**Comparison between the methods**

**1**. Newton’s Method

Recall the motivation for the gradient descent step at x: we minimize the quadratic function (i.e. Cost Function).

Newton’s method uses in a sense a better quadratic function minimization. It's better because it uses the quadratic approximation (i.e. first AND second partial derivatives).

You can imagine it as a twisted Gradient Descent with the Hessian (the Hessian is a square matrix of second-order partial derivatives of order n X n).

Moreover, the geometric interpretation of Newton's method is that at each iteration one approximates f(x) by a quadratic function around xn, and then takes a step towards the maximum/minimum of that quadratic function (in higher dimensions, this may also be a saddle point). Note that if f(x) happens to be a quadratic function, then the exact extremum is found in one step.

Drawbacks:

It’s computationally expensive because of the Hessian Matrix (i.e. second partial derivatives calculations).

It attracts to Saddle Points which are common in multivariable optimization (i.e. a point that its partial derivatives disagree over whether this input should be a maximum or a minimum point!).

**2. Limited-memory Broyden–Fletcher–Goldfarb–Shanno Algorithm:**

In a nutshell, it is an analogue of Newton’s Method, yet here the Hessian matrix is approximated using updates specified by gradient evaluations (or approximate gradient evaluations). In other words, using estimation to the inverse Hessian matrix.

The term Limited-memory simply means it stores only a few vectors that represent the approximation implicitly.

If I dare say that when the dataset is small, L-BFGS relatively performs the best compared to other methods especially because it saves a lot of memory, however, there are some “serious” drawbacks such that if it is unsafeguarded, it may not converge to anything.

Side note: This solver has become the default solver in sklearn LogisticRegression since version 0.22, replacing LIBLINEAR.

**3. A Library for Large Linear Classification:**

It’s a linear classification that supports logistic regression and linear support vector machines.

The solver uses a Coordinate Descent (CD) algorithm that solves optimization problems by successively performing approximate minimization along coordinate directions or coordinate hyperplanes.

LIBLINEAR is the winner of the ICML 2008 large-scale learning challenge. It applies automatic parameter selection (a.k.a L1 Regularization) and it’s recommended when you have high dimension dataset (recommended for solving large-scale classification problems)

Drawbacks:

It may get stuck at a non-stationary point (i.e. non-optima) if the level curves of a function are not smooth.

Also cannot run in parallel.

It cannot learn a true multinomial (multiclass) model; instead, the optimization problem is decomposed in a “one-vs-rest” fashion, so separate binary classifiers are trained for all classes.

Side note: According to Scikit Documentation: The “liblinear” solver was the one used by default for historical reasons before version 0.22. Since then, the default use is Limited-memory Broyden–Fletcher–Goldfarb–Shanno Algorithm.

**4. Stochastic Average Gradient:**

The SAG method optimizes the sum of a finite number of smooth convex functions. Like stochastic gradient (SG) methods, the SAG method's iteration cost is independent of the number of terms in the sum. However, by incorporating a memory of previous gradient values, the SAG method achieves a faster convergence rate than black-box SG methods.

It is faster than other solvers for large datasets when both the number of samples and the number of features are large.

Drawbacks:

It only supports L2 penalization.

This is not really a drawback, but more like a comparison: although SAG is suitable for large datasets, with a memory cost of O(N), it can be less practical for very large N (as the most recent gradient evaluation for each function needs to be maintained in the memory). This is usually not a problem, but a better option would be SVRG 1, 2 which is unfortunately not implemented in scikit-learn!

**5. SAGA:**

The SAGA solver is a variant of SAG that also supports the non-smooth penalty L1 option (i.e. L1 Regularization). This is therefore the solver of choice for sparse multinomial logistic regression. It also has a better theoretical convergence compared to SAG.

Drawbacks:

This is not really a drawback, but more like a comparison: SAGA is similar to SAG with regard to memory cost. That's it's suitable for large datasets, yet in edge cases where the dataset is very large, the SVRG 1, 2 would be a better option (unfortunately not implemented in scikit-learn)!

Side note: According to Scikit Documentation: The SAGA solver is often the best choice.

Please note the attributes "Large" and "Small" used in Scikit-Learn and in this comparison are relative. AFAIK, there is no universal unanimous and accurate definition of the dataset boundaries to be considered as "Large", "Too Large", "Small", "Too Small"...etc!

Here we have a summary of the penalties for each method according to:

<https://scikit-learn.org/stable/modules/linear_model.html>

Tabela

Descrição gerada automaticamente