Command Line Interface, Stochastic SVD[*]

1 Overview.

Stochasitc SVD method in Mahout produces reduced rank Singular Value Decomposition output in its strict mathematical definition:

$$\mathbf{A} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^{\top},$$

i. e. it creates outputs for matrices $\mathbf{U}$, $\mathbf{V}$ and $\boldsymbol{\Sigma}$, each of which may be requested individually. The desired rank of decomposition, henceforth denoted as k, is a parameter of the algorithm. The singular values inside diagonal matrix $\boldsymbol{\Sigma}$ satisfy $\sigma_{i+1} \geq \sigma_i \ \forall i \in [1, k-1]$, i.e. sorted from biggest to smallest. Cases of rank deficiency $\operatorname{rank}(\mathbf{A}) < k$ are handled by producing 0s in singular value positions once deficiency takes place.

On top of it, there's an option to present decomposition output in a form of

$$\mathbf{A} = \mathbf{U}_{\sigma} V_{\sigma}^{\top},$$

where one can request $\mathbf{U}_{\sigma} = \mathbf{U}\boldsymbol{\Sigma}^{0.5}$ instead of $\mathbf{U}$ (but not both), $\mathbf{V}_{\sigma} = \mathbf{V}\boldsymbol{\Sigma}^{0.5}$ instead of $\mathbf{V}$ (but not both). Here, notation $\boldsymbol{\Sigma}^{0.5}$ implies diagonal matrix containing square roots of the singular values:

$$\boldsymbol{\Sigma}^{0.5} = \begin{pmatrix} \sqrt{\sigma_1} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sqrt{\sigma_k} \end{pmatrix}.$$

Original singular values $\boldsymbol{\Sigma}$ are still produced and saved regardless.

[*] Dmitriy Lyubimov, dlyubimov at apache dot org

This option is a nod to a common need of comparing actors represented by both input rows and input columns in a common space. E.g. if LSI is performed such that rows are documents and columns are terms then it is possible to compare documents and terms (ether existing or fold in new ones) in one common space and perform similarity measurement between a document and a term, rather than computing just term2term or document2document similarities.

Some of common applications for SVD include Latent Semantic Analysis (LSA), Principal Component Analysis (PCA) and others.

2 File formats

Input $\mathbf{A}$, as well as outputs $\mathbf{U}\left(\mathbf{U}_{\sigma}\right), \mathbf{V}\left(\mathbf{V}_{\sigma}\right)$, are Mahout's Distributed Row Matrix format, i.e. set of sequence files where value is of `VectorWritable` type. As far as keys are concerned, rows of $\mathbf{A}$ may be keyed (identified) by any `Writable` (for as long as it is instantiable thru a default constructor). That, among others, means that this method can be applied directly on the output of `seq2sparse` where keys were of `Text` type[1].

Definition of output $\mathbf{U}$ $\left(\mathbf{U}_{\sigma}\right)$ is identical to definition of the input matrix $\mathbf{A}$, and the keys of corresponding rows in $\mathbf{A}$ are copied to corresponding rows of output $\mathbf{U}$ $\left(\mathbf{U}_{\sigma}\right)$.

Definition of output $\mathbf{V}$ $\left(\mathbf{V}_{\sigma}\right)$ is always sequence file(s) of `(IntWritable, VectorWritable)` where key corresponds to a row index of the input $\mathbf{A}$.

Output of $\mathbf{\Sigma}$ is encoded by a single output file with a single vector value `(VectorWritable)` with main diagonal entries of $\mathbf{\Sigma}$ aka singular values $\begin{pmatrix}\sigma_1 & \cdots & \sigma_k\end{pmatrix}$.

3 Usage[2]

```
mahout ssvd <options>
```

Options.

`-k, --rank <int-value>` (required): the requested SVD rank (minimum number of singular values and dimensions in U, V matrices)

[1] (TODO: re-verify)
[2] As of Mahout 0.6 trunk

`-p, --oversampling <int-value>` (required): stochastic SVD oversampling. *k+p=500 is probably more than reasonable.* p doesn't seem to have to be very significant (perhaps 5..10).

`-q, --powerIter <int-value>` (optional, default 0): number of power iterations to perform. This helps fighting data noise and improve precision significantly more than just increasing p. Each additional power iteration adds 2 more steps (map/reduce + map-only). Experimental data suggests using $q = 1$ is already producing quite good results which are hard to much improve upon.

`-r, --blockHeight <int-value>` (optional, default 10,000): the number of rows of source matrix for block computations. Taller blocking causes more memory use but produces less blocks and therefore somewhat better running times. The most optimal mode from the running time point of view should be 1 block per 1 mapper.

`-oh, --outerProdBlockHeight <int-value>` (optional, default 10,000): the block height in multiplication operations. With extreme sparse matrices increasing that parameter will lead to better performance by reducing computational pressure on the shuffle and sort and grouping sparse records together. However, setting it too high may cause larger block values formed and written and may cause OOM.[3]

`-s, --minSplitSize <int-value>` (optional, default: use Hadoop's default): minimum split size to use in mappers reading **A** input. [4]

`--computeU <true|false>` (optional, default true). Request computation of the U matrix

`--computeV <true|false>` (optional, default true). Request computation of the V matrix

[3] Matrix mutliplications are the biggest bottleneck of this method as of the time of this writing. Tweaking this parameters for bigger blocks will help tremendeously but may cause OOM and/or GC thrashing which will again either decrease performance dramatically or even derail the whole job. So sweet balance must be striken here. Default is good for dense inputs and safe for sparse inputs).

[4] *As of this day, I haven't heard of a case where somebody would actually have to use this option and actually increase split size and how it has played out. So this option is experimental.*

Since in this version projection block formation happens in mappers, for significantly large -r and width of the input matrix the algorithm may not be able to read minimum $k + p$ rows and form a block of minimum height required, so in that case the job would bail out at the very first mapping step. If this happens, one of the recourses available is to force increase in the MapReduce split size using SequenceFileInputFormat.setMinSplitSize() property. Increasing this significantly over HDFS size may result in network IO to mappers. Another caveat is that you sometimes don't want too many mappers because it may in fact increase time of the computation. Consquently, this option should probably left alone unless one has significant amount of mappers (as in thousands of map tasks) at which point reducing amount of mappers may actually improve the thruput (just a guesstimate at this point).

`--vHalfSigma <true|false>` (optional, default: false): compute $\mathbf{V}_\sigma = \mathbf{V}\mathbf{\Sigma}^{0.5}$ instead of $\mathbf{V}$ (see overview for explanation).

`--uHalfSigma <true|false>` (optional, default: false): compute $\mathbf{U}_\sigma = \mathbf{U}\mathbf{\Sigma}^{0.5}$ instead of $\mathbf{U}$.

`--reduceTasks <int-value>` optional. The number of reducers to use (where applicable): depends on size of the hadoop cluster. At this point it could also be overwritten by a standard hadoop property using -D option[5].

Standard Options.

`--input <glob>` HDFS glob specification where the DistributedRowMatrix input to be found.

`--output <hdfs-dir>` non-existent hdfs directory where to output $\mathbf{U}, \mathbf{V}$ and $\mathbf{\Sigma}$ (singular values) files.

`--tempDir <temp-dir>` temporary dir where to store intermediate files (cleaned up upon normal completion). This is a standard Mahout optional parameter.

`-ow` overwrite output if exists.

4 Embedded use

It is possible to instantiate and use SSVDSolver class in embedded fashion in Hadoop-enabled applications. This class would have getter and setter methods for each option available via command line. See javadoc for details.

[5] TODO: reverify