Spark Meetup

Big Data Analytics
Verizon Lab, Palo Alto

July 28th, 2015

Similarity Computation

Similarity Computation Flows

- Column based flow for tall-skinny matrices (60 M users, 100K items)
  - Mapper: emit (item-i, item-j), score-ij
  - Reducer: reduce over (item-i, item-j) to get similarity-ij
  - Spark 1.2 RowMatrix.columnSimilarities

- Row based flow https://issues.apache.org/jira/browse/SPARK-4823
  - Column similarity in tall-wide matrices
    - 60M users,1M-10M items from advertising use-cases
  - Kernel generation for tall-skinny matrices
    - 60M users, 50-400 latent factors from advertising use-cases
    - 10M devices, skinny features from IoT use-cases

Row Based Flow

- Preprocess
  - Column similarity in tall-wide matrices : Transpose data matrix
  - Kernel generation for tall-skinny matrices : Input data matrix
- Algorithm
  - Distributed matrix multiply using blocked cartesian pattern
  - Shuffle space control using topK and similarity threshold
  - User specified kernel for vector dot product
  - Supported kernels: Cosine, Euclidean, RBF, ScaledProduct
- Code optimization
  - Norm caching for efficiency (kernel abstraction differ from scikit-learn)
  - DGEMM for dense vectors : Spark 1.4 recommendForAll
  - BLAS.dot for sparse vectors : https://github.com/apache/spark/pull/6213

Kernel Examples

CosineKernel: item->item similarity

```
case class CosineKernel(rowNorms: Map[Long, Double], threshold: Double) extends Kernel {
  override def compute(vi: Vector, indexi: Long, vj: Vector, indexj: Long): Double = {
    val similarity = BLAS.dot(vi, vj) / rowNorms(indexi) / rowNorms(indexj)
    if (similarity <= threshold) return 0.0
    similarity
  }
}
```

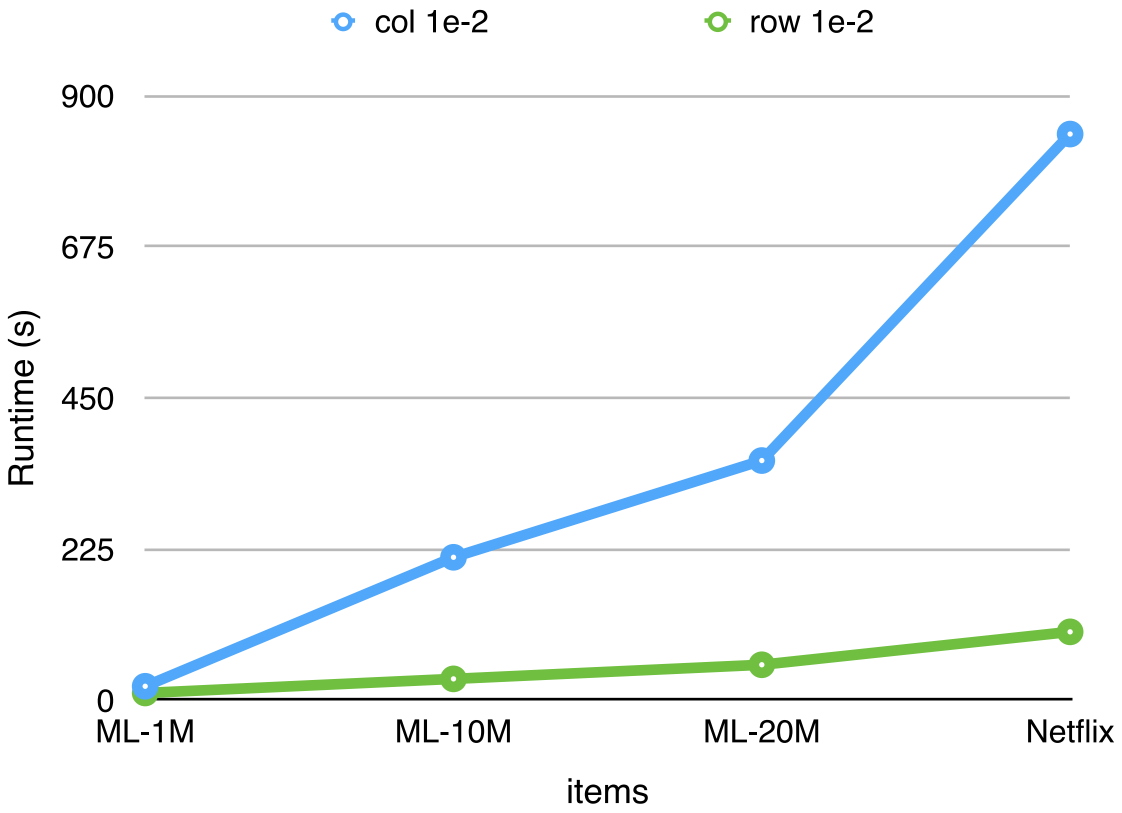
ScaledProductKernel: memory based recommendation

```
case class ScaledProductKernel(rowNorms: Map[Long, Double]) extends Kernel {
  override def compute(vi: Vector, indexi: Long, vj: Vector, indexj: Long): Double = {
    BLAS.dot(vi, vj) / rowNorms(indexi)
  }
}
```

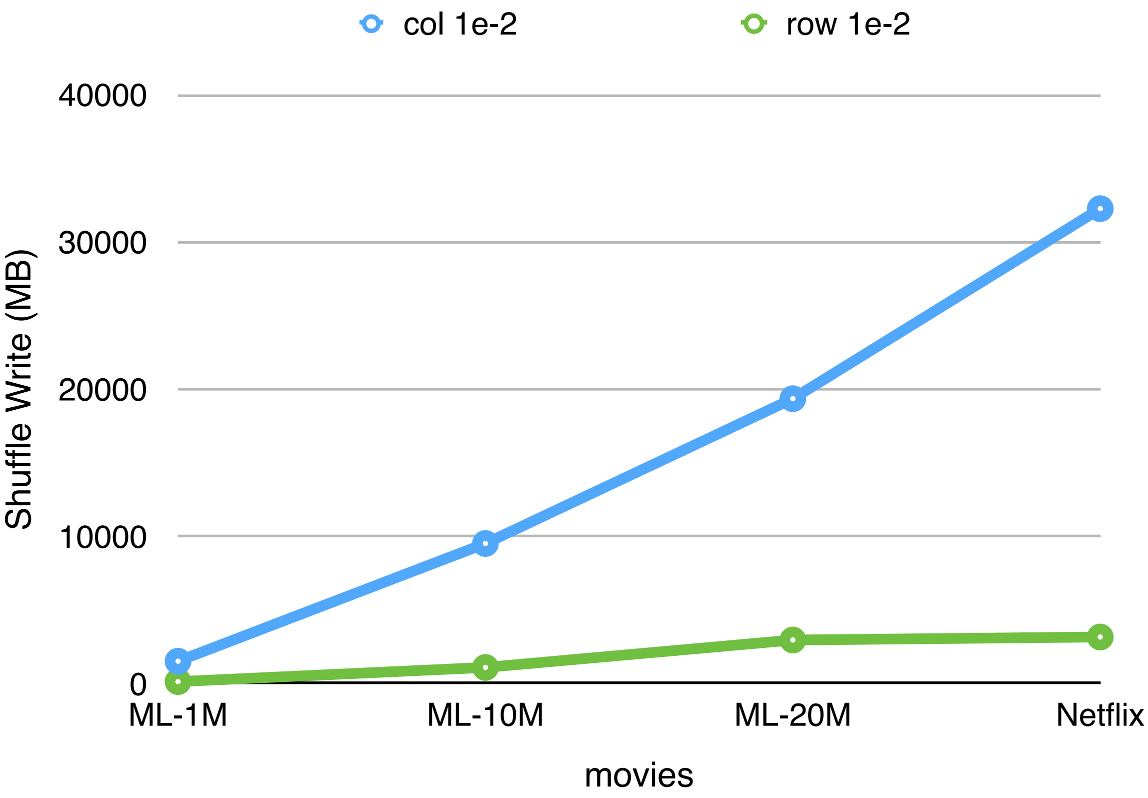
Runtime Analysis

Dataset Details				
	ML-1M	**ML-10M**	**ML-20M**	**Netflix**
ratings	1M	10M	20M	100M
users	6040	69878	138493	480189
items	3706	10677	26744	17770

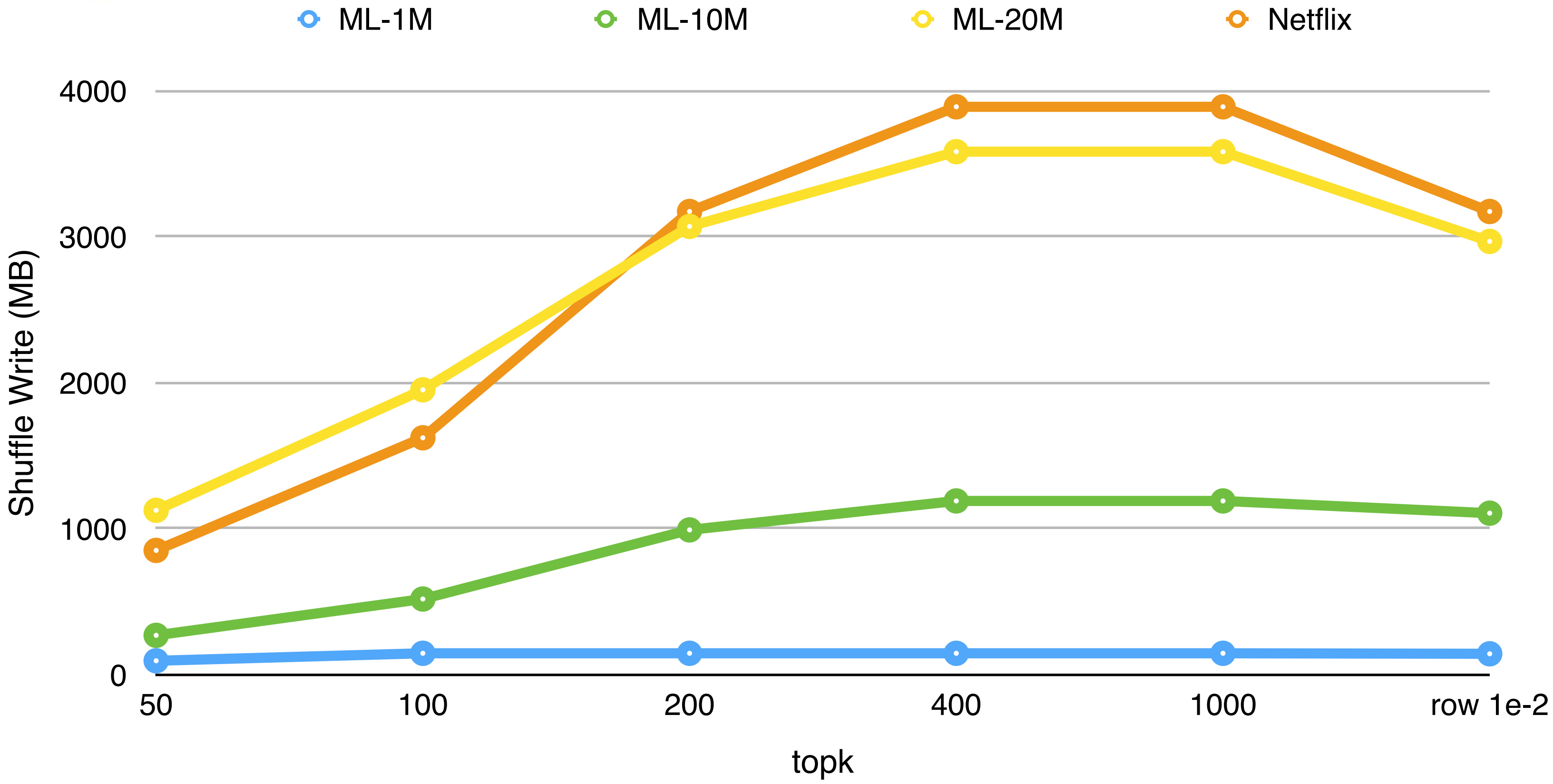
- Production Examples
- Data matrix: 60 M x 2.5 M
- minSupport: 500
- itemThreshold: 1000
- Runtime: ~ 4 hrs



Shuffle Write Analysis



TopK Shuffle Write Analysis For Row Based Flow



Recommendation Engine

Recommendation Algorithms

- Memory based: kNN based recommendation algorithm using similarity engine
- Model based: ALS based implicit feedback formulation
- Datasets
  - MovieLens 1M
  - Netflix
- Mapped ratings to binary features for comparison
- Evaluate recommendation performance using
  - RMSE
  - Precision @ *k*

kNN Based Formulation

Predicted rating $p_{ui} = \frac{\sum_{k \in \{\text{neighbors of item i}\}} (r_{uk} \times s_{ik})}{\sum_{k \in \{\text{neighbors of item i}\}} |s_{ik}|}$

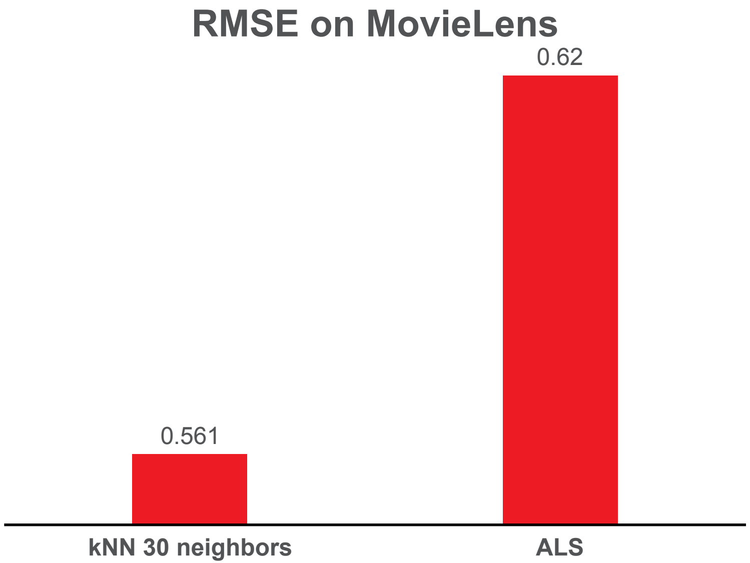
```
val similarItems = SimilarityMatrix.rowSimilarities(
                        itemFeatures,
                        numNeighbors, threshold)

val kernel = new ScaledProductKernel(rowNorms)

val recommendation = SimilarityMatrix.multiply(
                        similarItems,
                        userFeatures,
                        kernel, k)
```

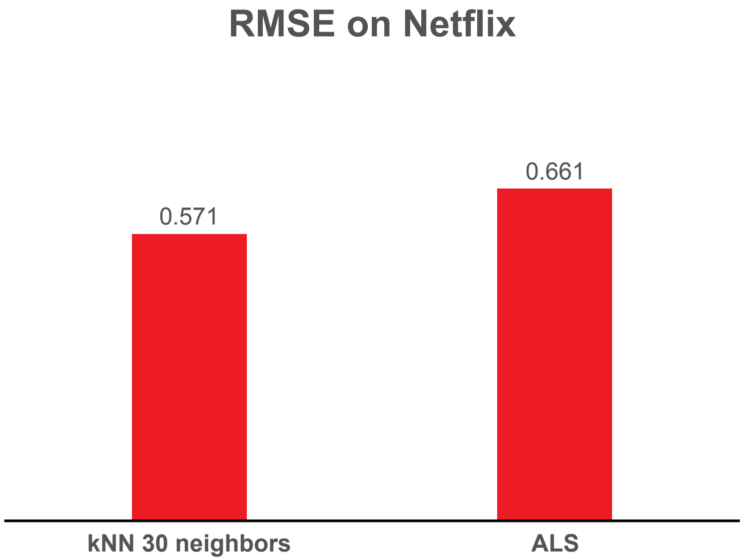
ALS Formulation

- Implicit feedback datasets: Unobserved items are considered 0 (implicit feedback)
  - Minimize $\sum_{i,j}(1 + \alpha r_{ij})(p_{ij} - w_i \times h_j)^2 + \lambda(||W|| + ||H||)$
  - Needs Gram matrix aggregation for 0-ratings

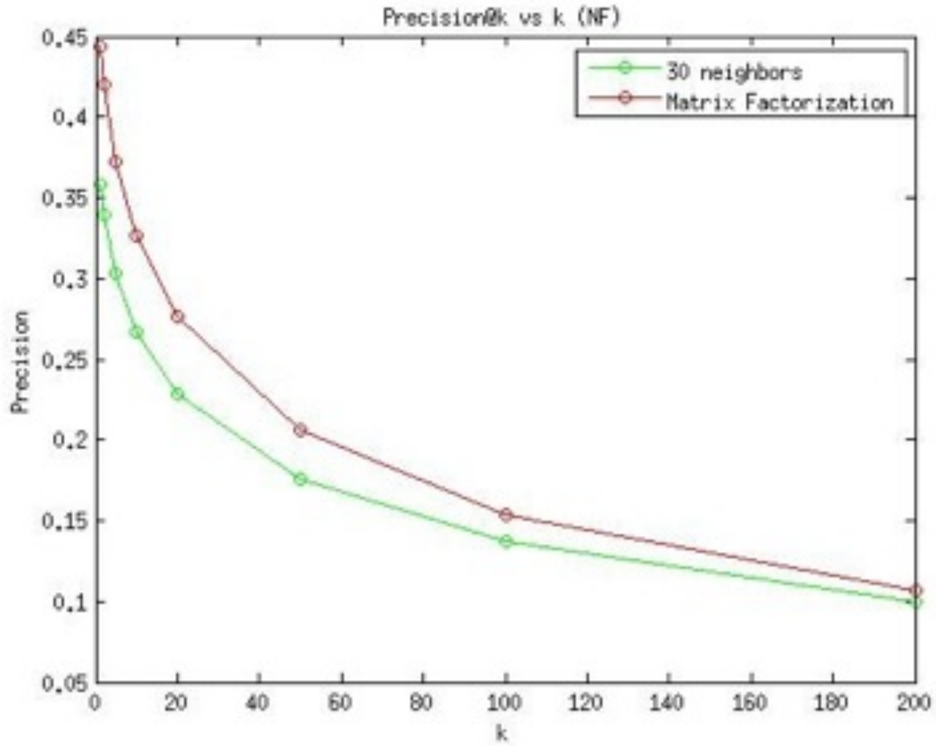
```
val als = new ALSQp()
  .setRank(params.rank)
  .setIterations(params. numIterations)
  .setUserConstraint(Constraints.SMOOTH)
  .setItemConstraint(Constraints.SMOOTH)
  .setImplicitPrefs(true)
  .setLambda(params.lambda)
  .setAlpha(params.alpha)

val mfModel = als.run(training)
RankingUtils.recommendItemsForUsers(mfModel, k, skipItems)
```

Comparing kNN and ALS on RMSE





Comparing kNN and ALS on Prec@k (Netflix)



Segmentation Engine

Segmentation Feature Extraction

- Input data contains location and time information along with other features
- Extract time-unit features for each location (zip code)

Raw data

Id	Time (Hour)	Zip Code	websites
abc	*10*	*94301*	*website1*
abc	*15*	*94085*	*website2*
def	*10*	*94301*	*website1*
⋮	⋮	⋮	⋮

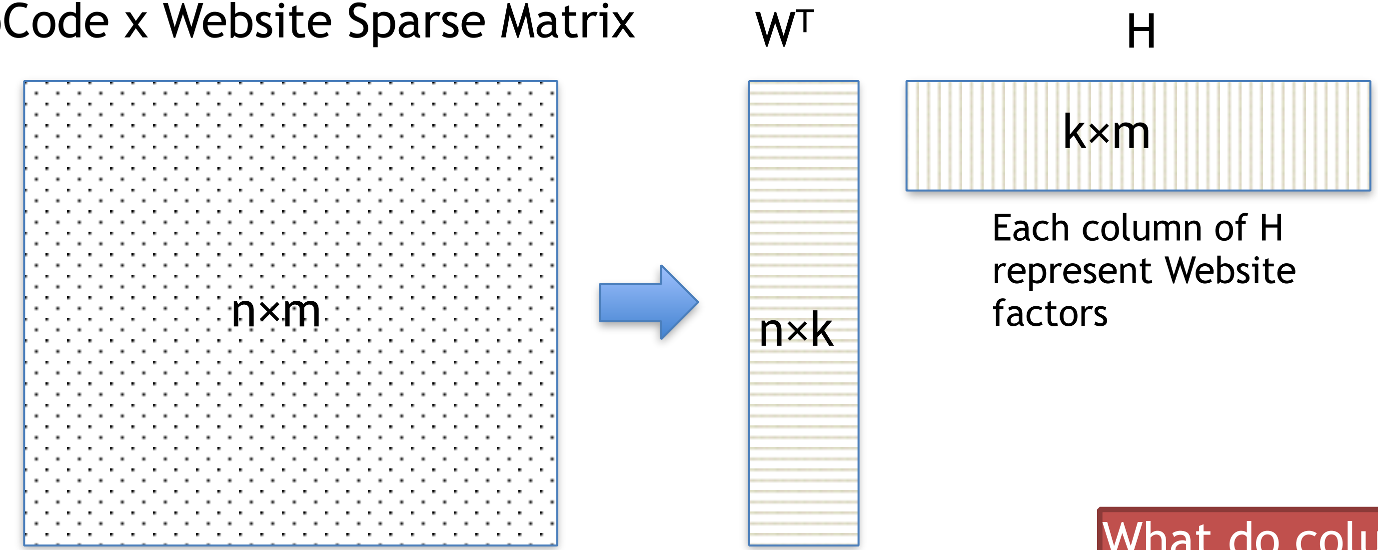
Sparse Website Matrix

	website1	website2	...
94301	*# of hours (1-24)*	*# of hours (1-24)*	...
94085	*# of hours (1-24)*	*# of hours (1-24)*	...
⋮	⋮	⋮	⋮

Column	Count
Zip codes	31516
Websites	11646
Ratings	45M

ALS with Positive Constraints

ZipCode x Website Sparse Matrix



Each column of H represent Website factors

Each row of W^T represent ZipCode

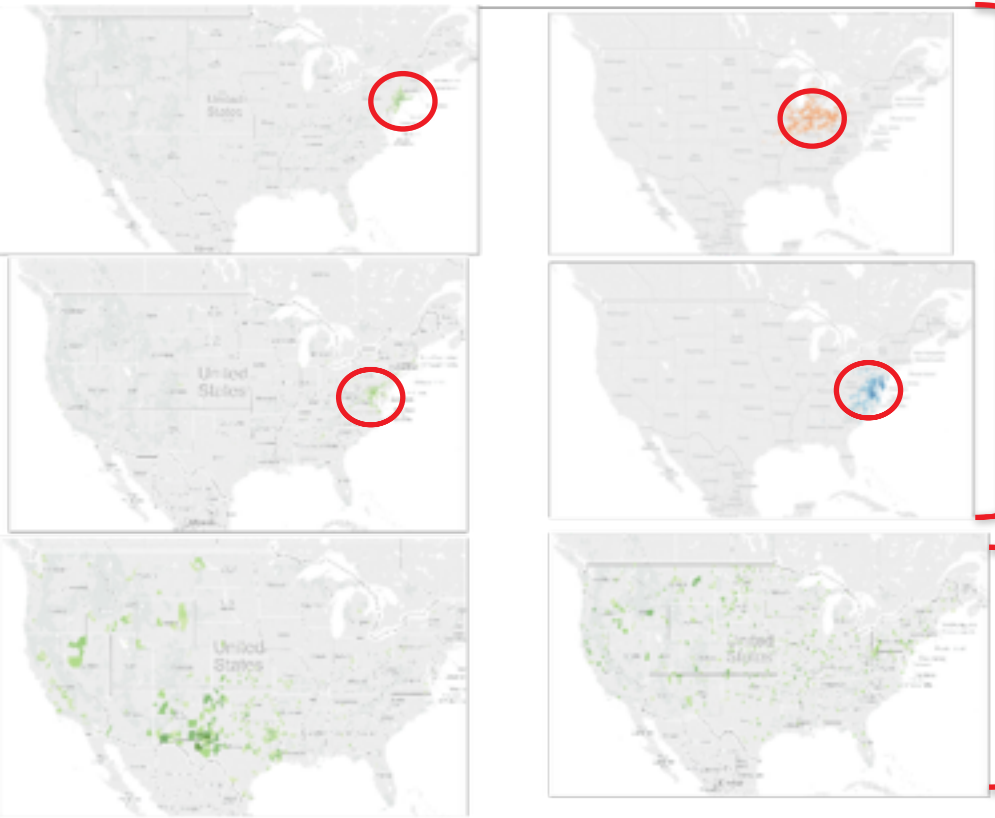
What do columns of W^T and rows of H represent?

minimize

$$\sum_{i,j}(1 + \alpha r_{ij})(p_{ij} - w_i \times h_j)^2 + \lambda(||W|| + ||H||)$$

$$\text{s.t } W \geq 0 \ H \geq 0$$

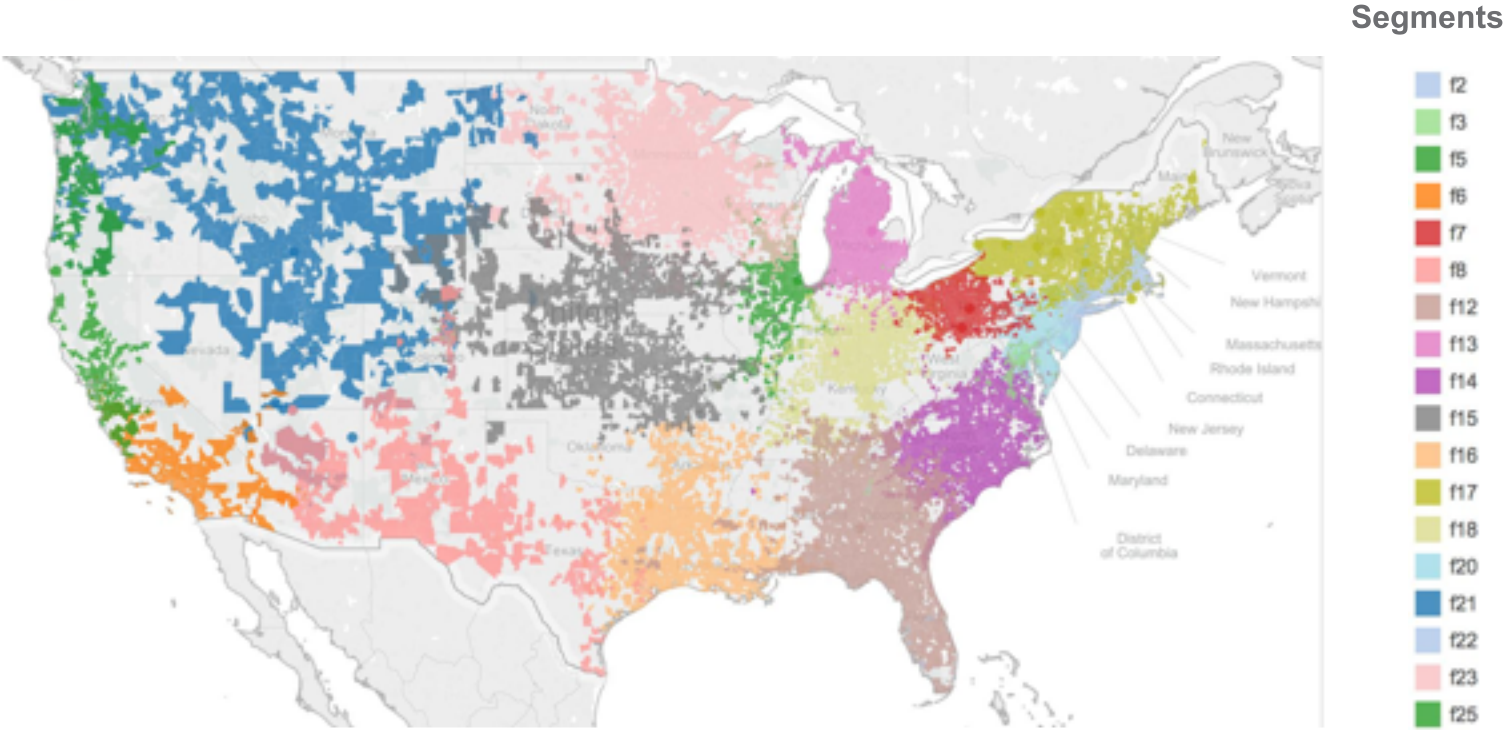
Segment Analysis I



Local websites

Global websites

Segment Analysis II



Most factors display geographic affinity.

Use ALSQp for Nonnegative Matrix Factorization

```
val als = new ALSQp()
  .setRank(params.rank)
  .setIterations(params. numIterations)
  .setUserConstraint(Constraints.POSITIVE)
  .setItemConstraint(Constraints.POSITIVE)
  .setImplicitPrefs(true)
  .setLambda(params.lambda)

val mfModel = als.run(training)
```

Other constraints:

`.setItemConstraint(Constraints.SIMPLEX) // 1ᵀw = s, w>=0 and s – constant`

https://github.com/apache/spark/pull/3221

Q and A