Module/framework/package	**Name and brief description of algorithm**	**An example of a situation where using the provided GLM implementation provides superior performance compared to that of base R or its equivalent in Python (identify the equivalent in Python)**
Base R	IRLS (Iteratively Reweighted Least Squares) - The algorithm performs Fisher scoring by solving weighted least squares problems sequentially until it reaches convergence. The algorithm uses score function and Fisher information matrix computations to update parameter estimates through matrix inversion.	The reference implementation Base R delivers outstanding statistical functionality. The Python implementation of this model exists within the GLM module of statsmodels. The system provides complete diagnostic tools (deviance, AIC, etc.) and achieves robust convergence on well-conditioned problems alongside tested numerical stability despite not being designed for big data.
Big Data version of R	Chunked and Distributed IRLS - The bigglm data processing system divides information into sections to work with large datasets that exceed system memory capacity. The distribution of computation across cores or nodes happens through frameworks that include snowfall, foreach and future. The system distributes data subsets among workers who calculate results which are combined through communication protocols.	The combination of bigglm and future.apply and batchtools enables R to handle enormous datasets beyond RAM memory limits and realize linear scaling with extra cores and reach hundreds of computational nodes while equivalent Python implementations using Dask and statsmodels are less mature.
Dask ML	ADMM and Proximal Methods - The optimization problem using ADMM (Alternating Direction Method of Multipliers) gets divided between workers for independent processing. Each	This system provides superior performance for datasets that exceed memory limits yet do not need a complete Spark cluster because it offers lazy evaluation capabilities for datasets 10-100× larger than

	worker handles its assigned data portion before results synchronize through proximal operator communication. The system implements proximal gradient methods together with L-BFGS and Newton variants that optimize distributed computation.	RAM while minimizing communication overhead through efficient task scheduling and excels with L1 regularization on large sparse datasets when using proximal methods. It outperforms scikit-learn for out-of-core processing on single machines and small clusters. The R equivalent for this operation can be found in the future package.
Spark R	Distributed Quasi-Newton Methods - Implements L-BFGS (Limited-memory Broyden–Fletcher–Goldfarb–Shanno) and OWL-QN (Orthant-Wise Limited-memory Quasi-Newton) with IRR (Iterative Residual Rescaling). The methods use limited memory to approximate second-order information which makes them suitable for distributed processing.	The implementation uses massive datasets along with features that make it process terabyte-scale data across hundreds of nodes while providing fault tolerance in unreliable clusters and efficient feature encoding with customizable string handling options when compared to PySpark. The system provides optimal performance during analysis of large datasets that contain numerous categorical attributes.
Spark optimization	Distributed First-Order Optimization - The framework executes gradient descent methods (SGD and L-BFGS) through tree-based gradient aggregation protocols. The package contains adaptive step size algorithms together with minibatch algorithms and specialized convergence optimization methods which work specifically in distributed computing environments.	The system provides exceptional results when used for distributed massive-scale training because it utilizes tree aggregation structures to minimize cluster communication and achieves optimal performance through the processing of minibatches on diverse hardware configurations. It works best with models exceeding one million features. Users of Python access Spark through PySpark whereas R users operate SparkR.

Scikit-Learn	Specialized Solvers Suite - The package features various optimization methods named 'newton-cg', 'lbfgs', 'liblinear', and 'sag'/'saga.' Each optimization method shows excellence in solving problems with different data attributes and regularization types.	The 'sag'/ 'saga' algorithms demonstrate better performance as they scale linearly with large datasets yet 'liblinear' thrives when working with sparse matrices while employing L1 regularization and 'lbfgs' shows strong convergence acting on L2 problems. SAGA stands as the best choice for large sparse L1 datasets. The glmnet package from R serves as an equivalent to scikit-learn but scikit-learn provides superior integration with the Python environment.