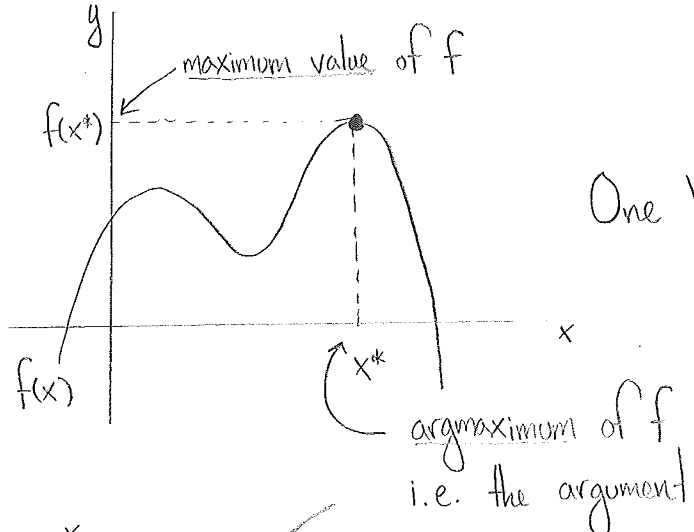
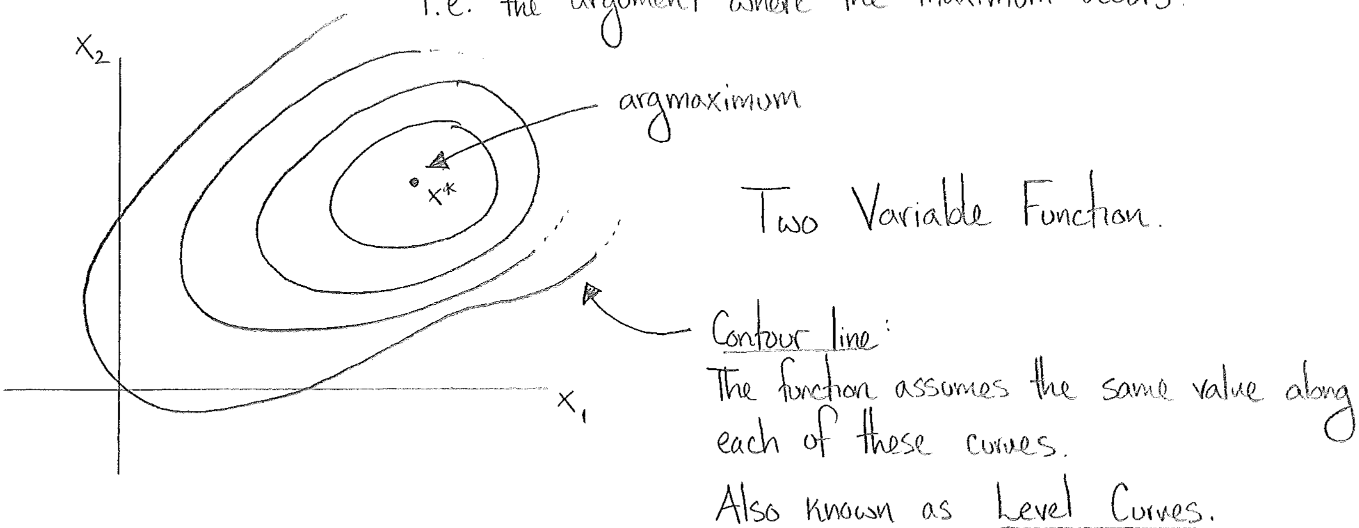
Mathematical/Numeric Optimization

<u>Optimization</u> is the study of methods for finding the maximal or minimal value of a function.



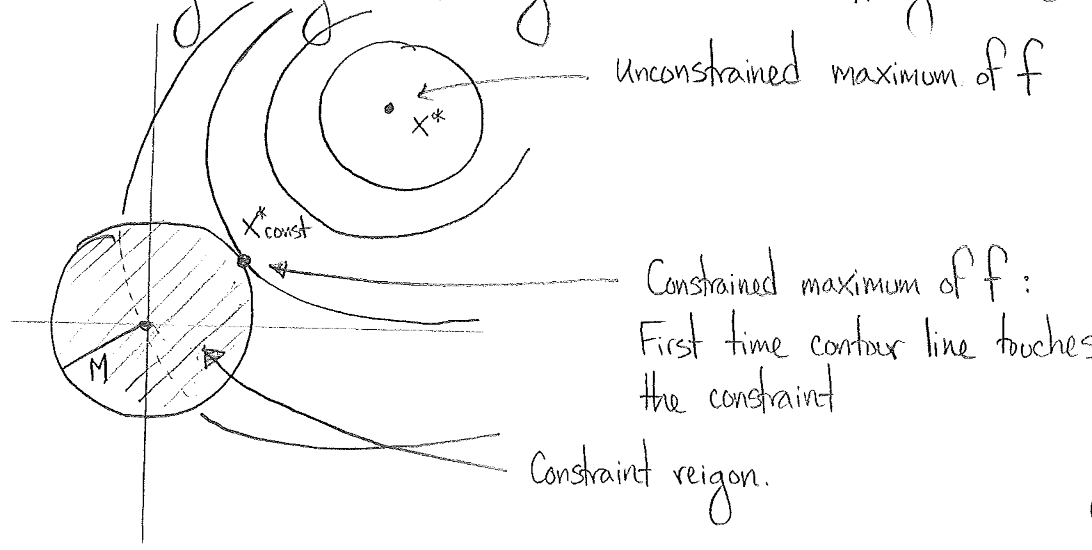
One Variable Function.



Two Variable Function.

Sometimes optimization problems are <u>constrained</u>: we are intersected in the maximum/minimum value of f but only among those arguments that satisfy some constraints.

maximize $f(x)$
subject to constraints
$x_1^2 + x_2^2 \leq M^2$



Examples

① Linear Regression:

Looking for the optimal values of those parameters.

$$\underset{\vec{\beta}}{\text{minimize}} \left\{ \sum_{i=1}^{n} \left(y_i - \sum_{j=0}^{m} \beta_j x_{ij} \right)^2 \right\}$$

sum over all the predictions

sum over all the data

② Logistic Regression

Each of these is either 0 or 1

$$\underset{\vec{\beta}}{\text{minimize}} \left\{ -\sum_{i=1}^{n} y_i \log\left(\underbrace{\frac{1}{1 + e^{-\sum_j \beta_j x_{ij}}}}_{\text{These are between 0 or 1, so probabilities.}} \right) + (1-y_i) \log\left(1 - \frac{1}{1 + e^{-\sum_j \beta_j x_{ij}}} \right) \right\}$$

This is often written

$$\underset{\vec{\beta}}{\text{minimize}} \left\{ -\sum_{i=1}^{n} y_i \log(p_i) + (1-y_i) \log(1-p_i) \right\}$$

$$\text{where } p_i = \frac{1}{1 + e^{-\sum_j \beta_j x_{ij}}} \longleftarrow \text{Logistic Function}$$

③ Regularized Linear Regression

Regurization strength hyperparameter

$$\underset{\vec{\beta}}{\text{minimize}} \left\{ \sum_{i=1}^{n} \left(y_i - \sum_{j=0}^{m} \beta_j x_{ij} \right)^2 + \lambda \underbrace{\sum_{j=1}^{m} \beta_j^2}_{} \right\}$$

Regularization Penalty term : Incentivizes smaller coefficients.

There is an equivelent <u>constrained</u> form of regularized linear regression

$$\underset{\vec{\beta}}{\text{minimize}} \left\{ \sum_{i=1}^{n} \left(y_i - \sum_{j=0}^{m} \beta_j x_{ij} \right)^2 \right\}$$

subject to the constraints :

$$\sum_{j=1}^{m} \beta_j^2 \leq M$$

Regularization strength hyperparameter.

This is representative of a general pattern :

"Every constrained optimization problem has an equivelent unconstrained optimization problem"

The process of moving from the constrained problem to the unconstrained problem is called <u>Lagrange Multipliers</u>.

The Gradient

We want to develop methods for solving optimization problems, and all methods start with a common concept, the Gradient.

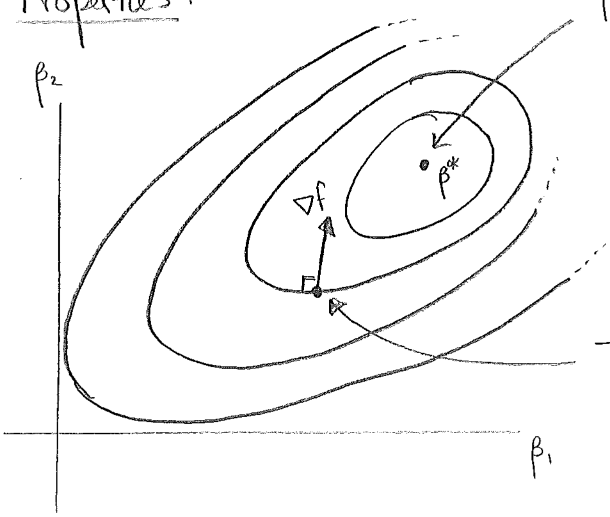
Definition

Suppose that $f(\beta_0, \beta_1, \beta_2, \cdots, \beta_m)$ is a function of multiple arguments. The gradient of f is:

$$\nabla f(\beta_0, \beta_1, \cdots, \beta_m) = \underbrace{\left(\frac{\partial f}{\partial \beta_0}, \frac{\partial f}{\partial \beta_1}, \cdots, \frac{\partial f}{\partial \beta_m}\right)}_{\text{Vector of partial derivatives.}}$$

Note: The gradient is a <u>vector</u> that varies depending on the point $(\beta_0, \beta_1, \cdots, \beta_m)$, sometimes called a vector field.

Properties:



This is a maximum / minimum, the gradient is zero here.

This is <u>not</u> a maximum / minimum, the gradient is perpendicular / orthogonal to the contour lines.

- At a maximum / minimum, the gradient is the <u>zero</u> vector.
- At a non-maximum or minimum, the gradient is perpendicular to the contour line. This is the <u>direction the function increases most quickly</u>.

Using The Gradient To Find Maxima / Minima

Solving Explicitly:

Very rarely, the following algorithm works:

1. Write down the function you want to find the max/min of.
2. Use pen/paper to work out the gradient (sometimes fails)
3. Set the gradient equal to zero, solve the resulting equations (often fails).

Example:

1. $f(\beta_1, \beta_2) = \beta_1^2 - 2\beta_1\beta_2 + 3\beta_2^2 + 4\beta_1$

$$\left.\begin{aligned} \frac{df}{d\beta_1} &= 2\beta_1 - 2\beta_2 + 4 \\ \frac{df}{d\beta_2} &= -2\beta_1 + 6\beta_2 \end{aligned}\right\} \text{These partial derivatives are the components of the gradient.}$$

$$\nabla f(\beta_1, \beta_2) = \left(2\beta_1 - 2\beta_2 + 4,\ -2\beta_1 + 6\beta_2\right)$$

The equations we need to solve are $\nabla f = 0$, which is this system of equations:

$$\begin{cases} 2\beta_1 - 2\beta_2 = -4 \\ -2\beta_1 + 6\beta_2 = 0 \end{cases}$$

Which can be solved with scipy and numpy

$$\text{scipy.linalg.solve}\left(\begin{bmatrix} 2 & -2 \\ -2 & 6 \end{bmatrix}, \begin{bmatrix} -4 \\ 0 \end{bmatrix}\right)$$

The solution is $\begin{pmatrix} -3 \\ -1 \end{pmatrix}$, which is where the minimum occurs.

Example : Linear Regression

If we apply the same procedure to the linear regression problem, we get the result :

$$\hat{\beta} = \text{solve}(X^t X, X^t y)$$

Which is often written

$$\hat{\beta} = (X^t X)^{-1} X^t y$$

Logistic regression <u>can</u> <u>not</u> be solved this way.

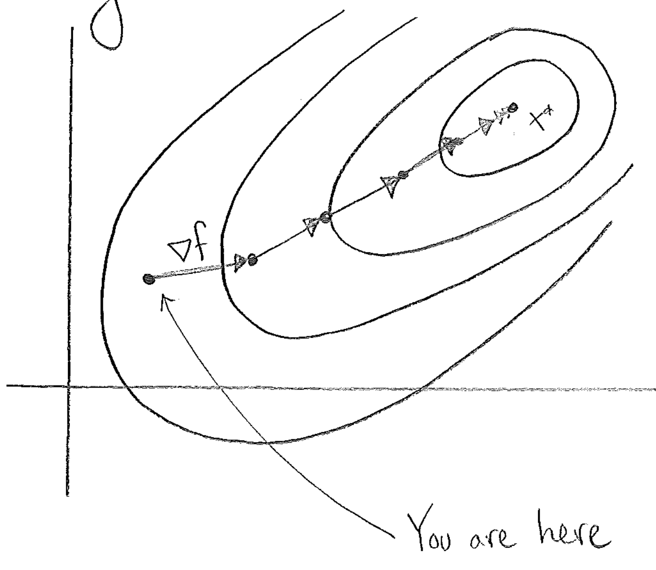
Gradient Descent

If explicit solutions often fail, how can we still achieve results, i.e. how are the solutions to logistic regression achieved?

The idea comes from :

"The gradient points in the direction that the function increases most quickly"

<u>Idea</u>: Repeatedly walk in the direction of greatest increase / <u>decrease</u>.



Algorithm

Pre-Algorithm:

- Write down the function to minimize/maximize.
- Calculate the gradient with pen and paper.

Algorithm:

- Choose an initial point x_0.
- Plug x_0 into the gradient to get $\nabla f(x_0)$.
- Move in the direction of the gradient to a knew point:

$$x_{i+1} = x_i + \alpha \nabla f(x_i) \longleftarrow \text{maximization.}$$

$$x_{i+1} = x_i - \alpha \nabla f(x_i) \longleftarrow \text{minimization.}$$

- Repeat.

The learning rate, a positive number, less than one. Guarentees that you dont "overshoot".

When to stop:

Option #1: When $|\nabla f| < \varepsilon$. ← Called the convergence threashold

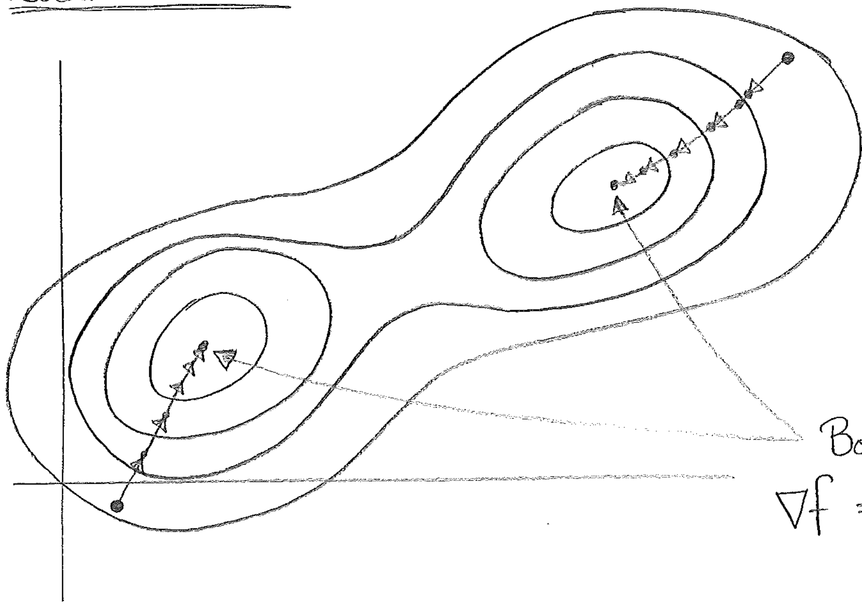
I.e. when the length of the gradient is small.

Option #2: When $\dfrac{|f(x_{i+1}) - f(x_i)|}{|f(x_i)|} < \varepsilon$.

I.e. when the percentage decrease is small.

Trouble Shooting:

Local Extrema.



Both of these are points where $\nabla f = 0$.

When the function has multiple *local* maxima/minima, gradient descent will converge to one of them.

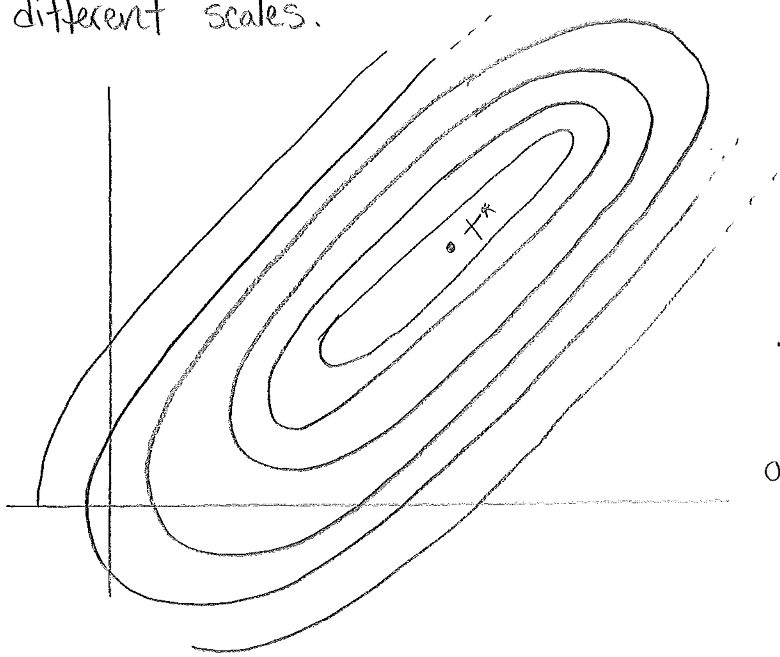
In this case you may not end up at the *global* maximum/minimum.

Solutions:

- Many common problems (linear + logistic regression) have <u>exactly</u> <u>one</u> local maximum/minimum, so this issue does not arise.
- Add an inertia term to your desent to "power through" some local maxima/minima.
- In general, there is no silver bullet, you must add some randomization to your algorithm.
  - Start at many random points.
  - Add random noise to the gradient.

Feature Scaling

In regression problems, sometimes different fealures are measred on very different scales.



In this situation the level curves of the loss function becomes very <u>long</u> and <u>thin</u>.

It becomes <u>very</u> <u>easy</u> to overshoot in the thin direction.

<u>Solution</u>: Standardize the features.

$$\text{Replace } \vec{x}_i \text{ with } \frac{\vec{x}_i - \text{mean}(\vec{x}_i)}{\text{standard-dev}(\vec{x}_i)}.$$

All the standardized features have mean zero and variance one.

The level curves of the loss function with the <u>new</u> features are much more round.

Stochastic Gradient Descent

Many important cost functions in machine learning are sums of a simpler function.

Linear Regression:

$$L(\vec{\beta}) = \sum_{i=1}^{n} \underbrace{\left(y_i - \sum_{j=0}^{m} \beta_j x_{ij}\right)^2}_{\text{One term for each datapoint}} \qquad \ell(\vec{\beta}; \vec{x}, y) = \left(y - \sum_{j=0}^{m} \beta_j x_j\right)^2$$

Logistic Regression

$$L(\vec{\beta}) = -\sum_{i=1}^{n} y_i \log(p_i) + (1-y_i)\log(1-p_i)$$

$$\ell(\beta; \vec{x}, y) = -y\log(p) - (1-y)\log(1-p)$$

When using gradient descent to minimize these cost functions, we use the gradient of the entire sum

$$\frac{\partial L_{\text{linear}}}{\partial \beta_j} = 2\sum_{i=1}^{n}\left(y_i - \sum_{j=0}^{m}\beta_j x_{ij}\right)x_{ij}$$

This uses the standard relationship $\frac{d\,\text{logistic}(x)}{dx} = \text{logistic}(x)(1-\text{logistic}(x))$ for the logistic function.

$$\frac{\partial L_{\text{logistic}}}{\partial \beta_j} = -\sum_{i=1}^{n} y_i(1-p_i)x_{ij} + (1-y_i)p_i x_{ij}$$

$$= -\sum_{i=1}^{n}(y_i - p_i)x_{ij}$$

To calculate each of these gradients, we need to use all the data.

Sometimes we don't have access to all the data!

- Huge datasets do not fit in RAM.
- Our data comes in a stream, and we only have access to data temporarily
- Our data comes in a stream, and we want to improve our model in real time (this is called <u>online learning</u>)

In <u>Stochastic Gradient Descent</u> we update our model using one data point at a time:

<u>Regular Gradient Descent</u>:

- until convergence:

$$\vec{\beta}_{i+1} = \vec{\beta}_i + \nabla L(\vec{\beta}_i)$$

<u>Stochastic Gradient Descent</u>:

- until convergence:
  - for datapoints arranged in a random order:

$$\vec{\beta}_{i+1} = \vec{\beta}_i + \nabla \ell(\vec{\beta}_i ; \underbrace{\vec{x}, y}_{\text{the random datapoint.}})$$