`SystemML` Algorithms Reference

August 22, 2014

1 Descriptive Statistics

Descriptive statistics are used to quantitatively describe the main characteristics of the data. They provide meaningful summaries computed over different observations or data records collected in a study. These summaries typically form the basis of the initial data exploration as part of a more extensive statistical analysis. Such a quantitative analysis assumes that every variable (also known as, attribute or column) in the data have a specific **level of measurement**.

The measurement level of a variable, often called as **variable type**, can either be *scale* or *categorical*. A *scale* variable represents the data measured on an interval scale or ratio scale. Examples of scale variables include 'Height', 'Weight', 'Salary', and 'Temperature'. Scale variables are often referred to as *continuous* variables. In contrast, a *categorical* variable denotes the data with a limited number of distinct values or categories. Examples of categorical variables include 'Gender', 'Region', 'Hair color', 'Zipcode', and 'Level of Satisfaction'. Categorical variables can further be classfified into two types, *nominal* and *ordinal*, depending on whether the categories in the variable can be ordered via an intrinsic ranking. For example, there is no meaningful ranking among distinct values in 'Hair color' variable, while the categories in 'Level of Satisfaction' can be ranked from highly dissatisfied to highly satisfied.

1.1 Univariate Statistics

Description

Univariate statistics are the simplest form of *Descriptive Statistics* in quantitative analysis. They are used to quantitatively describe the main characteristics of each variable or column in the data. For a given data set, the `Univariate.dml` script computes all relevant univariate statistics for each variable in the data. The variable type governs the exact set of statistics computed for that variable. For example, the statistic *average* can only be computed on a continuous variable like 'Height' and 'Temperature'. It does not make sense to compute an *average* of a categorical attribute like 'Hair Color'.

For a scale or continuous variable, the `Univariate.dml` script computes the following univariate statistics:

- **Order statistics**:
  - *minimum*: The smallest among all values.
  - *maximum*: The largest among all values.
  - *range*: The difference between the largest and the smallest value. It provides information about the spread of values.
  - *median*: The middle or center value that separates higher half of the data (in a sorted order) from the lower half. Note that there can be a unique median only when the total number of data points is odd. When the number of data points is even, the median is computed as the mean of two middle values. For example, the median of $\{1, 5, 6, 8, 10\}$ is 6 whereas the median of $\{1, 5, 6, 8, 10, 14\}$ is $\frac{6+8}{2} = 7$. Median is same as the 50^{th} *percentile* or 2^{nd} *quartile*.
  - 1^{st} *and* 3^{rd} *quartiles*: Quartiles are the data points that divide an ordered/sorted set of data points into four equal groups. The 1^{st} *quartile* or the 25^{th} *percentile* splits sorted data into lowest 25% and highest 75%. In other words, it is the middle number between the smallest value and the median. Similarly, the 3^{rd} *quartile* or the 75^{th} *percentile* divides the sorted data into lowest 75% and highest 25% – i.e., it is the middle number between the median and the largest value in the data set.
- **Measures of Central Tendency**:
  - *mean*: The sample mean of values in the variable, which is computed as the ratio between sum of values and the number of values. It is also often referred to as *average*. Mean is often used as a "typical" value – for example, while imputing missing values.
  - *standard error in mean*: It is the standard deviation of the sample mean, which is an estimate of population's true mean. It is computed using the following formula: $\frac{sd}{\sqrt{n}}$, where sd is the estimate of population standard deviation that is computed using sample values, and n is the sample size i.e., number of values in the data set.
  - *Inter Quartile Mean*: It is the mean of values between 1^{st} and 3^{rd} quartiles i.e., lowest 25% and the highest 25% of the values are not considered for the computation of this measure.
- **Measures of Dispersion**:
  - *variance*: It measures how the values in the given data set are spread out around the mean. Variance is same as the the second central moment.
  - *standard deviation*: It is the square root of variance.

- *skewness*: It measures how symmetrically the values are spread out around the mean. If the left tail (the distribution of given values below mean) is longer than the right tail then the value of skewness will be negative. On the other hand, a positive value indicates the right tail is longer than the left tail. Skewness is a function of third centeral moment. For a perfectly normal distribution of values, the value of skewness will be close to zero.
- *standard error in skewness*: The ratio of skewness to its standard error can be used as a test of normality (that is, you can reject normality if the ratio is less than -2 or greater than +2). A large positive value for skewness indicates a long right tail; an extreme negative value indicates a long left tail.
- *kurtosis*: kurtosis is also a measure of shape of the distribution of values. In particular, it measures the "peakedness" of the distribution. In other words, it quantifies how tall and sharp the central peak is, relative to a standard bell curve. It is a function of fourth central moment.
- *standard error in kurtosis*: The ratio of kurtosis to its standard error can be used as a test of normality (that is, you can reject normality if the ratio is less than -2 or greater than +2). A large positive value for kurtosis indicates that the tails of the distribution are longer than those of a normal distribution; a negative value for kurtosis indicates shorter tails (becoming like those of a box-shaped uniform distribution).

Usage

`-f` *path*`/univar-stats.dml -nvargs X=`*path*`/`*file* TYPES=*path*`/`*file* STATS=*path*`/`*file*

Arguments

- X: Location (on HDFS) of input data.
- TYPES: Location (on HDFS) of a 1-row matrix whose i^{th} column contains the type (measurement level) of the i^{th} attribute in the data. See details to find out how to indicate what type.
- STATS: Location directory (on HDFS) where all computed statistics will be stored.

Details

Given an input matrix X, this script computes the set of all relevant univariate statistics for each attribute or column in X. The list of statistics to be computed depends on the *type* or *measurement level* of each column. TYPES argument specifies the type of all columns in a 1 row matrix. The types must be provided as per the following convention:

- For scale, 1.
- For nominal, 2.
- For ordinal, 3.

STATS argument points to the location on HDFS where the output matrix of computed statistics is stored.

Returns

The output matrix containing all computed statistics is of size 17 rows and as many columns as in the input matrix X. Each row corresponds to a particular statistic, according to the following convention:

- Row 1: Minimum.
- Row 2: Maximum.
- Row 3: Range.
- Row 4: Mean.
- Row 5: Variance.
- Row 6: Standard deviation.
- Row 7: Standard error of mean.
- Row 8: Coefficient of variation.
- Row 9: Skewness.
- Row 10: Kurtosis.
- Row 11: Standard error of skewness.
- Row 12: Standard error of kurtosis.
- Row 13: Median.
- Row 14: Inter quartile mean.
- Row 15: Number of categories.
- Row 16: Mode.
- Row 17: Number of modes.

The first 14 statistics are applicable for *scale* columns, and the last 3 statistics (including 'Number of categories', 'Mode', and 'Number of modes') are applicable for categorical i.e., nominal and ordinal columns.

Examples

```
hadoop jar SystemML.jar -f univar-stats.dml -nvargs X=/user/biadmin/X.mtx
                                                    TYPES=/user/biadmin/types.mtx
                                                    STATS=/user/biadmin/stats.mtx
```

1.2 Bivariate Statistics

Description

The `bivar-stats.dml` script computes common bivariate statistics, such as Pearson's-r correlation and chi-squared statistic, for many pairs of variables.

Usage

`-f` *path*`/bivar-stats.dml -nvargs X=`*path*`/`*file* index$_1$=*path*`/`*file* index$_2$=*path*`/`*file* types$_1$=*path*`/`*file* types$_2$=*path*`/`*file* `OUTDIR=`*path*`/`*file*

Arguments

- X: Location (on HDFS) of the data whose inter-column correlations are required.
- index$_1$: Location (on HDFS) of a 1-row matrix whose i^{th} column contains the *first* index of the i^{th} pair whose correlation needs to be computed.
- index$_2$: Location (on HDFS) of a 1-row matrix whose i^{th} column contains the *second* index of the i^{th} pair whose correlation needs to be computed.
- types$_1$: Location (on HDFS) of a 1-row matrix whose i^{th} column contains the type of the *first* column in the i^{th} pair whose correlation needs to be computed. See details to find out how to indicate what type.
- types$_2$: Location (on HDFS) of a 1-row matrix whose i^{th} column contains the type of the *second* column in the i^{th} pair whose correlation needs to be computed. See details to find out how to indicate what type.
- OUTDIR: Location directory (on HDFS) where all results will be stored.

Details

This script takes an input matrix X and can perform for each pair of columns the relevant correlation computation. In general, numerous kinds correlations have been proposed. Most of these test how strongly or weakly correlated the values in the two columns are and thus, are also referred to as statistical significance tests. This script includes the following statistical significance tests:

- For a pair of scale columns (numeric values including fractions), Pearson's-R.
- For a pair of nominal columns (with finite-sized, fixed, unordered domains), chi-squared test.
- For a pair consisting of one scale column and one nominal column, F-test.
- For a pair of ordinal columns (ordered domains depicting ranks), Spearman's Rho.

Type	Code
Scale	1
Nominal	2
Ordinal	3

Table 1: Types with their codes.

Name	Row=1	Row=2	Specialized Rows
bivar.scale.scale.stats	Column ID 1	Column ID 2	Pearson's R (Row 3)
bivar.nominal.nominal.stats	Column ID 1	Column ID 2	Chi-squared statistic (Row 3) Degrees of freedom (Row 4) P-value (Row 5) Cramer's V (Row 6)
bivar.nominal.scale.stats	Column ID 1	Column ID 2	Eta (Row 3) F statistic (Row 4)
bivar.ordinal.ordinal.stats	Column ID 1	Column ID 2	Spearman's Rho (Row 3)

Table 2: Description of output matrices containing significance test results.

A user can feed as input the matrix X containing all columns whose correlations need to be computed. Using the index_1 and index_2 arguments, one can pass in a set of indices referring to the columns in X whose correlations are of interest. Note that both index_1 and index_2 should consist of 1 row each. In each iteration, the script will pick an index from index_1 and index_2 and access those columns from X. Correspondingly, the types of each pair should be provided in the types_1 and types_2 arguments which should also each consist of 1 row. Table 1 lists the types the script supports.

The script orgainizes its results into (potentially) four output matrices, one per type of test. The type of the test is defined using the types of the columns that were used for the test. Table 2 describes what each column in each output matrix contains.

Note that, in Table 2 "Column ID 1" and "Column ID 2" refer to column indices of the data matrix X which were used to perform the test. Moreover, if the output matrix does not contain a value in a certain cell then it should be interpreted as a 0 (sparse matrix representation).

Returns

A collection of (potentially) 4 matrices. Each matrix contains correlations resulting from a different type of test. There's one each for scale-scale (Pearson's R), nominal-nominal (chi-squared test), nominal-scale (F-test) and ordinal-ordinal (Spearman's Rho) correlation results. If any of these matrices is not produced then no pair of columns required the corresponding type of test.

Examples

```
hadoop jar SystemML.jar -f bivar-stats.dml -nvargs X=/user/biadmin/X.mtx
```

Month of the year	October		November		December		Oct – Dec	
Customers, millions	0.6	1.4	1.4	0.6	3.0	1.0	5.0	3.0
Promotion (0 or 1)	0	1	0	1	0	1	0	1
Avg. sales per 1000	0.4	0.5	0.9	1.0	2.5	2.6	1.8	1.3

Table 3: Stratification example: the effect of the promotion on average sales becomes reversed and amplified (from +0.1 to −0.5) if we ignore the months.

```
index1=/user/biadmin/S1.mtx
index2=/user/biadmin/S2.mtx
types1=/user/biadmin/K1.mtx
types2=/user/biadmin/K2.mtx
OUTDIR=/user/biadmin/stats.mtx
```

1.3 Stratified Bivariate Statistics

Description

The `stratstats.dml` script computes common bivariate statistics, such as correlation, slope, and their p-value, in parallel for many pairs of input variables in the presence of a confounding categorical variable. The values of this confounding variable group the records into strata (subpopulations), in which all bivariate pairs are assumed free of confounding. The script uses the same data model as in one-way analysis of covariance (ANCOVA), with strata representing population samples. It also outputs univariate stratified and bivariate unstratified statistics.

To see how data stratification mitigates confounding, consider an (artificial) example in Table 3. A highly seasonal retail item was marketed with and without a promotion over the final 3 months of the year. In each month the sale was more likely with the promotion than without it. But during the peak holiday season, when shoppers came in greater numbers and bought the item more often, the promotion was less frequently used. As a result, if the 4-th quarter data is pooled together, the promotion's effect becomes reversed and magnified. Stratifying by month restores the positive correlation.

The script computes its statistics in parallel over all possible pairs from two specified sets of covariates. The 1-st covariate is a column in input matrix X and the 2-nd covariate is a column in input matrix Y; matrices X and Y may be the same or different. The columns of interest are given by their index numbers in special matrices. The stratum column, specified in its own matrix, is the same for all covariate pairs.

Both covariates in each pair must be numerical, with the 2-nd covariate normally distributed given the 1-st covariate (see Details). Missing covariate values or strata are represented by "NaN". Records with NaN's are selectively omitted wherever their NaN's are material to the output statistic.

Usage

`-f` *path/*`stratstats.dml` `-nvargs` `X=`*path/file* `Xcid=`*path/file* `Y=`*path/file* `Ycid=`*path/file* `S=`*path/file* `Scid=`*int* `O=`*path/file* `fmt=`*format*

Arguments

`X:` Location (on HDFS) to read matrix X whose columns we want to use as the 1-st covariate (i.e. as the feature variable)

`Xcid:` (default: `" "`) Location to read the single-row matrix that lists all index numbers of the X-columns used as the 1-st covariate; the default value means "use all X-columns"

`Y:` (default: `" "`) Location to read matrix Y whose columns we want to use as the 2-nd covariate (i.e. as the response variable); the default value means "use X in place of Y"

`Ycid:` (default: `" "`) Location to read the single-row matrix that lists all index numbers of the Y-columns used as the 2-nd covariate; the default value means "use all Y-columns"

`S:` (default: `" "`) Location to read matrix S that has the stratum column. Note: the stratum column must contain small positive integers; all fractional values are rounded; stratum IDs of value ≤ 0 or NaN are treated as missing. The default value for `S` means "use X in place of S"

`Scid:` (default: `1`) The index number of the stratum column in S

`O:` Location to store the output matrix defined in Table 4

`fmt:` (default: `"text"`) Matrix file output format, such as `text`, `mm`, or `csv`; see read/write functions in SystemML Language Reference for details.

Details

Suppose we have n records of format (i, x, y), where $i \in \{1, \ldots, k\}$ is a stratum number and (x, y) are two numerical covariates. We want to analyze conditional linear relationship between y and x conditioned by i. Note that x, but not y, may represent a categorical variable if we assign a numerical value to each category, for example 0 and 1 for two categories.

We assume a linear regression model for y:

$$y_{i,j} \,=\, \alpha_i + \beta x_{i,j} + \varepsilon_{i,j}\,, \quad \text{where} \;\; \varepsilon_{i,j} \sim \text{Normal}(0, \sigma^2) \tag{1}$$

Here $i = 1 \ldots k$ is a stratum number and $j = 1 \ldots n_i$ is a record number in stratum i; by n_i we denote the number of records available in stratum i. The noise term $\varepsilon_{i,j}$ is assumed to have the same variance in all strata. When $n_i > 0$, we can estimate the means of $x_{i,j}$ and $y_{i,j}$ in stratum i as

$$\bar{x}_i \,=\, \Big(\sum\nolimits_{j=1}^{n_i} x_{i,j}\Big)/n_i\,; \quad \bar{y}_i \,=\, \Big(\sum\nolimits_{j=1}^{n_i} y_{i,j}\Big)/n_i$$

If β is known, the best estimate for α_i is $\bar{y}_i - \beta\bar{x}_i$, which gives the prediction error sum-of-squares of

$$\sum\nolimits_{i=1}^{k} \sum\nolimits_{j=1}^{n_i} \big(y_{i,j} - \beta x_{i,j} - (\bar{y}_i - \beta\bar{x}_i)\big)^2 \;=\; \beta^2\, V_x \,-\, 2\beta\, V_{x,y} \,+\, V_y \tag{2}$$

	Col.#	Meaning		Col.#	Meaning
1-st covariate	01	X-column number	2-nd covariate	11	Y-column number
	02	presence count for x		12	presence count for y
	03	global mean (x)		13	global mean (y)
	04	global std. dev. (x)		14	global std. dev. (y)
	05	stratified std. dev. (x)		15	stratified std. dev. (y)
	06	R^2 for $x \sim$ strata		16	R^2 for $y \sim$ strata
	07	adjusted R^2 for $x \sim$ strata		17	adjusted R^2 for $y \sim$ strata
	08	p-value, $x \sim$ strata		18	p-value, $y \sim$ strata
	09–10	reserved		19–20	reserved
$y \sim x$, NO strata	21	presence count (x, y)	$y \sim x$ AND strata	31	presence count (x, y, s)
	22	regression slope		32	regression slope
	23	regres. slope std. dev.		33	regres. slope std. dev.
	24	correlation $= \pm\sqrt{R^2}$		34	correlation $= \pm\sqrt{R^2}$
	25	residual std. dev.		35	residual std. dev.
	26	R^2 in y due to x		36	R^2 in y due to x
	27	adjusted R^2 in y due to x		37	adjusted R^2 in y due to x
	28	p-value for "slope = 0"		38	p-value for "slope = 0"
	29	reserved		39	# strata with ≥ 2 count
	30	reserved		40	reserved

Table 4: The `stratstats.dml` output matrix has one row per each distinct pair of 1-st and 2-nd covariates, and 40 columns with the statistics described here.

where V_x, V_y, and $V_{x,y}$ are, correspondingly, the "stratified" sample estimates of variance $\mathrm{Var}(x)$ and $\mathrm{Var}(y)$ and covariance $\mathrm{Cov}(x, y)$ computed as

$$V_x \,=\, \sum\nolimits_{i=1}^{k} \sum\nolimits_{j=1}^{n_i} \big(x_{i,j} - \bar{x}_i\big)^2; \quad V_y \,=\, \sum\nolimits_{i=1}^{k} \sum\nolimits_{j=1}^{n_i} \big(y_{i,j} - \bar{y}_i\big)^2;$$
$$V_{x,y} \,=\, \sum\nolimits_{i=1}^{k} \sum\nolimits_{j=1}^{n_i} \big(x_{i,j} - \bar{x}_i\big)\big(y_{i,j} - \bar{y}_i\big)$$

They are stratified because we compute the sample (co-)variances in each stratum i separately, then combine by summation. The stratified estimates for $\mathrm{Var}(X)$ and $\mathrm{Var}(Y)$ tend to be smaller than the non-stratified ones (with the global mean instead of $\bar{x}_i$ and $\bar{y}_i$) since $\bar{x}_i$ and $\bar{y}_i$ fit closer to $x_{i,j}$ and $y_{i,j}$ than the global means. The stratified variance estimates the uncertainty in $x_{i,j}$ and $y_{i,j}$ given their stratum i.

Minimizing over β the error sum-of-squares (2) gives us the regression slope estimate $\hat{\beta} = V_{x,y}/V_x$, with (2) becoming the residual sum-of-squares (RSS):

$$\mathrm{RSS} \,=\, \sum\nolimits_{i=1}^{k} \sum\nolimits_{j=1}^{n_i} \big(y_{i,j} - \hat{\beta} x_{i,j} - (\bar{y}_i - \hat{\beta}\bar{x}_i)\big)^2 \,=\, V_y\big(1 \,-\, V_{x,y}^2/(V_x V_y)\big)$$

The quantity $\hat{R}^2 = V_{x,y}^2/(V_x V_y)$, called *R-squared*, estimates the fraction of stratified variance in $y_{i,j}$ explained by covariate $x_{i,j}$ in the linear regression model (1). We define *stratified correlation* as the square root of $\hat{R}^2$ taken with the sign of $V_{x,y}$. We also use RSS to estimate the residual standard deviation σ

in (1) that models the prediction error of $y_{i,j}$ given $x_{i,j}$ and the stratum:

$$\hat{\beta} = \frac{V_{x,y}}{V_x};\;\; \hat{R} = \frac{V_{x,y}}{\sqrt{V_x V_y}};\;\; \hat{R}^2 = \frac{V_{x,y}^2}{V_x V_y};\;\; \hat{\sigma} = \sqrt{\frac{\mathrm{RSS}}{n-k-1}}\;\; \left(n = \sum_{i=1}^{k} n_i\right)$$

The t-test and the F-test for the null-hypothesis of "$\beta = 0$" are obtained by considering the effect of $\hat{\beta}$ on the residual sum-of-squares, measured by the decrease from V_y to RSS. The F-statistic is the ratio of the "explained" sum-of-squares to the residual sum-of-squares, divided by their corresponding degrees of freedom. There are $n-k$ degrees of freedom for V_y, parameter β reduces that to $n-k-1$ for RSS, and their difference $V_y - \mathrm{RSS}$ has just 1 degree of freedom:

$$F = \frac{(V_y - \mathrm{RSS})/1}{\mathrm{RSS}/(n-k-1)} = \frac{\hat{R}^2\,(n-k-1)}{1-\hat{R}^2};\quad t = \hat{R}\sqrt{\frac{n-k-1}{1-\hat{R}^2}}.$$

The t-statistic is simply the square root of the F-statistic with the appropriate choice of sign. If the null hypothesis and the linear model are both true, the t-statistic has Student t-distribution with $n-k-1$ degrees of freedom. We can also compute it if we divide $\hat{\beta}$ by its estimated standard deviation:

$$\mathrm{st.dev}(\hat{\beta})_{\mathrm{est}} = \hat{\sigma}\,/\sqrt{V_x} \quad\Longrightarrow\quad t = \hat{R}\sqrt{V_y}\,/\,\hat{\sigma} = \beta\,/\,\mathrm{st.dev}(\hat{\beta})_{\mathrm{est}}$$

The standard deviation estimate for β is included in `stratstats.dml` output.

Returns

The output matrix format is defined in Table 4.

Examples

```
hadoop jar SystemML.jar -f stratstats.dml -nvargs
   X=/user/biadmin/X.mtx Xcid=/user/biadmin/Xcid.mtx
   Y=/user/biadmin/Y.mtx Ycid=/user/biadmin/Ycid.mtx
   S=/user/biadmin/S.mtx Scid=2 O=/user/biadmin/Out.mtx fmt=csv
hadoop jar SystemML.jar -f stratstats.dml -nvargs
   X=/user/biadmin/Data.mtx Xcid=/user/biadmin/Xcid.mtx
   Ycid=/user/biadmin/Ycid.mtx Scid=7 O=/user/biadmin/Out.mtx
```

2 Classification

2.1 Multinomial Logistic Regression

Description

Our logistic regression script performs both binomial and multinomial logistic regression. The script is given a dataset (X, Y) where matrix X has m columns and matrix Y has one column; both X and Y have n rows. The rows of X and Y are viewed as a collection of records: $(X, Y) = (x_i, y_i)_{i=1}^{n}$ where x_i is a numerical vector of explanatory (feature) variables and y_i is a

categorical response variable. Each row x_i in X has size $\dim x_i = m$, while its corresponding y_i is an integer that represents the observed response value for record i.

The goal of logistic regression is to learn a linear model over the feature vector x_i that can be used to predict how likely each categorical label is expected to be observed as the actual y_i. Note that logistic regression predicts more than a label: it predicts the probability for every possible label. The binomial case allows only two possible labels, the multinomial case has no such restriction.

Just as linear regression estimates the mean value μ_i of a numerical response variable, logistic regression does the same for category label probabilities. In linear regression, the mean of y_i is estimated as a linear combination of the features: $\mu_i = \beta_0 + \beta_1 x_{i,1} + \ldots + \beta_m x_{i,m} = \beta_0 + x_i \beta_{1:m}$. In logistic regression, the label probability has to lie between 0 and 1, so a link function is applied to connect it to $\beta_0 + x_i \beta_{1:m}$. If there are just two possible category labels, for example 0 and 1, the logistic link looks as follows:

$$\text{Prob}[y_i = 1 \mid x_i; \beta] \;=\; \frac{e^{\,\beta_0 + x_i \beta_{1:m}}}{1 + e^{\,\beta_0 + x_i \beta_{1:m}}}; \quad \text{Prob}[y_i = 0 \mid x_i; \beta] \;=\; \frac{1}{1 + e^{\,\beta_0 + x_i \beta_{1:m}}}$$

Here category label 0 serves as the *baseline*, and function $\exp(\beta_0 + x_i \beta_{1:m})$ shows how likely we expect to see "$y_i = 1$" in comparison to the baseline. Like in a loaded coin, the predicted odds of seeing 1 versus 0 are $\exp(\beta_0 + x_i \beta_{1:m})$ to 1, with each feature $x_{i,j}$ multiplying its own factor $\exp(\beta_j x_{i,j})$ to the odds. Given a large collection of pairs (x_i, y_i), $i = 1 \ldots n$, logistic regression seeks to find the β_j's that maximize the product of probabilities $\text{Prob}[y_i \mid x_i; \beta]$ for actually observed y_i-labels (assuming no regularization).

Multinomial logistic regression [1] extends this link to $k \geq 3$ possible categories. Again we identify one category as the baseline, for example the k-th category. Instead of a coin, here we have a loaded multisided die, one side per category. Each non-baseline category $l = 1 \ldots k-1$ has its own vector $(\beta_{0,l}, \beta_{1,l}, \ldots, \beta_{m,l})$ of regression parameters with the intercept, making up a matrix B of size $(m+1) \times (k-1)$. The predicted odds of seeing non-baseline category l versus the baseline k are $\exp\left(\beta_{0,l} + \sum_{j=1}^{m} x_{i,j} \beta_{j,l}\right)$ to 1, and the predicted probabilities are:

$$l < k: \quad \text{Prob}[y_i = l \mid x_i; B] \;=\; \frac{\exp\left(\beta_{0,l} + \sum_{j=1}^{m} x_{i,j} \beta_{j,l}\right)}{1 \;+\; \sum_{l'=1}^{k-1} \exp\left(\beta_{0,l'} + \sum_{j=1}^{m} x_{i,j} \beta_{j,l'}\right)}; \tag{3}$$

$$\text{Prob}[y_i = k \mid x_i; B] \;=\; \frac{1}{1 \;+\; \sum_{l'=1}^{k-1} \exp\left(\beta_{0,l'} + \sum_{j=1}^{m} x_{i,j} \beta_{j,l'}\right)}. \tag{4}$$

The goal of the regression is to estimate the parameter matrix B from the provided dataset $(X, Y) = (x_i, y_i)_{i=1}^{n}$ by maximizing the product of $\text{Prob}[y_i \mid x_i; B]$ over the observed labels y_i. Taking its logarithm, negating, and adding a regularization term gives us a minimization objective:

$$f(B; X, Y) \;=\; -\sum_{i=1}^{n} \log \text{Prob}[y_i \mid x_i; B] \;+\; \frac{\lambda}{2} \sum_{j=1}^{m} \sum_{l=1}^{k-1} |\beta_{j,l}|^2 \;\to\; \min \tag{5}$$

The optional regularization term is added to mitigate overfitting and degeneracy in the data; to reduce bias, the intercepts $\beta_{0,l}$ are not regularized. Once the $\beta_{j,l}$'s are accurately estimated, we can make predictions about the category label y for a new feature vector x using Eqs. (3) and (4).

Usage

`-f` *path/*`MultiLogReg.dml` `-nvargs` `X=`*path/file* `Y=`*path/file* `B=`*path/file* `Log=`*path/file* `icpt=`*int* `reg=`*double* `tol=`*double* `moi=`*int* `mii=`*int* `fmt=`*format*

Arguments

`X:` Location (on HDFS) to read the input matrix of feature vectors; each row constitutes one feature vector.

`Y:` Location to read the input one-column matrix of category labels that correspond to feature vectors in `X`. Note the following:
– Each non-baseline category label must be a positive integer.
– If all labels are positive, the largest represents the baseline category.
– If non-positive labels such as -1 or 0 are present, then they represent the (same) baseline category and are converted to label $\max(\texttt{Y}) + 1$.

`B:` Location to store the matrix of estimated regression parameters (the $\beta_{j,l}$'s), with the intercept parameters $\beta_{0,l}$ at position `B[`$m+1$, l`]` if available. The size of `B` is $(m+1) \times (k-1)$ with the intercepts or $m \times (k-1)$ without the intercepts, one column per non-baseline category and one row per feature.

`Log:` (default: `" "`) Location to store iteration-specific variables for monitoring and debugging purposes, see Table 5 for details.

`icpt:` (default: `0`) Intercept and shifting/rescaling of the features in X:
`0` = no intercept (hence no β_0), no shifting/rescaling of the features;
`1` = add intercept, but do not shift/rescale the features in X;
`2` = add intercept, shift/rescale the features in X to mean 0, variance 1

`reg:` (default: `0.0`) L2-regularization parameter (lambda)

`tol:` (default: `0.000001`) Tolerance (epsilon) used in the convergence criterion

`moi:` (default: `100`) Maximum number of outer (Fisher scoring) iterations

`mii:` (default: `0`) Maximum number of inner (conjugate gradient) iterations, or 0 if no maximum limit provided

`fmt:` (default: `"text"`) Matrix file output format, such as `text`, `mm`, or `csv`; see read/write functions in SystemML Language Reference for details.

Details

We estimate the logistic regression parameters via L2-regularized negative log-likelihood minimization (5). The optimization method used in the script closely follows the trust region Newton method for logistic regression described in [6]. For convenience, let us make some changes in notation:

- Convert the input vector of observed category labels into an indicator matrix Y of size $n \times k$ such that $Y_{i,l} = 1$ if the i-th category label is l and $Y_{i,l} = 0$ otherwise;

Name	Meaning
LINEAR_TERM_MIN	The minimum value of X %*% B, used to check for overflows
LINEAR_TERM_MAX	The maximum value of X %*% B, used to check for overflows
NUM_CG_ITERS	Number of inner (Conj. Gradient) iterations in this outer iteration
IS_TRUST_REACHED	1 = trust region boundary was reached, 0 = otherwise
POINT_STEP_NORM	L2-norm of iteration step from old point (matrix B) to new point
OBJECTIVE	The loss function we minimize (negative regularized log-likelihood)
OBJ_DROP_REAL	Reduction in the objective during this iteration, actual value
OBJ_DROP_PRED	Reduction in the objective predicted by a quadratic approximation
OBJ_DROP_RATIO	Actual-to-predicted reduction ratio, used to update the trust region
IS_POINT_UPDATED	1 = new point accepted; 0 = new point rejected, old point restored
GRADIENT_NORM	L2-norm of the loss function gradient (omitted if point is rejected)
TRUST_DELTA	Updated trust region size, the "delta"

Table 5: The Log file for multinomial logistic regression contains the above per-iteration variables in CSV format, each line containing triple (Name, Iteration#, Value) with Iteration# being 0 for initial values.

- Append an extra column of all ones, i.e. $(1, 1, \ldots, 1)^T$, as the $m+1$-st column to the feature matrix X to represent the intercept;
- Append an all-zero column as the k-th column to B, the matrix of regression parameters, to represent the baseline category;
- Convert the regularization constant λ into matrix Λ of the same size as B, placing 0's into the $m+1$-st row to disable intercept regularization, and placing λ's everywhere else.

Now the $(n \times k)$-matrix of predicted probabilities given by (3) and (4) and the objective function f in (5) have the matrix form

$$\begin{aligned} P &= \exp(XB) \;/\; \big(\exp(XB)\, 1_{k\times k}\big) \\ f &= -\sum Y \cdot (XB) + \sum \log\big(\exp(XB)\, 1_{k\times 1}\big) + (1/2) \sum \Lambda \cdot B \cdot B \end{aligned}$$

where operations $\cdot$, /, exp, and log are applied cellwise, and $\sum$ denotes the sum of all cells in a matrix. The gradient of f with respect to B can be represented as a matrix too:

$$\nabla f \;=\; X^T(P - Y) + \Lambda \cdot B$$

The Hessian $\mathcal{H}$ of f is a tensor, but, fortunately, the conjugate gradient inner loop of the trust region algorithm in [6] does not need to instantiate it. We only need to multiply $\mathcal{H}$ by ordinary matrices of the same size as B and ∇f, and this can be done in matrix form:

$$\mathcal{H}V \;=\; X^T\big(Q \,-\, P \cdot (Q\, 1_{k\times k})\big) \,+\, \Lambda \cdot V, \quad \text{where} \;\; Q \,=\, P \cdot (XV)$$

At each Newton iteration (the *outer* iteration) the minimization algorithm approximates the difference $\Delta f(S; B) = f(B + S; X, Y) - f(B; X, Y)$ attained in the objective function after a step $B \mapsto B + S$ by a second-degree formula

$$\Delta f(S; B) \;\approx\; (1/2) \sum S \cdot \mathcal{H}S \,+\, \sum S \cdot \nabla f$$

This approximation is then minimized by trust-region conjugate gradient iterations (the *inner* iterations) subject to the constraint $\|S\|_2 \leq \delta$. The trust region size δ is initialized as $0.5\sqrt{m} \,/\, \max_i \|x_i\|_2$ and updated as described in [6]. Users can specify the maximum number of the outer and the inner iterations with input parameters `moi` and `mii`, respectively. The iterative minimizer terminates successfully if $\|\nabla f\|_2 < \varepsilon \, \|\nabla f_{B=0}\|_2$, where $\varepsilon > 0$ is a tolerance supplied by the user via input parameter `tol`.

Returns

The estimated regression parameters (the $\hat{\beta}_{j,l}$) are populated into a matrix and written to an HDFS file whose path/name was provided as the "`B`" input argument. Only the non-baseline categories ($1 \leq l \leq k-1$) have their $\hat{\beta}_{j,l}$ in the output; to add the baseline category, just append a column of zeros. If `icpt=0` in the input command line, no intercepts are used and `B` has size $m \times (k-1)$; otherwise `B` has size $(m+1) \times (k-1)$ and the intercepts are in the $m+1$-st row. If `icpt=2`, then initially the feature columns in X are shifted to mean = 0 and rescaled to variance = 1. After the iterations converge, the $\hat{\beta}_{j,l}$'s are rescaled and shifted to work with the original features.

Examples

```
hadoop jar SystemML.jar -f MultiLogReg.dml -nvargs
   X=/user/biadmin/X.mtx Y=/user/biadmin/Y.mtx
   B=/user/biadmin/B.mtx fmt=csv icpt=2 reg=1.0 tol=0.0001
   moi=100 mii=10 Log=/user/biadmin/log.csv
```

References

- A. Agresti. *Categorical Data Analysis*. Wiley Series in Probability and Statistics. Wiley-Interscience, second edition, 2002.

2.2 Support Vector Machines

2.2.1 Binary-class Support Vector Machines

Description

Support Vector Machines are used to model the relationship between a categorical dependent variable y and one or more explanatory variables denoted X. This implementation learns (and predicts with) a binary class support vector machine (y with domain size 2).

Usage

```
-f path/l2-svm.dml -nvargs X=path/file Y=path/file icpt=int tol=double
                           reg=double maxiter=int model=path/file
                           Log=path/file fmt=csv|text
```

```
-f path/l2-svm-predict.dml -nvargs X=path/file Y=path/file icpt=int model=path/file
                                   scores=path/file accuracy=path/file
                                   confusion=path/file fmt=csv|text
```

Arguments

- X: Location (on HDFS) to read the matrix of feature vectors; each row constitutes one feature vector.
- Y: Location to read the one-column matrix of (categorical) labels that correspond to feature vectors in X. Binary class labels can be expressed in one of two choices: ± 1 or $1/2$. Note that, this argument is optional for prediction.
- icpt (default: `0`): If set to 1 then a constant bias column is added to X.
- tol (default: `0.001`): Procedure terminates early if the reduction in objective function value is less than tolerance times the initial objective function value.
- reg (default: `1`): Regularization constant. See details to find out where lambda appears in the objective function. If one were interested in drawing an analogy with the C parameter in C-SVM, then C = 2/lambda. Usually, cross validation is employed to determine the optimum value of lambda.
- maxiter (default: `100`): The maximum number of iterations.
- model: Location (on HDFS) that contains the learnt weights.
- Log: Location (on HDFS) to collect various metrics (e.g., objective function value etc.) that depict progress across iterations while training.
- fmt (default: `text`): Specifies the output format. Choice of comma-separated values (csv) or as a sparse-matrix (text).
- scores: Location (on HDFS) to store scores for a held-out test set. Note that, this is an optional argument.
- accuracy: Location (on HDFS) to store the accuracy computed on a held-out test set. Note that, this is an optional argument.
- confusion: Location (on HDFS) to store the confusion matrix computed using a held-out test set. Note that, this is an optional argument.

Details

Support vector machines learn a classification function by solving the following optimization problem (L_2-SVM):

$$\begin{aligned} \text{argmin}_w \quad & \frac{\lambda}{2}||w||_2^2 + \sum_i \xi_i^2 \\ \text{subject to:} \quad & y_i w^\top x_i \geq 1 - \xi_i \ \forall i \end{aligned}$$

where x_i is an example from the training set with its label given by y_i, w is the vector of parameters and λ is the regularization constant specified by the user.

To account for the missing bias term, one may augment the data with a column of constants which is achieved by setting intercept argument to 1 (C-J Hsieh et al, 2008).

This implementation optimizes the primal directly (Chapelle, 2007). It uses nonlinear conjugate gradient descent to minimize the objective function coupled with choosing step-sizes by performing one-dimensional Newton minimization in the direction of the gradient.

Returns

The learnt weights produced by l2-svm.dml are populated into a single column matrix and written to file on HDFS (see model in section Arguments). The number of rows in this matrix is ncol(X) if intercept was set to 0 during invocation and ncol(X) + 1 otherwise. The bias term, if used, is placed in the last row. Depending on what arguments are provided during invocation, l2-svm-predict.dml may compute one or more of scores, accuracy and confusion matrix in the output format specified.

Examples

```
hadoop jar SystemML.jar -f l2-svm.dml -nvargs X=/user/biadmin/X.mtx
                                              Y=/user/biadmin/y.mtx
                                              icpt=0 tol=0.001 fmt=csv
                                              reg=1.0 maxiter=100
                                              model=/user/biadmin/weights.csv
                                              Log=/user/biadmin/Log.csv

hadoop jar SystemML.jar -f l2-svm-predict.dml -nvargs X=/user/biadmin/X.mtx
                                                      Y=/user/biadmin/y.mtx
                                                      icpt=0 fmt=csv
                                                      model=/user/biadmin/weights.csv
                                                      scores=/user/biadmin/scores.csv
                                                      accuracy=/user/biadmin/accuracy.csv
                                                      confusion=/user/biadmin/confusion.csv
```

References

- W. T. Vetterling and B. P. Flannery. *Conjugate Gradient Methods in Multidimensions in Numerical Recipes in C - The Art in Scientific Computing.* W. H. Press and S. A. Teukolsky (eds.), Cambridge University Press, 1992.
- J. Nocedal and S. J. Wright. Numerical Optimization, Springer-Verlag, 1999.
- C-J Hsieh, K-W Chang, C-J Lin, S. S. Keerthi and S. Sundararajan. *A Dual Coordinate Descent Method for Large-scale Linear SVM.* International Conference of Machine Learning (ICML), 2008.

- Olivier Chapelle. *Training a Support Vector Machine in the Primal*. Neural Computation, 2007.

2.2.2 Multi-class Support Vector Machines

Description

Support Vector Machines are used to model the relationship between a categorical dependent variable y and one or more explanatory variables denoted X. This implementation supports dependent variables that have domain size greater or equal to 2 and hence is not restricted to binary class labels.

Usage

```
-f path/m-svm.dml -nvargs X=path/file Y=path/file icpt=int classes=int
                          tol=double reg=double maxiter=int model=path/file
                          Log=path/file fmt=csv|text
```

```
-f path/m-svm-predict.dml -nvargs X=path/file Y=path/file icpt=int model=path/file
                                  scores=path/file accuracy=path/file
                                  confusion=path/file fmt=csv|text
```

Arguments

- X: Location (on HDFS) containing the explanatory variables in a matrix. Each row constitutes an example.
- Y: Location (on HDFS) containing a 1-column matrix specifying the categorical dependent variable (label). Labels are assumed to be contiguously numbered from 1 ... #classes. Note that, this argument is optional for prediction.
- icpt (default: `0`): If set to 1 then a constant bias column is added to X.
- classes: Number of classes in the data.
- tol (default: `0.001`): Procedure terminates early if the reduction in objective function value is less than tolerance times the initial objective function value.
- reg (default: `1`): Regularization constant. See details to find out where lambda appears in the objective function. If one were interested in drawing an analogy with C-SVM, then C = 2/lambda. Usually, cross validation is employed to determine the optimum value of lambda.
- maxiter (default: `100`): The maximum number of iterations.
- model: Location (on HDFS) that contains the learnt weights.
- Log: Location (on HDFS) to collect various metrics (e.g., objective function value etc.) that depict progress across iterations while training.

- fmt (default: `text`): Specifies the output format. Choice of comma-separated values (csv) or as a sparse-matrix (text).
- scores: Location (on HDFS) to store scores for a held-out test set. Note that, this is an optional argument.
- accuracy: Location (on HDFS) to store the accuracy computed on a held-out test set. Note that, this is an optional argument.
- confusion: Location (on HDFS) to store the confusion matrix computed using a held-out test set. Note that, this is an optional argument.

Details

Support vector machines learn a classification function by solving the following optimization problem (L_2-SVM):

$$\begin{aligned} \text{argmin}_w \quad & \frac{\lambda}{2}||w||_2^2 + \sum_i \xi_i^2 \\ \text{subject to:} \quad & y_i w^\top x_i \geq 1 - \xi_i \ \forall i \end{aligned}$$

where x_i is an example from the training set with its label given by y_i, w is the vector of parameters and λ is the regularization constant specified by the user.

To extend the above formulation (binary class SVM) to the multiclass setting, one standard approache is to learn one binary class SVM per class that separates data belonging to that class from the rest of the training data (one-against-the-rest SVM, see C. Scholkopf, 1995).

To account for the missing bias term, one may augment the data with a column of constants which is achieved by setting intercept argument to 1 (C-J Hsieh et al, 2008).

This implementation optimizes the primal directly (Chapelle, 2007). It uses nonlinear conjugate gradient descent to minimize the objective function coupled with choosing step-sizes by performing one-dimensional Newton minimization in the direction of the gradient.

Returns

The learnt weights produced by m-svm.dml are populated into a matrix that has as many columns as there are classes in the training data, and written to file provided on HDFS (see model in section Arguments). The number of rows in this matrix is ncol(X) if intercept was set to 0 during invocation and ncol(X) + 1 otherwise. The bias terms, if used, are placed in the last row. Depending on what arguments are provided during invocation, m-svm-predict.dml may compute one or more of scores, accuracy and confusion matrix in the output format specified.

Examples

```
hadoop jar SystemML.jar -f m-svm.dml -nvargs X=/user/biadmin/X.mtx
```

```
                                                Y=/user/biadmin/y.mtx
                                                icpt=0 classes=10 tol=0.001
                                                reg=1.0 maxiter=100 fmt=csv
                                                model=/user/biadmin/weights.csv
                                                Log=/user/biadmin/Log.csv

hadoop jar SystemML.jar -f m-svm-predict.dml -nvargs X=/user/biadmin/X.mtx
                                                     Y=/user/biadmin/y.mtx
                                                     icpt=0 fmt=csv
                                                     model=/user/biadmin/weights.csv
                                                     scores=/user/biadmin/scores.csv
                                                     accuracy=/user/biadmin/accuracy.csv
                                                     confusion=/user/biadmin/confusion.csv
```

References

- W. T. Vetterling and B. P. Flannery. *Conjugate Gradient Methods in Multidimensions in Numerical Recipes in C - The Art in Scientific Computing.* W. H. Press and S. A. Teukolsky (eds.), Cambridge University Press, 1992.
- J. Nocedal and S. J. Wright. *Numerical Optimization.* Springer-Verlag, 1999.
- C-J Hsieh, K-W Chang, C-J Lin, S. S. Keerthi and S. Sundararajan. *A Dual Coordinate Descent Method for Large-scale Linear SVM.* International Conference of Machine Learning (ICML), 2008.
- Olivier Chapelle. *Training a Support Vector Machine in the Primal.* Neural Computation, 2007.
- B. Scholkopf, C. Burges and V. Vapnik. *Extracting Support Data for a Given Task.* International Conference on Knowledge Discovery and Data Mining (ICDM), 1995.

2.3 Naive Bayes

Description

Naive Bayes is very simple generative model used for classifying data. This implementation learns a multinomial naive Bayes classifier which is applicable when all features are counts of categorical values.

Usage

```
-f path/naive-bayes.dml -nvargs X=path/file Y=path/file classes=int laplace=double
                               prior=path/file conditionals=path/file
                               accuracy=path/file fmt=csv|text
```

`-f` *path*`/naive-bayes-predict.dml -nvargs` `X=`*path*`/`*file* `Y=`*path*`/`*file* `prior=`*path*`/`*file*
`conditionals=`*path*`/`*file* `fmt=`*csv*|*text*
`accuracy=`*path*`/`*file* `confusion=`*path*`/`*file*
`probabilities=`*path*`/`*file*

Arguments

- X: Location (on HDFS) to read the matrix of feature vectors; each row constitutes one feature vector.
- Y: Location (on HDFS) to read the one-column matrix of (categorical) labels that correspond to feature vectors in X. Classes are assumed to be contiguously labeled beginning from 1. Note that, this argument is optional for prediction.
- classes: Number of classes in the data.
- laplace (default: `1`): Laplace smoothing specified by the user to avoid creation of 0 probabilities.
- prior: Location (on HDFS) that contains the class prior probabilites.
- conditionals: Location (on HDFS) that contains the class conditional feature distributions.
- fmt (default: `text`): Specifies the output format. Choice of comma-separated values (csv) or as a sparse-matrix (text).
- probabilities: Location (on HDFS) to store class membership probabilities for a held-out test set. Note that, this is an optional argument.
- accuracy: Location (on HDFS) to store the training accuracy during learning and testing accuracy from a held-out test set during prediction. Note that, this is an optional argument for prediction.
- confusion: Location (on HDFS) to store the confusion matrix computed using a held-out test set. Note that, this is an optional argument.

Details

Naive Bayes is a very simple generative classification model. It posits that given the class label, features can be generated independently of each other. More precisely, the (multinomial) naive Bayes model uses the following equation to estimate the joint probability of a feature vector x belonging to class y:

$$\text{Prob}(y, x) = \pi_y \prod_{i \in x} \theta_{iy}^{n(i,x)}$$

where π_y denotes the prior probability of class y, i denotes a feature present in x with $n(i, x)$ denoting its count and θ_{iy} denotes the class conditional probability

of feature i in class y. The usual constraints hold on π and θ:

$$\pi_y \geq 0,\ \sum_{y \in \mathcal{C}} \pi_y = 1$$
$$\forall y \in \mathcal{C}: \qquad \theta_{iy} \geq 0,\ \sum_i \theta_{iy} = 1$$

where $\mathcal{C}$ is the set of classes.

Given a fully labeled training dataset, it is possible to learn a naive Bayes model using simple counting (group-by aggregates). To compute the class conditional probabilities, it is usually advisable to avoid setting θ_{iy} to 0. One way to achieve this is using additive smoothing or Laplace smoothing. Some authors have argued that this should in fact be add-one smoothing. This implementation uses add-one smoothing by default but lets the user specify her/his own constant, if required.

This implementation is sometimes referred to as *multinomial* naive Bayes. Other flavours of naive Bayes are also popular.

Returns

The learnt model produced by naive-bayes.dml is stored in two separate files. The first file stores the class prior (a single-column matrix). The second file stores the class conditional probabilities organized into a matrix with as many rows as there are class labels and as many columns as there are features. Depending on what arguments are provided during invocation, naive-bayes-predict.dml may compute one or more of probabilities, accuracy and confusion matrix in the output format specified.

Examples

```
hadoop jar SystemML.jar -f naive-bayes.dml -nvargs
                           X=/user/biadmin/X.mtx
                           Y=/user/biadmin/y.mtx
                           classes=5 laplace=1 fmt=csv
                           prior=/user/biadmin/prior.csv
                           conditionals=/user/biadmin/conditionals.csv
                           accuracy=/user/biadmin/accuracy.csv

hadoop jar SystemML.jar -f naive-bayes-predict.dml -nvargs
                           X=/user/biadmin/X.mtx
                           Y=/user/biadmin/y.mtx
                           prior=/user/biadmin/prior.csv
                           conditionals=/user/biadmin/conditionals.csv
                           fmt=csv
                           accuracy=/user/biadmin/accuracy.csv
                           probabilities=/user/biadmin/probabilities.csv
                           confusion=/user/biadmin/confusion.csv
```

References

- S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall, 2009.
- A. McCallum and K. Nigam. *A comparison of event models for naive bayes text classification.* AAAI-98 workshop on learning for text categorization, 1998.

3 Clustering

3.1 K-Means Clustering

Description

Given a collection of n records with a pairwise similarity measure, the goal of clustering is to assign a category label to each record so that similar records tend to get the same label. In contrast to multinomial logistic regression, clustering is an *unsupervised* learning problem with neither category assignments nor label interpretations given in advance. In k-means clustering, the records $x_1, x_2, \ldots, x_n$ are numerical feature vectors of $\dim x_i = m$ with the squared Euclidean distance $\|x_i - x_{i'}\|_2^2$ as the similarity measure. We want to partition $\{x_1, \ldots, x_n\}$ into k clusters $\{S_1, \ldots, S_k\}$ so that the aggregated squared distance from records to their cluster means is minimized:

$$\text{WCSS} \;=\; \sum_{i=1}^{n} \big\|x_i - \text{mean}(S_j : x_i \in S_j)\big\|_2^2 \;\to\; \min \tag{6}$$

The aggregated distance measure in (6) is called the *within-cluster sum of squares* (WCSS). It can be viewed as a measure of residual variance that remains in the data after the clustering assignment, conceptually similar to the residual sum of squares (RSS) in linear regression. However, unlike for the RSS, the minimization of (6) is an NP-hard problem [2].

Rather than searching for the global optimum in (6), a heuristic algorithm called Lloyd's algorithm is typically used. This iterative algorithm maintains and updates a set of k *centroids* $\{c_1, \ldots, c_k\}$, one centroid per cluster. It defines each cluster S_j as the set of all records closer to c_j than to any other centroid. Each iteration of the algorithm reduces the WCSS in two steps:

1. Assign each record to the closest centroid, making $\text{mean}(S_j) \neq c_j$;
2. Reset each centroid to its cluster's mean: $c_j := \text{mean}(S_j)$.

After Step 1 the centroids are generally different from the cluster means, so we can compute another "within-cluster sum of squares" based on the centroids:

$$\text{WCSS\_C} \;=\; \sum_{i=1}^{n} \big\|x_i - \text{centroid}(S_j : x_i \in S_j)\big\|_2^2 \tag{7}$$

This WCSS_C after Step 1 is less than the means-based WCSS before Step 1 (or equal if convergence achieved), and in Step 2 the WCSS cannot exceed the WCSS_C for *the same* clustering; hence the WCSS reduction.

Exact convergence is reached when each record becomes closer to its cluster's mean than to any other cluster's mean, so there are no more re-assignments and the centroids coincide with the means. In practice, iterations may be stopped when the reduction in WCSS (or in WCSS_C) falls below a minimum threshold, or upon reaching the maximum number of iterations. The initialization of the centroids is also an important part of the algorithm. The smallest WCSS obtained by the algorithm is not the global minimum and varies depending on the initial centroids. We implement multiple parallel runs with different initial centroids and report the best result.

Scoring Our scoring script evaluates the clustering output by comparing it with a known category assignment. Since cluster labels have no prior correspondence to the categories, we cannot count "correct" and "wrong" cluster assignments. Instead, we quantify them in two ways:

1. Count how many same-category and different-category pairs of records end up in the same cluster or in different clusters;
2. For each category, count the prevalence of its most common cluster; for each cluster, count the prevalence of its most common category.

The number of categories and the number of clusters (k) do not have to be equal. A same-category pair of records clustered into the same cluster is viewed as a "true positive," a different-category pair clustered together is a "false positive," a same-category pair clustered apart is a "false negative" etc.

Usage: K-means Script

`-f` *path/*`Kmeans.dml` `-nvargs` `X=`*path/file* `C=`*path/file* `k=`*int* `runs=`*int* `maxi=`*int* `tol=`*double* `samp=`*int* `isY=`*int* `Y=`*path/file* `fmt=`*format* `verb=`*int*

Usage: K-means Scoring/Prediction

`-f` *path/*`Kmeans-predict.dml` `-nvargs` `X=`*path/file* `C=`*path/file* `spY=`*path/file* `prY=`*path/file* `fmt=`*format* `O=`*path/file*

Arguments

`X:` Location to read matrix X with the input data records as rows

`C:` (default: `"C.mtx"`) Location to store the output matrix with the best available cluster centroids as rows

`k:` Number of clusters (and centroids)

`runs:` (default: `10`) Number of parallel runs, each run with different initial centroids

`maxi:` (default: `1000`) Maximum number of iterations per run

`tol:` (default: `0.000001`) Tolerance (epsilon) for single-iteration WCSS_C change ratio

`samp:` (default: `50`) Average number of records per centroid in data samples used in the centroid initialization procedure

`Y:` (default: `"Y.mtx"`) Location to store the one-column matrix Y with the best available mapping of records to clusters (defined by the output centroids)

`isY:` (default: `0`) `0` = do not write matrix Y, `1` = write Y

`fmt:` (default: `"text"`) Matrix file output format, such as `text`, `mm`, or `csv`; see read/write functions in SystemML Language Reference for details.

`verb:` (default: `0`) `0` = do not print per-iteration statistics for each run, `1` = print them (the "verbose" option)

Arguments — Scoring/Prediction

`X:` (default: `" "`) Location to read matrix X with the input data records as rows, optional when `prY` input is provided

`C:` (default: `" "`) Location to read matrix C with cluster centroids as rows, optional when `prY` input is provided; NOTE: if both `X` and `C` are provided, `prY` is an output, not input

`spY:` (default: `" "`) Location to read a one-column matrix with the externally specified "true" assignment of records (rows) to categories, optional for prediction without scoring

`prY:` (default: `" "`) Location to read (or write, if `X` and `C` are present) a column-vector with the predicted assignment of rows to clusters; NOTE: No prior correspondence is assumed between the predicted cluster labels and the externally specified categories

`fmt:` (default: `"text"`) Matrix file output format for `prY`, such as `text`, `mm`, or `csv`; see read/write functions in SystemML Language Reference for details

`O:` (default: `" "`) Location to write the output statistics defined in Table 6, by default print them to the standard output

Details

Our clustering script proceeds in 3 stages: centroid initialization, parallel k-means iterations, and the best-available output generation. Centroids are initialized at random from the input records (the rows of X), biased towards being chosen far apart from each other. The initialization method is based on the `k-means++` heuristic from [3], with one important difference: to reduce the number of passes through X, we take a small sample of X and run the `k-means++` heuristic over this sample. Here is, conceptually, our centroid initialization algorithm for one clustering run:

1. Sample the rows of X uniformly at random, picking each row with probability $p = ks/n$ where
   - k is the number of centroids,
   - n is the number of records, and
   - s is the `samp` input parameter.

Name	CID	Meaning
TSS		Total Sum of Squares (from the total mean)
WCSS_M		Within-Cluster Sum of Squares (means as centers)
WCSS_M_PC		Within-Cluster Sum of Squares (means), in % of TSS
BCSS_M		Between-Cluster Sum of Squares (means as centers)
BCSS_M_PC		Between-Cluster Sum of Squares (means), in % of TSS
WCSS_C		Within-Cluster Sum of Squares (centroids as centers)
WCSS_C_PC		Within-Cluster Sum of Squares (centroids), % of TSS
BCSS_C		Between-Cluster Sum of Squares (centroids as centers)
BCSS_C_PC		Between-Cluster Sum of Squares (centroids), % of TSS
TRUE_SAME_CT		Same-category pairs predicted as Same-cluster, count
TRUE_SAME_PC		Same-category pairs predicted as Same-cluster, %
TRUE_DIFF_CT		Diff-category pairs predicted as Diff-cluster, count
TRUE_DIFF_PC		Diff-category pairs predicted as Diff-cluster, %
FALSE_SAME_CT		Diff-category pairs predicted as Same-cluster, count
FALSE_SAME_PC		Diff-category pairs predicted as Same-cluster, %
FALSE_DIFF_CT		Same-category pairs predicted as Diff-cluster, count
FALSE_DIFF_PC		Same-category pairs predicted as Diff-cluster, %
SPEC_TO_PRED	+	For specified category, the best predicted cluster id
SPEC_FULL_CT	+	For specified category, its full count
SPEC_MATCH_CT	+	For specified category, best-cluster matching count
SPEC_MATCH_PC	+	For specified category, % of matching to full count
PRED_TO_SPEC	+	For predicted cluster, the best specified category id
PRED_FULL_CT	+	For predicted cluster, its full count
PRED_MATCH_CT	+	For predicted cluster, best-category matching count
PRED_MATCH_PC	+	For predicted cluster, % of matching to full count

Table 6: The O-file for Kmeans-predict provides the output statistics in CSV format, one per line, in the following format: (NAME, [CID], VALUE). Note: the 1st group statistics are given if X input is available; the 2nd group statistics are given if X and C inputs are available; the 3rd and 4th group statistics are given if spY input is available; only the 4th group statistics contain a nonempty CID value; when present, CID contains either the specified category label or the predicted cluster label.

If $ks \geq n$, the entire X is used in place of its sample.

2. Choose the first centroid uniformly at random from the sampled rows.
3. Choose each subsequent centroid from the sampled rows, at random, with probability proportional to the squared Euclidean distance between the row and the nearest already-chosen centroid.

The sampling of X and the selection of centroids are performed independently and in parallel for each run of the k-means algorithm. When we sample the rows of X, rather than tossing a random coin for each row, we compute the number of rows to skip until the next sampled row as $\lceil \log(u)/\log(1-p) \rceil$ where $u \in (0,1)$ is uniformly random. This time-saving trick works because

$$\text{Prob}[k-1 < \log_{1-p}(u) < k] \;=\; p(1-p)^{k-1} \;=\; \text{Prob}[\text{skip } k-1 \text{ rows}]$$

However, it requires us to estimate the maximum sample size, which we set near $ks + 10\sqrt{ks}$ to make it generous enough.

Once we selected the initial centroid sets, we start the k-means iterations independently in parallel for all clustering runs. The number of clustering runs is given as the `runs` input parameter. Each iteration of each clustering run performs the following steps:

- Compute the centroid-dependent part of squared Euclidean distances from all records (rows of X) to each of the k centroids using matrix product;
- Take the minimum of the above for each record;
- Update the current within-cluster sum of squares (WCSS) value, with centroids substituted instead of the means for efficiency;
- Check the convergence criterion: $\text{WCSS}_{\text{old}} - \text{WCSS}_{\text{new}} < \varepsilon \cdot \text{WCSS}_{\text{new}}$ as well as the number of iterations limit;
- Find the closest centroid for each record, sharing equally any records with multiple closest centroids;
- Compute the number of records closest to each centroid, checking for "runaway" centroids with no records left (in which case the run fails);
- Compute the new centroids by averaging the records in their clusters.

When a termination condition is satisfied, we store the centroids and the WCSS value and exit this run. A run has to satisfy the WCSS convergence criterion to be considered successful. Upon the termination of all runs, we select the smallest WCSS value among the successful runs, and write out this run's centroids. If requested, we also compute the cluster assignment of all records in X, using integers from 1 to k as the cluster labels. The scoring script can then be used to compare the cluster assignment with an externally specified category assignment.

Returns

We output the k centroids for the best available clustering, i. e. whose WCSS is the smallest of all successful runs. The centroids are written as the rows of the $k \times m$-matrix into the output file whose path/name was provided as the "`C`" input argument. If the input parameter "`isY`" was set to `1`, we also output the one-column matrix with the cluster assignment for all the records. This assignment is written into the file whose path/name was provided as the "`Y`" input argument. The best WCSS value, as well as some information about the performance of the other runs, is printed during the script execution. The scoring script `Kmeans-predict` prints all its results in a self-explanatory manner, as defined in Table 6.

Examples

```
hadoop jar SystemML.jar -f Kmeans.dml -nvargs
   X=/user/biadmin/X.mtx k=5 C=/user/biadmin/centroids.mtx
   fmt=csv
hadoop jar SystemML.jar -f Kmeans.dml -nvargs
   X=/user/biadmin/X.mtx k=5 runs=100 maxi=5000
```

```
   tol=0.00000001 samp=20 C=/user/biadmin/centroids.mtx isY=1
   Y=/user/biadmin/Yout.mtx verb=1
```

To predict `Y` given `X` and `C`:

```
hadoop jar SystemML.jar -f Kmeans-predict.dml -nvargs
   X=/user/biadmin/X.mtx C=/user/biadmin/C.mtx
   prY=/user/biadmin/PredY.mtx O=/user/biadmin/stats.csv
```

To compare "actual" labels `spY` with "predicted" labels given `X` and `C`:

```
hadoop jar SystemML.jar -f Kmeans-predict.dml -nvargs
   X=/user/biadmin/X.mtx C=/user/biadmin/C.mtx
   spY=/user/biadmin/Y.mtx O=/user/biadmin/stats.csv
```

To compare "actual" labels `spY` with given "predicted" labels `prY`:

```
hadoop jar SystemML.jar -f Kmeans-predict.dml -nvargs
   spY=/user/biadmin/Y.mtx prY=/user/biadmin/PredY.mtx
   O=/user/biadmin/stats.csv
```

References

- D. Aloise, A. Deshpande, P. Hansen, and P. Popat. NP-hardness of Euclidean sum-of-squares clustering. *Machine Learning*, 75(2):245–248, May 2009.
- D. Arthur and S. Vassilvitskii. `k-means++`: The advantages of careful seeding. In *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2007)*, pages 1027–1035, New Orleans LA, USA, January 7–9 2007.

4 Regression

4.1 Linear Regression

Description

Linear Regression scripts are used to model the relationship between one numerical response variable and one or more explanatory (feature) variables. The scripts are given a dataset $(X, Y) = (x_i, y_i)_{i=1}^n$ where x_i is a numerical vector of feature variables and y_i is a numerical response value for each training data record. The feature vectors are provided as a matrix X of size $n \times m$, where n is the number of records and m is the number of features. The observed response values are provided as a 1-column matrix Y, with a numerical value y_i for each x_i in the corresponding row of matrix X.

In linear regression, we predict the distribution of the response y_i based on a fixed linear combination of the features in x_i. We assume that there exist constant regression coefficients $\beta_0, \beta_1, \ldots, \beta_m$ and a constant residual variance σ^2 such that

$$y_i \sim \text{Normal}(\mu_i, \sigma^2) \text{ where } \mu_i = \beta_0 + \beta_1 x_{i,1} + \ldots + \beta_m x_{i,m} \tag{8}$$

Distribution $y_i \sim \text{Normal}(\mu_i, \sigma^2)$ models the "unexplained" residual noise and is assumed independent across different records.

The goal is to estimate the regression coefficients and the residual variance. Once they are accurately estimated, we can make predictions about y_i given x_i in new records. We can also use the β_j's to analyze the influence of individual features on the response value, and assess the quality of this model by comparing residual variance in the response, left after prediction, with its total variance.

There are two scripts in our library, both doing the same estimation, but using different computational methods. Depending on the size and the sparsity of the feature matrix X, one or the other script may be more efficient. The "direct solve" script `LinearRegDS` is more efficient when the number of features m is relatively small ($m \sim 1000$ or less) and matrix X is either tall or fairly dense (has $\gg m^2$ nonzeros); otherwise, the "conjugate gradient" script `LinearRegCG` is more efficient. If $m > 50000$, use only `LinearRegCG`.

Usage

`-f` *path/*`LinearRegDS.dml` `-nvargs` `X=`*path/file* `Y=`*path/file* `B=`*path/file* `O=`*path/file* `icpt=`*int* `reg=`*double* `fmt=`*format*

`-f` *path/*`LinearRegCG.dml` `-nvargs` `X=`*path/file* `Y=`*path/file* `B=`*path/file* `O=`*path/file* `Log=`*path/file* `icpt=`*int* `reg=`*double* `tol=`*double* `maxi=`*int* `fmt=`*format*

Arguments

`X:` Location (on HDFS) to read the matrix of feature vectors, each row constitutes one feature vector

`Y:` Location to read the 1-column matrix of response values

`B:` Location to store the estimated regression parameters (the β_j's), with the intercept parameter β_0 at position `B[`$m+1$`, 1]` if available

`O:` (default: `" "`) Location to store the CSV-file of summary statistics defined in Table 7, the default is to print it to the standard output

`Log:` (default: `" "`, `LinearRegCG` only) Location to store iteration-specific variables for monitoring and debugging purposes, see Table 8 for details.

`icpt:` (default: `0`) Intercept presence and shifting/rescaling the features in X:
`0` = no intercept (hence no β_0), no shifting or rescaling of the features;
`1` = add intercept, but do not shift/rescale the features in X;
`2` = add intercept, shift/rescale the features in X to mean 0, variance 1

`reg:` (default: `0.000001`) L2-regularization parameter $\lambda \geq 0$; set to nonzero for highly dependent, sparse, or numerous ($m \gtrsim n/10$) features

`tol:` (default: `0.000001`, `LinearRegCG` only) Tolerance $\varepsilon \geq 0$ used in the convergence criterion: we terminate conjugate gradient iterations when the β-residual reduces in L2-norm by this factor

`maxi:` (default: `0`, `LinearRegCG` only) Maximum number of conjugate gradient iterations, or 0 if no maximum limit provided

`fmt:` (default: `"text"`) Matrix file output format, such as `text`, `mm`, or `csv`; see read/write functions in SystemML Language Reference for details.

Name	Meaning
AVG_TOT_Y	Average of the response value Y
STDEV_TOT_Y	Standard Deviation of the response value Y
AVG_RES_Y	Average of the residual $Y - \text{pred}(Y\|X)$, i.e. residual bias
STDEV_RES_Y	Standard Deviation of the residual $Y - \text{pred}(Y\|X)$
DISPERSION	GLM-style dispersion, i.e. residual sum of squares / #deg. fr.
PLAIN_R2	Plain R^2 of residual with bias included vs. total average
ADJUSTED_R2	Adjusted R^2 of residual with bias included vs. total average
PLAIN_R2_NOBIAS	Plain R^2 of residual with bias subtracted vs. total average
ADJUSTED_R2_NOBIAS	Adjusted R^2 of residual with bias subtracted vs. total average
PLAIN_R2_VS_0	*Plain R^2 of residual with bias included vs. zero constant
ADJUSTED_R2_VS_0	*Adjusted R^2 of residual with bias included vs. zero constant

* The last two statistics are only printed if there is no intercept (icpt=0)

Table 7: Besides β, linear regression scripts compute a few summary statistics listed above. The statistics are provided in CSV format, one comma-separated name-value pair per each line.

Name	Meaning
CG_RESIDUAL_NORM	L2-norm of conjug. grad. residual, which is A %*% $\beta - t(X)$ %*% y where $A = t(X)$ %*% $X + \text{diag}(\lambda)$, or a similar quantity
CG_RESIDUAL_RATIO	Ratio of current L2-norm of conjug. grad. residual over the initial

Table 8: The Log file for LinearRegCG script contains the above per-iteration variables in CSV format, each line containing triple (Name, Iteration#, Value) with Iteration# being 0 for initial values.

Details

To solve a linear regression problem over feature matrix X and response vector Y, we can find coefficients $\beta_0, \beta_1, \ldots, \beta_m$ and σ^2 that maximize the joint likelihood of all y_i for $i = 1 \ldots n$, defined by the assumed statistical model (8). Since the joint likelihood of the independent $y_i \sim \text{Normal}(\mu_i, \sigma^2)$ is proportional to the product of $\exp\big(-(y_i - \mu_i)^2/(2\sigma^2)\big)$, we can take the logarithm of this product, then multiply by $-2\sigma^2 < 0$ to obtain a least squares problem:

$$\sum_{i=1}^{n} (y_i - \mu_i)^2 \;=\; \sum_{i=1}^{n} \Big(y_i - \beta_0 - \sum_{j=1}^{m} \beta_j x_{i,j}\Big)^2 \;\to\; \min \tag{9}$$

This may not be enough, however. The minimum may sometimes be attained over infinitely many β-vectors, for example if X has an all-0 column, or has linearly dependent columns, or has fewer rows than columns ($n < m$). Even if (9) has a unique solution, other β-vectors may be just a little suboptimal[1], yet give significantly different predictions for new feature vectors. This results in *overfitting*: prediction error for the training data (X and Y) is much smaller than for the test data (new records).

[1]Smaller likelihood difference between two models suggests less statistical evidence to pick one model over the other.

Overfitting and degeneracy in the data is commonly mitigated by adding a regularization penalty term to the least squares function:

$$\sum_{i=1}^{n}\Big(y_i - \beta_0 - \sum_{j=1}^{m}\beta_j x_{i,j}\Big)^2 \;+\; \lambda\sum_{j=1}^{m}\beta_j^2 \;\to\; \min \tag{10}$$

The choice of $\lambda > 0$, the regularization constant, typically involves cross-validation where the dataset is repeatedly split into a training part (to estimate the β_j's) and a test part (to evaluate prediction accuracy), with the goal of maximizing the test accuracy. In our scripts, λ is provided as input parameter `reg`.

The solution to least squares problem (10), through taking the derivative and setting it to 0, has the matrix linear equation form

$$A\begin{bmatrix}\beta_{1:m}\\ \beta_0\end{bmatrix} = \big[X,\,1\big]^T Y, \;\text{ where }\; A = \big[X,\,1\big]^T\big[X,\,1\big] + \operatorname{diag}(\underbrace{\lambda,\ldots,\lambda}_{m},0) \tag{11}$$

where $[X,\,1]$ is X with an extra column of 1s appended on the right, and the diagonal matrix of λ's has a zero to keep the intercept β_0 unregularized. If the intercept is disabled by setting `icpt=0`, the equation is simply $X^TX\beta = X^TY$.

We implemented two scripts for solving equation (11): one is a "direct solver" that computes A and then solves $A\beta = [X,\,1]^TY$ by calling an external package, the other performs linear conjugate gradient (CG) iterations without ever materializing A. The CG algorithm closely follows Algorithm 5.2 in Chapter 5 of [9]. Each step in the CG algorithm computes a matrix-vector multiplication $q = Ap$ by first computing $[X,\,1]\,p$ and then $[X,\,1]^T[X,\,1]\,p$. Usually the number of such multiplications, one per CG iteration, is much smaller than m. The user can put a hard bound on it with input parameter `maxi`, or use the default maximum of $m+1$ (or m if no intercept) by having `maxi=0`. The CG iterations terminate when the L2-norm of vector $r = A\beta - [X,\,1]^TY$ decreases from its initial value (for $\beta = 0$) by the tolerance factor specified in input parameter `tol`.

The CG algorithm is more efficient if computing $[X,\,1]^T\big([X,\,1]\,p\big)$ is much faster than materializing A, an $(m+1)\times(m+1)$ matrix. The Direct Solver (DS) is more efficient if X takes up a lot more memory than A (i.e. X has a lot more nonzeros than m^2) and if m^2 is small enough for the external solver ($m \lesssim 50000$). A more precise determination between CG and DS is subject to further research.

In addition to the β-vector, the scripts estimate the residual standard deviation σ and the R^2, the ratio of "explained" variance to the total variance of the response variable. These statistics only make sense if the number of degrees of freedom $n-m-1$ is positive and the regularization constant λ is negligible or zero. The formulas for σ and R^2 are:

$$R^2_{\text{plain}} = 1 - \frac{\text{RSS}}{\text{TSS}}, \quad \sigma \;=\; \sqrt{\frac{\text{RSS}}{n-m-1}}, \quad R^2_{\text{adj.}} = 1 - \frac{\sigma^2(n-1)}{\text{TSS}}$$

where

$$\text{RSS} \;=\; \sum_{i=1}^{n}\Big(y_i - \hat{\mu}_i - \frac{1}{n}\sum_{i'=1}^{n}(y_{i'} - \hat{\mu}_{i'})\Big)^2; \quad \text{TSS} \;=\; \sum_{i=1}^{n}\Big(y_i - \frac{1}{n}\sum_{i'=1}^{n}y_{i'}\Big)^2$$

Here $\hat{\mu}_i$ are the predicted means for y_i based on the estimated regression coefficients and the feature vectors. They may be biased when no intercept is present, hence the RSS formula subtracts the bias.

Lastly, note that by choosing the input option `icpt=2` the user can shift and rescale the columns of X to have zero average and the variance of 1. This is particularly important when using regularization over highly disbalanced features, because regularization tends to penalize small-variance columns (which need large β_j's) more than large-variance columns (with small β_j's). At the end, the estimated regression coefficients are shifted and rescaled to apply to the original features.

Returns

The estimated regression coefficients (the $\hat{\beta}_j$'s) are populated into a matrix and written to an HDFS file whose path/name was provided as the "`B`" input argument. What this matrix contains, and its size, depends on the input argument `icpt`, which specifies the user's intercept and rescaling choice:

`icpt=0:` No intercept, matrix B has size $m \times 1$, with $B[j, 1] = \hat{\beta}_j$ for each j from 1 to $m = \text{ncol}(X)$.

`icpt=1:` There is intercept, but no shifting/rescaling of X; matrix B has size $(m+1) \times 1$, with $B[j, 1] = \hat{\beta}_j$ for j from 1 to m, and $B[m+1, 1] = \hat{\beta}_0$, the estimated intercept coefficient.

`icpt=2:` There is intercept, and the features in X are shifted to mean = 0 and rescaled to variance = 1; then there are two versions of the $\hat{\beta}_j$'s, one for the original features and another for the shifted/rescaled features. Now matrix B has size $(m+1) \times 2$, with $B[\cdot, 1]$ for the original features and $B[\cdot, 2]$ for the shifted/rescaled features, in the above format. Note that $B[\cdot, 2]$ are iteratively estimated and $B[\cdot, 1]$ are obtained from $B[\cdot, 2]$ by complementary shifting and rescaling.

The estimated summary statistics, including residual standard deviation σ and the R^2, are printed out or sent into a file (if specified) in CSV format as defined in Table 7. For conjugate gradient iterations, a log file with monitoring variables can also be made available, see Table 8.

Examples

```
hadoop jar SystemML.jar -f LinearRegCG.dml -nvargs
   X=/user/biadmin/X.mtx Y=/user/biadmin/Y.mtx
   B=/user/biadmin/B.mtx fmt=csv O=/user/biadmin/stats.csv icpt=2
   reg=1.0 tol=0.00000001 maxi=100 Log=/user/biadmin/log.csv
hadoop jar SystemML.jar -f LinearRegDS.dml -nvargs
   X=/user/biadmin/X.mtx Y=/user/biadmin/Y.mtx
   B=/user/biadmin/B.mtx fmt=csv O=/user/biadmin/stats.csv icpt=2
   reg=1.0
```

4.2 Generalized Linear Models (GLM)

Description

Generalized Linear Models [5, 7, 8] extend the methodology of linear and logistic regression to a variety of distributions commonly assumed as noise effects in the response variable. As before, we are given a collection of records (x_1, y_1), ..., (x_n, y_n) where x_i is a numerical vector of explanatory (feature) variables of size $\dim x_i = m$, and y_i is the response (dependent) variable observed for this vector. GLMs assume that some linear combination of the features in x_i determines the *mean* μ_i of y_i, while the observed y_i is a random outcome of a noise distribution $\text{Prob}[y \mid \mu_i]$ [2] with that mean μ_i:

$$x_i \quad \mapsto \quad \eta_i = \beta_0 + \sum\nolimits_{j=1}^{m} \beta_j x_{i,j} \quad \mapsto \quad \mu_i \quad \mapsto \quad y_i \sim \text{Prob}[y \mid \mu_i]$$

In linear regression the response mean μ_i *equals* some linear combination over x_i, denoted above by η_i. In logistic regression with $y \in \{0, 1\}$ (Bernoulli) the mean of y is the same as $\text{Prob}[y = 1]$ and equals $1/(1 + e^{-\eta_i})$, the logistic function of η_i. In GLM, μ_i and η_i can be related via any given smooth monotone function called the *link function*: $\eta_i = g(\mu_i)$. The unknown linear combination parameters β_j are assumed to be the same for all records.

The goal of the regression is to estimate the parameters β_j from the observed data. Once the β_j's are accurately estimated, we can make predictions about y for a new feature vector x. To do so, compute η from x and use the inverted link function $\mu = g^{-1}(\eta)$ to compute the mean μ of y; then use the distribution $\text{Prob}[y \mid \mu]$ to make predictions about y. Both $g(\mu)$ and $\text{Prob}[y \mid \mu]$ are user-provided. Our GLM script supports a standard set of distributions and link functions, see below for details.

Usage

`-f` *path/*`GLM.dml` `-nvargs` `X=`*path/file* `Y=`*path/file* `B=`*path/file* `fmt=`*format* `O=`*path/file* `Log=`*path/file* `dfam=`*int* `vpow=`*double* `link=`*int* `lpow=`*double* `yneg=`*double* `icpt=`*int* `reg=`*double* `tol=`*double* `disp=`*double* `moi=`*int* `mii=`*int*

Arguments

`X`: Location (on HDFS) to read the matrix of feature vectors; each row constitutes an example.

`Y`: Location to read the response matrix, which may have 1 or 2 columns

`B`: Location to store the estimated regression parameters (the β_j's), with the intercept parameter β_0 at position `B[`$m+1$`, 1]` if available

`fmt`: (default: `"text"`) Matrix file output format, such as `text`, `mm`, or `csv`; see read/write functions in SystemML Language Reference for details.

`O`: (default: `" "`) Location to write certain summary statistics described in Table 9, by default it is standard output.

`Log`: (default: `" "`) Location to store iteration-specific variables for monitoring and debugging purposes, see Table 10 for details.

[2] $\text{Prob}[y \mid \mu_i]$ is given by a density function if y is continuous.

`dfam:` (default: `1`) Distribution family code to specify Prob[$y \mid \mu$], see Table 11:
`1` = power distributions with Var(y) = μ^α; `2` = binomial or Bernoulli

`vpow:` (default: `0.0`) When `dfam=1`, this provides the q in Var(y) = $a\mu^q$, the power dependence of the variance of y on its mean. In particular, use:
`0.0` = Gaussian, `1.0` = Poisson, `2.0` = Gamma, `3.0` = inverse Gaussian

`link:` (default: `0`) Link function code to determine the link function $\eta = g(\mu)$:
`0` = canonical link (depends on the distribution family), see Table 11;
`1` = power functions, `2` = logit, `3` = probit, `4` = cloglog, `5` = cauchit

`lpow:` (default: `1.0`) When `link=1`, this provides the s in $\eta = \mu^s$, the power link function; `lpow=0.0` gives the log link $\eta = \log \mu$. Common power links:
`-2.0` = $1/\mu^2$, `-1.0` = reciprocal, `0.0` = log, `0.5` = sqrt, `1.0` = identity

`yneg:` (default: `0.0`) When `dfam=2` and the response matrix Y has 1 column, this specifies the y-value used for Bernoulli "No" label ("Yes" is 1):
`0.0` when $y \in \{0, 1\}$; `-1.0` when $y \in \{-1, 1\}$

`icpt:` (default: `0`) Intercept and shifting/rescaling of the features in X:
`0` = no intercept (hence no β_0), no shifting/rescaling of the features;
`1` = add intercept, but do not shift/rescale the features in X;
`2` = add intercept, shift/rescale the features in X to mean 0, variance 1

`reg:` (default: `0.0`) L2-regularization parameter (lambda)

`tol:` (default: `0.000001`) Tolerance (epsilon) used in the convergence criterion: we terminate the outer iterations when the deviance changes by less than this factor; see below for details

`disp:` (default: `0.0`) Dispersion parameter, or `0.0` to estimate it from data

`moi:` (default: `200`) Maximum number of outer (Fisher scoring) iterations

`mii:` (default: `0`) Maximum number of inner (conjugate gradient) iterations, or 0 if no maximum limit provided

Details

In GLM, the noise distribution Prob[$y \mid \mu$] of the response variable y given its mean μ is restricted to have the *exponential family* form

$$Y \sim \text{Prob}[y \mid \mu] = \exp\left(\frac{y\theta - b(\theta)}{a} + c(y, a)\right), \text{ where } \mu = \text{E}(Y) = b'(\theta). \tag{12}$$

Changing the mean in such a distribution simply multiplies all Prob[$y \mid \mu$] by $e^{y\theta/a}$ and rescales them so that they again integrate to 1. Parameter θ is called *canonical*, and the function $\theta = {b'}^{-1}(\mu)$ that relates it to the mean is called the *canonical link*; constant a is called *dispersion* and rescales the variance of y. Many common distributions can be put into this form, see Table 11. The canonical parameter θ is often chosen to coincide with η, the linear combination of the regression features; other choices for η are possible too.

Rather than specifying the canonical link, GLM distributions are commonly defined by their variance Var(y) as the function of the mean μ. It can be shown from Eq. (12) that Var(y) = $a\, b''(\theta) = a\, b''({b'}^{-1}(\mu))$. For example,

Name	Meaning
TERMINATION_CODE	A positive integer indicating success/failure as follows: 1 = Converged successfully; 2 = Maximum # of iterations reached; 3 = Input (X, Y) out of range; 4 = Distribution/link not supported
BETA_MIN	Smallest beta value (regression coefficient), excluding the intercept
BETA_MIN_INDEX	Column index for the smallest beta value
BETA_MAX	Largest beta value (regression coefficient), excluding the intercept
BETA_MAX_INDEX	Column index for the largest beta value
INTERCEPT	Intercept value, or NaN if there is no intercept (if icpt=0)
DISPERSION	Dispersion used to scale deviance, provided in disp input argument or estimated (same as DISPERSION_EST) if disp argument is ≤ 0
DISPERSION_EST	Dispersion estimated from the dataset
DEVIANCE_UNSCALED	Deviance from the saturated model, assuming dispersion = 1.0
DEVIANCE_SCALED	Deviance from the saturated model, scaled by DISPERSION value

Table 9: Besides β, GLM regression script computes a few summary statistics listed above. They are provided in CSV format, one comma-separated name-value pair per each line.

for the Bernoulli distribution $\mathrm{Var}(y) = \mu(1-\mu)$, for the Poisson distribution $\mathrm{Var}(y) = \mu$, and for the Gaussian distribution $\mathrm{Var}(y) = a \cdot 1 = \sigma^2$. It turns out that for many common distributions $\mathrm{Var}(y) = a\mu^q$, a power function. We support all distributions where $\mathrm{Var}(y) = a\mu^q$, as well as the Bernoulli and the binomial distributions.

For distributions with $\mathrm{Var}(y) = a\mu^q$ the canonical link is also a power function, namely $\theta = \mu^{1-q}/(1-q)$, except for the Poisson ($q = 1$) whose canonical link is $\theta = \log \mu$. We support all power link functions in the form $\eta = \mu^s$, dropping any constant factor, with $\eta = \log \mu$ for $s = 0$. The binomial distribution has its own family of link functions, which includes logit (the canonical link), probit, cloglog, and cauchit (see Table 12); we support these only for the binomial and Bernoulli distributions. Links and distributions are specified via four input parameters: dfam, vpow, link, and lpow (see Table 11).

The observed response values are provided to the regression script as a matrix Y having 1 or 2 columns. If a power distribution family is selected (dfam=1), matrix Y must have 1 column that provides y_i for each x_i in the corresponding row of matrix X. When dfam=2 and Y has 1 column, we assume the Bernoulli distribution for $y_i \in \{y_{\mathrm{neg}}, 1\}$ with y_{neg} from the input parameter yneg. When dfam=2 and Y has 2 columns, we assume the binomial distribution; for each row i in X, cells $Y[i,1]$ and $Y[i,2]$ provide the positive and the negative binomial counts respectively. Internally we convert the 1-column Bernoulli into the 2-column binomial with 0-versus-1 counts.

We estimate the regression parameters via L2-regularized negative log-likelihood minimization:

$$f(\beta; X, Y) \;=\; -\sum\nolimits_{i=1}^{n} \big(y_i\theta_i - b(\theta_i)\big) \;+\; (\lambda/2)\sum\nolimits_{j=1}^{m} \beta_j^2 \;\to\; \min$$

where θ_i and $b(\theta_i)$ are from (12); note that a and $c(y,a)$ are constant w.r.t. β

Name	Meaning
NUM_CG_ITERS	Number of inner (Conj. Gradient) iterations in this outer iteration
IS_TRUST_REACHED	1 = trust region boundary was reached, 0 = otherwise
POINT_STEP_NORM	L2-norm of iteration step from old point (β-vector) to new point
OBJECTIVE	The loss function we minimize (negative partial log-likelihood)
OBJ_DROP_REAL	Reduction in the objective during this iteration, actual value
OBJ_DROP_PRED	Reduction in the objective predicted by a quadratic approximation
OBJ_DROP_RATIO	Actual-to-predicted reduction ratio, used to update the trust region
GRADIENT_NORM	L2-norm of the loss function gradient (omitted if point is rejected)
LINEAR_TERM_MIN	The minimum value of X %*% β, used to check for overflows
LINEAR_TERM_MAX	The maximum value of X %*% β, used to check for overflows
IS_POINT_UPDATED	1 = new point accepted; 0 = new point rejected, old point restored
TRUST_DELTA	Updated trust region size, the "delta"

Table 10: The Log file for GLM regression contains the above per-iteration variables in CSV format, each line containing triple (Name, Iteration#, Value) with Iteration# being 0 for initial values.

and can be ignored here. The canonical parameter θ_i depends on both β and x_i:

$$\theta_i \;=\; b'^{-1}(\mu_i) \;=\; b'^{-1}\big(g^{-1}(\eta_i)\big) \;=\; \big(b'^{-1} \circ g^{-1}\big)\left(\beta_0 + \sum\nolimits_{j=1}^{m} \beta_j x_{i,j}\right)$$

The user-provided (via reg) regularization coefficient $\lambda \geq 0$ can be used to mitigate overfitting and degeneracy in the data. Note that the intercept is never regularized.

Our iterative minimizer for $f(\beta; X, Y)$ uses the Fisher scoring approximation to the difference $\Delta f(z; \beta) = f(\beta + z; X, Y) - f(\beta; X, Y)$, recomputed at each iteration:

$$\begin{gathered}\Delta f(z;\beta) \;\approx\; 1/2 \cdot z^T A z + G^T z, \quad \text{where} \quad A = X^T \mathrm{diag}(w) X + \lambda I \\ \text{and} \quad G = -X^T u + \lambda\beta, \quad \text{with } n \times 1 \text{ vectors } w \text{ and } u \text{ given by} \\ \forall\, i = 1 \ldots n: \quad w_i = \big[v(\mu_i)\, g'(\mu_i)^2\big]^{-1}, \quad u_i = (y_i - \mu_i)\big[v(\mu_i)\, g'(\mu_i)\big]^{-1}\end{gathered}$$

Here $v(\mu_i) = \mathrm{Var}(y_i)/a$, the variance of y_i as the function of the mean, and $g'(\mu_i) = d\eta_i/d\mu_i$ is the link function derivative. The Fisher scoring approximation is minimized by trust-region conjugate gradient iterations (called the *inner* iterations, with the Fisher scoring iterations as the *outer* iterations), which approximately solve the following problem:

$$1/2 \cdot z^T A z + G^T z \;\to\; \min \quad \text{subject to} \quad \|z\|_2 \leq \delta$$

The conjugate gradient algorithm closely follows Algorithm 7.2 on page 171 of [9]. The trust region size δ is initialized as $0.5\sqrt{m}\,/\max_i \|x_i\|_2$ and updated as described in [9]. The user can specify the maximum number of the outer and the inner iterations with input parameters moi and mii, respectively. The Fisher scoring algorithm terminates successfully if $2|\Delta f(z;\beta)| < (D_1(\beta) + 0.1)\varepsilon$

INPUT PARAMETERS				Distribution	Link	Cano-
dfam	vpow	link	lpow	family	function	nical?
1	0.0	1	-1.0	Gaussian	inverse	
1	0.0	1	0.0	Gaussian	log	
1	0.0	1	1.0	Gaussian	identity	Yes
1	1.0	1	0.0	Poisson	log	Yes
1	1.0	1	0.5	Poisson	sq.root	
1	1.0	1	1.0	Poisson	identity	
1	2.0	1	-1.0	Gamma	inverse	Yes
1	2.0	1	0.0	Gamma	log	
1	2.0	1	1.0	Gamma	identity	
1	3.0	1	-2.0	Inverse Gauss	$1/\mu^2$	Yes
1	3.0	1	-1.0	Inverse Gauss	inverse	
1	3.0	1	0.0	Inverse Gauss	log	
1	3.0	1	1.0	Inverse Gauss	identity	
2	*	1	0.0	Binomial	log	
2	*	1	0.5	Binomial	sq.root	
2	*	2	*	Binomial	logit	Yes
2	*	3	*	Binomial	probit	
2	*	4	*	Binomial	cloglog	
2	*	5	*	Binomial	cauchit	

Table 11: Common GLM distribution families and link functions. (Here "*" stands for "any value.")

where $\varepsilon > 0$ is a tolerance supplied by the user via `tol`, and $D_1(\beta)$ is the unit-dispersion deviance estimated as

$$D_1(\beta) \;=\; 2 \cdot \left(\text{Prob}[Y \mid {}^{\text{saturated}}_{\text{model}}, a=1] \;-\; \text{Prob}[Y \mid X, \beta, a=1] \right)$$

The deviance estimate is also produced as part of the output. Once the Fisher scoring algorithm terminates, if requested by the user, we estimate the dispersion a from Eq. 12 using Pearson residuals

$$\hat{a} \;=\; \frac{1}{n-m} \cdot \sum_{i=1}^{n} \frac{(y_i - \mu_i)^2}{v(\mu_i)} \tag{13}$$

and use it to adjust our deviance estimate: $D_{\hat{a}}(\beta) = D_1(\beta)/\hat{a}$. If input argument `disp` is `0.0` we estimate $\hat{a}$, otherwise we use its value as a. Note that in (13) m counts the intercept ($m \leftarrow m+1$) if it is present.

Returns

The estimated regression parameters (the $\hat{\beta}_j$'s) are populated into a matrix and written to an HDFS file whose path/name was provided as the "`B`" input argument. What this matrix contains, and its size, depends on the input argument `icpt`, which specifies the user's intercept and rescaling choice:

Name	Link function	Name	Link function
Logit	$\eta = 1/\big(1+e^{-\mu}\big)$	Cloglog	$\eta = \log\big(-\log(1-\mu)\big)$
Probit	$\mu = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\eta} e^{-\frac{t^2}{2}}\,dt$	Cauchit	$\eta = \tan\pi(\mu - 1/2)$

Table 12: The supported non-power link functions for the Bernoulli and the binomial distributions. (Here μ is the Bernoulli mean.)

`icpt=0:` No intercept, matrix B has size $m \times 1$, with $B[j,1] = \hat{\beta}_j$ for each j from 1 to $m = \text{ncol}(X)$.

`icpt=1:` There is intercept, but no shifting/rescaling of X; matrix B has size $(m+1) \times 1$, with $B[j,1] = \hat{\beta}_j$ for j from 1 to m, and $B[m+1,1] = \hat{\beta}_0$, the estimated intercept coefficient.

`icpt=2:` There is intercept, and the features in X are shifted to mean = 0 and rescaled to variance = 1; then there are two versions of the $\hat{\beta}_j$'s, one for the original features and another for the shifted/rescaled features. Now matrix B has size $(m+1) \times 2$, with $B[\cdot,1]$ for the original features and $B[\cdot,2]$ for the shifted/rescaled features, in the above format. Note that $B[\cdot,2]$ are iteratively estimated and $B[\cdot,1]$ are obtained from $B[\cdot,2]$ by complementary shifting and rescaling.

Our script also estimates the dispersion $\hat{a}$ (or takes it from the user's input) and the deviances $D_1(\hat{\beta})$ and $D_{\hat{a}}(\hat{\beta})$, see Table 9 for details. A log file with variables monitoring progress through the iterations can also be made available, see Table 10.

Examples

```
hadoop jar SystemML.jar -f GLM.dml -nvargs X=/user/biadmin/X.mtx
   Y=/user/biadmin/Y.mtx B=/user/biadmin/B.mtx fmt=csv
   dfam=2 link=2 yneg=-1.0 icpt=2 reg=0.01 tol=0.00000001
   disp=1.0 moi=100 mii=10 O=/user/biadmin/stats.csv
   Log=/user/biadmin/log.csv
```

See Also

In case of binary classification problems, consider using L2-SVM or binary logistic regression; for multiclass classification, use multiclass SVM or multinomial logistic regression. For the special cases of linear regression and logistic regression, it may be more efficient to use the corresponding specialized scripts instead of GLM.

4.3 Regression Scoring and Prediction

Description

Script `GLM-predict.dml` is intended to cover all linear model based regressions, including linear regression, binomial and multinomial logistic regression,

and GLM regressions (Poisson, gamma, binomial with probit link etc.). Having just one scoring script for all these regressions simplifies maintenance and enhancement while ensuring compatible interpretations for output statistics.

The script performs two functions, prediction and scoring. To perform prediction, the script takes two matrix inputs: a collection of records X (without the response attribute) and the estimated regression parameters B, also known as β. To perform scoring, in addition to X and B, the script takes the matrix of actual response values Y that are compared to the predictions made with X and B. Of course there are other, non-matrix, input arguments that specify the model and the output format, see below for the full list.

We assume that our test/scoring dataset is given by $n \times m$-matrix X of numerical feature vectors, where each row x_i represents one feature vector of one record; we have $\dim x_i = m$. Each record also includes the response variable y_i that may be numerical, single-label categorical, or multi-label categorical. A single-label categorical y_i is an integer category label, one label per record; a multi-label y_i is a vector of integer counts, one count for each possible label, which represents multiple single-label events (observations) for the same x_i. Internally we convert single-label categoricals into multi-label categoricals by replacing each label l with an indicator vector $(0, \ldots, 0, 1_l, 0, \ldots, 0)$. In prediction-only tasks the actual y_i's are not needed to the script, but they are needed for scoring.

To perform prediction, the script matrix-multiplies X and B, adding the intercept if available, then applies the inverse of the model's link function. All GLMs assume that the linear combination of the features in x_i and the betas in B determines the means μ_i of the y_i's (in numerical or multi-label categorical form) with $\dim \mu_i = \dim y_i$. The observed y_i is assumed to follow a specified GLM family distribution $\text{Prob}[y \mid \mu_i]$ with mean(s) μ_i:

$$x_i \quad \mapsto \quad \eta_i = \beta_0 + \sum\nolimits_{j=1}^{m} \beta_j x_{i,j} \quad \mapsto \quad \mu_i \quad \mapsto \quad y_i \sim \text{Prob}[y \mid \mu_i]$$

If y_i is numerical, the predicted mean μ_i is a real number. Then our script's output matrix M is the $n \times 1$-vector of these means μ_i. Note that μ_i predicts the mean of y_i, not the actual y_i. For example, in Poisson distribution, the mean is usually fractional, but the actual y_i is always integer.

If y_i is categorical, i.e. a vector of label counts for record i, then μ_i is a vector of non-negative real numbers, one number $\mu_{i,l}$ per each label l. In this case we divide the $\mu_{i,l}$ by their sum $\sum_l \mu_{i,l}$ to obtain predicted label probabilities $p_{i,l} \in [0, 1]$. The output matrix M is the $n \times (k+1)$-matrix of these probabilities, where n is the number of records and $k+1$ is the number of categories[3]. Note again that we do not predict the labels themselves, nor their actual counts per record, but we predict the labels' probabilities.

Going from predicted probabilities to predicted labels, in the single-label categorical case, requires extra information such as the cost of false positive

[3] We use $k+1$ because there are k non-baseline categories and one baseline category, with regression parameters B having k columns.

versus false negative errors. For example, if there are 5 categories and we *accurately* predicted their probabilities as $(0.1, 0.3, 0.15, 0.2, 0.25)$, just picking the highest-probability label would be wrong 70% of the time, whereas picking the lowest-probability label might be right if, say, it represents a diagnosis of cancer or another rare and serious outcome. Hence, we keep this step outside the scope of `GLM-predict.dml` for now.

Usage

`-f` *path/*`GLM-predict.dml` `-nvargs` `X=`*path/file* `Y=`*path/file* `B=`*path/file* `M=`*path/file* `O=`*path/file* `dfam=`*int* `vpow=`*double* `link=`*int* `lpow=`*double* `disp=`*double* `fmt=`*format*

Arguments

`X:` Location (on HDFS) to read the $n \times m$-matrix X of feature vectors, each row constitutes one feature vector (one record)

`Y:` (default: `" "`) Location to read the response matrix Y needed for scoring (but optional for prediction), with the following dimensions:
$n \times 1$: acceptable for all distributions (`dfam=1` or `2` or `3`)
$n \times 2$: for binomial (`dfam=2`) if given by (#pos, #neg) counts
$n \times k+1$: for multinomial (`dfam=3`) if given by category counts

`M:` (default: `" "`) Location to write, if requested, the matrix of predicted response means (for `dfam=1`) or probabilities (for `dfam=2` or `3`):
$n \times 1$: for power-type distributions (`dfam=1`)
$n \times 2$: for binomial distribution (`dfam=2`), col# 2 is the "No" probability
$n \times k+1$: for multinomial logit (`dfam=3`), col# $k+1$ is for the baseline

`B:` Location to read matrix B of the betas, i.e. estimated GLM regression parameters, with the intercept at row# $m+1$ if available:
$\dim(B) = m \times k$: do not add intercept
$\dim(B) = m+1 \times k$: add intercept as given by the last B-row
if $k > 1$, use only B`[, 1]` unless it is Multinomial Logit (`dfam=3`)

`O:` (default: `" "`) Location to store the CSV-file with goodness-of-fit statistics defined in Table 13, the default is to print them to the standard output

`dfam:` (default: `1`) GLM distribution family code to specify the type of distribution $\text{Prob}[y \mid \mu]$ that we assume:
`1` = power distributions with $\text{Var}(y) = \mu^\alpha$, see Table 11;
`2` = binomial; `3` = multinomial logit

`vpow:` (default: `0.0`) Power for variance defined as $(\text{mean})^{\text{power}}$ (ignored if `dfam` $\neq 1$): when `dfam=1`, this provides the q in $\text{Var}(y) = a\mu^q$, the power dependence of the variance of y on its mean. In particular, use:
`0.0` = Gaussian, `1.0` = Poisson, `2.0` = Gamma, `3.0` = inverse Gaussian

`link:` (default: `0`) Link function code to determine the link function $\eta = g(\mu)$, ignored for multinomial logit (`dfam=3`):
`0` = canonical link (depends on the distribution family), see Table 11;
`1` = power functions, `2` = logit, `3` = probit, `4` = cloglog, `5` = cauchit

Name	CID	Disp?	Meaning
LOGLHOOD_Z		+	Log-likelihood Z-score (in st. dev.'s from the mean)
LOGLHOOD_Z_PVAL		+	Log-likelihood Z-score p-value, two-sided
PEARSON_X2		+	Pearson residual X^2-statistic
PEARSON_X2_BY_DF		+	Pearson X^2 divided by degrees of freedom
PEARSON_X2_PVAL		+	Pearson X^2 p-value
DEVIANCE_G2		+	Deviance from the saturated model G^2-statistic
DEVIANCE_G2_BY_DF		+	Deviance G^2 divided by degrees of freedom
DEVIANCE_G2_PVAL		+	Deviance G^2 p-value
AVG_TOT_Y	+		Y-column average for an individual response value
STDEV_TOT_Y	+		Y-column st. dev. for an individual response value
AVG_RES_Y	+		Y-column residual average of $Y -$ pred. mean$(Y\|X)$
STDEV_RES_Y	+		Y-column residual st. dev. of $Y -$ pred. mean$(Y\|X)$
PRED_STDEV_RES	+	+	Model-predicted Y-column residual st. deviation
PLAIN_R2	+		Plain R^2 of Y-column residual with bias included
ADJUSTED_R2	+		Adjusted R^2 of Y-column residual w. bias included
PLAIN_R2_NOBIAS	+		Plain R^2 of Y-column residual, bias subtracted
ADJUSTED_R2_NOBIAS	+		Adjusted R^2 of Y-column residual, bias subtracted

Table 13: The above goodness-of-fit statistics are provided in CSV format, one per each line, with four columns: (Name, [CID], [Disp?], Value). The columns are: "Name" is the string identifier for the statistic, see the table; "CID" is an optional integer value that specifies the Y-column index for per-column statistics (note that a bi-/multinomial one-column `Y`-input is converted into multi-column); "Disp?" is an optional Boolean value (`TRUE` or `FALSE`) that tells us whether or not scaling by the input dispersion parameter `disp` has been applied to this statistic; "Value" is the value of the statistic.

`lpow`: (default: `1.0`) Power for link function defined as (mean)$^{\text{power}}$ (ignored if `link` $\neq 1$): when `link=1`, this provides the s in $\eta = \mu^s$, the power link function; `lpow=0.0` gives the log link $\eta = \log \mu$. Common power links: `-2.0` $= 1/\mu^2$, `-1.0` = reciprocal, `0.0` = log, `0.5` = sqrt, `1.0` = identity

`disp`: (default: `1.0`) Dispersion value, when available; must be positive

`fmt`: (default: `"text"`) Matrix `M` file output format, such as `text`, `mm`, or `csv`; see read/write functions in SystemML Language Reference for details.

Details

The output matrix M of predicted means (or probabilities) is computed by matrix-multiplying X with the first column of B or with the whole B in the multinomial case, adding the intercept if available (conceptually, appending an extra column of ones to X); then applying the inverse of the model's link function. The difference between "means" and "probabilities" in the categorical case becomes significant when there are ≥ 2 observations per record (with the multi-label records) or when the labels such as -1 and 1 are viewed and averaged as numerical response values (with the single-label records). To avoid any mix-up or information loss, we separately return the predicted probability of each category label for each record.

When the "actual" response values Y are available, the summary statistics are computed and written out as described in Table 13. Below we discuss each of these statistics in detail. Note that in the categorical case (binomial and multinomial) Y is internally represented as the matrix of observation counts for each label in each record, rather than just the label ID for each record. The input Y may already be a matrix of counts, in which case it is used as-is. But if Y is given as a vector of response labels, each response label is converted into an indicator vector $(0, \ldots, 0, 1_l, 0, \ldots, 0)$ where l is the label ID for this record. All negative (e.g. -1) or zero label IDs are converted to the $1+$maximum label ID. The largest label ID is viewed as the "baseline" as explained in the section on Multinomial Logistic Regression. We assume that there are $k \geq 1$ non-baseline categories and one (last) baseline category.

We also estimate residual variances for each response value, although we do not output them, but use them only inside the summary statistics, scaled and unscaled by the input dispersion parameter `disp`, as described below.

`LOGLHOOD_Z` and `LOGLHOOD_Z_PVAL` statistics measure how far the log-likelihood of Y deviates from its expected value according to the model. The script implements them only for the binomial and the multinomial distributions, returning NaN for all other distributions. Pearson's X^2 and deviance G^2 often perform poorly with bi- and multinomial distributions due to low cell counts, hence we need this extra goodness-of-fit measure. To compute these statistics, we use:

- the $n \times (k+1)$-matrix Y of multi-label response counts, in which $y_{i,j}$ is the number of times label j was observed in record i;
- the model-estimated probability matrix P of the same dimensions that satisfies $\sum_{j=1}^{k+1} p_{i,j} = 1$ for all $i = 1, \ldots, n$ and where $p_{i,j}$ is the model probability of observing label j in record i;
- the $n \times 1$-vector N where N_i is the aggregated count of observations in record i (all $N_i = 1$ if each record has only one response label).

We start by computing the multinomial log-likelihood of Y given P and N, as well as the expected log-likelihood given a random Y and the variance of this log-likelihood if Y indeed follows the proposed distribution:

$$\begin{aligned}
\ell(Y) &= \log \mathrm{Prob}[Y \mid P, N] = \sum_{i=1}^{n} \sum_{j=1}^{k+1} y_{i,j} \log p_{i,j} \\
\mathrm{E}_Y\, \ell(Y) &= \sum_{i=1}^{n} \sum_{j=1}^{k+1} \mu_{i,j} \log p_{i,j} = \sum_{i=1}^{n} N_i \sum_{j=1}^{k+1} p_{i,j} \log p_{i,j} \\
\mathrm{Var}_Y\, \ell(Y) &= \sum_{i=1}^{n} N_i \left(\sum_{j=1}^{k+1} p_{i,j} \big(\log p_{i,j}\big)^2 - \left(\sum_{j=1}^{k+1} p_{i,j} \log p_{i,j} \right)^2 \right)
\end{aligned}$$

Then we compute the Z-score as the difference between the actual and the expected log-likelihood $\ell(Y)$ divided by its expected standard deviation, and its

two-sided p-value in the Normal distribution assumption ($\ell(Y)$ should approach normality due to the Central Limit Theorem):

$$Z = \frac{\ell(Y) - \mathrm{E}_Y\,\ell(Y)}{\sqrt{\mathrm{Var}_Y\,\ell(Y)}}; \quad \text{p-value}(Z) = \mathrm{Prob}\left[\,\left|\,\mathrm{Normal}(0,1)\right| > |Z|\,\right]$$

A low p-value would indicate "underfitting" if $Z \ll 0$ or "overfitting" if $Z \gg 0$. Here "overfitting" means that higher-probability labels occur more often than their probabilities suggest.

We also apply the dispersion input (`disp`) to compute the "scaled" version of the Z-score and its p-value. Since $\ell(Y)$ is a linear function of Y, multiplying the GLM-predicted variance of Y by `disp` results in multiplying $\mathrm{Var}_Y\,\ell(Y)$ by the same `disp`. This, in turn, translates into dividing the Z-score by the square root of the dispersion:

$$Z_{\texttt{disp}} = \big(\ell(Y) - \mathrm{E}_Y\,\ell(Y)\big) \,/\, \sqrt{\texttt{disp} \cdot \mathrm{Var}_Y\,\ell(Y)} = Z/\sqrt{\texttt{disp}}$$

Finally, we recalculate the p-value with this new Z-score.

`PEARSON_X2`, `PEARSON_X2_BY_DF`, and `PEARSON_X2_PVAL`: Pearson's residual X^2-statistic is a commonly used goodness-of-fit measure for linear models [7]. The idea is to measure how well the model-predicted means and variances match the actual behavior of response values. For each record i, we estimate the mean μ_i and the variance v_i (or $\texttt{disp} \cdot v_i$) and use them to normalize the residual: $r_i = (y_i - \mu_i)/\sqrt{v_i}$. These normalized residuals are then squared, aggregated by summation, and tested against an appropriate χ^2 distribution. The computation of X^2 is slightly different for categorical data (bi- and multinomial) than it is for numerical data, since y_i has multiple correlated dimensions [7]:

$$X^2\,(\text{numer.}) = \sum_{i=1}^{n} \frac{(y_i - \mu_i)^2}{v_i}; \quad X^2\,(\text{categ.}) = \sum_{i=1}^{n} \sum_{j=1}^{k+1} \frac{(y_{i,j} - N_i p_{i,j})^2}{N_i p_{i,j}}$$

The number of degrees of freedom #d.f. for the χ^2 distribution is $n - m$ for numerical data and $(n - m)k$ for categorical data, where $k = \texttt{ncol}(Y) - 1$. Given the dispersion parameter `disp`, the X^2 statistic is scaled by division: $X^2_{\texttt{disp}} = X^2/\texttt{disp}$. If the dispersion is accurate, $X^2/\texttt{disp}$ should be close to #d.f. In fact, $X^2/\#\text{d.f.}$ over the *training* data is the dispersion estimator used in our `GLM.dml` script, see (13). Here we provide $X^2/\#\text{d.f.}$ and $X^2_{\texttt{disp}}/\#\text{d.f.}$ as `PEARSON_X2_BY_DF` to enable dispersion comparison between the training data and the test data.

NOTE: For categorical data, both Pearson's X^2 and the deviance G^2 are unreliable (i.e. do not approach the χ^2 distribution) unless the predicted means of multi-label counts $\mu_{i,j} = N_i p_{i,j}$ are fairly large: all ≥ 1 and 80% are at least 5 [4]. They should not be used for "one label per record" categoricals.

`DEVIANCE_G2`, `DEVIANCE_G2_BY_DF`, and `DEVIANCE_G2_PVAL`: Deviance G^2 is the log of the likelihood ratio between the "saturated" model and the linear

model being tested for the given dataset, multiplied by two:

$$G^2 \;=\; 2\,\log\frac{\text{Prob}[Y \mid \text{saturated model}]}{\text{Prob}[Y \mid \text{tested linear model}]} \tag{14}$$

The "saturated" model sets the mean μ_i^{sat} to equal y_i for every record (for categorical data, $p_{i,j}^{\text{sat}} = y_{i,j}/N_i$), which represents the "perfect fit." For records with $y_{i,j} \in \{0, N_i\}$ or otherwise at a boundary, by continuity we set $0 \log 0 = 0$. The GLM likelihood functions defined in (12) become simplified in ratio (14) due to canceling out the term $c(y, a)$ since it is the same in both models.

The log of a likelihood ratio between two nested models, times two, is known to approach a χ^2 distribution as $n \to \infty$ if both models have fixed parameter spaces. But this is not the case for the "saturated" model: it adds more parameters with each record. In practice, however, χ^2 distributions are used to compute the p-value of G^2 [7]. The number of degrees of freedom #d.f. and the treatment of dispersion are the same as for Pearson's X^2, see above.

Column-wise statistics. The rest of the statistics are computed separately for each column of Y. As explained above, Y has two or more columns in bi- and multinomial case, either at input or after conversion. Moreover, each $y_{i,j}$ in record i with $N_i \geq 2$ is counted as N_i separate observations $y_{i,j,l}$ of 0 or 1 (where $l = 1, \ldots, N_i$) with $y_{i,j}$ ones and $N_i - y_{i,j}$ zeros. For power distributions, including linear regression, Y has only one column and all $N_i = 1$, so the statistics are computed for all Y with each record counted once. Below we denote $N = \sum_{i=1}^{n} N_i \geq n$. Here is the total average and the residual average (residual bias) of $y_{i,j,l}$ for each Y-column:

$$\texttt{AVG\_TOT\_Y}_j \;=\; \frac{1}{N}\sum_{i=1}^{n} y_{i,j}; \quad \texttt{AVG\_RES\_Y}_j \;=\; \frac{1}{N}\sum_{i=1}^{n}(y_{i,j} - \mu_{i,j})$$

Dividing by N (rather than n) gives the averages for $y_{i,j,l}$ (rather than $y_{i,j}$). The total variance, and the standard deviation, for individual observations $y_{i,j,l}$ is estimated from the total variance for response values $y_{i,j}$ using independence assumption: $\text{Var}\, y_{i,j} = \text{Var} \sum_{l=1}^{N_i} y_{i,j,l} = \sum_{l=1}^{N_i} \text{Var}\, y_{i,j,l}$. This allows us to estimate the sum of squares for $y_{i,j,l}$ via the sum of squares for $y_{i,j}$:

$$\texttt{STDEV\_TOT\_Y}_j \;=\; \left[\frac{1}{N-1}\sum_{i=1}^{n}\Big(y_{i,j} - \frac{N_i}{N}\sum_{i'=1}^{n} y_{i',j}\Big)^2\right]^{1/2}$$

Analogously, we estimate the standard deviation of the residual $y_{i,j,l} - \mu_{i,j,l}$:

$$\texttt{STDEV\_RES\_Y}_j \;=\; \left[\frac{1}{N-m'}\sum_{i=1}^{n}\Big(y_{i,j} - \mu_{i,j} - \frac{N_i}{N}\sum_{i'=1}^{n}(y_{i',j} - \mu_{i',j})\Big)^2\right]^{1/2}$$

Here $m' = m$ if m includes the intercept as a feature and $m' = m + 1$ if it does not. The estimated standard deviations can be compared to the model-predicted residual standard deviation computed from the predicted means by

R^2 where the residual sum-of-squares includes the bias contribution:

$\texttt{PLAIN\_R2}_j \;=\; 1 - \dfrac{\sum_{i=1}^{n} (y_{i,j} - \mu_{i,j})^2}{\sum_{i=1}^{n} \left(y_{i,j} - \frac{N_i}{N} \sum_{i'=1}^{n} y_{i',j}\right)^2}$	$\texttt{ADJUSTED\_R2}_j \;=\; 1 - \frac{N-1}{N-m} \dfrac{\sum_{i=1}^{n} (y_{i,j} - \mu_{i,j})^2}{\sum_{i=1}^{n} \left(y_{i,j} - \frac{N_i}{N} \sum_{i'=1}^{n} y_{i',j}\right)^2}$

R^2 where the residual sum-of-squares is centered so that the bias is subtracted:

$\texttt{PLAIN\_R2\_NOBIAS}_j \;=\; 1 - \dfrac{\sum_{i=1}^{n} \left(y_{i,j} - \mu_{i,j} - \frac{N_i}{N} \sum_{i'=1}^{n} (y_{i',j} - \mu_{i',j})\right)^2}{\sum_{i=1}^{n} \left(y_{i,j} - \frac{N_i}{N} \sum_{i'=1}^{n} y_{i',j}\right)^2}$	$\texttt{ADJUSTED\_R2\_NOBIAS}_j \;=\; 1 - \frac{N-1}{N-m'} \dfrac{\sum_{i=1}^{n} \left(y_{i,j} - \mu_{i,j} - \frac{N_i}{N} \sum_{i'=1}^{n} (y_{i',j} - \mu_{i',j})\right)^2}{\sum_{i=1}^{n} \left(y_{i,j} - \frac{N_i}{N} \sum_{i'=1}^{n} y_{i',j}\right)^2}$

Table 14: The R^2 statistics we compute in `GLM-predict.dml`

the GLM variance formula and scaled by the dispersion:

$$\texttt{PRED\_STDEV\_RES}_j \;=\; \left[\frac{\texttt{disp}}{N} \sum_{i=1}^{n} v(\mu_{i,j})\right]^{1/2}$$

We also compute the R^2 statistics for each column of Y, see Table 14 for details. We compute two versions of R^2: in one version the residual sum-of-squares (RSS) includes any bias in the residual that might be present (due to the lack of, or inaccuracy in, the intercept); in the other version of RSS the bias is subtracted by "centering" the residual. In both cases we subtract the bias in the total sum-of-squares (in the denominator), and m' equals m with the intercept or $m + 1$ without the intercept.

Returns

The matrix of predicted means (if the response is numerical) or probabilities (if the response is categorical), see "Description" subsection above for more information. Given `Y`, we return some statistics in CSV format as described in Table 13 and in the above text.

Examples

Note that in the examples below the value for "`disp`" input argument is set arbitrarily. The correct dispersion value should be computed from the training data during model estimation, or omitted if unknown (which sets it to `1.0`).

Linear regression example:

```
hadoop jar SystemML.jar -f GLM-predict.dml -nvargs dfam=1
   vpow=0.0 link=1 lpow=1.0 disp=5.67 X=/user/biadmin/X.mtx
   B=/user/biadmin/B.mtx M=/user/biadmin/Means.mtx fmt=csv
   Y=/user/biadmin/Y.mtx O=/user/biadmin/stats.csv
```

Linear regression example, prediction only (no `Y` given):

```
hadoop jar SystemML.jar -f GLM-predict.dml -nvargs
   dfam=1 vpow=0.0 link=1 lpow=1.0 X=/user/biadmin/X.mtx
   B=/user/biadmin/B.mtx M=/user/biadmin/Means.mtx fmt=csv
```

Binomial logistic regression example:

```
hadoop jar SystemML.jar -f GLM-predict.dml -nvargs
   dfam=2 link=2 disp=3.0004464 X=/user/biadmin/X.mtx
   B=/user/biadmin/B.mtx M=/user/biadmin/Probabilities.mtx
   fmt=csv Y=/user/biadmin/Y.mtx O=/user/biadmin/stats.csv
```

Binomial probit regression example:

```
hadoop jar SystemML.jar -f GLM-predict.dml -nvargs
   dfam=2 link=3 disp=3.0004464 X=/user/biadmin/X.mtx
   B=/user/biadmin/B.mtx M=/user/biadmin/Probabilities.mtx
   fmt=csv Y=/user/biadmin/Y.mtx O=/user/biadmin/stats.csv
```

Multinomial logistic regression example:

```
hadoop jar SystemML.jar -f GLM-predict.dml -nvargs
   dfam=3 X=/user/biadmin/X.mtx B=/user/biadmin/B.mtx
   M=/user/biadmin/Probabilities.mtx fmt=csv
   Y=/user/biadmin/Y.mtx O=/user/biadmin/stats.csv
```

Poisson regression with the log link example:

```
hadoop jar SystemML.jar -f GLM-predict.dml -nvargs dfam=1
   vpow=1.0 link=1 lpow=0.0 disp=3.45 X=/user/biadmin/X.mtx
   B=/user/biadmin/B.mtx M=/user/biadmin/Means.mtx fmt=csv
   Y=/user/biadmin/Y.mtx O=/user/biadmin/stats.csv
```

Gamma regression with the inverse (reciprocal) link example:

```
hadoop jar SystemML.jar -f GLM-predict.dml -nvargs dfam=1
   vpow=2.0 link=1 lpow=-1.0 disp=1.99118 X=/user/biadmin/X.mtx
   B=/user/biadmin/B.mtx M=/user/biadmin/Means.mtx fmt=csv
   Y=/user/biadmin/Y.mtx O=/user/biadmin/stats.csv
```

References

[1] A. Agresti. *Categorical Data Analysis*. Wiley Series in Probability and Statistics. Wiley-Interscience, second edition, 2002.

[2] D. Aloise, A. Deshpande, P. Hansen, and P. Popat. NP-hardness of Euclidean sum-of-squares clustering. *Machine Learning*, 75(2):245–248, May 2009.

[3] D. Arthur and S. Vassilvitskii. `k-means++`: The advantages of careful seeding. In *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2007)*, pages 1027–1035, New Orleans LA, USA, January 7–9 2007.

[4] W. G. Cochran. Some methods for strengthening the common χ^2 tests. *Biometrics*, 10(4):417–451, December 1954.

[5] J. Gill. *Generalized Linear Models: A Unified Approach*. Number 07-134 in Sage University Papers Series on Quantitative Applications in the Social Sciences. Sage Publications, 2000.

[6] C.-J. Lin, R. C. Weng, and S. S. Keerthi. Trust region Newton method for large-scale logistic regression. *Journal of Machine Learning Research*, 9:627–650, April 2008.

[7] P. McCullagh and J. A. Nelder. *Generalized Linear Models*. Number 37 in Monographs on Statistics and Applied Probability. Chapman & Hall/CRC, second edition, 1989.

[8] J. A. Nelder and R. W. M. Wedderburn. Generalized linear models. *Journal of the Royal Statistical Society, Series A (General)*, 135(3):370–384, 1972.

[9] J. Nocedal and S. Wright. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer, second edition, 2006.