`SystemDS` Algorithms Reference

April 8, 2020

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, feature, or column) in the data has 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 also referred to as *quantitative* or *continuous* variables. In contrast, a *categorical* variable has a fixed 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 classified 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.

The input dataset for descriptive statistics is provided in the form of a matrix, whose rows are the records (data points) and whose columns are the features (i.e. variables). Some scripts allow this matrix to be vertically split into two or three matrices. Descriptive statistics are computed over the specified features (columns) in the matrix. Which statistics are computed depends on the types of the features. It is important to keep in mind the following caveats and restrictions:

1. Given a finite set of data records, i.e. a *sample*, we take their feature values and compute their *sample statistics*. These statistics will vary from sample to sample even if the underlying distribution of feature values remains the same. Sample statistics are accurate for the given sample only. If the goal is to estimate the *distribution statistics* that are parameters of the (hypothesized) underlying distribution of the features, the corresponding sample statistics may sometimes be used as approximations, but their accuracy will vary.

2. In particular, the accuracy of the estimated distribution statistics will be low if the number of values in the sample is small. That is, for small samples, the computed statistics may depend on the randomness of the individual sample values more than on the underlying distribution of the features.
3. The accuracy will also be low if the sample records cannot be assumed mutually independent and identically distributed (i.i.d.), that is, sampled at random from the same underlying distribution. In practice, feature values in one record often depend on other features and other records, including unknown ones.
4. Most of the computed statistics will have low estimation accuracy in the presence of extreme values (outliers) or if the underlying distribution has heavy tails, for example obeys a power law. However, a few of the computed statistics, such as the median and Spearman's rank correlation coefficient, are *robust* to outliers.
5. Some sample statistics are reported with their *sample standard errors* in an attempt to quantify their accuracy as distribution parameter estimators. But these sample standard errors, in turn, only estimate the underlying distribution's standard errors and will have low accuracy for small or non-i.i.d. samples, outliers in samples, or heavy-tailed distributions.
6. We assume that the quantitative (scale) feature columns do not contain missing values, infinite values, `NaN`s, or coded non-numeric values, unless otherwise specified. We assume that each categorical feature column contains positive integers from 1 to the number of categories; for ordinal features, the natural order on the integers should coincide with the order on the categories.

1.1 Univariate Statistics

Description

Univariate statistics are the simplest form of descriptive statistics in data analysis. They are used to quantitatively describe the main characteristics of each feature in the data. For a given dataset matrix, script `Univar-Stats.dml` computes certain univariate statistics for each feature column in the matrix. The feature type governs the exact set of statistics computed for that feature. For example, the statistic *mean* can only be computed on a quantitative (scale) feature like 'Height' and 'Temperature'. It does not make sense to compute the mean of a categorical attribute like 'Hair Color'.

Usage

`-f Univar-Stats.dml -nvargs X=`*path/file* `TYPES=`*path/file* `STATS=`*path/file*

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

Table 1: The output matrix of `Univar-Stats.dml` has one row per each univariate statistic and one column per input feature. This table lists the meaning of each row. Signs "+" show applicability to scale or/and to categorical features.

Arguments

`X`: Location (on HDFS) to read the data matrix X whose columns we want to analyze as the features.

`TYPES`: Location (on HDFS) to read the single-row matrix whose i^{th} column-cell contains the type of the i^{th} feature column `X[, i]` in the data matrix. Feature types must be encoded by integer numbers: 1 = scale, 2 = nominal, 3 = ordinal.

`STATS`: Location (on HDFS) where the output matrix of computed statistics will be stored. The format of the output matrix is defined by Table 1.

Details

Given an input matrix `X`, this script computes the set of all relevant univariate statistics for each feature column `X[, i]` in `X`. The list of statistics to be computed depends on the *type*, or *measurement level*, of each column. The TYPES command-line argument points to a vector containing the types of all columns. The types must be provided as per the following convention: 1 = scale, 2 = nominal, 3 = ordinal.

Below we list all univariate statistics computed by script `Univar-Stats.dml`. The statistics are collected by relevance into several groups, namely: central tendency, dispersion, shape, and categorical measures. The first three groups

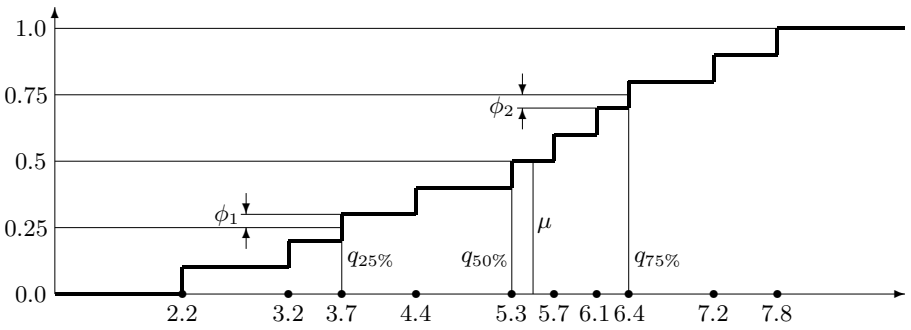


Figure 1: The computation of quartiles, median, and interquartile mean from the empirical distribution function of the 10-point sample {2.2, 3.2, 3.7, 4.4, 5.3, 5.7, 6.1, 6.4, 7.2, 7.8}. Each vertical step in the graph has height $1/n = 0.1$. Values $q_{25\%}$, $q_{50\%}$, and $q_{75\%}$ denote the 1st, 2nd, and 3rd quartiles correspondingly; value μ denotes the median. Values ϕ_1 and ϕ_2 show the partial contribution of border points (quartiles) $v_3 = 3.7$ and $v_8 = 6.4$ into the interquartile mean.

contain statistics computed for a quantitative (also known as: numerical, scale, or continuous) feature; the last group contains the statistics for a categorical (either nominal or ordinal) feature.

Let n be the number of data records (rows) with feature values. In what follows we fix a column index `idx` and consider sample statistics of feature column `X[ , idx]`. Let $v = (v_1, v_2, \ldots, v_n)$ be the values of `X[ , idx]` in their original unsorted order: $v_i =$ `X[`i`, idx]`. Let $v^s = (v^s_1, v^s_2, \ldots, v^s_n)$ be the same values in the sorted order, preserving duplicates: $v^s_1 \leq v^s_2 \leq \ldots \leq v^s_n$.

Central tendency measures. Sample statistics that describe the location of the quantitative (scale) feature distribution, represent it with a single value.

Mean (output row 4): The arithmetic average over a sample of a quantitative feature. Computed as the ratio between the sum of values and the number of values: $(\sum_{i=1}^{n} v_i)/n$. Example: the mean of sample {2.2, 3.2, 3.7, 4.4, 5.3, 5.7, 6.1, 6.4, 7.2, 7.8} equals 5.2.

Note that the mean is significantly affected by extreme values in the sample and may be misleading as a central tendency measure if the feature varies on exponential scale. For example, the mean of {0.01, 0.1, 1.0, 10.0, 100.0} is 22.222, greater than all the sample values except the largest.

Median (output row 13): The "middle" value that separates the higher half of the sample values (in a sorted order) from the lower half. To compute the median, we sort the sample in the increasing order, preserving duplicates: $v^s_1 \leq v^s_2 \leq \ldots \leq v^s_n$. If n is odd, the median equals v^s_i where $i = (n+1)\,/\,2$, same as the 50th percentile of the sample. If n is even, there are two "middle" values $v^s_{n/2}$ and $v^s_{n/2+1}$, so we compute the median as the mean of these two values. (For even n we compute the 50th percentile as $v^s_{n/2}$,

not as the median.) Example: the median of sample {2.2, 3.2, 3.7, 4.4, 5.3, 5.7, 6.1, 6.4, 7.2, 7.8} equals $(5.3 + 5.7) \,/\, 2 = 5.5$, see Figure 1.1.

Unlike the mean, the median is not sensitive to extreme values in the sample, i.e. it is robust to outliers. It works better as a measure of central tendency for heavy-tailed distributions and features that vary on exponential scale. However, the median is sensitive to small sample size.

Interquartile mean (output row 14): For a sample of a quantitative feature, this is the mean of the values greater than or equal to the 1st quartile and less than or equal the 3rd quartile. In other words, it is a "truncated mean" where the lowest 25% and the highest 25% of the sorted values are omitted in its computation. The two "border values", i.e. the 1st and the 3rd quartiles themselves, contribute to this mean only partially. This measure is occasionally used as the "robust" version of the mean that is less sensitive to the extreme values.

To compute the measure, we sort the sample in the increasing order, preserving duplicates: $v_1^s \le v_2^s \le \ldots \le v_n^s$. We set $j = \lceil n/4 \rceil$ for the 1st quartile index and $k = \lceil 3n/4 \rceil$ for the 3rd quartile index, then compute the following weighted mean:

$$\frac{1}{3/4 - 1/4} \left[\left(\frac{j}{n} - \frac{1}{4} \right) v_j^s \; + \sum_{j<i<k} \left(\frac{i}{n} - \frac{i-1}{n} \right) v_i^s \; + \; \left(\frac{3}{4} - \frac{k-1}{n} \right) v_k^s \right]$$

In other words, all sample values between the 1st and the 3rd quartile enter the sum with weights $2/n$, times their number of duplicates, while the two quartiles themselves enter the sum with reduced weights. The weights are proportional to the vertical steps in the empirical distribution function of the sample, see Figure 1.1 for an illustration. Example: the interquartile mean of sample {2.2, 3.2, 3.7, 4.4, 5.3, 5.7, 6.1, 6.4, 7.2, 7.8} equals the sum $0.1(3.7 + 6.4) + 0.2(4.4 + 5.3 + 5.7 + 6.1)$, which equals 5.31.

Dispersion measures. Statistics that describe the amount of variation or spread in a quantitative (scale) data feature.

Variance (output row 5): A measure of dispersion, or spread-out, of sample values around their mean, expressed in units that are the square of those of the feature itself. Computed as the sum of squared differences between the values in the sample and their mean, divided by one less than the number of values: $\sum_{i=1}^{n} (v_i - \bar{v})^2 \,/\, (n-1)$ where $\bar{v} = (\sum_{i=1}^{n} v_i)/n$. Example: the variance of sample {2.2, 3.2, 3.7, 4.4, 5.3, 5.7, 6.1, 6.4, 7.2, 7.8} equals 3.24. Note that at least two values ($n \ge 2$) are required to avoid division by zero. Sample variance is sensitive to outliers, even more than the mean.

Standard deviation (output row 6): A measure of dispersion around the mean, the square root of variance. Computed by taking the square root of the sample variance; see *Variance* above on computing the variance. Example:

the standard deviation of sample {2.2, 3.2, 3.7, 4.4, 5.3, 5.7, 6.1, 6.4, 7.2, 7.8} equals 1.8. At least two values are required to avoid division by zero. Note that standard deviation is sensitive to outliers.

Standard deviation is used in conjunction with the mean to determine an interval containing a given percentage of the feature values, assuming the normal distribution. In a large sample from a normal distribution, around 68% of the cases fall within one standard deviation and around 95% of cases fall within two standard deviations of the mean. For example, if the mean age is 45 with a standard deviation of 10, around 95% of the cases would be between 25 and 65 in a normal distribution.

Coefficient of variation (output row 8): The ratio of the standard deviation to the mean, i.e. the *relative* standard deviation, of a quantitative feature sample. Computed by dividing the sample *standard deviation* by the sample *mean*, see above for their computation details. Example: the coefficient of variation for sample {2.2, 3.2, 3.7, 4.4, 5.3, 5.7, 6.1, 6.4, 7.2, 7.8} equals $1.8 \,/\, 5.2 \approx 0.346$.

This metric is used primarily with non-negative features such as financial or population data. It is sensitive to outliers. Note: zero mean causes division by zero, returning infinity or `NaN`. At least two values (records) are required to compute the standard deviation.

Minimum (output row 1): The smallest value of a quantitative sample, computed as $\min v = v_1^s$. Example: the minimum of sample {2.2, 3.2, 3.7, 4.4, 5.3, 5.7, 6.1, 6.4, 7.2, 7.8} equals 2.2.

Maximum (output row 2): The largest value of a quantitative sample, computed as $\max v = v_n^s$. Example: the maximum of sample {2.2, 3.2, 3.7, 4.4, 5.3, 5.7, 6.1, 6.4, 7.2, 7.8} equals 7.8.

Range (output row 3): The difference between the largest and the smallest value of a quantitative sample, computed as $\max v - \min v = v_n^s - v_1^s$. It provides information about the overall spread of the sample values. Example: the range of sample {2.2, 3.2, 3.7, 4.4, 5.3, 5.7, 6.1, 6.4, 7.2, 7.8} equals $7.8 - 2.2 = 5.6$.

Standard error of the mean (output row 7): A measure of how much the value of the sample mean may vary from sample to sample taken from the same (hypothesized) distribution of the feature. It helps to roughly bound the distribution mean, i.e. the limit of the sample mean as the sample size tends to infinity. Under certain assumptions (e.g. normality and large sample), the difference between the distribution mean and the sample mean is unlikely to exceed 2 standard errors.

The measure is computed by dividing the sample standard deviation by the square root of the number of values n; see *standard deviation* for its computation details. Ensure $n \geq 2$ to avoid division by 0. Example: for sample {2.2, 3.2, 3.7, 4.4, 5.3, 5.7, 6.1, 6.4, 7.2, 7.8} with the mean of 5.2 the standard error of the mean equals $1.8 \,/\sqrt{10} \approx 0.569$.

Note that the standard error itself is subject to sample randomness. Its accuracy as an error estimator may be low if the sample size is small or non-i.i.d., if there are outliers, or if the distribution has heavy tails.

Shape measures. Statistics that describe the shape and symmetry of the quantitative (scale) feature distribution estimated from a sample of its values.

Skewness (output row 9): It measures how symmetrically the values of a feature are spread out around the mean. A significant positive skewness implies a longer (or fatter) right tail, i.e. feature values tend to lie farther away from the mean on the right side. A significant negative skewness implies a longer (or fatter) left tail. The normal distribution is symmetric and has a skewness value of 0; however, its sample skewness is likely to be nonzero, just close to zero. As a guideline, a skewness value more than twice its standard error is taken to indicate a departure from symmetry.

Skewness is computed as the 3rd central moment divided by the cube of the standard deviation. We estimate the 3rd central moment as the sum of cubed differences between the values in the feature column and their sample mean, divided by the number of values: $\sum_{i=1}^{n}(v_i - \bar{v})^3/n$ where $\bar{v} = (\sum_{i=1}^{n} v_i)/n$. The standard deviation is computed as described above in *standard deviation*. To avoid division by 0, at least two different sample values are required. Example: for sample {2.2, 3.2, 3.7, 4.4, 5.3, 5.7, 6.1, 6.4, 7.2, 7.8} with the mean of 5.2 and the standard deviation of 1.8 skewness is estimated as $-1.0728 \,/\, 1.8^3 \approx -0.184$. Note: skewness is sensitive to outliers.

Standard error in skewness (output row 11): A measure of how much the sample skewness may vary from sample to sample, assuming that the feature is normally distributed, which makes its distribution skewness equal 0. Given the number n of sample values, the standard error is computed as

$$\sqrt{\frac{6n\,(n-1)}{(n-2)(n+1)(n+3)}}$$

This measure can tell us, for example:

- If the sample skewness lands within two standard errors from 0, its positive or negative sign is non-significant, may just be accidental.
- If the sample skewness lands outside this interval, the feature is unlikely to be normally distributed.

At least 3 values ($n \geq 3$) are required to avoid arithmetic failure. Note that the standard error is inaccurate if the feature distribution is far from normal or if the number of samples is small.

Kurtosis (output row 10): As a distribution parameter, kurtosis is a measure of the extent to which feature values cluster around a central point. In other

words, it quantifies "peakedness" of the distribution: how tall and sharp the central peak is relative to a standard bell curve.

Positive kurtosis (*leptokurtic* distribution) indicates that, relative to a normal distribution:

- observations cluster more about the center (peak-shaped),
- the tails are thinner at non-extreme values,
- the tails are thicker at extreme values.

Negative kurtosis (*platykurtic* distribution) indicates that, relative to a normal distribution:

- observations cluster less about the center (box-shaped),
- the tails are thicker at non-extreme values,
- the tails are thinner at extreme values.

Kurtosis of a normal distribution is zero; however, the sample kurtosis (computed here) is likely to deviate from zero.

Sample kurtosis is computed as the 4^{th} central moment divided by the 4^{th} power of the standard deviation, minus 3. We estimate the 4^{th} central moment as the sum of the 4^{th} powers of differences between the values in the feature column and their sample mean, divided by the number of values: $\sum_{i=1}^{n}(v_i - \bar{v})^4/n$ where $\bar{v} = (\sum_{i=1}^{n} v_i)/n$. The standard deviation is computed as described above, see *standard deviation*.

Note that kurtosis is sensitive to outliers, and requires at least two different sample values. Example: for sample {2.2, 3.2, 3.7, 4.4, 5.3, 5.7, 6.1, 6.4, 7.2, 7.8} with the mean of 5.2 and the standard deviation of 1.8, sample kurtosis equals $16.6962 \,/\, 1.8^4 - 3 \approx -1.41$.

Standard error in kurtosis (output row 12): A measure of how much the sample kurtosis may vary from sample to sample, assuming that the feature is normally distributed, which makes its distribution kurtosis equal 0. Given the number n of sample values, the standard error is computed as

$$\sqrt{\frac{24n\,(n-1)^2}{(n-3)(n-2)(n+3)(n+5)}}$$

This measure can tell us, for example:

- If the sample kurtosis lands within two standard errors from 0, its positive or negative sign is non-significant, may just be accidental.
- If the sample kurtosis lands outside this interval, the feature is unlikely to be normally distributed.

At least 4 values ($n \geq 4$) are required to avoid arithmetic failure. Note that the standard error is inaccurate if the feature distribution is far from normal or if the number of samples is small.

Categorical measures. Statistics that describe the sample of a categorical feature, either nominal or ordinal. We represent all categories by integers from 1 to the number of categories; we call these integers *category IDs*.

Number of categories (output row 15): The maximum category ID that occurs in the sample. Note that some categories with IDs *smaller* than this maximum ID may have no occurrences in the sample, without reducing the number of categories. However, any categories with IDs *larger* than the maximum ID with no occurrences in the sample will not be counted. Example: in sample {1, 3, 3, 3, 3, 4, 4, 5, 7, 7, 7, 7, 8, 8, 8} the number of categories is reported as 8. Category IDs 2 and 6, which have zero occurrences, are still counted; but if there is a category with ID = 9 and zero occurrences, it is not counted.

Mode (output row 16): The most frequently occurring category value. If several values share the greatest frequency of occurrence, then each of them is a mode; but here we report only the smallest of these modes. Example: in sample {1, 3, 3, 3, 3, 4, 4, 5, 7, 7, 7, 7, 8, 8, 8} the modes are 3 and 7, with 3 reported.

Computed by counting the number of occurrences for each category, then taking the smallest category ID that has the maximum count. Note that the sample modes may be different from the distribution modes, i.e. the categories whose (hypothesized) underlying probability is the maximum over all categories.

Number of modes (output row 17): The number of category values that each have the largest frequency count in the sample. Example: in sample {1, 3, 3, 3, 3, 4, 4, 5, 7, 7, 7, 7, 8, 8, 8} there are two category IDs (3 and 7) that occur the maximum count of 4 times; hence, we return 2.

Computed by counting the number of occurrences for each category, then counting how many categories have the maximum count. Note that the sample modes may be different from the distribution modes, i.e. the categories whose (hypothesized) underlying probability is the maximum over all categories.

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 convention specified in Table 1. The first 14 statistics are applicable for *scale* columns, and the last 3 statistics are applicable for categorical, i.e. nominal and ordinal, columns.

Examples

```
hadoop jar SystemDS.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

Bivariate statistics are used to quantitatively describe the association between two features, such as test their statistical (in-)dependence or measure the accuracy of one data feature predicting the other feature, in a sample. The `bivar-stats.dml` script computes common bivariate statistics, such as Pearson's correlation coefficient and Pearson's χ^2, in parallel for many pairs of data features. For a given dataset matrix, script `bivar-stats.dml` computes certain bivariate statistics for the given feature (column) pairs in the matrix. The feature types govern the exact set of statistics computed for that pair. For example, Pearson's correlation coefficient can only be computed on two quantitative (scale) features like 'Height' and 'Temperature'. It does not make sense to compute the linear correlation of two categorical attributes like 'Hair Color'.

Usage

`-f` *path/*`bivar-stats.dml` `-nvargs` `X=`*path/file* `index1=`*path/file* `index2=`*path/file* `types1=`*path/file* `types2=`*path/file* `OUTDIR=`*path*

Arguments

`X`: Location (on HDFS) to read the data matrix X whose columns are the features that we want to compare and correlate with bivariate statistics.

`index1`: Location (on HDFS) to read the single-row matrix that lists the column indices of the *first-argument* features in pairwise statistics. Its i^{th} entry (i.e. i^{th} column-cell) contains the index k of column `X[,` k`]` in the data matrix whose bivariate statistics need to be computed.

`index2`: Location (on HDFS) to read the single-row matrix that lists the column indices of the *second-argument* features in pairwise statistics. Its j^{th} entry (i.e. j^{th} column-cell) contains the index l of column `X[,` l`]` in the data matrix whose bivariate statistics need to be computed.

`types1`: Location (on HDFS) to read the single-row matrix that lists the *types* of the *first-argument* features in pairwise statistics. Its i^{th} entry (i.e. i^{th} column-cell) contains the type of column `X[,` k`]` in the data matrix, where k is the i^{th} entry in the `index1` matrix. Feature types must be encoded by integer numbers: 1 = scale, 2 = nominal, 3 = ordinal.

`types2`: Location (on HDFS) to read the single-row matrix that lists the *types* of the *second-argument* features in pairwise statistics. Its j^{th} entry (i.e. j^{th} column-cell) contains the type of column `X[,` l`]` in the data matrix, where l is the j^{th} entry in the `index2` matrix. Feature types must be encoded by integer numbers: 1 = scale, 2 = nominal, 3 = ordinal.

`OUTDIR`: Location path (on HDFS) where the output matrices with computed bivariate statistics will be stored. The matrices' file names and format are defined in Table 2.

Ouput File / Matrix	Row #	Name of Statistic
All Files	1	1-st feature column
"	2	2-nd feature column
bivar.scale.scale.stats	3	Pearson's correlation coefficient
bivar.nominal.nominal.stats	3	Pearson's χ^2
"	4	Degrees of freedom
"	5	P-value of Pearson's χ^2
"	6	Cramér's V
bivar.nominal.scale.stats	3	Eta statistic
"	4	F statistic
bivar.ordinal.ordinal.stats	3	Spearman's rank correlation coefficient

Table 2: The output matrices of `bivar-stats.dml` have one row per one bivariate statistic and one column per one pair of input features. This table lists the meaning of each matrix and each row.

Details

Script `bivar-stats.dml` takes an input matrix `X` whose columns represent the features and whose rows represent the records of a data sample. Given `X`, the script computes certain relevant bivariate statistics for specified pairs of feature columns `X[,` i`]` and `X[,` j`]`. Command-line parameters `index1` and `index2` specify the files with column pairs of interest to the user. Namely, the file given by `index1` contains the vector of the 1st-attribute column indices and the file given by `index2` has the vector of the 2nd-attribute column indices, with "1st" and "2nd" referring to their places in bivariate statistics. Note that both `index1` and `index2` files should contain a 1-row matrix of positive integers.

The bivariate statistics to be computed depend on the *types*, or *measurement levels*, of the two columns. The types for each pair are provided in the files whose locations are specified by `types1` and `types2` command-line parameters. These files are also 1-row matrices, i.e. vectors, that list the 1st-attribute and the 2nd-attribute column types in the same order as their indices in the `index1` and `index2` files. The types must be provided as per the following convention: 1 = scale, 2 = nominal, 3 = ordinal.

The script orgainizes its results into (potentially) four output matrices, one per each type combination. The types of bivariate statistics are defined using the types of the columns that were used for their arguments, with "ordinal" sometimes retrogressing to "nominal." Table 2 describes what each column in each output matrix contains. In particular, the script includes the following statistics:

- For a pair of scale (quantitative) columns, Pearson's correlation coefficient;
- For a pair of nominal columns (with finite-sized, fixed, unordered do-

mains), the Pearson's χ^2 and its p-value;

- For a pair of one scale column and one nominal column, F statistic;
- For a pair of ordinal columns (ordered domains depicting ranks), Spearman's rank correlation coefficient.

Note that, as shown in Table 2, the output matrices contain the column indices of the features involved in each statistic. 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).

Below we list all bivariate statistics computed by script `bivar-stats.dml`. The statistics are collected into several groups by the type of their input features. We refer to the two input features as v_1 and v_2 unless specified otherwise; the value pairs are $(v_{1,i}, v_{2,i})$ for $i = 1, \ldots, n$, where n is the number of rows in `X`, i.e. the sample size.

Scale-vs-scale statistics. Sample statistics that describe association between two quantitative (scale) features. A scale feature has numerical values, with the natural ordering relation.

Pearson's correlation coefficient : A measure of linear dependence between two numerical features:

$$r \;=\; \frac{\mathrm{Cov}(v_1, v_2)}{\sqrt{\mathrm{Var}\, v_1 \,\mathrm{Var}\, v_2}} \;=\; \frac{\sum_{i=1}^{n}(v_{1,i} - \bar{v}_1)(v_{2,i} - \bar{v}_2)}{\sqrt{\sum_{i=1}^{n}(v_{1,i} - \bar{v}_1)^2 \cdot \sum_{i=1}^{n}(v_{2,i} - \bar{v}_2)^2}}$$

Commonly denoted by r, correlation ranges between -1 and $+1$, reaching ± 1 when all value pairs $(v_{1,i}, v_{2,i})$ lie on the same line. Correlation near 0 means that a line is not a good way to represent the dependence between the two features; however, this does not imply independence. The sign indicates direction of the linear association: $r > 0$ ($r < 0$) if one feature tends to linearly increase (decrease) when the other feature increases. Nonlinear association, if present, may disobey this sign. Pearson's correlation coefficient is symmetric: $r(v_1, v_2) = r(v_2, v_1)$; it does not change if we transform v_1 and v_2 to $a + bv_1$ and $c + dv_2$ where a, b, c, d are constants and $b, d > 0$.

Suppose that we use simple linear regression to represent one feature given the other, say represent $v_{2,i} \approx \alpha + \beta v_{1,i}$ by selecting α and β to minimize the least-squares error $\sum_{i=1}^{n}(v_{2,i} - \alpha - \beta v_{1,i})^2$. Then the best error equals

$$\min_{\alpha,\beta} \sum_{i=1}^{n} \big(v_{2,i} - \alpha - \beta v_{1,i}\big)^2 \;=\; (1 - r^2) \sum_{i=1}^{n} \big(v_{2,i} - \bar{v}_2\big)^2$$

In other words, $1 - r^2$ is the ratio of the residual sum of squares to the total sum of squares. Hence, r^2 is an accuracy measure of the linear regression.

Nominal-vs-nominal statistics. Sample statistics that describe association between two nominal categorical features. Both features' value domains are encoded with positive integers in arbitrary order: nominal features do not order their value domains.

Pearson's χ^2 : A measure of how much the frequencies of value pairs of two categorical features deviate from statistical independence. Under independence, the probability of every value pair must equal the product of probabilities of each value in the pair: $\text{Prob}[a, b] - \text{Prob}[a]\ \text{Prob}[b] = 0$. But we do not know these (hypothesized) probabilities; we only know the sample frequency counts. Let $n_{a,b}$ be the frequency count of pair (a, b), let n_a and n_b be the frequency counts of a alone and of b alone. Under independence, difference $n_{a,b}/n - (n_a/n)(n_b/n)$ is unlikely to be exactly 0 due to sample randomness, yet it is unlikely to be too far from 0. For some pairs (a, b) it may deviate from 0 farther than for other pairs. Pearson's χ^2 is an aggregate measure that combines squares of these differences across all value pairs:

$$\chi^2 \;=\; \sum_{a,\,b} \Big(\frac{n_a n_b}{n}\Big)^{-1} \Big(n_{a,b} - \frac{n_a n_b}{n}\Big)^2 \;=\; \sum_{a,\,b} \frac{(O_{a,b} - E_{a,b})^2}{E_{a,b}}$$

where $O_{a,b} = n_{a,b}$ are the *observed* frequencies and $E_{a,b} = (n_a n_b)/n$ are the *expected* frequencies for all pairs (a, b). Under independence (plus other standard assumptions) the sample χ^2 closely follows a well-known distribution, making it a basis for statistical tests for independence, see *P-value of Pearson's* χ^2 for details. Note that Pearson's χ^2 does *not* measure the strength of dependence: even very weak dependence may result in a significant deviation from independence if the counts are large enough. Use Cramér's V instead to measure the strength of dependence.

Degrees of freedom : An integer parameter required for the interpretation of Pearson's χ^2 measure. Under independence (plus other standard assumptions) the sample χ^2 statistic is approximately distributed as the sum of d squares of independent normal random variables with mean 0 and variance 1, where d is this integer parameter. For a pair of categorical features such that the 1^{st} feature has k_1 categories and the 2^{nd} feature has k_2 categories, the number of degrees of freedom is $d = (k_1 - 1)(k_2 - 1)$.

P-value of Pearson's χ^2 : A measure of how likely we would observe the current frequencies of value pairs of two categorical features assuming their statistical independence. More precisely, it computes the probability that the sum of d squares of independent normal random variables with mean 0 and variance 1 (called the χ^2 distribution with d degrees of freedom) generates a value at least as large as the current sample Pearson's χ^2. The d parameter is *degrees of freedom*, see above. Under independence (plus other standard assumptions) the sample Pearson's χ^2 closely follows the χ^2 distribution and is unlikely to land very far into its tail. On the other hand, if the two features are dependent, their sample Pearson's χ^2 becomes

arbitrarily large as $n \to \infty$ and lands extremely far into the tail of the χ^2 distribution given a large enough data sample. P-value of Pearson's χ^2 returns the tail "weight" on the right-hand side of Pearson's χ^2:

$$P \;=\; \text{Prob}\left[r \geq \text{Pearson's } \chi^2 \;\middle|\; r \sim \text{the } \chi^2 \text{ distribution}\right]$$

As any probability, P ranges between 0 and 1. If $P \leq 0.05$, the dependence between the two features may be considered statistically significant (i.e. their independence is considered statistically ruled out). For highly dependent features, it is not unusual to have $P \leq 10^{-20}$ or less, in which case our script will simply return $P = 0$. Independent features should have their $P \geq 0.05$ in about 95% cases.

Cramér's V : A measure for the strength of association, i.e. of statistical dependence, between two categorical features, conceptually similar to Pearson's correlation coefficient. It divides the observed Pearson's χ^2 by the maximum possible $\chi^2_{\max}$ given n and the number k_1, k_2 of categories in each feature, then takes the square root. Thus, Cramér's V ranges from 0 to 1, where 0 implies no association and 1 implies the maximum possible association (one-to-one correspondence) between the two features. See *Pearson's* χ^2 for the computation of χ^2; its maximum $= n \cdot \min\{k_1 - 1, k_2 - 1\}$ where the 1^{st} feature has k_1 categories and the 2^{nd} feature has k_2 categories [**?**], so

$$\text{Cramér's } V \;=\; \sqrt{\frac{\text{Pearson's } \chi^2}{n \cdot \min\{k_1 - 1, k_2 - 1\}}}$$

As opposed to P-value of Pearson's χ^2, which goes to 0 (rapidly) as the features' dependence increases, Cramér's V goes towards 1 (slowly) as the dependence increases. Both Pearson's χ^2 and P-value of Pearson's χ^2 are very sensitive to n, but in Cramér's V this is mitigated by taking the ratio.

Nominal-vs-scale statistics. Sample statistics that describe association between a categorical feature (order ignored) and a quantitative (scale) feature. The values of the categorical feature must be coded as positive integers.

Eta statistic : A measure for the strength of association (statistical dependence) between a nominal feature and a scale feature, conceptually similar to Pearson's correlation coefficient. Ranges from 0 to 1, approaching 0 when there is no association and approaching 1 when there is a strong association. The nominal feature, treated as the independent variable, is assumed to have relatively few possible values, all with large frequency counts. The scale feature is treated as the dependent variable. Denoting the nominal feature by x and the scale feature by y, we have:

$$\eta^2 \;=\; 1 - \frac{\sum_{i=1}^{n} \big(y_i - \hat{y}[x_i]\big)^2}{\sum_{i=1}^{n} (y_i - \bar{y})^2}, \quad \text{where} \quad \hat{y}[x] = \frac{1}{\text{freq}(x)} \sum_{i=1}^{n} \begin{cases} y_i & \text{if } x_i = x \\ 0 & \text{otherwise} \end{cases}$$

and $\bar{y} = (1/n)\sum_{i=1}^{n} y_i$ is the mean. Value $\hat{y}[x]$ is the average of y_i among all records where $x_i = x$; it can also be viewed as the "predictor" of y given x. Then $\sum_{i=1}^{n}(y_i - \hat{y}[x_i])^2$ is the residual error sum-of-squares and $\sum_{i=1}^{n}(y_i - \bar{y})^2$ is the total sum-of-squares for y. Hence, η^2 measures the accuracy of predicting y with x, just like the "R-squared" statistic measures the accuracy of linear regression. Our output η is the square root of η^2.

F statistic : A measure of how much the values of the scale feature, denoted here by y, deviate from statistical independence on the nominal feature, denoted by x. The same measure appears in the one-way analysis of variance (ANOVA). Like Pearson's χ^2, F statistic is used to test the hypothesis that y is independent from x, given the following assumptions:

- The scale feature y has approximately normal distribution whose mean may depend only on x and variance is the same for all x;
- The nominal feature x has relatively small value domain with large frequency counts, the x_i-values are treated as fixed (non-random);
- All records are sampled independently of each other.

To compute F statistic, we first compute $\hat{y}[x]$ as the average of y_i among all records where $x_i = x$. These $\hat{y}[x]$ can be viewed as "predictors" of y given x; if y is independent on x, they should "predict" only the global mean $\bar{y}$. Then we form two sums-of-squares:

- *Residual* sum-of-squares of the "predictor" accuracy: $y_i - \hat{y}[x_i]$;
- *Explained* sum-of-squares of the "predictor" variability: $\hat{y}[x_i] - \bar{y}$.

F statistic is the ratio of the explained sum-of-squares to the residual sum-of-squares, each divided by their corresponding degrees of freedom:

$$F \;=\; \frac{\sum_x \mathrm{freq}(x)\,\big(\hat{y}[x] - \bar{y}\big)^2 \,/\, (k-1)}{\sum_{i=1}^{n}\big(y_i - \hat{y}[x_i]\big)^2 \,/\, (n-k)} \;=\; \frac{n-k}{k-1}\cdot\frac{\eta^2}{1-\eta^2}$$

Here k is the domain size of the nominal feature x. The k "predictors" lose 1 freedom due to their linear dependence with $\bar{y}$; similarly, the n y_i-s lose k freedoms due to the "predictors".

The statistic can test if the independence hypothesis of y from x is reasonable; more generally (with relaxed normality assumptions) it can test the hypothesis that *the mean* of y among records with a given x is the same for all x. Under this hypothesis F statistic has, or approximates, the $F(k-1, n-k)$-distribution. But if the mean of y given x depends on x, F statistic becomes arbitrarily large as $n \to \infty$ (with k fixed) and lands extremely far into the tail of the $F(k-1, n-k)$-distribution given a large enough data sample.

Ordinal-vs-ordinal statistics. Sample statistics that describe association between two ordinal categorical features. Both features' value domains are encoded with positive integers, so that the natural order of the integers coincides with the order in each value domain.

Spearman's rank correlation coefficient : A measure for the strength of association (statistical dependence) between two ordinal features, conceptually similar to Pearson's correlation coefficient. Specifically, it is Pearson's correlation coefficient applied to the feature vectors in which all values are replaced by their ranks, i.e. their positions if the vector is sorted. The ranks of identical (duplicate) values are replaced with their average rank. For example, in vector $(15, 11, 26, 15, 8)$ the value "15" occurs twice with ranks 3 and 4 per the sorted order $(8_1, 11_2, 15_3, 15_4, 26_5)$; so, both values are assigned their average rank of $3.5 = (3+4)\,/\,2$ and the vector is replaced by (3.5, 2, 5, 3.5, 1).

Our implementation of Spearman's rank correlation coefficient is geared towards features having small value domains and large counts for the values. Given the two input vectors, we form a contingency table T of pairwise frequency counts, as well as a vector of frequency counts for each feature: f_1 and f_2. Here in $T_{i,j}$, $f_{1,i}$, $f_{2,j}$ indices i and j refer to the order-preserving integer encoding of the feature values. We use prefix sums over f_1 and f_2 to compute the values' average ranks: $r_{1,i} = \sum_{j=1}^{i-1} f_{1,j} + (f_{1,i}+1)/2$, and analogously for r_2. Finally, we compute rank variances for r_1, r_2 weighted by counts f_1, f_2 and their covariance weighted by T, before applying the standard formula for Pearson's correlation coefficient:

$$\rho \;=\; \frac{\mathrm{Cov}_T(r_1, r_2)}{\sqrt{\mathrm{Var}_{f_1}(r_1)\,\mathrm{Var}_{f_2}(r_2)}} \;=\; \frac{\sum_{i,j} T_{i,j}(r_{1,i} - \bar{r}_1)(r_{2,j} - \bar{r}_2)}{\sqrt{\sum_i f_{1,i}(r_{1,i} - \bar{r}_1)^2 \cdot \sum_j f_{2,j}(r_{2,j} - \bar{r}_2)^2}}$$

where $\bar{r}_1 = \sum_i r_{1,i} f_{1,i}/n$, analogously for $\bar{r}_2$. The value of ρ lies between -1 and $+1$, with sign indicating the prevalent direction of the association: $\rho > 0$ ($\rho < 0$) means that one feature tends to increase (decrease) when the other feature increases. The correlation becomes 1 when the two features are monotonically related.

Returns

A collection of (potentially) 4 matrices. Each matrix contains bivariate statistics that resulted from a different combination of feature types. There is one matrix for scale-scale statistics (which includes Pearson's correlation coefficient), one for nominal-nominal statistics (includes Pearson's χ^2), one for nominal-scale statistics (includes F statistic) and one for ordinal-ordinal statistics (includes Spearman's rank correlation coefficient). If any of these matrices is not produced, then no pair of columns required the corresponding type combination. See Table 2 for the matrix naming and format details.

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.

Examples

```
hadoop jar SystemDS.jar -f bivar-stats.dml -nvargs
   X=/user/biadmin/X.mtx 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 SystemDS 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 SystemDS.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 SystemDS.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 [**?**] 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:

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

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

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:

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

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 SystemDS 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 [**?**]. 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 [**?**] 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 [**?**]. 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 SystemDS.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 SystemDS.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 SystemDS.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
                          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.
- 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 SystemDS.jar -f m-svm.dml -nvargs X=/user/biadmin/X.mtx
```

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

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

hadoop jar SystemDS.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.

2.4 Decision Trees

Description

Decision tree (for classification) is a classifier that is considered more interpretable than other statistical classifiers. This implementation is well-suited to handle large-scale data and builds a (binary) decision tree in parallel.

Usage

`-f` *path/*`decision-tree.dml -nvargs X=`*path/file* `Y=`*path/file* `R=`*path/file* `bins=`*integer* `depth=`*integer* `num_leaf=`*integer* `num_samples=`*integer* `impurity=`*Gini*|*entropy* `M=`*path/file* `O=`*path/file* `S_map=`*path/file* `C_map=`*path/file* `fmt=`*format*

Usage: Prediction

`-f` *path/*`decision-tree-predict.dml -nvargs X=`*path/file* `Y=`*path/file* `R=`*path/file* `M=`*path/file* `P=`*path/file* `A=`*path/file* `CM=`*path/file* `fmt=`*format*

Arguments

`X:` Location (on HDFS) to read the matrix of feature vectors; each row constitutes one feature vector. Note that categorical features in X need to be both recoded and dummy coded.

`Y:` Location (on HDFS) to read the matrix of (categorical) labels that correspond to feature vectors in X. Note that class labels are assumed to be both recoded and dummy coded. This argument is optional for prediction.

`R:` (default: `" "`) Location (on HDFS) to read matrix R which for each feature in X contains column-ids (first column), start indices (second column), and end indices (third column). If R is not provided by default all features are assumed to be continuous-valued.

`bins:` (default: `20`) Number of thresholds to choose for each continuous-valued feature (determined by equi-height binning).

`depth:` (default: `25`) Maximum depth of the learned tree

`num_leaf:` (default: `10`) Parameter that controls pruning. The tree is not expanded if a node receives less than `num_leaf` training examples.

`num_samples:` (default: `3000`) Parameter that decides when to switch to in-memory building of subtrees. If a node v receives less than `num_samples` training examples then this implementation switches to an in-memory subtree building procedure to build the subtree under v in its entirety.

`impurity:` (default: `"Gini"`) Impurity measure used at internal nodes of the tree for selecting which features to split on. Possible value are entropy or Gini.

`M:` Location (on HDFS) to write matrix M containing the learned decision tree (see below for the schema)

`O:` (default: `" "`) Location (on HDFS) to store the training accuracy (%). Note that this argument is optional.

`A:` (default: `" "`) Location (on HDFS) to store the testing accuracy (%) from a held-out test set during prediction. Note that this argument is optional.

`P:` Location (on HDFS) to store predictions for a held-out test set

`CM:` (default: `" "`) Location (on HDFS) to store the confusion matrix computed using a held-out test set. Note that this argument is optional.

`S_map:` (default: `" "`) Location (on HDFS) to write the mappings from the continuous-valued feature-ids to the global feature-ids in X (see below for details). Note that this argument is optional.

`C_map:` (default: `" "`) Location (on HDFS) to write the mappings from the categorical feature-ids to the global feature-ids in X (see below for details). Note that this argument is optional.

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

Details

Decision trees [**?**] are simple models of classification that, due to their structure, are easy to interpret. Given an example feature vector, each node in the learned tree runs a simple test on it. Based on the result of the test, the example is either diverted to the left subtree or to the right subtree. Once the example reaches a leaf, then the label stored at the leaf is returned as the prediction for the example.

Building a decision tree from a fully labeled training set entails choosing appropriate splitting tests for each internal node in the tree and this is usually performed in a top-down manner. The splitting test (denoted by s) requires first choosing a feature j and depending on the type of j, either a threshold σ, in case j is continuous-valued, or a subset of values $S \subseteq \text{Dom}(j)$ where $\text{Dom}(j)$ denotes domain of j, in case it is categorical. For continuous-valued features the test is thus of form $x_j < \sigma$ and for categorical features it is of form $x_j \in S$, where x_j denotes the jth feature value of feature vector x. One way to determine which test to include, is to compare impurities of the tree nodes induced by the test. The *node impurity* measures the homogeneity of the labels at the node. This implementation supports two commonly used impurity measures (denoted by $\mathcal{I}$): *Entropy* $\mathcal{E} = \sum_{i=1}^{C} -f_i \log f_i$, as well as *Gini impurity* $\mathcal{G} = \sum_{i=1}^{C} f_i(1 - f_i)$, where C denotes the number of unique labels and f_i is the frequency of label i. Once the impurity at the tree nodes has been obtained, the *best split* is chosen from a set of possible splits that maximizes the *information gain* at the node, i.e., $\arg\max_s \mathcal{IG}(X, s)$, where $\mathcal{IG}(X, s)$ denotes the information gain when the splitting test s partitions the feature matrix X. Assuming that s partitions X that contains N feature vectors into X_{left} and X_{right} each including N_{left} and N_{right} feature vectors, respectively, $\mathcal{IG}(X, s)$ is given by

$$\mathcal{IG}(X, s) = \mathcal{I}(X) - \frac{N_{\text{left}}}{N}\mathcal{I}(X_{\text{left}}) - \frac{N_{\text{right}}}{N}\mathcal{I}(X_{\text{right}}),$$

where $\mathcal{I} \in \{\mathcal{E}, \mathcal{G}\}$. In the following we discuss the implementation details specific to `decision-tree.dml`.

Input format. In general implementations of the decision tree algorithm do not require categorical features to be dummy coded. For improved efficiency and reducing the training time, our implementation however assumes dummy coded categorical features and dummy coded class labels.

Tree construction. Learning a decision tree on large-scale data has received some attention in the literature. The current implementation includes logic for choosing tests for multiple nodes that belong to the same level in the decision tree in parallel (breadth-first expansion) and for building entire subtrees under multiple nodes in parallel (depth-first subtree building). Empirically it has been demonstrated that it is advantageous to perform breadth-first expansion for the nodes belonging to the top levels of the tree and to perform depth-first subtree building for nodes belonging to the lower levels of the tree [**?**]. The parameter `num_samples` controls when we switch to depth-first subtree building. Any node in the decision tree that receives $\leq$ `num_samples` training examples, the subtree under it is built in its entirety in one shot.

Stopping rule and pruning. The splitting of data at the internal nodes stops when at least one the following criteria is satisfied:

- the depth of the internal node reaches the input parameter `depth` controlling the maximum depth of the learned tree, or
- no candidate split achieves information gain.

This implementation also allows for some automated pruning via the argument `num_leaf`. If a node receives $\leq$ `num_leaf` training examples, then a leaf is built in its place.

Continuous-valued features. For a continuous-valued feature j the number of candidate thresholds σ to choose from is of the order of the number of examples present in the training set. Since for large-scale data this can result in a large number of candidate thresholds, the user can limit this number via the arguments `bins` which controls the number of candidate thresholds considered for each continuous-valued feature. For each continuous-valued feature, the implementation computes an equi-height histogram to generate one candidate threshold per equi-height bin.

Categorical features. In order to determine the best value subset to split on in the case of categorical features, this implementation greedily includes values from the feature's domain until the information gain stops improving. In particular, for a categorical feature j the $|Dom(j)|$ feature values are sorted by impurity and the resulting split candidates $|Dom(j)| - 1$ are examined; the sequence of feature values which results in the maximum information gain is then selected.

Description of the model. The learned decision tree is represented in a matrix M that contains at least 6 rows. Each column in the matrix contains the parameters relevant to a single node in the tree. Note that for building the tree model, our implementation splits the feature matrix X into X_{cont} containing continuous-valued features and X_{cat} containing categorical features. In the following, the continuous-valued (resp. categorical) feature-ids correspond to the

indices of the features in X_{cont} (resp. X_{cat}). Moreover, we refer to an internal node as a continuous-valued (categorical) node if the feature that this nodes looks at is continuous-valued (categorical). Below is a description of what each row in the matrix contains.

- Row 1: stores the node-ids. These ids correspond to the node-ids in a complete binary tree.
- Row 2: for internal nodes stores the offsets (the number of columns) in M to the left child, and otherwise 0.
- Row 3: stores the feature index of the feature (id of a continuous-valued feature in X_{cont} if the feature is continuous-valued or id of a categorical feature in X_{cat} if the feature is categorical) that this node looks at if the node is an internal node, otherwise 0.
- Row 4: store the type of the feature that this node looks at if the node is an internal node: 1 for continuous-valued and 2 for categorical features, otherwise the label this leaf node is supposed to predict.
- Row 5: for the internal nodes contains 1 if the feature chosen for the node is continuous-valued, or the size of the subset of values used for splitting at the node stored in rows 6,7,... if the feature chosen for the node is categorical. For the leaf nodes, Row 5 contains the number of misclassified training examples reaching at this node.
- Row 6,7,...: for the internal nodes, row 6 stores the threshold to which the example's feature value is compared if the feature chosen for this node is continuous-valued, otherwise if the feature chosen for this node is categorical rows 6,7,... store the value subset chosen for the node. For the leaf nodes, row 6 contains 1 if the node is impure and the number of training examples at the node is greater than `num_leaf`, otherwise 0.

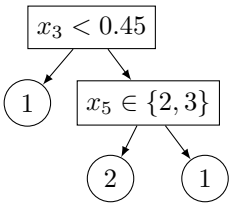
As an example, Figure 2 shows a decision tree with 5 nodes and its matrix representation.

Returns

The matrix corresponding to the learned model as well as the training accuracy (if requested) is written to a file in the format specified. See details where the structure of the model matrix is described. Recall that in our implementation X is split into X_{cont} and X_{cat}. If requested, the mappings of the continuous-valued feature-ids in X_{cont} (stored at `S_map`) and the categorical feature-ids in X_{cat} (stored at `C_map`) to the global feature-ids in X will be provided. Depending on what arguments are provided during invocation, the `decision-tree-predict.dml` script may compute one or more of predictions, accuracy and confusion matrix in the requested output format.

Examples

```
hadoop jar SystemDS.jar -f decision-tree.dml -nvargs
  X=/user/biadmin/X.mtx Y=/user/biadmin/Y.mtx
```



(a)

	Col 1	Col 2	Col 3	Col 4	Col 5
Row 1	1	2	3	6	7
Row 2	1	0	1	0	0
Row 3	3	5	0	0	0
Row 4	1	1	2	2	1
Row 5	1	0	2	0	0
Row 6	0.45	0	2	0	0
Row 7			3		

(b)

Figure 2: (a) An example tree and its (b) matrix representation. x denotes an example and x_j is the value of the jth continuous-valued (resp. categorical) feature in X_{cont} (resp. X_{cat}). In this example all leaf nodes are pure and no training example is misclassified.

```
  R=/user/biadmin/R.csv M=/user/biadmin/model.csv bins=20
  depth=25 num_leaf=10 num_samples=3000 impurity=Gini fmt=csv
```

To compute predictions:

```
hadoop jar SystemDS.jar -f decision-tree-predict.dml
  -nvargs X=/user/biadmin/X.mtx Y=/user/biadmin/Y.mtx
  R=/user/biadmin/R.csv M=/user/biadmin/model.csv
  P=/user/biadmin/predictions.csv A=/user/biadmin/accuracy.csv
  CM=/user/biadmin/confusion.csv fmt=csv
```

2.5 Random Forests

Description

Random forest is one of the most successful machine learning methods for classification and regression. It is an ensemble learning method that creates a model composed of a set of tree models. This implementation is well-suited to handle large-scale data and builds a random forest model for classification in parallel.

Usage

`-f` *path/*`random-forest.dml -nvargs X=`*path/file* `Y=`*path/file* `R=`*path/file* `bins=`*integer* `depth=`*integer* `num_leaf=`*integer* `num_samples=`*integer* `num_trees=`*integer* `subsamp_rate=`*double* `feature_subset=`*double* `impurity=`*Gini*|*entropy* `M=`*path/file* `C=`*path/file* `S_map=`*path/file* `C_map=`*path/file* `fmt=`*format*

Usage: Prediction

`-f` *path/*`random-forest-predict.dml -nvargs X=`*path/file* `Y=`*path/file* `R=`*path/file* `M=`*path/file* `C=`*path/file* `P=`*path/file* `A=`*path/file* `OOB=`*path/file* `CM=`*path/file* `fmt=`*format*

Arguments

`X`: Location (on HDFS) to read the matrix of feature vectors; each row constitutes one feature vector. Note that categorical features in X need to be both recoded and dummy coded.

`Y`: Location (on HDFS) to read the matrix of (categorical) labels that correspond to feature vectors in X. Note that classes are assumed to be both recoded and dummy coded. This argument is optional for prediction.

`R`: (default: `" "`) Location (on HDFS) to read matrix R which for each feature in X contains column-ids (first column), start indices (second column), and end indices (third column). If R is not provided by default all features are assumed to be continuous-valued.

`bins`: (default: `20`) Number of thresholds to choose for each continuous-valued feature (determined by equi-height binning).

`depth`: (default: `25`) Maximum depth of the learned trees in the random forest model

`num_leaf`: (default: `10`) Parameter that controls pruning. The tree is not expanded if a node receives less than `num_leaf` training examples.

`num_samples`: (default: `3000`) Parameter that decides when to switch to in-memory building of the subtrees in each tree of the random forest model. If a node v receives less than `num_samples` training examples then this implementation switches to an in-memory subtree building procedure to build the subtree under v in its entirety.

`num_trees`: (default: `10`) Number of trees to be learned in the random forest model

`subsamp_rate`: (default: `1.0`) Parameter controlling the size of each tree in the random forest model; samples are selected from a Poisson distribution with parameter `subsamp_rate`.

`feature_subset`: (default: `0.5`) Parameter that controls the number of feature used as candidates for splitting at each tree node as a power of the number of features in the data, i.e., assuming the training set has D features $D^{\texttt{feature\_subset}}$ are used at each tree node.

`impurity`: (default: `"Gini"`) Impurity measure used at internal nodes of the trees in the random forest model for selecting which features to split on. Possible value are entropy or Gini.

`M`: Location (on HDFS) to write matrix M containing the learned random forest (see Section 2.4 and below for the schema)

`C`: (default: `" "`) Location (on HDFS) to store the number of counts (generated according to a Poisson distribution with parameter `subsamp_rate`) for each feature vector. Note that this argument is optional. If Out-Of-Bag (OOB) error estimate needs to be computed this parameter is passed as input to `random-forest-predict.dml`.

`A:` (default: `" "`) Location (on HDFS) to store the testing accuracy (%) from a held-out test set during prediction. Note that this argument is optional.

`OOB:` (default: `" "`) Location (on HDFS) to store the Out-Of-Bag (OOB) error estimate of the training set. Note that the matrix of sample counts (stored at `C`) needs to be provided for computing OOB error estimate. Note that this argument is optional.

`P:` Location (on HDFS) to store predictions for a held-out test set

`CM:` (default: `" "`) Location (on HDFS) to store the confusion matrix computed using a held-out test set. Note that this argument is optional.

`S_map:` (default: `" "`) Location (on HDFS) to write the mappings from the continuous-valued feature-ids to the global feature-ids in X (see below for details). Note that this argument is optional.

`C_map:` (default: `" "`) Location (on HDFS) to write the mappings from the categorical feature-ids to the global feature-ids in X (see below for details). Note that this argument is optional.

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

Details

Random forests [**?**] are learning algorithms for ensembles of decision trees. The main idea is to build a number of decision trees on bootstrapped training samples, i.e., by taking repeatedly samples from a (single) training set. Moreover, instead of considering all the features when building the trees only a random subset of the features—typically $\approx \sqrt{D}$, where D is the number of features—is chosen each time a split test at a tree node is performed. This procedure *decorrelates* the trees and makes it less prone to overfitting. To build decision trees we utilize the techniques discussed in Section 2.4 proposed in [**?**]; the implementation details are similar to those of the decision trees script. Below we review some features of our implementation which differ from `decision-tree.dml`.

Bootstrapped sampling. Each decision tree is fitted to a bootstrapped training set sampled with replacement (WR). To improve efficiency, we generate N sample counts according to a Poisson distribution with parameter `subsamp_rate`, where N denotes the total number of training points. These sample counts approximate WR sampling when N is large enough and are generated upfront for each decision tree.

Bagging. Decision trees suffer from *high variance* resulting in different models whenever trained on a random subsets of the data points. *Bagging* is a general-purpose method to reduce the variance of a statistical learning method like decision trees. In the context of decision trees (for classification), for a given test feature vector the prediction is computed by taking a *majority vote*: the overall prediction is the most commonly occurring class among all the tree predictions.

Out-Of-Bag error estimation. Note that each bagged tree in a random forest model is trained on a subset (around $\frac{2}{3}$) of the observations (i.e., feature

vectors). The remaining ($\frac{1}{3}$ of the) observations not used for training is called the *Out-Of-Bag* (OOB) observations. This gives us a straightforward way to estimate the test error: to predict the class label of each test observation i we use the trees in which i was OOB. Our `random-forest-predict.dml` script provides the OOB error estimate for a given training set if requested.

Description of the model. Similar to decision trees, the learned random forest model is presented in a matrix M with at least 7 rows. The information stored in the model is similar to that of decision trees with the difference that the tree-ids are stored in the second row and rows $2, 3, \ldots$ from the decision tree model are shifted by one. See Section 2.4 for a description of the model.

Returns

The matrix corresponding to the learned model is written to a file in the format specified. See Section 2.4 where the details about the structure of the model matrix is described. Similar to `decision-tree.dml`, X is split into X_{cont} and X_{cat}. If requested, the mappings of the continuous feature-ids in X_{cont} (stored at `S_map`) as well as the categorical feature-ids in X_{cat} (stored at `C_map`) to the global feature-ids in X will be provided. The `random-forest-predict.dml` script may compute one or more of predictions, accuracy, confusion matrix, and OOB error estimate in the requested output format depending on the input arguments used.

Examples

```
hadoop jar SystemDS.jar -f random-forest.dml -nvargs
   X=/user/biadmin/X.mtx Y=/user/biadmin/Y.mtx
   R=/user/biadmin/R.csv M=/user/biadmin/model.csv bins=20
   depth=25 num_leaf=10 num_samples=3000 num_trees=10
   impurity=Gini fmt=csv
```

To compute predictions:

```
hadoop jar SystemDS.jar -f random-forest-predict.dml
   -nvargs X=/user/biadmin/X.mtx Y=/user/biadmin/Y.mtx
   R=/user/biadmin/R.csv M=/user/biadmin/model.csv
   P=/user/biadmin/predictions.csv A=/user/biadmin/accuracy.csv
   CM=/user/biadmin/confusion.csv fmt=csv
```

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:

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

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 [**?**].

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:

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

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 SystemDS 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

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.

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 SystemDS 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 [**?**], 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.

   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 SystemDS.jar -f Kmeans.dml -nvargs
   X=/user/biadmin/X.mtx k=5 C=/user/biadmin/centroids.mtx
   fmt=csv
hadoop jar SystemDS.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 SystemDS.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 SystemDS.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 SystemDS.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*

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.

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 SystemDS Language Reference for details.

Details

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.

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 $\exp\left(-(y_i - \mu_i)^2/(2\sigma^2)\right)$, 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).

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

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

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 $X^T X \beta = X^T Y$.

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

We implemented two scripts for solving equation (11): one is a "direct solver" that computes A and then solves $A\beta = [X,\, 1]^T Y$ 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 [**?**]. 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 $r = A\beta - [X,\, 1]^T Y$ 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 SystemDS.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 SystemDS.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 Stepwise Linear Regression

Description

Our stepwise linear regression script selects a linear model based on the Akaike information criterion (AIC): the model that gives rise to the lowest AIC is computed.

Usage

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

Arguments

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

`Y:` Location (on HDFS) to read the 1-column matrix of response values

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

`S:` (default: `" "`) Location (on HDFS) to store the selected feature-ids in the order as computed by the algorithm; by default the selected feature-ids are forwarded to the standard output.

`O:` (default: `" "`) Location (on HDFS) to store the CSV-file of summary statistics defined in Table 7; by default the summary statistics are forwarded to the standard output.

`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

`thr:` (default: `0.01`) Threshold to stop the algorithm: if the decrease in the value of the AIC falls below `thr` no further features are being checked and the algorithm stops.

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

Details

Stepwise linear regression iteratively selects predictive variables in an automated procedure. Currently, our implementation supports forward selection: starting from an empty model (without any variable) the algorithm examines the addition of each variable based on the AIC as a model comparison criterion. The AIC is defined as

$$AIC = -2 \log L + 2edf, \tag{12}$$

where L denotes the likelihood of the fitted model and edf is the equivalent degrees of freedom, i.e., the number of estimated parameters. This procedure is repeated until including no additional variable improves the model by a certain threshold specified in the input parameter `thr`.

For fitting a model in each iteration we use the "direct solve" method as in the script `LinearRegDS.dml` discussed in Section 4.1.

Returns

Similar to the outputs from `LinearRegDS.dml` the stepwise linear regression script computes the estimated regression coefficients and stores them in matrix B on HDFS. The format of matrix B is identical to the one produced by the scripts for linear regression (see Section 4.1). Additionally, `StepLinearRegDS.dml` outputs the variable indices (stored in the 1-column matrix S) in the order they have been selected by the algorithm, i.e., ith entry in matrix S corresponds to the variable which improves the AIC the most in ith iteration. If the model with the lowest AIC includes no variables matrix S will be empty (contains one 0). Moreover, the estimated summary statistics as defined in Table 7 are printed out or stored in a file (if requested). In the case where an empty model achieves the best AIC these statistics will not be produced.

Examples

```
hadoop jar SystemDS.jar -f StepLinearRegDS.dml
   -nvargs X=/user/biadmin/X.mtx Y=/user/biadmin/Y.mtx
   B=/user/biadmin/B.mtx S=/user/biadmin/selected.csv
   O=/user/biadmin/stats.csv icpt=2 thr=0.05 fmt=csv
```

4.3 Generalized Linear Models (GLM)

Description

Generalized Linear Models [**?**, **?**, **?**] 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:

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

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 SystemDS Language Reference for details.

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

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

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

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

`vpow:` (default: `0.0`) When `dfam=1`, this provides the q in $\mathrm{Var}(y) = a\mu^q$, 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. All other y-values are treated as the "Yes" label. For example, `yneg=-1.0` may be used when $y \in \{-1, 1\}$; either `yneg=1.0` or `yneg=2.0` may be used when $y \in \{1, 2\}$.

`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

$$Y \sim \text{Prob}[y \mid \mu] = \exp\left(\frac{y\theta - b(\theta)}{a} + c(y, a)\right), \text{ where } \mu = \mathrm{E}(Y) = b'(\theta). \tag{13}$$

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.

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.

Rather than specifying the canonical link, GLM distributions are commonly defined by their variance $\mathrm{Var}(y)$ as the function of the mean μ. It can be shown from Eq. (13) that $\mathrm{Var}(y) = a\,b''(\theta) = a\,b''(b'^{-1}(\mu))$. For example, 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 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 $y_i \in \{y_{\mathrm{neg}}, y_{\mathrm{pos}}\}$ with y_{neg} from the input parameter yneg and with $y_{\mathrm{pos}} \neq y_{\mathrm{neg}}$. 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-

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.

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 (13); note that a and $c(y, a)$ are constant w.r.t. β 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:

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

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:

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

The conjugate gradient algorithm closely follows Algorithm 7.2 on page 171 of [**?**]. The trust region size δ is initialized as $0.5\sqrt{m} \,/\, \max_i \|x_i\|_2$ and updated

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.")

as described in [**?**]. 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 $2|\Delta f(z;\beta)| < (D_1(\beta)+0.1)\varepsilon$ where $\varepsilon > 0$ is a tolerance supplied by the user via `tol`, and $D_1(\beta)$ is the unit-dispersion deviance estimated as

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

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. 13 using Pearson residuals

$$\hat{a} \;=\; \frac{1}{n-m}\cdot\sum_{i=1}^{n}\frac{(y_i-\mu_i)^2}{v(\mu_i)} \tag{14}$$

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 (14) 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`"

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.)

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.

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 SystemDS.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.

4.4 Stepwise Generalized Linear Regression

Description

Our stepwise generalized linear regression script selects a model based on the Akaike information criterion (AIC): the model that gives rise to the lowest AIC is provided. Note that currently only the Bernoulli distribution family is supported (see below for details).

Usage

`-f` *path/*`StepGLM.dml` `-nvargs` `X=`*path/file* `Y=`*path/file* `B=`*path/file* `S=`*path/file* `O=`*path/file* `link=`*int* `yneg=`*double* `icpt=`*int* `tol=`*double* `disp=`*double* `moi=`*int* `mii=`*int* `thr=`*double* `fmt=`*format*

Arguments

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

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

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

`S:` (default: `" "`) Location (on HDFS) to store the selected feature-ids in the order as computed by the algorithm, by default it is standard output.

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

`link:` (default: `2`) Link function code to determine the link function $\eta = g(\mu)$, see Table 11; currently the following link functions are supported:
`1` = log, `2` = logit, `3` = probit, `4` = cloglog.

`yneg:` (default: `0.0`) Response value for Bernoulli "No" label, usually 0.0 or -1.0

`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

`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

`thr:` (default: `0.01`) Threshold to stop the algorithm: if the decrease in the value of the AIC falls below `thr` no further features are being checked and the algorithm stops.

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

Details

Similar to `StepLinearRegDS.dml` our stepwise GLM script builds a model by iteratively selecting predictive variables using a forward selection strategy based on the AIC (12). Note that currently only the Bernoulli distribution family (`fam=2` in Table 11) together with the following link functions are supported: log, logit, probit, and cloglog (`link` $\in \{1, 2, 3, 4\}$ in Table 11).

Returns

Similar to the outputs from `GLM.dml` the stepwise GLM script computes the estimated regression coefficients and stores them in matrix B on HDFS; matrix B follows the same format as the one produced by `GLM.dml` (see Section 4.3). Additionally, `StepGLM.dml` outputs the variable indices (stored in the 1-column matrix S) in the order they have been selected by the algorithm, i.e., ith entry in matrix S stores the variable which improves the AIC the most in ith iteration. If the model with the lowest AIC includes no variables matrix S will be empty. Moreover, the estimated summary statistics as defined in Table 9 are printed out or stored in a file on HDFS (if requested); these statistics will be provided only if the selected model is nonempty, i.e., contains at least one variable.

Examples

```
hadoop jar SystemDS.jar -f StepGLM.dml -nvargs
   X=/user/biadmin/X.mtx Y=/user/biadmin/Y.mtx
   B=/user/biadmin/B.mtx S=/user/biadmin/selected.csv
   O=/user/biadmin/stats.csv link=2 yneg=-1.0 icpt=2 tol=0.000001
   moi=100 mii=10 thr=0.05 fmt=csv
```

4.5 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 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)

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

`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

`lpow:` (default: `1.0`) Power for link function defined as $(\text{mean})^{\text{power}}$ (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 SystemDS 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

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 $Y - \text{pred. mean}(Y\|X)$
STDEV_RES_Y	+		Y-column residual st. dev. of $Y - \text{pred. mean}(Y\|X)$
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.

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:

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

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):

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

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 [**?**]. 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 [**?**]:

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

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 (14). 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 [**?**]. 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:

$$G^2 \;=\; 2 \log \frac{\text{Prob}[Y \mid \text{saturated model}]}{\text{Prob}[Y \mid \text{tested linear model}]} \tag{15}$$

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 (13) become simplified in ratio (15) 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 [**?**]. 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: $\operatorname{Var} y_{i,j} = \operatorname{Var} \sum_{l=1}^{N_i} y_{i,j,l} = \sum_{l=1}^{N_i} \operatorname{Var} y_{i,j,l}$. 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 the GLM variance formula and scaled by the dispersion:

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

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.

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`

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 SystemDS.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 SystemDS.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 SystemDS.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 SystemDS.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 SystemDS.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 SystemDS.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 SystemDS.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
```

5 Matrix Factorization

5.1 Principal Component Analysis

Description

Principal Component Analysis (PCA) is a simple, non-parametric method to transform the given data set with possibly correlated columns into a set of linearly uncorrelated or orthogonal columns, called *principal components*. The principal components are ordered in such a way that the first component accounts for the largest possible variance, followed by remaining principal components in the decreasing order of the amount of variance captured from the data. PCA is often used as a dimensionality reduction technique, where the original data is projected or rotated onto a low-dimensional space with basis vectors defined by top-K (for a given value of K) principal components.

Usage

```
-f path/PCA.dml -nvargs INPUT=path/file K=int
                        CENTER=0/1 SCALE=0/1
                        PROJDATA=0/1 OFMT=csv/text
                        MODEL=path|file OUTPUT=path/file
```

Arguments

- INPUT: Location (on HDFS) to read the input matrix.
- K: Indicates dimension of the new vector space constructed from K principal components. It must be a value between 1 and the number of columns in the input data.
- CENTER (default: 0): Indicates whether or not to *center* input data prior to the computation of principal components.
- SCALE (default: 0): Indicates whether or not to *scale* input data prior to the computation of principal components.

- PROJDATA: Indicates whether or not the input data must be projected on to new vector space defined over principal components.
- OFMT (default: `csv`): Specifies the output format. Choice of comma-separated values (csv) or as a sparse-matrix (text).
- MODEL: Either the location (on HDFS) where the computed model is stored; or the location of an existing model.
- OUTPUT: Location (on HDFS) to store the data rotated on to the new vector space.

Details

Principal Component Analysis (PCA) is a non-parametric procedure for orthogonal linear transformation of the input data to a new coordinate system, such that the greatest variance by some projection of the data comes to lie on the first coordinate (called the first principal component), the second greatest variance on the second coordinate, and so on. In other words, PCA first selects a normalized direction in m-dimensional space (m is the number of columns in the input data) along which the variance in input data is maximized – this is referred to as the first principal component. It then repeatedly finds other directions (principal components) in which the variance is maximized. At every step, PCA restricts the search for only those directions that are perpendicular to all previously selected directions. By doing so, PCA aims to reduce the redundancy among input variables. To understand the notion of redundancy, consider an extreme scenario with a data set comprising of two variables, where the first one denotes some quantity expressed in meters, and the other variable represents the same quantity but in inches. Both these variables evidently capture redundant information, and hence one of them can be removed. In a general scenario, keeping solely the linear combination of input variables would both express the data more concisely and reduce the number of variables. This is why PCA is often used as a dimensionality reduction technique.

The specific method to compute such a new coordinate system is as follows – compute a covariance matrix C that measures the strength of correlation among all pairs of variables in the input data; factorize C according to eigen decomposition to calculate its eigenvalues and eigenvectors; and finally, order eigenvectors in the decreasing order of their corresponding eigenvalue. The computed eigenvectors (also known as *loadings*) define the new coordinate system and the square root of eigen values provide the amount of variance in the input data explained by each coordinate or eigenvector.

Returns When MODEL is not provided, PCA procedure is applied on INPUT data to generate MODEL as well as the rotated data OUTPUT (if PROJDATA is set to 1) in the new coordinate system. The produced model consists of basis vectors MODEL/*dominant.eigen.vectors* for the new coordinate system; eigen values MODEL/*dominant.eigen.values*; and the standard deviation MODEL/*dominant.eigen.standard.deviations* of principal components. When

MODEL is provided, INPUT data is rotated according to the coordinate system defined by MODEL/*dominant.eigen.vectors*. The resulting data is stored at location OUTPUT.

Examples

```
hadoop jar SystemDS.jar -f PCA.dml -nvargs
            INPUT=/user/biuser/input.mtx  K=10
            CENTER=1  SCALE=1
            OFMT=csv PROJDATA=1
    # location to store model and rotated data
            OUTPUT=/user/biuser/pca_output/

hadoop jar SystemDS.jar -f PCA.dml -nvargs
            INPUT=/user/biuser/test_input.mtx  K=10
            CENTER=1  SCALE=1
            OFMT=csv PROJDATA=1
    # location of an existing model
            MODEL=/user/biuser/pca_output/
    # location of rotated data
            OUTPUT=/user/biuser/test_output.mtx
```

5.2 Matrix Completion via Alternating Minimizations

Description

Low-rank matrix completion is an effective technique for statistical data analysis widely used in the data mining and machine learning applications. Matrix completion is a variant of low-rank matrix factorization with the goal of recovering a partially observed and potentially noisy matrix from a subset of its revealed entries. Perhaps the most popular applications in which matrix completion has been successfully applied is in the context of collaborative filtering in recommender systems. In this setting, the rows in the data matrix correspond to users, the columns to items such as movies, and entries to feedback provided by users for items. The goal is to predict missing entries of the rating matrix. This implementation uses the alternating least-squares (ALS) technique for solving large-scale matrix completion problems.

Usage

`-f` *path/*`ALS.dml -nvargs V=`*path/file* `L=`*path/file* `R=`*path/file* `rank=`*int* `reg=`*L2*|*wL2* `lambda=`*double* `fmt=`*format*

Arguments

`V:` Location (on HDFS) to read the input (user-item) matrix V to be factorized

`L:` Location (on HDFS) to write the left (user) factor matrix L

`R:` Location (on HDFS) to write the right (item) factor matrix R

`rank:` (default: `10`) Rank of the factorization

`reg` (default: `L2`) Regularization:
`L2` = L2 regularization;
`wL2` = weighted L2 regularization;
if `reg` is not provided no regularization will be performed.

`lambda:` (default: `0.000001`) Regularization parameter

`maxi:` (default: `50`) Maximum number of iterations

`check:` (default: `FALSE`) Check for convergence after every iteration, i.e., updating L and R once

`thr:` (default: `0.0001`) Assuming `check=TRUE`, the algorithm stops and convergence is declared if the decrease in loss in any two consecutive iterations falls below threshold `thr`; if `check=FALSE` parameter `thr` is ignored.

`fmt:` (default: `"text"`) Matrix file output format, such as `text`, `mm`, or `csv`

Usage: ALS Prediction/Top-K Prediction

`-f` *path/*`ALS_predict.dml` `-nvargs` `X=`*path/file* `Y=`*path/file* `L=`*path/file* `R=`*path/file* `Vrows=`*int* `Vcols=`*int* `fmt=`*format*

`-f` *path/*`ALS_topk_predict.dml` `-nvargs` `X=`*path/file* `Y=`*path/file* `L=`*path/file* `R=`*path/file* `V=`*path/file* `K=`*int* `fmt=`*format*

Arguments — Prediction/Top-K Prediction

`V:` Location (on HDFS) to read the user-item matrix V

`X:` Location (on HDFS) to read the input matrix X with following format:

- for ALS_predict.dml: a 2-column matrix that contains the user-ids (first column) and the item-ids (second column),
- for ALS_topk_predict.dml: a 1-column matrix that contains the user-ids.

`Y:` Location (on HDFS) to write the output of prediction with the following format:

- for ALS_predict.dml: a 3-column matrix that contains the user-ids (first column), the item-ids (second column) and the predicted ratings (third column),
- for ALS_topk_predict.dml: a $(K+1)$-column matrix that contains the user-ids in the first column and the top-K item-ids in the remaining K columns will be stored at `Y`. Additionally, a matrix with the same dimensions that contains the corresponding actual top-K ratings will be stored at `Y.ratings`; see below for details.

`L:` Location (on HDFS) to read the left (user) factor matrix L

`R:` Location (on HDFS) to write the right (item) factor matrix R

`Vrows:` Number of rows of V (i.e., number of users)

Loss	Definition
$\mathcal{L}_{\text{Nzsl}}$	$\sum_{i,j:V_{ij}\neq 0}(V_{ij}-[LR]_{ij})^2$
$\mathcal{L}_{\text{Nzsl+L2}}$	$\mathcal{L}_{\text{Nzsl}}+\lambda\Big(\sum_{ik}L_{ik}^2+\sum_{kj}R_{kj}^2\Big)$
$\mathcal{L}_{\text{Nzsl+wL2}}$	$\mathcal{L}_{\text{Nzsl}}+\lambda\Big(\sum_{ik}N_{i*}L_{ik}^2+\sum_{kj}N_{*j}R_{kj}^2\Big)$

Table 15: Popular loss functions supported by our ALS implementation; N_{i*} and N_{*j}, respectively, denote the number of nonzero entries in row i and column j of V.

`Vcols` Number of columns of V (i.e., number of items)

`K:` (default: `5`) Number of top-K items for top-K prediction

`fmt:` (default: `"text"`) Matrix file output format, such as `text`, `mm`, or `csv`

Details

Given an $m \times n$ input matrix V and a rank parameter $r \ll \min(m, n)$, low-rank matrix factorization seeks to find an $m \times r$ matrix L and an $r \times n$ matrix R such that $V \approx LR$, i.e., we aim to approximate V by the low-rank matrix LR. The quality of the approximation is determined by an application-dependent loss function $\mathcal{L}$. We aim at finding the loss-minimizing factor matrices, i.e.,

$$(L^*, R^*) = \text{argmin}_{L,R}\mathcal{L}(V, L, R). \tag{16}$$

In the context of collaborative filtering in the recommender systems it is often the case that the input matrix V contains several missing entries. Such entries are coded with the 0 value and the loss function is computed only based on the nonzero entries in V, i.e.,

$$\mathcal{L} = \sum_{(i,j)\in\Omega} l(V_{ij}, L_{i*}, R_{*j}),$$

where L_{i*} and R_{*j}, respectively, denote the ith row of L and the jth column of R, $\Omega = \{\omega_1, \ldots, \omega_N\}$ denotes the training set containing the observed (nonzero) entries in V, and l is some local loss function.

ALS is an optimization technique that can be used to solve quadratic problems. For matrix completion, the algorithm repeatedly keeps one of the unknown matrices (L or R) fixed and optimizes the other one. In particular, ALS alternates between recomputing the rows of L in one step and the columns of R in the subsequent step. Our implementation of the ALS algorithm supports the loss functions summarized in Table 5.2 commonly used for matrix completion [**?**].

Note that the matrix completion problem as defined in (16) is a non-convex problem for all loss functions from Table 5.2. However, when fixing one of the matrices L or R, we get a least-squares problem with a globally optimal solution. For example, for the case of $\mathcal{L}_{\text{Nzsl+wL2}}$ we have the following closed

form solutions

$$L^{\top}_{n+1,i*} \leftarrow (R^{(i)}_n[R^{(i)}_n]^{\top} + \lambda N_2 I)^{-1} R_n V^{\top}_{i*},$$
$$R_{n+1,*j} \leftarrow ([L^{(j)}_{n+1}]^{\top} L^{(j)}_{n+1} + \lambda N_1 I)^{-1} L^{\top}_{n+1} V_{*j},$$

where $L_{n+1,i*}$ (resp. $R_{n+1,*j}$) denotes the ith row of L_{n+1} (resp. jth column of R_{n+1}), λ denotes the regularization parameter, I is the identity matrix of appropriate dimensionality, V_{i*} (resp. V_{*j}) denotes the revealed entries in row i (column j), $R^{(i)}_n$ (resp. $L^{(j)}_{n+1}$) refers to the corresponding columns of R_n (rows of L_{n+1}), and N_1 (resp. N_2) denotes a diagonal matrix that contains the number of nonzero entries in row i (column j) of V.

Prediction. Based on the factor matrices computed by ALS we provide two prediction scripts:

1. `ALS_predict.dml` computes the predicted ratings for a given list of users and items;
2. `ALS_topk_predict.dml` computes top-K item (where K is given as input) with highest predicted ratings together with their corresponding ratings for a given list of users.

Returns

We output the factor matrices L and R after the algorithm has converged. The algorithm is declared as converged if one of the two criteria is meet: (1) the decrease in the value of loss function falls below `thr` given as an input parameter (if parameter `check=TRUE`), or (2) the maximum number of iterations (defined as parameter `maxi`) is reached. Note that for a given user i prediction is possible only if user i has rated at least one item, i.e., row i in matrix V has at least one nonzero entry. In case, some users have not rated any items the corresponding factor in L will be all 0s. Similarly if some items have not been rated at all the corresponding factors in R will contain only 0s. Our prediction scripts output the predicted ratings for a given list of users and items as well as the top-K items with highest predicted ratings together with the predicted ratings for a given list of users. Note that the predictions will only be provided for the users who have rated at least one item, i.e., the corresponding rows contain at least one nonzero entry.

Examples

```
hadoop jar SystemDS.jar -f ALS.dml -nvargs V=/user/biadmin/V
   L=/user/biadmin/L R=/user/biadmin/R rank=10 reg="wL2"
   lambda=0.0001 maxi=50 check=TRUE thr=0.001 fmt=csv
```

To compute predicted ratings for a given list of users and items:

```
hadoop jar SystemDS.jar -f ALS-predict.dml -nvargs
   X=/user/biadmin/X Y=/user/biadmin/Y L=/user/biadmin/L
   R=/user/biadmin/R Vrows=100000 Vcols=10000 fmt=csv
```

To compute top-K items with highest predicted ratings together with the predicted ratings for a given list of users:

```
hadoop jar SystemDS.jar -f ALS-top-predict.dml -nvargs
   X=/user/biadmin/X Y=/user/biadmin/Y L=/user/biadmin/L
   R=/user/biadmin/R V=/user/biadmin/V K=10 fmt=csv
```

6 Survival Analysis

6.1 Kaplan-Meier Survival Analysis

Description

Survival analysis examines the time needed for a particular event of interest to occur. In medical research, for example, the prototypical such event is the death of a patient but the methodology can be applied to other application areas, e.g., completing a task by an individual in a psychological experiment or the failure of electrical components in engineering. Kaplan-Meier or (product limit) method is a simple non-parametric approach for estimating survival probabilities from both censored and uncensored survival times.

Usage

`-f` *path/*`KM.dml` `-nvargs` `X=`*path/file* `TE=`*path/file* `GI=`*path/file* `SI=`*path/file* `O=`*path/file* `M=`*path/file* `T=`*path/file* `alpha=`*double* `etype=`*greenwood*|*peto* `ctype=`*plain*|*log*|*log-log* `ttype=`*none*|*log-rank*|*wilcoxon* `fmt=`*format*

Arguments

`X:` Location (on HDFS) to read the input matrix of the survival data containing:

- timestamps,
- whether event occurred (1) or data is censored (0),
- a number of factors (i.e., categorical features) for grouping and/or stratifying

`TE:` Location (on HDFS) to read the 1-column matrix TE that contains the column indices of the input matrix X corresponding to timestamps (first entry) and event information (second entry)

`GI:` Location (on HDFS) to read the 1-column matrix GI that contains the column indices of the input matrix X corresponding to the factors (i.e., categorical features) to be used for grouping

`SI:` Location (on HDFS) to read the 1-column matrix SI that contains the column indices of the input matrix X corresponding to the factors (i.e., categorical features) to be used for grouping

`O:` Location (on HDFS) to write the matrix containing the results of the Kaplan-Meier analysis KM

`M`: Location (on HDFS) to write Matrix M containing the following statistics: total number of events, median and its confidence intervals; if survival data for multiple groups and strata are provided each row of M contains the above statistics per group and stratum.

`T`: If survival data from multiple groups is available and `ttype=log-rank` or `ttype=wilcoxon`, location (on HDFS) to write the two matrices that contains the result of the (stratified) test for comparing these groups; see below for details.

`alpha`: (default: `0.05`) Parameter to compute $100(1-\alpha)\%$ confidence intervals for the survivor function and its median

`etype`: (default: `"greenwood"`) Parameter to specify the error type according to "greenwood" or "peto"

`ctype`: (default: `"log"`) Parameter to modify the confidence interval; "plain" keeps the lower and upper bound of the confidence interval unmodified, "log" corresponds to logistic transformation and "log-log" corresponds to the complementary log-log transformation

`ttype`: (default: `"none"`) If survival data for multiple groups is available specifies which test to perform for comparing survival data across multiple groups: "none", "log-rank" or "wilcoxon" test

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

Details

The Kaplan-Meier estimate is a non-parametric maximum likelihood estimate (MLE) of the survival function $S(t)$, i.e., the probability of survival from the time origin to a given future time. As an illustration suppose that there are n individuals with observed survival times $t_1, t_2, \ldots t_n$ out of which there are $r \leq n$ distinct death times $t_{(1)} \leq t_{(2)} \leq t_{(r)}$—since some of the observations may be censored, in the sense that the end-point of interest has not been observed for those individuals, and there may be more than one individual with the same survival time. Let $S(t_j)$ denote the probability of survival until time t_j, d_j be the number of events at time t_j, and n_j denote the number of individual at risk (i.e., those who die at time t_j or later). Assuming that the events occur independently, in Kaplan-Meier method the probability of surviving from t_j to t_{j+1} is estimated from $S(t_j)$ and given by

$$\hat{S}(t) = \prod_{j=1}^{k} \left(\frac{n_j - d_j}{n_j} \right),$$

for $t_k \leq t < t_{k+1}$, $k = 1, 2, \ldots r$, $\hat{S}(t) = 1$ for $t < t_{(1)}$, and $t_{(r+1)} = \infty$. Note that the value of $\hat{S}(t)$ is constant between times of event and therefore the estimate is a step function with jumps at observed event times. If there are no censored data this estimator would simply reduce to the empirical survivor function defined

as $\frac{n_j}{n}$. Thus, the Kaplan-Meier estimate can be seen as the generalization of the empirical survivor function that handles censored observations.

The methodology used in our `KM.dml` script closely follows [**?**, Sec. 2]. For completeness we briefly discuss the equations used in our implementation.

Standard error of the survivor function. The standard error of the estimated survivor function (controlled by parameter `etype`) can be calculated as

$$\text{se}\{\hat{S}(t)\} \approx \hat{S}(t)\left\{\sum_{j=1}^{k} \frac{d_j}{n_j(n_j - d_j)}\right\}^2,$$

for $t_{(k)} \leq t < t_{(k+1)}$. This equation is known as the *Greenwood's* formula. An alternative approach is to apply the *Petos's* expression

$$\text{se}\{\hat{S}(t)\} = \frac{\hat{S}(t)\sqrt{1 - \hat{S}(t)}}{\sqrt{n_k}},$$

for $t_{(k)} \leq t < t_{(k+1)}$. Once the standard error of $\hat{S}$ has been found we compute the following types of confidence intervals (controlled by parameter `cctype`): The "plain" $100(1-\alpha)\%$ confidence interval for $S(t)$ is computed using

$$\hat{S}(t) \pm z_{\alpha/2}\text{se}\{\hat{S}(t)\},$$

where $z_{\alpha/2}$ is the upper $\alpha/2$-point of the standard normal distribution. Alternatively, we can apply the "log" transformation using

$$\hat{S}(t)^{\exp[\pm z_{\alpha/2}\text{se}\{\hat{S}(t)\}/\hat{S}(t)]}$$

or the "log-log" transformation using

$$\hat{S}(t)^{\exp[\pm z_{\alpha/2}\text{se}\{\log[-\log \hat{S}(t)]\}]}.$$

Median, its standard error and confidence interval. Denote by $\hat{t}(50)$ the estimated median of $\hat{S}$, i.e., $\hat{t}(50) = \min\{t_i \mid \hat{S}(t_i) < 0.5\}$, where t_i is the observed survival time for individual i. The standard error of $\hat{t}(50)$ is given by

$$\text{se}\{\hat{t}(50)\} = \frac{1}{\hat{f}\{\hat{t}(50)\}}\text{se}[\hat{S}\{\hat{t}(50)\}],$$

where $\hat{f}\{\hat{t}(50)\}$ can be found from

$$\hat{f}\{\hat{t}(50)\} = \frac{\hat{S}\{\hat{u}(50)\} - \hat{S}\{\hat{l}(50)\}}{\hat{l}(50) - \hat{u}(50)}.$$

Above, $\hat{u}(50)$ is the largest survival time for which $\hat{S}$ exceeds $0.5+\epsilon$, i.e., $\hat{u}(50) = \max\{t_{(j)} \mid \hat{S}(t_{(j)}) \geq 0.5 + \epsilon\}$, and $\hat{l}(50)$ is the smallest survivor time for which $\hat{S}$ is less than $0.5 - \epsilon$, i.e., $\hat{l}(50) = \min\{t_{(j)} \mid \hat{S}(t_{(j)}) \leq 0.5 + \epsilon\}$, for small ϵ.

Log-rank test and Wilcoxon test. Our implementation supports comparison of survival data from several groups using two non-parametric procedures (controlled by parameter `ttype`): the *log-rank test* and the *Wilcoxon test* (also known as the *Breslow test*). Assume that the survival times in $g \geq 2$ groups of survival data are to be compared. Consider the *null hypothesis* that there is no difference in the survival times of the individuals in different groups. One way to examine the null hypothesis is to consider the difference between the observed number of deaths with the numbers expected under the null hypothesis. In both tests we define the U-statistics (U_L for the log-rank test and U_W for the Wilcoxon test) to compare the observed and the expected number of deaths in $1, 2, \ldots, g-1$ groups as follows:

$$U_{Lk} = \sum_{j=1}^{r} \left(d_{kj} - \frac{n_{kj} d_j}{n_j} \right),$$

$$U_{Wk} = \sum_{j=1}^{r} n_j \left(d_{kj} - \frac{n_{kj} d_j}{n_j} \right),$$

where d_{kj} is the of number deaths at time $t_{(j)}$ in group k, n_{kj} is the number of individuals at risk at time $t_{(j)}$ in group k, and $k = 1, 2, \ldots, g-1$ to form the vectors U_L and U_W with $(g-1)$ components. The covariance (variance) between U_{Lk} and $U_{Lk'}$ (when $k = k'$) is computed as

$$V_{Lkk'} = \sum_{j=1}^{r} \frac{n_{kj} d_j (n_j - d_j)}{n_j (n_j - 1)} \left(\delta_{kk'} - \frac{n_{k'j}}{n_j} \right),$$

for $k, k' = 1, 2, \ldots, g-1$, with

$$\delta_{kk'} = \begin{cases} 1 & \text{if } k = k' \\ 0 & \text{otherwise.} \end{cases}$$

These terms are combined in a *variance-covariance* matrix V_L (referred to as the V-statistic). Similarly, the variance-covariance matrix for the Wilcoxon test V_W is a matrix where the entry at position (k, k') is given by

$$V_{Wkk'} = \sum_{j=1}^{r} n_j^2 \frac{n_{kj} d_j (n_j - d_j)}{n_j (n_j - 1)} \left(\delta_{kk'} - \frac{n_{k'j}}{n_j} \right).$$

Under the null hypothesis of no group differences, the test statistics $U_L^\top V_L^{-1} U_L$ for the log-rank test and $U_W^\top V_W^{-1} U_W$ for the Wilcoxon test have a Chi-squared distribution on $(g-1)$ degrees of freedom. Our `KM.dml` script also provides a stratified version of the log-rank or Wilcoxon test if requested. In this case, the values of the U- and V- statistics are computed for each stratum and then combined over all strata.

Returns

Blow we list the results of the survival analysis computed by `KM.dml`. The calculated statistics are stored in matrix KM with the following schema:

- Column 1: timestamps
- Column 2: number of individuals at risk
- Column 3: number of events
- Column 4: Kaplan-Meier estimate of the survivor function $\hat{S}$
- Column 5: standard error of $\hat{S}$
- Column 6: lower bound of $100(1-\alpha)\%$ confidence interval for $\hat{S}$
- Column 7: upper bound of $100(1-\alpha)\%$ confidence interval for $\hat{S}$

Note that if survival data for multiple groups and/or strata is available, each collection of 7 columns in KM stores the results per group and/or per stratum. In this case KM has $7g+7s$ columns, where $g \geq 1$ and $s \geq 1$ denote the number of groups and strata, respectively.

Additionally, `KM.dml` stores the following statistics in the 1-row matrix M whose number of columns depends on the number of groups (g) and strata (s) in the data. Below k denotes the number of factors used for grouping and l denotes the number of factors used for stratifying.

- Columns 1 to k: unique combination of values in the k factors used for grouping
- Columns $k+1$ to $k+l$: unique combination of values in the l factors used for stratifying
- Column $k+l+1$: total number of records
- Column $k+l+2$: total number of events
- Column $k+l+3$: median of $\hat{S}$
- Column $k+l+4$: lower bound of $100(1-\alpha)\%$ confidence interval for the median of $\hat{S}$
- Column $k+l+5$: upper bound of $100(1-\alpha)\%$ confidence interval for the median of $\hat{S}$.

If there is only 1 group and 1 stratum available M will be a 1-row matrix with 5 columns where

- Column 1: total number of records
- Column 2: total number of events
- Column 3: median of $\hat{S}$
- Column 4: lower bound of $100(1-\alpha)\%$ confidence interval for the median of $\hat{S}$

- Column 5: upper bound of $100(1-\alpha)\%$ confidence interval for the median of $\hat{S}$.

If a comparison of the survival data across multiple groups needs to be performed, `KM.dml` computes two matrices T and T_GROUPS_OE that contain a summary of the test. The 1-row matrix T stores the following statistics:

- Column 1: number of groups in the survival data
- Column 2: degree of freedom for Chi-squared distributed test statistic
- Column 3: value of test statistic
- Column 4: P-value.

Matrix T_GROUPS_OE contains the following statistics for each of g groups:

- Column 1: number of events
- Column 2: number of observed death times (O)
- Column 3: number of expected death times (E)
- Column 4: $(O-E)^2/E$
- Column 5: $(O-E)^2/V$.

Examples

```
hadoop jar SystemDS.jar -f KM.dml -nvargs X=/user/biadmin/X.mtx
   TE=/user/biadmin/TE GI=/user/biadmin/GI SI=/user/biadmin/SI
   O=/user/biadmin/kaplan-meier.csv M=/user/biadmin/model.csv
   alpha=0.01 etype=greenwood ctype=plain fmt=csv
hadoop jar SystemDS.jar -f KM.dml -nvargs X=/user/biadmin/X.mtx
   TE=/user/biadmin/TE GI=/user/biadmin/GI SI=/user/biadmin/SI
   O=/user/biadmin/kaplan-meier.csv M=/user/biadmin/model.csv
   T=/user/biadmin/test.csv alpha=0.01 etype=peto ctype=log
   ttype=log-rank fmt=csv
```

6.2 Cox Proportional Hazard Regression Model

Description

The Cox (proportional hazard or PH) is a semi-parametric statistical approach commonly used for analyzing survival data. Unlike non-parametric approaches, e.g., the Kaplan-Meier estimates (Section 6.1), which can be used to analyze single sample of survival data or to compare between groups of survival times, the Cox PH models the dependency of the survival times on the values of *explanatory variables* (i.e., covariates) recorded for each individual at the time origin. Our focus is on covariates that do not change value over time, i.e., time-independent covariates, and that may be categorical (ordinal or

nominal) as well as continuous-valued.

Usage

`-f` *path/*`Cox.dml -nvargs X=`*path/file* `TE=`*path/file* `F=`*path/file* `R=`*path/file* `M=`*path/file* `S=`*path/file* `T=`*path/file* `COV=`*path/file* `RT=`*path/file* `XO=`*path/file* `MF=`*path/file* `alpha=`*double* `fmt=`*format*

Arguments — Model Fitting/Prediction

`X:` Location (on HDFS) to read the input matrix of the survival data containing:

- timestamps,
- whether event occurred (1) or data is censored (0),
- feature vectors

`Y:` Location (on HDFS) to the read matrix used for prediction

`TE:` Location (on HDFS) to read the 1-column matrix TE that contains the column indices of the input matrix X corresponding to timestamps (first entry) and event information (second entry)

`F:` Location (on HDFS) to read the 1-column matrix F that contains the column indices of the input matrix X corresponding to the features to be used for fitting the Cox model

`R:` (default: `" "`) If factors (i.e., categorical features) are available in the input matrix X, location (on HDFS) to read matrix R containing the start (first column) and end (second column) indices of each factor in X; alternatively, user can specify the indices of the baseline level of each factor which needs to be removed from X. If R is not provided by default all variables are considered to be continuous-valued.

`M:` Location (on HDFS) to store the results of Cox regression analysis including regression coefficients β_js, their standard errors, confidence intervals, and P-values

`S:` (default: `" "`) Location (on HDFS) to store a summary of some statistics of the fitted model including number of records, number of events, log-likelihood, AIC, Rsquare (Cox & Snell), and maximum possible Rsquare

`T:` (default: `" "`) Location (on HDFS) to store the results of Likelihood ratio test, Wald test, and Score (log-rank) test of the fitted model

`COV:` Location (on HDFS) to store the variance-covariance matrix of β_js; note that parameter `COV` needs to provided as input to prediction.

`RT:` Location (on HDFS) to store matrix RT containing the order-preserving recoded timestamps from X; note that parameter `RT` needs to provided as input for prediction.

`XO:` Location (on HDFS) to store the input matrix X ordered by the timestamps; note that parameter `XO` needs to provided as input for prediction.

`MF:` Location (on HDFS) to store column indices of X excluding the baseline factors if available; note that parameter `MF` needs to provided as input for prediction.

`P` Location (on HDFS) to store matrix P containing the results of prediction

`alpha` (default: `0.05`) Parameter to compute a $100(1-\alpha)\%$ confidence interval for β_js

`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 SystemDS Language Reference for details.

Usage: Cox Prediction

`-f` *path/*`Cox-predict.dml` `-nvargs` `X=`*path/file* `RT=`*path/file* `M=`*path/file* `Y=`*path/file* `COV=`*path/file* `MF=`*path/file* `P=`*path/file* `fmt=`*format*

Details

In Cox PH regression model the relationship between the hazard function—i.e., the probability of event occurrence at a given time—and the covariates is described as

$$h_i(t) = h_0(t) \exp\Big\{\sum_{j=1}^{p} \beta_j x_{ij}\Big\}, \tag{17}$$

where the hazard function for the ith individual ($i \in \{1, 2, \ldots, n\}$) depends on a set of p covariates $x_i = (x_{i1}, x_{i2}, \ldots, x_{ip})$, whose importance is measured by the magnitude of the corresponding coefficients $\beta = (\beta_1, \beta_2, \ldots, \beta_p)$. The term $h_0(t)$ is the baseline hazard and is related to a hazard value if all covariates equal 0. In the Cox PH model the hazard function for the individuals may vary over time, however the baseline hazard is estimated non-parametrically and can take any form. Note that re-writing (17) we have

$$\log\Big\{\frac{h_i(t)}{h_0(t)}\Big\} = \sum_{j=1}^{p} \beta_j x_{ij}.$$

Thus, the Cox PH model is essentially a linear model for the logarithm of the hazard ratio and the hazard of event for any individual is a constant multiple of the hazard of any other. We follow similar notation and methodology as in [**?**, Sec. 3]. For completeness we briefly discuss the equations used in our implementation.

Factors in the model. Note that if some of the feature variables are factors they need to *dummy code* as follows. Let α be such a variable (i.e., a factor) with a levels. We introduce $a-1$ indicator (or dummy coded) variables $X_2, X_3 \ldots, X_a$ with $X_j = 1$ if $\alpha = j$ and 0 otherwise, for $j \in \{2, 3, \ldots, a\}$. In particular, one of a levels of α will be considered as the baseline and is not

included in the model. In our implementation, user can specify a baseline level for each of the factor (as selecting the baseline level for each factor is arbitrary). On the other hand, if for a given factor α no baseline is specified by the user, the most frequent level of α will be considered as the baseline.

Fitting the model. We estimate the coefficients of the Cox model via negative log-likelihood method. In particular the Cox PH model is fitted by using trust region Newton method with conjugate gradient [**?**]. Define the risk set $R(t_j)$ at time t_j to be the set of individuals who die at time t_i or later. The PH model assumes that survival times are distinct. In order to handle tied observations we use the *Breslow* approximation of the likelihood function

$$\mathcal{L} = \prod_{j=1}^{r} \frac{\exp(\beta^\top s_j)}{\left\{ \sum_{l \in R(t_j)} \exp(\beta^\top x_l) \right\}^{d_j}},$$

where d_j is number individuals who die at time t_j and s_j denotes the element-wise sum of the covariates for those individuals who die at time t_j, $j = 1, 2, \ldots, r$, i.e., the hth element of s_j is given by $s_{hj} = \sum_{k=1}^{d_j} x_{hjk}$, where x_{hjk} is the value of hth variable ($h \in \{1, 2, \ldots, p\}$) for the kth of the d_j individuals ($k \in \{1, 2, \ldots, d_j\}$) who die at the jth death time ($j \in \{1, 2, \ldots, r\}$).

Standard error and confidence interval for coefficients. Note that the variance-covariance matrix of the estimated coefficients $\hat{\beta}$ can be approximated by the inverse of the Hessian evaluated at $\hat{\beta}$. The square root of the diagonal elements of this matrix are the standard errors of estimated coefficients. Once the standard errors of the coefficients $se(\hat{\beta})$ is obtained we can compute a $100(1-\alpha)\%$ confidence interval using $\hat{\beta} \pm z_{\alpha/2} se(\hat{\beta})$, where $z_{\alpha/2}$ is the upper $\alpha/2$-point of the standard normal distribution. In `Cox.dml`, we utilize the build-in function `inv()` to compute the inverse of the Hessian. Note that this build-in function can be used only if the Hessian fits in the main memory of a single machine.

Wald test, likelihood ratio test, and log-rank test. In order to test the *null hypothesis* that all of the coefficients β_js are 0, our implementation provides three statistical test: *Wald test*, *likelihood ratio test*, the *log-rank test* (also known as the *score test*). Let p be the number of coefficients. The Wald test is based on the test statistic $\hat{\beta}^2 / se(\hat{\beta})^2$, which is compared to percentage points of the Chi-squared distribution to obtain the P-value. The likelihood ratio test relies on the test statistic $-2\log\{L(\mathbf{0})/L(\hat{\beta})\}$ ($\mathbf{0}$ denotes a zero vector of size p) which has an approximate Chi-squared distribution with p degrees of freedom under the null hypothesis that all β_js are 0. The Log-rank test is based on the test statistic $l = \nabla^\top L(\mathbf{0}) \mathcal{H}^{-1}(\mathbf{0}) \nabla L(\mathbf{0})$, where $\nabla L(\mathbf{0})$ is the gradient of L and $\mathcal{H}(\mathbf{0})$ is the Hessian of L evaluated at $\mathbf{0}$. Under the null hypothesis that $\beta = \mathbf{0}$, l has a Chi-squared distribution on p degrees of freedom.

Prediction. Once the parameters of the model are fitted, we compute the following predictions together with their standard errors

- linear predictors,
- risk, and

- estimated cumulative hazard.

Given feature vector X_i for individual i, we obtain the above predictions at time t as follows. The linear predictors (denoted as $\mathcal{LP}$) as well as the risk (denoted as $\mathcal{R}$) are computed relative to a baseline whose feature values are the mean of the values in the corresponding features. Let $X_i^{\text{rel}} = X_i - \mu$, where μ is a row vector that contains the mean values for each feature. We have $\mathcal{LP} = X_i^{\text{rel}}\hat{\beta}$ and $\mathcal{R} = \exp\{X_i^{\text{rel}}\hat{\beta}\}$. The standard errors of the linear predictors $se\{\mathcal{LP}\}$ are computed as the square root of $(X_i^{\text{rel}})^\top V(\hat{\beta})X_i^{\text{rel}}$ and the standard error of the risk $se\{\mathcal{R}\}$ are given by the square root of $(X_i^{\text{rel}} \odot \mathcal{R})^\top V(\hat{\beta})(X_i^{\text{rel}} \odot \mathcal{R})$, where $V(\hat{\beta})$ is the variance-covariance matrix of the coefficients and $\odot$ is the element-wise multiplication.

We estimate the cumulative hazard function for individual i by

$$\hat{H}_i(t) = \exp(\hat{\beta}^\top X_i)\hat{H}_0(t),$$

where $\hat{H}_0(t)$ is the *Breslow estimate* of the cumulative baseline hazard given by

$$\hat{H}_0(t) = \sum_{j=1}^{k} \frac{d_j}{\sum_{l \in R(t_{(j)})} \exp(\hat{\beta}^\top X_l)}.$$

In the equation above, as before, d_j is the number of deaths, and $R(t_{(j)})$ is the risk set at time $t_{(j)}$, for $t_{(k)} \leq t \leq t_{(k+1)}$, $k = 1, 2, \ldots, r-1$. The standard error of $\hat{H}_i(t)$ is obtained using the estimation

$$se\{\hat{H}_i(t)\} = \sum_{j=1}^{k} \frac{d_j}{\left[\sum_{l \in R(t_{(j)})} \exp(X_l\hat{\beta})\right]^2} + J_i^\top(t)V(\hat{\beta})J_i(t),$$

where

$$J_i(t) = \sum_{j-1}^{k} d_j \frac{\sum_{l \in R(t_{(j)})}(X_l - X_i)\exp\{(X_l - X_i)\hat{\beta}\}}{\left[\sum_{l \in R(t_{(j)})} \exp\{(X_l - X_i)\hat{\beta}\}\right]^2},$$

for $t_{(k)} \leq t \leq t_{(k+1)}$, $k = 1, 2, \ldots, r-1$.

Returns

Blow we list the results of fitting a Cox regression model stored in matrix `M` with the following schema:

- Column 1: estimated regression coefficients $\hat{\beta}$
- Column 2: $\exp(\hat{\beta})$
- Column 3: standard error of the estimated coefficients $se\{\hat{\beta}\}$
- Column 4: ratio of $\hat{\beta}$ to $se\{\hat{\beta}\}$ denoted by Z
- Column 5: P-value of Z

- Column 6: lower bound of $100(1-\alpha)\%$ confidence interval for $\hat{\beta}$
- Column 7: upper bound of $100(1-\alpha)\%$ confidence interval for $\hat{\beta}$.

Note that above Z is the Wald test statistic which is asymptotically standard normal under the hypothesis that $\beta = \mathbf{0}$.

Moreover, `Cox.dml` outputs two log files `S` and `T` containing a summary statistics of the fitted model as follows. File `S` stores the following information

- Line 1: total number of observations
- Line 2: total number of events
- Line 3: log-likelihood (of the fitted model)
- Line 4: AIC
- Line 5: Cox & Snell Rsquare
- Line 6: maximum possible Rsquare.

Above, the AIC is computed as in (12), the Cox & Snell Rsquare is equal to $1-\exp\{-l/n\}$, where l is the log-rank test statistic as discussed above and n is total number of observations, and the maximum possible Rsquare computed as $1-\exp\{-2L(\mathbf{0})/n\}$, where $L(\mathbf{0})$ denotes the initial likelihood.

File `T` contains the following information

- Line 1: Likelihood ratio test statistic, degree of freedom of the corresponding Chi-squared distribution, P-value
- Line 2: Wald test statistic, degree of freedom of the corresponding Chi-squared distribution, P-value
- Line 3: Score (log-rank) test statistic, degree of freedom of the corresponding Chi-squared distribution, P-value.

Additionally, the following matrices will be stored. Note that these matrices are required for prediction.

- Order-preserving recoded timestamps RT, i.e., contiguously numbered from 1 ... #timestamps
- Feature matrix ordered by the timestamps XO
- Variance-covariance matrix of the coefficients COV
- Column indices of the feature matrix with baseline factors removed (if available) MF.

Prediction Finally, the results of prediction is stored in Matrix P with the following schema

- Column 1: linear predictors

- Column 2: standard error of the linear predictors
- Column 3: risk
- Column 4: standard error of the risk
- Column 5: estimated cumulative hazard
- Column 6: standard error of the estimated cumulative hazard.

Examples

```
hadoop jar SystemDS.jar -f Cox.dml -nvargs X=/user/biadmin/X.mtx
   TE=/user/biadmin/TE F=/user/biadmin/F R=/user/biadmin/R
   M=/user/biadmin/model.csv T=/user/biadmin/test.csv
   COV=/user/biadmin/var-covar.csv XO=/user/biadmin/X-sorted.mtx
   fmt=csv
```

```
hadoop jar SystemDS.jar -f Cox.dml -nvargs
   X=/user/biadmin/X.mtx TE=/user/biadmin/TE F=/user/biadmin/F
   R=/user/biadmin/R M=/user/biadmin/model.csv
   T=/user/biadmin/test.csv COV=/user/biadmin/var-covar.csv
   RT=/user/biadmin/recoded-timestamps.csv
   XO=/user/biadmin/X-sorted.csv MF=/user/biadmin/baseline.csv
   alpha=0.01 tol=0.000001 moi=100 mii=20 fmt=csv
```

To compute predictions:

```
hadoop jar SystemDS.jar -f Cox-predict.dml -nvargs
   X=/user/biadmin/X-sorted.mtx RT=/user/biadmin/recoded-timestamps.csv
   M=/user/biadmin/model.csv Y=/user/biadmin/Y.mtx
   COV=/user/biadmin/var-covar.csv MF=/user/biadmin/baseline.csv
   P=/user/biadmin/predictions.csv fmt=csv
```