A Hierarchical Algorithm for Extreme Clustering

Ari Kobren*, Nicholas Monath*
Akshay Krishnamurthy, Andrew McCallum

College of Information and Computer Science
University of Massachusetts Amherst
Clustering
Clustering

Partition dataset $X$ into clusters $C_1 ... C_K$
Clustering

Partition dataset X into clusters $C_1 \ldots C_K$
Partition dataset $X$ into clusters $C_1 \ldots C_K$
Clustering

Partition dataset $X$ into clusters $C_1 \ldots C_K$

Analysis & Visualization

[Few, 2014]

[Heer et al, 2014]
Clustering

Partition dataset X into clusters $C_1 \ldots C_K$

Analysis & Visualization

[Few, 2014]

Feature Engineering

[Heer et al, 2014]

[Brown et al, 1993]
Clustering

Partition dataset X into clusters $C_1 \ldots C_K$

Analysis & Visualization

Feature Engineering

Deduplication

[Few, 2014]

[Heer et al, 2014]

[Brown et al, 1993]
Clustering

Partition dataset $X$ into clusters $C_1 \ldots C_K$

Analysis & Visualization

Feature Engineering

Deduplication

Image Segmentation

[Martin et al, 2001]

[Few, 2014]

[Heer et al, 2014]

[Brown et al, 1993]
Extreme Clustering
Extreme Clustering

Large Number of Clusters $K$
& Large Number of Points $N$
Extreme Clustering

Large Number of Clusters $K$
& Large Number of Points $N$

Speaker Recognition

NIST I-VECTOR Challenge

$N = 36,572$ Samples
$K = 4,958$ Speakers
Extreme Clustering

Large Number of Clusters $K$
& Large Number of Points $N$

Speaker Recognition

NIST I-VECTOR Challenge

$N = 36,572$ Samples
$K = 4,958$ Speakers

Image Clustering

$N = 14$ Million Images
$K = 21,000+$ Object Classes
Extreme Clustering

Large Number of Clusters $K$
& Large Number of Points $N$

Speaker Recognition
NIST I-VECTOR Challenge
$N = 36,572$ Samples
$K = 4,958$ Speakers

Image Clustering
$N = 14$ Million Images
$K = 21,000+$ Object Classes

Entity Resolution

Author Coreference. $N = 10$M Records, $K = 1$M Authors
Extreme Clustering

Large Number of Clusters $K$
& Large Number of Points $N$

Speaker Recognition
NIST I-VECTOR Challenge
$N = 36,572$ Samples
$K = 4,958$ Speakers

Image Clustering

Entity Resolution

Author Coreference. $N = 10M$ Records, $K = 1M$ Authors

$N = 14$ Million Images
$K = 21,000+$ Object Classes
Extreme Clustering

Large Number of Clusters $K$
& Large Number of Points $N$
def kmeans(x_1...x_N,K):
    until convergence
    for x in x_1...x_N:
        for c in clusters:
            if ||c - x|| < min_c:
                min_dist = ||c - x||
                min_c = x
            assign(x,min_c)
    update(clusters)
def kmeans(x_1, ..., x_N, K):
    until convergence
    for x in x_1, ..., x_N:
        for c in clusters:
            if ||c - x|| < min_c:
                min_dist = ||c - x||
                min_c = x
                assign(x, min_c)
    update(clusters)

Running time of k-means
def kmeans(x_1...x_N,K):
    until convergence
    for x in x_1...x_N:
        for c in clusters:
            if ||c - x|| < min_c:
                min_dist = ||c - x||
                min_c = x
            assign(x,min_c)
    update(clusters)
def kmeans(x_1...x_N,K):
    until convergence
    for x in x_1...x_N:
        for c in clusters:
            if ||c - x|| < min_c:
                min_dist = ||c - x||
                min_c = x
            assign(x,min_c)
    update(clusters)

Linear in K, O(NK)
Extreme Clustering

Large Number of Clusters $K$
& Large Number of Points $N$

```python
def kmeans(x_1...x_N, K):
    until convergence
        for x in x_1...x_N:
            for c in clusters:
                if ||c - x|| < min_c:
                    min_dist = ||c - x||
                    min_c = x
                assign(x, min_c)
            update(clusters)
```

For large $K$, we’d like to be sublinear.
Existing Approaches
## Existing Approaches

<table>
<thead>
<tr>
<th>Method</th>
<th>Scales in N</th>
<th>Scales in K</th>
<th>Non-Greedy</th>
<th>In Practice</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIRCH</td>
<td></td>
<td></td>
<td></td>
<td>Fast, but low accuracy</td>
</tr>
<tr>
<td>SGD/MiniBatch</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>StreamKM++</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BICO</td>
<td></td>
<td></td>
<td></td>
<td>Number of coresets does not scale with K</td>
</tr>
</tbody>
</table>

References:
- [Zhang et al, 1998](#)
- [Sculley, 2010](#)
- [Ackermann et al, 2012](#)
- [Fichtenberger, et al, 2013](#)
- [Bottou and Bengio, 1995](#)
### Existing Approaches

<table>
<thead>
<tr>
<th></th>
<th>Scales in N</th>
<th>Scales in K</th>
<th>Non-Greedy</th>
<th>In Practice</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BIRCH</strong></td>
<td>✔</td>
<td>✔</td>
<td></td>
<td>Fast, but low accuracy</td>
</tr>
<tr>
<td>[Zhang et al, 1998]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Existing Approaches

<table>
<thead>
<tr>
<th></th>
<th>Scales in N</th>
<th>Scales in K</th>
<th>Non-Greedy</th>
<th>In Practice</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BIRCH</strong></td>
<td>✔</td>
<td>✔</td>
<td>—</td>
<td>Fast, but low accuracy</td>
</tr>
<tr>
<td>[Zhang et al, 1998]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>SGD/MiniBatch</strong></td>
<td>✔</td>
<td>×</td>
<td>✔</td>
<td>Very effective, but does not scale with K</td>
</tr>
<tr>
<td><strong>K-Means</strong></td>
<td>✔</td>
<td>×</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>[Bottou and Bengio, 1995]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[Sculley, 2010]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Existing Approaches

<table>
<thead>
<tr>
<th>Method</th>
<th>Scales in N</th>
<th>Scales in K</th>
<th>Non-Greedy</th>
<th>In Practice</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIRCH</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>Fast, but low accuracy</td>
</tr>
<tr>
<td>[Zhang et al, 1998]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SGD/MiniBatch K-Means</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>Very effective, but does not scale with K</td>
</tr>
<tr>
<td>[Bottou and Bengio, 1995]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[Sculley, 2010]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>StreamKM++ BICO</td>
<td>✓</td>
<td>✗</td>
<td></td>
<td>Number of coresets does not scale with K</td>
</tr>
<tr>
<td>[Ackermann et al, 2012]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[Fichtenberger, et al, 2013]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Hierarchical Clustering

Advantages for Online Extreme Clustering

Efficiency
Hierarchical Clustering

Advantages for Online Extreme Clustering

Efficiency

Extreme Multiclass Classification:
[Choromanska et al, 2015]
[Daumé III et al, 2016]
Hierarchical Clustering

Advantages for Online Extreme Clustering

Efficiency

Extreme Multiclass Classification:
[Choromanska et al, 2015]
[Daumé III et al, 2016]

Top-Down Log-Time Search
Hierarchical Clustering

Advantages for Online Extreme Clustering

Non-greediness
Hierarchical Clustering

Advantages for Online Extreme Clustering

Non-greediness

Simultaneously Represent Multiple Alternative Clusterings
Hierarchical Clustering

Advantages for Online Extreme Clustering

Non-greediness

Simultaneously Represent Multiple Alternative Clusterings
Hierarchical Clustering

Advantages for Online Extreme Clustering

Non-greediness

Simultaneously Represent Multiple Alternative Clusterings
Hierarchical Clustering

Advantages for Online Extreme Clustering

Non-greediness

Simultaneously Represent Multiple Alternative Clusterings
PERCH
Purity Enhancing Rotations for Cluster Hierarchies
PERCH

Purity Enhancing Rotations for Cluster Hierarchies

Incrementally build hierarchical clustering
PERCH

Purity Enhancing Rotations for Cluster Hierarchies

Incrementally build hierarchical clustering

Route point to nearest neighbor
PERCH

Purity Enhancing Rotations for Cluster Hierarchies

Incrementally build hierarchical clustering

Route point to nearest neighbor

Tree maintenance using rotation operations
Dataset
True Clustering

(labels withheld from clustering algorithm)
def perch(x1...x_N,T):
    for x_i in x_1...x_N:
        n = nearestNeigh(x_i,T)
        v = split(n)
        recursiveRotate(v,T)
def perch(x1..x_N,T):
    for x_i in x1..x_N:
        n = nearestNeigh(x_i,T)
        v = split(n)
        recursiveRotate(v,T)
def perch(x1...x_N, T):
    for x_i in x1...x_N:
        n = nearestNeigh(x_i, T)
        v = split(n)
        recursiveRotate(v, T)
def perch(x_1…x_N,T):
    for x_i in x_1…x_N:
        n = nearestNeigh(x_i,T)
        v = split(n)
        recursiveRotate(v,T)
def perch(x_1, x_N, T):
    for x_i in x_1, x_N:
        n = nearestNeigh(x_i, T)
        v = split(n)
        recursiveRotate(v, T)
def perch(x₁…xₙ, T):
    for xᵢ in x₁…xₙ:
        n = nearestNeigh(xᵢ, T)
        v = split(n)
        recursiveRotate(v, T)
def perch(x_1..x_N,T):
    for x_i in x_1..x_N:
        n = nearestNeigh(x_i,T)
        v = split(n)
        recursiveRotate(v,T)
```python
def perch(x_{1\ldots N}, T):
    for x_{i} in x_{1\ldots N}:
        n = nearestNeigh(x_{i}, T)
        v = split(n)
        recursiveRotate(v, T)
```
```python
def perch(x_1..x_N, T):
    for x_i in x_1..x_N:
        n = nearestNeigh(x_i, T)
        v = split(n)
        recursiveRotate(v, T)
```
```python
def perch(x_1...x_N, T):
    for x_i in x_1...x_N:
        n = nearestNeigh(x_i, T)
        v = split(n)
        recursiveRotate(v, T)
```
def perch(x_1..x_N,T):
    for x_i in x_1..x_N:
        n = nearestNeigh(x_i,T)
        v = split(n)
        recursiveRotate(v,T)
def perch(x_1..x_N, T):
    for x_i in x_1..x_N:
        n = nearestNeigh(x_i, T)
        v = split(n)
        recursiveRotate(v, T)
```python
def perch(x_1..x_N, T):
    for x_i in x_1..x_N:
        n = nearestNeigh(x_i, T)
        v = split(n)
        recursiveRotate(v, T)
```
def perch(x_1..x_N, T):
    for x_i in x_1..x_N:
        n = nearestNeigh(x_i, T)
        v = split(n)
        recursiveRotate(v, T)
def perch(x_1..x_N, T):
    for x_i in x_1..x_N:
        n = nearestNeigh(x_i, T)
        v = split(n)
        recursiveRotate(v, T)
```python
def perch(x1..x_N,T):
    for x_i in x1..x_N:
        n = nearestNeigh(x_i,T)
        v = split(n)
        recursiveRotate(v,T)
```

*Data structures used for efficiency?*
def perch(x_1..x_N, T):
    for x_i in x_1..x_N:
        n = nearestNeigh(x_i, T)
        v = split(n)
        recursiveRotate(v, T)
def perch(x_1...x_N, T):
    for x_i in x_1...x_N:
        n = nearestNeigh(x_i, T)
        v = split(n)
        recursiveRotate(v, T)
def perch(x_1...x_N, T):
    for x_i in x_1...x_N:
        n = nearestNeigh(x_i, T)
        v = split(n)
        recursiveRotate(v, T)
def perch(x_1...x_N, T):
    for x_i in x_1...x_N:
        n = nearestNeigh(x_i, T)
        v = split(n)
        recursiveRotate(v, T)
```python
def perch(x_1..x_N,T):
    for x_i in x_1..x_N:
        n = nearestNeigh(x_i,T)
        v = split(n)
        recursiveRotate(v,T)
```
def perch(x_1,...,x_N,T):
    for x_i in x_1,...,x_N:
        n = nearestNeigh(x_i,T)
        v = split(n)
        recursiveRotate(v,T)
def perch(x_1…x_N, T):
    for x_i in x_1…x_N:
        n = nearestNeigh(x_i, T)
        v = split(n)
        recursiveRotate(v, T)
```python
def perch(x_1...x_N, T):
    for x_i in x_1...x_N:
        n = nearestNeigh(x_i, T)
        v = split(n)
        recursiveRotate(v, T)
```
```python
def perch(x_1..x_N, T):
    for x_i in x_1..x_N:
        n = nearestNeigh(x_i, T)
        v = split(n)
        recursiveRotate(v, T)
```

**Perch**

Fast Forward
```python
def perch(x_1...x_N, T):
    for x_i in x_1...x_N:
        n = nearestNeigh(x_i, T)
        v = split(n)
        recursiveRotate(v, T)
```

Perch
PERCH

Additional Efficiency Components:
Additional Efficiency Components:

Balance Rotations
- Improve tree balance without sacrificing purity
- Speed up nearest neighbor search
Additional Efficiency Components:

Balance Rotations
- Improve tree balance without sacrificing purity
- Speed up nearest neighbor search

Collapsed Mode
- Restrict the number of nodes in the tree
- Allows for clustering data that doesn’t fit in memory
PERCH

Theoretical Guarantees:

On separated data, PERCH constructs trees with dendrogram purity 1.0, even when using balance rotations and collapsing.
Results
Results

ILSVRC12 (50k): Running Time vs. Dendrogram Purity

- Perch
- Birch (online)
- Birch (rebuild)
- HKMeans

Graph showing running time vs. dendrogram purity for different algorithms.
## Results

<table>
<thead>
<tr>
<th>Method</th>
<th>CoverType</th>
<th>ILSVRC12 (50k)</th>
<th>ALOI</th>
<th>ILSVRC 12</th>
<th>Speaker</th>
<th>ImageNet (100k)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PERCH</td>
<td>0.45 ± 0.004</td>
<td>0.53 ± 0.003</td>
<td>0.44 ± 0.004</td>
<td>—</td>
<td>0.37 ± 0.002</td>
<td>0.07 ± 0.00</td>
</tr>
<tr>
<td>PERCH-BC</td>
<td>0.45 ± 0.004</td>
<td>0.36 ± 0.005</td>
<td>0.37 ± 0.008</td>
<td>0.21 ± 0.017</td>
<td>0.09 ± 0.001</td>
<td>0.03 ± 0.00</td>
</tr>
<tr>
<td>BIRCH (online)</td>
<td>0.44 ± 0.002</td>
<td>0.09 ± 0.006</td>
<td>0.21 ± 0.004</td>
<td>0.11 ± 0.006</td>
<td>0.02 ± 0.002</td>
<td>0.02 ± 0.00</td>
</tr>
<tr>
<td>MB-HAC-Com.</td>
<td>—</td>
<td>0.43 ± 0.005</td>
<td>0.15 ± 0.003</td>
<td>—</td>
<td>0.01 ± 0.002</td>
<td>—</td>
</tr>
<tr>
<td>MB-HAC-Cent.</td>
<td>0.44 ± 0.005</td>
<td>0.02 ± 0.000</td>
<td>0.30 ± 0.002</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>HKMmeans</td>
<td>0.44 ± 0.001</td>
<td>0.12 ± 0.002</td>
<td>0.44 ± 0.001</td>
<td>0.11 ± 0.003</td>
<td>0.12 ± 0.002</td>
<td>0.02 ± 0.00</td>
</tr>
<tr>
<td>BIRCH (rebuild)</td>
<td>0.44 ± 0.002</td>
<td>0.26 ± 0.003</td>
<td>0.32 ± 0.002</td>
<td>—</td>
<td>0.22 ± 0.006</td>
<td>0.03 ± 0.00</td>
</tr>
</tbody>
</table>

(a) Dendrogram Purity for Hierarchical Clustering.

<table>
<thead>
<tr>
<th>Method</th>
<th>CoverType</th>
<th>ILSVRC12 (50k)</th>
<th>ALOI</th>
<th>ILSVRC 12</th>
<th>Speaker</th>
<th>ImageNet (100K)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PERCH</td>
<td>22.96 ± 0.7</td>
<td>54.30 ± 0.3</td>
<td>44.21 ± 0.2</td>
<td>—</td>
<td>31.80 ± 0.1</td>
<td>6.178 ± 0.0</td>
</tr>
<tr>
<td>PERCH-BC</td>
<td>22.97 ± 0.8</td>
<td>37.98 ± 0.5</td>
<td>37.48 ± 0.7</td>
<td>25.75 ± 1.7</td>
<td>1.05 ± 0.1</td>
<td>4.144 ± 0.04</td>
</tr>
<tr>
<td>SKM++</td>
<td>23.80 ± 0.4</td>
<td>28.46 ± 2.2</td>
<td>37.53 ± 1.0</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>BICO</td>
<td>24.53 ± 0.4</td>
<td>45.18 ± 1.0</td>
<td>32.984 ± 3.4</td>
<td>—</td>
<td>1.73 ± 0.141</td>
<td>5.642 ± 0.00</td>
</tr>
<tr>
<td>MB-KM</td>
<td>24.27 ± 0.6</td>
<td>51.73 ± 1.8</td>
<td>40.84 ± 0.5</td>
<td>56.17 ± 0.4</td>
<td>22.63</td>
<td>—</td>
</tr>
<tr>
<td>DBSCAN</td>
<td>—</td>
<td>16.95</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>

(b) Pairwise F1 for Flat Clustering.
Thanks!

Questions?

https://arxiv.org/abs/1704.01858

https://github.com/iesl/xcluster