Machine Learning on Spark

Shivaram Venkataraman
UC Berkeley
Machine learning

Computer Science  Statistics
Machine learning

- Spam filters
- Click prediction
- Recommendations
- Search ranking
Machine learning techniques

- Classification
- Clustering
- Regression
- Active learning
- Collaborative filtering
Implementing Machine Learning

- Machine learning algorithms are
  - Complex, multi-stage
  - Iterative

- MapReduce/Hadoop unsuitable
- Need efficient primitives for data sharing
Machine Learning using Spark

- Spark RDDs → efficient data sharing

- In-memory caching accelerates performance
  - Up to 20x faster than Hadoop

- Easy to use high-level programming interface
  - Express complex algorithms ~100 lines.
Machine learning techniques

- Classification
- **Clustering**
- Regression
- Active learning
- Collaborative filtering
K-Means Clustering using Spark

Focus: Implementation and Performance
Clustering

Grouping data according to similarity

E.g. archaeological dig

Distance North

Distance East
Clustering

Grouping data according to similarity

Distance North

Distance East

E.g. archaeological dig
K-Means Algorithm

Benefits

• Popular
• Fast
• Conceptually straightforward

E.g. archaeological dig
**K-Means: preliminaries**

**Data:** Collection of values

```
data = lines.map(line=>
    parseVector(line))
```

The data is represented graphically with two features, Feature 1 and Feature 2. The point $x_3 = (1.5, 6.2)$ is highlighted on the graph.
K-Means: preliminaries

Dissimilarity:
Squared Euclidean distance

\[ \text{dist} = p.\text{squaredDist}(q) \]
K-Means: preliminaries

K = Number of clusters
\( \mu_1, \mu_2, \ldots, \mu_K \)

Data assignments to clusters
\( S_1, S_2, \ldots, S_K \)
K-Means: preliminaries

\[ \mathbf{K} = \text{Number of clusters} \]
\[ \mu_1, \mu_2, \ldots, \mu_K \]

Data assignments to clusters
\[ S_1, S_2, \ldots, S_K \]
K-Means Algorithm

• Initialize K cluster centers
• Repeat until convergence:
  Assign each data point to the cluster with the closest center.
  Assign each cluster center to be the mean of its cluster’s data points.
**K-Means Algorithm**

- Initialize $K$ cluster centers
- Repeat until convergence:
  Assign each data point to the cluster with the closest center.
  Assign each cluster center to be the mean of its cluster’s data points.
K-Means Algorithm

• Initialize K cluster centers
  
  centers = data.takeSample( 
      false, K, seed)

• Repeat until convergence:
  Assign each data point to the cluster with the closest center.
  Assign each cluster center to be the mean of its cluster’s data points.
K-Means Algorithm

- Initialize K cluster centers
  
  \[
  \text{centers} = \text{data}.\text{takeSample}(\text{false, K, seed})
  \]

- Repeat until convergence:
  
  Assign each data point to the cluster with the closest center.
  Assign each cluster center to be the mean of its cluster’s data points.
K-Means Algorithm

- Initialize K cluster centers
  
  \[
  \text{centers} = \text{data}.\text{takeSample}(\text{false, K, seed})
  \]

- Repeat until convergence:
  
  Assign each data point to the cluster with the closest center.
  
  Assign each cluster center to be the mean of its cluster’s data points.
K-Means Algorithm

- Initialize K cluster centers
  
  $$\text{centers} = \text{data}.\text{takeSample}(\text{false, K, seed})$$

- Repeat until convergence:
  
  $$\text{closest} = \text{data}.\text{map}(p \Rightarrow (\text{closestPoint}(p,\text{centers}), p))$$

Assign each cluster center to be the mean of its cluster’s data points.
K-Means Algorithm

- Initialize K cluster centers
  
  ```
  centers = data.takeSample(false, K, seed)
  ```

- Repeat until convergence:

  ```
  closest = data.map(p => (closestPoint(p, centers), p))
  ```

Assign each cluster center to be the mean of its cluster’s data points.
K-Means Algorithm

- Initialize K cluster centers
  
  ```
  centers = data.takeSample(false, K, seed)
  ```

- Repeat until convergence:
  
  ```
  closest = data.map(p =>
    (closestPoint(p, centers), p))
  ```

Assign each cluster center to be the mean of its cluster’s data points.
K-Means Algorithm

- Initialize K cluster centers
  
  \[
  \text{centers} = \text{data}\text{.takeSample}(\text{false, K, seed})
  \]

- Repeat until convergence:
  
  \[
  \text{closest} = \text{data}\text{.map}(p => \\
  (\text{closestPoint}(p,\text{centers}),p))
  \]

  \[
  \text{pointsGroup} = \text{closest}\text{.groupByKey()}
  \]
K-Means Algorithm

• Initialize K cluster centers
  centers = data.takeSample(false, K, seed)

• Repeat until convergence:
  closest = data.map(p =>
     (closestPoint(p, centers), p))
  pointsGroup = closest.groupByKey()
  newCenters = pointsGroup.mapValues(
     ps => average(ps))
K-Means Algorithm

- Initialize K cluster centers
  
  ```
  centers = data.takeSample(
    false, K, seed)
  ```

- Repeat until convergence:
  
  ```
  closest = data.map(p =>
    (closestPoint(p,centers),p))
  ```
  
  ```
  pointsGroup =
    closest.groupByKey()
  ```
  
  ```
  newCenters = pointsGroup.mapValues(
    ps => average(ps))
  ```
K-Means Algorithm

• Initialize K cluster centers
  
  \[ \text{centers} = \text{data}.\text{takeSample(} \text{false, K, seed)} \]

• Repeat until convergence:
  
  \[
  \text{closest} = \text{data}.\text{map}(p => \\
  \quad (\text{closestPoint}(p, \text{centers}), p))
  \]

  \[
  \text{pointsGroup} = \\
  \quad \text{closest}.\text{groupByKey()}
  \]

  \[
  \text{newCenters} = \text{pointsGroup}.\text{mapValues(} \\
  \quad \text{ps} => \text{average(ps)})
  \]
K-Means Algorithm

• Initialize K cluster centers
  
centers = data.takeSample(
    false, K, seed)

• Repeat until convergence:
  while (dist(centers, newCenters) > ε)
  
closest = data.map(p =>
    (closestPoint(p,centers),p))
  
pointsGroup =
    closest.groupByKey()
  
newCenters = pointsGroup.mapValues(
    ps => average(ps))
K-Means Algorithm

- Initialize $K$ cluster centers
  
  \[
  \text{centers} = \text{data}.\text{takeSample(}
  \text{false, K, seed)}
  \]

- Repeat until convergence:
  
  \[
  \text{while (dist(centers, newCenters) > } \epsilon) \]
  
  \[
  \text{closest} = \text{data}.\text{map(p =>}
  \text{ (closestPoint(p,centers),p))}
  \]
  
  \[
  \text{pointsGroup =}
  \text{ closest}.\text{groupByKey()}
  \]
  
  \[
  \text{newCenters} = \text{pointsGroup}.\text{mapValues(}
  \text{ps => average(ps))}
  \]
centers = data.takeSample(
    false, K, seed)
while (d > ε)
{
    closest = data.map(p =>
        (closestPoint(p,centers),p))
    pointsGroup =
        closest.groupByKey()
    newCenters = pointsGroup.mapValues(
        ps => average(ps))
    d = distance(centers, newCenters)
    centers = newCenters.map(_)
}
Ease of use

- Interactive shell:
  Useful for featurization, pre-processing data

- Lines of code for K-Means
  - Spark ~ 90 lines – (Part of hands-on tutorial !)
  - Hadoop/Mahout ~ 4 files, > 300 lines
Performance

K-Means

- Hadoop
- HadoopBinMem
- Spark

Logistic Regression

- Hadoop
- HadoopBinMem
- Spark

[Zaharia et al., NSDI'12]
Conclusion

- Spark: Framework for cluster computing
- **Fast** and **easy** machine learning programs

- K means clustering using Spark
- Hands-on exercise this afternoon!

Examples and more: [www.spark-project.org](http://www.spark-project.org)