Tangent Normalization in Spark notes:

This document is for developers.

S : *Number of case samples*
S_E: *Number of eigensamples*
T : *Number of targets (this is usually the largest count, by far)*

A : *Reduced panel* $[T \, x \, S_E]$
C : *Cases being projected* $[T \, x \, S]$
P: *Pseudoinverse of the reduced panel* $[S_E \, x \, T]$
$\hat{A}$: *projection of case samples into the reduced hyperplane.* $[T \, x \, S]$

$\hat{\beta} = C^T P^T$ [SxS$_E$]
$A\hat{\beta} = \hat{A}$ [TxS]
$C - \hat{A}$ [TxS]

$APC = \hat{A}$ Unfortunately, this can eat a lot of RAM, since AP is [T x T].

So why not do A(PC), which never keeps a [TxT] matrix in RAM?
The issue with doing that is a practical concern when using Spark. When you do a matrix multiply in Spark, the distributed matrix (RowMatrix) is always on the left (see the javadoc API). Multiplying two distributed matrices is not trivially supported. If you were to implement A(PC), your workflow would be:

1. Convert P to RowMatrix
2. Multiply P by C to get a new RowMatrix (PC)
3. Convert PC to local matrix (spark collect is called)
4. Convert A to RowMatrix
5. Multiply A by PC and convert to local matrix (spark collect is called).

The two collect calls will be expensive.

So… for ease of Spark

$$\hat{A} = (AP)C$$
$$\Rightarrow \hat{A}^T = C^T (P^T A^T)$$
$$\Rightarrow \hat{A}^T = (C^T P^T) A^T$$

Now the RowMatrix is C^T. $C^T P^T$ is [S x S$_E$] and $(C^T P^T) A^T$ is [S x T]

Then call collect.

Wed, January 6, 2016