Clustering

This page describes clustering algorithms in MLlib. The guide for clustering in the RDD-based API also has relevant information about these algorithms.

Table of Contents

K-means

k-means is one of the most commonly used clustering algorithms that clusters the data points into a predefined number of clusters. The MLlib implementation includes a parallelized variant of the k-means++ method called kmeans||.

`KMeans` is implemented as an `Estimator` and generates a `KMeansModel` as the base model.

Input Columns

Param name	Type(s)	Default	Description
featuresCol	Vector	"features"	Feature vector

Output Columns

Param name	Type(s)	Default	Description
predictionCol	Int	"prediction"	Predicted cluster center

Examples

Scala Java Python R

Refer to the Python API docs for more details.

```
from pyspark.ml.clustering import KMeans

# Loads data.
dataset = spark.read.format("libsvm").load("data/mllib/sample_kmeans_data.txt")
```

```
# Trains a k-means model.
kmeans = KMeans().setK(2).setSeed(1)
model = kmeans.fit(dataset)

# Evaluate clustering by computing Within Set Sum of Squared Errors.
wssse = model.computeCost(dataset)
print("Within Set Sum of Squared Errors = " + str(wssse))

# Shows the result.
centers = model.clusterCenters()
print("Cluster Centers: ")
for center in centers:
    print(center)
```

Find full example code at "examples/src/main/python/ml/kmeans_example.py" in the Spark repo.

Latent Dirichlet allocation (LDA)

`LDA` is implemented as an `Estimator` that supports both `EMLDAOptimizer` and `OnlineLDAOptimizer`, and generates a `LDAModel` as the base model. Expert users may cast a `LDAModel` generated by `EMLDAOptimizer` to a `DistributedLDAModel` if needed.

Examples

Scala | Java | Python | R

Refer to the Python API docs for more details.

```
from pyspark.ml.clustering import LDA

# Loads data.
dataset = spark.read.format("libsvm").load("data/mllib/sample_lda_libsvm_data.txt")

# Trains a LDA model.
lda = LDA(k=10, maxIter=10)
model = lda.fit(dataset)

ll = model.logLikelihood(dataset)
lp = model.logPerplexity(dataset)
print("The lower bound on the log likelihood of the entire corpus: " + str(ll))
print("The upper bound on perplexity: " + str(lp))

# Describe topics.
topics = model.describeTopics(3)
print("The topics described by their top-weighted terms:")
topics.show(truncate=False)

# Shows the result
transformed = model.transform(dataset)
transformed.show(truncate=False)
```

Find full example code at "examples/src/main/python/ml/lda_example.py" in the Spark repo.

Bisecting k-means

Bisecting k-means is a kind of hierarchical clustering using a divisive (or “top-down”) approach: all observations start in one cluster, and splits are performed recursively as one moves down the hierarchy.

Bisecting K-means can often be much faster than regular K-means, but it will generally produce a different clustering.

`BisectingKMeans` is implemented as an `Estimator` and generates a `BisectingKMeansModel` as the base model.

Examples

Scala | Java | Python | R

Refer to the Python API docs for more details.

```python
from pyspark.ml.clustering import BisectingKMeans

# Loads data.
dataset = spark.read.format("libsvm").load("data/mllib/sample_kmeans_data.txt")

# Trains a bisecting k-means model.
bkm = BisectingKMeans().setK(2).setSeed(1)
model = bkm.fit(dataset)

# Evaluate clustering.
cost = model.computeCost(dataset)
print("Within Set Sum of Squared Errors = " + str(cost))

# Shows the result.
print("Cluster Centers: ")
centers = model.clusterCenters()
for center in centers:
    print(center)
```

Find full example code at "examples/src/main/python/ml/bisecting_k_means_example.py" in the Spark repo.

Gaussian Mixture Model (GMM)

A Gaussian Mixture Model represents a composite distribution whereby points are drawn from one of *k* Gaussian sub-distributions, each with its own probability. The `spark.ml` implementation uses the expectation-maximization algorithm to induce the maximum-likelihood model given a set of samples.

`GaussianMixture` is implemented as an `Estimator` and generates a `GaussianMixtureModel` as the base model.

Input Columns

Param name	Type(s)	Default	Description
featuresCol	Vector	"features"	Feature vector

Output Columns

Param name	Type(s)	Default	Description
predictionCol	Int	"prediction"	Predicted cluster center
probabilityCol	Vector	"probability"	Probability of each cluster

Examples

Scala | Java | Python | R

Refer to the Python API docs for more details.

```
from pyspark.ml.clustering import GaussianMixture

# loads data
dataset = spark.read.format("libsvm").load("data/mllib/sample_kmeans_data.txt")

gmm = GaussianMixture().setK(2).setSeed(538009335)
model = gmm.fit(dataset)

print("Gaussians shown as a DataFrame: ")
model.gaussiansDF.show(truncate=False)
```

Find full example code at "examples/src/main/python/ml/gaussian_mixture_example.py" in the Spark repo.