Gradient Descent

The output of a predictor (classifier or regressor) is a function of the input and the predictor’s parameters. We can write this as

\[ y^* = f(w,x) \]

where \( x \) is the input data and \( w \) are the parameters (for example \( k \) in \( k \)-nearest neighbors)
Gradient Descent

In many situations we can define an error function and a parameter set in such a way that the error is \textit{differentiable} with respect to the parameters. Then we can use rules derived from calculus to find the optimal values (i.e. those that minimize the error function).
Gradient Descent

This observation is the basis of several machine learning algorithms, including:

- Linear regression
- Logistic regression
- Neural networks
Gradient Descent

The simplest and still most commonly used algorithm to find this optimal parameters is *gradient descent*.
Gradient Descent

Idea: find the partial derivative of the error with respect to each of the model’s parameters (the gradient).
Modify each of the parameters in the direction that will decrease the error most quickly (the negative gradient)
Gradient Descent

Algorithm:
Repeat until convergence

• Compute the numerical value of the partial derivative of the error with respect to each of the model parameters
• Modify each parameter in the direction of decreased error
Linear Regression

Suppose our model is a simple linear model and training examples have two features:

\[ y^*_i = w_0 + w_1x_{i1} + w_2x_{i2} \]

Given a training set \((X,Y)\), how do we find the values of \(w_0\), \(w_2\) and \(w_2\) that results in the best model performance?

Note: if we have \(n\) training examples, \(X\) is an \(n\)-by-2 array and \(Y\) is an \(n\)-by-1 array.
Linear Regression

We define our loss function as the mean-squared error of the predictions over the training set:

\[ L(w,X,Y) = \frac{1}{2} \sum (y^*_i - y_i)^2 / n \]

Where \( y_i \) is the actual target value and \( y^*_i \) is the model’s prediction given input \( x_i \).
Linear Regression

Combining:

\[ y^*_i = w_0 + w_1 x_{i1} + w_2 x_{i2} \]

and

\[ L(w, X, Y) = \frac{1}{2} \sum (y^*_i - y_i)^2 / n \]

we get:

\[ L(w, X, Y) = \frac{1}{2} \sum (w_0 + w_1 x_{i1} + w_2 x_{i2} - y_i)^2 / n \]
Gradient Descent for Linear Regression

\[ L(w, X, Y) = \frac{1}{2} \sum (w_0 + w_1 x_{i1} + w_2 x_{i2} - y_i)^2/n \]

Now we need to find the gradient (i.e. the partial derivative of the error with respect to each of the parameters).

\[
\frac{\delta L(w, X, Y)}{\delta w_0} = \\
\sum (w_0 + w_1 x_{i1} + w_2 x_{i2} - y_i) \frac{\delta w_0}{n} \\
= \sum (w_0 + w_1 x_{i1} + w_2 x_{i2} - y_i)(1)/n \\
= \sum (y^* - y_i)/n
\]
Gradient Descent for Linear Regression

\[ L(w,X,Y) = \frac{1}{2} \sum (w_0 + w_1x_{i1} + w_2x_{i2} - y_i)^2 / n \]

Similarly:

\[ \frac{\delta L(w,X,Y)}{\delta w_1} = \]

\[ \sum (w_0 + w_1x_{i1} + w_2x_{i2} - y_i) \frac{\delta w_1}{n} \]

\[ = \sum (w_0 + w_1x_{i1} + w_2x_{i2} - y_i) x_{i1} / n \]

\[ = \sum (y^*_i - y_i) x_{i1} / n \]
Gradient Descent for Linear Regression

\[ L(w,X,Y) = \frac{1}{2} \sum (w_0 + w_1 x_{i1} + w_2 x_{i2} - y_i)^2 / n \]

And:
\[ \frac{\delta L(w,X,Y)}{\delta w_2} = \]
\[ \sum (w_0 + w_1 x_{i1} + w_2 x_{i2} - y_i) \delta w_2 / n \]
\[ = \sum (w_0 + w_1 x_{i1} + w_2 x_{i2} - y_i) x_{i2} / n \]
\[ = \sum (y^* - y_i) x_{i2} / n \]
Gradient Descent for Linear Regression

def gd(x, y):
    w = [0, 0, 0]
    lambda = 0.0001  # Learning rate
    for i in range(max_iterations):
        p = w[0] + w[1] * x[:, 0] + w[2] * x[:, 1]
        if np.mean((p - y) * (p - w)) < tolerance: break
        w[0] += lambda * np.mean(p - y)
        w[1] += lambda * np.mean((p - y) * x[:, 0])
        w[2] += lambda * np.mean((p - y) * x[:, 1])
    return w
Linear Regression

For the linear case, we can find a solution without search by setting
\[ \frac{\delta L(w, X, Y)}{\delta w_i} = 0 \]
and solving for \( w \)

The result is
\[ w = (X^T \ast X)^{-1} \ast X^T \]
Logistic Regression

Used for binary classification (despite its name)

Predicts the probability that an example belongs to the positive class

\[
y^*_i = p(y==1) = \frac{1}{1 + e^{-(w_0 + w_1x_1 + w_2x_2)}}
\]

\[
P(y==0) = 1 - p(y==1)
\]
Logistic Regression

Useful to learn probabilities.

\[ \text{sig}(t) = \frac{1}{1 + e^{-t}} \]
Logistic Regression

\[ y^*_i = p(y==1) = \frac{1}{1 + e^{-(w_0 + w_1 x_1 + w_2 x_2)}} \]

\( w_0, w_1, \) and \( w_2 \) are computed from the training data using an optimization algorithm (gradient descent or similar)
Logistic Regression

Loss Function:

\[ L(w,X,Y) = - \sum (y_i \log (y^*_i) + (1 - y_i) \log (1-y^*_i))/n \]

This loss function is called the Cross Entropy

Intuition: Suppose example \((x_i,y_i)\) is correctly classified:

if \(y_i = 1\), and \(y^*_i \approx 1\), then \(L(w,x_i,y_i) \approx 0\), since \((1 - y_i) = 0\) and \(\log (y^*_i) \approx 0\)

if \(y_i = 0\), and \(y^*_i \approx 0\), then \(L(w,x_i,y_i) \approx 0\), since \(y_i = 0\) and \(\log(1-y^*_i) \approx 0\)
Logistic Regression

\[ L(w,X,Y) = -\sum (y_i \log (y_i^*) + (1 - y_i) \log (1 - y_i^*)) / n \]

Intuition: Suppose example \((x_i, y_i)\) is incorrectly classified:
if \(y_i = 1\), and \(y_i^* \approx 0\), then \(L(w,x_i,y_i)\) is large, since \(-\log (y_i^*)\) is large
if \(y_i = 0\), and \(y_i^* \approx 1\), then \(L(w,x_i,y_i)\) is large, since \((1 - y_i) = 1\) and
- \(\log (1 - y_i^*)\) is large
Gradient Descent for Logistic Regression

\[ L(w, X, Y) = - \sum (y_i \log (y^*_i) + (1 - y_i) \log (1-y^*_i))/n \]

We won’t derive the equations, but surprisingly, the gradient update rules turn out to be the same as those for linear regression.