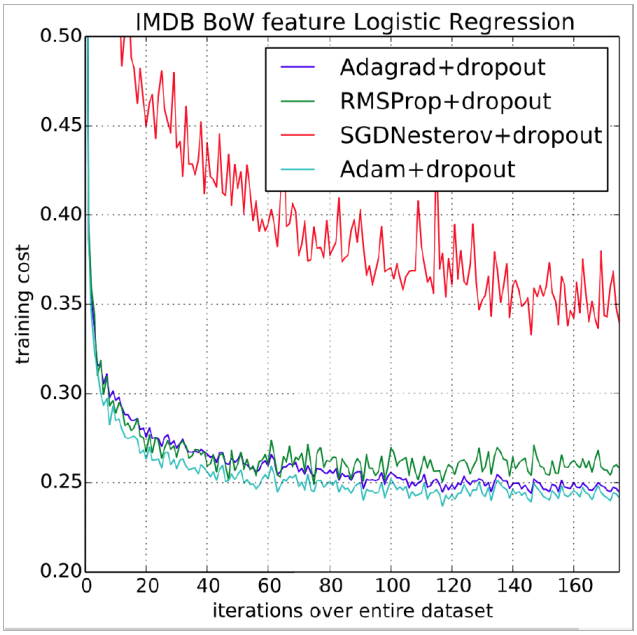
Further Gradient Descent Optimization Variations

- Adaptive gradient algorithm (AdaGrad)
- Root Mean Square Propagation (RMSProp)
- Adaptive Moment Estimation (Adam)

ADAM: A Method for Stochastic Optimization

Figure 3.12: Adap Optimizer Performance





Logistic regression training negative log likelihood on MNIST and IMDB movie reviews with 10,000 bag-of-words (BoW) feature vectors. (Image from Kingma et. al 2017)

Summary

To summarize, gradient descent is a great optimization algorithm and widely applicable in many different machine learning methods. It is a **first-order** iterative optimization algorithm and is easy to use. But its convergence can be slow.

3.7.12. Newton's Method

Gradient descent is great, because of the following reasons:

- GD is simple and easy to use.
- GD is widely applicable.

But convergence can be slow.

Can we optimize it better than GD? Sure!

Optimization - Second-Order Methods

It's a class of iterative optimization methods.

- Use not only first partial derivatives but second derivatives as well (Second-Order means using second derivatives).
- It is an approach that uses Newton's method for finding zero of a function $F()$.
- Finding zeros of a function $F()$ means finding its roots or solutions.

Consequences:

- Improved convergence speeds
- Increased Costs: more complex computations
- Increased Costs: quadratic in number of variables

Newton's Method is a classic second-order method for optimization.

- Comes from Newton's method for **finding zero** of a function $F(\theta)$.
- θ is the model parameter vector of variables.

How Does It Work?

$\theta^{(iter)} \leftarrow$ an initial guess for θ^*

repeat

$$\theta \leftarrow \theta - \frac{F(\theta)}{F'(\theta)}$$

until (Stop Condition)

1. Start from a guess for θ^*
2. Compute $F(\theta)$
3. Compute derivative at $F'(\theta)$
4. Update θ, and repeat until convergence of θ

What About for Optimization? We can use Netwon's Method for Optimization.

- In data science, we don't want to find zeros, we want to find max/min of our Loss function $L()$.
- So just find the root (zero) of the derivative $L'()$.

Our Algorithm Becomes:

$\theta^{(iter)} \leftarrow$ an initial guess for θ^*

repeat

$$\theta \leftarrow \theta - \frac{L'(\theta)}{L''(\theta)}$$

until (Stop Condition)

Multi-Variate Newton's Method

Consider that we have a multi-variate function $F(\theta) : \mathbb{R}^d \rightarrow \mathbb{R}^d$

- $F(\Theta) = \langle F_1(\Theta), F_2(\Theta), \ldots, F_d(\Theta) \rangle$
  Consider $\Theta = \langle \theta_1, \theta_2, \ldots, \theta_d \rangle$
- You can see that index i output of the above function is defined by $F_1()$.

We want to find the $\Theta = \langle \theta_1, \theta_2, \ldots, \theta_d \rangle$ so that:

$$F_1(\theta_1, \theta_2, \ldots, \theta_d) = 0$$
$$F_2(\theta_1, \theta_2, \ldots, \theta_d) = 0$$
$$\ldots$$
$$F_d(\theta_1, \theta_2, \ldots, \theta_d) = 0$$

How can we solve this?

- We would not do the derivation (relies on multi-variate Taylor expansion).
- Define $F(\theta)$ to be the vector:
  $\langle F_1(\theta_1, \theta_2, \ldots, \theta_d), F_2(\theta_1, \theta_2, \ldots, \theta_d), F_d(\theta_1, \theta_2, \ldots, \theta_d) \rangle$
- A ***Jacobian Matrix*** contains all the partial first derivatives of a function
- Define the "**Jacobian**" of $F_1, F_2, \ldots, F_d$ to be:

$$J_F = \begin{pmatrix} \frac{\partial F_1}{\partial \theta_1} & \frac{\partial F_1}{\partial \theta_2} & \frac{\partial F_1}{\partial \theta_3} & \ldots \\ \frac{\partial F_2}{\partial \theta_1} & \frac{\partial F_2}{\partial \theta_2} & \frac{\partial F_2}{\partial \theta_3} & \ldots \\ \frac{\partial F_3}{\partial \theta_1} & \frac{\partial F_3}{\partial \theta_2} & \frac{\partial F_3}{\partial \theta_3} & \ldots \\ \ldots & \ldots & \ldots & \ldots \end{pmatrix}$$

Note: This is a matrix of functions!

So $J_F(\theta)$ is a matrix of scalars.

Example 6: Example of Jacobian Matrix

$$f\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) = \begin{bmatrix} f_1(x, y) \\ f_2(x, y) \end{bmatrix} = \begin{bmatrix} x^3 y \\ x^2 + sin(y) \end{bmatrix}$$

So that the Jacobian Matrix of function f is:

$$J_F = \begin{bmatrix} \frac{\partial f_1}{\partial x}, \frac{\partial f_1}{\partial y} \\ \frac{\partial f_2}{\partial x}, \frac{\partial f_2}{\partial y} \end{bmatrix}$$

$$J_F = \begin{bmatrix} 3x^2 y\,, & x^3 \\ 2x\,, & cos(y) \end{bmatrix}$$

Pros and Cons of Newton's Method

To summarize, gradient descent is a great optimization algorithm and widely applicable in many different machine learning methods. It is a **first-order** iterative optimization algorithm and is easy to use. But its convergence can be slow.

Advantages:	Disadvantages:
• Convergence is quadratic; that is, error decreases quadratically. • Hundreds/thousands of iterations (gradient descent) become tens.	• It is way more complicated than gradient descent! • Quadratic cost of each iteration (cost of gradient descent was linear) Hessian Matrix includes quadratic number of variables. • There is additional cost for inverting the matrix. • The second derivative has to exist.

Newton's method is not used much in practice since in high dimensions. Quasi-Newton methods are used instead.

3.7.13. Quasi-Newton Method

Any methods that replace the exact Jacobian with an approximation is a ***Quasi-Newton Method***. Quasi-Newton Methods use just an estimation of Hessian Matrix.

Some of the most famous Quasi-Newton Methods:

- Limited-memory BFGS (Broyden-Fletcher-Goldfarb-Shanno algorithm)
- Broyden's method

3.8. Learning a Model: Probabilistic - Maximum Likelihood Estimation (MLE)

Often, we have a stochastic process and we want to build a stochastic data model.

Example 7

Consider this Example: observed data 18, 22, 45, 49, 86.

We might think that the data is generated by sampling from an Exponential distribution, $f(x_i|\lambda) = \lambda e^{\lambda x}$. The unknown model parameter is the λ.

How to estimate?

Most common learning approach is to perform Maximum Likelihood Estimation (MLE).

3.8.1. Likelihood

First of all, we need to understand the notion of a "***likelihood function***".

The best way is to illustrate the likelihood function with an example:

- In our case (Example 7), $f(x_i|\lambda) = \lambda e^{\lambda x}$
- So that, $f(x_1, x_2, \ldots x_n|\lambda) = \prod_{n=1} e^{\lambda x_i}$ (iid!)

A "likelihood function" simply turns the parameterization around.

- So $L(\lambda|x_1, x_2, \ldots x_n) = \prod_{n=1} e^{\lambda x_i}$
- Now likelihood function L measures the goodness of the parameter λ in our model.
- This is NOT how likely the data set of $x_1, x_2, \ldots, x_n$ are given the data model.

Given $L(\Theta|D)$ (Θ is set of model parameters, D is data), the MLE $\hat{\Theta}$ for Θ is defined as the value such that

$$\forall \hat{\Theta}', L(\hat{\Theta}'|D) \leq L(\hat{\Theta}|D)$$

This is closely related to least squares because we look to find the maximum of function.

We like MLE because:

- Under many conditions, it is the ***minimum-variance unbiased estimator (MVUE)***. We want to have an estimator for our model parameter with the minimum variance.
- Under many conditions, error is ***asymptotically normal***.

Example 8

We observed $\{18, 22, 45, 49, 86\}$.

- So $L(\lambda|x_1, x_2, \ldots x_n) = \prod_{n=1} e^{-\lambda x_i}$
- Typically, we maximize the log-likelihood (LLH) instead:

$$\sum_{i=1} \log(e^{-\lambda x_i}) = \sum_{i=1} -\lambda x_i + \log(\lambda) \quad (3.3)$$

- Again we find that this function is a convex function:

$$\begin{aligned} L'(\lambda) &= \sum_{i=1} x_i + \lambda^{-1} \\ &= 220 + 5\lambda^{-1} \end{aligned}$$

Set it to zero:

$$\begin{aligned} 0 &= 220 + 5\lambda^{-1} \\ \frac{220}{5} &= \lambda^{-1} \\ \lambda &= \frac{5}{220} \end{aligned}$$

Example 9: A More Complicated MLE

Now, imagine the data $\{18, 22, 45, 49, 86\}$ are assignment completion times by students. We know that only 5 out of 10 finished at time tick 100.

What's a problem with the last model?

- Other 5 student have not completed their assignments yet and have not contribute the data yet!
- How to build a predictive data model for this problem? Each of 5 who have not yet arrived have $x_i \geq 100$
- Cumulative Distribution Function (CDF) of exponential distribution is $1 - e^{\lambda x}$
- So for $i \geq 5$, $Pr[no\ submission] = 1 - (1 - e^{\lambda x})$
- So $L(\lambda | x_1, x_2, \ldots x_n) = \prod_{i=1}^{5} \lambda e^{-\lambda x_i} \times \prod_{i=6}^{10} e^{-\lambda 100}$

$L(\lambda|\cdot) = \prod_{i=1}^{5} \lambda e^{-\lambda x_i} \times \prod_{i=6}^{10} e^{-\lambda 100}$

LLH instead: $L(\lambda|\cdot) = \sum_{i=1}^{5} -\lambda x_i + \log(\lambda) + \sum_{i=6}^{10} -\lambda 100$

Now, minimizing:

$$\begin{aligned} L'(\lambda) &= -\sum_{i=1}^{5} x_i + \frac{1}{\lambda} - \sum_{i=6}^{10} 100 \\ &= \frac{5}{\lambda} - 500 - \sum_{i=1}^{5} x_i \\ &= \frac{5}{\lambda} - 720 \end{aligned}$$

Set it to zero, we have

$$\begin{aligned} 0 &= \frac{5}{\lambda} - 720 \\ 720 &= \frac{5}{\lambda} \\ \lambda &= \frac{5}{720} \end{aligned}$$

3.9. Learning a Model: Bayesian

One complaint regarding MLE approach is: "It assumes zero knowledge about the parameter(s) you are trying to estimate." Do we ever have zero knowledge?

Back to our example about the scores in a class.

- Scores so far: $\{99, 92, 94, 94, 88\}$
- Is mean best estimated as $(99 + 92 + 94 + 94 + 88)/5$?
- What if I'd never had an assignment with avg > 90 in my life?

To a Bayesian:

- "Learning" is all about updating one's prior opinions in response to evidence.

"**Prior opinions**" are formally given in the form of a "prior distribution". For example:

- Pretend I'm really nasty.
- My average assignment score is around 5.
- Highest ever was 70.
- Lowest ever was 3.
- So I choose Normal(50, 5) as the "prior" on the mean assignment score μ.

A Bayesian uses data X to update the prior on the parameter set Θ:

- Resulting distribution—$P(\Theta|X)$ is called the "posterior".

Update is accomplished via "Bayes' Rule"

$$P(\Theta|X) = \frac{P(\Theta)P(X|\Theta)}{P(X)}$$

Can usually drop $P(X)$ as a constant, so we have

$$P(\Theta|X) \propto P(\Theta)P(X|\Theta)$$

Example 10: Bayes' Rule Example

Scores so far: $\{99, 92, 94, 94, 88\}$.

- Mean score $\mu \sim Normal(50, 5)$
- Each score $x_i \sim Normal(\mu, 4)$
- Applying Bayes' rule:

$$P(\mu|data) \propto Normal(\mu|50, 5) \prod_{i=1} Normal(x_i|\mu, 4)$$

Let's do some math here!

$$P(\mu|data)$$

$$P(\mu|data) \propto Normal(\mu|50,5)\prod_i Normal(x_i|\mu,4)$$
$$= 5^{-1}(2\pi)^{-\frac{1}{2}}e^{-\frac{1}{2}(\mu-50)^2 5^{-2}}\prod_i 4^{-1}(2\pi)^{-\frac{1}{2}}e^{-\frac{1}{2}(\mu-x_i)^2 4^{-2}}$$
$$\propto e^{-\frac{1}{2}(\mu-50)^2 5^{-2}}\prod_i e^{-\frac{1}{2}(\mu-x_i)^2 4^{-2}}$$
$$= e^{-\frac{1}{2}\left((\mu-50)^2 5^{-2}+\sum_i(\mu-x_i)^2 4^{-2}\right)}$$
$$= e^{-\frac{1}{2}\left(5^{-2}\mu^2-100\times5^{-2}\mu+2500\times5^{-2}+\sum_i 4^{-2}\mu^2-2\times4^{-2}\mu\times x_i+4^{-2}x_i{}^2\right)}$$

A bit more Math …

$$= e^{-\frac{1}{2}\left(5^{-2}\mu^2-100\times5^{-2}\mu+2500\times5^{-2}+\sum_i 4^{-2}\mu^2-2\times4^{-2}\mu\times x_i+4^{-2}x_i{}^2\right)}$$
$$\propto e^{-\frac{1}{2}\left(5^{-2}\mu^2-4\mu+\sum_i 4^2\mu^2-2\times4^{-2}\mu\times x_i\right)}$$
$$= e^{\left(2+\frac{1}{16}\sum_i x_i\right)\mu-\left(\frac{1}{50}+\frac{5}{32}\right)\mu^2}$$
$$= e^{a\mu^2+b\mu}$$

Where

$$a = -\frac{1}{50}-\frac{5}{32}$$
$$b = 2+\frac{1}{16}\sum_i x_i$$

Now this looks better and quite simple …

We have

$$P(\mu|data) \propto e^{a\mu^2+b\mu}$$

where:

$$a = -\tfrac{1}{50}-\tfrac{5}{32} = -0.17625,\ b = 2+\tfrac{1}{16}\textstyle\sum_i x_i = 31.1875$$

- By definition, this is $\propto Normal(-b/(2a), \sqrt{-1/(2a)})$
- Or, Normal(88.475, 1.7)

3.9.1. Conjugate Priors

That was a lot of work! Easier to use a table of conjugate priors.

What exactly is the ***conjugate priors***?

- When you have $\Theta \sim f(\theta_{prior})$
- And you have $X \sim g(.)$

- And you can prove $P(\Theta|X) = f(\Theta|\theta_{post})$
- That is, the posterior for Θ is the same family as the prior
- Then we say f is a "conjugate prior" for g.

There are lots of conjugate priors that are key tools in Bayesian's toolbox.

Why is it useful?

Usually simple rules for computing θ_{post} from X, θ_{prior} (Google search "Wikipedia conjugate prior" …first result, find row under "Continuous Distributions").

- When $g(.)$ (likelihood) is Normal with known σ
- And $f(\theta_{prior})$ is $Normal(\mu_0, \sigma_0)$
- Then posterior is easy!
- In θ post , we have:

$$\begin{aligned}\mu &= \left(\frac{\mu_0}{\sigma_0^2} + \frac{\sum x_i}{\sigma^2}\right) / \left(\frac{1}{\sigma_0^2} + \frac{n}{\sigma^2}\right) \\ &= \left(\frac{50}{25} + \frac{467}{16}\right) / \left(\frac{1}{25} + \frac{5}{16}\right) \\ \sigma^2 &= \left(\frac{1}{\sigma_0^2} + \frac{n}{\sigma^2}\right)^{-1} \\ &= \left(\frac{1}{25} + \frac{5}{16}\right)^{-1}\end{aligned}$$

Gives the same result, much less work!

3.10. What We Have Learned

In this module, we have learned about:

- The basic concepts of data modeling
- Three main methods of data models, optimization based, MLE, and Bayesian
- We have learned more in depth about Gradient Descent and Newtons Method.
- We have learned about Likelihood and MLE.

3.10.1. Looking Ahead

In the next module, we will learn about:

- Supervised Learning Methods
- General Linear Model

- Classification with Logistic Regression
- Classification with Support Vector Machine
- Overfitting and Regularization

3.11. Further Reading References

- Diederik P. Kingma, Jimmy Ba (2015). *Adam: A Method for Stochastic Optimization*. In proceeding of ICLR 2015: San Diego, CA, USA.
- John Duchi, Elad Hazan, and Yoram Singer (2011). *Adaptive Subgradient Methods for Online Learning and Stochastic Optimization*. Journal of Machine Learning Research, 12:2121–2159, 2011.
- Timothy Dozat (2016). Incorporating Nesterov Momentum into Adam. ICLR Workshop, (1):2013–2016, 2016.
- Bottou, Léon (1998). *Online Algorithms and Stochastic Approximations*. Online Learning and Neural Networks. Cambridge University Press. ISBN 978-0-521-65263-6.
- Bottou, Léon (2010). *Large-scale machine learning with SGD*. Proceedings of COMPSTAT'2010. Physica-Verlag HD, 2010. 177-186.
- Bottou, Léon (2012). *SGD tricks. Neural Networks: Tricks of the Trade.* Springer Berlin Heidelberg, 2012. 421-436.