HDS 5230: High Performance Computing

Week 08 - GLM Investigation

Harika Pamulapati

Professor: Adam Doyle

TA: Rohith Odela

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	Fisher Scoring with QR Decomposition. Base R performs GLM fitting by using Fisher scoring which substitutes the expected information matrix for the observed Hessian in Newton's method. The algorithm resolves weighted least squares problems through QR decomposition at each step to achieve numerical stability when dealing with near-collinear predictors.	The GLM implementation in Base R provides the best solution for experimental design analysis that requires complex interaction terms and nested random effects. The dispersion parameter estimation and quasi-family distribution capabilities of Base R GLM exceed those of Python statsmodels for ecological count data analysis particularly when data exhibits overdispersion. The diagnostic capabilities include Cook's distance and hat values among others.
Big Data version of R	The algorithm performs Block Coordinate Descent through Data Partitioning. Through sparklyr users can access Spark MLlib block coordinate descent algorithms which distribute data processing across Spark nodes. The distributed implementation of GLMs works across worker nodes to conduct model updates by processing sufficient statistics in an optimized fashion which minimizes network communication.	The telecommunication data analysis using sparklyr handles tens of millions of customers with hundreds of features more effectively than base R's implementation which would result in a system crash. The system offers superior R ecosystem compatibility than PySpark while also providing a simple distributed/local operation transition through dplyr syntax for feature engineering applications.
Dask ML	Proximal Gradient Methods with Out-of-Core Processing. Dask ML uses proximal gradient methods for GLMs	Dask ML provides superior performance than scikit-learn for fitting elastic-net regularized logistic regression

	to process extensive datasets through chunked data handling. The algorithm applies proximal operators for regularization processing and takes advantage of Dask's task management protocols to allocate computational tasks on available resources against memory limitations.	on genomic data consisting of millions of SNPs across thousands of samples through its out-of-core processing capabilities. The system enables the analysis of datasets that exceed RAM capacity by 5-10 times without needing a complete cluster configuration. Through its incremental processing approach Dask ML provides better early stopping capabilities which base scikit-learn implementations lack.
Spark R	Distributed Newtonization with Tree Aggregation. The Spark R implementation distributes gradient and Hessian calculations through a communication-efficient version of Newton's method. The system makes use of tree aggregation patterns to minimize network overhead when gathering model statistics and includes specialized categorical variable encoders that reduce memory consumption during model training.	The process of one-hot encoding hundreds of categorical variables for insurance risk modeling produces thousands of model parameters which Spark R handles more efficiently than base R or Python pandas+statsmodels. Categorical variables and null value handling within the system operates eficiently which reduces data preparation work. Actuaries who understand R syntax can easily transition to big data applications through Spark R rather than equivalent Python tools.
Spark optimization	LBFGS-OWLQN with RDD-based Parallelism. The optimization module of Spark uses Orthant-Wise Limited-memory Quasi-Newton (OWLQN) as its L1-regularized problem solver. The system distributes computationentionally splits both data and computation through Resilient Distributed	The click prediction model creation process for online advertisements with billions of sparse features works better with Spark MLlib's OWLQN implementation than with scikit-learn or R's glmnet methods. The efficient processing of extreme sparsity levels (greater than 99.9% zeros) combined with

	Datasets (RDDs) while optimizing sparse data structures and implementing techniques to speed up convergence.	numerical stability makes OWLQN superior to standalone implementations in R and Python when selecting features from millions of potential predictors.
Scikit-Learn	Specialized Dual and Primal Solvers with Automatic Selection. The Scikit-learn framework provides users with three different solver approaches which include dual coordinate descent (liblinear) and SAG/SAGA (Stochastic Average Gradient) and Newton-based methods. The system chooses the best available strategies automatically from the combination of problem characteristics such as dataset size and specific parameters and it also features a warm-start feature for hyperparameter optimization.	The SAGA solver from scikit-learn performs better than glmnet from R for document classification using TF-IDF features that consist of tens of thousands of documents with high-dimensional sparse feature matrices. This tool performs well for NLP pipelines because its vectorized operations work efficiently with specialized sparse matrix handling when integrating feature extraction with model training. Through its sklearn Pipeline API the system provides a smoother integration process than comparable workflows in R.

References

1. Base R https://www.rdocumentation.org/packages/stats/versions/3.6.2/topics/glm

2. Big data version of R https://cran.r-project.org/web/views/HighPerformanceComputing.html

3. Dask ML https://ml.dask.org/glm.html

4. Spark R https://spark.apache.org/docs/3.5.0/api/R/reference/spark.glm.html

5. Spark Optimization https://github.com/apache/spark/blob/master/docs/mllib-optimization.md

6. Scikit learn https://scikit-learn.org/stable/modules/linear_model.html